1985

COMPLEX DIGITAL SIGNAL PROCESSING USING QUADRATIC RESIDUE NUMBER SYSTEMS.

RAMASAMY. KRISHNAN
University of Windsor

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

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

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

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

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

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

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

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

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

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

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

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

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

NL-339 (r. 82/08)

Canada

COMPLEX DIGITAL SIGNAL PROCESSING USING QUADRATIC RESIDUE NUMBER SYSTEMS

by

Ramasamy Krishnan

A Dissertation

Submitted to the Faculty of Graduate Studies
through the Department of Electrical Engineering
in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada

1985

ABSTRACT

This work presents the development of complex digital signal processing algorithms using number theoretic techniques.

Residue number principles and techniques are applied to process complex signal information in Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filters. Residue coding of complex samples and arithmetic for processing complex data have been presented using principles of quadratic residues in the Residue Number System (RNS). In this work, we have presented modifications to the Quadratic Residue Number System (QRNS), which we have termed the Modified Quadratic Residue Number System (MQRNS), to process complex integers. New results and theorems have been obtained for the selection of operators to code complex integers into the new MQRNS representation. A novel scheme for residue to binary conversion has been presented for implementation using both the QRNS and MQRNS.

Hardware implementations of multiplication intensive complex non-recursive and recursive digital filters have been presented where the QRNS and MQRNS structures are realized using a bit-slice architectural approach. The computation of Complex Number Theoretic Transforms (CNTTs) and the hardware implementation of a radix-2 NTT butterfly structure, using high density ROM arrays, are presented in both the QRNS and MQRNS systems. As an illustration, the computation of the CNTT developed in this work, is used to compute Cyclic Convolution for complex sequences. These results are verified by computer programs.

The recursive FIR filter structure for uniformly spaced frequency samples on the unit circle developed by adapting the Complex Number Theoretic z-transform, has been implemented using the QRNS and MQRNS. In this work, the filter structure is extended for non-uniformly spaced frequency samples and has been termed the generalized number theoretic filter structure. It is shown that for the implementation of this generalized structure, the MQRNS is more efficient than the conventional RNS; the QRNS does not support appropriate fields for the generalized structure.

ACKNOWLEDGMENT

I would like to express my sincere thanks and appreciation to my supervisor, Dr. G.A. Jullien, for his valuable advice, guidance and constant support and encouragement throughout the progress of this research. The valuable advice of Dr. W.C. Miller, Dr. M. Shridhar, Dr. J.J. Soltis and other faculty members is also gratefully acknowledged.

I extend my sincere thanks and gratitude to my wife, Rajarajeswari, and to my mother and brothers for their help, moral support and constant encouragement at all times.

Thanks are also due to the Inter-Loan Library Department and the Photocopy Centre of the Leddy Library for their valuable and timely help in getting the research materials.

Thanks are due as well to Mrs. Linda Kennedy for her diligence in typing this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\{A, A^*\}$	element pairs of the extension ring
$a \mid b$	a divides b
gcd(a,b)	Greatest common divisor of elements a and b.
$a \equiv b \pmod{m}$	Integer a is congruent to integer b modulo m
a^{-1}	Multiplicative inverse of a multiplicative inverse of a modulo m
$\{a_i\}$	Set consisting of the elements a
CNT z-transform	Complex Number Theoretic z-transform
$C(m) = \{a + \sqrt{-1}\ b \mid a, b \in GF(m)\}$	Finite field of Gaussian integers
F[x]	Set of polynomials in the element x
$F[\beta]$	The extension field obtained by adjoining β to field F
$GF(m^n)$	Galois Field with m^n elements
$GF(m) - \{0\}$	Multiplicative group of $GF(m^n)$
$GF(m^2)$	$\{a + \hat{i}b \mid a, b \in GF(m)\}$ Finite field of quadratic integers. $\hat{i}$ is the solution to quadratic monic polynomial.
H(z)	Transfer function (z-transform of the impulse response.
$h(s) * x(x)$	Convolution
$h(s) \circledast x(s)$	Cyclic Convolution
$\text{ind}_\alpha a$	Integer of a, relative to α, α a generator of the multiplicative group
j	root of a quadratic monic modulo 4K + 1
$\hat{j}$	root of a quadratic monic polynomial $x^2 - n = 0$.

L	Number of moduli
m_i	ith moduli
M	product of the moduli m_i
MOPS	Million Operations Per Second
MQR(m)	Modified Quadratic Residue Ring
QR(M)	Quadratic Residue Ring
r_i	ith residue
R(m) or Z(m)	is the ring of integers modulo m
$\|x\|_m$	Residue of x modulo m
$\phi(m)$	Euler's phi function
α	Generator of the multiplicative group
β	root of irreducible polynomial
$\sum \oplus$	direct sum of the rings
$\odot$	Operations of addition, subtration or multiplication
$\lceil \rceil$	Ceiling function
$\left\|\frac{1}{a}\right\|_m$	multiplicative inverse of a modulo m

LIST OF APPENDICES

Chapter I

INTRODUCTION

During the recent years researchers have devoted considerable effort in the practical implementation of the number theoretic techniques including the theory of the Residue Number System (RNS), for the implementation of digital signal processing algorithms.

The earliest known work in the Residue Number System was by a Chinese mathematician, Sun-Tsu, of the first century AD, who stated the Chinese Remainder Theorem in its original form. Although the theoretical foundations of the residue arithmetic were developed by mathematicians, such as Euler, Fermat and Gauss, in the eighteenth and nineteenth centuries, the application of number theory has become prominent in the last two decades.

In the RNS, the binary operations of addition, subtraction and multiplication can be implemented with the absence of carry propagation which provides RNS an inherent speed advantage compared with the conventional weighted number system. The binary operations are performed in L independent parallel paths; the required dynamic range can be obtained by varying the number of parallel paths in the RNS. These characteristics lend themselves to fast arithmetic operations in digital computer architectures.

In the early 1960's researchers explored the basic properties of the Residue Number System (RNS) and demonstrated the advantages of the RNS for general purpose, high speed computer applications. The early investigators such as Savaboda and Valach [6] were interested in using

the RNS for the design and construction of a general purpose computer which resulted in the EPOS general purpose computer. In this respect the work of Szabo and Tanaka [79] is a very important reference. In [7], Garner studied the arithmetic properties of residue numbers with the objective of deriving error correcting codes for computers. Since the RNS is an integer number system, division is, unfortunately, not, in general, a closed operation. The RNS has difficulties in implementing operations of division, scaling, sign detection and magnitude comparison. For example, magnitude comparison in the RNS is known to be difficult to achieve. All the reported overflow detection algorithms make use of the Chinese Remainder Theorem (CRT) or the mixed-radix conversion algorithms. Since the above said operations are awkward in the RNS, it was abandoned for general purpose computers.

Although many shortcomings still exist in applying the RNS to general purpose computer design, the techniques are well suited to the implementation of digital signal processing algorithms [46,84]. The operations required are only addition, subtraction, multiplication and scaling (division) operations. A restricted form of division in which the divisor is a fixed constant is usually required in digital filters to prevent overflow and to optimize the signal to noise ratios in various parts of the filter. An efficient implementation of the scaling operation was reported by Jullien [1] in 1978.

The RNS architectures for Finite Impulse Response (FIR) digital filters were proposed by Jenkins and Leon [2] in 1977. The application

to FIR filtering requires only addition and multiplication, the fast RNS operations. Jenkins and Leon showed how good cost/performance figure could be obtained using the RNS filters based upon the read only memory (ROM) look-up table approach.

In the case of Finite Impulse Response filters (FIR) algorithm which employ indirect filtering via transforms, have provided efficient implementation, using the residue number system techniques. These specific classes of algorithms have been referred to in the literature, as Number Theoretic Transform (NTT) for which the transform is defined over a finite field (or ring) rather than the field of complex numbers over which the commonly used transforms are defined. Digital filtering by means of finite field transforms requires no scaling, and so it can take advantage of the strong features of the RNS. NTTs computed over residue class rings or fields have been extended to the more general setting that includes the so called Complex Number Theoretic Transforms (CNTTs). Unlike DFTs, the NTT based filters are error free and fast. Recently, various authors have proposed Number Theoretic Transforms over different finite fields and rings in [23,24,26,27,71].

Infinite Impulse Response (IIR) filters are implemented recursively and enjoy many cost and speed advantages over FIR filters. The use of feedback allows an IIR realization that is of a lower order than a FIR executing a comparable function. In this sense an IIR is more economical in terms of speed and required hardware. However, these benefits are acquired at a cost of sensitivity to roundoff error accumulation because of the feedback in the IIR realization. Many stable, recursive filters have fractional fiter coefficients. Since fractions are

not allowed in the RNS, the fractional coefficients must be scaled up for the computation, and then the output must be scaled down before being fed back into the recursion [1]. This operation is termed scaling. Hence the implementation of RNS recursive digital filters have centered upon efficient scaling algorithms. Scaling is not an easy operation in the RNS and cannot be implemented exactly; hence the scaling operation introduces quantization errors in the recursive filters. The implementation of IIR filters using RNS techniques has been studied using efficient scaling algorithms for low quantization noise by Jullien [1] and Jenkins [13]. Soderstrand [33] has applied the RNS techniques to the implementation of second order recursive digital filters based on lossless discrete integrators. Here the scaling is avoided by storing the complete multiplication by fractions in look-up tables but adaptive filtering is not possible using this approach.

In addition to the implementation of digital filters, the RNS finds some applications in error detection and correction, where both transmission and computational errors are to be controlled. The modularity and parallelism of the RNS allows the isolation of hardware faults, giving the system unique error-correcting properties. In the RNS, the error detecting and correcting capability is usually achieved by adding one or more redundant residue digits. The structure is well suited to current circuit layout techniques that benefit from regularity and modularity. The possibility of the RNS error detection and correction have been mentioned early by Szabo and Tanaka [79], but Mandelbaum's [37] results contain the earliest proofs of the RNS error properties. In 1973, Barsi and Maestrini published a significant paper on this

subject [41]. It contains proofs and procedures for RNS error correction and serves as a foundation for the subsequent work in the field. These results were applied by Jenkins and Etzel [38] to fault tolerant digital filtering. Their results included efficient error correction algorithms and simulation of a gradually degrading RNS recursive filter. In [21], the authors have extended this work to the redundant Quadratic Residue Number System (QRNS).

Using the RNS based filter structure, requires a binary to residue interface at the input and residue to binary conversion at the output. For instance, during output conversion of signed numbers, the operation of sign detection and residue to binary conversion must be performed simultaneously. Residue to binary conversion uses basically two methods: Chinese Remainder Theorem (CRT) and the mixed radix conversion. Very recently, an efficient residue to binary conversion scheme has been proposed by Soderstrand in [4] using the Chinese Remainder Theorem, and by Baraniecka and Jullien in [5] using mixed radix conversion techniques. Numerous techniques on this subject have been discussed [3,11].

From the above mentioned works and the works of various other authors, the conclusion that emerges is that the RNS is becoming an attractive and useful tool for the implementation of the digital signal processing algorithms. The recent breakthroughs in high density memory technology and microprocessor hardware have made the RNS become increasingly important in the application of digital signal processing.

1.1 The Objective and Review of the Research Work

The principal objective of this research is to explore the application of residue number techniques in the construction of digital signal processing hardware to process complex data sequences. As a starting point, we explore the basic principles involved in performing complex arithmetic operations using the Residue Number System. We have also explored the basic hardware principles involved in performing arithmetic operations in the RNS.

The motivation for this work is based upon the recently published results showing the attractiveness of using the Quadratic Residue Number System (QRNS) to obtain very high speed complex arithmetic processing hardware. In the QRNS the complex number will be decomposed into element pairs (element and conjugate). Using these element pairs, the complex multiplication can be performed with two real multiplications with restrictions of the form of the prime moduli for the RNS processing. The motivation for this research stemmed from the fact that the QRNS has been defined for a particular set of moduli with prime factors of the form $4K + 1$.

The primary aim of this research concentrates on the modification of the QRNS, so that moduli of any form can be used for the RNS processing. The resulting number system has been termed the Modified Quadratic Residue Number System (MQRNS). In the MQRNS, moduli of any form can be used with an increase in multiplications from 2 to 3. New theorems and results based on concepts from abstract algebra and number theory are presented, that allow selection of the operators for the MQRNS. Using the

QRNS and MQRNS we have developed algorithms for the implementation of Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filters. The conventional complex digital filters and the QRNS and MQRNS digital filters have been investigated by comparing the hardware requirements and the computational speed. For this work we have considered several filter architectures. From the above investigation we have discovered that the QRNS and MQRNS based filter architectures offer the most efficient results compared with architecture based on the conventional RNS arithmetic.

In the case of the residue to binary conversion, we have developed a method for the direct decoding of the element pairs of the QRNS and MQRNS. An in-depth study of the computational aspects of Complex Number Theoretic Transform and the hardware implementation of the butterfly structures in the QRNS and MQRNS have been investigated. A generalized frequency sampling filter structure (Legrange structures) has been developed using the concept of the Complex Number Theoretic z-transform which utilizes the MQRNS arithmetic for the implementation.

1.2 Organization of Thesis

Chapter II covers the pre-requisite material for the research. In particular a concise review of some of the fundamentals on finite ring and field structures is presented and the mathematic concepts of Residue Number System (RNS) are introduced. This chapter provides a background reference for definitions and notations used throughout the thesis.

Chapter III describes the modification of the Quadratic Residue Number System, which we have termed the Modified Quadratic Residue Number System.

Chapter IV covers the implementation concepts of the residue arithmetic in the QRNS and MQRNS including techniques for performing the arithmetic along with methods for interfacing QRNS and MQRNS based systems with binary based systems. It also deals with the implementation of the QRNS and MQRNS using VLSI circuit fabrication.

Chapter V deals with the implementation of recursive and non-recursive filters using the QRNS and MQRNS. The implementation of direct FIR filter architecture and bit-slice architectures are discussed.

Chapter VI presents the computation of the Complex Number Theoretic Transform along with the hardware implementation of the butterfly structures. It also presents the development of the generalized FIR filter structure (Lagrange structure) using the Complex Number Theoretic z-transform.

Chapter VII summarizes the results of this research.

Chapter II

CONCEPTS OF NUMBER THEORY AND ABSTRACT ALGEBRA APPLICABLE TO DIGITAL SIGNAL PROCESSING

2.1 Introduction

In this research, an extension of the Quadratic Residue Number System (QRNS), the Modified Quadratic Residue Number System has been developed by using the concepts of quadratic residue rings for complex digital filtering. As a pre-requisite, this section introduces the concepts of residue algebra and presents a brief review of background material on the ring and field structure as a foundation for the works in the succeeding chapters.

2.2 Finite Rings and Fields

Since we are going to define the Quadratic Residue Number Systems (QRNS) over quadratic residue rings, we will briefly present the structure of rings and fields, a branch of abstract algebra. As a starting point, it would seem appropriate to formally define the notation of a ring and field.

Definition - 1: A non-empty set R is said to be a ring if the operations of addition and multiplication satisfy the following postulates.

1. $a + b$ is in ring R
2. $a + b = b + a$ (Commutative law of addition)
3. $(a + b) + c = a + (b + c)$ (Associative law of addition)
4. There is an element 0 in R such that $a + 0 = a$ for every a in R (existence of zero).
5. There exists an element $-a$ in R such that $a + (-a) = 0$ (Existence of additive inverse)

6. $a \cdot b$ is in ring R

7. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ (Associative law of multiplication)

8. $a \cdot (b + c) = a \cdot b + a \cdot c$ and

$(b + c) \cdot a = b \cdot a + c \cdot a$ (Distributive laws)

where a, b and c are arbitrary elements in R.

If, further, the following law holds $a \cdot b = b \cdot a$ for every $a, b \in R$, then R is called a commutative ring. A ring with identity is a ring in which there exists an identity element for the operation of multiplication $a \cdot 1 = 1 \cdot a = a$ for all $a \in R$. 1 is the identity.

The ring of integers is an example for a commutative ring with identity. In abstract algebra, elements of a ring are not necessarily the integers, even not necessarily numbers, it can be polynomials, for example.

If a is in R and $a \neq 0$, then there exists an element in R called the reciprocal of a and denoted by a^{-1} (multiplicative inverse), such that $a \cdot a^{-1} = 1$. The set of all invertible elements of a ring is a group with respect to the operation of multiplication and is called a "multiplicative group."

Definition - 2: An element a of a ring R, $a \neq 0$, is said to be a divisor of zero in R if there exists a non-zero element b of R, $b \neq 0$ such that $a \cdot b = 0$.

Definition - 3: A commutative ring with identity is an integral domain, if it has no zero divisors.

Definition - 4: A commutative ring $F - \{0\}$ with more than one element and having a multiplicative identity is said to be a field if every non-

zero element of F has a multiplicative inverse in F.

It follows [77,76] that every field is an integral domain. The rings (fields) with a finite number of elements are called finite rings (fields).

A ring of integers with addition and multiplication computed modulo m, denoted here as Z(m) is an example of a finite ring.

Definition 5: For any prime m and any positive integer n, there exists a finite field with m^n elements. This field is commonly denoted by the symbol $GF(m^n)$ and is called a Galois field. Since any finite field with m^n elements is a simple algebraic extension of field, a brief review of the basic concepts about the extensions of a given field will be presented.

Let F be a field. Then any field K containing F is an extension of F. If β is the algebraic over F, i.e, if β is a root of some irreducible polynomial

$$f(x) = x^2 + 1. \in F[x] \text{ such that } f(\beta) = 0,$$

then the extension field arising from a field F by the adjunction of a root β is called a simple algebraic extension, denoted by $F(\beta)$. Each element of $F(\beta)$ can be uniquely represented as a polynomial

$a_0 + a_1 \beta + \ldots\ldots + a_{n-1} \beta^{n-1}$, $a \in F$.

This unique representation closely resembles the representation of a vector in terms of the vectors of the basis "1, β ---- β^{n-1}" . The vector space concepts are sometimes applied to the extension fields, and $F(\beta)$ is considered as a vector space dimension n over F.

The field of complex numbers is an example of an extension of real numbers; it is generated by adjoining a root $\hat{\imath} = \sqrt{-1}$ of a irreducible polynomial $x^2 + 1 = 0$.

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 usually defined [76,90] as a quotient field Z(m) [x]/f(x), i.e., the field of residue classes of polynomials of Z(m) [x] reduced modulo (f(x)). All fields containing m^n elements are isomorphic to each other. In particular, Z(m)[x]/(f(x)) is isomorphic to the simple algebraic extension Z(m) (β) where β is a root of f(x) = 0.

2.3 Cyclic Sub-Groups

Consider a finite field with m-elements. In this finite field, the non-zero elements form a multiplicative group. This multiplicative group of order m-1 is cyclic. That is it contains an element whose powers generate all elements of the group. This element is called a generator α, and the order of α is m-1. The order of any element ω in the multiplicative group is the least positive integer 'k' such that $\omega^k = 1$, $\omega^s \neq 1$, $1 < s < k$, the order, k, is a divisor of m-1 and ω is called a primitive k-th root of unity.

The multiplicative group $\{\omega \in F - \{0\}, |\cdot|_m\}$ is isomorphic to an additive group $\{T \in F-\{m-1\}, |+|_{m-1}\}$ where $|\ |$ implies that the operation computed modulo m. The isomorphic mapping between the group elements is given

$$\omega = \alpha^T, \quad T \in \{0, 1, \text{——} m-2\} \tag{2.1}$$

The integer T is called the index of ω relative to the base, denoted $\text{ind}_\alpha\ \omega$.

2.4 The Ring of Residue Classes

For given integers a and b, $b \neq 0$, there exists two unique integers, q and r such that

$$a = bq + r \qquad 0 \leq r < b \tag{2.2}$$

It is clear that q is the integer value of quotient a/b. The quantity r is the least positive (integer) remainder of the division of a by b and is designated as the residue of a modulo b, or $|a|_b$. We will say that b divides a (written $b|a$) if there exist an integer k such that $a = b \cdot k$.

Definition - 6: If a, b and m are integers $m > 0$, then we say that a is congruent to b, modulo m written as

$$a \equiv b \mod m \text{ if and only if } m \text{ divides } a - b.$$

Since $m|0$, $c \equiv c \mod m$ by definition. We can define the congruence in another form as follows.

Two integers a and b are congruent modulo m if and only if they leave the same remainder when divided by m.

(i.e.) $$|c|_m = |d|_m$$

Definition - 7: A set of integers containing exactly those integers which are congruent modulo m, to a fixed integer is called a residue class modulo m.

The residue class (mod m) forms a commutative ring with identity with respect to modulo m addition and multiplication traditionally known as the ring of integers modulo m or the residue ring and denoted by $Z(m)$. This ring of residue classes (mod m) contains exactly m distinct

elements. The ring of residue classes (mod m) is a field if and only if m is a prime number, and there exists multiplicative inverse, denoted by a^{-1} (mod m) for each non-zero $a \in Z(m)$. Thus the non-zero classes of Z(m) form a cyclic multiplicative of order m-1, {1,....m-1} with multiplication modulo m is isomorphic to the additive group {0,1,...m-2} with addition modulo m-1.

If m is a composite, Z(m) is not a field. Multiplicative inverses do not exist for non-zero elements $a \in Z(m)$, for which $\gcd(a,m) \neq 1$.

At this stage let us introduce Euler's totient function, denoted by $\phi(m)$. It is defined as the number of positive integers less than m and relatively prime to m. It follows that the number of invertible elements of Z(m) is equal to $\phi(m)$[80].

$$\phi(m) = m[1-1/m_1][1-1/m_2]....[1-1/m_L] \qquad (2.3)$$

where m has prime factorization

$$m = m_1 \cdot m_2 \cdot \cdots \cdot m_L$$

The Euler-Fermat theorem states that if a is an integer, m is a positive integer and gcd(a,m) = 1,

then

$$a^{\phi(m)} \equiv 1 \bmod m \qquad (2.4)$$

The useful property of this theorem is that there is an upper limit on the order of any element $a \in Z(m)$. Specifically, the order t is a divisor of $\phi(m)$.

2.5 Quadratic Residues

The concept of quadratic residues is important to the theory of Quadratic Residue Number Systems (QRNS) which form the basis for this work.

Given two integers, r and m, such that (r,m) = 1, r is called a quadratic residue, modulo m, if the congruence $x^2 \equiv r \bmod m$ has a solution. If no solution exists r is called a quadratic non-residue mod m.

If m is a prime, m > 2, there are (m-1)/2 quadratic residues and (m-1)/2 quadratic non-residues modulo m [75].

Example:

For m = 11 the quadratic residues are obtained as shown below

$$1^2 = 10^2 = 1 \bmod 11$$
$$2^2 = 9^2 = 4 \bmod 11$$
$$3^2 = 8^2 = 9 \bmod 11$$
$$4^2 = 7^2 = 5 \bmod 11$$
$$5^2 = 6^2 = 3 \bmod 11$$

Thus the quadratic residues modulo 11 are 1, 3, 4, 5 and 9 and the quadratic non-residues are 2, 6, 7, 8 and 10.

If a is a quadratic residue modulo m, then

$$a^{\frac{m-1}{2}} \equiv 1 \bmod m \tag{2.5}$$

If a is a quadratic non-residue modulo m, then

$$a^{\frac{m-1}{2}} \equiv -1 \bmod m \tag{2.6}$$

If we use Legendre symbol [80] in equation 2.6, it is evident that,

$$\left(\frac{a}{m} \right) \equiv a^{\frac{m-1}{2}} \mod m \tag{2.7}$$

or we can say that

$$\left(\frac{-1}{m} \right) \equiv (-1)^{\frac{m-1}{2}} \tag{2.8}$$

As $(m-1)/2$ is even for m of the form $4K + 1$ and odd for m of the form $4K + 3$, it follows that

Theorem - 1: [81] The number -1 is a quadratic residue of all primes of the form $4K + 1$ and quadratic non-residue of all the primes of the form $4K + 3$.

2.6 Residue Number Systems

In the Residue Number System (RNS), each integer can be represented by an L-tuple of residue digits:

$$X = (X_0, X_1, \ldots\ldots\ldots X_L\text{-}1) \tag{2.9}$$

where $x_i = |X|_{m_i}$ is the ith residue and m_i is the ith modulus, $|\cdot|_{m_i}$ represents modulo m_i operation. The binary operations of two residue sequences $(x_0, x_1, \ldots x_{L-1})$ and $y_0, y_1, \ldots y_{L-1})$ is given by $(r_0, r_1, \ldots r_{L-1})$ with $r_i = |x_i \odot y_i|_{m_i}$ where $\odot$ represents the binary operation of addition, subtraction or multiplication within $R(m_i)$ where $R(m_i)$ is the ring of integers modulo m_i.

If all the m_i are relatively prime, it can be shown that [79] there is a unique representation for each number in the range

$$0 \leq X < \prod_{i=0}^{L-1} m_i = M \tag{2.10}$$

where L is the number of moduli.

For applications in signal processing, it is helpful to define a dynamic range for the RNS with positive and negative integers. Hence, the dynamic range is defined as $[-(M-1)/2, (M-1)/2]$ for M odd, and as $[-M/2, M/2-1]$ for M even. Any integer X within the dynamic range can be represented by L-distinct residue digits as in (2.9). It provides the relationship between the ring of integers modulo M and the direct product of the L subrings $R(m_i)$, $i \in \{1,2....L\}$. This fact can be established using the Chinese Remainder Theorem (CRT) [79].

$$X = \left| \sum_{i=0}^{L-1} \hat{m}_i \left| x_i \right| \tilde{m}_i^{-1} \right|_{m_i} \Bigg|_M \tag{2.11}$$

where

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

$\hat{m}_i = M/m_i$, and $(m_i, m_j) = 1$ for all $i \neq j$.

The requirement that the moduli be relatively prime ensures the existence of $\hat{m}_i^{-1}$ mod m_i.

In algebraic terms the RNS is a on-to-one correspondence (an isomorphism) between the ring of integers modulo M and the direct product of L smaller rings modulo m_i.

The additive inverse in modulo complement form is given as $\bar{X} = M-X$ and for each residue $\bar{x}_i = m_i - x_i$ so that $|X + \bar{X}|_M = 0$.

The interesting feature of the RNS is that the intermediate overflows of an arithmetic computation can be ignored and we obtain the correct answer if the final result is within the range of the number system.

2.7 Summary

This chapter has presented a concise review of the fundamental number theoretic and abstract algebra concepts related to the digital signal processing algorithms developed in this thesis. The chapter can be used as a background reference for definitions, nomenclature and notations used throughout this thesis.

The notations used in finite ring and field algebra have been introduced. The concept of quadratic residues which has been presented is the pre-requisite knowledge for the development of Quadratic Residue Number Systems (QRNS) over quadratic residue rings and Modified Quadratic Residue Number System (MQRNS) over modified quadratic residue rings. The concept of the primitive root, which is introduced in this chapter, will provide the knowledge required for the development of algorithms for the computation of Complex Number Theoretic Transform (CNTT) in the QRNS and MQRNS.

The notion of a multiplicative inverse and conditions for its existence has been discussed. Euler's totient function has been presented as a function of the mathematical concept of the residue number system. The basic properties of residue number systems have been described, and the independence of binary operations between the residue digits has been discussed.

Chapter III

THE QUADRATIC RESIDUE NUMBER SYSTEM (QRNS) AND MODIFIED QUADRATIC RESIDUE NUMBER SYSTEM (MQRNS)

3.1 Introduction

Many signal processing applications require the representation of complex data and filter coefficients. A familiar example is the quadrature modulated communication system. In fact, complex signal processing has many applications in base band processing for narrow band r.f signals; homomorphic speech processing; spectral analysis; matched filters for coherent Radars [87]; a range gated pulse dopler Radar System[61]. Processing complex data involves multiplication intensive (four real multiplications to realize the complex multiplication) operations. If the binary number system is employed in implementing the hardware for processing complex data, a comparatively large amount of time will be spent in the multiplication operations, and large register lengths will be required. Since the operation of multiplication does not require the generation of partial products in the RNS, the multiplication operations are performed at high speed. For situations where complex data has to be processed, the usual procedure in the RNS is to replace the conventional complex multiplication by modular operations. The Quadratic Residue Number System (QRNS) provides an alternative for processing complex data.

Complex RNS arithmetic has been studied, primarily as a means to increase efficiency in signal processing applications that are dominated by complex multiplications. To perform the complex multiplication, a method based on real index calculus has been proposed in [59]. In this

method, every complex multiplication has to undergo three stages of operations: index look-up, 2's complement adder, and correction/decoding. In this method, for a prime modulus P, the index addition has to be performed modulo $(p^2 - 1)$. The practical value of p is limited with in 61 because the number of address bits or the total size of the ROM required for the implementation is a major criterian.

A conventional modular approach in the RNS to performing the complex multiplication require that separate real and imaginary channels are to be formed; the multiplication of these complex quantities requires special considerations for handling the cross product terms. Recently it was shown that certain types of special number systems based on modular arithmetic admit representation of complex data as a set of element pairs (element plus conjugate). This representation leads to complete decoupling of the real and imaginary channels. This type of number system was first proposed by Nussbaumer for Fermat primes which are of the form $4K + 1$ [19], but the full exploitation of its potential was not explored. Recently in [20] the utilization of the above number system has been investigated for any prime moduli of the form $4K + 1$, and the number system has been termed the Quadratic Residue Number System (QRNS). Although the QRNS is undoubtedly well known in certain areas of mathematical literature, it has been applied to the area of digital filtering to process complex digital signals in [20,21] very recently. Using the QRNS we can perform complex multiplication with only two real RNS multiplications. For the price of encoding and decoding complex integers, the QRNS allows complex operations to be performed by two real operations.

This chapter concentrates on the development of the MQRNS by defining the quadratic residue ring. In order to show the algebraic structure of the complex numbers, the complex extension field will be discussed in the following section.

3.2 Complex Extension Fields

Let us consider a finite set of positive integers $p = (0,....p-1)$, together with mod p addition and multiplication forms a finite integer ring R(p) for any positive integer p. R(p) is a real ring because all elements of p are real integers. If p is a prime, the structure becomes a more sophisticated algebraic structure than a ring. To form a complex extension field, the results of theorem - 1 from Chapter II is used to determine the solution to the quadratic monic equation.

First consider the case in which the modulus is a prime of the form $p = 4K + 3$. Then the monic quadratic is non-solvable and the solution

$$\hat{i}^2 \equiv -1 \bmod p \tag{3.1}$$

cannot be found in GF(p). A complex modular structure, represented by the second degree extension field $GF(p^2)$, can be formed by taking ordered pairs

$$(a, b) \tilde{=} a + \hat{i}b, \quad a, b \in GF(p)$$

Now the binary operations of addition and multiplication in $GF(p^2)$ can be defined as follows:

$$\begin{aligned} (a,b) + (c,d) &= (Z_R, Z_I) \\ (a,b) \cdot (c,d) &= (Y_R, Y_I) \end{aligned} \tag{3.2}$$

where

$$Z_R = |a + c|_p$$
$$Z_I = |b + d|_p$$
$$Y_R = |ac - bd|_p$$
$$Y_I = |ad + bc|_p$$

The arithmetic operation defined in (3.2) is similar to conventional complex arithmetic, except that the real and imaginary components are computed with mod p arithmetic. The structure is a finite field containing p^2 elements and as such possesses all the algebraic properties of the complex modular fields. Such a complex field has interesting properties that may be useful in complex digitial signal processing. In this case, the complex multiplication requires four modular multiplications and two modular additions.

If p is of the form 4K + 1, then the monic quadratic equation is solvable in GF(p). This will generate the Quadratic Residue Number System (QRNS). Let us first discuss the conditions for generating the QRNS.

3.3 Quadratic Residue Number System

If j denotes the solution of the monic quadratic, then

$$j^2 \equiv -1 \bmod P \tag{3.3}$$

where $j \in GF(P)$. There exists a multiplicative inverse of j modulo P and is equal to -j.

Although we cannot build an extension field on the solution of (3.3), we can generate an extension ring [15], which we will call the quadratic

residue ring, in the following way:

$$\text{Let} \quad M = \prod_{i=1}^{L} P_i \quad \text{where} \quad P_i = 4K_i + 1$$

and let j be a solution to the monic polynomial $x^2 + 1 = 0$ over the ring R(M). This solution will exist because of the assumed form of M.

Define an extension element, $A^{(Q)} = (A, A^*)$ (3.4)

where $A = |a-jb|_M$ and $A^* = |a-jb|_M$, $a,b \in R(M)$ and $A, A^* \in R(M)$. We now define the quadratic residue ring as

$$QR(M) = \{A^{(Q)}: \ +, \cdot \} \tag{3.5}$$

where the binary operations are computed modulo M as follows:

Addition

$$A^{(Q)} + B^{(Q)} = (A+B, \ A^* + B^*) \tag{3.6}$$

Multiplication

$$A^{(Q)} \cdot B^{(Q)} = (A \cdot B, \ A^* \cdot B^*) \tag{3.7}$$

If the required binary operations are performed as shown in (3.6, 3.7) the resulting number system is called a Quadratic Residue Number System (QRNS).

Now let us show that the binary operations over QR(M) (3.6, 3.7) are isomorphic to the algebraic operations of the complex residue ring C(M) [21].

$$(a,b) \in R(M) \quad , \ (A \ , \ A^* \ ) \in \ R(M) \tag{3.8}$$

where

$$A = |a + jb|_M$$
$$A^* = |a - jb|_M$$

and

$$a = |2^{-1} (A + A^*)|_M$$

$$b = |2^{-1} \quad j^{-1} (A - A^*)|_M$$

Let $C1 = \alpha + \hat{i}\beta$ and $C2 = \gamma + \hat{i}\delta$ be the two complex numbers where $\hat{i} = \sqrt{-1}$

Then the addition is

$$C1 + C2 = |\alpha + \gamma|_M + \hat{i}|\beta + \delta|_M \qquad (3.9)$$

The element pairs C1 and C2 can be computed as:

$$A = |\alpha + j\beta|_M \qquad A^* = |\alpha - j\beta|_M$$

$$B = |\gamma + j\delta|_M \qquad B^* = |\gamma - j\delta|_M$$

Then

$$Q = |A + B|_M \qquad Q^* = |A^* + B^*|_M$$

Q, Q^* can also be computed directly from (3.9)

(i.e.) $$Q = | \, | (\alpha + \gamma) \; |_M + j| \; (\beta + \delta)|_M|_M$$

$$Q^* = | \, | (\alpha + \gamma) \; |_M - j| \; (\beta + \delta)|_M|_M$$

Similarly the multiplication of C1 and C2 can be computed as follows:

$$C1 \cdot C2 = |\alpha\gamma - \beta\delta|_M + \hat{i} \; |\alpha\delta + \beta\gamma|_M \qquad (3.10)$$

The multiplication using the elements of QR(M) can be computed as:

$$Q = |A \cdot B|_M \qquad Q^* = |A^* \cdot B^*|_M$$

The Q and Q^* can be computed from (7) directly as follows:

$$Q = | \, | (\alpha\gamma - \beta\delta)|_M + j| \, (\alpha\delta + \beta\gamma)|_M|_M$$

$$Q^* = \left| \left| (\alpha\gamma - \beta\delta) \right|_M - j \left| (\alpha\delta + \beta\gamma) \right|_M \right|_M$$

Therefore we can show that

$$(C(M), +, .) \cong (QR(M), +, .) \tag{3.11}$$

Due to this isomorphism, computations can be performed in either ring and results can be translated between the two. It is often more convenient to do complex computations in the QRNS where a complex multiplication can be performed by two real multiplications.

As an illustration of multiplication over QR(m), let us consider two complex numbers $a_i + j_i\, b_i$, $c_i + j_i\, d_i$: $a_i, b_i, c_i, d_i \in R(m_i)$ where $R(m_i)$ is the residue ring mod m_i, and j_i is the solution to $j_i^2 \equiv -1 \bmod m_i$, $j_i \in R(m_i)$, $i \in \{1, 2, \ldots L\}$

We can now compute the element pairs as follows:

$$A_i = |a_i + j_i\, b_i|_{m_i} \qquad A_i^* = |a_i - j_i\, b_i|_{m_i} \tag{3.12}$$

$$B_i = |c_i + j_i\, d_i|_{m_i} \qquad B_i^* = |c_i - j_i\, d_i|_{m_i} \tag{3.13}$$

if we let $$Q_i = |A_i \cdot B_i|_{m_i} \qquad Q_i^* = |A_i \cdot B_i|_{m_i} \tag{3.14}$$

then the real and imaginary parts of the product can be formed as

$$\begin{aligned} Y_{R_i} &= |2^{-1} \cdot (Q_i + Q_i^*)|_{m_i} \\ Y_{I_i} &= |2^{-1} \cdot j_i^{-1} \cdot (Q_i - Q_i^*)|_{m_i} \end{aligned} \tag{3.15}$$

Since the operations defined by (3.14) can be cascaded before the conversion defined by (3.15), the complex multiplication can be effectively performed with two real multiplications.

For computation over QR(M), we use the isomorphism

$$R(M) \cong QR(m1) \oplus QR(m2) \oplus \ldots\ldots\ldots \oplus QR(mL) \tag{3.16}$$

to provide L parallel computation elements where $M = \prod_{i=1}^{L} mi$ where the prime factors of M should be of the form 4K + 1.

In [22], the QRNS has been extended for composite numbers with prime factors of the form 4K + 1 and the diminished -1 binary code [60] has been suggested for efficient binary arithmetic.

In order to remove the restriction of the moduli in the QRNS, a modification to the QRNS has been termed the Modified Quadratic Residue Number System (MQRNS) [15]. The MQRNS is discussed in the following section.

3.4 Modified Quadratic Residue Number System

The primary disadvantage of the QRNS is the restriction of the form of moduli for RNS processing. For example, the only moduli which are acceptable for the QRNS and can be represented by 5 bits or less are 2,5,10,13,17,24,25,26 and 29. Since the RNS is defined for mutually prime moduli, the only acceptable 4 bits moduli are 10 and 13. These are the only moduli of 4 bits or less that can be used if dynamic range is to be maximized because 2 and 5 cannot be used with 10, hence (2,13), (5,13) and (10,13) are the only possibilities with (10,13) maximizing the dynamic range to 7.01 bits. For 5 bits, we may use 17,25,26, 29 and this moduli set will give a dynamic range of 18.29 bits. Although this moduli set is acceptable for the QRNS, the real and imaginary parts of the complex numbers cannot be recovered by using any even moduli. This constraint reduces the dynamic range of the moduli set using 4 and 5 bits to 17.29 bits, and the moduli set is 13,17,25 and 29. Since 25 is not a prime, this may introduce some difficulties if multiplicative inverse of 5 is needed in the calculation.

More recently Soderstrand and Poe have extended the QRNS to moduli of any form but with considerable dynamic range reduction. This number system has been termed the Quadratic-Like Residue Number System (QLRNS) [18] QLRNS is discussed later in this chapter.) Here when mapping the complex numbers into the QLRNS representation, the imaginary term is scaled by an appropriate factor and truncated to an integer. Thus the resolution of the imaginary term is reduced. This is a serious handicap. In QLRNS, the real and imaginary parts will have different resolution. In this thesis, the QRNS has been extended for moduli of any form with no resolution reduction but with an increase in RNS multiplications from 2 to 3, this number system has been termed the Modified Quadratic Residue Number System (MQRNS) [15,16].

Using a modulus, m, of any form except of the form $4K + 1$, the monic quadratic equation $x^2 + 1 = 0$ is irreducible in R(m). Clearly this will not admit a QRNS since we require -1 to be a quadratic residue. We will generalize the monic quadratic so that a solution (other than -1) exists over R(m). Let the extension element be

$$A^{(MQ)} = (A, A^*) \tag{3.17}$$

where $A = |a + \hat{j}b|_m$ and $A^* = |a - \hat{j}b|_m$, a, b ε R(m) and A, A^* ε R(m) with $\hat{j}$ as the solution to the monic quadratic $x^2 \pm n \equiv 0 \bmod m$

We can now define the modified quadratic residue ring as

$$MQR(m) = \{A^{(MQ)} \quad +, \cdot\} \tag{3.18}$$

where the binary operations are computed modulo m as follows:

Addition:

$$A^{MQ} + B^{MQ} = (A + B, A^* + B^*) \tag{3.19}$$

Multiplication

$$A^{(MQ)} \cdot B^{(MQ)} = \{(A \cdot B) - S, (A^* \cdot B^*) - S\} \tag{3.20}$$

where $S = \left|\left|\hat{j}^2 + 1\right|_m \cdot b \cdot d\right|_m$, b, d are the imaginary parts of the complex samples.

In the MQRNS, $|\hat{j}^2|_m \neq -1$ and so this particular constrain will alter the real component of the complex multiplication. In order to correct this we have to compute the term S, so as to obtain the correct value of the real component; this results in one additional multiplication over that required for the QRNS. It can be shown that the MQR(m) is isomorphic to the complex residue ring C(m).

The complex multiplication can be performed using the MQRNS as follows:

Consider two complex numbers $a_i + \hat{j}_i b_i$, $a_i + \hat{j}_i d_i$; a_i, b_i, c_i, $d_i \in R(m_i)$ and an integer $\hat{j} \equiv \sqrt{n} \bmod m_i$, $\hat{j}_i \in R(m_i), i \in \{1,2,\ldots.L\}$.

let $$D_i = \left|(\hat{j}_i^2 + 1)\right|_{m_i} \tag{3.21}$$

Calculate the element pairs as before:

$$A_i = \left|a_i + \hat{j}_i b_i\right|_{m_i} \qquad A_i^* = \left|a - \hat{j}_i b_i\right|_{m_i} \tag{3.22}$$

$$B_i = \left|c_i + \hat{j}_i d_i\right|_{m_i} \qquad B_i^* = \left|c_i - \hat{j}_i d_i\right|_{m_i} \tag{3.23}$$

let $Q_i = \left|A_i \cdot B_i\right|_{m_i}$ $Q_i = \left|A_i^* \cdot B_i^*\right|_{m_i}$ and $S_i = \left|D \cdot b_i \cdot d_i\right|_{m_i}$

(3.24)

The real and imaginary parts of the product can be formed as

$$Y_{R_i} = \left|\left|2^{-1} \cdot (Q_i + Q_i^*)\right|_{m_i} - S_i\right|_{m_i}$$

$$Y_{I_i} = \left|2^{-1} \cdot \hat{j}^{-1} \cdot (Q_i - Q_i^*)\right|_{m_i} \tag{3.25}$$

Instead of subtracting the term S_i as in (3.25), we can subtract it from Q_i and Q_i^* directly in (3.24). In this case, the real and imaginary parts of the product can be computed using (3.15). It is observed from (3.24) that the MQRNS complex multiplication can be performed with three real multiplications.

The MQRNS uses the isomorphism

$$R(M) \cong MQR(m_1) \oplus MQR(m_2) \oplus \ldots\ldots\ldots MQR(m_L) \qquad (3.26)$$

to provide L parallel computation elements where $M = \prod_{i=1}^{L} mi$.

3.5 Quadratic Like Residue Number System (QLRNS)

The concept behind the QLRNS is to identify within the valid RNS system a number which when squared yields a small negative number. Then the complex numbers in the QLRNS can be defined as follows [18]:

Let

$x + jy$ be a complex number

then $x + jy \simeq m + nj\sqrt{a}$

where $m = x$

$n \simeq y/\sqrt{a}$

The integer n will be obtained after rounding or truncation. The resolution or the dynamic range will thus be reduced by the length of the j vector. Once the complex numbers are decoded into the QLRNS, then the extension ring elements can be defined over the quadratic ring as in the QRNS. The binary operations of the extension ring can be computed by using (3.6,3.7) Although the QLRNS gained efficiency by using the QRNS complex arithmetic, the dynamic range was considerably reduced. For example, a 4 bit moduli

set {11,13,15,16 }, offers a resolution of 15 bits. If we use QLRNS, the resolution reduces from 15 to 12 bits [18].

In order to recover the real and imaginary parts, the even moduli are not allowed, because the multiplicative inverse of 2 and $(2j\sqrt{a})$ do not exist within the RNS. For example, if we consider the moduli set {11,13,15,16}, 16 is an even moduli. For such a system of moduli, the QLRNS is classified as the QLRNS2 [18]. In the QLRNS2, the conversion from the element pairs to real and imaginary part, must be performed by the scaling techniques which will require additional hardware.

One of the interesting features about the QRNS (and also the MQRNS, QLRNS) is that once the input has been transformed into a QRNS (MQRNS, QLRNS) representation, this form can be maintained throughout all computations, with conversion back to a conventional representation being performed at the end. This means that the additions associated with QRNS complex multiplications are, in effect, transparent, and only one or two actual additions are required for the multiplication depending on whether (3.25) or (3.24) is used for the mapping from the QRNS to RNS. The only extra overhead is the coding into and out of the QRNS (MQRNS, QLRNS) representations, at the beginning and end of a chain of calculations.

In the RNS, table look-up operations are often used for both addition and multiplication; in this case the total number of operations will be the most important consideration. Table 3a gives the number of multiplies and adds required in complex binary, QRNS, QLRNS and MQRNS.

Although both the QRNS and QLRNS require the same number of operations,

Table 3a

NUMBER OF MULTIPLIES AND ADDS REQUIRED IN COMPLEX BINARY, QRNS, QLRNS AND MQRNS

Number Systems	Number of Multiplies	Number Of Adds
Conventional	4	2
Conventional Using 3 Multiplies	3	5
QRNS and QLRNS	2	–
MQRNS	3	1

the QLRNS notation requires extra hardware to compensate for the dynamic range reduction and for converting the imaginary part into a scaled integer value [18]. From our work with the QLRNS the dynamic range reduction is typically at least equivalent to one modulus. There is also the problem of locating an extra modulus which maintains the property of minimizing the dynamic range reduction.

In order to demonstrate the complex multiplication in the QRNS and MQRNS, let us consider two examples in the following section.

3.6 Examples

3.6.1 Example - 1

Let us consider two complex numbers 2 + j3 and 1 + j4 and the modulus 5 where the operator j = 2. The binary operations of addition and multiplication on the complex numbers can be performed in the QRNS as follows: The element pairs can be computed as

$$A = |2 + 2 \cdot 3|_5 = 3 \qquad A^* |2 - 2 \cdot 3|_5 = 1$$

$$B = |1 + 2 \cdot 4|_5 = 4 \qquad B^* |1 - 2 \cdot 4|_5 = 3$$

The multiplication can be performed as

$$Q = |3 \cdot 4|_5 = 2 \qquad Q^* = |1 \cdot 3|_5 = 3$$

The real and imaginary part of the complex multiplication is

$$Y_R = |2^{-1} \cdot (2 + 3)|_5 = 0$$

$$Y_I = |2^{-1} \cdot 2^{-1} \cdot (2.3)|_5 = 1$$

The addition can be performed as

$$Q = |3 + 4|_5 = 2 \qquad Q = |1 + 3|_5 = 4$$

The real and imaginary part of the complex addition is

$$Y_R = |2^{-1} \cdot (2 + 4)|_5 = 3$$
$$Y_I = |2^{-1} \cdot 2^{-1} \cdot (2 - 4)|_5 = 2$$

It can be verified that the direct computation of the complex multiplication and addition modulo 5 yield the same results.

3.6.2 Example -2

Consider two complex numbers $2 + \hat{j}6$ and $3 + \hat{j}2$ and a modulus 7 where the operator $\hat{j} = 1$. The binary operations of multiplication and addition can be performed in the MQRNS as follows:

The element pairs can be computed as

Let $D = |1 + 1| = 2$

$$A = |2 + 6|_7 = 1 \qquad A^* = |2 - 6|_7 = 3$$
$$B = |3 + 2|_7 = 5 \qquad B^* = |3 - 2|_7 = 1$$

The complex multiplication can be done as

$$Q = |1 \cdot 5|_7 = 5 \qquad Q^* = |3 \cdot 1|_7 = 3 \qquad S = |2 \cdot 6 \cdot 2|_7 = 3$$

The real and imaginary parts of the complex multiplication are

$$Y_R = ||2^{-1} \cdot (5 + 3)|_7 - 3|_7 = 1$$
$$Y_I = ||2^{-1} \cdot 1^{-1} \cdot (5 - 3)|_7 - 3|_7 = 1$$

The multiplication can be done in another way as follows:

$$Q = ||1 \cdot 5|_7 - 3|_7 = 2 \qquad Q^* = ||3 \cdot 1|_7 - 3|_7 = 0$$

The real and imaginary parts are

$$Y_R = |2^{-1} \cdot (2 + 0)|_7 = 1$$
$$Y_I = |2^{-1} \cdot 1^{-1} \cdot (2 - 0)|_7 = 1$$

The complex addition can be performed as

$$Q = |1 + 5|_7 = 6 \qquad Q^* = |3 + 1|_7 = 4$$

The real and imaginary part of the complex addition is

$$Y_R = |2^{-1} \cdot (6 + 4)|_7 = 5$$

$$Y_I = |2^{-1} \cdot 1^{-1} \cdot (6 - 4)|_7 = 1$$

It can be verified that the direct computation of the complex multiplication and addition yield the same results.

3.7 Summary

The Quadratic Residue Number System has been defined over the Quadratic residue rings. The Modified Quadratic Residue Number System (MQRNS) has been defined over the modified residue ring. A brief review of the Quadratic Like Residue Number System (QLRNS) has been presented. The isomorphism between the conventional complex ring and the quadratic and modified quadratic residue rings has been established.

The MQRNS has been defined for any moduli. The complex multiplication in the QRNS requires an additional multiplication compared to the QRNS or QLRNS. Although the QLRNS requires the same number of operations as the QRNS, the QLRNS requires extra hardware to compensate for the dynamic range reduction and for converting the imaginary part into a scaled integer value. In working with the QRNS, the dynamic range reduction is typically at least equivalent to one modulus. If the QLRNS2 [18] is used, the conversion from element pairs to real and imaginary parts requires scaling operation which will reduce the speed of the system.

In viewing all the above characteristics of the QLRNS, in order to offer a complete choice of moduli, the MQRNS has been offered as an alternative to the QRNS.

Chapter IV

HARDWARE IMPLEMENTATION OF THE QUADRATIC RESIDUE NUMBER SYSTEMS AND BINARY TO RESIDUE AND RESIDUE TO BINARY CONVERSION TECHNIQUES

4.1 Introduction

In general, signal processing algorithms consist of repeated multiplications and additions. In the early years, the hardware implementation of most of the digital signal processing algorithms was developed using the binary number system (fixed point and floating point). Binary multiplication [43,44,83] requires more time and more hardware than binary addition, and so dominates the speed and cost. In the Residue Number System (RNS), the binary operations of addition or subtraction have no inter-digit carries or borrows, and multiplication does not generate paritial products. In fact, in some hardware realizations multiplication and addition have identical speed and cost, and, in certain cases, multiplication by constants can be a 'free' operation. Hence arithmetic operations can be performed at high speed using the RNS [10,35,36,40,50]. The primary advantage of the Residue Number System (RNS) is that the binary operations of addition, subtraction and multiplication can be performed on the respective residue digits in L independent parallel sections. Then the Residue digits can be mapped into the binary number system using the Chinese Remainder Theorem (CRT).

Although arithmetic operations can be performed at a very high speed, there are difficulties in implementing some operations. Division, scaling, sign detection and magnitude comparison are such operations. General division in the RNS is a complicated process. Scaling is easier

than general division, but in scaling the divisor is limited to a predetermined constant. For an efficient implementation of scaling, the constant scale factor must be a product of some of the moduli [1,79], or powers of 2 [42]. A restricted form of division of this type is usually required in digital filters to prevent overflow and to optimize the signal-to-noise ratio in various parts of the filter.

In the following section some hardware implementations of RNS operations are presented.

4.2 Hardware Implementation of RNS Operations

For the implementation of arithmetic operations in hardware, several basic approaches are available. They are classified as follows:

1. Logical implementation of the Boolean function directly specified by the operation.

2. A combination of binary elements and look-up tables for precomputed functions.

3. Storing software algorithms in a general purpose computer.

The second method is more suitable than the other two in the case of the RNS, and many schemes are available to perform the addition, subtraction and multiplication. For example, a scheme consisting of a binary adder or subtractor followed by a look-up table to perform the operation of addition or subtraction or multiplication, has been presented in [73] (see Fig. 4.4). In this approach, the memory space is used more efficiently than the direct storage of the look-up table for the arithmetic operations.

In another example, the well known quarter square multiplier is considered. The quarter square multiplier can be expressed as:

$$a \cdot b = [(a + b)^2 - (a - b)^2]/4$$

This operation can be implemented using the binary addition and subtraction with the square function stored in ROMs. The final correction and divide by 4 are stored in an output ROM. A complete design of a large moduli multiplier is presented in [31] showing the pipeline structure of a 28 MHz throughput filter for the moduli of the form 2^{N+1}, 2^{N}, 2^{N-1} using the quarter square multiplier.

The recent breakthroughs in the area of high density Read-Only Memory (ROM) technology and the continuous reduction in the memory cost have made the above approach more attractive . In the direct look-up table approach, a memory size of $(2^{2B}$ XB) bits is required to store the look-up table for a modulus of B bits; that is each modulus m_i, requires m_i^2 entries in the table.

The data width B is given by

$$B = \lceil \log_2 m_i \rceil \tag{4.1}$$

where $\lceil\ \rceil$ represents the ceiling function for integer truncation.

The combination of binary elements and look-up table approach requires a memory size of $(2^{B+1}$ XB) bits. In the case of the direct look-up table approach, for large modulus this requirement tends to be very high but if the modulus is limited to six bits, currently available single chip memory packages can be used to store a look-up table. For example, binary operations for moduli $m_i < 64$, can be implemented using (4K X 8), (1K X 8) and (256 X 4) bit commercially available PROM packages, respectively. The look-up table approach is not feasible in the case of binary number system because of the enormous storage requirements.

For example, the ROM structure of Fig. 4.1 illustrates a modulo addition and subtraction for the modulus 61, followed by a residue multiplier to implement the function $||a + b|_{61} \cdot |c - d|_{61}|_{61}$. Since each residue can be represented by a maximum of 6 bits, the total of two inputs to each look-up table is 12 bits, and the output is taken from 6 of the 8 output bits. Another advantage of the look-up table approach is that it allows easy pipelinging [14]. For every latch pulse, the output of each ROM is stored and becomes part of the address for the next ROM. The only control signal required to clock the pipeline is a latch pulse. For every latch pulse, new input is accepted, and new output is generated. The throughput rate of the array is equal to the inverse of the ROM access time plus latch settling time. Using ECL, bipolar or HMOS devices, and moduli sizes from 5 to 12 bits in width, data rates of 3 to 100 MOPS (Million Operations Per Second) can be achieved [95].

The direct use of ROM for stored table look-up residue arithmetic leads to memory intensive structures [70]. Memories which are manufactured using TTL technology, tend to use a large amount of power. NMOS or CMOS memories reduce the amount of power dissipation. The tradeoff in using the TTL and NMOS or CMOS chips is throughput and power dissipation. In TTL the throughput is high and the power dissipation is also high, but in NMOS or CMOS the throughput is low and the power dissipation is also low.

The look-up table approach can provide great savings in hardware if some of the operands in a chain of binary operations are known constants. For example, the function

$$y_i = |[(x_1 \odot k_1) \odot (x_2 \odot k_2)] \odot k_3|_{m_i}$$

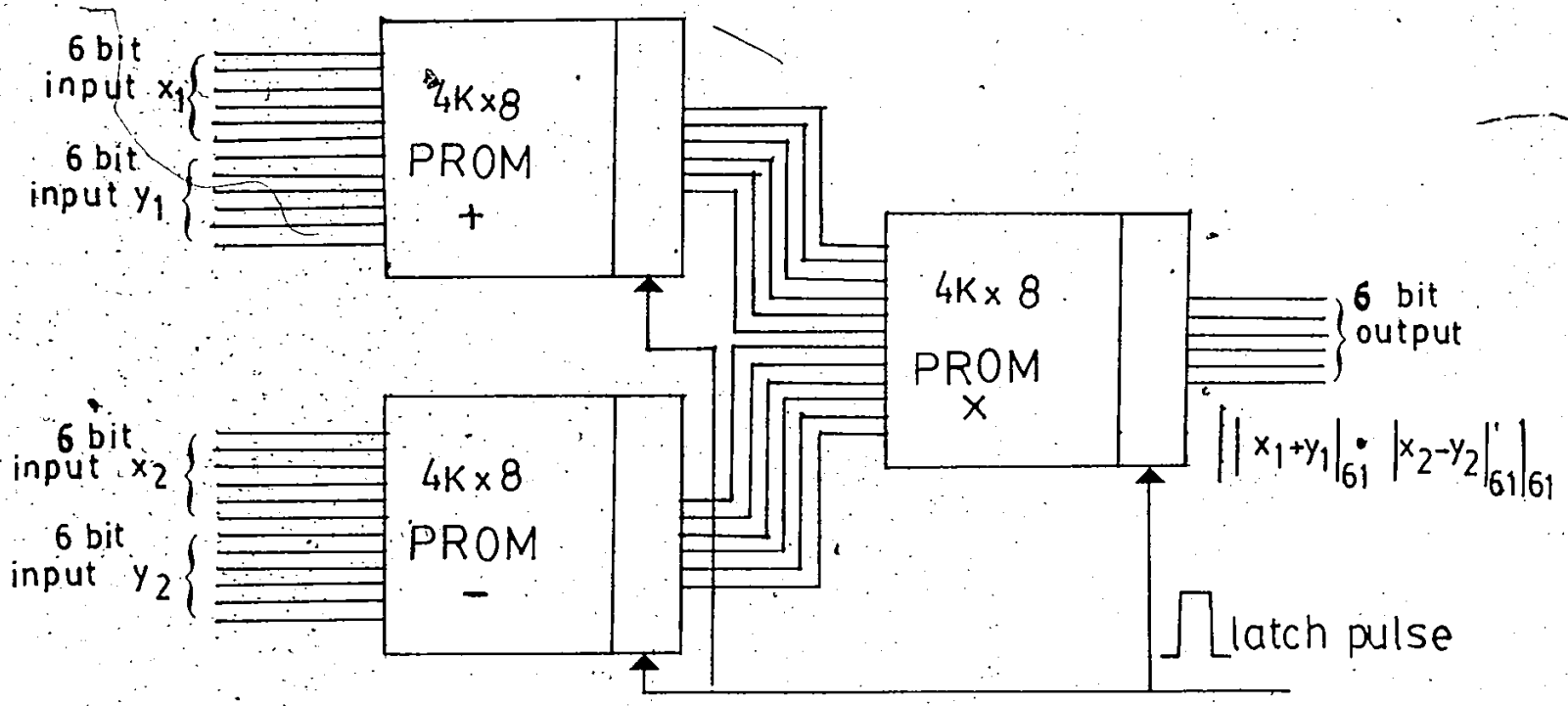


Fig. 4.1 Pipe-lining Array

where $\odot$ represents the binary operations of addition (subtraction) or multiplication and x_1, x_2, and y are variables and k_1, k_2 and k_3 are constants is shown in Fig. 4.2. The function can be computed with just one look-up table in one look-up cycle without any additional hardware [1]. This has great significance in many of the digital signal processing operations. In order to use the currently available hardware, restrictions have to be placed on the size of the maximum modulus in a system. If we restrict the modulus to a maximum size of 32, then the following table gives the moduli in descending order. (Table 4a) The moduli are relatively prime, and the range associates with the product of each modulus and all the preceding moduli [1].

Table 4a A Set of Moduli and the Product of the Moduli

i	m_i	M
0	32	1.00×2^5
1	31	1.94×2^9
2	29	1.76×2^{14}
3	27	1.48×2^{19}
4	25	1.16×2^{24}
5	23	1.66×2^{28}
6	19	1.98×2^{32}
7	17	1.05×2^{37}
8	13	1.71×2^{40}
9	11	1.17×2^{44}
10	7	1.03×2^{47}

The above explained look-up table approach can be used for the implementation of the QRNS and MQRNS.

In complex digital signal processing hardware, based on the QRNS or

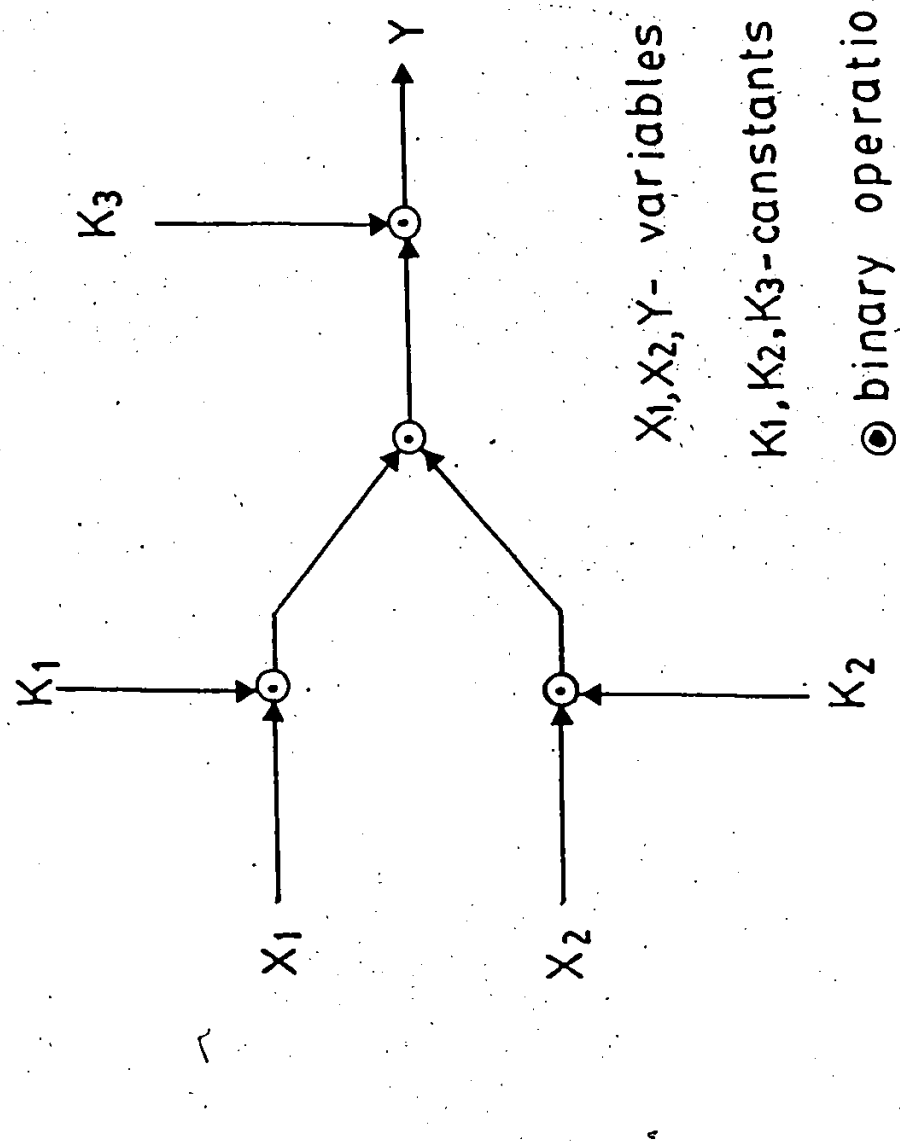


Fig. 4.2 Binary operations that can be combined into one look-up table.

MQRNS, the binary or analog input signals must first be converted to the QRNS or MQRNS representation. Then the filtering operation is performed in the QRNS or MQRNS representation. Afterwards the QRNS or MQRNS represenation is converted to the RNS representation and then the RNS representation is converted back to the binary representation. Clearly we require to look at the coding and de-coding hardware, and in the following section the concepts of Binary to Residue interface techniques are discussed.

4.3 Conversion From Binary to Residue Digits

Binary to residue conversion can be easily implemented for L moduli in L independent channels using the L ROMs. The ith residue of a number X is derived by

$$x_i = \left| \sum_{k=1}^{B} a_k 2^k \right|_{m_i} \qquad (4.2)$$

where $(a_1, a_2, \ldots . a_B)$ are the B bits of the binary representation of X.

Assuming that we have available ROMs with 2 addressable locations, then the binary to residue conversion can take place in one look-up cycle. For example, if we wish to convert a 10 bit number into the residue form, then for $m < 256$, L 8K ROMs (organized as 1K x 8 bits) can be used where L is the number of parallel channels in the RNS. An alternate procedure for coding large values of B is to split the binary representations, code each part and recombine by summing in the residue representation.

For example,

$$x_i = \left| \left| \sum_{k=0}^{\frac{B}{2}-1} a_k 2^k \right|_{m_i} + \left| \sum_{k=B/2}^{B-1} a_k 2^k \right|_{m_i} \right|_{m_i} \qquad (4.3)$$

In this case the coding operation requires 3L ROMs and 2 look-up cycles. If we assume a two's complement binary representation, the sign numbers can be coded by treating the part containing a_{k-1} as signed numbers and the remaining part as unsigned positive numbers.

4.4 Conversion From the RNS to the QRNS or MQRNS Representation

In order to convert the RNS representation to the QRNS or MQRNS representation, the elements of the quadratic residue ring QR(M) or the modified quadratic residue ring MQR(M), A, A^* (3.17) have to be computed. We require 2 ROMs to compute A and A^*. In the hardware implementation of the MQRNS, an interesting feature is present which will help to reduce the memory requirements to a reasonable size [15]. The calculation of the integers A and A^* can be implemented by using a single multiplexed ROM. There is, however, a restriction on the form of the modulus. If the modulus can be represented exactly by $2^B - 1$ (where B = number of bits), a negative integer $(-x) \in R(m)$ can be expressed simply by taking the 1's complement of x. Hence the table look-up for $A = |a + \hat{j}b|_m$ can be used to obtain $A^* = |a - \hat{j}b|_m$ as shown below.

For example let us consider a complex sample as

$$a + \hat{j}b = 5 + \hat{j}6$$

Let the prime modulus m and the operator $\hat{j}$ be 7 and 6 respectively.

Then
$$A = |a + \hat{j}b|_m = |5 + 6 \cdot 6|_7 = 6$$
$$A^* = |a - \hat{j}b|_m = |5 - 6 \cdot 6|_7$$
$$= |5 + 1 \cdot 6|_7$$
$$= 4$$

Table 4b

The Table Look-Up to Obtain A and A^* in the MQRNS

CM

	0 (0)	1 (6)	2 (5)	3 (4)	4 (3)	5 (2)	6 (1)	7 (0)
0	0	6	5	4	3	2	1	0
1	1	0	6	5	4	3	2	1
2	2	1	0	6	5	4	3	2
3	3	2	1	0	6	5	4	3
4	4	3	2	1	0	6	5	4
5	5	(4)	3	2	1	0	(6)	5
6	6	5	4	3	2	1	0	6

CM=COMPLEMENT/MULTIPLEX

Table 4c

The Table Look-Up to Obtain A and A^* in the QRNS

CSM

	0 (0)	1 (2)	2 (4)	3 (1)	4 (4)	5 (0)
0	0	2	4	1	3	0
1	1	3	0	2	4	1
2	2	(4)	1	3	(0)	2
3	3	0	2	4	1	3
4	4	1	3	0	2	4

CSM=COMPLEMENT/SUBTRACT /MULTIPLEX

The implementation of this computation is explained in Table 4b.

This approach can be extended to the other types of prime moduli as follows: Let us consider the prime moduli of the form 4K + 1; for example m = 5. A negative integer $-x = |5 - x|_5 = |(7 - x) - 2|_5$. Therefore the operation consists of first taking the ones complement (7 - x) and then subtracting 2. In the case of the QRNS, in obtaining A and A^*, there is additional hardware required in implementing the complement/ subtract/ multiplex operation, over that required for the restricted MQRNS implementation.

For example, let us consider a complex sample as

$$a + jb = 2 + j4$$

Let the modulus m and the operator j be 5 and 2 respectively.

Then $$A = |a + jb|_m = |2 + 2 \cdot 4|_5 = 0$$

$$A^* = |a - jb|_m = |2 - 2 \cdot 4|_5$$

$$= |2 - 2|(7 - 4) - 2|_5|_5$$

$$= 4$$

Table 4c gives the computation of A and A^* in the case of the QRNS.

After performing the filtering operation on the QRNS or MQRNS represenatation, the RNS representation of the real and imaginary parts have to be computed using (3.15, 3.25). For the hardware implementation of this operation, the look-up table approach can be used. After this the real and imaginary samples have to be converted back to binary representation. The conversion from residue to binary is more difficult than

the conversion from binary to residue. In particular, in the QRNS, it has been presented in [20,21] that after computing the real and imaginary parts from the element pairs in the individual channels of the RNS, the CRT is used to convert the complex residue digit to complex binary digits. In the QRNS and MQRNS representations, we will present techniques for the direct decoding of element pairs into a binary representation using the Chinese Remainder Theorem (CRT) and mixed-radix conversion algorithms. Such decoding techniques will help to save memory space in the look-up table implementation.

4.5 Use of Chinese Remainder Theorem (CRT) on Residue Codes

Implementing the CRT on residue code and conjugate pairs is illustrated as follows for three cases. Case - 1, MQRNS; case - 2, QRNS; case - 3, a combination of the QRNS and MQRNS.

As is well known, the CRT [79] can be used to convert RNS coded numbers to binary coded numbers as follows:

$$B_{m_i} = \hat{m}_i \cdot |r_i \cdot \hat{m}_i^{-1}|_{m_i}$$

$$X = \left| \sum_{i=1}^{L} B_{m_i} \right|_M \tag{4.4}$$

where

$$M = \prod_{i=1}^{L} m_i$$

$\hat{m}_i = M/m_i$

X = binary representation

L = number of moduli

m_i = ith modulus

r_i = the ith RNS digit.

4.5.1 Case -1

Let m_i be moduli of any form,

$a + \hat{j}b$, $c + \hat{j}d$ are two complex elements in $R(m_i)$.

$i \in \{1, 2, \ldots L\}$

To obtain the value of the operator $\hat{j}$, the following theorem can be used.

Theorem: -2 Let $M = \prod_{i=1}^{L} m_i$ where m_i are ordered moduli with $m_i < m_i + 1$, If $\hat{j}$ is a solution to a quadratic monic polynomial modulo m, then the same value of $\hat{j}$ will be a solution to a quadratic monic polynomial modulo

m_2, m_3, ... or m_L or M.

The proof: The quadratic residue is found from

$$x_i^2 \equiv n_i \bmod m_i \quad \text{with } x, n \in Z\ m_i \text{ [80]}. \tag{4.5}$$

Clearly, for each x, the following holds

$$x_i^2 \equiv (-x_i)^2 \equiv n_i \bmod m_i$$

and

$$x_i^2 \equiv (m - x_i)^2 \equiv n_i \bmod m_i \tag{4.6}$$

for which there are exactly $(m - 1)/2$ non-zero solutions [$x_i \in \{1,2,\ldots\ldots(m_i - 1)/2\}$] [80]. Hence there are exactly $(m_i - 1)/2$ ordered quadratic residues and $[(m_i - 1) - (m_i - 1)/2] = (m_i - 1)/2$ quadratic non-residues.

Since the $\{m_i\}$ are all ordered, the solution to the quadratic monic polynomial

$$x^2 \equiv n_1 \bmod m_1 \tag{4.7}$$

will also be a solution to the quadratic monic polynomial

$$x^2 \equiv n_i \bmod m_i \qquad m_i \in \{i = 2, 3....L\} \tag{4.8}$$

The same solution will also map to the value in the ring, modulo M, isomorphic to $Zm_1 \oplus Zm_2 \oplus .....\oplus Zm_L$.

Hence it will be a solution to the quadratic monic polynomial modulo M.

Even though this theorem is valid for moduli of the form $4K_i + 1$, we do not have the advantage of choosing the operator j according to this theorem. We will therefore consider case -1 to be for moduli of anyform except $4K + 1$. Having established the value of the operator, $\hat{j}$, from theorem 2,

let

$$D_i = |\hat{j}^2 + 1|_{m_i} \tag{4.9}$$

The element pairs are

$$\begin{aligned} A_i &= |a + \hat{j}b|_{m_i} & A_i^* &= |a - \hat{j}b|_{m_i} \\ B_i &= |c + \hat{j}d|_{m_i} & B_i^* &= |c - \hat{j}d|_{m_i} \\ S_i &= |D_i \cdot b \cdot c|_{m_i} \end{aligned} \tag{4.10}$$

The binary operations over the element pairs in the MQRNS can be defined as

$$(Q_i, Q_i^*) = (A_i , A_i^*) \ominus (B_i , B_i^*) \tag{4.11}$$

where $\ominus$ represents the binary operations of addition, subtraction or multiplication.

Then the addition is

$$\begin{aligned} Q_i &= |A_i + B_i|_{m_i} \\ Q_i^* &= |A_i^* + B_i^*|_m \end{aligned} \tag{4.12}$$

and the multiplication is

$$\begin{aligned} Q_i &= |A_i \cdot B_i|_{m_i} - S_i|_{m_i} \\ Q_i^* &= |A_i^* \cdot B_i^*|_{m_i} - S_i|_{m_i} \end{aligned} \tag{4.13}$$

Using (4.4) on the residue codes in (4.13) we obtain

$$\begin{aligned} Q &= \Big| \sum_{i=1}^{L} \hat{m}_i |Q_i \cdot \hat{m}_i^{-1}|_{m_i}|_M \\ Q^* &= \Big| \sum_{i=1}^{L} \hat{m}_i | Q_i^* \cdot \hat{m}_i^{-1}|_{m_i}|_M \end{aligned} \tag{4.14}$$

Hence the decoded value of the real and imaginary parts are

$$\begin{aligned} \text{Real} &= |2^{-1} (Q + Q^*)|_M \\ \text{Imaginary} &= |2^{-1} \cdot \hat{j}^{-1} (Q - Q^*)|_M \end{aligned} \tag{4.15}$$

4.5.2 Case -2:

Let $m_i = 4K_i + 1$

$a + j_i b$ and $c + j_i d$ are the two complex elements in $R(m_i)$

where $j_i^2 \equiv -1 \bmod m_i, j_i \in R(mi)$ and $i \in \{1,2,....L\}$

Let A_i, B_i, A_i^*, B_i^* be the element pairs of the two complex

numbers which are computed as:

$$A_i = |a + jb|_{m_i} \qquad A_i^* = |a - jb|_{m_i}$$
$$B_i = |c + jd|_{m_i} \qquad B_i^* = |c - jd|_{m_i} \tag{4.16}$$

The binary operations over the element pairs can be defined as in (4.10). Hence

$$Q_i = |(A_i \quad B_i)|_{m_i}$$
$$Q_i^* = |(A_i^* \quad B_i^*)|_{m_i} \tag{4.17}$$

Using (4.4) on the residue codes in (4.17) we obtain

$$Q = \left| \sum_{i=1}^{L} \hat{m}_i |Q_i \cdot \hat{m}_i^{-1}|_{m_i} \right|_M$$
$$Q^* = \left| \sum_{i=1}^{L} \hat{m}_i |Q_i^* \cdot \hat{m}_i^{-1}|_{m_i} \right|_M \tag{4.18}$$

Hence the decoded value of the real and imaginary parts are

$$\text{Real} = |2^{-1} (Q + Q^*)|_M$$
$$\text{Imaginary} = |2^{-1} \cdot j^{-1} (Q - Q^*)|_M \tag{4.19}$$

Where the operator j can be precomputed using CRT.

4.6 Use of Mixed Radix Conversion on Residue Codes

In implementing the CRT technique, we need a modulo M-adder-shifter [2]. Modulo M adders ($M \neq 2^B$) are not available commercially and implementation by using commercial packages is expensive (circuits to detect wraparound and correct). Since M is generally very large, the problem associated with the modulo M adder can be handled if we employ the mixed

radix conversion techniques proposed in [5].

The method of translating the residue samples into the mixed-radix form with the RNS moduli and obtaining the natural integer, X can be explained using the following expression:

$$X = a_{N-1} \prod_{k=0}^{N-2} m_k + \ldots\ldots\ldots + a_2 m_0 m_1 + a_1 m_0 + a_0 = \sum_{i=0}^{N-1} a_i P_i \qquad (4.20)$$

where $P_0 = 1 \qquad P_i = \prod_{k=0}^{i-1} m_k$

and $\{a_i\}$ are mixed radix digits with the range $0 \leq a_i < m_i$. The multiplication in (4.20) can be eliminated by applying the bit-slice technique of Peled and Liu as shown in [5]. The mixed radix digits can be computed using the recursive algorithm in [1].

The element pairs of the QRNS or MQRNS are converted to the mixed radix form associated with the RNS moduli using the expression (4.20) as allows:

$$\begin{aligned} Q &= a_{N-1} \prod_{k=0}^{N-2} m_k + \ldots\ldots\ldots + a_2 m_0 m_1 + a_1 m_0 + a_0 = \sum_{i=0}^{N-1} a_i P_i \\ Q^* &= a_{N-1} \prod_{k=0}^{N-2} m_k + \ldots\ldots\ldots + a_2 m_0 m_1 + a_1 m_0 + a_0 = \sum_{i=0}^{N-1} a_i P_i \end{aligned} \qquad (4.21)$$

4.7 An Improved Algorithm for Residue to Binary Conversion

Even though the problems associated with using modulo M adders in the residue to binary conversion is solved using the mixed radix conversion technique, we still have to compute the binary version of the real and imaginary parts (4.15, 4.19). This will require another two modulo M adders. Therefore, for high speed implementation, the mixed radix conversion cannot solve the problem fully. An alternate viable approach to

implementing the modulo M adder operation is to use a conversion technique based on the CRT, proposed in [4]. This fractional method is fast with a slight reduction in the dynamic range and requires the least hardware of all methods so far considered. The fractional method is explained as follows:

Divide both sides of (4.4) by M, the product of the moduli.

We obtain

$$X/M = FRAC\ [\sum_{i=1}^{L} B_{mi}/M]$$

$$= FRAC\ [\sum_{i=1}^{L} b_{mi}] \qquad (4.22)$$

where

$$bm_i = |Bm_i/M = \hat{m}_i/M|_{r_i}\ |\tilde{m}_i^{-1}|_{m_i},$$

FRAC is the fractional part of the quantity enclosed in brackets. Thus we have converted the mod M operation into a simple FRAC operation.

The key point from the hardware point of view is that (4.22) defines mapping between the RNS digit r_i and the intermediate digits bm_i. We are effectively mapping $R(m_i) \cong R(2^{\lceil \log_2 m_i \rceil})$. Note this mapping is not isomorphic in that we do not have a one-to-one correspondence in both directions. Conversion from r_i to b_{mi} can be implemented by a single level of ROM table look-up. The ROM also performs part of the addition.

Fig. 4.3 shows the hardware implementation of a 4 moduli RNS to binary conversion based on the fractional algorithm.

The operation of the [illegible]re in the Fig. 4.3 can be explained [4] as follows:

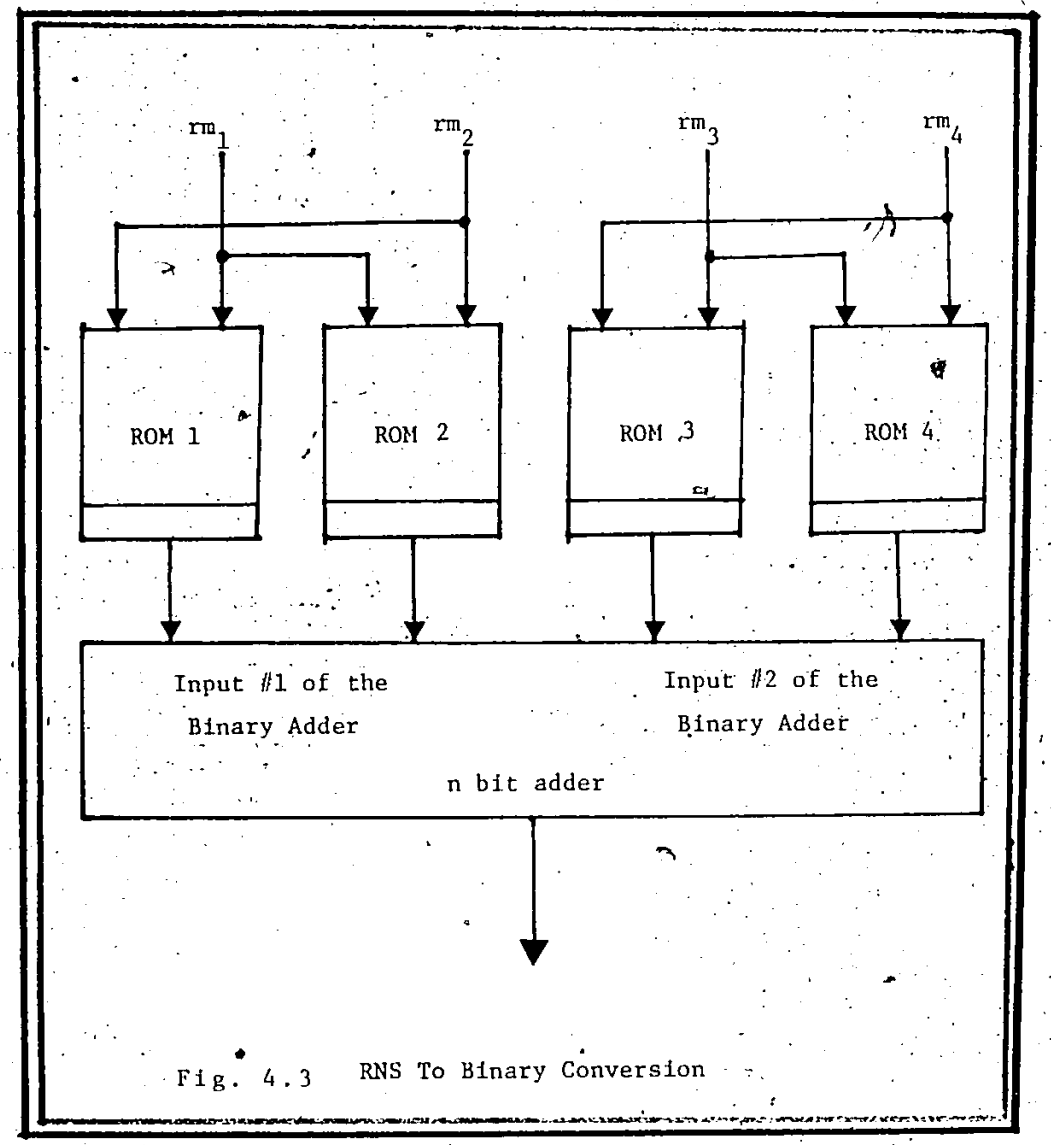


Fig. 4.3 RNS To Binary Conversion

1) ROMs 1 and 2 calculate b_{m1} and b_{m2} from r_{m1} and r_{m2} respectively, and then present the n-bit sum $b_{m1} + b_{m2}$ to input number 1 of the n-bit binary adder. The overflow in the n-bit adder is ignored, because the overflow bit corresponds to integer M, the product of the moduli, and the content of the adder is in fractional form.

2) ROMs 3 and 4 calculate b_{m3} and b_{m2} from r_{m3} and r_{m4} respectively, and then present the n-bit sum b_{m3} to b_{m4} to input number 2 of the n-bit binary adder. The overflow in the n-bit adder is ignored, because the overflow bit corresponds to integer M, the product of the moduli, and the content of the adder is in fractional form.

3) The output of the n-bit adder must be multiplied by M to obtain the correct integer result, and this can be accomplished in the final conversion.

The fractional method can be explained using a simple numerical example.

Example:

Consider a moduli set {2, 3, 5, 7} and the residue representation as (1, 1, 2, 4)

Using the CRT, the binary representation of the above residue set is 67. The binary conversion according to the proposed algorithm is as follows:

$$M = 210$$

$\hat{m}_1/M = 0.5$ $\qquad$ $\hat{m}_2/M = 0.3333333$

$\hat{m}_3/M = 0.2$ $\qquad$ $\hat{m}_4/M = 0.1428571$

The following Table 4d gives the binary representation of the residue set by considering the number of bits to represent the fractional value.

It can be observed from the table that 4d if we use 8 bits and above to represent the fractional value, the binary value of the residue set is almost the same as the value obtained using direct implementation of the CRT. The modulo M addition is entirely lifted in this improved method by means of fractional (binary) addition, and, hence memory size is not a problem.

Table 4d

Binary Representation of the Residue Digit

No. of Bits	Binary Representation	2^{-n}, n=1 ... 16
1	105	0.5
2	157.5	0.25
3	26.25	0.125
4	52.5	0.0625
5	52.5	0.03125
6	62.36475	0.015625
7	63.984375	0.0078125
8	65.625	0.00390625
9	66.4453125	0.001953125
10	66.5611390625	0.0009765625
11	66.7529296875	0.00048828125
12	66.9580078125	0.000244140625
13	66.9580078125	0.0001220703125
14	66.9714551953125	0.00006103515625
15	66.99064442856	0.000030517578125
16	66.9970853414415	0.0000152587890625

same amount of hardware for implementation and give almost the same data rate. This aspect will be discussed in more detail in Chapter V.

The following section will briefly discuss the implementation of the QRNS and MQRNS operations using Very Large Scale Integration (VLSI) technology.

4.8 Implementation of the QRNS and MQRNS Using VLSI

With the rapid advances in Large Scale Integration (LSI) and Very Large Scale Integration (VLSI), a growing number of complex digital signal processing applications are becoming economically feasible. The desirable properties of the LSI and VLSI systems are: small size, low power, low cost, noise immunity and reliability. In general, the currently available integrated microprocessors do not have the special hardware capability for complex arithmetic, but rather they depend on a softeware implementation of complex arithmetic.

Due to advances in semiconductor fabrication technology, it has become possible to fabricate large scale integrated semiconductor arrays containing tens of thousands of gates. This integrated technology has moved from Large Scale Integration (LSI) to Very Large Scale Integration (VLSI). VLSI is characterized by the number of gate count per chip ranges 10^5 - 10^6 or it can be characterized by the physical dimension of a transistor channel. With VLSI technology it will soon be possible to implement entire computing systems with 32 bits and associated memories on one monolithic silicon chip. This high density semiconductor technology has some constraints including power dissipation, Input/Output (I/O) pin counts, relatively long communication delays, and difficulty in design and

layout. All these important problems in VLSI, are much less critical in other technologies. As compensation for these problems, however, VLSI offers very fast and inexpensive computational elements with some unique and exciting properties. The advent of VLSI circuit technology offers new opportunities and poses new design problems in implementing digital signal processing architectures. A tremendous amount of potential is available in VLSI to bring a single chip to perform the complex multiplication based on the Quadratic Residue Number Systems techniques.

The well known advantage of the RNS for digital signal processing applications is that high speed integer addition, subtraction and multiplication can be performed carry free. These binary operations can be performed in parallel with independent processing within each modulus. Another advantage of the RNS is the adaptability to look-up table implementation. Hence, the above binary operations can be performed in a parallel architecture using high density ROM arrays. These attributes are quite suitable for VLSI implementations because:

1) VLSI supports highly parallel computational architectures
2) Memories have the largest transistor packing density of all logical functions

The circuit used to store information, the memory cell, is a prime element in determining semiconductor memory cost. Desirable characteristics of the memory cell are:

i) small area
ii) high bit storage density
iii) light loading to the drive and sense circuitry

iv) the ability to write and read with small delay

v) the ability to retain information with low power dissipation.

Metal-Oxide Semiconductor Field Effect (MOSFET) devices are attractive for memory compared to bipolar devices. The MOSFET devices have a number of advantages for memory users. Of these the most important is the simpler processing, allowing high yields and a greater degree of integration resulting in lower cost. The layout efficiencies, and small size of MOSFET devices, also contribute to achieving lower cost. Memory is particularly sensitive to cost since it must be cheaper than logic, or else it would be replaced by logic.

In order to use the drive and sense circuitry efficiently, it is desirable to put as many cells on a given line as possible. The loading of the MOSFET device is mainly capacitive. To avoid deterioration of the array performance over that of an individual cell by introducing long time constants, the array may be driven and sensed by low impedance bipolar transistor circuits [92]. Thus the array, numerically the predominant portion of the memory system, is made in high yield MOS technology, while only the relatively small number of peripheral circuits is made in bipolar technology [92]. Thus the advantages of both technologies may be achieved simultaneously. For many applications, the optimum system is obtained by interfacing MOSFET array chips with bipolar support chips [92,93,94].

The traditional way to achieve high density memory chips is to shrink design rule geometries and channel length of the memory cell. The important system parameters for a cell are:

a) channel length (l)

b) transit time (τ)

c) switching power (Esw)

d) system clock period (T).

Table 4e summarizes values of the above system parameters of a cell for technology in 1978 and for future technology according to Mead and Conway [82].

Table 4e

VLSI System Parameters

	1978	19XX
Minimum Feature Size	6 μm	0.3 μm,
τ	0.3 to 1 nsec	0.02 nsec
Esw	10^{-12} joule	$2X\ 10^{-16}$ joule
System Clock	30 to 59 nsecs.	2 to 4 nsecs.

Some of the phenomena that are apparently likely to limit the functioning of miniaturized highly integrated MOSFET devices are:

1) Electron migration

2) Hot electrons and breakdown

3) Wiring complexity

4) Ohmic resistance

5) Inter connections

6) Power dissipation

It is obvious that VLSI has the potential for high density ROM memory circuits. The extensive use of microprogramming techniques in the digital-control, data processing and RNS digital filtering creates a need

for versatile Read-Only Memories (ROMs) with the following features.

1) long term non-volatility

2) on-chip address decoding

3) electrical reprogrammability

4) high yield and high speed of operation.

After this brief introduction, let us concentrate some on the computational modules developed using the look-up tables.

4.9 Computational Modules

To start with, a Standard Computational Element (SCE) for the VLSI realization of digital processors has been proposed in [73] to perform the binary operations of addition, subtraction and multiplication. This SCE has been developed based on the following facts. The look-up table approach, has a symmetry property in the storage. In the direct storage of the look-up table, the memory space is used inefficiently. It is, therefore, desirable to consider an alternate circuit structure for implementing the residue operations which will lead to high speed and more efficient use of memory space. To achieve this we can use a structure which consists of a binary 2's complement adder with two b bit inputs and a b + 1 bit output and a ROM with a b + 1 bit address space and b bit memory locations as shown in Fig. 4.4. This structure is called the SCE.

In addition to the binary operations, schemes to implement the first order recursive structure, and the mixed radix conversion have also been proposed using the SCE. A specific design approach for the

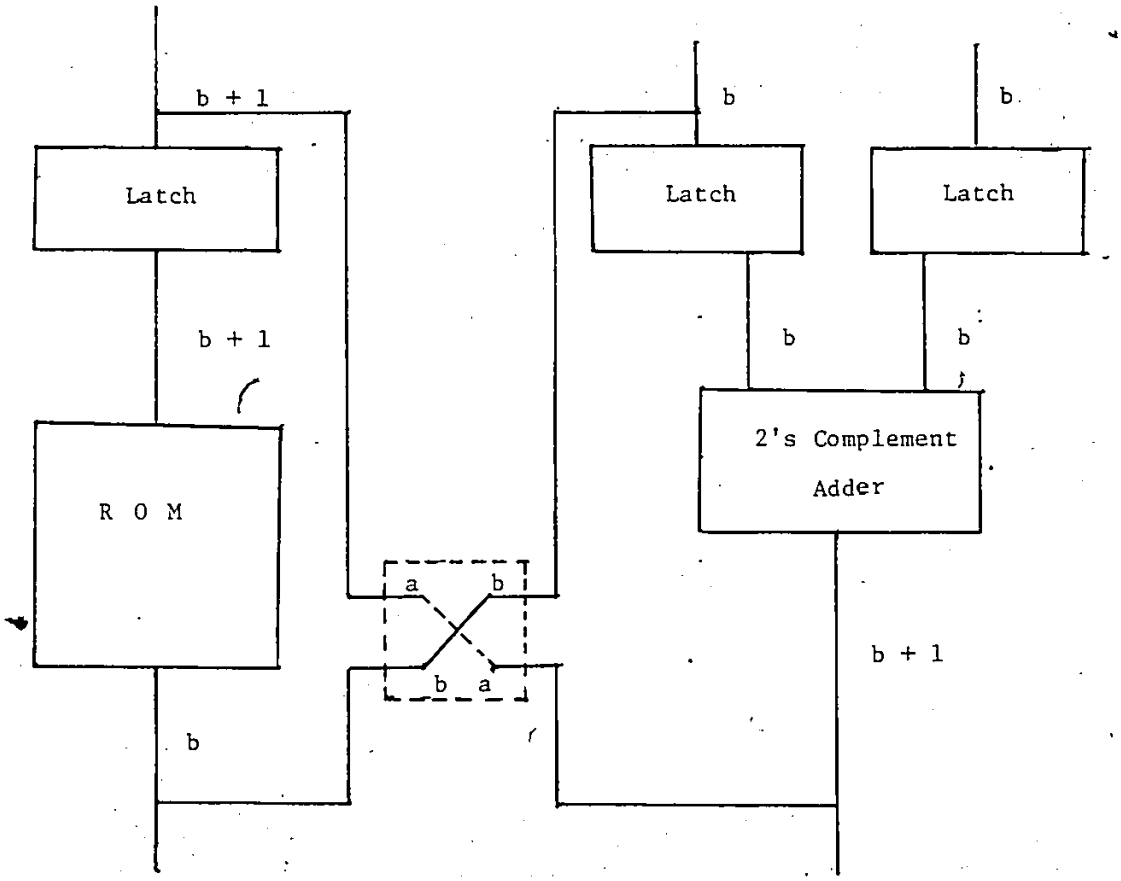


Fig. 4.4 Standard computational Element (SCE).

memory look-up tables has, however, not been presented in [73].

In [89], a set of computational modules has been presented as shown in Fig. 4.5; they are the J-module (Joint-module), F-module (Fork-module and JF-module (Joint, Fork-module) all developed by adapting the hierarchical memory structure proposed by Mead and Martin in [91]. An algorithm has been presented in [89], to compute the physical dimensions of the look-up tables and the channel length of the computational modules. These look-up table modules can be used to implement most of the signal processing algorithms in RNS based digital filters. The development of the QRNS and MQRNS structures using these modules is discussed in the following sections.

4.10 Implementation of Complex Multiplication Using VLSI Modules in the QRNS and MQRNS

In this section, we develop circuit structures to perform complex multiplication in the QRNS and MQRNS adapting the J-modules, F-modules and JF-modules.

I. QRNS

To implement complex multiplication in the QRNS let us consider two complex samples $a + \hat{i}b$ and $c + \hat{i}d$ where $\hat{i} = \sqrt{-1}$.

In the QRNS, the operator $j^2 \equiv -1 \bmod m$, where m is of the form $4 + 1$.

$$A = |a + jb|_m \qquad A^* = |a - jb|_m$$
$$B = |c + jd|_m \qquad B^* = |c - jd|_m \tag{4.23}$$

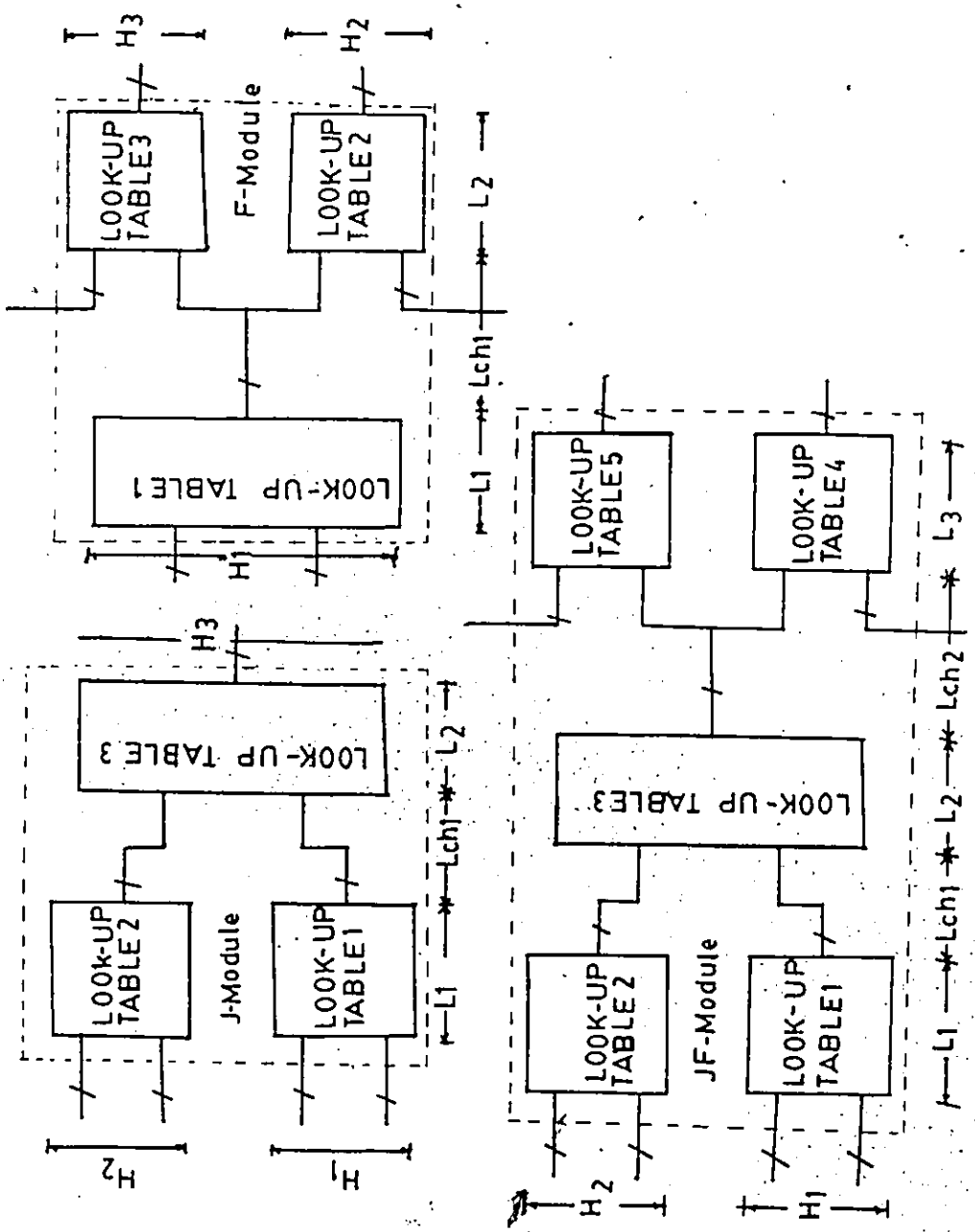


Fig. 4:5 VLSI computational modules using look-up tables.

The multiplication of the element pairs can be computed as:

$$Q = |A \cdot B|_m \qquad Q^* = |A^* \cdot B^*|_m \tag{4.24}$$

Then the real and imaginary parts of the complex multiplication can be computed as:

$$Y_R = |2^{-1} (Q + Q^*)|_m$$
$$Y_I = |2^{-1} \cdot j^{-1} (Q - Q^*)|_m \tag{4.25}$$

In implementing the complex multiplication in the QRNS, we require one J-module and one JF-module. The computation of the element pairs A, A^*, B and B^* (4.23) can be implemented using the first two look-up tables from the J-module and JF-module, respectively, as shown in Fig. 4.6. Following this, the multiplication of element pairs can be implemented using one look-up table from the same J and JF-modules respectively. The computation of real and imaginary parts can be implemented using the remaining 2 look-up tables from the JF-module as shown in Fig. 4.6.

II. MQRNS

In the MQRNS, the operator $\hat{j}^2 \equiv n \mod m$ where m is of any form except of the form $4K + 1$.

Let $$D = |\hat{j}^2 + 1|_m \quad \text{and} \quad S = |D \cdot b \cdot d \cdot |_m \tag{4.26}$$

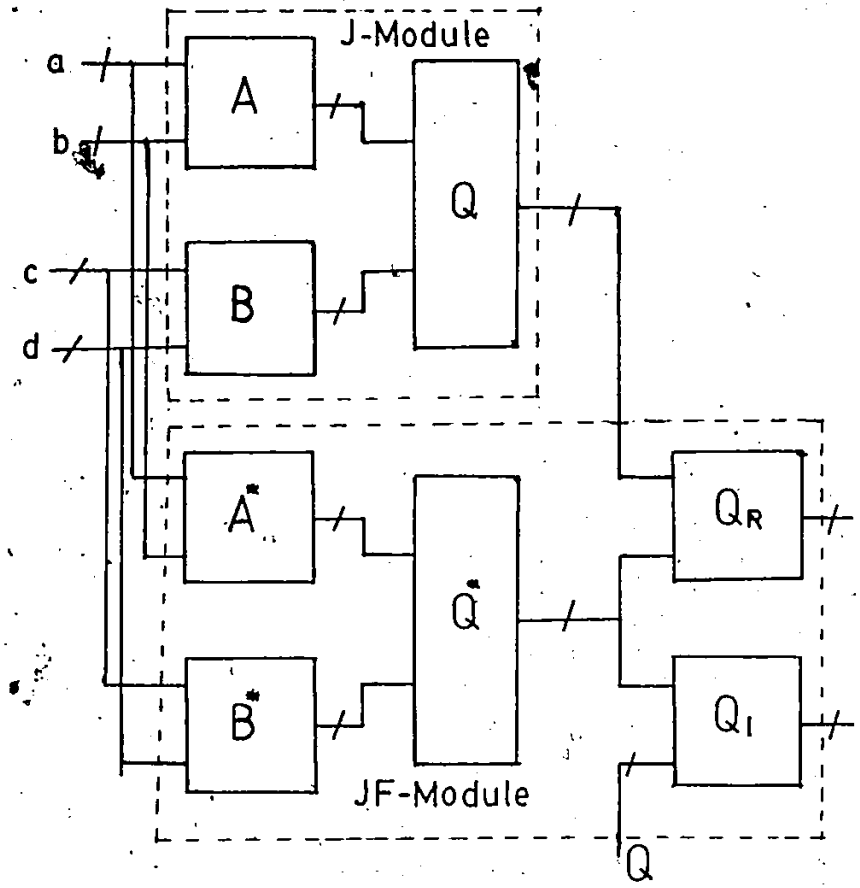


Fig. 4.6 Implementation of the QRNS using VLSI memory modules.

The element pairs of the two complex samples in the MQRNS can be computed as:

$$A = |a + \hat{j}b|_m \qquad A^* = |a - \hat{j}b|_m$$
$$B = |c + \hat{j}d|_m \qquad B^* = |c - \hat{j}d|_m \tag{4.27}$$

The multiplication of the element pairs can be computed as:

$$Q = |A \cdot B - S|_m \qquad Q^* = |A^* \cdot B^* - S|_m \tag{4.28}$$

Then the real and imaginary parts of the complex multiplication can be computed as:

$$Y_R = 2^{-1} (Q + Q^*)_m$$
$$Y_I = |2^{-1} \hat{j}^{-1} (Q - Q^*)|_m \tag{4.29}$$

In implementing the complex multiplication in the MQRNS, we require two J-modules and an F-module. In addition to this, we require one module consisting of 2 look-up tables as shown in Fig. 4.7. The computation of the element pairs A, A^*, B and B^* can be implemented using the first two look-up tables, from the 2 J-modules, respectively. The term S in (4.26) can be computed using the first look-up table in the F-module. The computation of Q and Q^* can be implemented using the remaining look-up tables from the 2 J-modules and an F-module respectively as explained in equation (4.28). In order to compute the real and imaginary parts of the complex multiplication, we require two more look-up tables as shown in Fig. 4.7.

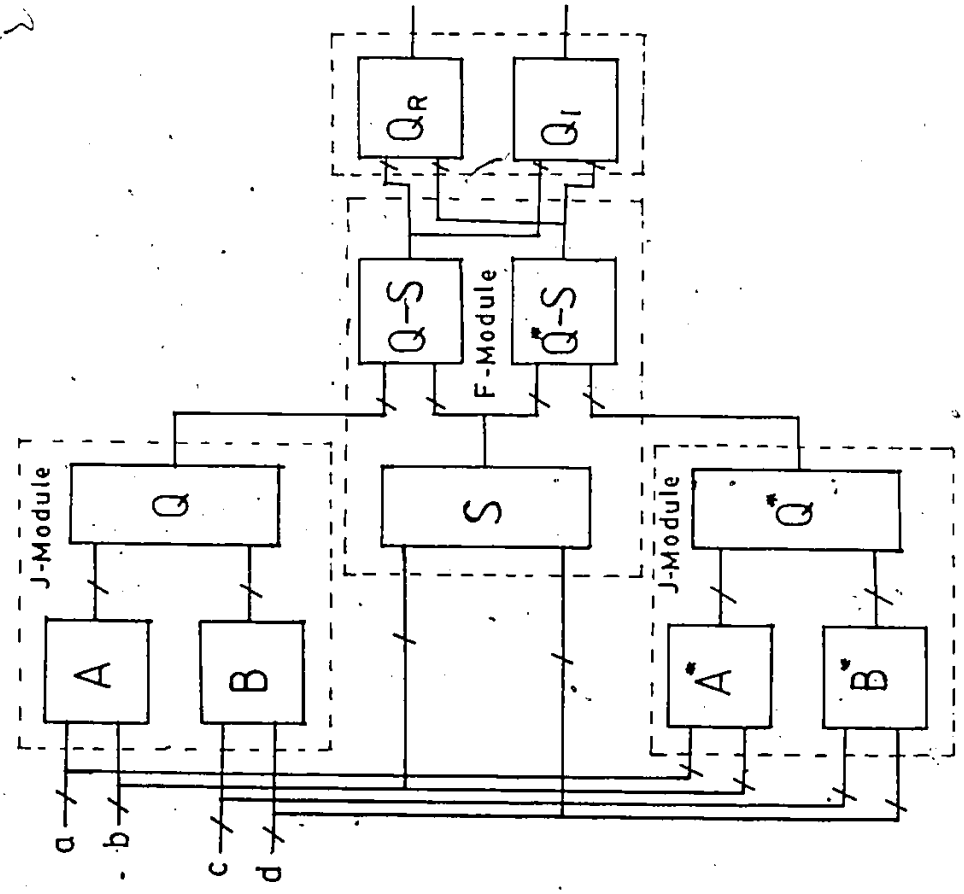


Fig. 4.7 Implementation of the MQRNS using VLSI memory modules.

Thus the complex multiplication is implemented using the VLSI modules in the QRNS and MQRNS. These modules can also be used to implement the Fast Number Theoretic Transform (FNTT) butterfly structures and many other structures using the QRNS and MQRNS.

4.11 Summary

This chapter has covered the implementation aspect of the Quadratic Residue Number Systems. Hardware implementations are described using Read-only Memory (ROM) arrays. Techniques for binary to residue, residue to the QRNS or MQRNS and residue to binary conversion have been discussed. A scheme for direct coding of element pairs into a binary representation using CRT and mixed-radix conversion has been presented. A scheme for the computation of element pairs using a single multiplexed ROM in the MQRNS and QRNS has been presented. It has been observed that the scheme in the MQRNS is faster than the scheme for the QRNS in obtaining the element pairs using a single multiplexed ROM. Finally, the implementation of the QRNS and MQRNS has been presented using VLSI modules proposed for RNS implementation of digital signal processing operations.

Chapter V

IMPLEMENTATION OF NON-RECURSIVE AND RECURSIVE COMPLEX DIGITAL FILTERS USING THE QRNS AND MQRNS

5.1 Introduction

Digital filters have become an increasingly attractive replacement for analog filters due to recent advances in semi-conductor technology. A linear time invariant discrete-time system is commonly called a digital filter. There are two types of digital filters commonly encountered in digital signal processing:

1. Finite Impulse Response (FIR)

2. Infinite Impulse Response (IIR) or Recursive Digital filters.

The FIR digital filter produces an output based on a weighted sum of present and past inputs. This type of filter is inherently stable, with linear phase as an easy requirement. However, the large number of additions and multiplications required for the direct implementation of FIR filters limits the speed and efficiency. The FIR filter can be described by the input-output relationship:

$$y(n) = \sum_{k=0}^{N-1} h(k)\, x(n-k) \qquad (5.1)$$

where $\{x(n-k)\}$ are the input samples $\{y(n)\}$ the output samples and $h(k)$ are the filter coefficients and N is the order of the filter.

A recursive digital filter is characterized by a difference equation of the form:

$$y(n) = \sum_{k=0}^{N-1} a(k) \cdot x(n-k) - \sum_{k=1}^{N-1} b(k) \cdot y(n-k) \qquad (5.2)$$

where $\{x(n-k)\}$ is the input sequence, $\{y(n)\}$ the output sequence and $\{a(k)\}$, $\{b(k)\}$ are the filter coefficients; N is the order of the filter. Since recursive filters incorporate feedback to modify the output with weighted samples of previous outputs, they can generate an infinite impulse response with the requirement of only a finite number of computations per output sample. Therefore, recursive filters are usually more economical than non-recursive filters in terms of computational time and memory in producing similar magnitude-frequency response characters. However, these benefits are acquired at a cost of sensitivity to roundoff error accumulation due to feedback [68,69].

The digital filters in (5.1,5.2) form one of the most important classes of digital signal processing systems and have found many applications in a number of diverse fields of science and engineering. In many of the signal processing applications, the input sequences and the filter coefficients will be represented in complex form, and those filters can be called complex digital filters. In the following sections, we will concentrate on the implementation of complex FIR and recursive digital filters using the Quadratic and Modified Quadratic Residue Number System (QRNS, MQRNS).

5.2 Implementation of Non-Recursive Filters Using the QRNS and MQRNS

In processing complex data with the QRNS and MQRNS, we are dealing with integers. When working with digital signals, we can assume without any loss of generality that the data from the A/D converters are treated

as integers. In implementing non-recursive digital filters using the QRNS and MQRNS, we have considered the following digital filter structures:

1. Direct FIR filter architectures
2. Bit-slice architectures.

5.2.1 Direct FIR Filter Architectures

A non-recursive digital filter is characterized by an input output relationship as shown in (5.1); when N is very large, transform techniques such as the Fast Fourier Transform (FFT) [51] or the Fast Number Theoretic Transform (FNIT) [23] usually provide the most efficient implementation of FIR filters. Among the various tradeoffs in the implementation of the FFT processors or the FNTT processors, Jenkins and Leon have proposed a method of direct implementation of the FIR filter when N is small ($N < 100$) using RNS techniques.

In their implementation, they deal with real integer sequences. In [20], the implementation of the direct FIR filter architecture has been proposed using the QRNS for complex sequences. This direct FIR filter architecture utilizes the Chinese Remainder Theorem (CRT) for residue to binary interfacing after converting the element pairs into real and imaginary parts in the respective channels of the RNS structure. But in this work, we have presented methods to decode directly the element pairs into binary representations using the CRT and mixed radix conversion (Chapter IV). Using this new decoding scheme, the implementation of the direct FIR filter architectures in the QRNS and MQRNS are discussed in the following sections.

The FIR filter in (5.1) can be represented in the z-domain, with z representing a unit delay as shown below.

$$y(z) = \sum_{k=1}^{N} h(k) \cdot x(z)\ z^{-k} \tag{5.3}$$

This non-recursive digital filter can be represented via the structure shown in Fig. 5.1.

The direct implementation of a FIR filter can be considered for 3-cases: case 1, MQRNS; case 2, QRNS; and case 3, a combination of the QRNS and MQRNS in the following sections.

Let $h(k) = a(k) + \hat{i}b(k)$ be the impulse response and

$x(n - k) = c(n - k) + \hat{i}d(n - k)$ be the complex sequence of the FIR filter in (5.1), where $\hat{i} = \sqrt{-1}$

Case-1

For the MQRNS the operator $\hat{j}$ can be computed using

$$\hat{j}_t^2 \equiv n_t \bmod m_t,\ \hat{j}_t \epsilon\ R(m_t) \tag{5.4}$$

where $R(m_t)$ is a finite integer ring, and m_t is of any form except of the form $4K + 1$.

In order to find a general solution of (5.4), the result of the theorem -2 (Chapter IV) can be used. Having established the value of the operator $\hat{j}$,

let, $$D_t = |\hat{j}^2 + 1|_{m_t} \tag{5.5}$$

and the element pairs of the input samples and the filter coefficients can be computed as follows;

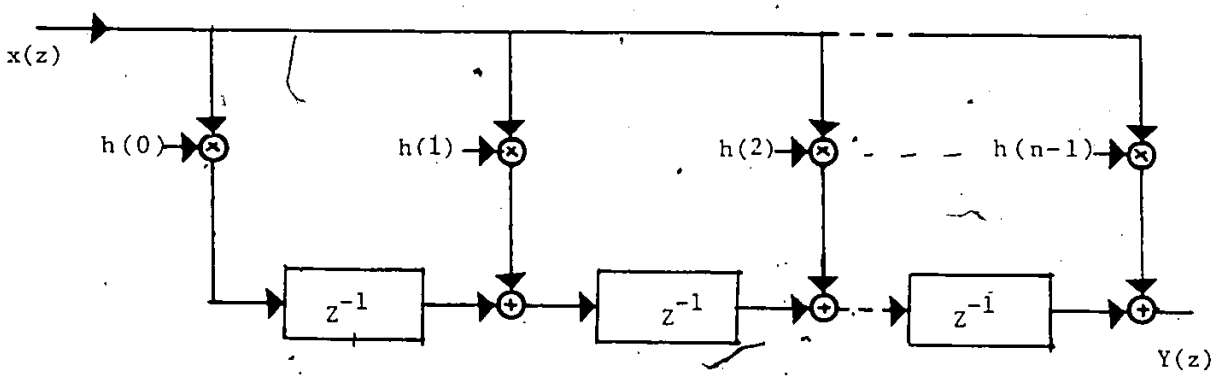


Fig. 5.1 FIR filter structure.

$$h(k)_t = |a(k)_t + \hat{j}\, b(k)_t|_{m_t} \qquad h^*(k)_t = |a(k)_t - \hat{j}\, b(k)_t|_{m_t} \tag{5.6}$$

$$x(n-k)_t = |c(n-k)_t + \hat{j}\, d(n-k)_t|_{m_t} \qquad x^*(n-k)_t = |c(n-k)_t - \hat{j}\, d(n-k)_t|_{m_t} \tag{5.7}$$

where $a(k)_t$, $b(k)_t$, $c(n-k)_t$, $d(n-k)_t \in R(m_t)$.

The FIR filter output in (5.1) can be written in terms of the residue pairs as

$$Q(n)_t = \left| \sum_{k=0}^{N-1} |\, h(k)_t \cdot x(n-k)_t \,|_{m_t} - s(k)_t \right|_{m_t} \tag{5.8}$$

$$Q^*(n)_t = \left| \sum_{k=0}^{N-1} |\, h^*(k)_t \cdot x^*(n-k)_t \,|_{m_t} - s(k)_t \right|_{m_t} \tag{5.9}$$

where $s(k)_t = |D \cdot b(k)_t \cdot d(n-k)_t|_{m_t}$

The real and imaginary parts of the output can be given as

$$Y_R(n)_t = |2^{-1} (Q(n)_t + Q^*(n)_t|_{m_t}$$

$$Y_I(n)_t = |2^{-1} \hat{j}^{-1} (Q(n)_t - Q^*(n)_t|_{m_t} \tag{5.10}$$

Case-2

The operator j for the QRNS can be computed using

$$j_t^2 \equiv -1 \bmod m_t \text{ and } t \in \{1,2, \ldots L\} \tag{5.11}$$

and m is of the form 4K + 1.

The element pairs of the input samples and the filter coefficients

can be computed as follows:

$$h(k)_t = |a(k)_t + j\ b(k)|_{m_t} \qquad h^*(k)_t = |a(k)_t - j\ b(k)|_{m_t} \tag{5.12}$$

$$x(n-k)_t = |c(n-k) + j\ d(n-k)|_{m_t} \qquad x^*(n-k)_t = |c(n-k)_t - j\ d(n-k)|_{m_t} \tag{5.13}$$

where $a^*k)_t$, $b(k)_t$, $c(n-k)_t$, $d(n-k)_t \in R(m_t)$.

The equation (5.1) can be decomposed into

$$Q(n)_t = \left| \sum_{k=0}^{N-1} h(k)_t \quad x(n-k)_t \right|_{m_t}$$

$$Q^*(n)_t = \left| \sum_{k=0}^{N-1} h^*(k)_t \cdot x^*(n-k)_t \right|_{m_t} \tag{5.14}$$

where $Q(n)_t$ and $Q^*(n)_t$ are the residue codes and the conjugate pairs of the output of the FIR filter.

Let $Y_R(n)$ and $Y_I(n)$ be the real and imaginary parts of the filter output. Hence

$$Y_R(n)_t = |2^{-1} (Q(n)_t + Q^*(n)_t)|_{m_t}$$

$$Y_I(n)_t = |2^{-1} j^{-1} (Q(n)_t - Q^*(n)_t)|_{m_t} \tag{5.15}$$

Example

In order to demonstrate the direct implementation of the FIR filter, let us consider the second order FIR filter equation

$$y(n) = h(0)\ x\ (n) + h(1)\ x\ (n-1) + h(2)\ x\ (n-2) \tag{5.16}$$

where $h(0) = 15 + j8$, $h(1) = 6 + j20$ $h(2) = 5 + j27$ and

$x(n) = 9 + j12$, $x(n-1) = 15 + j2$, $x(n-2) = 8 + j4$

For this example, we will choose the prime moduli in the case of the MQRNS to be 7, 11, 19, 31 and in the case of the QRNS 5, 13, 17, 29. The moduli set {7, 11, 19, 31} provides a dynamic range of about 15.4 bits and the moduli set {5, 13, 17, 29} provides a dynamic range of about 14.9 bits which are reasonably adequate for a large number of applications.

We can also combine the two types of prime moduli, and for example, a moduli set of 5, 7, 11, 13 can be selected which will give a dynamic range of 12.2 bits. Although we have lost some dynamic range, we have gained the ability to implement the procedure with fast 4 bit wide ROMs of size 64 x 4 and 256 x 4. We will refer to this procedure as case-3 where computations in case-1 and case-2 are repeated here and the operator $\tilde{j}$ (the imaginary part for the ring isomorphic to the direct sums) can be precomputed using the CRT.

In implementing the FIR filter equation, we require three complex multiplications. We illustrate the multiplication using the above four moduli in the three cases as follows (Table 52, 5b, 5c).

Table 5a and 5b shows the residue code and conjugate pairs for the sequence and the filter coefficients in the MQRNS and QRNS respectively. Table 5c summarizes the results after implementing the CRT techniques proposed in Chapter IV. The direct implementation of the FIR filter equation using conventional complex multiplication gives the same real and imaginary part of the filter output as obtained in the above three cases.

In implementing the CRT technique, we need a modulo M-adder-shifter [2]. Modulo M adders ($M \neq 2^B$) are not available commercially and the

Table 5a

Residue Codes for the FIR Filter in MQRNS

	M1=7 $\hat{j}$=1	M2=11 $\hat{j}$=1	M3=19 $\hat{j}$=1	M4=31 $\hat{j}$=1
Q	4	5	6	12
Q^*	4	7	15	4
Q	5	10	1	21
Q^*	4	2	4	17
Q	0	3	16	13
Q^*	4	4	0	6
Q	2	7	4	15
Q^*	5	2	0	27

Table 5b

Residue Codes for the FIR Filter in QRNS

	M1=5 j=2	M2=13 j=5	M3=17 j=4	M4=29 j=12
Q	3	12	10	18
Q^*	0	1	0	2
Q	4	11	6	24
Q^*	1	11	9	18
Q	4	7	9	9
Q^*	0	0	8	0
Q	1	4	8	22
Q^*	1	12	0	20

Table 5c.

The Results After Implementing CRT for the Residue Codes in Table 5a and 5b

	M	Q	Q^*	$j/\hat{j}/\tilde{j}$	$j^{-1}/\hat{j}^{-1}/\tilde{j}^{-1}$	Real	Imaginary
Case-1	45353	821	44574	1	1	21	800
Case-2	32045	22091	9996	17412	14633	21	800
Case-3	5005	4671	376	463	3697	21	800

implementation by other means can be expensive. The problem associated with the modulo M adder can be handled if we employ the mixed radix conversion technique proposed in (Chapter IV).

Even though the problem associated with using modulo M adders in the residue to binary conversion is solved using the mixed radix conversion technique, we still have to compute the binary version of the real and imaginary parts (5.10,5.15), which will require another modulo M adder. Therefore for high speed implementation the mixed radix conversion cannot solve the problem fully. An alternative approach is by using a conversion technique based on the CRT, as proposed in [4]. This fractional method is fast and requires the least hardware of all methods so far considered.

In the case of the MQRNS this fractional method is perfectly suitable, if we select the operator $\hat{j} = 1$. Now the resulting Q and Q*, computed using the new method, are in fractional form and the computation of the real and imaginary parts can be implemented using a binary adder/subtractor. The resulting output is right shifted to perform the multiplication by 2^{-1} as shown in (5.10). As mentioned in [4], to obtain the correct integer result, the multiplication by M is accomplished in the final conversion.

In the case of the QRNS this fractional method is not suitable because, while computing the imaginary part, the operation shown in (5.15) drastically alters the imaginary part. Hence, in the case of the QRNS, the only viable approach is first to map the residue codes into complex integers and then to use the conversion technique proposed in [4]. For mapping the residue codes into complex integers, using the QRNS, we

require 2 ROMs for each modulus. However, in the case of the MQRNS, these 2 ROMs will be utilized in performing the operations described in (5.9). Hence both the QRNS and MQRNS require almost the same amount of hardware for the implementation and give almost the same data rate. This can be observed from Table 5d which gives the number of hardware elements required and the data rate per stage (for each modulus) in the direct implementation of a FIR filter using the QRNS and MQRNS. Here the data rate is limited by the time required starting from the computation of Q and Q^* to the decoded output of the real and imaginary parts $Y_R(n)$, $Y_I(n)$. Fig. 5.2 and Fig. 5.3 present the FIR filter architecture for the QRNS and MQRNS systems respectively. Fig. 5.4 presents the direct FIR filter implementation using the combination of both the QRNS and MQRNS.

The direct FIR filter structure is simulated using software in the QRNS and MQRNS. The program is presented in Appendix B.

Table 5e gives the number of operations required in converting from RNS to binary in the QRNS, QLRNS and MQRNS.

The decoding method proposed in Chapter IV will bring a considerable reduction in the number of memory chips needed. But the problem associated with the modulo M adder, in computing the real and imaginary parts, will be expensive.

Even though the NTT provides the most efficient implementation of the FIR filter, the direct implementation of FIR filter has some important applications. The computation of very high speed complex arithmetic is important in spectral analysis and in the processing of complex baseband waveforms that result from quadrature demodulation in Radar and communication

Table 5d

T=Memory access time
t=Binary addition time
N=Sequence length

Hardware Requirement For a FIR Filter Implementation Per Stage in QRNS and MQRNS

	No. of ROMs	No. of Accumulator	No. of Binary Adder/Subtractor	Data Rate in Hz
QRNS	16	2	2	1/(4NT+t)
MQRNS	17	2	4	1/(4NT+2t)

Table 5e

Number of Operation Requires for the Conversion From RNS to Binary in the QRNS, QLRNS and MQRNS.

Number System	Number of Look-up Table Operations	Number of Binary Addition Operation
QRNS & QLRNS	2	1
MQRNS	1	2

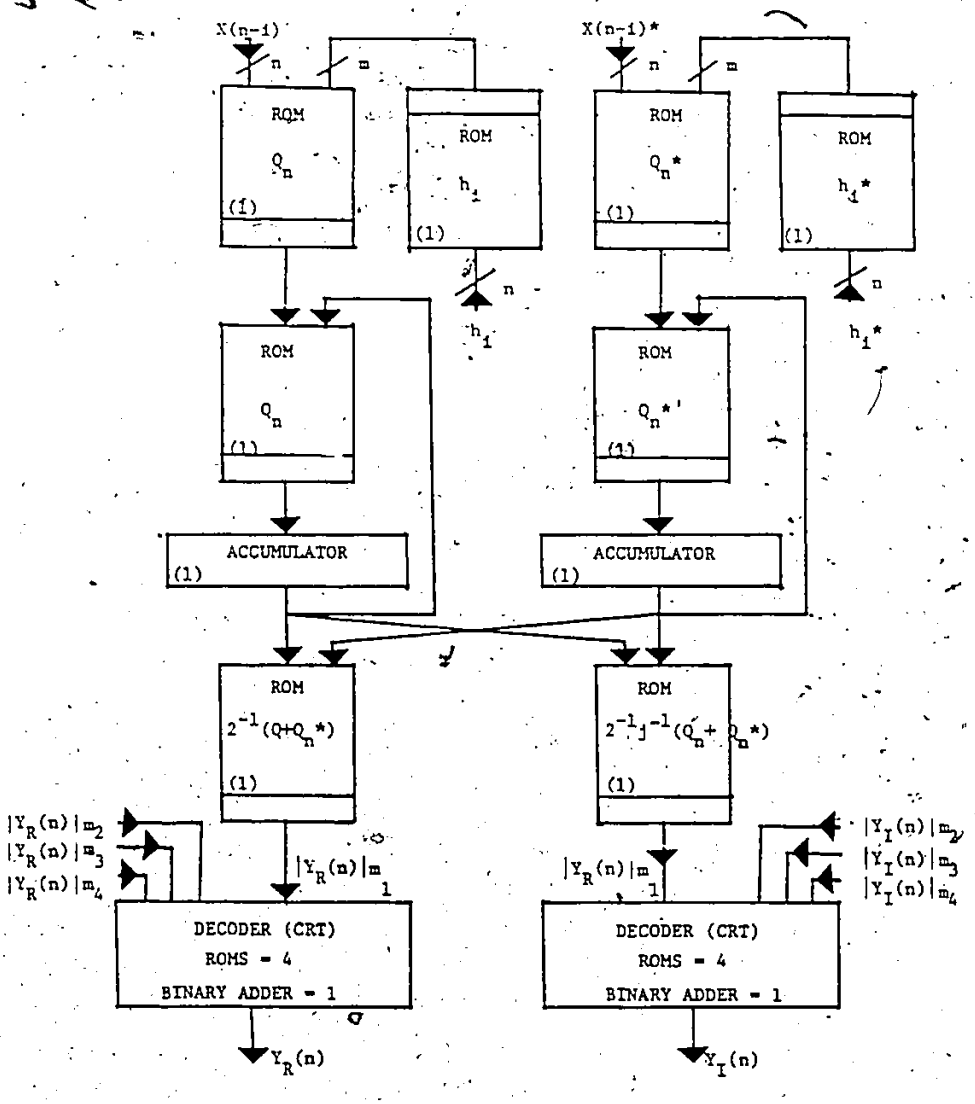


Fig. 5.2 Complex FIR filter architecture in the QRNS.

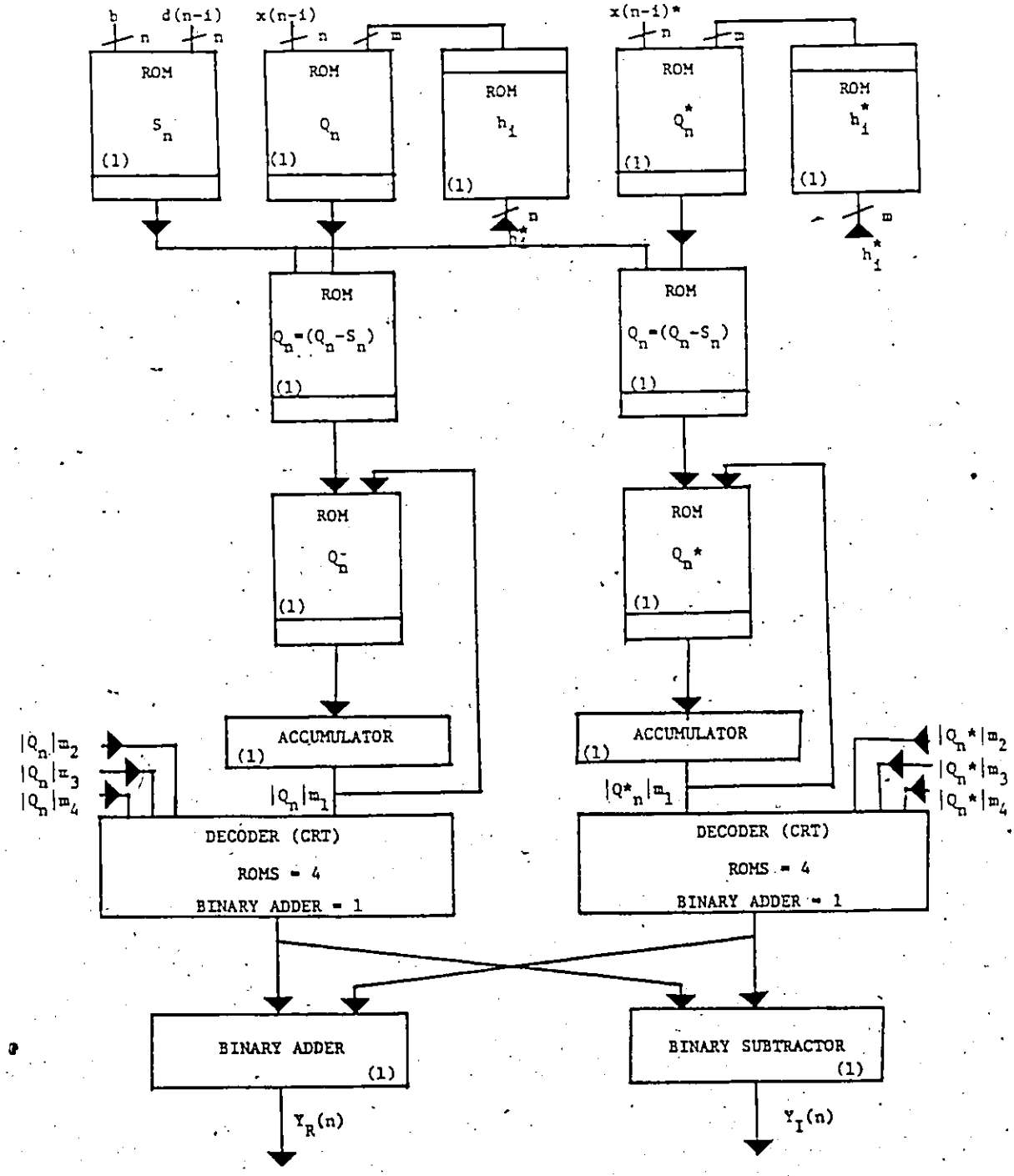


Fig. 5.3 Complex FIR filter architecture in the MQRNS.

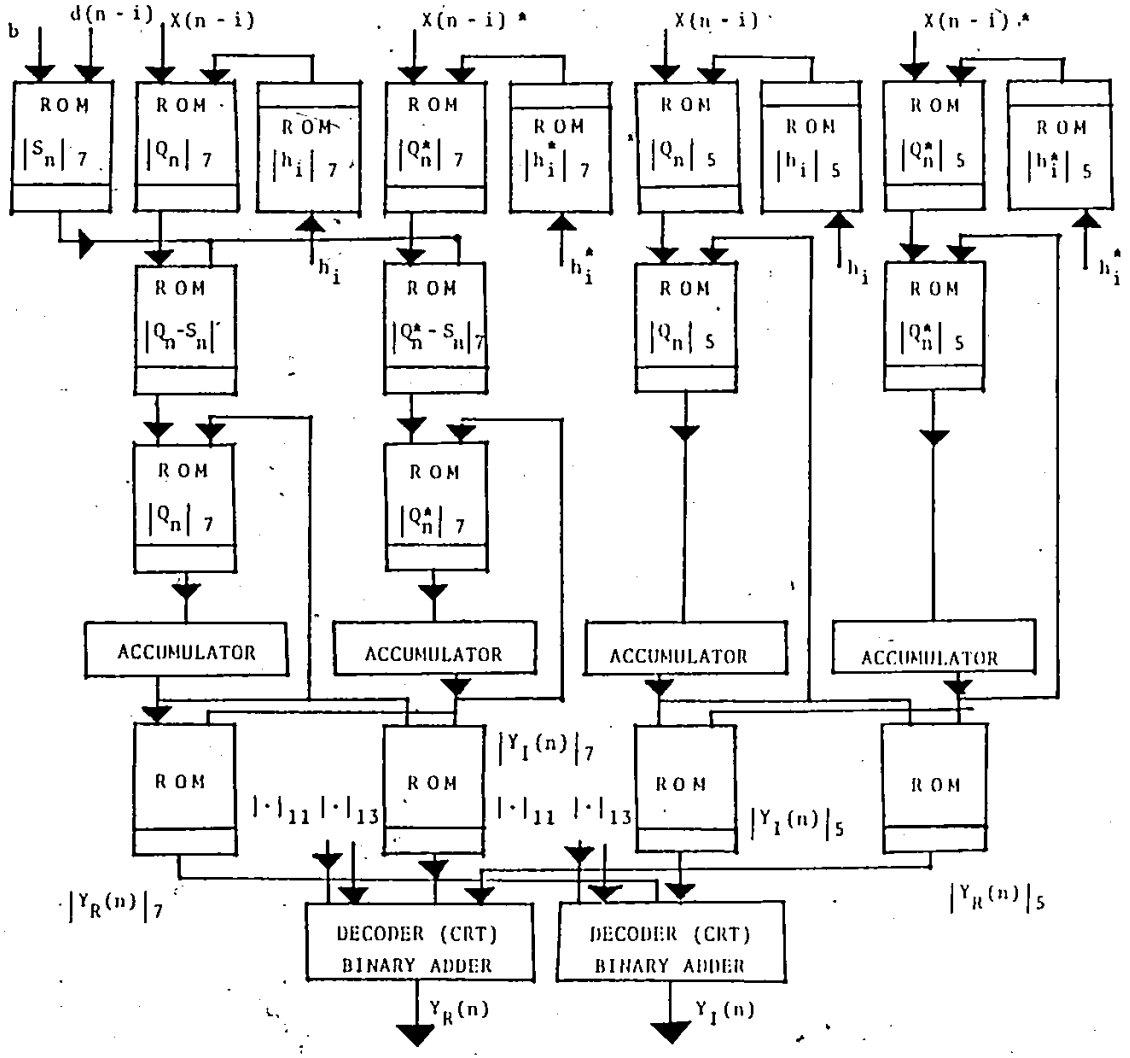


Fig. 5.4 Complex FIR filter architecture in the ORNS and MQRNS.

systems. For example, modern synthetic aperture Radars operating in the spotlight mode required a polar-rectangular format interpolation filter that can operate in real time when aircraft flies by the scene. These interpolation filters are, ideally, spacially varying complex FIR filters of relatively low order ($5 < N < 15$) [59]. So for this type of short sequence FIR filter, the direct implementation is more suitable than using the transform approach.

In [12, Peled and Liu have proposed a modular approach for the hardware implementation of the multiplication operation for fixed multipliers in digital filters. This approach is called the bit-slice technique, and is presented in the following section.

5.2.2 Bit-slice Technique

In general, most signal processing algorithms perform repeated multiplications of the form

$$Y(n) = \sum_{k=1}^{N} a_k \cdot x_k \qquad (5.17)$$

where $\{a_k\}$ are the predetermined filter coefficients and $\{x_k\}$ are the input samples.

If $\{x_k\}$ are encoded in the hardware as binary integers then

$$x_k = \sum_{\ell=0}^{B-1} 2^{\ell}\, x_k^{\ell} \qquad (5.18)$$

where B is the number of binary bits and

$$x_k^{\ell} = 0 \text{ or } 1$$

Equation (5.17) can be written as

$$y(n) = \sum_{k=1}^{N} a_k \sum_{\ell=0}^{B-1} 2^{\ell} x_k^{\ell} \tag{5.19}$$

and this can be re-arranged as

$$y(n) = \sum_{\ell=0}^{B-1} 2^{\ell} F(x_1^{\ell}, x_2^{\ell} \ldots . x_N^{\ell}) \tag{5.20}$$

where $F(x_1^{\ell}, x_2^{\ell}, \ldots . x_N^{\ell}) = \sum_{k=1}^{N} x_k^{\ell} \cdot a_k$

Equation (5.20) is the stored function of a_k. This function can be pre-computed and stored in a PROM. The function $F(x_1^{\ell}, x_2^{\ell}, \ldots . x_N^{\ell})$ is the precomputed partial sum of the product. $y(n)$ can be computed by addressing the stored function $F(x_1^{\ell}, x_2^{\ell}, \ldots . x_N^{\ell})$, followed by shifting and adding operations. Digital filter architectures which are built using this bit-slice technique are called bit-slice architectures or, sometimes combinatorial digital filter architectures [13].

In the following section, the bit-slice technique is used to implement complex FIR filter structures using the QRNS and MQRNS.

5.2.3 Implementation of FIR Filters Using Bit-slice Techniques

The element pairs of the output of the FIR filter is expressed in equation (5.14) for the QRNS.

The residue codes of the input sequence can be represented by B binary bits as follows:

$$x(n-k)_t = \sum_{i=0}^{B-1} 2^i \, x(n-k)^{(i)}$$

$$x^*(n-k)_t = \sum_{i=0}^{B-1} 2^i \, x^*(n-k)^{(i)} \tag{5.21}$$

where $x(n-k)^{(i)} = 0$ or 1

and

$$x^*(n-k)^{(i)} = 0 \text{ or } 1$$

So equation (5.14) can be written as

$$Q(n)_t = \left| \sum_{k=0}^{N-1} h(k)_t \sum_{i=0}^{B-1} 2^i \, x(n-k)^{(i)} \right|_{m_t}$$

$$Q^*(n)_t = \left| \sum_{k=0}^{N-1} h^*(k)_t \sum_{i=0}^{B-1} 2^i \, x^*(n-k)^{(i)} \right|_{m_t} \qquad (5.22)$$

(5.22) can be written with the order of summation interchanged

$$Q(n)_t = \left| \sum_{i=0}^{B-1} 2^i \sum_{k=0}^{N-1} x(n-k)^{(i)} \cdot h(k)_t \right|_{m_t}$$

$$Q^*(n)_t = \left| \sum_{i=0}^{B-1} 2^i \sum_{k=0}^{N-1} x^*(n-k)^{(i)} \cdot h(k)_t \right|_{m_t} \qquad (5.23)$$

Let $Y_R(n)_t$ and $Y_I(n)_t$ be the real and imaginary parts of the filter output. Hence

$$Y_R(n)_t = \left| 2^{-1} \, (Q(n)_t + Q^*(n)_t) \right|_{m_t}$$

$$Y_I(n)_t = \left| 2^{-1} \, j^{-1} \cdot (Q(n)_t - Q^*(n)_t) \right|_{m_t} \qquad (5.24)$$

Equations (5.24) can be implemented using the bit-slice technique [12] as shown in Fig. 5.5 for a sequence length of 64. From this figure we can observe that we require only two channels of hardware for the implementation of the direct FIR filter structure. But to implement the same filter using conventional methods, we need four channels to compute

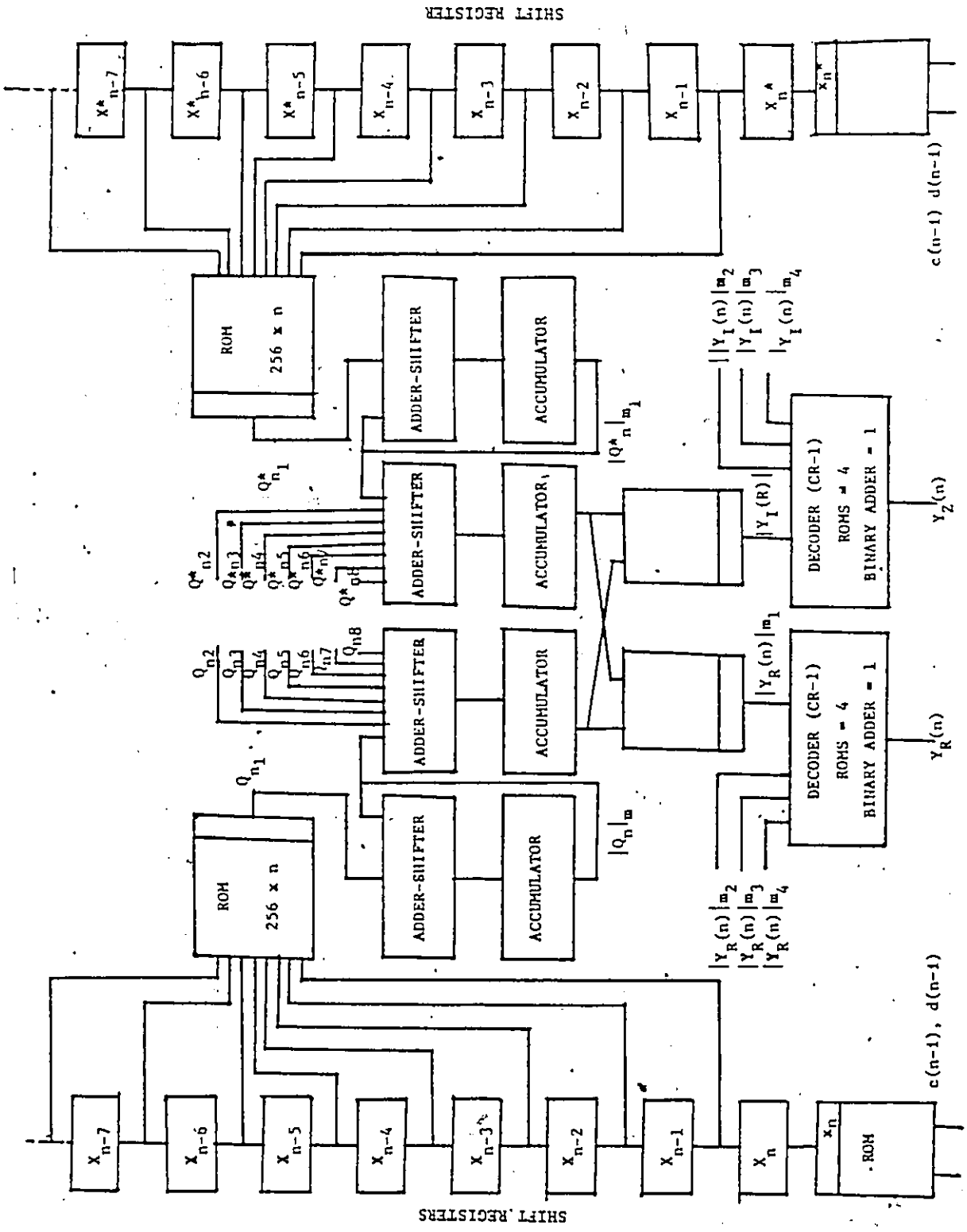


Fig. 5.5 FIR filter bit-slice architecture for complex sequence using the QRNS for a sequence length of 64

the complex multiplication (four real multiplications). That is, there is a 50% hardware reduction if we use the QRNS for the direct FIR filter implementation.

In the case of the MQRNS, we need one more channel to perform the third multiplication. Thus it is observed that for implementing FIR filter using bit-slice architecture in the MQRNS requires about 75% of the hardware required by the conventional RNS structure.

5.3 Recursive Digital Filters

As shown in (5.2), a recursive digital filter is characterized by an input output relationship of the form:

$$y(n) = \sum_{k=0}^{N-1} a(k)\cdot x(n-k) - \sum_{k=0}^{N-1} b(k)\cdot y(n-k)$$

where $\{x(k)\}$ is the input sequence, $\{y(n)\}$ the output sequence and $a(k)$, $b(k)$ are the filter coefficients. For the hardware realization, it is convenient to represent the above equation using delay elements, written as

$$H(z) = \sum_{k=0}^{N-1} a_k z^{-k}/(1 + \sum_{k=1}^{N-1} b_k z^{-k}) \qquad (5.25)$$

where z^{-1} is the delay operator. Among the numerous configurations of realizing (5.25), the most advantageous is to factor the transfer function into second order sections and combine these in either a cascade or parallel form.

The cascade form corresponds to a factorization of numerator and denominator polynomials of (5.25) which can be represented as:

$$H(z) = \alpha_o \prod_{j=1}^{M} \left| \frac{\alpha_{2j} z^{-2} + \alpha_{ij} z^{-1} + 1}{\beta_{2j} z^{-2} + \beta_{1j} z^{-1} + 1} \right| \qquad (5.26)$$

where $M \geq N/2$

The parallel form results from the partial fraction expansion of (5.25), which can be represented as

$$H(z) = \gamma_o + \sum_{j=1}^{M} \left| \frac{(\gamma_{1j} z^{-1} + \gamma_{0j})}{(\beta_{2j} z^{-2} + \beta_{1j} z^{-1} + 1)} \right| \qquad (5.27)$$

where $M \geq N/2$

Fig. 5.6 gives the cascade form and Fig. 5.7 gives the parallel form realizations.

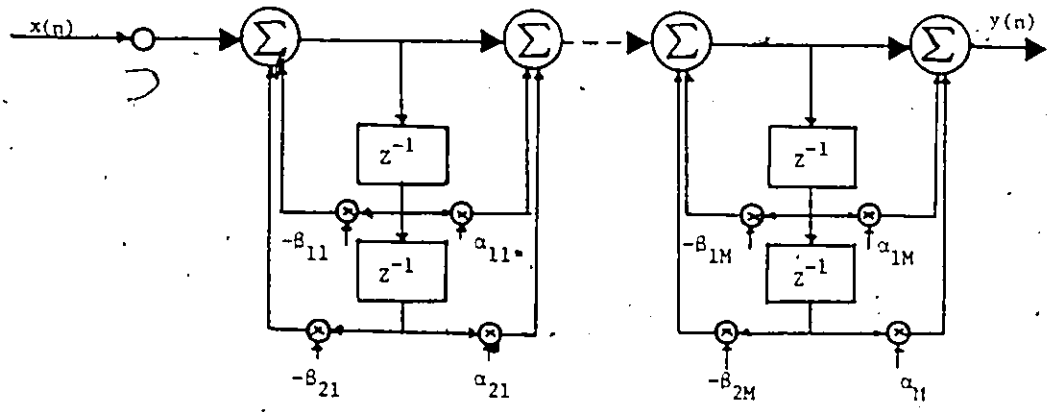
Any one-dimensional recursive filter can be realized using a second order cannonic section either in a cascade or parallel form. Let us focus on the implementation of the second order cannonic section as the building block. In the Z-transform

$$H(z) = \frac{a_{(0)} + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \qquad (5.28)$$

The input output relationship in the data domain can be written

$$Y(n) = a(0)x(n) + a(1)x(n-1) + a(2)x(n-2) - b(1)y(n-1) - b(2)Y(n-2) \qquad (5.29)$$

Fig. 5.6 Cascade form



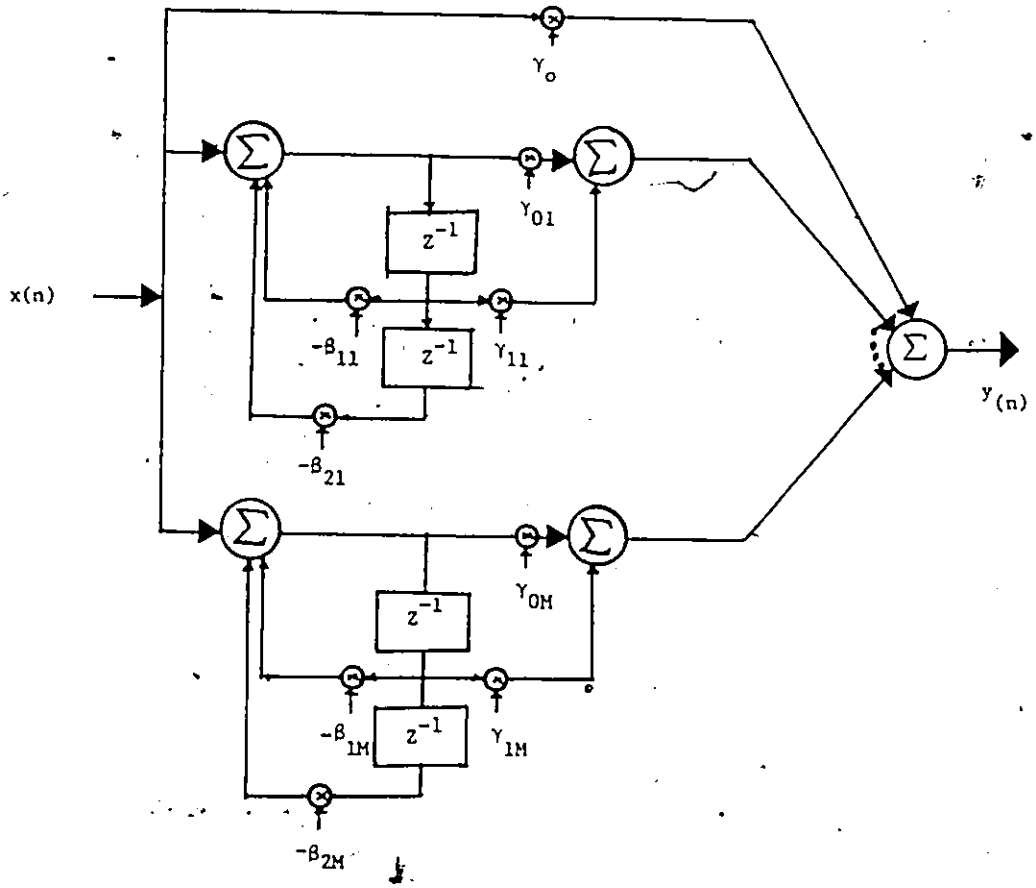


Fig. 5.7 Parallel form.

Equation (5.29) is usually realized using one of the following forms:

i) Direct form-1, shown in Fig. 5.8, implementing the difference equation in (5.29).

ii) Direct form-2, also called canonic form, (using less number of delays) shown in Fig. 5.9 implementing the pair of difference equations

$$Y(n) = a(0)\ W(n) + a(1)\ W(n-1) + a(2)\ W(n-2)$$
$$W(n) = x(n) - b(1)\ x(n-1) - b(2)\ x(n-2) \qquad (5.30)$$

where W(n) is an internal node in the filter.

The direct implementation of a second order section in direct and canonic form requires four adders and five multipliers. The number of delay units in the canonic form is less than the direct form as shown in Fig. 5.9.

If the digital filters in Fig. 5.8, 5.9 are implemented using binary arithmetic with finite wordlength representation, the output of the digital filter deviates from the desired characteristics. A great deal of study [48,49,58,68,] has been devoted to the effects of quantization error in recursive digital filters and error accumulation for different filter topologies using conventional binary arithmetic (fixed or floating point).

Recently, RNS techniques have been extended to the implementation of recursive digital filters. Many stable recursive filters have fractional filter coefficients. Since fractions are not allowed in the RNS, scaling,

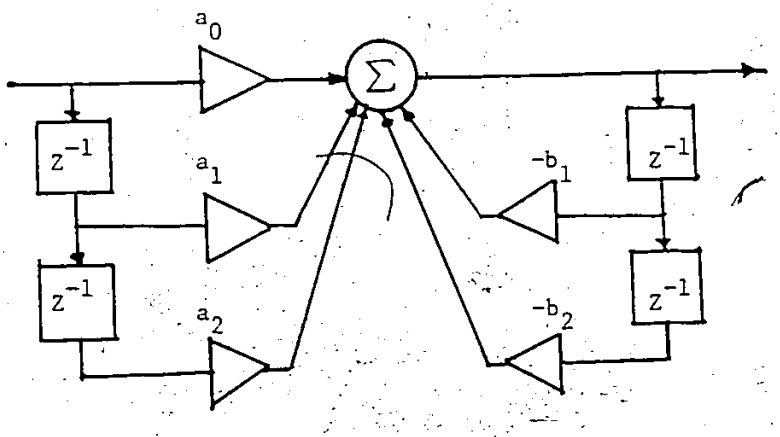


Fig. 5.8 Block diagram representation of direct form 1 second order section of a digital filter.

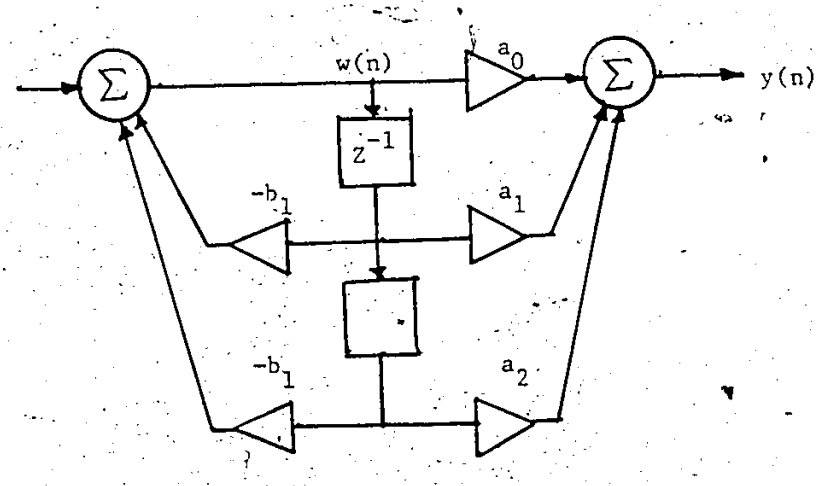


Fig. 5.9 Block diagram representation of canonical form second order section.

the multiplication by a fixed constant to represent the filter coefficients to integers is required. Investigations of RNS recursive digital filters have been centered upon research into efficient scaling algorithms. Jullien [1] has presented the implementation of recursive filters, with efficient scaling algorithms for the second order canonic section, which is one of the basic building blocks for any recursive filter. Jenkins [13] has proposed scaling techniques for the implementation of residue coded recursive digital filters and has presented a hardware architecture combining residue number concepts and bit-slice techniques. Soderstrand [33] has applied RNS techniques to the implementation of second order digital filters based on a Lossless Discrete Integration (LDI) ladder structure originally introduced by Bruton [74]. Here the scaling is avoided by storing complete look-up tables for multiplication by fractions. This type of LDI filter, using the RNS, has been shown to offer substantial reduction in the cost, and improvement in the speed over the structure developed by binary arithmetic. For the adaptive filtering scheme, the RNS based LDI filters can not be used. RNS-LDI architecture is not attractive for standard RNS integer multiplication because twice as many scaling arrays are required.

The RNS provides the potential for high speed efficient implementation if the structures used as a basis for RNS implementation have a large number of addition and multiplication operations and few scaling operations. Moreover, in the residue number system, addition, subtraction and multiplication are performed with extended precision, and errors occur only during the scaling process; such a structure will also exhibit low

sensitivity to quantization noise. The analysis of the quantization noise and the effects of limit cycles in implementing recursive digital filters using RNS techniques have been presented in [39].

In this work, we have concentrated on implementing recursive filters using the QRNS and MQRNS in the bit-slice architectures. Even though the bit-slice architecture is not suitable for the adaptive filtering or multiplexing schemes, where the filter coefficients are to be changed dynamically, this architecture is one of the most efficient realizations of a second order filter structure. This architecture provides the capability for obtaining high speed and high precision simultaneously. In the case of adaptive filtering, the architecture with general RNS multiplication using ROM arrays is proposed in [13].

In the following section we are interested in the implementation of recursive digital filters using the QRNS and MQRNS with the bit-slice architecture proposed by Peled and Liu [12].

5.4 Implementation of Recursive Digital Filters Using the QRNS and MQRNS

A recursive digital filter is characterized by the input-output relationship of Eqn. 5.2.

Since we are dealing with an integer system we need to choose an appropriate scaling factor to multiply the fractional filter coefficients in order to convert them to integers. Numerous methods for scaling are available [1] [13]. Let S be a scaling factor then the recursive filter in (5.2) can be modified as:

$$y(n)' = 1/S\left\{\sum_{k=0}^{N-1} S \cdot A(k) \cdot x(n-k) - \sum_{k=1}^{N-1} S \cdot B(k) \cdot y(n-k)'\right\} \qquad (5.31)$$

The expression within the brackets $\{ \cdot \}$ can be expressed as follows:

$$y(n) = \sum_{k=0}^{N-1} a(k) \cdot x(n-k) - \sum_{k=1}^{N-1} b(k) \cdot y(n-k) \qquad (5.32)$$

where $a(k) = S \cdot A(k)$ and $b(k) = S \cdot B(k)$

Let $x(n-k) = c(n-k) + \hat{i}\, d(n-k)$

and $a(k) = \alpha(k) + \hat{i}\, \beta(k)$,

$b(k) = \gamma(k) + \hat{i}\, \delta(k)$

The recursive filter (5.32) can be implemented using the residue codes as follows:

$$Q(n)_t = \left| \sum_{k=0}^{N-1} a(k) \cdot x(n-k) - \sum_{k=1}^{N-1} b(k) \cdot Q(n-k) \right|_{m_t}$$

$$Q^*(n)_t = \left| \sum_{k=0}^{N-1} a^*(k) \cdot x^*(n-k) - \sum_{k=1}^{N-1} b^*(k)\, Q^*(n-k) \right|_{m_t} \qquad (5.33)$$

where

$$x(n-k) = |c(n-k) + j_t\, d(n-k)|_{m_t} \qquad x(n-k)^* = |c(n-k) - j_t\, d(n-k)|_{m_t}$$

$$a(k) = |\alpha(k) + j_t\, \beta(k)|_{m_t} \qquad a(k)^* = |\alpha(k) - j_t\, \beta(k)|_{m_t}$$

$$b(k) = |\gamma(k) + j_t\, \delta(k)|_{m_t} \qquad b(k)^* = |\gamma(k) - j_t\, \delta(k)|_{m_t}$$

with $j_t \equiv -1 \bmod m_t$ and $t \in \{ 1,2,\ldots L \}$

Consider the implementation of a second order section, which can be specified as follows:

$$Q(n)_t = |a(0)\cdot x(n)+a(1)\cdot x(n-1)+a(2)\cdot x(n-2)-b(1)\cdot Q(n-1)-b(2)\cdot Q(n-2)|_{m_t}$$

$$Q^*(n)_t = |a^*(0)\cdot x^*(n)+a^*(1)\cdot x^*(n-1)+a^*(2)\cdot x^*(n-2)-b^*(1)\cdot Q^*(n-1)-b^*(2)\cdot Q^*(n-2)|_{m_t}$$

(5.34)

Let us assume that the residue codes of the input sequence can be represented by B binary bits. So (5.34) can be written as

$$Q(n)_t = |a(0) \sum_{k=0}^{B-1} 2^k\, x(n)^{(k)} + a(1) \sum_{k=0}^{B-1} 2^k\, x(n-1)^{(k)} + a(2) \sum_{k=0}^{N-1} 2^k\, x(n-2)^{(k)}$$

$$- b(1) \sum_{k=0}^{B-1} 2^k\, Q(n-1)^{(k)} - b(2) \sum_{k=0}^{B-1} 2^k\, Q(n-2)^{(k)}|_{m_t} \qquad (5.35)$$

$$Q^*(n)_t = |a^*(0) \sum_{k=0}^{B-1} 2^k\, x^*(n)^{(k)} + a^*(1) \sum_{k=0}^{B-1} 2^k\, x^*(n-1)^{(k)}$$

$$+ a^*(2) \sum_{k=0}^{B-1} 2^k\, x^*(n-2)^{(k)} - b^*(i) \sum_{k=0}^{B-1} 2^k\, Q^*(n-1)^{(k)}$$

$$- b^*(2) \sum_{k=0}^{N-1} 2^k\, Q^*(n-2)^{(k)}|_{m_t} \qquad (5.36)$$

where $x(n-k)^{(k)}$ and $x(n-k)^{(k)} = 0$ or 1

and $Q^*(n-k)^{(k)}$ and $Q^*(n-k)^{(t)} = 0$ or 1

Now the resulting $Q(n)_t$ and $Q^*(n)_t$ have to be divided by S and quantized to integers for the next recursive computation. The above expressions (5.35,5.36) can be implemented using the bit-slice technique as shown in Fig. 5.10.

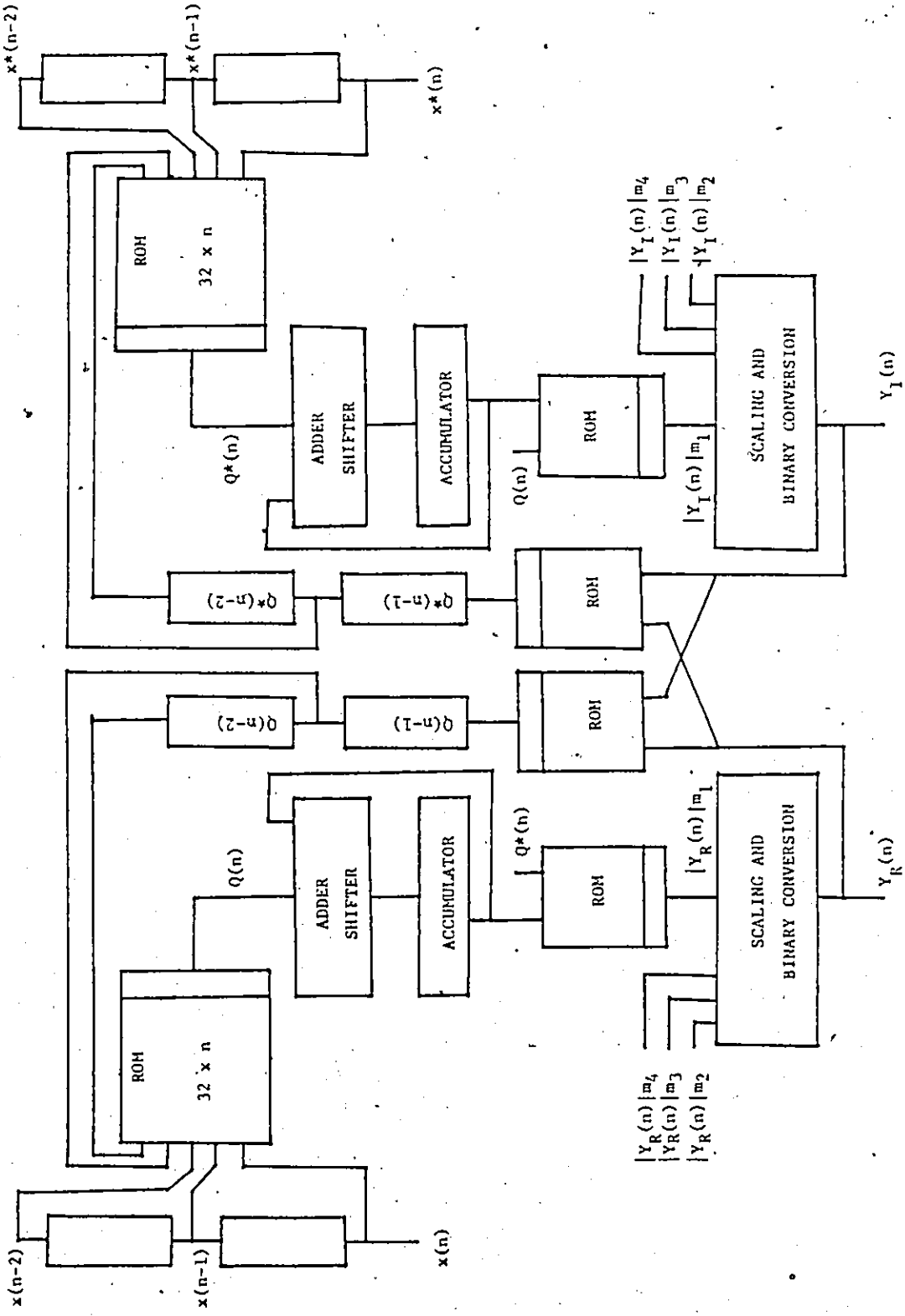


Fig. 5.10 Recursive filter bit-slice architecture for complex sequence in the QRNS.

The recursive filter architectures in Fig. 5.10 can be modified for the MQRNS by incorporating one additional channel to perform the third multiplication in (3.25) (Chapter III). It can be observed that both FIR and recursive filters require either 50% or 75% of the hardware required by conventional RNS filters for the QRNS and MQRNS respectively.

5.5 Summary

The implementation of the non-recursive digital filter has been treated in two different ways using the QRNS and MQRNS:

1. The direct FIR filter implementation
2. The direct FIR filter implementation using the bit-slice architectures.

We have also discussed the implementation of the recursive digital filters using the QRNS and MQRNS in the bit-slice architectures. In the implementation of direct FIR filters, the direct decoding of element pairs into a binary representation has been used. In this implementation we have shown that the improved Chinese Remainder Theorem techniques (fractional method) for the residue to binary conversion is perfectly suitable for the MQRNS. Therefore, both QRNS and MQRNS require almost the same amount of hardware and give the same data rate in implementing the direct FIR filter architectures. The implementation of bit-slice architectures using the QRNS requires 50% of the hardware using conventional RNS methods, whereas the implementation of the bit-slice architecture using the MQRNS requires 75% of the hardware. The 25% increase in hardware is due to the third multiplication required in the MQRNS.

Chapter VI

IMPLEMENTATION OF COMPLEX NUMBER THEORETIC TRANSFORM USING THE QRNS AND MQRNS

6.1 Introduction

Linear convolution can be described using equation (5.1) in Chapter V. Often the impulse response sequence will be a short finite sequence, and the data sequence will be a long sequence (speech, Radar etc.). The direct computations of $y(n)$ using (5.1) require N^2 complex multiplications and N-1 complex additions for each output sample $y(n)$. As N becomes larger, the direct implementation of (5.1) requires an excessive amount of computation. It is obvious that the large number of additions and multiplications required for the direct implementation of FIR filters limits the speed and efficiency. It has become an established fact in signal processing that transforms, with a cyclic convolution property, can be used indirectly to compute linear, aperiodic convolution. Indirect filtering, using the Discrete Fourier Transform (DFT) with a Fast Fourier Transform (FFT) type of algorithm [51], provides, in many cases, improvement in computational efficiency, but introduces quantization errors because of the reqirement for irrational coefficients (Sines and Cosines). The coefficients are generated from a primitive root of unity of the complex field, which can only be approximately represented. The general form of a DFT can be defined over a finite ring or field in which case the primitve root is exactly represented; such transforms are termed Number Theoretic Transforms (NTT). The main advantages of the NTT

are that indirect computation of convolution is error free, and computation over finite fields can be made very efficiently. The NTT has an identical structure to the DFT with the exception that it is computed over a finite ring or field rather than over the field of complex numbers. The NTT can be defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)\ \alpha^{kn} \qquad (6.1)$$

Like the DFT, the generator α is an Nth root of unity. In the case of an NTT, the generator belongs to either the finite field or ring.

6.2 Review of the State-of-the-Art of NTT

Over the last fifteen years, various authors have proposed the use of NTT over a variety of finite fields or rings, for error free, fast, efficient computation of convolution [23,24,25,26,27,28,34,54,63,64,66,67]. The NTT has been defined using algebraic properties of data over any integral domain by Nicholson in [52], and over any finite field by Pollard in [53] and over any finite ring by Dubois and Venetsanopoulos in [54]. In particular, Radar [26] proposed the Mersenne Number Transform (MNT) computed over a ring (or field) of integers modulo Mersenne number, $M = 2^P - 1$, p prime and showed that 2 is a p-th root of unity, and -2 is a 2p-th root of unity. The disadvantage of this multiplication free MNT is that FFT type algorithms cannot be used for the computation since the order of the transform is not a power of 2 and not even highly composite. Following Radar, Aggarwal and Burrus [27] proposed NTTs over rings or fields whose moduli are Fermat numbers $F = 2^b + 1$, $b = 2^n$, referred to as the Fermat Number Transforms (FNT) and $\alpha = 2$ allows transform length $N = 2^{n+1}$ and

$\alpha = 2^{2n-2}\ (2^{2n-1} - 1)$ allows $N = 2^{n+2}$. Hence the FNT can be computed using the FFT algorithm. The hardware implementation of an FNT is described in [55] for a sequence length of 64. In [56,57] Nussbaumer introduced the pseudo-Fermat and pseudo-Mersenne number theoretic transform.

The general form of the MNT and FNT is:

$$X(f) = \sum_{t=0}^{N-1} x(t)\ \alpha^{ft} \quad \text{for } f = 0, 1, \ldots N-1 \tag{6.2}$$

where α is a primitive Nth root of unity over the finite ring. The inverse transform is:

$$x(t) = N^{-1} \sum_{f=0}^{N-1} X(f)\ \alpha^{-ft} \quad \text{for } t = 0, 1, \ldots N-1 \tag{6.3}$$

with the restriction that N belongs to the ring. However, the main disadvantage of the MNT and FNT is the rigid relationship between the dynamic range and attainable transform length.

In order to extend the transofrm length, we can define an NTT over the extension field $GF(m^2)$ built from the Galois field GF(m) using the solution of the irreducible polynomial $x^2 + n = 0$ [24]. For primes of the form $4K + 3$, $n = 1$ is suitable. The resulting extension field operations obey the rules of the complex arithmetic. Such transforms are termed Complex Number Theoretic Transforms (CNTTs). The structural properties of complex residue rings have been proposed in [65] and [24]. The implementation of CNTTs over the complex residue rings has been proposed by Baraniecka and Jullien for primes of the form $4K + 1$ and $4K + 3$ [23,47].

To extend the dynamic range of complex number convolution, transforms over a ring isomorphic to the direct sum of Galois Fields have been developed [25]. The computation of a CNTT requires complex field multiplication; the usual procedure is to directly implement the rules of complex multiplication via real multiplications over the base field.

Recently, it has been shown that complex multiplication over Quadratic Rings [20,21] can be reduced to 2 base field multiplications with the proviso that the field modulus is of the form 4K + 1. This work was directed towards the study of RNS architectures in which parallel independent operations on several small quadratic rings were used to produce a result over a ring isomorphic to the direct sum of the smaller rings. The resulting number system is termed the Quadratic Residue Number System (QRNS) [20,21]. The results, however, also apply to operations computed over a single ring. These results have been extended to the Modified Quadratic Residue Number System (MQRNS) [15,16] and ring operations for moduli of the form 4K + 3.

In order to compute the Complex Number Theoretic Transform (CNTT), we can either use the extension field, modulo 4K + 3, with complex arithmetic implicit in the extension field operations, or we can explicitly form a complex residue ring, by computing complex arithmetic operations on elements of the base field. In the latter case, we can use primes of the form 4K + 1 and efficiently implement the computation using the QRNS. A restriction, in using the QRNS, is the form of the prime moduli (4K + 1). This restricts the choice of prime moduli which will allow appropriate transform length and dynamic range (see Table 6d). If we want

to clear this restriction by forming a complex extension field with prime moduli of the form 4K + 3, then we use the MQRNS, which is not as efficient as the QRNS but offers efficiencies over the direct computation [15].

It is obvious that the primary disadvantage of the QRNS is the restriction of the form of the moduli for RNS processing. It is not possible to choose small moduli or prime moduli to suit the required dynamic range for a CNTT processor in the QRNS because the CNTT has certain constraints over the moduli, transform length and primitive N-th root of unity./

For primes of the form, $m_i = 4K + 1$, we wish to have $N = 2^{\ell+1}$ because $\sqrt{-1}$ is in $R(m_i)$. This means we have to choose N such a way that $N \mid m_i - 1$. For $m_i = 4K + 3$, transform length can be chosen as 2^{P+1} or any divisor of 2^{P+1}. Table 6a and 6b lists all the prime moduli of the form 4K + 1 and 4K + 3 with radix-2 transform length, taking into consideration up to 8 bits for the implementation using look-up tables. Table 6c shows the moduli set for the QRNS and MQRNS and the combination of both for sequence lengths of 32 and 64. The maximum dynamic range offered by this set of moduli is also given.

It is observed from Table 6a that, if the transform length is above 64, there is no prime modulus available to implement the transform, and for a sequence length of 64, there is only one prime modulus {193} available. In the MQRNS, we have sufficient prime moduli for a sequence length of 64 {31,127, 191,223} which offers a dynamic range of 28.1 bits and is adequate for most of the practical problems.

In order to show the computation of the CNTT using the QRNS and MQRNS, let us define the CNTT in the following section.

Table 6a

Table of Primes $m = 4K + 1$ less than 257

K	m_i	Representation $t\ 2^{\ell} + 1$	Maximum Radix-2 Transform Length
1	5	2^2+1	4
3	13	$3\cdot2^2+1$	4
4	17	2^4+1	16
7	29	$7\cdot2^2+1$	4
9	37	$9\cdot2^2+1$	4
10	41	$5\cdot2^3+1$	8
13	53	$13\cdot2^2+1$	4
15	61	$15\cdot2^2+1$	4
18	73	$9\cdot2^3+1$	8
22	89	$11\cdot2^3+1$	8
24	97	$3\cdot2^5+1$	32
25	101	$25\cdot2^2+1$	4
27	109	$9\cdot2^2+1$	4
28	113	$7\cdot2^4+1$	16
34	137	$17\cdot2^3+1$	8
37	149	$37\cdot2^2+1$	4
39	157	$39\cdot2^2+1$	4
43	173	$43\cdot2^2+1$	4
45	181	$45\cdot2^2+1$	4
48	193	$3\cdot2^6+1$	64
49	197	$49\cdot2^2+1$	4
57	229	$57\cdot2^2+1$	4
58	233	$29\cdot2^3+1$	8
60	241	$15\cdot2^4+1$	16

Table 6b

Table of Primes m = 4K + 3
less than 257

K	m_i	Representation $q.2^P - 1$	Factorization of $m_i^2 - 1$	Maximum Radix-2 Transform Length
0	3	$2^2 - 1$	2^3	8
1	7	$2^3 - 1$	$3 \cdot 2^4$	16
2	11	$3 \cdot 2^2 - 1$	$5 \cdot 3 \cdot 2^3$	8
4	19	$5 \cdot 2^2 - 1$	$5 \cdot 3^2 \cdot 2^3$	8
5	23	$3 \cdot 2^3 - 1$	$11 \cdot 3 \cdot 2^4$	16
7	31	$2^5 - 1$	$5 \cdot 3 \cdot 2^6$	64
10	43	$11 \cdot 2^2 - 1$	$11 \cdot 7 \cdot 3 \cdot 2^3$	8
11	47	$3 \cdot 2^4 - 1$	$\cdot 23 \cdot 3 \cdot 2^5$	32
14	59	$15 \cdot 2^2 - 1$	$29 \cdot 5 \cdot 3 \cdot 2^3$	8
16	67	$17 \cdot 2^2 - 1$	$17 \cdot 11 \cdot 3 \cdot 2^3$	8
17	71	$9 \cdot 2^3 - 1$	$7 \cdot 5 \cdot 3^2 \cdot 2^4$	16
19	79	$5 \cdot 2^4 - 1$	$13 \cdot 5 \cdot 3 \cdot 2^5$	32
20	83	$21 \cdot 2^2 - 1$	$41 \cdot 7 \cdot 3 \cdot 2^3$	8
25	103	$13 \cdot 2^3 - 1$	$17 \cdot 13 \cdot 3 \cdot 2^4$	16
26	107	$27 \cdot 2^2 - 1$	$53 \cdot 3^3 \cdot 2^3$	8
31	127	$2^7 - 1$	$7 \cdot 3^2 \cdot 2^8$	256
32	131	$33 \cdot 2^2 - 1$	$13 \cdot 11 \cdot 5 \cdot 3 \cdot 2^3$	8
34	139	$35 \cdot 2^2 - 1$	$23 \cdot 7 \cdot 5 \cdot 3 \cdot 2^3$	8
37	151	$19 \cdot 2^3 - 1$	$19 \cdot 5^2 \cdot 3 \cdot 2^4$	16
40	163	$41 \cdot 2^2 - 1$	$41 \cdot 3^4 \cdot 2^3$	8
41	167	$21 \cdot 2^3 - 1$	$87 \cdot 7 \cdot 3 \cdot 2^4$	16
44	179	$45 \cdot 2^2 - 1$	$89 \cdot 5 \cdot 3^2 \cdot 2^3$	8
47	191	$3 \cdot 2^6 - 1$	$19 \cdot 5 \cdot 3 \cdot 2^7$	128
49	199	$25 \cdot 2^3 - 1$	$11 \cdot 5^2 3^2 \cdot 2^4$	16
52	211	$53 \cdot 2^2 - 1$	$53 \cdot 7 \cdot 5 \cdot 3 \cdot 2^3$	8
55	223	$7 \cdot 2^5 - 1$	$37 \cdot 7 \cdot 3 \cdot 2^6$	64
56	227	$57 \cdot 2^2 - 1$	$113 \cdot 19 \cdot 3 \cdot 2^3$	8
59	239	$15 \cdot 2^4 - 1$	$17 \cdot 7 \cdot 5 \cdot 3 \cdot 2^5$	32
62	251	$63 \cdot 2^2 - 1$	$7 \cdot 5^3 3^2 \cdot 2^3$	8

Table 6c

Prime Moduli Required in the QRNS and MQRNS for the Sequence Length of 32 and 64.

	Moduli Set and Dynamic Range					
N	QRNS	Dynamic Range	MQRNS	Dynamic Range	QRNS & MQRNS	Dynamic Range
32	97,193	14.4 bits	31,47,79, 127,191, 223,239	47.1 bits	31,47,79, 97,127,191 193,223,239	61.5 bits
64	193	7.5 bits	31,127, 191,223	28.1 bits	31,127,191 193,223	35.1 bits

6.3 Complex Number Theoretic Transform

The CNTT can be defined as

$$A(k) = \sum_{t=0}^{N-1} a(t)\, \alpha^{kt} \quad \text{for } k = 0, 1, \ldots N-1$$

$$a(t) = N^{-1} \sum_{k=0}^{N-1} A(k)\, \alpha^{-kt} \quad \text{for } t = 0, 1, \ldots N-1 \tag{6.4}$$

where $A(k), \alpha \in GF(m^2)$

with $A(k) = a(k) + \hat{i}b(k)$

and $\alpha = \gamma + \hat{i}\delta$

and $\hat{i}$ is a solution to the irreducible polynomial $x^2 + 1 = 0$ defined over $GF(m^2)$ and $a(k), b(k) \in GF(m))$, $N \mid m^2-1$ and $\mid \alpha^N \mid_M = 1$, $\mid \alpha^{N'} \mid_M \neq 1$, $N' < N$.. As m and N are mutually prime, there exists a multiplicative inverse of N modulo m.

The residue arithmetic can be efficiently implemented for the computation of the transform, if we use relatively small integers for each prime modulus, and generate the required dynamic range by combining a sufficient number of prime moduli [79]. Thus computing the transform in a finite ring which is a direct sum of several Galois fields of second degree, $GF(m_i^2)$, $i \in (1,2,\ldots,L)$,

$$R \tilde{=} GF(m_1^2) \oplus GF(m_2^2) \oplus \ldots\ldots \oplus GF(m_L^2) \tag{6.5}$$

increase the dynamic range to $\prod_{L-1}^{L} m_i$.

To support the implementation of a fast algorithm for the computation of the transform, the transform length should be highly composite and

ideally $N = 2^B \cdot \alpha$ is a root of unity of order N which is the generator of N-elements defined as

$$\mu = \{1, \alpha, \ldots . \alpha^{N-1}\} \tag{6.6}$$

There are a number of constraints imposed on the transform length N, modulus m and primitive root α. For 8 bit prime moduli, the following Table 6d gives the transform length, suitable prime moduli, and the primitive root of unity for sequence lengths of 32 to 256.

In the following sections, we present the computation of the CNTT using the QRNS and MQRNS [17].

Table 6d
Transform lengths and primitive root of unity
for prime moduli 4K + 1 and 4K +3

	4K + 1		4K +3	
Sequence Length	Prime Moduli	Primitive Root	Prime Moduli	Primitive Root
32	97,193	$41+\hat{i}$, $48+\hat{i}14$	31,47,79, 127,191, 223,239	$11+\hat{i}2$, $18+\hat{i}2$, $33+\hat{i}4$, $30+\hat{i}25$, $152+\hat{i}77$, $56+\hat{i}87$, $75+i50$
64	193	$7+\hat{i}$	31,127, 191,223	$14+\hat{i}4$, $120+\hat{i}29$, $118+\hat{i}28$, $94+\hat{i}23$
128	None		127,191	$39+\hat{i}2$, $66+\hat{i}6$
256	None		127	$54+\hat{i}2$

6.4 Computation of CNTT Using the QRNS and MQRNS

Let us consider a complex sequence

$$a(t) = x(t) + \hat{i}y(t) \quad \text{for } t = 0,1,\ldots .N-1 \tag{6.7}$$

and let the primitive Nth root of unity be

$$\alpha = \gamma + \hat{i}\delta \tag{6.8}$$

The CNTT defined in (6.4) can be computed using the QRNS as follows:

$$A(k) = \Big| \sum_{t=0}^{N-1} a(t) \cdot B(tk) \Big|_m$$

$$A^*(k) = \Big| \sum_{t=0}^{N-1} a^*(k) \cdot B^*(tk) \Big|_m \tag{6.9}$$

and its inverse is

$$a(t) = \Big| N^{-1} \sum_{k=0}^{N-1} A(k) \quad B_{(tk)}^{(-)} \Big|_m$$

$$a^*(t) = \Big| N^{-1} \sum_{k=0}^{N-1} A^*(k) \quad B^{*(-)}_{(tk)} \Big|_m \tag{6.10}$$

where $a(t)$, $a^*(t)$, $B(tk)$ and $B^*(tk)$ can be computed in the QRNS as follows

$$a(t) = |x(t) + jy(t)|_m \qquad a^*(t) = |x(t) - jy(t)|_m$$

$$B(tk) = |\gamma(tk) + j\delta(tk)|_m \qquad B^*(tk) = |\gamma(tk) - j\delta(tk)|_m \tag{6.11}$$

where $j \equiv \sqrt{-1} \mod m$ and m is of the form $4K + 1$. The CNTT defined in (6.4) can be computed using the MQRNS as follows:

$$A(k) = \Big| \Big| \sum_{t=0}^{N-1} a(t) \cdot B(tk) \Big|_m - S(tk) \Big|_m$$

$$A^*(k) = \Big| \Big| \sum_{t=0}^{N-1} a^*(t) \cdot B^*(tk) \Big|_m - S(tk) \Big|_m \tag{6.12}$$

and its inverse is

$$a(t) = \Big| \Big| N^{-1} \sum_{k=0}^{N-1} A(k) \cdot B_{(tk)}^{(-)} \Big|_m - S(tk) \Big|_m$$

$$a^*(t) = \left|\left|N^{-1} \sum_{k=0}^{N-1} A^*(k) \cdot B^{*(-)}(tk)\right|_m - S(tk)\right|_m \qquad (6.13)$$

where $a(t)$, $a^*(t)$, $B(tk)$, $B^*(tk)$ and $S(tk)$ can be computed in the MQRNS as follows:

$$\begin{aligned} a(t) &= |x(t) + \hat{j}y(t)|_m & a^*(t) &= |x(t) - \hat{j}y(t)|_m \\ B(tk) &= |\gamma(tk) + \hat{j}\delta(tk)|_m & B^*(t) &= |\gamma(tk - \hat{j}\,\delta(tk)|_m \\ S(tk) &= \left|\left|\hat{j}^2 + 1\right|_m \cdot y(t) \cdot S(tk)\right|_m \end{aligned} \qquad (6.14)$$

and $\hat{j} = \sqrt{-1} \mod m$ and m is of any form except $4K + 1$. $\gamma(tk)$ and $\delta(tk)$ are the real and imaginary part of the elements in (6.6).

The computation of the finite cyclic convolution using (6.9,6.10) can be explained in the following section in the QRNS and MQRNS.

6.5 Computation of Cyclic Convolution in the QRNS and MQRNS

Digital convolution can be implemented either directly in the time domain, (see section 5.2) or in the transform domain. In this section the digital convolution is implemented in the transform domain, of a CNTT. We will symbollically define digital convolution as:

$$y(n) = h(n) \circledast x(n) \qquad (6.15)$$

The cyclic convolution property states that

$$\text{CNTT}\,[h(n) \circledast x(n)] = \text{CNTT}\,[h(n) \circledast \text{CNTT}\,[x(n)] \qquad (6.16)$$

and the convolution output

$$y(n) = \text{ICNTT}\,\{\text{CNTT}\,[h(n)] \cdot \text{CNTT}\,[x(n)]\} \qquad (6.17)$$

where ICNTT is the inverse CNTT. This implies that the N-point convolution can be obtained by an inverse transform of the pairwise product of two vectors in the transform domain.

The convolution implemented using (6.17) is called circular or cyclic or periodic convolution. The results of the finite and circular convolution are equal if zeros are appended to h(n) and x(n) to prevent folding or aliasing [83], so that transform length is at least equal to N + L - 1 where N and L are the duration of h(n) and x(n) respectively.

Long input sequences, filtered by a FIR filter whose kernel is relatively short, can be divided into blocks and the conventional over-lap-add or over-lap-save techniques [83,86] can be used to compute the output signal from the results of the circular convolutions.

The technique of convolving two finite sequences, using transform techniques has been called fast convolution. The term 'fast' is used, because the transform can be computed rapidly and efficiently by the Fast Fourier Transform (FFT) algorithm [51].

6.5.1 Computation of the Cyclic Convolution in the QRNS

$$\text{Let } h(n) = a(n) + \hat{i}b(n); \quad \text{and } x(n) = c(n) + \hat{i}d(n) \tag{6.18}$$

be the two complex sequences to be convolved for n = 0, 1,... N-1.

Let $\alpha = \gamma + \hat{i}\delta$ be the primitive element of order N. The element pairs of the two complex sequences can be calculated as follows:

$$\begin{aligned} h(n) &= |a(n) + jb(n)|_m & h^*(n) &= |a(n) - jb(n)|_m \\ x(n) &= |c(n) + jd(n)|_m & x^*(n) &= |c(n) - jd(n)|_m \end{aligned} \tag{6.19}$$

where a(n), b(n), c(n), d(n) ϵ R(m) and m is of the form 4K + 1 and $j \equiv \sqrt{-1} \mod m$, $j \epsilon R(m)$. The CNTT [h(n)] and CNTT [x(n)] are computed using the element pairs in the QRNS as follows.

$$T(k) = \left| \sum_{n=0}^{N-1} h(n) \cdot B(kn) \right|_m$$
$$T^*(k) = \left| \sum_{n=0}^{N-1} h(n) \cdot B^*(kn) \right|_m \qquad (6.20)$$

$$U(k) = \left| \sum_{n=0}^{N-1} x(n) \cdot B(kn) \right|_m$$
$$U^*(k) = \left| \sum_{n=0}^{N-1} x(n) \cdot B^*(kn) \right|_m, \text{ where} \qquad (6.21)$$

B(kn) and $B^*(kn)$ are the element pairs of the twiddle factors.

Let V(n) and $V^*(n)$ be the multiplied values of the element pairs of (6.20, 6.21), which are computed as

$$V(n) = \left| \sum_{k=0}^{N-1} T(k) \cdot U(k) \right|_m$$
$$V^*(n) = \left| \sum_{k=0}^{N-1} T(k) \cdot U(k) \right|_m \quad \text{for } n = 0, 1, \ldots N-1 \qquad (6.22)$$

Let Qn and Q*n be the element pairs of the inverse CNTT of Vn and V^*n which can be computed using (6.10) as:

$$Q(n) = \left| N^{-1} \sum_{t=0}^{N-1} V(t) \cdot B^{(-)}(tk) \right|_m$$
$$Q^*(n) = \left| N^{-1} \sum_{t=0}^{N-1} V^*(t) \cdot B^{*(-1)}_{(tk)} \right|_m \quad \text{for } k = 0, 1, \ldots N-1 \qquad (6.23)$$

The real and imaginary part of the convolution output is

$$Y_R(n) = \left|2^{-1}\ (Q(n) + Q^*(n))\right|_m$$
$$Y_I(n) = \left|2^{-1} \cdot j^{-1}\ (Q(n) - Q^*(n))\right|_m \tag{6.24}$$

6.5.2 Computation of the Cyclic Convolution in the MQRNS

Let us consider the same two sequences as in (6.19) and $\alpha = \eta + i\zeta$ be the primitive Nth root of unity.

Let $D = \left|\hat{j}^2 + 1\right|_m$

$$\mu = (1,\ \alpha, \ldots\ldots \alpha^{N-1})$$

The element pairs of the two complex sequences can be calculated as follows:

$$h(n) = \left|a(n) + \hat{j}b(n)\right|_m \qquad h^*(n) = \left|a(n) - \hat{j}b(n)\right|_m$$
$$x(n) = \left|c(n) + \hat{j}d(n)\right|_m \qquad x^*(n) = \left|c(n) - \hat{j}d(n)\right|_m \tag{6.25}$$

and

$$S(n)_1 = \left|D \cdot b(n) \cdot \zeta(n)\right|_m$$
$$S(n)_2 = \left|D\ \ d(n) \cdot \zeta(n)\right|_m \tag{6.26}$$

where $a(n)$, $b(n)$, $c(n)$, $d(n), \eta, \zeta \in R(m)$ and m is of any form except $4K + 1$ $\hat{j} \equiv \sqrt{n} \bmod m$, $\hat{j} \in R(m)$.

The CNTT $[h(n)]$ and CNTT $[x(n)]$ are computed in the MQRNS as follows:

$$T(k) = \left|\left| \sum_{n=0}^{N-1} h(n) \cdot \quad B^*(nk) \right|_m - S(n)_1 \right|_m$$

$$T^*(k) = \left|\left| \sum_{n=0}^{N-1} h^*(n) \quad B^*(nk) \right|_m - S(n)_1 \right|_m \qquad (6.27)$$

$$U(k) = \left|\left| \sum_{n=0}^{N-1} x(n) \quad B(nk) \right|_m - S(n)_2 \right|_m$$

$$U^*(k) = \left|\left| \sum_{n=0}^{N-1} x^*(n) \quad B^*(nk) \right|_m - S(n)_2 \right|_m \qquad (6.28)$$

Let

$$W(k) = \left|2^{-1} \cdot \hat{j}^{-1} \, (T(k) - T^*(k))\right|_m$$

$$Z(k) = \left|2^{-1} \quad \hat{j}^{-1} \, (U(k) - U^*(k))\right|_m$$

$$S(k) = \left|D \cdot W(k) \cdot Z(k)\right|_m$$

Let $V(n)$ and $V^*(n)$ be the multiplied values of the element pairs in (6.27,6.28), then

$$V(n) = \left| \sum_{k=0}^{N-1} T(k) \quad U(k) \right|_m - S(k)_m$$

$$V^*(n) = \left| \sum_{k=0}^{N-1} T^*(k) \quad U^*(k)_m - S(k) \right|_m \quad \text{for } n = 0,1,\ldots.N-1 \qquad (6.29)$$

Let

$$E(n) = \left|2^{-1} \cdot \hat{j}^{-1} \, (V(n) - V^*(n))\right|_m$$

and

$$r(n) = \left|D \cdot E(n) \cdot \zeta(n)\right|_m$$

where $\zeta(n)$ is the imaginary part of the twiddle factor.

Let $Q(n)$ and $Q^*(n)$ be the inverse CNTT of $V(n)$ and $V^*(n)$ which can

be computed as shown below:

$$Q(n) = \left| N^{-1} \sum_{k=0}^{N-1} V(k) \cdot B^{(-)}(nk) \right|_m - V(k)\Big|_m$$

$$Q^*(n) = \left| N^{-1} \sum_{k=0}^{N-1} V(k) \cdot B^{(-)}(nk) \right|_m - V(k)\Big|_m \qquad (6.30)$$

The real and imaginary part of the convolution output is

$$Y_R(n) = \left| 2^{-1} \cdot (Q(n) + Q^*(n)) \right|_m$$

$$Y_I(n) = \left| 2^{-1} \cdot \hat{j}^{-1} \cdot (Q(n) - Q^*(n)) \right|_m \qquad (6.31)$$

6.6 Convolution Over the Direct Sum of the Finite Ring

The computation of the convolution over the ring isomorphic to a direct sum of L quadratic residue rings, will increase the allowable dynamic range for the convolution sum to $M = \prod_{i=1}^{L} m_i$.

That is,

$$R(M) \tilde{=} QR(m_1) \oplus QR(m_2) \oplus \ldots\ldots\ldots \oplus QR(m_L) \qquad (6.32)$$

The CNTT can be computed modulo distinct primes $\{m_i\}$ and the result can be reconstructed according to the Chinese Remainder Theorem (CRT) or mixed radix conversion techniques.

As the sequences to be convolved are complex, the absolute upper bound for the real and imaginary parts of the sequence and the impulse response can be computed as follows:

Let $h(n) = a(n) + \hat{i}b(n)$ be the impulse response and

$x(n) = c(n) + \hat{i}d(n)$ be the complex sequence (6.33)

then

$$\text{Max}|c(n)| \cdot \text{Max}|a(n)| + \text{Max}|d(n)| \cdot \text{Max}|b(n)| \leq M-1/2N$$

$$\text{Max}|c(n)| \cdot \text{Max}|b(n)| + \text{Max}|d(n)| \cdot \text{Max}|a(n)| \leq M-1/2N \quad (6.34)$$

We will assume that

$$\text{Max}|a(n)| = \text{Max}|b(n)| = \text{Max}|c(n)| = \text{Max}|d(n)| = W \quad (6.35)$$

where W is the dynamic range of a(n), b(n), c(n) and d(n).

Then equation (6.34) can be written as

$$W = \sqrt{(M-1)/4N} \quad (6.36)$$

This bound on the dynamic range is pessimistic for many practical applications [23]. To implement the CNTT over a direct sum of the quadratic residue ring, we find a moduli set $\{m_i\}$ that can provide the same order of transform length $N = 2^B$. To provide sufficient dynamic range ($M = \prod_{i=1}^{L} m_i$), we can combine small prime moduli of the form $4K + 1$ and $4K + 3$ and the procedure in sections (6.5.1,6.5.2) can be used with the output being decoded using the techniques proposed in Chapter IV.

6.7 Example

In order to verify the procedure given, cyclic convolution in the QRNS and MQRNS has been simulated with a software FNTT. The topological structure of the (FNTT) is identical to the FFT algorithm, provided the

final convolution result does not overflow the ring modulus, then we have a direct correspondence between the result over the ring and the quantized sub-cover of the complex field over which the FFT has been computed. The FFT butterfly subroutine is modified to handle the binary operations of the quadratic residue ring; the input samples and the twiddle factors are fed into this butterfly as element pairs.

In implementing a convolution example, we have chosen an impulse response of a linear phase band pass filter and a random input sequence.

Various approaches for designing FIR digital filters using the theory of weighted Chebyshev approximation, are discussed in [45]. A general purpose computer program which is capable of designing a large class of optimum FIR linear phase digital filters has been written in Fortran [62]. This program gives the desired impulse response of the optimum FIR filter specified as low pass, high pass, band pass, etc.

Let us consider a linear phase band pass filter for a sequence length of 32. Table 6e shows the full precision impulse response designed using the program in [62] and Fig. 6.1 gives the log magnitude response.

The impulse response in Table 6e is multiplied by 10^4 and the samples rounded off to the closest integer value. The resulting integer valued impulse response is shown in Table 6f and the log magnitude response is shown in Fig. 6.2. From the figures 6.1 and 6.2, we can observe that the degradation due to the integer truncation is small.

This integer valued impulse response can be taken as one sequence

Table 6e

Finite Impulse Response (FIR) Linear Phase Digital Filter Design, Remez Exchange Algorithm, Bandpass Filter
Filter Length 32

Impulse Response				
H(1)	=	-0.53534121E-02	=	H(32)
H(2)	=	0.99027198E-03	=	H(31)
H(3)	=	0.75733545E-02	=	H(30)
H(4)	=	-0.65141192E-02	=	H(29)
H(5)	=	0.13960525E-01	=	H(28)
H(6)	=	0.22951469E-02	=	H(27)
H(7)	=	-0.19994067E-01	=	H(26)
H(8)	=	0.71369560E-02	=	H(25)
H(9)	=	-0.39657363E-01	=	H(24)
H(10)	=	0.11260114E-01	=	H(23)
H(11)	=	0.66233643E-01	=	H(22)
H(12)	=	-0.10497223E-01	=	H(21)
H(13)	=	0.85136133E-01	=	H(20)
H(14)	=	-0.12024993E 00	=	H(19)
H(15)	=	-0.29678577E 00	=	H(18)
H(16)	=	0.30410917E 00	=	H(17)

Table 6f

Integer Valued Impulse Response of the Linear Phase Bandpass Filter For the Length of 32

h(1)	=	-58	=	h(32)
h(2)	=	10	=	h(31)
h(3)	=	76	=	h(30)
H(4)	=	-65	=	h(29)
h(5)	=	140	=	h(28)
h(6)	=	23	=	h(27)
h(7)	=	-200	=	h(26)
h(8)	=	71	=	h(25)
h(9)	=	-397	=	h(24)
h(10)	=	113	=	h(23)
h(11)	=	662	=	h(22)
h(12)	=	-105	=	h(21)
h(13)	=	851	=	h(20)
h(14)	=	-1202	=	h(19)
h(15)	=	-2968	=	h(18)
h(16)	=	3041	=	h(17)

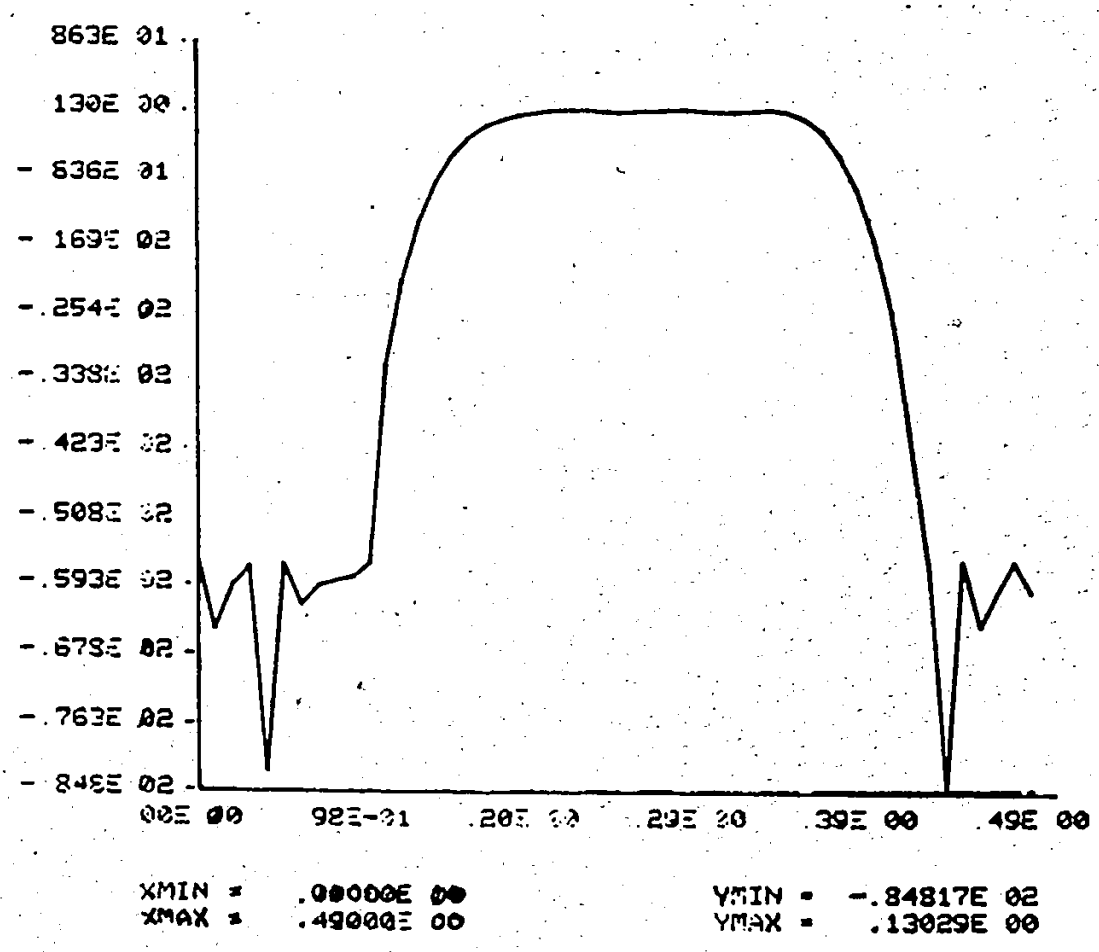


Fig. 6.1 Log magnitude response for an N = 32 bandpass filter.

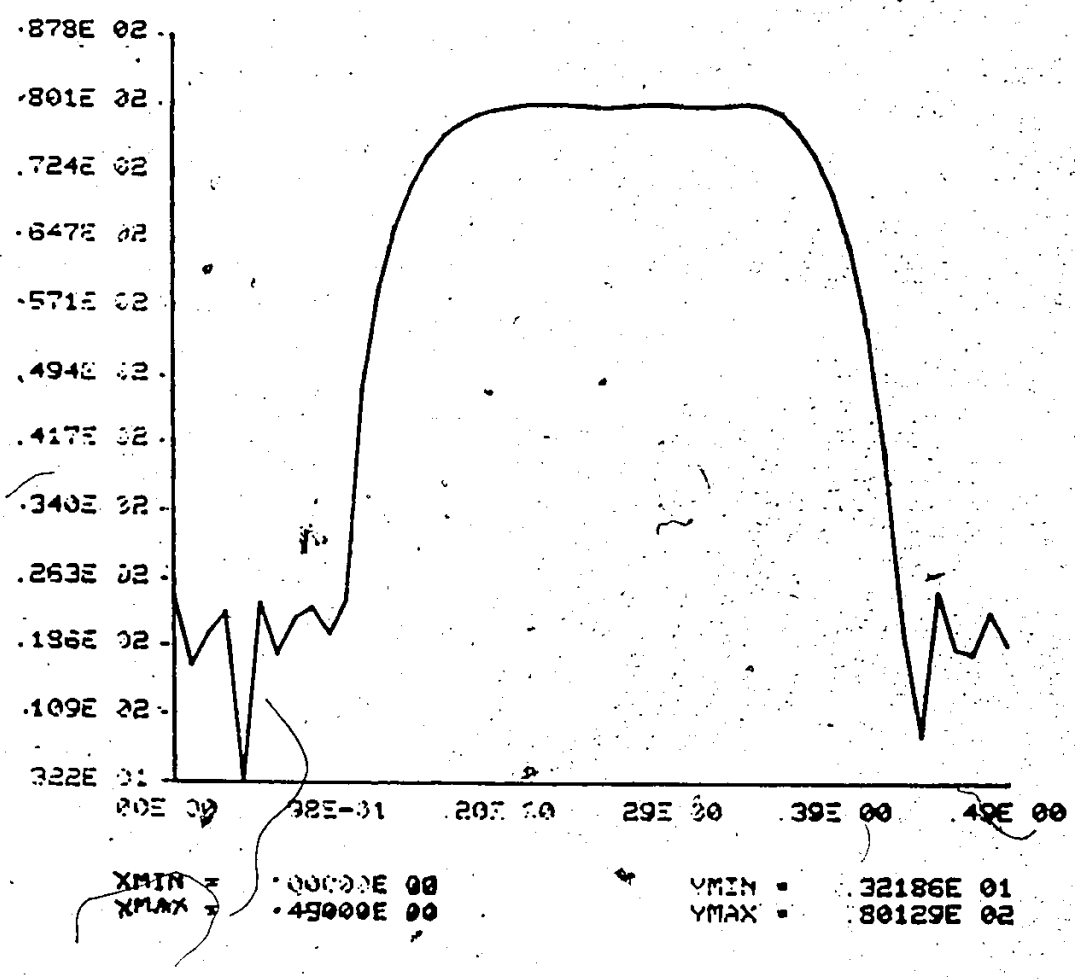


Fig. 6.2 Log magnitude response for an N = 32 bandpass integer rounded impulse response.

Table 6g

Element Pairs of the Integer-Valued Impulse Response

h(n)	$h^*(n)$
2147483589	2147483589
10	10
76	76
2147483582	2147483582
140	140
23	23
2147483447	2147483447
71	71
2147483250	2147483250
113	113
662	662
2147483447	2147483447
851	851
2147482445	2147482445
2147480679	2147480679
3041	3041
3041	3041
2147480679	2147480679
2147482445	2147482445
851	851
2147483542	2147483542
662	662
113	113
2147483250	2147483250
71	71
2147483447	2147483447
23	23
140	140
2147483582	2147483582
76	-76
10	10
2147483589	2147483589

Table 6h

Element Pairs of the Twiddle Factors

B_{kn}	B^*_{kn}
1	i
1241207368	1179735656
387824019	578122920
1380488182	1449692322
65536	0
1380488182	697791325
387824019	1569360727
273459377	2086011935
1	2147483646
61471712	1874024270
578122920	1759659628
1449692322	766995465
0	2147418111
697791325	766995465
1569360727	1759659628
2086011935	1874024270
2147482646	2147483646
1874024270	2086011935
1759659628	1569360727
766995465	697791325
2147418111	0
766995465	1449692322
1759659628	578122920
1874024270	61471712
2147483646	1
2086011935	273459377
1569360727	387824019
697791325	1380488182
0	65536
1449692322	1380488182
578122920	387824019
61471712	273459377

Table 6i

The Random Sequence

c(n)	d(n)
53	74
63	38
35	30
63	43
98	25
2	63
64	55
85	7
58	54
34	85
3	92
62	95
8	45
7	8
1	85
72	84
88	78
45	17
96	76
43	31
50	44
22	66
96	24
31	73
78	60
84	37
36	67
7	28
10	15
55	19
53	81
51	86

Table 6j

The Element Pairs of Random Sequence

x(n)	$x^*(n)$
127	2147483626
101	25
65	5
106	20
123	73
65	2147483586
119	9
92	78
112	4
119	2147483596
95	2147483558
157	2147483614
53	2147483610
15	2147483646
86	2147483563
156	2147483635
166	10
62	28
172	20
74	12
94	6
88	2147483603
120	72
104	2147483605
138	18
121	47
103	2147483616
35	2147483626
25	2147483642
74	36
134	2147483619
137	2147483612

Table 6k

Convolution Output in the MQRNS

$Y_R(n)$	$Y_I(n)$
2147428117	2147236403
2147406873	2147460211
182347	36699
2147389934	2147422440
2147281277	137630
174645	2147423692
2147475949	2147358721
2147299602	130491
237407	2147409547
179888	16574
2147262853	248772
2147275767	2147330052
29217	2147207177
224272	33614
20827	193095
2147367977	191499
93911	2147384436
2147329437	2147268979
2147382211	10186
382875	2147439141
2147409181	105278
2147099096	320134
190127	2147293829
249448	2147135520
2147334268	68494
2147278270	148171
74738	220400
274831	81079
2147448919	2147065152
2147158940	2147259955
5403	396330
248544	194144

by assuming the imaginary term of the sequence to be zero. Let us consider a prime modulus $M = 2^{31} - 1$ (2147483647), which is a Mersenne prime, and choose the operator $\hat{j} = 1$. Then the dynamic range of the input sequence for the finite convolution is bounded by approximately 12 bits. The primitive N-th root of unity is $1241207368 + \hat{i}1179735656$ which is computed using the algorithm in [24] and the element pairs of the twiddle factors is computed as shown in Table 6g.

The Table 6h shows the element pairs of the integer valued impulse response. We will select an input random sequence with statistical characteristics as given in [85]. The length of the random sequence to be 32 with each sample bounded within 2^{12}. The sequence is shown in the Table 6i. The element pairs of this random sequence are shown in Table 6j. The cyclic convolution is computed using the algorithm developed in section 6.5.2 and the finite convolution output is shown in the Table 6k.

6.8 Butterfly Implementation in the QRNS and MQRNS

The basic computational unit (BCU) to implement the CNTT, using the FNTT algorithm, is a radix-2 butterfly. The flow chart of the basic butterfly operation is as shown in Fig. 6.3. The important properties of the QRNS and MQRNS are that the separate binary ring operations of multiplication and addition on the elements and their conjugates yield results isomorphic to multiplication and addition respectively over the complex residue ring C(M)[21]. This means that once the input has been transformed into a QRNS (MQRNS) representation, this form can be maintained throughout for all the computations, with conversion back to a conventional representation being performed at the output stage. This particular property is

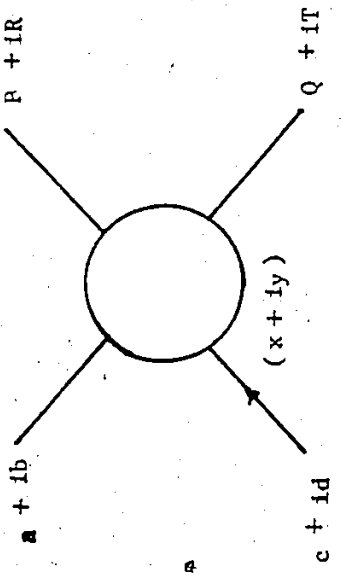


Fig. 6.3 Basic Butterfly Structure

useful for the computation of the butterfly structure in the implementation of the CNTT using the FNTT algorithm. Let $a + \hat{i}b$, $c + \hat{i}d$ be the two complex input samples and $x + \hat{i}y$ the twiddle factor. The radix-2 butterfly repeatedly performs the following computations.

$$\begin{aligned} Z &= \left|\left|(c + \hat{i}d)\cdot(x + \hat{i}y)\right|_m + (a + \hat{i}b)\right|_m \\ Y &= \left|\left|(c + \hat{i}d)\cdot(x + \hat{i}y)\right|_m - (a + \hat{i}b)\right|_m \end{aligned} \tag{6.37}$$

The equation (6.37) can be implemented using the QRNS or MQRNS

In the QRNS

$$\begin{aligned} Z &= \left|\left|A \cdot B\right|_m + c\right|_m & Y &= \left|\left|A \cdot B\right|_m - c\right|_m \\ Z^* &= \left|\left|A^* \cdot B^*\right|_m + c^*\right|_m & Y^* &= \left|\left|A^* \cdot B^*\right|_m - c^*\right|_m \end{aligned} \tag{6.38}$$

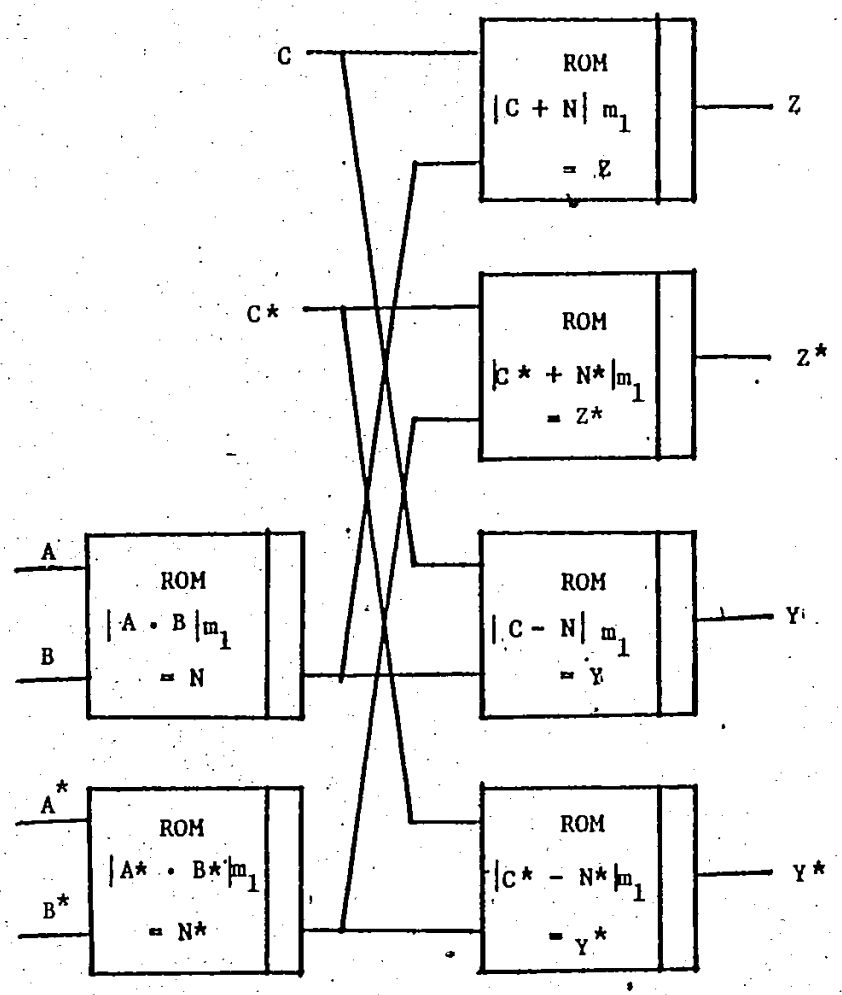
where A, A^*, B, B^*, C and C^* are the element pairs of the two input samples and the twiddle factor respectively, and Z, Z^*, Y and Y^* are the element pairs of the butterfly output.

In the MQRNS

$$\begin{aligned} Z &= \left|\left|\left|A \cdot B\right|_m - S\right|_m + C\right|_m & Y &= \left|\left|\left|A \cdot B\right|_m - S\right|_m - C\right|_m \\ Z^* &= \left|\left|\left|A^* \cdot B^*\right|_m - S\right|_m + C^*\right|_m & Y^* &= \left|\left|\left|A^* \cdot B^*\right|_m - S\right|_m - C^*\right|_m \end{aligned} \tag{6.39}$$

where $S = \left|\left|\hat{j}^2 + 1\right|_m \cdot d \cdot y\right|_m$.

The implementation of the above operations (6.38, 6.39) can be accomplished by the look-up table approach. For high speed realization, the look-up table approach using ROM arrays offers the better solution [1].



$C = |a + ib|_{m_1}$

$C^* = |a - ib|_{m_1}$

$A = |c + id|_{m_1}$

$A^* = |c - id|_{m_1}$

$B = |x + iy|_{m_1}$

$B^* = |x - iy|_{m_1}$

Fig. 6.4 Implementing butterfly structure in the QRNS.

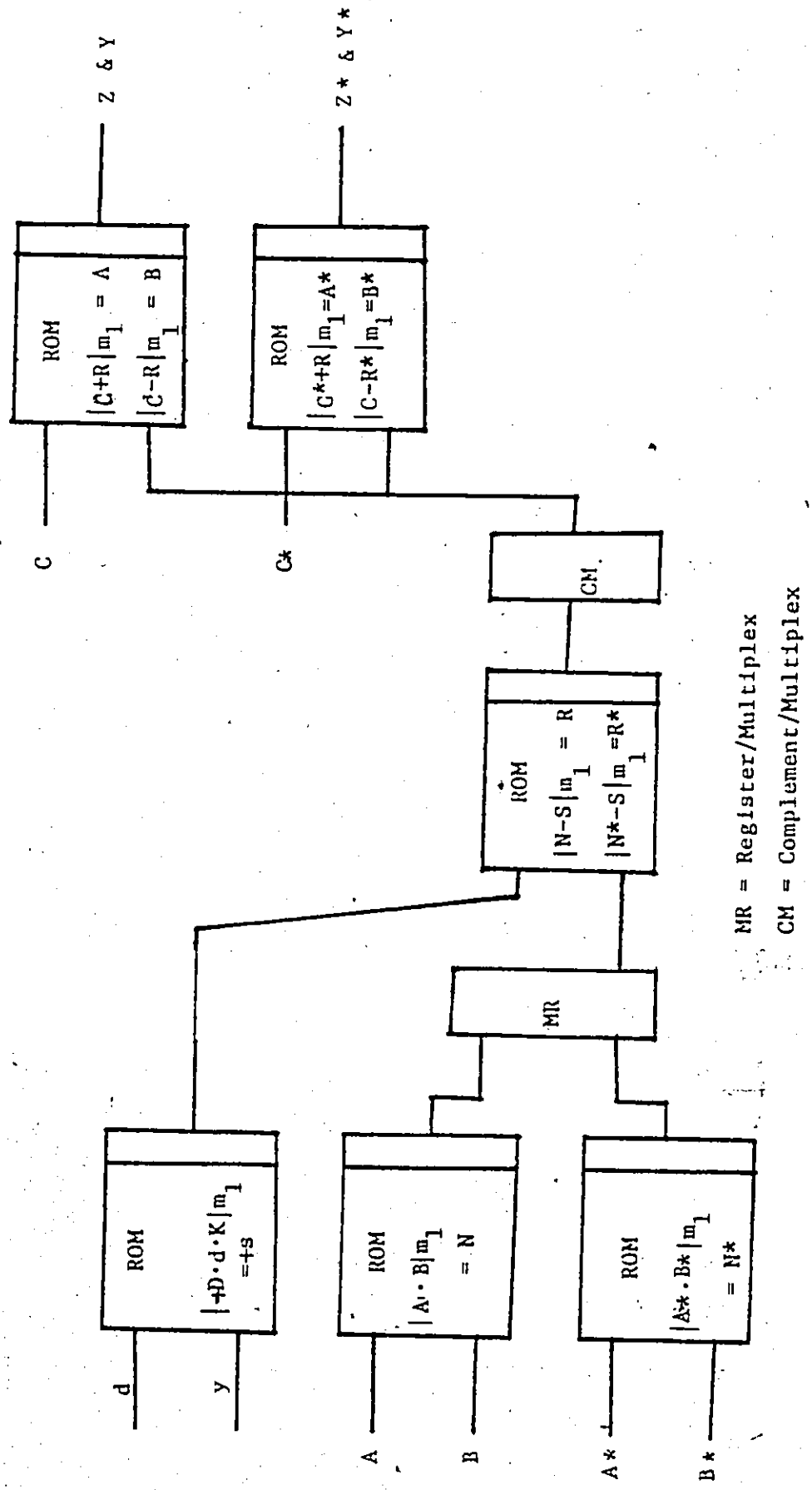


Fig. 6.5 Implementing butterfly structure in the MQRNS.

Table 61

Number of PROMs Required for a CNTT Butterfly in the Conventional RNS QRNS, MQRNS

Number of Sequences	Conventional RNS	QRNS	MQRNS
16	40	24	27
32	50	30	34
64	60	36	41
128	70	42	48
256	80	48	55

One of the advantages of structures using ROM arrays is the possibility of easy pipelining for high speed throughput [14]. Since we are interested in high speed implementation of the transform using the FNTT algorithm with PROM (Programmable ROM) arrays the range of the prime moduli and commercially available PROM packages are the major criteria.

According to the present state-of-the-art, the number of PROMs required per stage for the implementation of a butterfly operation in the conventional RNS for the complex sequence is 10 [78, pp.76]. Fig. 6.4 shows the implementation of a radix-2 butterfly (BCU) as in (6.38) using PROMs for a pipeline FNTT, in the QRNS. It requires only 6 PROMs per stage.

Fig. 6.5 shows the implementation of the same butterfly using the MQRNS. Here also, we need only 6 PROMs per stage, but the subsequent stages require 7 PROMs because we have to compute the imaginary part of the residue codes that are to be used in the next stage. These butterflies can be cascaded depending on the number of stages required according to the sequence length. Compared to the conventional RNS implementation of the CNTT, the QRNS and MQRNS offer better hardware savings. Table 6ℓ shows the number of PROMs required to implement the butterfly operation in the conventional RNS, QRNS and MQRNS.

Using the CNTT, in the following sections, a generalized FIR filter structure is developed.

6.9 Recursive FIR Filter Structures

A digital filter computed over the complex field can be represented using a set of linear difference equations with constant coefficients.

It has been established that the z-transform is the most suitable technique in the design and analysis of these difference equations.

Let

$$H(z) = \sum_{n=0}^{N-1} H(n)\, z^{-n} \tag{6.40}$$

be the transform of the sequence h(n), over the complex field [88], where N is the sequence length or window size. It can be shown that the z-transform of a sequence N can be represented in terms of N samples at equal spacing around the unit circle. For a FIR filter, the system function can be expressed as

$$H(z) = (1 - z^{-N})1/N \sum_{k=0}^{N-1} \hat{H}(k)/1 - W_N^{-k}\, z^{-1} \tag{6.41}$$

where

$$W_N^{-k} = e^{\hat{i}\frac{2\pi k}{N}}, \quad \hat{i} = \sqrt{-1}$$

$$\hat{H}(k) = H(z) \,|\, z = W_N^{-k} = \sum_{n=0}^{N-1} h(n)\, W_N^{-k} \tag{6.42}$$

The $\{\hat{H}(k)\}$ are called frequency samples. Fig. 6.6 shows the filter structure.

Equation (6.41) suggests that the FIR filter can be realized as a cascade of a simple FIR filter with an Infinite Impulse Response (IIR) filter [87]. The system function is $1-z^N$ and the IIR portion of the network consists of a parallel combination of N-complex first order systems with poles at $Z = \text{Exp}[j2\pi m/N]$. Since the first order systems have poles

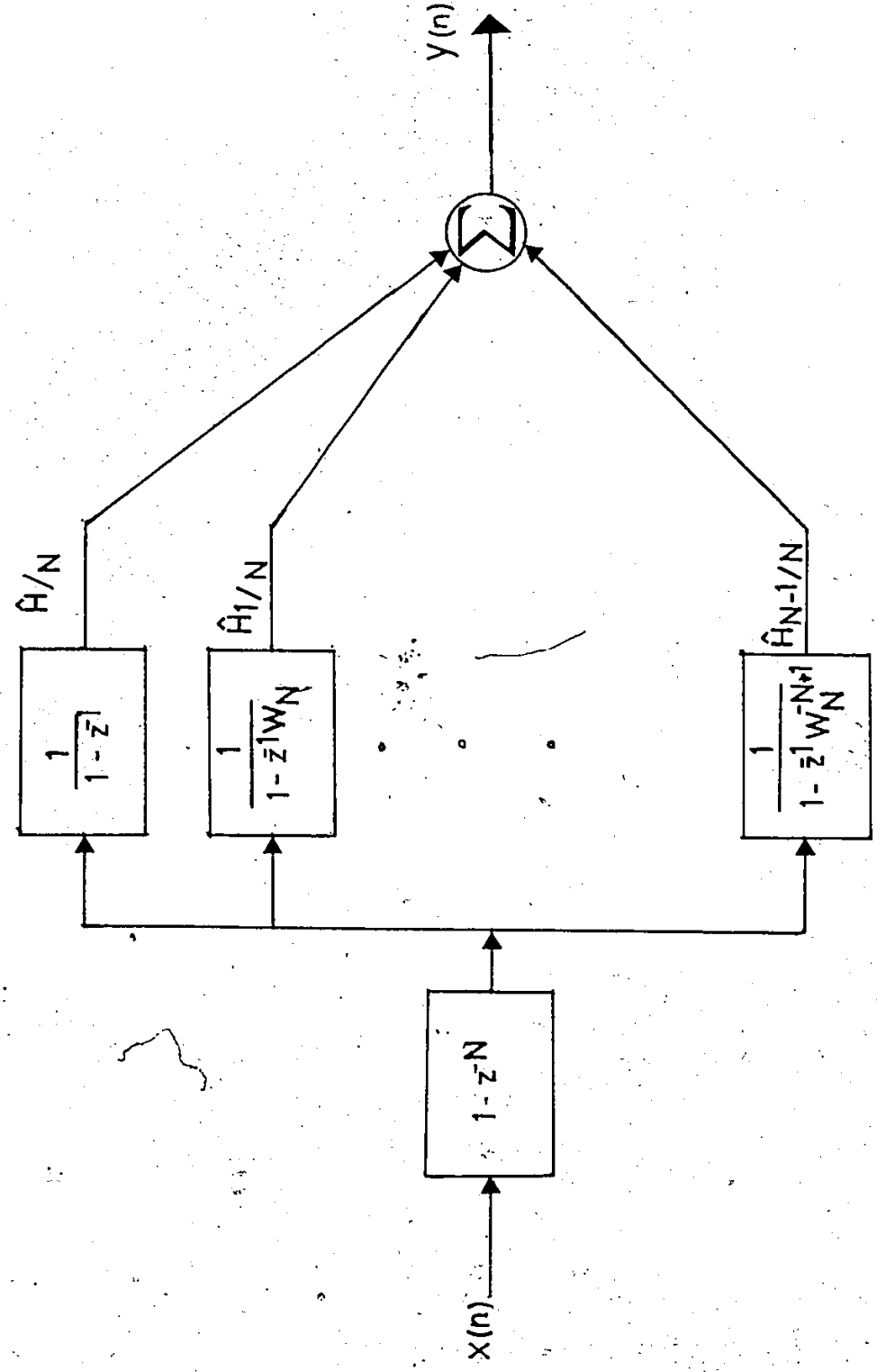


Fig. 6.6 Frequency sampling structure for implementing a recursive FIR system.

exactly on the unit circle, a question of stability arises for such types of filters in the complex field. The CNTT can be used to design a filter specified by a set of first-order difference equations over finite rings. The FIR filters developed in [29], over a finite ring, do not have accumulation of errors in the recursive computations. The question of stability due to particular design configurations does not arise. The results will be correct provided the output sequence satisfies the appropriate dynamic range constraint.

To generalize the filter design, a Complex Number Theoretic z-transform (CNT z-transform) has been introduced [30] as an addition to the results in [29]. It has been shown that the CNT z-transform representation enables one to design a FIR filter over a finite ring by using either recursive or non-recursive structures in a manner analogous to the z-transform method of filter design over the complex field. The CNT z-transform can be computed by an FFT type algorithm. Filter design over a finite complex ring requires input sequences to the filter to be complex integer values and the dynamic range constraint has to be satisfied according to (6.36). Such complex integer valued sequences can be achieved by proper scaling and quantization of the input sequences. The quantization and dynamic range constraint are inherent requirements in representing numbers in a digital computer since only finite length registers can be assigned to represent numbers.

In [30] the Complex Number Theoretic z-transform has been defined and using the CNT z-transform, a recursive realization of FIR filter structure has been proposed for uniformly spaced frequency samples around

the unit circle. In this work, we present an extension of the above technique for the realization of recursive FIR filter for non-uniformly spaced frequency samples around the unit circle, which we will refer here as the generalized FIR filter structure. We will also show that the MQRNS is the most suitable candidate for the implementation of such filters.

In order to develop the generalized FIR filter structure, let us bring out the definition of the CNT z-transform in the following section.

6.10 Complex Number Theoretic Z-Transform

Let q be a product of primes $q = \prod_{i=1}^{L} q_i$, and let $\hat{i}$ denote a solution of the equation $x^2 + 1 \equiv -1 \bmod q$. Let Rq be the ring of residues of integers mod q, where

$$Rq^2 = \{s + \hat{i}t | s, t \in Rq\} \tag{6.43}$$

has q^2 elements and the binary operation of addition, subtraction and multiplication between the elements of Rq^2 follow the rules of complex arithmetic.

Let α be a primitive N-th root of unity in Rq

and

$$S \in \{1, \alpha, \ldots . \alpha^{N-1}\} \tag{6.44}$$

Then the N-point CNT z-transform over Rq^2

$$H(z) = \sum_{n=0}^{N-1} h(n)\, z^n \tag{6.45}$$

and its inverse

$$h(n) = N^{-1} \sum_{Z \in S} H(Z)\, z^{-n} \tag{6.46}$$

Since N and q are relatively prime, $N \in R_q$. The structure of the CNT z-transform and the complex field z-transform (6.40) are identical. The circular convolution property, Parseval's theorem and the other general properties of the CNT z-transform are brought out clearly in [30].

It has been proved in [30] that every CNT z-transform has a unique representation of the form

$$H(z) = (1 - z^N) \sum_{m=0}^{N-1} a_m / 1 - \alpha^m z \tag{6.47}$$

where

$$a_m = N^{-1} H(z)\big|_{Z=\alpha^{-m}} = N^{-1} \sum_{n=0}^{N-1} h(n)\, \alpha^{-mn}$$

The structure of this equation is identical to the structure of equation (6.41,6.42). The resulting filter structure is shown in Fig. 6.7 and has uniformly spaced frequency samples around the unit circle with first order difference equations. This filter structure can be implemented using the QRNS and MQRNS in the following section.

6.11 Implementation of Recursive FIR Filter Structure Using the QRNS and MQRNS

6.11.1 In the Quadratic Residue Number System

$$h(n) = x(n) + \hat{i}y(n) \quad n \in \{0, 1, \ldots N-1\} \tag{6.48}$$

be the impulse response

and

$$\alpha^{mn} \in \{ 1, \alpha \ldots . \alpha^{N-1} \}, \tag{6.49}$$

where α is a primitive N-th root of unity.

Then the element pairs can be computed as follows:

$$h(n) = \left| x(n) + jy(n) \right|_m \qquad h^*(n) = \left| x(n) - jy(n) \right|_m \tag{6.50}$$

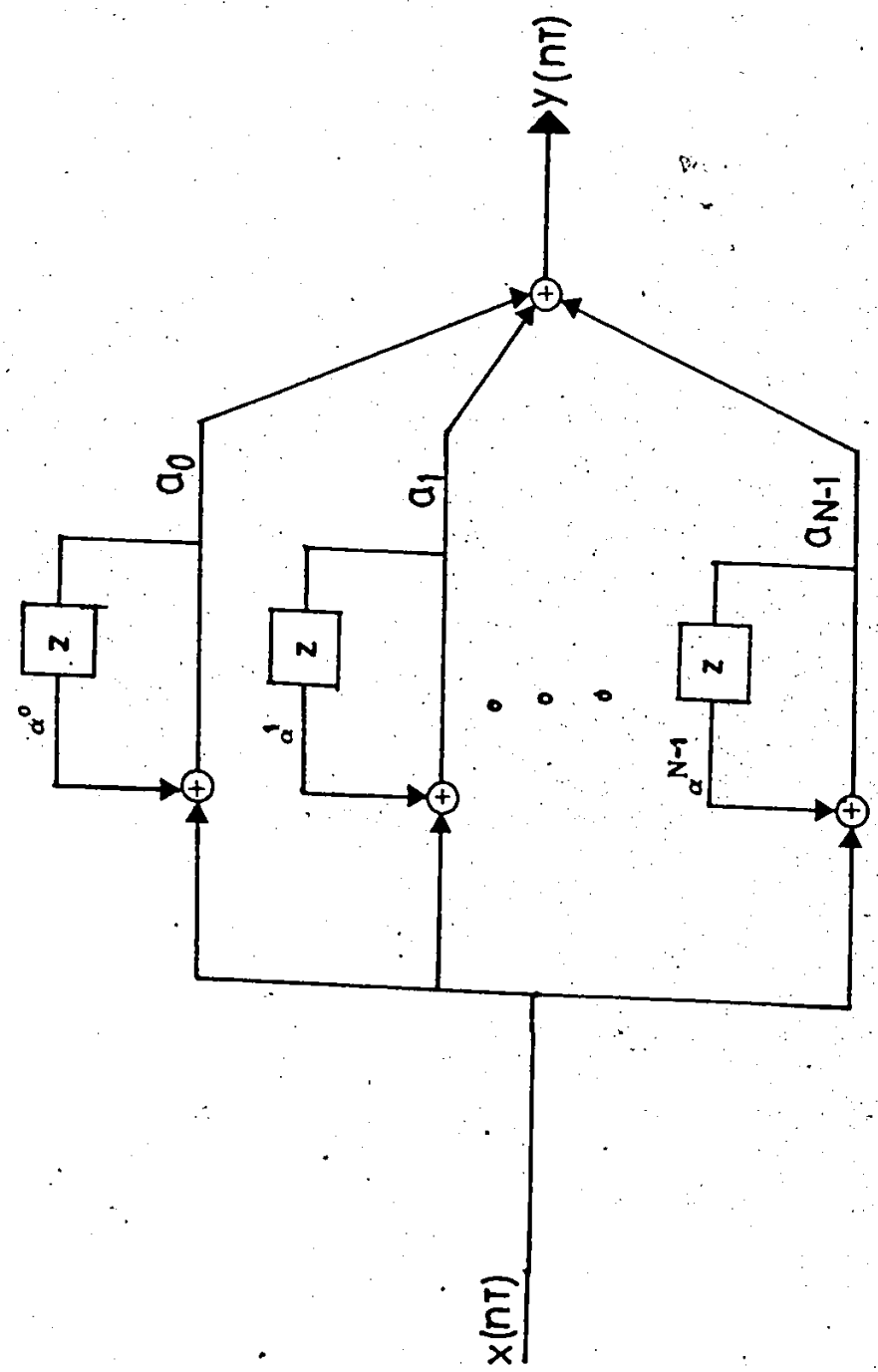


Fig. 6.7 Realization of uniformly spaced recursive FIR filter structure using the CNT z-transform.

$$B(mn) = |\gamma(mn) + j\delta(mn)|_m \quad B(mn) = |\gamma(mn) - j\delta(mn)|_m \tag{6.51}$$

where $j^2 \equiv -1 \bmod m$. m is of the form $4K + 1$ and $\gamma(mn)$ and $\delta(mn)$ are the real and imaginary parts of the elements in (6.49).

The element pairs of (6.47) can be computed using the procedure in [16] as follows:

$$a(t) = |N^{-1} \sum_{n=0}^{N-1} h(n) \cdot B(nt)|_m \text{ for } t = 0,1,\ldots.N-1 \tag{6.52}$$

$$a^*(t) = |N^{-1} \sum_{n=0}^{N-1} h^*(n) \cdot B^*(nt)|_m \text{ for } t = 0,1,\ldots N-1 \tag{6.53}$$

For the computation of (6.52,6.53), the Fast Number Theoretic Transform (FNTT) can be used.

If $b(t)$ and $b^*(t)$ are the outputs of the parallel sections of the filter then the element pairs of the filter output can be written as

$$Q(t) = \sum_{n=0}^{N-1} a(t) \cdot b(t) \tag{6.54}$$

$$Q^*(t) = \sum_{n=0}^{N-1} a^*(t) \cdot b(t) \tag{6.55}$$

Let $Y_R(t)$ and $Y_I(t)$ be the real and imaginary part of the filter output. Then

$$Y_R(t) = |2^{-1} (Q(t) + Q^*(t))|_m \tag{6.56}$$

$$Y_I(t) = |2^{-1} \cdot j^{-1} (Q(t) - Q^*(t))|_m \tag{6.57}$$

This filter can also be implemented using the MQRNS as follows:

6.11.2 In the Modified Quadratic Residue Number System

Let us consider the same impulse response as shown in (6.48) then

$$\alpha^{mn} \varepsilon \{1, \alpha, \ldots \alpha\} \tag{6.58}$$

where α is a primitive N-th root of unity and the powers of α are computed using modulo N operation.

$$h(n) = |x(n) + \hat{j}y(n)|_m \qquad h(n) = |x(n) - \hat{j}y(n)|_m \tag{6.59}$$

$$B(mn) = |\eta(mn) + \hat{j}\zeta(mn)|_m \qquad B^*(mn) = |\eta(mn) - \hat{j}\zeta(mn)|_m \tag{6.60}$$

$$S(mn) = ||\hat{j}^2 + 1| \cdot y(n) \cdot \zeta(mn)|_m \tag{6.61}$$

where $\hat{j}^2 \equiv n \bmod m$, m is of any form except of the form $4K + 1$ and $\eta(mn)$, $\zeta(mn)$ are the real and imaginary parts of the elements in (6.58)

$$a(t) = \left|\left|N^{-1} \sum_{n=0}^{N-1} h(n) \cdot B(nt)\right|_m - S(nt)\right|_m \quad \text{for } t = 0,1,\ldots N-1$$

$$a^*(t) = \left|\left|N^{-1} \sum_{n=0}^{N-1} h(n) \cdot B^*(nt)\right|_m - S(nt)\right|_m \quad \text{for } t = 0,1,\ldots N-1 \tag{6.62}$$

then

$$U(m) = |2^{-1} \cdot \hat{j}^{-1} \cdot (a(t) - a^*(t)|_m \tag{6.63}$$

The element pairs in (6.62) can be computed using the FNTT.

Let the element pairs b(t) and $b^*(t)$ be the output of the parallel section of the filter. The element pairs of the filter

output can be computed as

$$Q(t) = \left|\left| \sum_{t=0}^{N-1} a(t)\, b(t) \right|_m - S(t) \right|_m$$

$$Q(t) = \left|\left| \sum_{t=0}^{N-1} a(t)\, b(t) \right|_m - S(t) \right|_m \qquad (6.64)$$

where

$$S(t) = \left| \hat{j}^2 + 1 \right|_m \cdot U(t) \cdot c(t) \Big|_m$$

and (6.65)

$$c(t) = \left| 2^{-1} \cdot \hat{j}^{-1} \left(b(t) - b^*(t)\right) \right|_m$$

Let $Y_R(t)$ and $Y_I(t)$ be the real and imaginary part of the filter output. Then

$$Y_R(t) = \left| 2^{-1} \cdot \left(Q(t) + Q^*(t)\right) \right|_m$$

$$Y_I(t) = 2^{-1} \cdot \hat{j}^{-1} \left(Q(t) - Q^*(t)\right) \Big|_m \qquad (6.66)$$

In this section, we have restricted the recursive FIR filter structure to the case of the uniformly spaced frequency samples around the unit circle. In the following section, we will discuss an extension of the above technique to non-uniformly spaced frequency samples around the unit circle.

6.2 Generalized Number Theoretic FIR Filter Structure

Let h(n), n = 0, 1,...N-1 be the integer impulse response of a non-recursive filter with CNT z-transform H(z). It can be shown that N-independent values of H(z) can be specified for this filter by writing H(z)

in the following form

$$H(z) = \sum_{k=0}^{N-1} \frac{H_k}{\beta_k} \prod_{i=1}^{N-1} \frac{(1 - z^{-1} b_i)}{(1 - z^{-1} b_i)} \tag{6.67}$$

where

$$\beta_k = \prod_{i=0,\, i\neq k}^{N-1} (1 - \frac{b_i}{b_k})$$

and $\{b_k\}$ are the z-plane position at which $H(b_k) = H_k$. $H(z)$ can be shown to be an (N-1)th order polynomial in z^{-1}. Thus the design of a non-recursive filter can be thought of in terms of deriving suitable values for $\{b_k\}$ and H_k.

Here

$$b_k = \alpha^k \tag{6.68}$$

where the indices k vary non-uniformly.

In this case of non-uniformly spaced samples, the CNT z-transform can be manipulated as shown in (6.69) [32].

$$H(z) = \sum_{k=0}^{N-1} a_k \frac{\sum_{i=0}^{N-1} b_i z^{-i}}{(1 - z^{-1} \alpha^k)} \tag{6.69}$$

where $a_k = H_k/\beta_k$

The internal summation in (6.69) comes from expanding the product in the numerator of (6.67). The resulting filter structure is shown in Fig. 6.8. The internal summation is realized as a non-recursive filter,

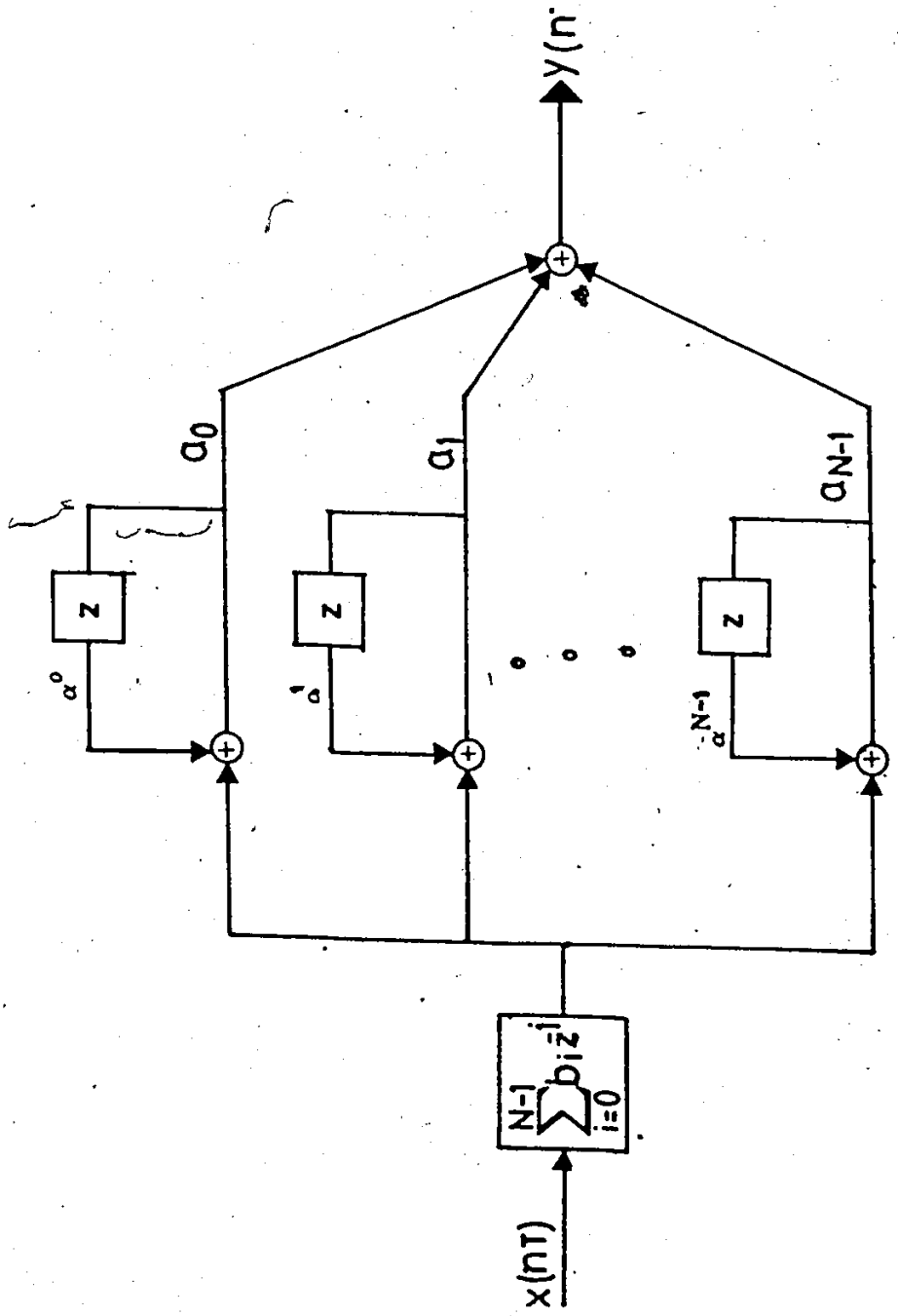


Fig. 6.8 Realization of non-uniformly spaced recursive FIR filters using the CNT z-transform.

whose output is fed into N-parallel channels. The output of the parallel channels is multiplied by the terms 'a_k'' and summed to give the filter output. The non-recursive and recursive structures can be implemented using the QRNS and MQRNS. In implementing the entire filter structure, the computation of 'a_k''' involves division. We cannot employ either the QRNS or MQRNS for the implementation of this filter structure directly. The reason for this is that some of the ring elements we need to use may not have multiplicative inverses. A viable approach in implementing the filter structure is as follows:

1. The non-recursive and recursive portions of the filter structure can be implemented using either the QRNS or MQRNS.
2. The element pairs of the numerator and denominator of a_k can be converted into complex representation.
3. Compute the multiplicative inverse of complex numbers (denominator of a_k).

For primes of the form 4K + 1, some of the complex numbers do not have multiplicative inverses because the structure is defined in the residue ring. For primes of the form 4K + 3, all the complex numbers will have multiplicative inverses because the complex numbers are defined over a second degree extension field. However, we show in Appendix A that the structure of the complex numbers for the primes of the form 4K + 1 is a ring.

The generalized FIR filter structure shown in Fig. 6.8 is termed the polynomial interpolation structure or Lagrange structure [83]. This type of frequency sampling structure in general requires more multiplications

and delays than either direct or cascade forms. If we employ the usual complex binary arithmetic in implementing this filter structure, the throughput of the filter will be low, and the filter is sensitive to quantization effects. If we employ the QRNS arithmetic, we can achieve a high throughput rate without any quantization effects. It results in the most efficient realization compared to the conventional RNS or the usual complex binary arithmetic implementation.

6.13 Summary

In this chapter, the computation of the CNTT using the Quadratic and Modified Quadratic Residue Number Systems has been discussed. The computation of the Cyclic Convolution using the QRNS and MQRNS has been presented. As an example, we have used the MQRNS arithmetic to convolve a random input sequence with a linear phase bandpass filter. We have shown that the implementation of CNTT butterfly structures using either the QRNS and MQRNS requires almost the same hardware, and both generate a savings of about 33% over the hardware required by the conventional RNS structure. The generalized FIR filter structure has been derived for non-uniformly spaced frequency samples. It has been shown that for implementing such structures, the MQRNS is the only technique that can be employed.

Chapter VII

CONCLUSION

The primary aim and objective of this dissertation is the development of algorithms for processing complex data using the Quadratic Residue Number Systems. Algorithms have been developed for both the direct and indirect implementation of Finite Impulse Response (FIR) filters and the implementation of Infinite Impulse Response (IIR) filters. The major results, contributions and conclusions may be summarized as follows:

7.1 Modified Quadratic Residue Number System

The Quadratic Residue Number System (QRNS) has been extended to allow moduli of any form. The resulting number system has been termed the Modified Quadratic Residue Number System (MQRNS). Complex multiplication in the MQRNS requires an additional multiplication compared to the QRNS. In order to offer a complete choice of moduli, the MQRNS has been offered as an alternative to the QRNS. The computation of element pairs using a single multiplexed (complement/multiplex) ROM is faster using the MQRNS than using the QRNS, because an additional operation of subtraction has to be performed in the case of the QRNS.

Residue to binary interfacing techniques have been developed for implementing the QRNS and MQRNS in a parallel processor. In these techniques, a recently developed fractional method based on the Chinese Remainder Theorem (CRT), has been used because it offers high speed and low hardware cost.

7.2 Non-Recursive and Recursive Digital Filters

The QRNS and its modification, the MQRNS, have been used to implement multiplication intensive complex recursive and non-recursive digital filters with considerable savings in hardware. The Peled and Liu bit-slice architecture has been adapted for both non-recursive and recursive digital filters, and suitable hardware structures for both QRNS and MQRNS implementations have been presented. In the case of the recursive filter, a second order section has been considered using a bit-slice architecture. In the case of non-recursive digital filters, a structure which can handle a sequence length of 64 has been presented.

The indirect implementation of FIR filters using Number Theoretic Transform techniques provides improvement in computational speed for large impulse sequence lengths. The computation of Complex Number Theoretic Transforms (CNTTs) and their adaption to performing cyclic convolution in the QRNS and MQRNS have been discussed. As an example, we have used MQRNS arithmetic to convolve a random input sequence with a linear phase bandpass digital filter.

The Basic Computational Unit (BCU) to implement the CNTT, using the FNTT algorithm, is a radix-2 butterfly. The implementation of a radix-2 butterfly using PROMs for a pipeline FNTT in the QRNS and MQRNS has been discussed.

The implementation of recursive FIR filter structures for uniformly spaced frequency samples has been considered, using the QRNS and MQRNS. The implementation of the generalized recursive FIR digital filter structure using the MQRNS has also been discussed. It has been shown that the QRNS can not be used in such a filter structure. The implementation of the

QRNS and MQRNS has been presented using VLSI modules presented for RNS implementation of digital signal processing operations.

We have observed the following facts in implementing the recursive and non-recursive digital filters using the QRNS and MQRNS.

1. Both the QRNS and MQRNS require almost the same amount of hardware in implementing a direct FIR filter structure and give almost the same data rate.
2. It has been observed that in implementing the bit-slice architecture both FIR and recursive filter require either 50% or 75% of the hardware required by the conventional RNS filters for the QRNS and MQRNS respectively.
3. It has been shown that the implementation of the CNTT butterfly structure using the QRNS and MQRNS require almost the same amount of hardware and generate a savings of about 33% over the hardware required by the conventional RNS structure.
4. It has been shown that in implementing the generalized recursive FIR filter structure (Lagrange structure) the MQRNS is the only suitable candidate.

The direct FIR filter architectures in the QRNS and MQRNS have been simulated in software. The computation of the Cyclic Convolution in the QRNS and MQRNS have also been simulated. The program listings for the simulation software are detailed in Appendix B.

7.3 The Advantages of the MQRNS

Although the QRNS is, in general, the most efficient realization of all quadratic type systems, only requiring 2 binary operations for a complex multiplication, the MQRNS total solution can, in certain cases, be almost as efficient as the QRNS total solution. This is due to two facts:

1. The element pairs can be computed by using one multiplex ROM.
2. The efficient method proposed in [4] for the decoding is only applicable to the MQRNS structure.

REFERENCES

1. G.A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," IEEE Trans. on Computers, Vol. C-27, No. 4, April 1978, pp. 325-336.

2. W.K. Jenkins and B.J. Leon, "The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters," IEEE Trans. Circuits, Syst., Vol. CAS-24, pp. 191-201, April 1977.

3. F.J. Taylor and A.S. Ramnarayanan, "An Efficient Residue-to-Decimal Converter," IEEE Trans. on Circuits and Syst., Vol. CAS-28, Dec. 1981, pp. 1164-1169.

4. M.A. Soderstrand, C. Vernia and J.H. Chang, "An Improved Residue Number System Digital-to-Analog Converter," IEEE Trans. on Circuits and System, Vol. CAS-30, No. 12, Dec. 1983, pp. 903-907.

5. A.Z. Baraniecka and G.A. Jullien, "On Decoding Techniques for Residue Number System Realizations of Digital Signal Processing Hardware," IEEE Trans. on Circuits and Syst., Vol. CAS-25, No. 11, Nov. 1978, pp. 935-936.

6. A. Savoboda and M. Valach, "Decimal Arithmetic Unit," Stroje Na Zpracovani, Vol. 8, Nakl. CSAU, Praha, 1962.

7. H.L. Harner, "The Residue Number System," IRE Trans. on Electronic Computers, Vol. EC-8, June 1959, pp. 140-147.

8. J.M. Pollard, "Implementation of Number Theoretic Transforms," Electronics Letter, Vol. 12, No. 22, July 1976, pp. 378-379.

9. H. Nussbaumer, "Digital Filtering Using Read-Only Memories," Electronic Letter, Vol. 11, 1976, pp. 294-295.

10. M.A. Soderstrand and F. L. Fields, "Multiplier for Residue-Number Arithmetic Digital Filters," Electronics Letter, Vol-13, No. 6, March 1977.

11. W.K. Jenkins, "Techniques for Residue to Analog Conversion for Residue Encoded Digital Filters," IEEE Trans. on Circuits and Syst., Vol. CAS-25, July 1978, pp. 555-562.

12. A. Peled, and B. Liu, "A New Hardware Realization of Digital Filters," IEEE Trans. on Acoust. Speech, Signal Processing, Vol. ASSP-22, No. 6, Dec. 1974, pp. 456-462.

13. W.K. Jenkins, "Recent Advances in Residue Number Techniques for Recursive Digital Filtering," IEEE Trans. on Acoust. Speech, Signal Processing, Vol. ASSP-27, No. 1, Feb. 1979, pp. 19-30.

14. G.A. Jullien, "Implementation of Multiplication Modulo a Prime Number with Application to Number Theoretic Transform," IEEE Trans. on Computers, Vol. C-29, No. 10, October 1980.

15. R. Krishnan, G.A. Jullien and W.C. Miller, "The Modified Quadratic Residue Number System (MQRNS) For Complex High Speed Signal Processing," submitted for publication in IEEE Transaction on Circuits and Systems.

16. R. Krishnan, G.A. Jullien and W.C. Miller, "Complex Digital Signal Processing Using Quadratic Residue Number Systems," submitted for publication in IEEE Trans. Acoust., Speech, Signal Processing.

17. R. Krishnan, G.A. Jullien and W.C. Miller, "Implementation of Complex Number Theoretic Transform Using the Quadratic Residue Number Systems," submitted for publication in IEEE Transaction on Circuits and Systems.

18. M.A. Soderstrand, G.D. Poe, "Applications of Quadratic-Like Complex Residue Number System Arithmetic to Ultrasonics," IEEE Int. Conf. on ASSP, Vol. 2, March 1984, pp. 28A.5.1-28A.5.4.

19. H.J. Nussbaumer, "Complex Convolution Via Fermat Number Transforms," IBM J. Res. Develop., Vol. 20, May 1976, pp. 282-284.

20. S.H. Leung, "Application of Residue Number Systems to Complex Digital Filters," Proceedings of the Fifteenth Asilomar Conference on Circuits and Systems and Computers, Pacific Grove, CA, Nov. 1981, pp. 70-74.

21. J.V. Krogmeier and W.K. Jenkins, "Error Detection and Correction in Quadratic Residue Number Systems," 26th Midwest Symposium on Circuits and Systems, Puebla, Mexico, August 1983.

22. W.K. Jenkins, "Quadratic Modular Number Codes for Complex Digital Signal Processing," IEEE Int. Symposium on Circuits and Syst., Montreal, Canada, May 1984, pp.

23. A.Z. Baraniecka and G.A. Jullien, "Residue Number System Implementations of Number Theoretic Transform in Complex Residue Ring," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. Assp-28,

24. I.S. Reed and T.K. Truong, "The Use of Finite Fields to Compute Convolutions," IEEE Trans. Inform. Theory, Vol. IT-21, March 1975, pp. 208-213.

25. I.S. Reed and T.K. Truong, "Complex Integer Convolutions over a Direct Sum of Galois Fields," IEEE Trans. Inform. Theory, Vol. IT-21, No. 6, Nov. 1975, No. 3, June 1978, pp. 285-291.

26. C.M. Rader, "Discrete Convolution via Mersenne Transforms," IEEE Trans. Comput., Vol. C-21, Dec. 1972, pp. 1269-1273.

27. R.C. Agarwal and C.S. Burrus, "Fast Convolution Using Fermet Number Transform with Application to Digital Filtering," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-22, No. 2, April 1974, pp. 87-97.

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

29. H. Murakami and I.S. Reed, "Recursive Realization of Finite Impulse Filters Using Finite Field Arithmetic," IEEE Trans. Inform. Theory, Vol. IT-23, No. 2, March 1977, pp. 232-242.

30. H. Murakami, I.S. Reed and A. Arcese, "Recursive FIR Digital Filter Design using a Z-transform on a Finite Ring," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-31, No. 5, October 1983, pp. 1155-1163.

31. F.J. Taylor, "Large Moduli Multipliers for Signal Processing," IEEE Trans. on Circuits Syst., Vol. CAS-28, pp. 732-736, July 1981.

32. L.R. Rabiner, B. Gold and C.A. McGonegal, "An Approach to the Approximation Problem for Non-Recursive Digital Filters," IEEE Trans. on Audio, Electroacout., Vol. AU-18, June 1970, pp. 83-106.

33. M.A. Soderstrand, "A High Speed Low Cost, Recursive Digital Filter Using Residue Number Arithmetic," Proc. IEEE, Vol. 65, No. 7, July 1977, pp. 1065-1067.

34. B.D. Tseng, G.A. Jullied and W.C. Miller, "Implementation of FFT Structures Using the Residue Number System," IEEE Trans. on Computers, Vol. C-28, No. 11, Nov. 1979, pp. 831-845.

35. F.J. Taylor and C.H. Huang, "An Autoscale Residue Multiplier," IEEE Trans. on Computers, Vol. C-31, No. 4, April 1982, pp. 321-325.

36. D.P. Agarwal, "Modulo $2^n + 1$ Arithmetic Logic," IEE Journal Electronic Circuits and Systems, Vol. 2, No. 6, pp. 186-188.

37. D. Mandelbaum, "Error Correction in Residue Arithmetic," IEEE Trans. on Computers, Vol. C-21, No. 6, June 1978, pp. 538-545.

38. M.Hetzel and W.K. Jenkins, "Redundant Residue Number Systems for Error Detection and Correction in Digital Filters," IEEE Trans. on Acoustic, Speech, Signal Processing, Vol. ASSP-29, No. 5, October 1980.

39. A.Z. Baraniecka and G.A. Jullien, "Quantization Error and Limit Cycle Analysis in Residue Number System Coded Recursive Filters," Proc. 1982 IEEE Intern. Conf. on Acoustic, Speech, Signal Processing, May 1982.

40. F.J. Taylor, "A VLSI Residue Arithmetic Multiplier," IEEE Trans. on Computers, Vol. C-31, No. 6, June 1982, pp. 540-546.

41. F. Barsi and P. Maestrini, "Error Correcting Properties of Redundant Residue Number Systems," IEEE Trans. on Computers, Vol. C-23, No. 3, March 1973, pp. 307-315.

42. C.H. Huang and F.J. Taylor, "High Speed DFT's Using Residue Numbers", IEEE Int. Conf. on Acoustic, Speech, Signal Processing, Colorado, April 1980, pp. 238-242.

43. M.J. Corinthios, "The Design of a Class of Fast Forier Transform Computers," IEEE Trans. on Computers, Vol. C-20, No. 60, June 1971, pp. 617-623.

44. S.D. Pezaris, "A 40ns 17 x 17 Array Multiplier," IEEE Trans. on Computers, Vol. C-20, pp. 442-447, April 1971.

45. L.R. Rabiner, J.H. McClellan and T.W. Parks, "FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximation," Proc. IEEE, Vol. 63, April 1975, pp. 595-610.

46. G.A. Jullien, W.C. Miller, J.J. Soltis, A.Z. Baraniecka and B. Tseng, "Hardware Realization of Digital Signal Processing Elements Using the Residue Number System," Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing, Hartford, April 1977.

47. A.Z. Baraniecka and G.A. Jullien, "Hardware Implementation of Convolution Using Number Theoretic Transforms," Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Washington, D.C., April 1979.

48. B. Liu, "Effects of Finite Wordlength on the Accuracy of Digital Filters - A Review," IEEE Trans. on Circuit Theory (Special Issues on Active and Digital Networks), Vol. CT-18, Nov. 1971, pp. 670-677.

49. T.A. Claasen, W. Mecklenbrauker and J.B. Peck, "Effects of Quantization and Overflow in Recursive Digital Filters," IEEE Trans. on Acoust. Speech, Signal Proc., Vol. ASSP-24, December 1976, pp. 517-529.

50. D.K. Banerji, "A Novel Implementation of Method for Addition and Subtraction in Residue Number System," IEEE Trans. on Computers, Vol. C-23, No. 1, January 1974, pp. 106-109.

51. J.W. Cooley and J.W. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," Math. Comp., V. 19, 1965, pp. 297-301.

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

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

54. E. Dubois and A.N. Venetsanapoulos, "The Discrete Fourier Transform Over Finite Rings with Application to Fast Convolution," IEEE Trans. on Computers, Vol. C-17, No. 7, July 1978, pp. 586-593.

55. J.H. McClellan, "Hardware Realization of a Fermat Number Transform," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-24, June 1976, pp. 216-225.

56. H. Nussbaumer, "Digital Filtering Using Complex Mersenne Transforms," IBM J. Res. Develop., Vol. 20, Sept. 1976, pp. 498-504.

57. H. Nussbaumer, "Digital Filtering Using Pseudo-Fermat Number Transforms," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-25, Feb. 1977, pp. 79-83.

58. J.F. Kaiser, "Some Practical Considerations in the Realization of Linear Digital Filters," 1965 Proc. 3rd Allerton Conf. on Circuits and System Theory, pp. 621-633.

59. W.K. Jenkins, "Complex Residue Number Arithmetic for High Speed Signal Processing," Electronic Letters, Vol. 16, No. 17, August 1980, pp. 660-661.

60. L.M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-24, No. 5, pp. 356-359.

61. G.K. Dillard, "Recursive Computation of the Discrete Fourier Transform with Application to a Pulse-Doppler Radar System," Comput. & Elect. Engineering, Vol. 1, Pergamon Press 1973, pp. 143-152.

62. J.H. McClellan, T.W. Parks and L.R. Rabiner, "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters," IEEE Trans. Audio, Electroacoust., Vol. AU-21, Dec. 1973, pp. 506-526.

63. K.Y. Liu, I.S. Reed and T.K. Truong, "Fast Number Theoretic Transform for Digital Filtering," Electronic Letters, Vol. 12, Nov. 1976, pp. 644-646.

64. M.C. Vanwormhoud, "On Number Theoretic Fourier Transforms in Residue Class Rings," IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-25, Dec. 1977, pp. 585-586.

65. M.C. Vanwormhoud, "Structural Properties of Complex Residue Rings Applied to Number Theoretic Fourier Transforms," IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-26, Feb. 1978, pp. 99-104.

66. K.Y. Liu, I.S. Reed and T.K. Truong, "Fast Algorithm for Complex Integer Transforms," IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-25, October 1977, pp. 450-452.

67. I.S. Reed and T.K. Troung, "Convolution Over Residue Class of Quadratic Integers," IEEE Trans. Inform. Theory, Vol. IT-22, July 1976, pp. 468-475.

68. A.V. Oppenheim and C.J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," Proc. of IEEE, Vol. 60, No. 8, August 1972, pp. 957-976.

69. L.B. Jackson, "Round-off Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," IEEE Trans. Audio Electroacoust (Special Issue on Digital Filtering), Vol. AU-18, June 1970, pp. 107-122.

70. D.H. Huang and F.J. Taylor, "A Memory Compression Scheme for Modular Arithmetic," IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-27, No. 6, December 1979, pp. 608-611.

71. H.K. Nagpal, G.A. Jullien and W.C. Miller, "Processor Architectures for Two-Dimensional Convolvers Using a Single Multiplexed Computational Element with Finite Field Arithmetic," IEEE Trans. on Computers, Vol. C-32, No. 11, Nov. 1983, pp. 989-999.

72. W.K. Jenkins, "A Highly Efficient Residue-Combinational Architecture for Digital Filters," Proc. IEEE (Lett.) Vol. 66, No. 6, June 1978, pp. 700-702.

73. W.K. Jenkins, "A Standard Computational Element for the VLSI Realization of Digital Processor Using Modular Arithmetic," Proc. 16th Asilomar Conf. Circuits, Syst., and Computers, Pacific Grove, CA, Nov. 1982, pp. 202-206.

74. L.T. Bruton, "Low Sensitivity Digital Lader Filters," IEEE Trans. on Circuits and Systems, Vol. CAS-22, No. 3, March 1975, pp. 168-176.

75. U. Dudley, "Elementary Number Theory," W.H. Freeman and Co., 1969.

76. G.H. Hardy and E.M. Wright, "An Introduction to Theory of Numbers," 4th Ed., Oxford Clarendon Press, 1960.

77. N. McCoy, "Fundamentals of Abstract Algebra," Allyn and Bacon, Inc., 1972.

78. A.Z. Baraniecka, "Digital Filtering Using Number Theoretic Techniques," Ph.D. Thesis, Department of Electrical Engineering, University of Windsor, Windsor, 1980.

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

80. I.M. Vinogradov, "Elements of Number Theory," Dover Publications, Inc.

81. T. Nagell, "Introduction to Number Theory," New York, Chelsea, 1964.

82. C. Mead and L. Conway, "Introduction to VLSI Systems," Addison-Wesley, Reading, MA, 1980.

83. L.R. Rabiner and B. Gold, "Theory and Application of Digital Signal Processing," Englewood Cliffs, N.J., Prentice-Hall, 1975.

84. J.H. McClellan and C.M. Rader, "Number Theory in Digital Signal Processing," Prentice-Hall, Inc., Englewood Cliffs, N.J. 1979.

85. A.M. Neville and J.B. Kennedy, "Basic Statistical Methods for Engineers and Scientists," International Text Book Company, Scranton, Pennsylvania, 1968.

86. A. Peled and B. Liu, "Digital Signal Processing," John Wiley & Sons, 1976.

87. A.V. Oppenheim, "Application of Digital Signal Processing," Englewood Cliffs, N.J., Prentice-Hall, 1978.

88. A.V. Oppenheim and R.W. Schafer, "Digital Signal Processing," Englewood Cliffs, N.J., Prentice-Hall, 1975.

89. M.A. Bayoumi, G.A. Jullien and W.C. Miller, "Models for VLSI Implementation of Residue Number System Arithmetic Modules," Proc. of IEEE VI Symp. on Comp. Arithmetic, pp. 174-182, June 1983.

90. L. Gall, "Classical Galois Theory," Chelsea Publishing Co., New York, 1971.

91. C.A. Mead and M. Reon, "Cost and Performance of VLSI Computing Structures," IEEE Trans. on Electron Devices, Vol. ED-26, No. 4, April 1979, pp. 533-540.

92. L.M. Terman, "MOSFET Memory Circuits," Invited Paper, IEEE Proc. Vol.-59, No. 7, July 1971, pp. 1044-1058.

93. P. Pleshko and L.M. Terman, "An Investigation of Potential of MOS Transistor Memories," IEEE Trans. on Electron. Computers, Vol. EC-15, Aug. 1966, pp. 423-427.

94. R.I. Garesh, T. Kurosawa, and T. Yaito, "A 150 ns Associative Memory Using Integrated MOS Transistors," Inter. Solid State Circuits Conf., Digest of Tech. Papers, Feb. 1966, pp. 104-105.

95. Bipolar LSI Data Book, Monolithic Memories, Sunnyvale, CA, 1981.

Appendix A

COMPUTATION OF MULTIPLICATIVE INVERSE FOR COMPLEX NUMBERS DEFINED OVER QUADRATIC RINGS

We will show that the structure of the complex numbers for the primes of the form $4K + 1$ is a ring as follows:

Let $a + \hat{i}b$ be a complex number generated from the Gaussian integers. where $\hat{i} = \sqrt{-1}$

The multiplicative inverse can be computed as follows:

$$(a + \hat{i}b)^{-1} = 1/(a + \hat{i}b) \tag{A.1}$$

Multiply both numerator and denominator by the complex conjugate of $a + \hat{i}b$ [78],

then

$$(a + \hat{i}b)^{-1} = (a - \hat{i}b)/(|a^2|_m + |b^2|_m \tag{A.2}$$

$$= |k(a - \hat{i}b)|_m \quad \text{where } K = \left| \frac{1}{|a^2|_m + |b^2|_m} \right|_m$$

Therefore

$$(a + \hat{i}b)^{-1} = |k \cdot a|_m - \hat{i}|k \cdot b|_m \tag{A.3}$$

Let us consider the denominator term in (A.2)

$$\text{i.e. } |a^2|_m + |b^2|_m$$

which can be represented as $X^2 + 1$ or $1 + X^2$ for some particular subset of elements $a + \hat{i}b$.

If the monic quadratic $X^2 + 1 = 0$ has solution in the ring, then a multiplicative inverse does not exist, and if the monic quadratic has no solution in the ring, then a multiplicative inverse exists. This fact can be established by using [19, theorem 80]. For primes of the form 4K + 1, -1 is a quadratic residue and for primes of the form 4K + 3, -1 is a quadratic non-residue. For some values of a and b the denominator term in (A. 2) will become zero and so we cannot have the multiplicative inverse of such complex numbers generated using the Gaussian integers modulo m where m is of the form 4K + 1. For primes of the form 4K + 3, all the complex numbers will have multiplicative inverse.

As an example, Table 6.13 and 6.14 show the multiplicative inverses of all the possible complex numbers generated using the Gaussian integers modulo 7 and modulo 5 respectively.

Table A1

Multiplicative Inverse of Complex Numbers in Modulo 7

X	X^{-1}	X	X^{-1}	X	X^{-1}	X	X^{-1}
$0 + \hat{i}$	$0 + \hat{i}6$	$2 + \hat{i}0$	$4 + \hat{i}0$	$4 + \hat{i}0$	$2 + \hat{i}0$	$6 + \hat{i}0$	$6 + \hat{i}0$
$0 + \hat{i}2$	$0 + \hat{i}3$	$2 + \hat{i}$	$6 + \hat{i}4$	$4 + \hat{i}$	$6 + \hat{i}2$	$6 + \hat{i}$	$3 + \hat{i}3$
$0 + \hat{i}3$	$0 + \hat{i}2$	$2 + \hat{i}2$	$2 + \hat{i}5$	$4 + \hat{i}2$	$3 + 2\hat{i}$	$6 + \hat{i}2$	$4 + \hat{i}$
$0 + \hat{i}4$	$0 + \hat{i}5$	$2 + \hat{i}3$	$5 + \hat{i}3$	$4 + \hat{i}3$	$1 + \hat{i}$	$6 + \hat{i}3$	$2 + \hat{i}6$
$0 + \hat{i}5$	$0 + \hat{i}4$	$2 + \hat{i}4$	$5 + \hat{i}4$	$4 + \hat{i}4$	$1 + \hat{i}6$	$6 + \hat{i}4$	$2 + \hat{i}$
$0 + \hat{i}6$	$0 + \hat{i}$	$2 + \hat{i}5$	$2 + \hat{i}2$	$4 + \hat{i}5$	$3 + \hat{i}5$	$6 + \hat{i}5$	$4 + \hat{i}6$
		$2 + \hat{i}6$	$6 + \hat{i}3$	$4 + \hat{i}6$	$6 + \hat{i}5$	$6 + \hat{i}6$	$4 + \hat{i}4$
$1 + \hat{i}0$	$1 + \hat{i}0$	$3 + \hat{i}0$	$5 + \hat{i}0$	$5 + \hat{i}0$	$3 + \hat{i}0$		
$1 + \hat{i}$	$4 + \hat{i}3$	$3 + \hat{i}$	$1 + \hat{i}2$	$5 + \hat{i}$	$1 + \hat{i}4$		
$1 + \hat{i}2$	$3 + \hat{i}$	$3 + \hat{i}2$	$4 + \hat{i}2$	$5 + \hat{i}2$	$5 + \hat{i}5$		
$1 + \hat{i}3$	$5 + \hat{i}6$	$3 + \hat{i}3$	$6 + \hat{i}$	$5 + \hat{i}3$	$2 + \hat{i}3$		
$1 + \hat{i}4$	$5 + \hat{i}$	$3 + \hat{i}4$	$6 + \hat{i}6$	$5 + \hat{i}4$	$2 + \hat{i}4$		
$1 + \hat{i}5$	$3 + \hat{i}6$	$3 + \hat{i}5$	$4 + \hat{i}5$	$5 + \hat{i}5$	$5 + \hat{i}2$		
$1 + \hat{i}6$	$4 + \hat{i}4$	$3 + \hat{i}6$	$1 + \hat{i}5$	$5 + \hat{i}6$	$1 + \hat{i}3$		

Table A2

Multiplicative Inverse of Complex Numbers In Modulo 5

X	X^{-1}	X	X^{-1}	X	X^{-1}	X	X^{-1}	X	X^{-1}
$0 + \hat{i}$	$0 + \hat{i}4$	$1 + \hat{i}0$	$1 + \hat{i}0$	$2 + \hat{i}0$	$3 + \hat{i}0$	$3 + \hat{i}0$	$2 + \hat{i}0$	$4 + \hat{i}0$	$4 + \hat{i}0$
$0 + \hat{i}$	$0 + \hat{i}4$	$1 + \hat{i}$	$3 + \hat{i}2$	$2 + \hat{i}$	*	$3 + \hat{i}$	*	$4 + \hat{i}$	$2 + \hat{i}2$
$0 + \hat{i}2$	$0 + \hat{i}2$	$1 + \hat{i}2$	*	$2 + \hat{i}2$	$4 + \hat{i}$	$3 + \hat{i}2$	$1 + \hat{i}$	$4 + \hat{i}2$	*
$0 + \hat{i}3$	$0 + \hat{i}3$	$\hat{i} + \hat{i}3$	*	$2 + \hat{i}3$	$4 + \hat{i}4$	$3 + \hat{i}3$	$1 + \hat{i}4$	$4 + \hat{i}3$	, *
$0 + \hat{i}4$	$0 + \hat{i}$	$1 + \hat{i}4$	$3.\hat{i}3$	$2 + \hat{i}4$	*	$3 + \hat{i}4$	*	$4 + \hat{i}4$	$2 + \hat{i}3$

* Indicates that there is no multiplicative inverse for the particular complex number modulo 5.

Appendix B

SOFTWARE DETAILS

The complex digital filtering algorithms developed in this thesis have been simulated in SEL and NOVA mini-computers. The programs listings are given in the following pages.

The first program is a subroutine to compute the residues. The second and third programs simulate the direct implementation of FIR filters using the QRNS and MQRNS respectively. The fourth program computes the generator of order N, required for the CNTT, for Mersenne primes. The generator for other types of the primes, can be computed using the program in [78].

Fifth and sixth programs simulate indirect filtering of complex data using complex NTT in the QRNS and MQRNS respectively.

```
C     ************************************************************
C     FILE NAME:-RESIDUE
C     ************************************************************
C
      SUBROUTINE RESIDUE(MEENU,MD,ID)
      IMPLICIT REAL*8 (A-Z)
      INTEGER*8 MD,MEENU,ID,KX,KY,KZ,ISD,I
      ISD=ID*MD
      KX=MEENU/MD
      IF(KX.LE.0) KX=-KX
      ST=KX
      PY=ID
      FX=ST/PY
      IF(FX.LE.1) GO TO 100
      KY=KX/ID
      KZ=(KX-(ID*KY))
      IF(MEENU.LT.0) GO TO 200
      DO 20 I=1,KY
      MEENU=MEENU-ISD
 20   CONTINUE
      DO 21 I=1,KZ
      MEENU=MEENU-MD
      IF(MEENU.LT.MD) GO TO 90
 21   CONTINUE
 200  DO 201 I=1,KY
      MEENU=MEENU+ISD
 201  CONTINUE
      DO 202 I=1,KZ
      MEENU=MEENU+MD
      IF(MEENU.GT.-MD) GO TO 90
 202  CONTINUE
 100  IF(MEENU.LT.0) GO TO 500
      DO 22 I=1,KX
      MEENU=MEENU-MD
      IF(MEENU.LT.MD) GO TO 90
 22   CONTINUE
 500  DO 205 I=1,KX
      MEENU=MEENU+MD
      IF(MEENU.GT.-MD) GO TO 90
 205  CONTINUE
C 90  WRITE(6,91) MEENU
   90 MEENU=MEENU
C 91  FORMAT(5X,'MEENU=',I25)
      RETURN
      END
```

```
$JOB ARUL    KRIS,RAJI
$OPTION 1 2 3 4 5 17
$EXECUTE FORTRAN
C         FILE NAME:- DIRECT FIR FILTER IMPLEMENTATION
          IMPLICIT REAL*8 (A-Z)
          INTEGER*2 I,J
          INTEGER*8 MD,N,ID,IR,IR1,ITEMP,JTEMP,IP1,LD,MD2
          INTEGER*8 IX1(32),IY1(32),IX2(32),IY2(32),IZR(32),IZI(32)
          INTEGER*8 IX(32),IY(32),JX(32),JY(32),ISUM1(32),ISUM2(32)
          INTEGER*8 IYR(32),IYI(32),ISR(32),ISI(32),IFIRR(32),IFIRI(32)
          DATA IX1/16*0,16*1/, IY1/16*0,16*1/
          DATA IX2/16*0,16*1/, IY2/16*0,16*0/
          TYPE *,'ENTER MODULUS, MD,N,ID....'
          ACCEPT *, MD,N,ID
          IR=0
1         IR=IR+1
          IR1=IR**2
          IR1=IR1+1
          CALL RESIDUE(IR1,MD,ID)
          IF(IR1.NE.0) GO TO 1
          TYPE *, IR
5         CONTINUE
          DO 10 I=1,2*N
          ISUM1(0)=0
          ISUM2(0)=0
          DO 11 J=1,2*N
          IF((I+1-J).LE.0) GO TO 20
          IX(I+1-J)=IX1(I+1-J)+IR*(IY1(I+1-J))
          JX(I+1-J)=IX1(I+1-J)-IR*(IY1(I+1-J))
          IY(J)=IX2(J)+IR*IY2(J)
          JY(J)=IX2(J)-IR*IY2(J)
          IZR(J)=IX(I+1-J)*IY(J)
          IZI(J)=JX(I+1-J)*JY(J)
          ISUM1(J)=ISUM1(J-1)+IZR(J)
          ISUM2(J)=ISUM2(J-1)+IZI(J)
          CALL RESIDUE(ISUM1(J),MD,ID)
          CALL RESIDUE(ISUM2(J),MD,ID)
          ITEMP=ISUM1(J)
          JTEMP=ISUM2(J)
11        CONTINUE
20        ISR(I)=ITEMP
          ISI(I)=JTEMP
10        CONTINUE
          IP1=MD-2
          LD=2
          DO 15 I=1,IP1-1
          MD2=IR1*IR
          CALL RESIDUE(MD2,MD,ID)
15        CONTINUE
          DO 22 I=1,2*N
          IYR(I)=ISR(I)+ISI(I)
          IYI(I)=ISR(I)-ISI(I)
          IYR(I)=IYR(I)*LD
          IYI(I)=IYI(I)*LD*IR
          TYPE *, IYR(I),IYI(I)
```

```
          CALL RESIDUE(IYR(I),MD,ID)
          CALL RESIDUE(IYI(I),MD,ID)
          IF(IYR(I).LT.0) IYR(I)=IYR(I)+MD
          IF(IYI(I).LT.0) IYI(I)=IYI(I)+MD
          IF(IYR(I).EQ.0.OR.IYI(I).EQ.0) GO TO 22
          IFIRR(I)=IYR(I)
          IFIRI(I)=IYI(I)
          WRITE(6,23) I,IFIRR(I),I,IFIRI(I)
23       FORMAT(5X,'IFIRR(',I3,')..',I12,1X,'IFIRI(',I3,')..',I12)
22        CONTINUE
          STOP
          END
          INCLUDE RESIDUE
$EXECUTE  CATALOG
A4 6=UT
BUILD FIR.SW,NOM
$EOJ
```

```
$JOB ARUL     KRIS,RAJI          SLOF=DUMMY
$OPTION    2 3 4 5 17
$EXECUTE FORTRAN
C         FILE NAME:-ALPHA FOR MERSENNE PRIMES
C         ##################################################################
C         TO CALCULATE THE UNITY ROOT OF ALPHA FOR MERSENNE PRIMES
C         THIS PROGRAM CAN BE USED TO CALCULATE THE UNITY ROOT FOR THE
C         MERSENNE PRIMES: (2**P)-1 FO P=3,5,7,13,17,19,31
C         ##################################################################
          IMPLICIT REAL*8 (A-Z)
          INTEGER*8 I,IX,JX,JY,JZ,KX,KY,KZ,IY,IZ,IYA,IYB,MD,ID,IXD,JT,K
          INTEGER*8 IDUMMY,JDUMMY,ISD,JSQR,JMSQ1,JMSQ,IRREAL,IIMAGE,MY
          COMPLEX*16 CDUMMY,DCMPLX
          TYPE *,'ENTER MODULUS, MD.....'
          ACCEPT *, MD
          TYPE *,'ENTER VALUE OF IX.....'
          ACCEPT *, IX
          TYPE *,'ENTER LEVEL OF OPERATION, K....'
          ACCEPT *, K
          TYPE *,'ENTER THE VALUE OF ID,....'
          ACCEPT *,ID
C         ****************************************************************
C         TO FIND AN ELEMENT OF ORDER 2**P+1
C                   A=((2)**2)**P-2
C         ****************************************************************
          IYA=2
          DO 1 I=1,IX
          IYA=IYA**2
          IF(IYA.LT.MD) GO TO 600
          CALL RESIDUE(IYA,MD,ID)
 600      WRITE(6,10) IYA
 10       FORMAT(5X,'IYA=',I25)
 1        CONTINUE
C         ****************************************************************
C         TO FIND AN ELEMENT OF ORDER 2**P+1
C                   B=((-3)**2)**P+1
C         ****************************************************************
          IYB=-3
          DO 12 I=1,IX
          IYB=IYB**2
          WRITE(6,6) IYB
 6        FORMAT(5X,'IYB=',I25)
          IF(IYB.LT.MD) GO TO 5
          JX=IYB/MD
          PX=JX
          PD=ID
          XY=PX/PY
          IF(XY.LT.1) GO TO 8
          IXD=ID*MD
          JY=JX/ID
          DO 7 I=1,JY
          IYB=IYB-IXD
 7        CONTINUE
          JZ=(JX-(ID*JY))
          DO 9 I=1,JZ
```

```
          IYB=IYB-MD
          IF(IYB.LT.MD) GO TO 5
 9        CONTINUE
 8        DO 2 I=1,JX
          IYB=IYB-MD
          IF(IYB.LT.MD) GO TO 5
 2        CONTINUE
 5        WRITE(6,4) IYB
 4        FORMAT(5X,'IYB=',I25)
 12       CONTINUE
C         ********************************************************************
C         COMPUTING THE ORDER OF ALPHA FOR NTT
C         ********************************************************************
          DO 11 J=1,K
          JSQR=IYA**2
          JMSQ1=IYB**2
          JMSQ=-JMSQ1
          IDUMMY=JSQR+JMSQ
          JDUMMY=IYA*IYB+IYB*IYA
C         WRITE(6,15) IDUMMY,JDUMMY
 15       FORMAT(5X,'IDUMMY=',I25,1X,'JDUMMY=',I25)
          CALL RESIDUE(IDUMMY,MD,ID)
          IRREAL=IDUMMY
          CALL RESIDUE(JDUMMY,MD,ID)
          IIMAGE=JDUMMY
          WRITE(6,91)   IRREAL,IIMAGE
 91       FORMAT(5X,'IRREAL=',I25,1X,'IIMAGE=',I25)
          IYA=IRREAL
          IYB=IIMAGE
 11       CONTINUE
          STOP
          END
          INCLUDE RESIDUE
$EXECUTE CATALOG
A4 6=UT
BUILD ALP.SV,NOM
$EOJ
```

```
$JOB ARUL     KRIS,RAJI          SLOF=DUMMY
$OPTION 1 2 3 4 5 17
$EXECUTE FORTRAN
C         FILE NAME:-QRNS CONVOLUTION
          IMPLICIT REAL*8 (A-Z)
          INTEGER*2 I
          INTEGER*8 MD,N,M,ID,NA,NB,LD,IR,IR1,IRS,IBETA,IGAMA,JSX,JSY
          INTEGER*8 LK,LG,IR1,IFR,INV,INV1,IA,IB
          INTEGER*8 IX1(32),IY1(32),IX2(32),IY2(32)
          INTEGER*8 IQRNS(32),JQRNS(32),KQRNS(32),LQRNS(32)
          INTEGER*8 ITR(32),ITI(32),ITWI1(32),ITWI2(32)
          DATA IX1/16*0,16*1/, IY1/16*0,16*1/
          DATA IX2/16*0,16*1/, IY2/16*0,16*0/
          TYPE *,'ENTER MODULUS, MD,N,M,ID..'
          ACCEPT *, MD,N,M,ID
          TYPE *,'ENTER VALUE OF IBETA,IGAMA....'
          ACCEPT *, IBETA,IGAMA
          LD=(MD+1)/2
          IR=0
1         IR=IR+1
          IR1=IR**2
          IR1=IR1+1
          CALL RESIDUE(IR1,MD,ID)
          IF(IR1.NE.0) GO TO 1
          TYPE *, IR
          WRITE(6,2)
          FORMAT(//,10X,'TWIDLEFACTORS FOR THE NTT')
          NA=IBETA
          NB=IGAMA
          JSX=NA+IR*NB
          JSY=NA-IR*NB
          CALL RESIDUE(JSX,MD,ID)
          CALL RESIDUE(JSY,MD,ID)
          ITWI1(1)=1
          ITWI2(1)=1
          ITWI1(2)=JSX
          ITWI2(2)=JSY
          LK=NA
          LG=NB
          IR1=-1
          DO 3 I=3,N
          IA=NA*LK+IR1*NB*LG
          IB=NA*LG+NB*LK
          CALL RESIDUE(IA,MD,ID)
          CALL RESIDUE(IB,MD,ID)
          ITWI1(I)=IA+IR*IB
          ITWI2(I)=IA-IR*IB
          CALL RESIDUE(ITWI1(I),MD,ID)
          CALL RESIDUE(ITWI2(I),MD,ID)
          LK=IA
          LG=IB
          WRITE(6,4) I,ITWI1(I),I,ITWI2(I)
4         FORMAT(5X,'ITWI1(',I3,')=',I16,1X,'ITWI2(',I3,')=',I16)
3         CONTINUE
          DO 5 I=1,N
```

```
      IQRNS(I)=IX1(I)+IR*IY1(I)
      JQRNS(I)=IX1(I)-IR*IY1(I)
      CALL RESIDUE(IQRNS(I),MD,ID)
      CALL RESIDUE(JQRNS(I),MD,ID)
      KQRNS(I)=IX2(I)+IR*IY2(I)
      LQRNS(I)=IX2(I)-IR*IY2(I)
      CALL RESIDUE(KQRNS(I),MD,ID)
      CALL RESIDUE(LQRNS(I),MD,ID)
5     CONTINUE
      IFR=0
      CALL NTT(IQRNS,JQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID)
      CALL NTT(KQRNS,LQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID)
      DO 6 I=1,N
      IQRNS(I)=IQRNS(I)*KQRNS(I)
      JQRNS(I)=JQRNS(I)*LQRNS(I)
      CALL RESIDUE(IQRNS(I),MD,ID)
      CALL RESIDUE(JQRNS(I),MD,ID)
6     CONTINUE
      INV=0
7     INV=INV+1
      INV1=INV*N
      CALL RESIDUE(INV1,MD,ID)
      IF(INV1.NE.1) GO TO 7
      DO 8 I=1,N
      IQRNS(I)=INV*IQRNS(I)
      JQRNS(I)=INV*JQRNS(I)
      CALL RESIDUE(IQRNS(I),MD,ID)
      CALL RESIDUE(JQRNS(I),MD,ID)
8     CONTINUE
      IFR=1
      CALL NTT(IQRNS,JQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID)
      DO 101 I=1,N
101   CONTINUE
      WRITE(6,9)
9     FORMAT(//,10X,'CONVOLUTION OUTPUT IN QRNS')
      IRS=(-IR+MD)
      DO 10 I=1,N
      ITR(I)=LD*(IQRNS(I)+JQRNS(I))
      ITI(I)=LD*IRS*(IQRNS(I)-JQRNS(I))
      CALL RESIDUE(ITR(I),MD,ID)
      CALL RESIDUE(ITI(I),MD,ID)
      IF(ITR(I).LT.0) ITR(I)=ITR(I)+MD
      IF(ITI(I).LT.0) ITI(I)=ITI(I)+MD
      WRITE(6,11) I,ITR(I),I,ITI(I)
11    FORMAT(5X,'ITR(',I3,')=',I16,1X,'ITI(',I3,')=',I16)
10    CONTINUE
      STOP
      END
      INCLUDE NTT3
      INCLUDE RESIDUE
$ALLOCATE 30000
$EXECUTE CATALOG
*A4 6=UT
BUILD QRNS.SU,NOM
$EOJ
```

```
$JOB ARUL    KRIS,RAJI                SLOF=DUMMY
$OPTION   2 3-4 5 17
$EXECUTE FORTRAN
C         FILE NAME:-MQRNS CONVOLUTION
          IMPLICIT REAL*8 (A-Z)
          INTEGER*2 I
          INTEGER*8 MD,N,M,ID,NA,NB,LD,IR,IR1,IRS,IBETA,IGAMA,JSX,JSY
          INTEGER*8 LK,LG,IR1,IFR,INV,INV1
          INTEGER*8 IX1(32),IY1(32),IX2(32),IY2(32)
          INTEGER*8 IQRNS(32),JQRNS(32),KQRNS(32),LQRNS(32)
          INTEGER*8 IA(32),IB(32),ITR(32),ITI(32)
          INTEGER*8 ITWI1(32),ITWI2(32)
          TYPE *,'ENTER MODULUS, MD,N,M,ID..'
          ACCEPT *, MD,N,M,ID
          TYPE *,'ENTER VALUE OF IBETA,IGAMA....'
          ACCEPT *, IBETA,IGAMA
C         DATA IX1/16*0,16*1/, IY1/16*0,16*1/
C         DATA IX2/16*0,16*1/, IY2/16*0,16*0/
          DO 201 I=1,N
          TYPE *,'ENTER THE VALUE OF IX1(I),IY1(I)...'
          ACCEPT *, IX1(I),IY1(I)
201       CONTINUE
          DO 200 I=1,N
          TYPE *,'ENTER THE VALUE OF IX2(I),IY2(I)...'
          ACCEPT *, IX2(I),IY2(I)
200       CONTINUE
          LD=(MD+1)/2
          WRITE(6,2)
2         FORMAT(//,10X,'TWIDLEFACTORS FOR THE NTT')
          NA=IBETA
          NB=IGAMA
          JSX=NA+NB
          JSY=NA-NB
          CALL RESIDUE(JSX,MD,ID)
          CALL RESIDUE(JSY,MD,ID)
          ITWI1(1)=1
          ITWI2(1)=1
          ITWI1(2)=JSX
          ITWI2(2)=JSY
          IA(1)=1
          IB(1)=0
          IA(2)=NA
          IB(2)=NB
          LK=NA
          LG=NB
          IR1=-1
          DO 3 I=3,N
          IA(I)=NA*LK+IR1*NB*LG
          IB(I)=NA*LG+NB*LK
          CALL RESIDUE(IA(I),MD,ID)
          CALL RESIDUE(IB(I),MD,ID)
          ITWI1(I)=IA(I)+IB(I)
          ITWI2(I)=IA(I)-IB(I)
          CALL RESIDUE(ITWI1(I),MD,ID)
          CALL RESIDUE(ITWI2(I),MD,ID)
```

```
      SUBROUTINE NTT(IQRNS,JQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID)
C      FILE:- NTT FOR QRNS
      IMPLICIT DOUBLE PRECISION (A-Z)
      INTEGER*8 IQRNS(256),JQRNS(256),ITWI1(256),ITWI2(256)
      INTEGER*8 MD,N,M,IFR,NV2,NM1,I,J,IT,K,L,LE,LE1,NLE,MI,IP
      INTEGER*8 IT1,IT2,IF1,IF2,ID
      NV2=N/2
      NM1=N-1
      J=1
C     BIT REVERSAL OF THE INPUT RESIDUE CODES
      DO 7 I=1,NM1
      IF(I.GT.J) GO TO 5
      IT=IQRNS(J)
      IQRNS(J)=IQRNS(I)
      IQRNS(I)=IT
      IT=JQRNS(J)
      JQRNS(J)=JQRNS(I)
      JQRNS(I)=IT
    5 K=NV2
    6 IF(K.GE.J) GO TO 7
      J=J-K
      K=K/2
      GO TO 6
    7 J=J+K
      XMD=MD
      DO 20 L=1,M
      LE=2**L
      LE1=LE/2
      NLE=N/LE
      MI=1
      DO 20 J=1,LE1
      IF1=ITWI1(MI)
      IF2=ITWI2(MI)
      IF(IFR.NE.1) GO TO 18
      IF(MI.EQ.1) THEN
      IF1=ITWI1(1)
      IF2=ITWI2(1)
      ENDIF
      IF(MI.NE.1) THEN
      IF1=ITWI1(N+2-MI)
      IF2=ITWI2(N+2-MI)
      ENDIF
   18 DO 10 I=J,N,LE
      IP=I+LE1
      IT1=IQRNS(IP)*IF1
      IT2=JQRNS(IP)*IF2
      CALL RESIDUE(IT1,MD,ID)
      CALL RESIDUE(IT2,MD,ID)
      IQRNS(IP)=IQRNS(I)-IT1
      JQRNS(IP)=JQRNS(I)-IT2
      CALL RESIDUE(IQRNS(IP),MD,ID)
      CALL RESIDUE(JQRNS(IP),MD,ID)
      IQRNS(I)=IQRNS(I)+IT1
      JQRNS(I)=JQRNS(I)+IT2
      CALL RESIDUE(JQRNS(I),MD,ID)
```

```
      CALL RESIDUE(IQRNS(I),MD,ID)
10    CONTINUE
      MI=MI+NLE
20    CONTINUE
      RETURN
      END
```

```
      IF(ITWI1(I).LT.0) ITWI1(I)=ITWI1(I)+MD
      IF(ITWI2(I).LT.0) ITWI2(I)=ITWI2(I)+MD
      WRITE(6,4) I,ITWI1(I),I,ITWI2(I)
4     FORMAT(5X,'ITWI1(',I3,')=',I25,1X,'ITWI2(',I3,')=',I25)
      LK=IA(I)
      LG=IB(I)
3     CONTINUE
      DO 5 I=1,N
      IQRNS(I)=IX1(I)+IY1(I)
      JQRNS(I)=IX1(I)-IY1(I)
      CALL RESIDUE(IQRNS(I),MD,ID)
      CALL RESIDUE(JQRNS(I),MD,ID)
       IF(IQRNS(I).LT.0) IQRNS(I)=IQRNS(I)+MD
       IF(JQRNS(I).LT.0) JQRNS(I)=JQRNS(I)+MD
      KQRNS(I)=IX2(I)+IY2(I)
      LQRNS(I)=IX2(I)-IY2(I)
      CALL RESIDUE(KQRNS(I),MD,ID)
      CALL RESIDUE(LQRNS(I),MD,ID)
      IF(KQRNS(I).LT.0) KQRNS(I)=KQRNS(I)+MD
      IF(LQRNS(I).LT.0) LQRNS(I)=LQRNS(I)+MD
5     CONTINUE
      WRITE(6,115)
115   FORMAT(//,10X,'ELEMENT PAIRS OF THE DATA')
      DO 111 I=1,N
      WRITE(6,110) I,IQRNS(I),I,JQRNS(I)
110     FORMAT(5X,'IQRNS(',I3,')=',I25,1X,'JQRNS(',I3,')=',I25)
111    CONTINUE
      DO 112 I=1,N
      CONTINUE
      WRITE(6,101) I,KQRNS(I),I,LQRNS(I)
101     FORMAT(5X,'KQRNS(',I3,')=',I25,1X,'LQRNS(',I3,')=',I25)
112    CONTINUE
      IFR=0
      CALL NTT(IQRNS,JQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID,IY1,IB)
      CALL NTT(KQRNS,LQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID,IY2,IB)
      DO 103 I=1,N
      IF(IQRNS(I).LT.0) IQRNS(I)=IQRNS(I)+MD
      IF(JQRNS(I).LT.0) JQRNS(I)=JQRNS(I)+MD
      IF(KQRNS(I).LT.0) KQRNS(I)=KQRNS(I)+MD
      IF(LQRNS(I).LT.0) LQRNS(I)=LQRNS(I)+MD
103     CONTINUE
      WRITE(6,127)
127   FORMAT(//,10X,'CNTT OF THE DATA')
       DO 125 I=1,N
      WRITE(6,122) I,IQRNS(I),I,JQRNS(I)
122     FORMAT(5X,'IQRNS(',I3,')=',I25,1X,'JQRNS(',I3,')=',I25)
125    CONTINUE
      DO 126 I=1,N
      WRITE(6,102) I,KQRNS(I),I,LQRNS(I)
102     FORMAT(5X,'KQRNS(',I3,')=',I25,1X,'LQRNS(',I3,')=',I25)
126   CONTINUE
      DO 6 I=1,N
      IQRNS(I)=IQRNS(I)*KQRNS(I)
      JQRNS(I)=JQRNS(I)*LQRNS(I)
      IY1(I)=2*IY1(I)*IY2(I)
```

```
            CALL RESIDUE(IY1(I),MD,ID)
            IQRNS(I)=IQRNS(I)-IY1(I)
            JQRNS(I)=JQRNS(I)-IY1(I)
            CALL RESIDUE(IQRNS(I),MD,ID)
            CALL RESIDUE(JQRNS(I),MD,ID)
6           CONTINUE
C           INV=0
C           INV=INV+1
C           INV1=INV*N
C           CALL RESIDUE(INV1,MD,ID)
C           IF(INV1.NE.1) GO TO 7
            INV=(MD+1)/N
            TYPE *, INV
            DO 8 I=1,N
            IQRNS(I)=INV*IQRNS(I)
            JQRNS(I)=INV*JQRNS(I)
            IY1(I)=LD*(IQRNS(I)-JQRNS(I))
            CALL RESIDUE(IQRNS(I),MD,ID)
            CALL RESIDUE(JQRNS(I),MD,ID)
            CALL RESIDUE(IY1(I),MD,ID)
8           CONTINUE
            IFR=1
            CALL NTT(IQRNS,JQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID,IY1,IB)
            WRITE(6,9)
9           FORMAT(//,10X,'CONVOLUTION OUTPUT IN MQRNS')
            DO 10 I=1,N
            ITR(I)=LD*(IQRNS(I)+JQRNS(I))
            ITI(I)=LD*(IQRNS(I)-JQRNS(I))
            CALL RESIDUE(ITR(I),MD,ID)
            CALL RESIDUE(ITI(I),MD,ID)
            IF(ITR(I).LT.0) ITR(I)=ITR(I)+MD
            IF(ITI(I).LT.0) ITI(I)=ITI(I)+MD
            WRITE(6,11) I,ITR(I),I,ITI(I)
11          FORMAT(5X,'ITR(',I3,')=',I25,1X,'ITI(',I3,')=',I25)
10          CONTINUE
            STOP
            END
            INCLUDE NTT4
            INCLUDE RESIDUE
$ALLOCATE 30000
$EXECUTE CATALOG
A4 6=UT
BUILD MQRNS.SU,NOM
$EOJ
```

```
      SUBROUTINE NTT(IQRNS,JQRNS,MD,N,M,ITWI1,ITWI2,IFR,ID,IY,IB)
C      FILE:- NTT FOR MQRNS
      IMPLICIT DOUBLE PRECISION (A-Z)
      INTEGER*8 IQRNS(256),JQRNS(256),ITWI1(256),ITWI2(256)
      INTEGER*8 IY(256),IB(256)
      INTEGER*8 MD,N,M,IFR,NV2,NM1,I,J,IT,K,L,LE,LE1,NLE,MI,IP
      INTEGER*8 IT1,IT2,IF1,IF2,ID,IF3,IT3
      NV2=N/2
      NM1=N-1
      J=1
C     BIT REVERSAL OF THE INPUT RESIDUE CODES
      DO 7 I=1,NM1
      IF(I.GT.J) GO TO 5
      IT=IQRNS(J)
      IQRNS(J)=IQRNS(I)
      IQRNS(I)=IT
      IT=JQRNS(J)
      JQRNS(J)=JQRNS(I)
      JQRNS(I)=IT
      IT=IY(J)
      IY(J)=IY(I)
      IY(I)=IT
    5 K=NV2
    6 IF(K.GE.J) GO TO 7
      J=J-K
      K=K/2
      GO TO 6
    7 J=J+K
      XMD=MD
      DO 20 L=1,M
      LE=2**L
      LE1=LE/2
      NLE=N/LE
      MI=1
      DO 20 J=1,LE1
      IF1=ITWI1(MI)
      IF2=ITWI2(MI)
      IF3=IB(MI)
      IF(IFR.NE.1) GO TO 18
      IF(MI.EQ.1) THEN
      IF1=ITWI1(1)
      IF2=ITWI2(1)
      IF3=IB(1)
      ENDIF
      IF(MI.NE.1) THEN
      IF1=ITWI1(N+2-MI)
      IF2=ITWI2(N+2-MI)
      IF3=IB(N+2-MI)
      ENDIF
   18 DO 10 I=J,N,LE
      IP=I+LE1
      IT1=IQRNS(IP)*IF1
      IT2=JQRNS(IP)*IF2
      IT3=2*IY(IP)*IF3
      CALL RESIDUE(IT1,MD,ID)
```

```
      CALL RESIDUE(IT2,MD,ID)
      CALL RESIDUE(IT3,MD,ID)
      IT1=IT1-IT3
      IT2=IT2-IT3
      IF(IT1.LT.0) IT1=IT1+MD
      IF(IT2.LT.0) IT2=IT2+MD
      IQRNS(IP)=IQRNS(I)-IT1
      JQRNS(IP)=JQRNS(I)-IT2
      CALL RESIDUE(IQRNS(IP),MD,ID)
      CALL RESIDUE(JQRNS(IP),MD,ID)
      LD=(MD+1)/2
      IY(IP)=LD*(IQRNS(IP)-JQRNS(IP))
      CALL RESIDUE(IY(IP),MD,ID)
      IQRNS(I)=IQRNS(I)+IT1
      JQRNS(I)=JQRNS(I)+IT2
      CALL RESIDUE(JQRNS(I),MD,ID)
      CALL RESIDUE(IQRNS(I),MD,ID)
      IY(I)=LD*(IQRNS(I)-JQRNS(I))
      CALL RESIDUE(IY(I),MD,ID)
   10 CONTINUE
      MI=MI+NLE
   20 CONTINUE
      RETURN
      END
```

VITA AUCTORIS

1949	Born on August 10, Mangammalpalti, Madras, India.
1965	Completed High School at Government High School, Peraiyur, Madras, India.
1966	Completed Pre-university at the University of Madras, Madras, India.
1972	Graduated from the University of Madras, Madras, India with the degree of Bachelor of Engineering in Electrical Engineering.
1974	Graduated from the University of Madras, Madras, India, with the degree of M.Sc. (Engg.) in Applied Electronics.
1974-1979	Associate Lecturer/Lecturer in Electronics and Communication Engineering at the University of Madras, (College of Engineering, Guindy).
1979-1982	Professor at William V.S. Tubman College of Technology, Liberia, West Africa in Electronics Engineering.
1985	Candidate for the degree of Doctor of Philosophy in Electrical Engineering, University of Windsor, Windsor, Ontario, Canada.