1980

DIGITAL FILTERING USING NUMBER-THEORETIC TECHNIQUES.

ANNA Z. BARANIECKA
University of Windsor

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

DIGITAL FILTERING USING NUMBER THEORETIC TECHNIQUES

by

Anna Z. Baraniecka

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

1980

ABSTRACT

This work is an investigation into the use of the number theoretic techniques for implementing digital signal processing algorithms.

In particular, residue coding principles are applied to fast digital convolution via the Number Theoretic Transform (NTT) and to digital filtering via recursive digital filters.

The initial study of implementing the NTT using the residue number system (RNS) leads to the idea of implementing the transform over a direct sum of several extension fields, or rings, with the RNS architectures of ROM arrays or microprocessor arrays. New results and theorems are obtained for transform parameters that have a simple form which allows a reduction in the number of binary operations and allowing efficient implementation. These results are verified by computer programs.

The application of the RNS for second order recursive digital filter sections is also investigated. Some of the design problems associated with read-only-memory implementation of second order sections, which can be used as building blocks for higher order recursive filters are considered along with a study of quantization error and limit cycle behavior.

A new scheme for translation of the residue coded output into a binary representation is also developed in this thesis.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr. G.A. Jullien for his invaluable advice, guidance and constant encouragement throughout the progress of this thesis. The advice of Dr. W.C. Miller, Dr. J.J. Soltis and the other faculty members of the Electrical Engineering Department is gratefully acknowledged.

I extend my sincerest thanks and gratitude to my husband, Marian, and to my parents for their moral support, help and constant encouragement at all times.

Thanks are also due to Mrs. S.A. Ouellette for her diligence in typing this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

During the last two decades there has been an increasing interest in the practical applications of number theoretic techniques, including the theory of the residue number system (RNS), for implementation of digital signal processing elements.

The theoretical foundations of the residue arithmetic were developed in the eighteenth and nineteenth century by Euler, Fermat and Gauss, although many results have been familiar to earlier mathematicians, eg., the so-called Chinese Remainder Theorem was first reported by the Chinese mathematician Sun-Tsu in the first century A.D. However, the application of number theory is of fairly recent origin.

Recent advances in high density memory technology and microprocessor hardware have stimulated a modern revival of studies on the use of residue number techniques as applied to the implementation of digital arithmetic processors.

The first published report on the modern work in residue arithmetic, in the context of electronic computers, was that of M.Valach and A.Svoboda in Czechoslovakia,[1]. They also designed and constructed the first general purpose computer, EPOS, based on the RNS concepts. The work of Szabo and Tanaka[2] is an important reference on the applications of the RNS principles to computer technology, though the investigations into the use of this number system for general purpose computers have not yielded many practical results.

The difficulties associated with implementing operations such as division, scaling, sign determination and magnitude comparison overshadowed the advantages of very fast binary operations of addition and

multiplication.

More recently, various investigators have considered the application of residue number systems to the implementation of digital signal processing structures that have an abundance of easy RNS operations, addition, subtraction and multiplication, and relatively few scaling operations (required to prevent overflow).

Many algorithms in the digital signal processing environment fall into this category and some of them have already been explored. In particular, Jenkins and Leon [3] investigated RNS techniques for non-recursive digital filters, Jullien [4] and Jenkins [5] considered the different realizations of recursive digital filters using ROM arrays, with emphasis placed on obtaining efficient scaling algorithms.

Soderstrand [6] studied Residue Number System lossless discrete integrator (LDI) ladder structures. Also Fast Fourier Transform implementation using RNS principles have been considered [7]. Multiple microprocessor implementations of various digital signal processing algorithms [8, 9, 10, 11] have also been studied.

The conclusion that emerges from these works is that Residue Number System is becoming an attractive and useful tool for the implementation of digital signal processors.

1.1 Objectives and Outline of the Research Work

The principal objective of this research is to explore the application of residue number techniques to the construction of digital signal processing hardware. The motivation for this work is based upon recently published results, showing the attractiveness of using residue number techniques to obtain very high speed digital arithmetic processing hardware.

As a starting point, we explore the basic hardware principles involved in performing arithmetic operations in the RNS, and in interfacing RNS hardware with conventional binary systems. Following this, the work explores, more directly, the application of these hardware principles to digital signal processing functions.

The scope of this work has, of necessity, been limited to a fundamental sub-class of digital signal processing functions: namely, digital filtering. The work described in this thesis addresses both finite impulse response (FIR) and infinite impulse response (IIR) digital filters. Reports are already available in the literature for residue number system hardware realizations of both types of filter, and so our objectives are to expand upon and augment the published ideas.

In the case of finite impulse response filters, we have found that algorithms which employ indirect filtering, via transforms, to be most interesting when implemented using residue number system ideas. The scope of this work has been such that our entire efforts in investigating finite impulse response filter realizations have been involved in transform techniques. The specific class of algorithms used have been referred to, in the literature, as a Number Theoretic Transforms, 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, so it can take advantage of the strong features of residue coding.

The motivation behind this work is to develop algorithms that will provide a degree of flexibility in the implementation of Number Theoretic Transforms using RNS architectures of ROM arrays and microprocessor

arrays. New theorems and results, based on concepts from abstract algebra and number theory, are presented that allow selection of transform parameters for implementation of NTTs over a direct sum of Galois Fields of order m_i^2. Under certain conditions the transform parameters can have a simple form which allows a more efficient implementation than that of the general form.

The objectives of our work in infinite impulse response (recursive) filters, are to generate means of identifying the most suitable structures for RNS implementation. The proposed structures should be attractive for adaptive filtering and have low sensitivity to quantization noise.

To this end we have examined different structures (already considered in the literature) for sensitivity to both quantization noise and limit cycle effects. The result of this work show that it is possible to achieve low sensitivity without compromising the speed of the realization.

Since, in general, digital recursive filters implemented with any fixed-point arithmetic are subject to automonous limit cycles, we will also examine the limit cycle phenomena in the proposed RNS based structures.

A critical review of existing material on the topics considered in this thesis, will be presented at the beginning of the appropriate chapter. The review is presented in this manner because of the diverse nature of the different topics studied; namely, recursive filter quantization effects and high-speed convolution (FIR filtering) using Number Theoretic Transforms.

1.2 Thesis Organization

Chapter 2 provides prerequisite material for the work which follows. In particular a concise review of some of the fundamentals on finite ring and field structures is presented and the mathematical concepts of residue arithmetic are introduced.

This chapter is included as a background reference for definitions, nomenclature and notations used throughout the text.

Chapter 3 covers the implementation aspects of residue arithmetic including techniques for performing the arithmetic along with methods for interfacing RNS based system with binary based systems.

In Chapter 4, algorithms for RNS based implementation of Fast Number Theoretic Transforms are developed. These transforms, having the cyclic convolution property, are useful for error-free FIR digital filter implementation. Many ramifications of the application of RNS techniques for the implementation of NTTs are described in this chapter.

In Chapter 5, the application of residue coding to second order recursive digital filter sections is investigated. An analysis of quantization error accumulation in the proposed recursive residue structures is presented along with an investigation of the existence of limit cycles.

Chapter 6 summarizes the results of this research.

CHAPTER 2

FINITE RINGS AND FIELDS. THE RESIDUE NUMBER SYSTEM ALGEBRA

2.1 Introduction

This section introduces the concept of residue algebra and presents a brief review of the background material on the ring and field structure, as a foundation for work in the succeeding chapters.

A list of notational definitions used throughout the text is given in the Appendix A. The Residue Number System (RNS) is an integer number system and is based on a theory of congruences, a part of a branch of mathematics called "Theory of Numbers". The word "numbers" in this connection is usually understood to mean integers 0, ±1, ±2, ...

In chapter 4, the residue number system will be used for coding the integers of a more general kind, like the residue classes of Gaussian or quadratic integers. For this purpose, we will briefly introduce the structures of extension rings and fields, a branch of abstract algebra.

2.2 Rings and Fields

As a starting point, it would seem appropriate to formally define the notion of a ring and field.

Definition 1: If R is a nonempty set on which there are defined binary operations of addition and multiplication, such that the following postulates (I) - (VI) hold, we say that R is a ring.

(I) commutative law of addition $a + b = b + a$

(II) associative laws $(a + b) + c = a + (b + c)$

(III) distributive laws $a \cdot (b + c) = a \cdot b + a \cdot c$

(IV) existence of an element denoted by the symbol 0 of R such that

$a + 0 = a$ for every $a \in R$

(V) , existence of additive inverses. For each $a \in R$, there exists $x \in R$ such that $a + x = 0$

(VI) closure

where it is understood that a, b, c are arbitrary elements of R.

A ring in which multiplication is a commutative operation 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 $a \in R$.

The ring of integers is a well known example of a commutative ring with identity. In abstract albebra, elements of a ring are not necessarily the integers, even not necessarily numbers, eg, they can be polynomials.

Given a ring R with identity 1, an element $a \in R$ is said to be invertible, or to be a unit, whenever a possesses an inverse with respect to multiplication. The multiplicative inverse, a^{-1}, is an element such that $a^{-1} \cdot a = 1$. 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: If R is a ring and $0 \neq a \in R$, then a is called a divisor of zero if there exists some $b \neq 0$ in R such that $ab = 0$.

Definition 3: An integral domain is a commutative ring with identity which has no divisors of zero.

Definition 4: A ring, F, is said to be a field provided that the set $F-\{0\}$ is a commutative group under multiplication.

Viewed otherwise: a field is a commutative ring with identity in which each non-zero element possesses an inverse under multiplication. It

follows [69] 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 modulo m, denoted here as Z_m is an example of a finite ring.

In every finite field with p elements, the nonzero elements form a multiplicative group. This multiplicative group of order p-1 is cyclic, ie, it contains an element α whose powers exhaust the entire group. This element is called a generator or a primitive root of unity and the period or order of α is p-1. The order of any element x in the multiplicative group is the least positive integer t such that $x^t = 1$, $x^s \neq 1$, $1 \leq s < t$. The order t is a divisor of p-1 and x is called a primitive t-th root of unity.

For every element x of the set F - {0} the mapping defined by

$$x = \alpha^{\tau} \quad , \tau \in \{0, 1, \ldots, p-2\} \tag{2.1}$$

is the isomorphism of the additive group of integers with addition modulo p-1 and the multiplicative group of the field F.

The integer τ is called the index of x relative to the base α, denoted $\text{ind}_{\alpha} x$.

2.2.1 The ring of residue classes

In order to describe the system, the notion of congruence should be introduced. The basic theorem, known as the "Division Algorithm" will be established first.

Division Algorithm:

For given integers a and b, b not zero, 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 the quotient $\frac{a}{b}$. 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 5: Two integers c and d are said to be congruent modulo m, written

$c \equiv d \pmod{m}$ if and only if $m|(c-d)$

Since $m|0$, $c \equiv c \pmod{m}$ by definition.

Another alternative definition of congruence can be stated as follows:

Two integers c and d are congruent modulo m if and only if they leave the same remainder when divided by m, $|c|_m = |d|_m$.

Definition 6: 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 classes (mod m) form a commutative ring with identity with respect to the modulo m addition and multiplication, traditionally known as the ring of integers modulo m or the residue ring and denoted Z_m. The 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, because multiplicative inverses, denoted $a^{-1} \pmod{m}$ or $\left|\frac{1}{a}\right|_m$, exist for each nonzero $a \in Z_m$. Thus the nonzero classes of Z_m form a cyclic multiplicative group of order m-1, $\{1, 2, \ldots, m-1\}$, with multiplication modulo m, isomorphic to the additive group $\{0, 1, \ldots, m-2\}$ with addition modulo m-1.

Example:

For modulo 7 multiplication, which is the binary operation in a multiplicative cyclic group of order 6, there exists two generators, namely 3 and 5. The mapping of modulo 7 multiplication onto modulo 6 addition, for $\alpha = 3$, is given in the table below:

x	$\text{ind}_3\ x$
1	0
2	2
3	1
4	4
5	5
6	3

eg., the multiplication $|3 \times 5|_7 = 1$ is mapped into the addition $|1 + 5|_6 = 0$.

If m is composite, Z_m is not a field. The multiplicative inverses do not exist for those non-zero elements $a \in Z_m$ for which gcd (a, m) $\neq 1$. The Euler's totient function, denoted $\phi(m)$, and obtained from (2.3) is a widely used number theoretic function and is defined as the number of positive integers less than m and relatively prime to it. It follows that the number of invertible elements of Z_m is equal to $\phi(m)$.

$$\phi(m) = m\left[1 - \frac{1}{m_1}\right]\left[1 - \frac{1}{m_2}\right] \cdots\cdots \left[1 - \frac{1}{m_L}\right] \qquad (2.3)$$

where m has prime power factorization $m = m_1^{e_1} \cdot m_2^{e_2} \cdot m_L^{e_L}$.

The Euler-Fermat Theorem states that if c is an integer, m is a positive integer and (c, m) = 1, then $c^{\phi(m)} \equiv 1 \pmod m$.

The important consequence 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 $\phi(m)$.

2.2.2 Galois Fields

For any prime m and any positive integer n, there exists a finite field with m^n elements. This (essentially unique) field is commonly denoted by the symbol $GF(m^n)$ and is called a Galois field in honor of the French mathematician Evariste Galois. Since any finite field with m^n elements is a simple algebraic extension of the field Z_m, 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 λ is algebraic over F, ie, if λ is a root of some irreducible polynomial $f(x) \in F[x]$ such that $f(\lambda) = 0$, then the extension field arising from a field F by the adjunction of a root λ is called a simple algebraic extension, denoted $F(\lambda)$.

Each element of $F(\lambda)$ can be uniquely represented as a polynomial $a_0 + a_1 \lambda + \ldots, a_{n-1} \lambda^{n-1}$, $a_i \in F$. This unique representation closely resembles the representation of a vector in terms of the vectors of a basis "$1, \lambda, \ldots, \lambda^{n-1}$". The vector space concepts are sometimes applied to the extension fields, and $F(\lambda)$ is considered as a vector space of dimension n over F.

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

If f(x) is an irreducible polynomial of degree n over Z_m, m prime, then the Galois Field with m^n elements, $GF(m^n)$ is usually defined ([46], [41]) as the quotient field $Z_m[x] / (f(x))$, ie, 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 $Z_m(\lambda)$, where λ is a root of $f(x) = 0$.

2.3 The Residue Number System

The way of coding numbers by forming a direct sum of several residue class rings (not necesarily fields) is known as the residue number system (RNS).

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

$$X = (x_0, x_1, \ldots, x_{L-1}) \tag{2.4}$$

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

$$(m_0, m_1, \ldots, m_{L-1}).$$

The residues are formally written $x_i = |X|_{m_i}$. The residue representation of a number is unique. The converse of this statement is true only if we consider the numbers within the range of the number system. Precisely, if all the m_i are relatively prime it can be shown [2] that there is a unique representation for each number in the range $0 \leq X < \prod_{i=0}^{L-1} m_i = M$. The binary operations under which the system is closed, viz., addition or multiplication between two variables represented in the RNS, can be performed by independent operations on the respective digits, ie,

$$Z = X \square Y \text{ implies } z_i = |x_i \square y_i|_{m_i} \tag{2.5}$$

where $\square \in (+, \cdot)$

A signed integer system can be developed by attaching a positive sign to numbers X in the range $0, 1, \ldots, \frac{M}{2} - 1$ for M even or $0, 1, \ldots, \frac{M-1}{2}$ for M odd, and a negative sign to the number (-X) in the range $\frac{M}{2}, \frac{M}{2} + 1, \ldots, M-1$ or $\frac{M+1}{2}, \ldots, M-1$ respectively. The additive inverse,

in modulo complement form, is given: $\bar{X} = M - X$ 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.

An example of RNS computation is given below:

Example:

Choose the moduli set {3,4}. The RNS coding of the numbers (with the dynamic range [-6, 5]) is shown in Table 2.1.

X	$\|X\|_3$	$\|X\|_4$
-6	0	2
-5	1	3
-4	2	0
-3	0	1
-2	1	2
-1	2	3
0	0	0
1	1	1
2	2	2
3	0	3
4	1	0
5	2	1

TABLE 2.1 A residue number system for $m_0 = 3$ and $m_1 = 4$

An example of a computation is illustrated below:

$2 \times 5 - 6 = 4$

$(2,2) \times (2,1) + (0,2) = (1,2) + (0,2) = (1,0)$

Multiplicative inverses exist for each nonzero residue digit using

modulus 3 and only for the residue digits 1 and 3 for the modulus 4. For example $\left| \frac{1}{3} \right|_4 = 3$. It can be seen from Table 2.1 that the sign of a number is not explicitly shown nor are the relative magnitudes of two numbers.

2.4 Summary

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

The notions of finite rings and finite (Galois) fields and the concept of primitive roots have been introduced. This is prerequisite knowledge for the development of algorithms for the computation of finite digital convolution using transform techniques.

The problem of the existence of multiplicative inverses has been discussed, and Euler's totient function has been defined. The idea of congruence has been presented as a foundation of the mathematical concept of the residue number system. The basic properties of the residue number system have been described and the independence of binary operations on the residue digits has been emphasized.

CHAPTER 3

RESIDUE NUMBER SYSTEM IMPLEMENTATION AND INTERFACE WITH CONVENTIONAL BINARY STRUCTURES

3.1 Implementation of Arithmetic Operations in the RNS

The unique advantage of the residue number system is that the binary operations of addition (subtraction) or multiplication on the respective residues can be performed independently and in parallel. Addition and subtraction have no inter-digit carries or borrows and multiplication does not need the generation of partial products, hence fast operating speeds can be obtained.

Another important aspect of the RNS is its adaptability to look-up table implementation. In the binary number system, the table look-up approach is not feasible because of the enormous storage required for useful bit lengths. For a wordlength of B bits, 2^{2B} entries would be required in the table; however, in the RNS with a comparable range, ie, $\prod_{i=1}^{L} m_i \approx 2^B$, each modulus, m_i, requires m_i^2 entries in the table, hence a total of $\sum_{i=1}^{L} m_i^2$ entries is needed. We obtain the obvious result $\sum_{i=1}^{L} m_i^2 << \prod_{i=1}^{L} m_i^2$ for reasonable values of L and $\{m_i\}$.

There have been several basic approaches to the design of modulo arithmetic hardware. They fall into three main groups:

1) direct logical implementation of the Boolean function related to the operation
2) storing all possible outcomes of the operation (look-up table)

3) storing software algorithms in a general purpose computer. (Mixtures of the above have been sometimes implemented [11], [48])

The first approach has been widely discussed [2], [12], [13], mainly for the implementation of addition, because the only practical implementations of multiplication are by means of stored multiplication tables or stored index tables. However, except for the case where $m = 2^B$ the Boolean functions become unweildy, and in most cases no obvious decomposition exists for those switching functions. The first approach also includes the modification of conventional arithmetic units [2], [3]. In a fixed point two's complement system, adder overflow results in the true sum being reduced mod 2^B. For the one's complement system, the reduction is mod (2^B-1) since the end-around carry is added back onto the least significant bit. This can be generalized to any modulus, m, by adding back a generalized end-around carry $C = 2^B-m$, where 2^B is the smallest power of 2 such that $2^B > m$,[3]. However, if m diverges from 2^B, complicated logic circuitry is required for detecting forbidden combinations and performing the necessary corrections. Due to recent advances in high-density memory technology ROM array and computer implementations, discussed in the next two sections, have become an increasingly attractive replacement for logic circuitry.

3.1.1 Look-up table implementation

As memory prices continue to decrease, table look-up operations become more and more attractive. Also, the look-up table approach offers the best solution for high speed realization. This is particularly advantageous in multiplication, which becomes as simple and fast as addition. In earlier reports of residue number system implementa-

tions [2], such look-up tables were realized with magnetic core matrices, which were expensive and slow.

With current advances in semiconductor high density memory systems, the look-up table approach is a much more viable option with large dynamic ranges being accommodated in quite small package counts, and with nominal amounts of supply power required. However, restrictions have to be placed on the size of the maximum modulus in a system. For a given modulus $m_i \leq 32$, the binary operations can be carried out by storing the mod m_i truth table in 5k bit read only memory (ROM). Thus, commercially available 8k bit single packages can be used. Moduli $m_i \leq 16$ can be implemented in 1k ROMs. For example, Figure 3.1 illustrates a residue multiplier for the modulus 31, followed by a residue adder to implement the function $\left|\left|a \times b\right|_{31} + \left|c \times d\right|_{31}\right|_{31}$. Since each residue can be represented by a maximum of 5 bits, the total of the 2 inputs to each look-up table is 10 bits and the output is taken from 5 of the 8 output bits. One of the advantages of structures using ROM arrays is the possibility of easy pipelining for high speed throughput [11]. This is also illustrated in Figure 3.1. The output of each ROM is stored in a latch and becomes a part of the address for the next ROM. The only control function required is a latch pulse. For every latch pulse new input is accepted and a new output result is generated. The throughput rate of the array is equal to the inverse of the ROM access time plus latch settling time. If we use currently available 8k PROMs, such as the 63 RA 883 Shottky PROM [14], incorporating edge triggered D registers, the throughput rate is in excess of 14 MHz.

The look-up table approach can provide great savings for hardware if some of the operands are fixed. The constant can be premultiplied or

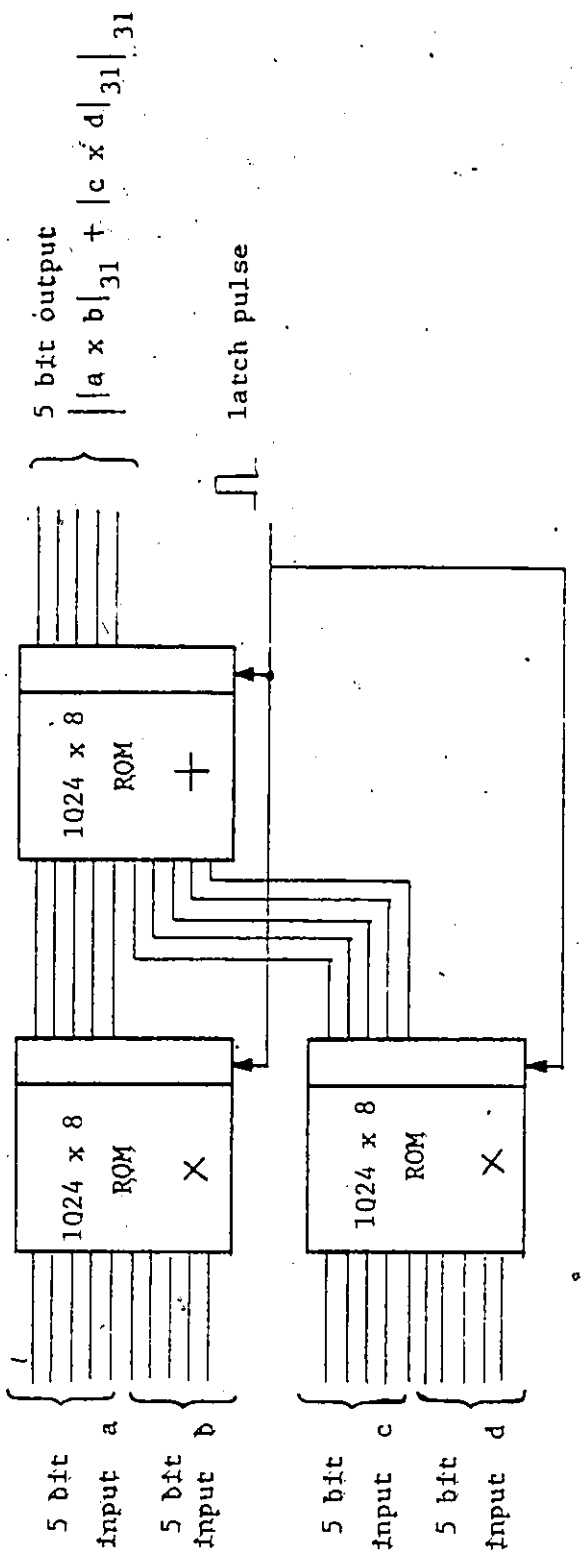


Figure 3.1 Pipelining Array

added, and a result stored along with the binary operation being implemented. If we restrict the modulus for a maximum size of 32, then the following moduli, given in descending order, are relatively prime (to preserve the nonredundant structure of the RNS) and can provide a maximum dynamic range of 1.03×2^{47}.

i	1	2	3	4	5	6	7	8	9	10	11
m_i	32	31	29	27	25	23	19	17	13	11	7

For larger moduli, the package count required for each modulus increases rapidly; however, an efficient implementation using a submodular approach has been proposed recently [11].

We can compute addition using ROM arrays for a composite modulus $u_i \cdot v_i > 2m_i$. Calculations are performed in Z_{u_i} and Z_{v_i} and the result is reconstructed, with overflow correction. Multiplication for a prime modulus can be performed using addition of indices [11].

3.1.2 Microprocessor implementation

The residue structure can also take advantage of recent advances in microprocessor hardware. Two techniques - ROM array and microprocessor array implementation-offer trade offs between the cost and the speed of operation. Again high precision implementation can be obtained from arrays of small word size microprocessors operating with residue arithmetic instead of slower double precision or multibyte arithmetic required for useful digital signal processing applications.

Instead of storing all possible outcomes of the residue operation, we can consider the use of single chip microprocessors to store complete algorithms within each modulus. Addition and subtraction (mod m_i) are easily programmed in machine language. Using 8 bit microprocessors, we

can compute modulo m_i addition for $m_i \leq 256$; to compute the addition, modulo m_i, we simply add the generalized end around carry, ie, $2^8 - m_i$, to the result of addition, if overflow is detected.

As an example, a subroutine for addition mod 251 of numbers in the accummulator and register B for Intel 8085 system is as follows:

Mnemonic	No. of cycles
ADD B	4
JC FIRST	10
CPI 251	7
JC NEXT	10
FIRST, ADI 5	7
NEXT,	38

For the modulus $m_i \leq 127$ the program can be simplified as follows:

Mnenomic	No. of cycles
ADD B	4
CPI 127	7
JC NEXT	10
SUI 127	7
NEXT,	28

It is also possible to use a mixture of binary addition and look-up table storage, for $m_i \leq 127$. For implementation of multiplication modulo a prime we can use the isomorphism between the multiplicative group, with multiplication mod m_i, and the additive group of indices, with addition mod $m_i - 1$.

The implementation of the residue number system addition and multiplication using an array of microprocessors has been discussed in [11] and illustrated with the examples of microprocessor routines for a

single chip 8 bit microcomputer from the Intel 8048 series.

3.2 Techniques for Binary to Residue and Residue to Binary Conversion

In order to interface digital structures using the residue number system to conventional digital systems it is required to have available efficient residue input and output converters.

For both ROM array and microprocessor array implementations, the conversion is best handled in a coder and decoder based on the concepts discussed in next two sections. A new, efficient, scheme for translation of the residue coded output into a binary representation, is one of the contributions of this research [15], and is discussed in section 3.2.2. A residue to analog translation technique has been discussed in [16].

3.2.1 Binary to residue encoder

The binary to residue parallel encoder for L moduli can easily be implemented with L ROMs. The i-th residue of a number X is obtained from

$$x_i = \left| \sum_{j=0}^{B-1} b_j \, 2^j \right|_{m_i} \qquad (3.1)$$

where $(b_0, b_1, \ldots, b_{B-1})$ are the B bits of the binary representation of X.

Assuming that we have available ROMs with 2^B addressable locations, then the conversion can take place in one look-up cycle. Thus, for example, if we wish to convert a 10 bit number into the residue form, then for $m_i < 256$, L 8k ROMs (organized as 1k x 8 bits) can be used. For the ROM array implementation, when we allow one of the moduli to be an even number the storage can be reduced. For example, as shown in Figure 3.2, for two moduli in the range $m_1 < 15$ and $m_2 \leq 31$, m_1 even,

only one 8k ROM is required for conversion of a 10 bit number into a residue form.

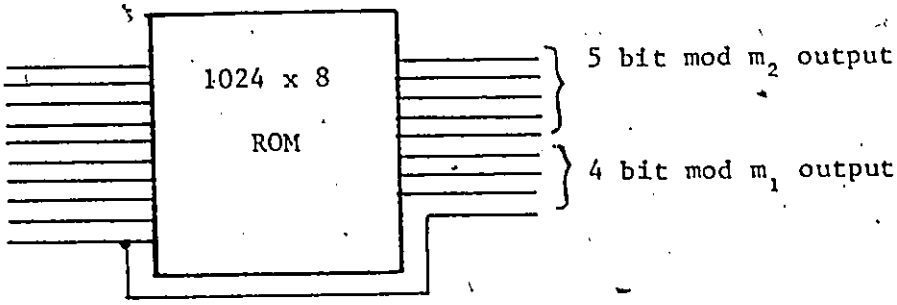


Figure 3.2 Binary to residue converter with restriction for m_1 to be even

In general, for m_1 even, we can directly use (bypass the table) one or more least significant bits as a consequence of the following. Let 2^v denote the smallest power of 2 such that $2^v > m_1$.

The residue of a number $X = \sum_{j=0}^{B-1} b_j 2^j$, $b_j \in \{0,1\}$, is generated according to

$$|X|_{m_1} = \left| \sum_{j=0}^{B-1} \left| 2^j \right|_{m_1} b_j \right|_{m_1}$$

Let $(c_v, \ldots, c_1, c_0)$ denote the binary representation of $|x|_{m_1}$.
If $m_1 = 2^v-2$, then $\left|2^j\right|_{2^v-2} = 2^s$, $s \in \{0, 1, \ldots, v-1\}$. The value of s can be determined from:

$$s = |j|_v + \beta \ , \quad \beta = \left[\frac{j}{v} \right] \ , \text{ for } s \leq v-1;$$

otherwise s is obtained iteratively, $s^{(r)} = \left|s^{(r-1)}\right|_v + \left[\frac{s^{(r-1)}}{v} \right]$ with $s^{(0)} = |j|_v + \beta$. The iteration is complete when $s^{(r)} = s^{(r-1)}$.

It is clear that $s = 0$ is possible only for $j = 0$. Therefore $c_0 = b_0$ and we only have to store the function

$$\left| \sum_{j=1}^{B-1} \left| 2^j \right|_{m_1} b_j \right|_{m_1} \cdot 2^{-1} .$$

If $m_1 = 2^v - 4$, then $\left| 2^j \right|_{2^v - 4} = 2^t$; $t \in \{0, 1, \ldots, v-1\}$.

The value of t can be determined from

$$t = |j|_v + 2\beta \qquad \text{for } t \leq v - 1 ;$$

otherwise t is obtained iteratively; $t = |t|_v + 2\theta$, $\theta = \left[\frac{t}{v} \right]$.

In this case, $t = 0$ only for $j = 0$ and $t = 1$ only for $j = 1$. Therefore $c_0 = b_0$ and $c_1 = b_1$ and we only need to store

$$\left| \sum_{j=2}^{B-1} \left| 2^j \right|_{m_1} b_j \right|_{m_1} \cdot 2^{-2} .$$

Thus, for example, only one 8k ROM is necessary for conversion of a 10 bit number into the residues modulo $m_1 = 28$ and $m_2 = 31$.

For large values of B, if the storage is required to be reduced, the binary representation can be split into a summation of smaller binary representations, eg.,

$$x_i = \left| \left| \sum_{j=0}^{B/2-1} b_j \, 2^j \right|_{m_i} + \left| \sum_{j=B/2}^{B-1} b_j \, 2^j \right|_{m_i} \right|_{m_i} \tag{3.2}$$

In this case the coding operation generally requires 3L ROMs and 2 look-up cycles. If high throughput rate is not a primary concern, the binary to residue encoder presented by Jenkins [3] can be built. The data bits are processed serially, but the storage is reduced.

3.2.2 Residue to binary decoder

The conversion of residue digits $(x_0, \ldots, x_{L-1})$ into a weighted binary representation is more difficult than the encoding into the residue form.

In [3] Jenkins presents a method of translating the residue samples into a natural integer that is based on the Chinese Remainder Theorem:

$$X = \left| \sum_{i=0}^{L-1} \hat{m}_i \left| \frac{x_i}{\hat{m}_i} \right|_{m_i} \right|_M \tag{3.3}$$

where $\hat{m}_i = \frac{M}{m_i}$; $M = \prod_{i=0}^{L-1} m_i$, $X \in [0,M)$

and $\left| \frac{1}{\hat{m}_i} \right|_{m_i}$ is the multiplicative inverse of $\left|\hat{m}_i\right|_{m_i}$.

Hardware implementation of the decoder based on this theorem and

the bit slice technique of Peled and Liu [17] generally will require L shift registers, 2^L ROM storage locations and a modulo M adder-shifter network. L look-up tables are also required for multiplication of x_i by the parameters $\left|\frac{1}{\hat{m}_i}\right|_{m_i}$. In the decoder scheme discussed in [3] for an FIR filter realization, these parameters are premultiplied by filter coefficients, and the results stored as a modified filter function.

If the filter coefficients are to be changed dynamically, as required in adaptive filtering or multiplexing schemes, L extra look-up tables are required for decoding. The modulo M adder shifter can be designed using techniques presented in [3]; however, it requires logic circuitry for detecting the forbidden states and performing the necessary correction.

The problems associated with the modulo M adder can be avoided if an alternate technique for decoding is used. This is discussed in the next section.

3.2.2.1 A mixed radix decoding technique

The decoding technique developed here and published in [15] is based on the mixed radix conversion. Any number X in the range [0,M) can be uniquely represented [2] in a special mixed-radix form:

$$X = \sum_{i=0}^{L-1} a_i p_i \tag{3.4}$$

where $p_o \triangleq 1$, $p_i = \prod_{k=0}^{i-1} m_k$,

the $\{a_i\}$ are mixed radix digits with the range $0 \leq a_i < m_i$ and the $\{p_i\}$ are the mixed-radix weights.

The mixed-radix digits can be easily generated from the residue digits, as shown later in this section. Therefore, the decoded output

is obtained from (3.4).

The multiplication in equation (3.4) can be eliminated by applying the bit slice technique of Peled and Liu [17], as in [3].

Since each mixed radix digit can be represented by

$$a_i = \sum_{j=0}^{B-1} 2^j a_{ij} \tag{3.5}$$

equation (3.4) can be written with the order of summation interchanged:

$$X = \sum_{j=0}^{B-1} 2^j \sum_{i=0}^{L-1} a_{ij} p_i \tag{3.6}$$

The inner summation requires 2^L storage locations, as in [3], however savings are obtained through a shorter wordlength. Memory contents for stored output functions are given in Table 3.1 for the same set of moduli as in [3], i.e., {19,23,29,31}.

Table 3.2, [3], shows the memory contents if the decoder implementation is based on the Chinese Remainder Theorem. In this case the stored function consist of all linear combinations (mod M) of the $\hat{m}_i$'s.

It can be seen that, for a mixed radix decoding technique, a maximum of 14 bits are required with an average of 10 bits, compared to a maximum of 16 bits and an average of 15 bits for the Chinese Remainder Theorem implementation.

In general, the linear combinations of p_i's always have shorter wordlengths than the linear combinations of $\hat{m}_i$'s.

a_{3j}	a_{2j}	a_{1j}	a_{0j}	Inner Summation
0	0	0	0	0
0	0	0	1	$p_0 = 1$
0	0	1	0	$p_1 = 19$
0	0	1	1	$p_0 + p_1 = 20$
0	1	0	0	$p_2 = 437$
0	1	0	1	$p_2 + p_0 = 438$
0	1	1	0	$p_2 + p_1 = 456$
0	1	1	1	$p_2 + p_1 + p_0 = 457$
1	0	0	0	$p_3 = 12673$
1	0	0	1	$p_3 + p_0 = 12674$
1	0	1	0	$p_3 + p_1 = 12692$
1	0	1	1	$p_3 + p_1 + p_0 = 12693$
1	1	0	0	$p_3 + p_2 = 13110$
1	1	0	1	$p_3 + p_2 + p_0 = 13111$
1	1	1	0	$p_3 + p_2 + p_1 = 13129$
1	1	1	1	$p_3 + p_2 + p_1 + p_0 = 13130$

TABLE 3.1 Memory contents for decoder based on a mixed radix conversion technique for the moduli set (19,23,29,31)

Address	Memory Contents
0 0 0 0	0 0 0 0 0
0 0 0 1	2 0 6 7 7
0 0 1 0	1 7 0 8 1
0 0 1 1	3 7 7 5 8
0 1 0 0	1 3 5 4 7
0 1 0 1	3 4 2 2 4
0 1 1 0	3 0 6 2 8
0 1 1 1	5 1 3 0 5
1 0 0 0	1 2 6 7 3
1 0 0 1	3 3 3 5 0
1 0 1 0	2 9 7 5 4
1 0 1 1	5 0 4 3 1
1 1 0 0	2 6 2 2 0
1 1 0 1	4 6 8 9 7
1 1 1 0	4 3 3 0 1
1 1 1 1	6 3 9 7 8

TABLE 3.2 Memory contents for decoder based on a Chinese Remainder Theory and the moduli set (19,23,29,31) [3]

The hardware realization of (3.6) is shown in figure 3.3. At each shift a new vector $\{a_{3j}, a_{2j}, a_{1j}, a_{0j}\}$ addresses the ROM, with the most significant bits leading.

The mixed radix digits can be found [4] from:

$$a_0 = x_0$$

$$a_i = \left|A^{(i)}\right|_{m_i}$$

$$\left|A^{(k+1)}\right|_{m_i} = \left|\left| A^{(k)} - a_k \right|_{m_i} \cdot \left| \frac{1}{m_k} \right|_{m_i} \right|_{m_i} \tag{3.7}$$

with $A^{(0)} = x_i$; $0 \le k \le i-1$ and therefore $1 \le i \le L-1$

or equivalently from:

$$a_i = \left| T(i,i,x_i) + \sum_{j=0}^{i-1} T(i,j,a_j) \right|_{m_i}$$

$$T(i,i,x_i) = \left| x_i \prod_{k=0}^{i-1} \left| \frac{1}{m_k} \right|_{m_i} \right|_{m_i} \tag{3.8}$$

$$T(i,j,a_j) = \left| - a_j \prod_{k=j}^{i-1} \left| \frac{1}{m_k} \right|_{m_i} \right|_{m_i}$$

Figure 3.4 shows the residue to mixed radix conversion array for L = 4. In general, the number of ROMs required for this conversion is:

$$N = (L - 1) \cdot L/2 \tag{3.9}$$

The functions stored in the look-up tables in figure 3.4 are:

$$T1 = \left| (x_1 - a_0) \cdot \left| \frac{1}{m_0} \right|_{m_1} \right|_{m_1}$$

$$T2 = \left| (x_2 - a_0) \cdot \left| \frac{1}{m_0} \right|_{m_2} \right|_{m_2}$$

$$T3 = \left| (T2 - a_1) \cdot \left| \frac{1}{m_1} \right|_{m_2} \right|_{m_2}$$

$$T4 = \left| (x_3 - a_0) \cdot \left| \frac{1}{m_0} \right|_{m_3} \right|_{m_3}$$

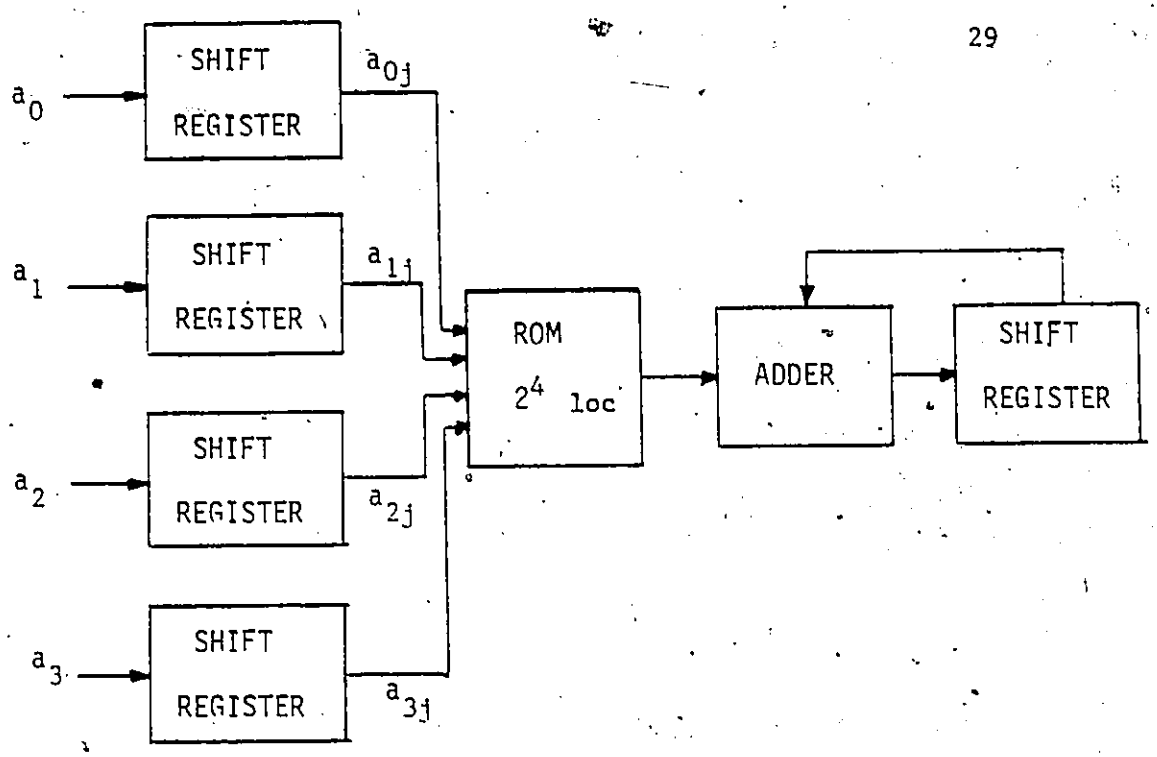


Figure 3.3 Mixed-radix to binary decoder

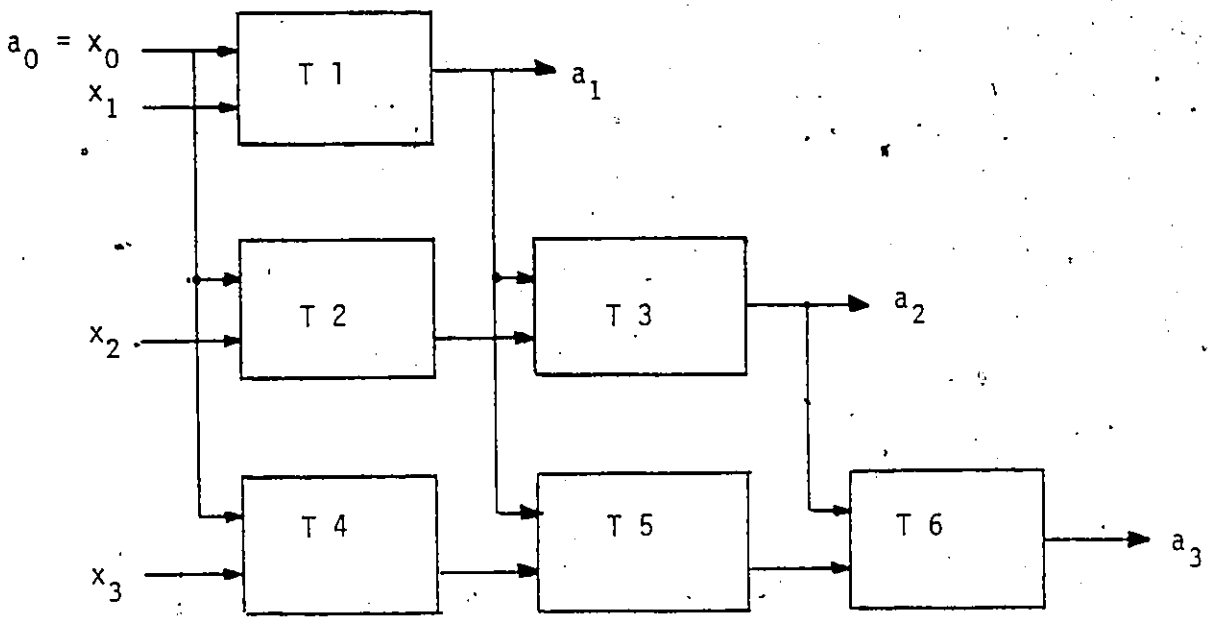


Figure 3.4 Residue to mixed radix conversion array for L = 4

$$T5 = \left| (T4 - a_1) \cdot \left| \frac{1}{m_1} \right|_{m_3} \right|_{m_3}$$

$$T6 = \left| (T5 - a_2) \cdot \left| \frac{1}{m_2} \right|_{m_3} \right|_{m_3}$$

The operation of the decoder can be illustrated by a simple numerical example.

An Example

Choose the moduli (8,5,7,3). Since $M = \prod_{i=0}^{L-1} m_i = 840$, the range of numbers that can be uniquely represented in this RNS is [-420, 419]. Let X = 127 = (7,2,1,1) be a number that is to be translated. For the mixed radix conversion, the multiplicative inverses $\left| \frac{1}{m_i} \right|_{m_j}$ are pre-computed as follows:

$$\left| \frac{1}{m_0} \right|_{m_1} = 2 \quad \left| \frac{1}{m_1} \right|_{m_2} = 3 \quad \left| \frac{1}{m_2} \right|_{m_3} = 1 \quad \left| \frac{1}{m_0} \right|_{m_2} = 1$$

$$\left| \frac{1}{m_1} \right|_{m_3} = 2 \quad \left| \frac{1}{m_0} \right|_{m_3} = 2$$

The mixed radix digits, obtained from the array in Fig.3.4 would be as follows:

$a_0 = 7 = 1\ 1\ 1$

$a_1 = 0 = 0\ 0\ 0$

$a_2 = 3 = 0\ 1\ 1$

$a_3 = 0 = 0\ 0\ 0$

The function that must be stored in ROM (figure 4) is given below:

a_{3j}	a_{2j}	a_{1j}	a_{0j}	Memory Contents	
0	0	0	0		0
0	0	0	1	P_0	= 1
0	0	1	0	P_1	= 8
0	0	1	1	$P_1 + P_0$	= 9
0	1	0	0	P_2	= 40
0	1	0	1	$P_2 + P_0$	= 41
0	1	1	0	$P_2 + P_1$	= 48
0	1	1	1	$P_2 + P_1 + P_0$	= 49
1	0	0	0	P_3	= 280
1	0	0	1	$P_3 + P_0$	= 281
1	0	1	0	$P_3 + P_1$	= 288
1	0	1	1	$P_3 + P_1 + P_0$	= 289
1	1	0	0	$P_3 + P_2$	= 320
1	1	0	1	$P_3 + P_2 + P_0$	= 321
1	1	1	0	$P_3 + P_2 + P_1$	= 328
1	1	1	1	$P_3 + P_2 + P_1 + P_0$	= 329

The retrieved contents will be the sequence 1, 41, 41, which is shifted and added to give the desired result:

$$(2 \times 1 + 41) \times 2 + 41 = 127$$

A similar approach to the design of residue decoders discussed in this section and published in [15], has been developed independently by Jenkins [16] for translation from residue into the analog form.

3.3 Summary

This section covered the implementation aspects of the Residue Number System arithmetic. Hardware implementations were described in terms of read only memory (ROM) arrays and microprocessor structures.

Techniques for binary to residue and residue to binary conversion have been discussed. A novel scheme for a residue to binary decoder based on the mixed radix conversion has been introduced.

CHAPTER 4

FINITE IMPULSE RESPONSE DIGITAL FILTERING USING FAST NUMBER THEORETIC TRANSFORMS

4.1. Introduction

Finite impulse response (FIR) digital filters produce an output based on a weighted sum of present and past inputs. This type of filter is inherently stable; however, the large number of additions and multiplications required for direct implementation of FIR filters limits the speed and efficiency.

Jenkins [3] presented a technique for direct implementation of FIR filters, using the Residue Number System as a way to increase computational speed. Indirect filtering using the Discrete Fourier Transform, with an FFT type algorithm, provides, in many cases, an improvement in computational efficiency; however, this technique introduces large quantization errors that can cause degeneration in the filter response.

Recently it has been established that transforms which can be used to indirectly compute convolution, and so may be useful for digital filtering, can be generalized to rings other than the complex numbers. Generalized DFTs defined over finite algebraic systems (see Chapter 2) are named Number Theoretic Transforms (NTTs). The NTT has the same form as the DFT with the exception that it is computed over a finite ring or field rather than over the field of complex numbers:

$$W_f = \sum_{t=0}^{N-1} w_t \, \alpha^{ft}$$

As with the DFT the 'generator' α is an N^{th} root of unity. For the case of NTT, this root is defined in the finite field or ring.

Because the NTT has the same structure as the DFT, we may use standard FFT type algorithms providing N is highly composite (eg. a power of 2), to compute the transform efficiently. Because of the 'exact' nature of the computations, NTTs eliminate any round-off error due to internal multiplications and truncation of irrational coefficients, that plague the DFT.

Within useful finite algebraic systems, various types of finite rings and fields that support NTTs have been studied in the literature, and the results will be reviewed in section 4.1.4.

Initial reported work on the implementation of NTTs considered finite fields over which the operations of addition and multiplication are computed modulo a prime. By choosing primes which allow easy binary implementation of modulo addition and multiplication, together with a simple binary form for the generator, the computation of the transform can be done easily with standard binary hardware and with an effective elimination of general multiplication. Such an approach produces severe limitations on the possible order, N, of the transform (always less than the theoretically allowed value), and there is an inherent coupling of the order with the dynamic range. In order to remove these limitations, it is essential that we have more choice over possible moduli and generators by removing restrictive hardware considerations.

This can be achieved if we consider RNS architectures of ROM or microprocessor arrays. Using the residue number system rather than conventional binary arithmetic, the moduli and generators can be selected freely to maximize the transform length. Also such an approach allows more flexibility in the choice of the dynamic range, by computing transform in parallel modulo several primes, m_i, so that dynamic range

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

However, if the NTT is defined in the residue class rings (direct sum of several finite fields of integers modulo m_i, $GF(m_i)$ (see Chapter 2), as discussed in section 4.1:5, the main problem is that for the small component moduli required for efficient RNS implementation the compatible transform length is impractically small. In order to reconcile this conflict, we can compute parallel transforms in several extension fields, $GF(m_i^n)$. This work considers mainly NTTs defined over several Galois Fields of second degree, $GF(m_i^2)$, since we claim that this degree of extension offers the best trade off between the efficiency of implementation (number of arithmetic operations required) and attainable power of two transform length, N.

For comparative purposes, some aspects of implementing NTTs in extension fields of higher degree than 2 are considered in section 4.5.

In the following section an in-depth study is undertaken to determine all possible Galois Fields $GF(m_i^2)$ which allow:

i) favorable transform length, such as powers of 2

ii) efficient RNS implementations; viz. transform parameters and irreducible polynomials are determined so that the number of binary operations required for multiplication of field elements is minimized.

The guidelines for selecting transform parameters are provided for two distinct RNS implementations, one using arrays of ROMs and other using arrays of microprocessors.

Two particular applications of NTTs are discussed separately.

i) convolution of two successive blocks of real data when

$GF(m_i^2)$ is a field of finite quadratic integers.

ii) convolution of two complex-valued sequences over a finite field of Gaussian integers.

4.1.1. Fast convolution

The computation of finite digital convolution

$$y(s) = \sum_{t=0}^{P-1} h(s-t)\, w(t) \tag{4.1}$$

symbolically denoted

$$y(s) = h(s) * w(s)$$

has extensive applications in digital processing of signals, eg. in implementation of finite impulse response (FIR) filters, computation of auto and cross correlation, polynomial multiplication. With the growing number of applications, efficient implementation of digital convolution is a matter of increasing importance.

Digital convolution can be implemented either directly, in the time domain, or with transforms, T, that have the cyclic convolution property (CCP). The cyclic convolution property can be stated as:

$$T[h(s) \circledast w(s)] = T[h(s)] \cdot T[w(s)]$$

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

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

$$y'(s) = T^{-1}\{T[h(s)] \cdot T[w(s)]\} \tag{4.2}$$

The convolution implemented by (4.2) is called circular (or cyclic) convolution:

$$y'(s) = h(s) \circledast w(s) = \sum_{t=0}^{N-1} h(|s-t|_N) \cdot w(t) \quad \dagger \qquad (4.3)$$

The results of finite and circular convolution, y and y', are equal if zeros are appended to w(s) and h(s) to prevent folding or aliasing [18], so that transform length is at least equal to K+P-1; K and P are durations of {w(s)} and {h(s)} respectively. Long input sequences filtered by an FIR filter whose kernel is relatively short can be divided into blocks and conventional overlap-add or overlap-save techniques [18] can be used to compute the output signal from the results of circular convolutions.

The technique of convolving two finite duration sequences, using transform techniques, has been called fast convolution, as opposed to the direct evaluation of equation 4.1, which is called direct convolution.

The term fast is used, because the transform can be evaluated rapidly and efficiently by the Fast Fourier Transform (FFT) algorithm, first introduced by Cooley and Tukey [19]. Many versions of the FFT algorithm have been described in detail ([18], [20] and references). The FFT algorithm is independent of the ring or field over which the transform is defined [21], although it has been first introduced for the field of complex numbers. The efficiency of computation depends on the degree of compositness of the sequence length N; the transform is simplest to compute when N is highly factorizable, eg. $N = 2^B$ for a radix-2 algorithm.

4.1.2 Number Theoretic Transform

Number Theoretic Transforms (NTT's) were discovered independently

---

† Because of the periodicity, the index is evaluated modulo N

by a number of researchers around 1970-1971 ([21] -[23]) as a generalization of the standard Discrete Fourier Transform (DFT) with respect to the ring over which the transform is defined. The underlying algebraic structure for a large class of NTTs is a finite ring or field and the basis for generalization is the CCP.

The general form of transformation which maps circular convolution of length N sequences, over ring R [†], into a term-by-term product is given by

$$T: \quad W(f) = \sum_{t=0}^{N-1} w(t)\, \alpha^{ft} \qquad f = 0, 1, \ldots, N-1 \tag{4.4}$$

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

The inverse transform has the form:

$$T^{-1}: \quad w(t) = N^{-1} \sum_{f=0}^{N-1} W(f)\, \alpha^{-ft} \qquad t = 0, 1, \ldots, N-1 \tag{4.5}$$

with the resitiction that N^{-1} belongs to the ring. Over the infinite complex field, the DFT with $\alpha = e^{-j2\pi/N}$ is the only transform having the cyclic convolution property. The complex field can support transforms of any length. If α, w(t) and W(f) are elements of a finite algebraic system, a wide choice of possible Number Theoretic Transforms exists depending upon the choice of the finite field or ring, the transform length and the root of unity. Unlike the DFT, NTTs are used only to compute convolutions, since the transform domain does not have any known practical value. Moreover, the transformed sequence W(f) depends on the choice of α, which of course is fixed throughout. An important advantage of NTT algorithms is that the indirect conputation of convol-

---

[†] term "ring" means a commutative ring with identity

ution is exact. That is, after the quantization of the input data and the filter coefficients, no additional quantization noise is introduced into the filtering process. Since the DFT involves irrational coefficients (sines and cosines) thereby making exact computation impossible on a digital processor, the NTT may be of great value when the sequences must be convolved exactly.

In finite arithmetic we are dealing with integers. When working with digitals signals we can assume without any loss of generality that data from the A/D converter are treated as integer numbers.

Care must be taken, however, that the final result does not overflow. The maximum output must be known and bounded by M; this analysis is aided in most digital filtering applications by a priori knowledge of the impulse response. Dynamic range constraints will be discussed in section 4.3.

Unlike the DFT, NTTs do not allow arbitrary transform lengths. Their maximum attainable transform length, N, depends upon the choice of the ring, or field. This is discussed in the next section.

4.1.3 Conditions for Existence of NTTs

Nicholson [21] has presented the algebraic theory of the generalized transform with a DFT structure in commutative rings $\hat{R}$ with identity and without zero divisors (ie, an integral domain). Following Nicholson, we say that necessary and sufficient conditions for $\hat{R}$ to support a generalized DFT of length N are:

1) that α is a primitive root of unity in $\hat{R}$, ie,
   $\alpha^N = 1$, $\alpha^j \neq 1$, $j = 1, \ldots, N-1$.
   If we denote by $\hat{R}^{\circledast}$ a multiplicative (cyclic) group of R, α is any generator of $\hat{R}^{\circledast}$ of order N.

2) that the multiplicative inverse of N, N^{-1}, belongs to $\hat{R}$.

The conditions for NTTs to exist over Galois Fields $GF(m^n)$, discussed by Pollard[22], are contained in the above, since a finite field is necessarily an integral domain. Cyclic group of $GF(m^n)$ has order m^n-1, so transform length can be any divisor of m^n-1. The transform in a finite field of integers modulo m, m prime, is a special case of the above, and its length is a factor of m-1. More general results for the existence of NTTs over finite commutative rings are presented in [26]. Let R be a finite commutative ring with identity. Then R decomposes uniquely as a direct sum of local rings †. R_i,

$$R \simeq R_1 \oplus \cdots \oplus R_L \tag{4.6}$$

Under this decomposition, an element $r \in R$ has L-tuple representation $(r_1, \ldots, r_L)$. R supports a generalized DFT of length N if and only if

1) each R_i contains a primitive N^{th} root of unity α_i

2) N^{-1} exists in R.

A primitive N^{th} root of unity in R_i is any generator of a multiplicative group of order N. If I_i is the maximal ideal of R_i, then R_i/I_i is a finite (Galois) field of $m_i^{n_i}$ elements [46]. Thus for any finite ring R a necessary and sufficient condition can be derived, that R supports a length N generalized DFT if and only if $N \mid \widetilde{\gcd}\ (m_i^{n_i} - 1,\ i = 1, \ldots, L)$. The same criteria have recently been obtained [67] for direct sum of Galois rings to support a generalized DFT.

4.1.4 State of the art review

Recently, various authors have proposed the use of Number Theoretic Transforms over different finite fields, or rings, for error free and

---

†A local ring is a commutative ring with identity which as a unique maximal ideal I (any element $a \notin I$ is invertible in local ring)

fast, efficient computation of cyclic convolution. The theoretical framework of transforms using number theoretic and algebraic properties of data has been provided for any integral domain [21], any finite field [22]and any finite ring [26].

Practical considerations dictate a selection of ring/fields that support transforms whose parameters lead to efficient implementation of modular arithmetic, either in hardware or software. Most of the reported work on Number Theoretic Transforms has supposed that the hardware will be implemented using the binary number system. In the conventional binary arithmetic, residue reduction is particularly easy when the modulus is of the form $2^K \pm 1$. The choice 2^K does not admit useful transforms since N=1. Also in order to simplify multiplications in the binary number system, the cyclic group generator α is chosen as a power of 2. When this constraint has to be fulfilled, the transform length is usually not a maximum, theoretically allowed length in a given field.

In addition, the transform length N should be highly composite so that a high speed convolution algorithm exists. A considerable effort has been made to provide rings and fields which allow for adequate dynamic range and satisfy the above constraints, so that the conflicts hidden in these constraints are alleviated. In particular, Rader [28] proposed transforms defined in the ring of integers modulo Mersenne numbers, $M = 2^p - 1$, p prime. These transforms are referred to as Mersenne Number Transforms (MNT). In the ring of integers, modulo a Mersenne number, 2 is a p-th root of unity and -2 is a 2p-th root of unity. Thus the disadvantage of this "multiplication-free" MNT is that it precludes the use of an FFT-type algorithm since the order of the transform is not a power of 2 and not even highly composite. Rader [28]

and Aggarwal and Burrus [24] proposed to compute the NTT with a modulus of the form of the t^{th} Fermat number, $F_t = 2^b + 1$, $b = 2^t$, referred to as the Fermat Number Transform (FNT). Fermat numbers up to F_4 are primes. In [24] it has been shown that an FNT with $\alpha = 2$ allows a transform length $N = 2^{t+1}$ and an FNT with $\alpha = 2^{2^t-2}\left(2^{2^t-1} - 1\right)$ (known as $\sqrt{2}$, since $\alpha^2 \equiv 2 \pmod{F_t}$) allows $N = 2^{t+2}$. Thus an FFT type algorithm can be used. The hardware implementation of a 64 point FNT is described in [27].

However, the main disadvantage of the MNT and the FNT is the rigid relationship between the dynamic range and attainable transform length. For example, with a 32 bit word machine using $F_5 = 2^{32} + 1$, $N = 64$ for $\alpha = 2$ and $N = 128$ for $\alpha = \sqrt{2}$. There is also a limited choice of possible wordlengths. This point is especially significant when a FNT is used and may results in a mismatch of wordlength and dynamic range required for the particular convolution, because of the large spacing between Fermat Numbers.

Aggarwal & Burrus [24] have also considered the case of an NTT with modulus $2^b + 1$ with $b \neq 2^t$. However, these moduli are never prime numbers [45, p.58], so this approach is of limited interest because of small transform lengths (section 4.1.3).

In [29], [30] Nussbaumer introduced pseudo-Fermat and pseudo-Mersenne number transforms. A pseudo-MNT is defined modulo an integer $M_1 = (2^p - 1)/q$, p composite and q some factor of $2^p - 1$, and pseudo-FNT is defined modulo an integer $M_2 = (2^b + 1)/s$, $b \neq 2^t$ and s some factor of $2^b + 1$. Because $2^p - 1$ and $2^b + 1$, defined as above are not prime, the corresponding transform would have a short length. Thus, if $2^b + 1$ and $2^p - 1$ contain small factors, these can be divided out in the pseudo-

Mersenne or pseudo-Fermat numbers to yield a longer transform length. The arithmetic implementation of the above transforms can be performed using arithmetic modulo 2^p-1 or 2^b+1, followed by a final reduction modulo M_1 or M_2.

Next in order of arithmetic complexity, using the conventional binary system are the moduli of the form $2^n \pm 2^m \pm 1$. An NTT over a field built with this type of prime has been briefly discussed in [31] and [32].

The method of computing convolutions in the residue class rings/fields has been extended to a more general setting that includes so-called complex-valued transforms. Structural properties of complex residue rings are discussed in [34] and [35]. Implementation considerations, associated with these complex Number Theoretic Transforms, have been considered for special types of moduli, such as Mersenne primes [28], [35], [36], [37], and Fermat numbers [38]. In some cases a restriction has been made to consider only complex residue fields or constructions such as a direct sum of these fields [36].

Other rings have also been considered for the construction of NTTs, such as the ring of Eisenstein integers [26] and the generalization of complex residue rings known as finite algebraic rings of quadratic integers [39]; algorithms have also been obtained for the implementation of arithmetic modulo Fermat numbers.

4.1.5 NTT in the residue class rings

Initial approaches to the implementation of Number Theoretic Transforms for convolution of real data concentrated on transformers defined over a ring of integers modulo M. The NTT in the residue class ring is illustrated by the transform pair:

$$X(f) = \left| \sum_{t=0}^{N-1} x(t)\, \alpha^{tf} \right|_M \qquad 0 \le f \le N-1$$

$$x(t) = \left| N^{-1} \sum_{f=0}^{N-1} X(f)\, \alpha^{-tf} \right|_M \qquad 0 \le t \le N-1 \tag{4.7}$$

where $\left|\alpha^N\right|_M = 1$; $x, X, \alpha \in Z_M$

The modulus M should be large enough to prevent overflow, since the components of circular convolution are required to remain in the interval

$$-\frac{M}{2} \le y < \frac{M}{2} \qquad \text{for M even}$$

$$-\frac{M-1}{2} \le y \le \frac{M-1}{2} \qquad \text{for M odd}$$

Instead of using a single large modulus transform, the transform can be computed modulo several distinct primes $\{m_i\}$, and the result, modulo M, reconstructed according to the Chinese Remainder Theorem. This procedure is equivalent to defining the NTT in the residue number system. In the RNS, the mapping δ: $\delta(X) = (x_1, \ldots, x_L)$ represents the isomorphism of Z_M onto the direct sum of rings $Z_{m_i^{e_i}}$,

$$\delta: Z_M \simeq Z_{m_1^{e_1}} \oplus Z_{m_2^{e_2}} \oplus \cdots \oplus Z_{m_L^{e_L}}$$

where $m_1^{e_1}, \ldots, m_L^{e_L}$ are the distinct prime power factors of M. This works because the binary operation $\square \in (+, \cdot)$ is preserved under the mapping δ:

$$\delta(X \square Y) = (x_1, \ldots, x_L) \square (y_1, \ldots, y_L) = \delta(X) \square \delta(Y)$$

The idea of a composite NTT, where transforms are computed in residue class rings $Z_{m_i^{e_i}}$, and the result modulo M is obtained in the final stage, has already been suggested [25], [33], [40].

Aggarwal and Burrus have derived necessary and sufficient conditions for N to be a possible transform length in Z_M, viz. that

$$N \mid \gcd(m_1-1, \ldots, m_L-1).$$

The same transform length can be obtained if we restrict the Galois rings, $\{Z_{m_i^{e_i}}\}$, to be Galois fields, $\{Z_{m_i}\}$. With this restriction, the implementation becomes very efficient.

Jenkins [40] extended the results of Aggarawal and Burrus, so that N divides $\min(m_1-1, \ldots, m_L-1)$, by allowing different transform lengths, N_i, modulo each prime, such that $N_i = m_i-1$. Since the original block length is N, (N_i-N) zeros must be appended to the transformed sequences modulo m_i. This approach leads to additional flexibility by allowing the choice of more prime moduli at the expense of the inefficiency of padding the component transforms with zeros.

However, even with this relaxation for the NTT defined in the residue class rings of integers, the power of two transform length is severely limited. The solution to this problem is found by computing the transform in extension fields, as discussed in section 4.2.

4.2. Transforms Over a Direct Sum of Galois Fields of m_i^2 Elements

In [22] Pollard has shown that transforms of the form (4.4, 4.5), defined over the Galois fields of m_i^n elements, $GF(m_i^n)$ where m_i is a prime, possess the CPP. The transform length N divides m_i^n-1 and α is a generator of an N element cyclic, multiplicative, subgroup in $GF(m_i^n)$. To construct $GF(m_i^n)$ we need to determine an irreducible polynomial of degree n over the base field Z_m. A polynomial of degree n of the form $f(x) = \sum_{i=0}^{n} a_i x^i \in Z_m[x]$ with $a_i \in Z_{m_i}$ and $a_n \neq 0$ is defined to be

irreducible [41] if it cannot be expressed as a product of two polynomials of positive degree over Z_{m_i}.

The quotient field $Z_{m_i}[x] \,/\, (f(x))$ is the required Galois field with m^n elements. Up to isomorphism, the field $GF(m_i^n)$ depends only on the degree of polynomial and not on its particular form [41]. Moreover, suppose that λ is a root of $f(x)$, $\lambda \in GF(m_i^n)$, hence we have the isomorphism $Z_{m_i}(\lambda) \simeq Z_{m_i}[x] \,/\, (f(x))$.

Addition and multiplication in $GF(m_i^n)$ is defined as polynomial addition and multiplication, followed by polynomial residue reduction modulo $f(x)$. The elements of $GF(m_i^n)$ can be written as polynomials of degree n-1 with coefficients in $GF(m_i)$ or as n-tuples of digits, each restricted to $GF(m_i)$. Hereafter polynomial notation will be used, to avoid confusion with L-tuple representation of elements of direct sum of fields or rings.

In the following sections, attention will be restricted to the Galois fields of second degree, ie, to the case $n = 2$. The definition of the NTT over $GF(m_i^2)$ (from Pollard [22]) is:

$$T_{GF(m_i^2)}: \; U_i(f) = \sum_{t=0}^{N-1} u_i(t)\, \alpha_i^{ft} \qquad f = 0, \ldots, N-1 \qquad (4.8)$$

where N divides m_i^2-1; $U_i(f)$, $u_i(t)\, \alpha_i \in GF(m_i^2)$ and α_i is a generator of an N element multiplicative cyclic subgroup in $GF(m_i^2)$.

Hence, transforms defined in $GF(m_i^2)$ allow greatly increased sample lengths over those defined in $GF(m_i)$; the maximum length is larger than the square of the maximum attainable length in the field of integers modulo m_i. Moreover, radix 2 transform lengths are reasonably large, a

fact of great convenience when implementing a fast transform algorithm.

The inverse transform is

$$T^{-1}_{GF(m_i^2)} : u_i(t) = N^{-1} \sum_{f=0}^{N-1} U_i(f) \cdot \alpha^{-ft} \tag{4.9}$$

Since N is a non-zero element of $GF(m_i^2)$, the multiplicative inverse, N^{-1}, exists in $GF(m_i^2)$. To demonstrate the invertibility of $T_{GF(m_i^2)}$, ie, that (4.8) implies (4.9), and the convolution property of $T_{GF(m_i^2)}$, observe first that if, and only if, α is a root of order N, we have the following:

$$\alpha^{N_j} - 1 = 0 \tag{4.10}$$

which can be factored as:

$$(\alpha^j - 1) \sum_{f=0}^{N-1} \alpha^{fj} = 0 \tag{4.11}$$

Therefore

$$\begin{aligned} &\sum_{f=0}^{N-1} \alpha^{fj} = N \text{ if } j \equiv 0 \pmod N \\ &\sum_{f=0}^{N-1} \alpha^{fj} = 0, \text{ otherwise} \end{aligned} \tag{4.12}$$

since for $j \not\equiv 0$, $\alpha^j - 1 \neq 0$.

Substituting (4.8) into (4.9) and using (4.12) yields:

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

$$= N^{-1} \cdot u(t) \cdot N = u(t).$$

The proof of the convolution property is a simple deduction from (4.12):

Let $U(f) = \sum_{t=0}^{N-1} u(t) \cdot \alpha^{tf}$,

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

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

Then, by (4.9), the inverse transform of $Y(f)$ is $y(s) = N^{-1} \sum_{f=0}^{N-1} U(f) \cdot H(f) \cdot \alpha^{-fs} = N^{-1} \sum_{f=0}^{N-1} \sum_{t=0}^{N-1} \sum_{v=0}^{N-1} u(t)\, h(v)\, \alpha^{f(v+t-s)} =$

$N^{-1} \sum_{t=0}^{N-1} u(t) \cdot h(s-t) \cdot N = \sum_{t=0}^{N-1} u(t)\, h(s-t)$. †

From section 3.1 it is clear that the residue arithmetic can be efficiently implemented if we use relatively small integers for each modulus, and generate the required dynamic range by combining a sufficient number of moduli. Thus computing the transform in a finite ring which is a direct sum of several Galois fields of second degree, $GF(m_i^2)$, $i = 1, \ldots, L$,

$$R \simeq GF(m_1^2) \oplus \ldots \oplus GF(m_L^2) \qquad (4.13)$$

increases the dynamic range to $\prod_{i=1}^{L} m_i$. The conditions for the transform length, N, have to be restated as follows. Since for each i, α_i must be a primitive N^{th} root of unity in $GF(m_i^2)$, $N \mid \gcd(m_i^2-1)$, $i = 1,\ldots,L$. The above is the special case of general conditions for the existence of NTTs in the direct sum of local rings [26], and is a result of the requirement that equations similar to (4.11) and (4.12) still hold for

---

† note that multiplication and addition is modulo N, hence this is the cyclic convolution property.

$\alpha \in R$, namely that

$$\sum_{f=0}^{N-1} \alpha^{fj} = N \quad \text{if} \quad j \equiv 0 \pmod{N}$$
$$\sum_{f=0}^{N-1} \alpha^{fj} = 0 \quad \text{otherwise} \tag{4.14}$$

Since α has the representation $(\alpha_1, \ldots, \alpha_L)$, $\alpha_i \in GF(m_i^2)$, conditions specified by (4.14) are possible only if each α_i is an N^{th} root of unity, therefore if N divides each m_i^2-1. Once we choose the moduli m_i that allow a suitable transform length, N, we are left with two problems to be solved, in order that the RNS implementation of the NTT be efficient; namely

i) we wish to construct the Galois field, $GF(m_i^2)$, in such a way that the multiplication and addition of field elements will require the smallest possible number of operations.

ii) we would like to search for the generator of an N-element cyclic subgroup in $GF(m_i^2)$, α, that has the simplest form possible, so that the number of operations required for multiplications by powers of α is minimized.

The complexity of multiplication of field elements is determined by the structure of the irreducible polynomial f(x). Let $f(x) = px^2 + qx + s$ $(p,q,s \in GF(m_i^2))$ be an irreducible polynomial of degree 2 over $GF(m_i)$. Then, the extension field in which the given polynomial has a root, denoted by λ, may be described by:

$$GF(m_i^2) = \{a + b\lambda \mid a, b \in GF(m_i)\} \tag{4.15}$$

Addition in $GF(m_i^2)$ is component wise, with the coefficients reduced modulo m_i, whilst multiplication has to be followed by the polynomial residue reduction to eliminate λ^2. It can be readily verified that the

number of binary operations required to perform the multiplication of field elements is minimized if we can find an irreducible polynomial $f(x)$ which is monic and has the binomial form, ie, $f(x) = x^2 + s$; for notational convenience, we can write the desired irreducible polynomial as $f(x) = x^2 - r$. The above statement can be illustrated by means of the example.

Example: Suppose that for some prime, m, the irreducible polynomial of degree 2 has the form $f(x) = x^2 + x + 1$. Let ω be a root of $f(x) = 0$. This polynomial is irreducible for example for $m = 5$. Then,

$$Z_5(\omega) \simeq GF(5^2) = \{a + b\omega \mid a, b \in Z_5\}$$

The field of the form $\{a + b\omega\}$ with ω a root of $\omega^2 + \omega + 1 = 0$ is sometimes referred to as the field of Einstein integers. The elements of $GF(5^2)$ are:

$$0,\ 1,\ 2,\ 3,\ 4,\ \omega,\ 2\omega,\ 3\omega,\ 4\omega,\ 1 + \omega,\ 2 + \omega,\ 3 + \omega,\ 4 + \omega,$$
$$1 + 2\omega,\ 2 + 2\omega,\ 3 + 2\omega,\ 4 + 2\omega,\ 1 + 3\omega,\ 2 + 3\omega,\ 3 + 3\omega,\ 4 + 3\omega,$$
$$1 + 4\omega,\ 2 + 4\omega,\ 3 + 4\omega,\ 4 + 4\omega.$$

Addition is component wise. For multiplication we perform the usual multiplication of polynomials, yielding

$$(a + b\omega)(a' + b'\omega) = aa' + \omega(ab' + a'b) + \omega^2 bb'$$

Now, to satisfy the field axions (field is closed under multiplication) we have to perform residue reduction modulo $\omega^2 + \omega + 1$ (polynomial division) and restore the coefficients so that they are elements of Z_5. Hence the multiplication is defined as:

$$(a + b\omega)(a' + b'\omega) = \left|aa' - bb'\right|_m + \omega\left|ab' + a'b - bb'\right|_m$$

Multiplication of field elements requires 5 binary multiplications and 3 binary additions. This is obviously not a very efficient operation. To illustrate the more efficient approach, let the irreducible polynomial of degree 2 have the form

$$f(x) = x^2 - r; \quad r \in Z_{m_i} \tag{4.16}$$

Then the extension field, in which the given polynomial has a root, may be described by

$$GF(m_i^2) = \{a + \lambda b \mid a, b \in Z_{m_i};\ \lambda^2 - r = 0\}$$

where $\lambda = \sqrt{r}$

Now, addition and multiplication are defined by:

$$(a + b\lambda) + (a' + b'\lambda) = \left|a + a'\right|_{m_i} + \left|b + b'\right|_{m_i} \lambda \tag{4.17}$$

$$(a + b\lambda) \cdot (a' + b'\lambda) = \left|aa' + rbb'\right|_{m_i} + \left|ab' + a'b\right|_{m_i} \lambda \tag{4.18}$$

It is interesting to observe that the residue reduction mod $(\lambda^2 - r)$ is particularly simple since $\lambda^2 = r$. The number of binary operations in the multiplication of field elements is now reduced to 4 multiplications and 2 additions. There is also one multiplication by a constant, r, but in the RNS implementation, using ROMs, this does not require separate hardware.

The multiplicative inverse can easily be verified to be:

$$(a + b\lambda)^{-1} = \left| (a - b\lambda) \cdot \left| \frac{1}{a^2 - rb^2} \right|_{m_i} \right|_{m_i}$$

$$= \left| a \cdot c \right|_{m_i} + \left| -b \cdot c \right|_{m_i} \lambda \tag{4.19}$$

where c denotes the multiplicative inverse of $\left|a^2 - rb^2\right|_{m_i}$

For the polynomial of degree 2 to be irreducible it is sufficient that f(x) has no root in $GF(m_i)$. This statement also holds for degree 3; however, it does not remain valid for degree 4 and higher.

Therefore, we have to establish the conditions under which the congruence

$$x^2 \equiv r \pmod{m_i} \tag{4.20}$$

is not solvable.

It is known [42, p.113] that the two-term congruence

$$x^n \equiv r \pmod{m_i} \tag{4.21}$$

is solvable if and only if ind r is a multiple of the gcd $(n, \phi(m_i))$. If the congruence (4.21) has solutions, then r is said to be an n-th power residue, otherwise r is said to be an n-th power non-residue. In particular, for n = 2 the residues on non-residues are said to be quadratic; for n = 3, cubic; for n = 4, biquadratic. Congruences of second degree have either two solutions or none [42, p.92] depending on whether r is a quadratic residue or a quadratic non-residue, modulo m.

Hence, the necessary and sufficient condition for non-solvability of the congruence (4.20) is that ind r is not a multiple of 2 (since we assume that m_i is an odd prime).

The following two sections will provide the techniques required for searching for suitable values of the quadratic non-residue r for different types of primes. It will be shown, that under certain conditions, the generator α of the multiplicative group of $GF(m_i^2)$ can have a simplified form; for other cases it is always possible to choose r = -1, which is the most convenient value for general implementation. If r = -1, $GF(m_i^2)$ is isomorphic to the residue class of complex, so-called Gaussian integers. We now present a new proof using indices that this holds only for primes of the form $m_i = 4\xi+3$. We wish to find a modulus m_i such that the field $GF(m_i)$ contains no square root of -1, ie,

$$x^2 \equiv -1 \pmod{m_i} \tag{4.22}$$

has no solution.

Let τ be the index of x, and σ be the index of -1.

Then

$$2\tau \equiv \sigma (\mathrm{mod}(m_i - 1)) \tag{4.23}$$

A solution will only exist for σ divisible by 2, since $\left|2^{-1}\right|_{m_i-1}$ does not exist. Conversely, the congruence has no solution (-1 is a quadratic non-residue) for the index of -1 odd. In order to find the index of -1, we use the congruence $(-1)^2 \equiv 1 \pmod{m_i}$. The index of 1 is $m_i - 1$, hence the index of -1, σ, is given by $2\sigma \equiv m_i - 1 \pmod{(m_i - 1)}$. It follows that the congruence (4.22) has no solution if $\sigma \equiv \frac{m_i - 1}{2}$ is odd; ie, if the modulus m is in the form $m_i = 4\xi + 3$. Conversely, if σ is even, ie, $m_i = 4\xi + 1$, we find 2 values of index τ which are incongruent modulo $(m_i - 1)$. Corresponding to these values we find two values of $\sqrt{-1}$, λ_1 and λ_2, which are incongruent modulo m_i with $\lambda_2 = m_i - \lambda_1$. It follows that when $m_i \equiv 1 \pmod 4$, $\sqrt{-1}$ may be considered as a member of $GF(m_i)$ and hence we cannot construct Galois field of second degree using polynomial $x^2 - 1$. For example if m = 5, $\sqrt{-1}$ is congruent modulo 5 to 2 and 3.

Having established the best structure of the irreducible polynomial, we are left with the problem of finding suitable transform factors in $GF(m_i^2)$, namely the pair N and α. The algorithm for finding the generator α of the multiplicative subgroup of order N in $GF(m_i^2)$ is developed in two following sections. General statements about transform parameters of the convenient form are presented; the primes of the form $4\xi + 1$ and $4\xi + 3$ are considered separately. Clearly all primes except the prime 2, which obviously has no value for implementation of NTT, are congruent modulo 4 either to 1 or 3.

4.2.1 Searching for transform factors in $GF(m_i^2)$ for primes of the form $4\xi + 3$.

It has been shown that for primes of the form $4\xi + 3$, $r \equiv -1 \pmod{m_i}$ is quadratic non-residue. Transforms in $GF(m_i^2)$ can be used to compute convolutions on complex data or convolutions on two blocks of real data. Transforms over $GF(m_i^2)$ for arbitrary quadratic non-residue r, $r \neq -1$, can only be used to compute convolutions on two blocks of real data.

The implementation of a transform is simplest when the transform length is a power of 2. Let $N = 2^B$ be the order of generator α, in $GF(m_i^2)$, such that B is the largest possible integer given that $N \mid (m_i^2 - 1)$.

The prime, $m_i = 4\xi + 3$, can be represented as:

$$m_i = q \cdot 2^p - 1 \tag{4.24}$$

where p is any positive integer and $(q, 2) = 1$.

The following theorem gives the maximum value for B.

Theorem 4.1:

Given a base field Z_{m_i} and an irreducible polynomial, $x^2 - r$ over $GF(m_i)$ [x], the extension field, $Z_{m_i}(\sqrt{r})$, has a cyclic subgroup of order $N = 2^B$. The maximum value of B is $p + 1$.

Proof

The order of a cyclic subgroup has to divide $m_i^2 - 1$. Therefore $N \mid (q^2 2^{2p} - q\, 2^{p+1})$. Since q is odd, $N \mid \psi 2^{p+1}$ where ψ is odd. Hence we can always find an $N = 2^B$ where the maximum valueof B is $p + 1$. For example, the prime 47 can be represented as $3 \cdot 2^4 - 1$. From the above theorem we can immediately determine that the maximum power of 2 transform length in $GF(47^2)$ is $2^5 = 32$. We will show that the generator

of order 2^B has to have the general polynomial form, ie, $\alpha = \gamma + \sqrt{r}\,\beta$ with $\gamma, \beta \neq 0$ for Max [B] $= p + 1$.

If $\beta = 0$, α has maximum multiplicative order $m_i - 1 < 2^{p+1}$; hence $\beta \neq 0$. Let $\alpha = \gamma + \sqrt{r}\,\beta = (c + \sqrt{r}\,d)^q$ have order 2^{p+1}. Then $(c+\sqrt{r}d)^{q2^p} = -1$, since $\alpha^{N/2} = -1$. The above can be written as

$$(c + \sqrt{r}\,d)(c + \sqrt{r}\,d)^{q2^p - 1} = -1 \tag{4.25}$$

If we employ the binomial theorem† and the property $r^{\frac{m_i-1}{2}} = -1$ [42], hence $\sqrt{r}^{\,m_i} = -\sqrt{r}$, equation (4.25) can be written as:

$$(c + \sqrt{r}\,d)(c - \sqrt{r}\,d) = -1 \text{ or } c^2 - rd^2 = -1$$

If we want $c = 0$, $rd^2 = 1$ has to have a solution; ie, $\text{ind}(r^{-1})$ has to be a multiple of 2. Since $\text{ind}(r^{-1}) = m_i - 1 - \text{ind } r$, then ind r also has to be a multiple of 2; a contradiction. It follows that $c \neq 0$, so $\gamma \neq 0$.

The next problem is to choose the quadratic nonresidue r, subject to the condition (ind r, 2) = 1 (tables of indices can be found in many books on Number theory, eg., in [42]). For RNS implementation, using ROM arrays, the choice for the value of r is not important, since there is no extra hardware required for the multiplication by a constant; instead of storing the function $b \cdot b'$ in (4.18), we store $r \cdot b \cdot b'$. However, for microprocessor implementation, it is more efficient to choose as r a number with a minimal binary weight. Since, for primes congruent to 3(mod 4), -1 is always a quadratic non-residue, the most suitable choice is $r \equiv -1 \pmod{m_i}$. A computer program has been written for searching for elements, of order N in $GF(m_i^2)$, of the form:

---

† The binomial theorem as applied to elements of $GF(m_i^2)$ is presented in Appendix B.

$$\alpha = \gamma + \sqrt{-1}\,\beta; \quad \gamma, \beta \in GF(m_i) \tag{4.26}$$

The search can be facilitated by noting that

i) we need to evaluate only N/2 powers of α, because

$$\alpha^{N/2} \equiv -1 \pmod{m_i}$$

ii) the search can be stopped when we find an element α' whose order N' is a multiple of N; ie, if $N' = \mu N$, then

$$\alpha = (\alpha')^{\mu}$$

There are $\phi(N)$ possible elements $\alpha = \gamma + \sqrt{r}\,\beta \in GF(m_i^2)$ of order N for each value of a quadratric non-residue r. This is an obvious extension of the following well known result for integers (eg. [43, p.75]). If a is an element of order D, then a^{ρ} has order D if and only if $(\rho, D) = 1$.

Although this theorem is dedicated to primitive D-th roots in $GF(m_i)$, the same proof readily holds and the same result is obtained when we consider elements of $GF(m_i^2)$.

In particular, there are 2^p possible values of generator $\alpha = \gamma + \sqrt{-1}\,\beta \in GF(m_i^2)$ of order $N = 2^{p+1}$.

Example: Let $m_i = 7 = 2^3 - 1$. The maximum radix 2 transform length over $GF(7^2)$ is $N = 2^{p+1} = 16$. Let $r \equiv -1 \pmod 7$; then one of the possible generators of the cyclic subgroup of order 16 in $GF(7^2)$ is $\alpha = 2 + \sqrt{-1}\,3$. All elements of this cyclic subgroup are listed below:

$\alpha = 2 + \sqrt{-1}\,3$

$\alpha^2 = 2 + \sqrt{-1}\,5$

$\alpha^3 = 3 + \sqrt{-1}\,2$

$\alpha^4 = 0 + \sqrt{-1}\,6$

$\alpha^5 = 3 + \sqrt{-1}\,5$

$\alpha^6 = 5 + \sqrt{-1}\,5$

$\alpha^7 = 2 + \sqrt{-1}\ 4$

$\alpha^8 = 6 + \sqrt{-1}\ 0$

$\alpha^9 = 5 + \sqrt{-1}\ 4$

$\alpha^{10} = 5 + \sqrt{-1}\ 2$

$\alpha^{11} = 4 + \sqrt{-1}\ 5$

$\alpha^{12} = 0 + \sqrt{-1}\ 1$

$\alpha^{13} = 4 + \sqrt{-1}\ 2$

$\alpha^{14} = 2 + \sqrt{-1}\ 2$

$\alpha^{15} = 5 + \sqrt{-1}\ 3$

$\alpha^{16} = 1$

Among the elements listed above there are exactly 8 possible generators of order 16, namely α, α^3, α^5, α^7, α^9, α^{11}, α^{13}, α^{15}. In the example above, it can be noticed that $\alpha^7 \equiv \alpha^{*}$ [†]. This is a general property of any element in $GF(m_i^2) = \{a + \sqrt{r}\ b;\ x^2 - r = 0\}$ that $(a + \sqrt{r}\ b)^{m_i} = a - \sqrt{r}\ b$. This result is proved in Appendix B.

4.2.2 Searching for transform factors in $GF(m_i^2)$ for primes of the form $4\xi + 1$.

We can represent $m_i = 4\xi + 1$, ξ any integer, as $m_i = 2^k\ell + 1$, ℓ odd. The largest possible radix 2 transform length in $GF(m_i^2)$ is $N = 2^{k+1}$. For primes of this form, we find the remarkable property that the generator of order 2^{k+1}, can have the simple form, $\alpha = \sqrt{r}$, where $x^2 - r$ is an irreducible binomial in $GF(m_i^2)$. This property is obtained from the following theorem.

Theorem 4.2

Let $m_i = \ell 2^k + 1$, $(\ell, 2) = 1$, be an odd prime number. Then:

[†] An asterisk in superscript form implies a conjugate operation

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

ii) If g is as in (i), then $\sqrt{g}$ has multiplicative order $\ell 2^{k+1}$ in $GF(m_i^2)$, where $GF(m_i^2) = \{a + b\sqrt{g} : a, b \in GF(m_i)\}$.

iii) We can find a generator $\sqrt{r}$, of a cyclic subgroup of order 2^{k+1} in $GF(m_i^2)$, where $r = g^{\rho\ell}$ with $(\rho, 2) = 1$ and $x^2 - r$ an irreducible polynomial in $GF(m_i)[x]$.

Proof:

i) Suppose $x^2 - g$ is reducible. Then, there exists $d \in GF(m_i^2)$ such that $d^2 = g$. From Fermat's theorem [43] $d^{m_i - 1} = 1 = d^{2\left\{\frac{m_i - 1}{2}\right\}} = g^{\frac{m_i - 1}{2}} = -1$. This is not the case for an odd prime, hence $x^2 - g$ is irreducible.

ii) $(\sqrt{g})^{\ell 2^{k+1}} = g^{\ell 2^k} = 1$. We now have to prove that $\ell 2^{k+1}$ is the smallest order of $\sqrt{g}$. Suppose $(\sqrt{g})^s = 1$ for some $1 \leq s < \ell 2^{k+1}$, then if $2 \mid s$, so that $s = 2v$, $1 \leq v < \ell 2^k$, we have $(\sqrt{g})^{2v} = g^v = 1$ which is impossible. If $2 \nmid s$, then $s \mid \ell$; but $s = vn + y$, $0 \leq y < n$, $n = \ell 2^k$, and so $g^y = g^{s - vn} = g^s (g^n)^{-v} = 1$. Therefore $y = 0$ and $s = vn$. Because $m_i - 1 = \ell 2^k$ is even then $k > 0$. Hence $2 \mid n$ and so $2 \mid s$, a contradiction.

iii) From (ii) $\left[\sqrt{g}\right]^{\ell 2^{k+1}} = 1$, therefore $\left[\sqrt{g}\right]^{\ell}$ has order 2^{k+1}. Since ℓ is odd, $(\sqrt{g})^{\ell} \notin GF(m_i)$ and so there exists an irreducible polynomial $x^2 - g^{\ell}$. We also know [43] that for some ρ, such that $(\rho, 2^{k+1}) = 1$, $(\sqrt{g})^{\rho\ell}$ has order 2^{k+1}. It is clear (from the prime factor decomposition of 2^{k+1}) that we only need to show $(\rho, 2) = 1$. We now set $r = g^{\rho\ell}$.

Since $\rho\ell$ is odd, x^2-r is an irreducible polynomial in $GF(m_i)[x]$.

Example: Let the prime from the progression $4\xi + 1$ be $m_i = 97 = 3 \cdot 2^5 + 1$. Maximum radix 2 transform length over $GF(97^2)$ is $N = 2^6 = 64$. From the tables of primitive roots (eg. [42]) it can be found that for prime 97, $g = 5$.

According to the Theorem 4.2, $\sqrt{5}$ will generate cyclic subgroup of order 192, and the generator of the multiplicative order 64 is given by $\alpha = \sqrt{r} = (\sqrt{5})^{3\rho}$ where $(\rho, 2)$. Arbitrarily we can choose $\alpha = \sqrt{28}$ for $\rho = 1$ or $\alpha = \sqrt{19}$ for $\rho = 27$. Tables of indices are helpful, since we are searching for an element r subject to the constraint $ind_5 r = 3 \cdot \rho$. As an illustration of this example, powers of $\alpha = \sqrt{19}$ are listed in Table 4.1

It is evident that the number of binary operations required for multiplication of an arbitrary element in $GF(m_i^2)$, $u = a + b\sqrt{r}$, by the powers of generator α with a simple form obtained according to the Theorem 4.2, $\alpha = \sqrt{r}$, is now significantly reduced.

Multiplication by even power of α, ie, by $\alpha^j = n$, requires only two binary multiplications. Multiplication by odd powers $\alpha = \theta\sqrt{r}$ requires two binary multiplications and one multiplication by a fixed operand r. This brings the potential for significant savings in the implementation of the transform.

TABLE 4.1. Elements $\alpha^j = \eta + \theta\sqrt{19}$ Generated By $\alpha = \sqrt{19}$ Over $GF(97^2)$

j	η	θ
1	0	1
2	19	0
3	0	19
4	70	0
5	0	70
6	69	0
7	0	69
8	50	0
9	0	50
10	77	0
11	0	77
12	8	0
13	0	8
14	55	0
15	0	55
16	75	0
17	0	75
18	67	0
19	0	67
20	12	0
21	0	12
22	34	0
23	0	34
24	64	0
25	0	64
26	52	0
27	0	52
28	18	0
29	0	18
30	51	0
31	0	51
32	96	0
33	0	96
34	78	0
35	0	78
36	27	0
37	0	27
38	28	0
39	0	28
40	47	0
41	0	47
42	20	0
43	0	20
44	89	0
45	0	89
46	42	0
47	0	42

j	η	θ
48	22	0
49	0	22
50	30	0
51	0	30
52	85	0
53	0	85
54	63	0
55	0	63
56	33	0
57	0	33
58	45	0
59	0	45
60	79	0
61	0	79
62	46	0
63	0	46
64	1	0

4.3 Design Procedures for NTT Convolution Filter for Real Data

4.3.1 The General Form for an NTT Convolution Filter Over $GF(m_i^2)$.

Let $u_i = a_i + \sqrt{r_i}\, b_i$ be an arbitrary element in $GF(m_i^2)$, m_i a prime, and $N = 2^B$ be the raddix 2 transform length over $GF(m_i^2)$, where Max [B] $= p + 1$ for $m_i \equiv 3 \pmod 4$ and Max [B] $= k + 1$ for $m_i \equiv 1 \pmod 4$. Let $\{w(t)\}$ be the input sequence that is to be convolved with the impulse response $\{h(t)\}$. Circular convolution can be implemented in blocks of length N by computing the transform of N samples of $\{w(t)\}$ and $\{h(t)\}$, multiplying the transforms and taking the inverse transform. Since we are using the NTT over $GF(m_i)$ for convolution of real sequences, two successive blocks of the input sequence can be transformed simultaneously via the same NTT by setting:

$$u_i(t) = w(t) + \sqrt{r_i}\, w(t + N) \tag{4.27}$$

which can be written in the residue form as

$$u_i(t) = \left| w(t) \right|_{m_i} + \sqrt{r_i}\, \left| w(t + N) \right|_{m_i}$$

where $u_i(t) \in GF(m_i^2)$. The result of the circular convolution associated with the elements $\{1, \sqrt{r_i}\}$, will yield the circular convolution of two successive blocks of $\{|w(t)|_{m_i}\}$ with $\{|h(t)|_{m_i}\}$:

$$\left| y(s) + \sqrt{r}\, y(s+N) \right|_{m_i} = \left| \sum_{t=0}^{N-1} h(|s-t|_N) \cdot u(|t|_N) \right|_{m_i}$$

$$= \left| \sum_{t=0}^{N-1} h(|s-t|_N) \cdot \{w(|t|_N) + \sqrt{r}\, w(|t|_N + N)\} \right|_{m_i} \qquad (4.28)$$

The convolution filter, using an NTT over $GF(m_i^2)$, is illustrated in figure 4.1.

To compute the convolution unambiguously, the components of the circular convolution sum in a single Galois field $GF(m_i)$, are required to have an upper bound m_i; ie, signed numbers should remain in the interval $-\frac{m_i-1}{2} \le y \le \frac{m_i-1}{2}$. The absolute upper bound on the input data and the impulse response is

$$\max |w| \cdot \max |h| \le \frac{m_i-1}{2N}.$$

This bound on the dynamic range is pessimistic for many practical applications and if the impulse response is known, and fixed, it is enough to have

$$\max |w| \le \frac{m_i-1}{2 \sum_{t=0}^{N-1} |h(t)|} \qquad (4.29)$$

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

$$\max w \le \frac{m_i-1}{\sum_{t=0}^{N-1} |h(t)|}.$$

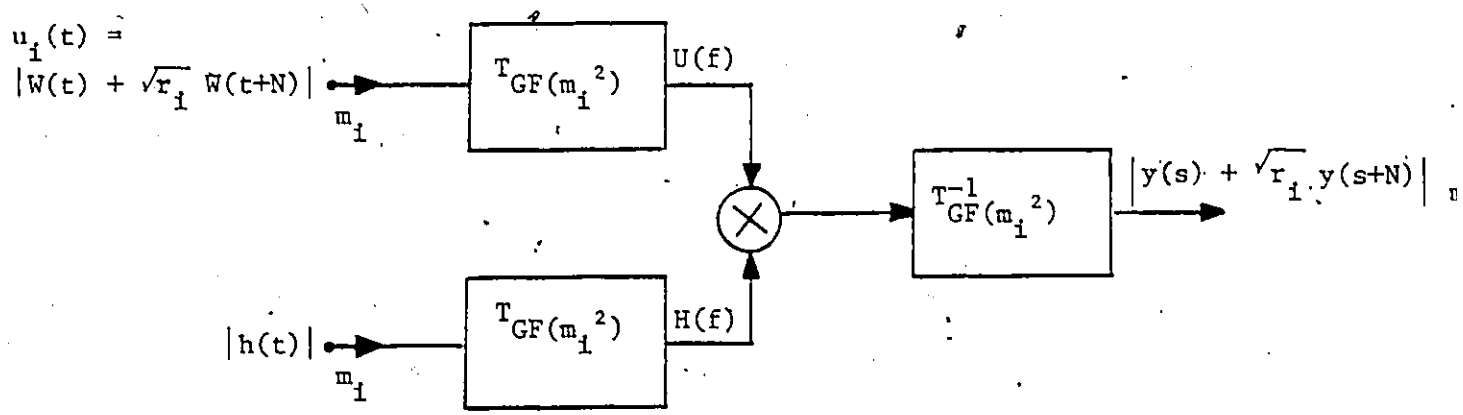


Figure 4.1. Convolution filter over $GF(m_i^2)$ for convolution of real data

4.3.2 A Computer Simulation of Convolution in $GF(m_i^2)$

The proposed convolution filter using NTT over $GF(m_i^2)$ has been simulated by computer and the program is given in Appendix C.

The NTT algorithms, consisting only of addition, subtraction and multiplication of elements in $GF(m_i^2)$, defined in section 4.2, can be easily simulated by using integer arithmetic followed by a residue reduction. The Fortran statement

IA = MOD (IA, M)

is used to compute the residue $|IA|_M$.

As an example of the operation of the program, two rectangular pulse trains were convolved with the impulse response defined by

$$h(t) = \begin{cases} 0 & 0 \le t \le 31 \\ 1 & 32 \le t < 64 \end{cases} \tag{4.30}$$

The prime, 97, was chosen as the modulus. As previously determined in section 4.2.2, the maximum power of 2 transform length is N = 64 and one of the possible values for the generator α is $\sqrt{19}$. The upper bound on the input function can be computed from 4.29, max w = 3. Arbitrarily we have chosen w = 1 in order to illustrate the convolution process as simply as possible. Figure 4.2 shows the input sequence, impulse response, their transforms and the result of the convolution. Dotted lines indicate the dynamic range constraints. Overlapping by 32 points and adding y(s) with y(s+N) will give the result of convolution, $\hat{y}(s)$, of two blocks of the input sequence with h(t) given by (4.30):

$$\hat{y}(s) = \begin{cases} y(s), & 0 \le s < 32 \\ y(s) + y(s+N-32), & 32 \le s < 63 \\ y(s+N-32), & 63 \le s < 96 \end{cases}$$

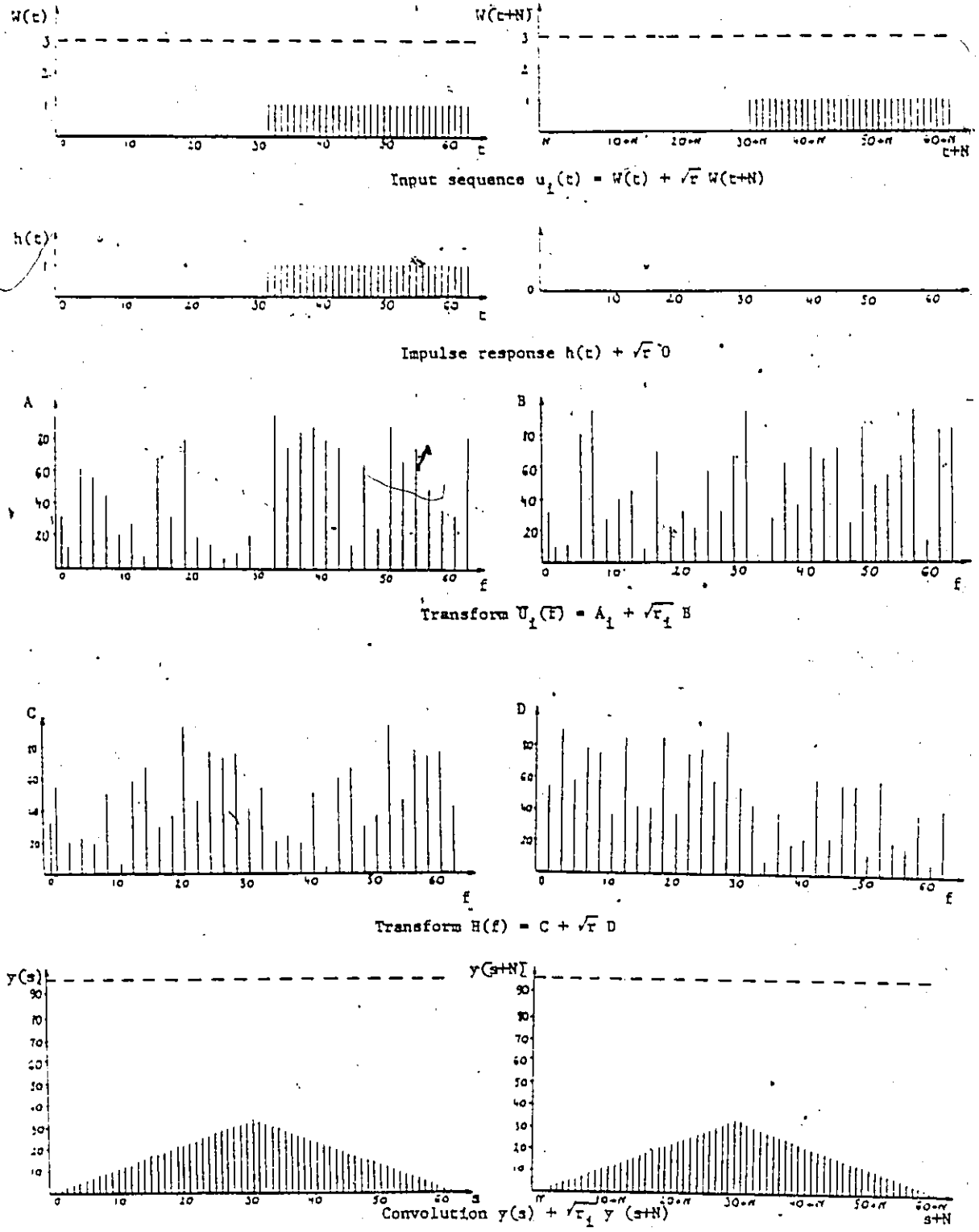


Figure 4.2 Convolution of two rectangular pulses using NTT over $GF(97^2)$

The values of the transforms of the input sequence, U(f), and the impulse response, H(f), are shown in Table 4.2. As an illustration that transform domain sequences depend on the choice of the generator α, Table 4.3 lists the values of the transforms U'(t) and H'(f) of the same input and the impulse response as above, but obtained with another generator; $\alpha' = \sqrt{28}$ (see Section 4.2.2). The result of the convolution is identical in both cases.

f	U(f) = A + $\sqrt{r}$ B		H(f) = C + $\sqrt{r}$ D	
	A	B	C	D
0	32	32	32	0
1	13	11	54	54
2	0	0	0	0
3	62	12	20	89
4	0	0	0	0
5	58	81	23	58
6	0	0	0	0
7	45	96	18	78
8	0	0	0	0
9	20	28	50	75
10	0	0	0	0
11	27	40	3	37
12	0	0	0	0
13	6	46	59	84
14	0	0	0	0
15	69	10	66	41
16	0	0	0	0
17	32	70	29	41
18	0	0	0	0
19	80	23	36	84
20	0	0	0	0
21	19	32	92	37
22	0	0	0	0
23	15	23	45	75
24	0	0	0	0
25	7	58	77	78
26	0	0	0	0
27	10	33	72	58
28	0	0	0	0
29	20	67	75	89
30	0	0	0	0
31	0	95	41	54
32	0	0	0	0
33	95	0	54	43
34	0	0	0	0

35	75	28	20	8
36	0	0	0	0
37	85	62	23	39
38	0	0	0	0
39	88	37	18	19
40	0	0	0	0
41	80	72	50	22
42	0	0	0	0
43	76	63	3	60
44	0	0	0	0
45	15	72	59	13
46	0	0	0	0
47	63	25	66	56
48	0	0	0	0
49	26	85	29	56
50	0	0	0	0
51	89	49	36	13
52	0	0	0	0
53	68	55	92	60
54	0	0	0	0
55	75	67	45	22
56	0	0	0	0
57	50	96	77	19
58	0	0	0	0
59	37	14	72	39
60	0	0	0	0
61	33	83	75	8
62	0	0	0	0
63	82	84	41	43

TABLE 4.2. Transform Sequences for $m_1 = 97$, $N = 64$ and $\alpha = \sqrt{19}$ (Fig. 4.2)

f	$U'(f) = A' + \sqrt{r}\,B'$		$H'(f) = C' + \sqrt{r}\,D'$	
	A'	B'	C'	D'
0	32	32	32	0
1	74	72	36	36
2	0	0	0	0
3	14	2	77	22
4	0	0	0	0
5	78	78	41	37
6	0	0	0	0
7	40	34	23	11
8	0	0	0	0
9	44	1	3	95
10	0	0	0	0
11	36	5	29	73
12	0	0	0	0
13	18	96	45	51
14	0	0	0	0
15	76	30	75	52
16	0	0	0	0
17	21	72	20	52
18	0	0	0	0
19	23	4	50	51
20	0	0	0	0
21	73	42	66	73
22	0	0	0	0
23	36	90	92	95
24	0	0	0	0
25	89	83	72	11
26	0	0	0	0
27	23	91	54	37
28	0	0	0	0
29	52	40	18	22
30	0	0	0	0
31	0	95	59	36
32	0	0	0	0

33	95	0	36	61
34	0	0	0	0
35	43	55	77	75
36	0	0	0	0
37	72	4	41	60
38	0	0	0	0
39	6	12	23	86
40	0	0	0	0
41	59	5	3	2
42	0	0	0	0
43	22	53	29	24
44	0	0	0	0
45	72	91	45	46
46	0	0	0	0
47	74	23	75	45
48	0	0	0	0
49	19	65	20	45
50	0	0	0	0
51	77	96	50	46
52	0	0	0	0
53	59	90	66	24
54	0	0	0	0
55	51	94	92	2
56	0	0	0	0
57	55	61	72	86
58	0	0	0	0
59	85	17	54	60
60	0	0	0	0
61	81	93	18	75
62	0	0	0	0
63	21	23	59	61

TABLE 4.3. Transform sequences for $m_i = 97$, $N = 64$ and $\alpha = \sqrt{28}$

4.3.3. NTT convolution over direct sum of $GF(m_i^2)$.

Computing the convo lution over a finite ring isomorphic to a direct sum of L Galois Fields, increases the allowable dynamic range for the convolution sum to $M = \prod_{i=1}^{L} m_i$. The condition $y < M$ is assured if the upper bound on the absolute value of the real input data is

$$\max |W| \leq \frac{M-1}{2 \sum_{t=0}^{N-1} |h(t)|}, \quad M \text{ odd} \tag{4.31}$$

$$\text{or} \quad \max |W| < \frac{M}{2 \sum_{t=0}^{N-1} h(t)}, \quad M \text{ even}$$

To implement the NTT over a ring R

$$\pi: R \simeq GF(m_1^2) \oplus \ldots \oplus GF(m_L^2) \tag{4.32}$$

we find a moduli set, $\{m_i\}$, that can provide the same order of transform, $N = 2^B$, for each m_i. Then we have the mapping

$$\pi(W(t) + \sqrt{r}\, W(t+N)) = (W_1(t) + \sqrt{r_1}\, W_1(t+N), \ldots, W_L(t) + \sqrt{r_L}\, W_L(t+N))$$

where W_i denotes the residue of W modulo m_i. Hence, the procedure for implementing the convolution filter over direct sum of L Galois Fields is as follows:

i) form L modulo m_i sequences of two successive blocks of input signal

ii) implement L parallel filter structures as shown in Fig.4.1. Compute overlap-add, or overlap-save, on the results of the convolution sums

iii) recover the results of the convolution sum using one of the methods discussed in Chapter 3.

To provide sufficient dynamic range, $M = \prod_{i=1}^{L} m_i$, for a required transform length, it may be necessary to combine moduli of the form $4\xi + 1$ and $4\xi + 3$. For primes of the form, $m_i = 4\xi + 1$, we wish to have $N = 2^{k+1}$ because generator α, of order 2^{k+1}, has the simple form. For $m_i = 4\xi + 3$ transform length can be chosen as 2^{p+1} or any divisor of 2^{p+1} without any influence on the hardware requirements. Tables 4.4 and 4,5 list all prime numbers requiring up to 8 bits for their representation in the arithmetic progression $4\xi + 1$ and $4\xi + 3$ respectively, together with the factorization of m_i^2-1, maximum power of 2 length N in $GF(m_i^2)$ and $GF(m_i)$ for comparison.

m_i	ξ	Representation $\ell 2^k+1$	Factorization of m_i^2-1	Maximum Radix 2 Length in GF(m_i^2)	Maximum Radix 2 Length in GF(m_i)
5	1	2^2+1	$3 \cdot 2^3$	8	4
13	3	$3 \cdot 2^2+1$	$7 \cdot 3 \cdot 2^3$	8	4
17	4	2^4+1	$3^2 \cdot 2^5$	32	16
29	8	$7 \cdot 2^2+1$	$7 \cdot 5 \cdot 3 \cdot 2^3$	8	4
37	9	$9 \cdot 2^2+1$	$19 \cdot 3^2 \cdot 2^3$	8	4
41	10	$5 \cdot 2^3+1$	$7 \cdot 5 \cdot 3 \cdot 2^4$	16	8
53	13	$13 \cdot 2^2+1$	$13 \cdot 3^3 \cdot 2^3$	8	4
61	15	$15 \cdot 2^2+1$	$31 \cdot 5 \cdot 3 \cdot 2^3$	8	4
73	18	$9 \cdot 2^3+1$	$3^2 \cdot 37 \cdot 2^4$	16	8
89	22	$11 \cdot 2^3+1$	$11 \cdot 5 \cdot 3^2 \cdot 2^4$	16	8
97	24	$3 \cdot 2^5+1$	$7^2 \cdot 3 \cdot 2^6$	64	32
101	25	$25 \cdot 2^2+1$	$17 \cdot 5^2 \cdot 3 \cdot 2^3$	8	4
109	27	$9 \cdot 2^2+1$	$11 \cdot 5 \cdot 3^3 \cdot 2^3$	8	4
113	28	$7 \cdot 2^4+1$	$113 \cdot 3 \cdot 2^5$	32	16
137	34	$17 \cdot 2^3+1$	$23 \cdot 17 \cdot 3 \cdot 2^4$	16	8
149	37	$37 \cdot 2^2+1$	$37 \cdot 5^2 \cdot 3 \cdot 2^3$	8	4
157	39	$39 \cdot 2^2+1$	$79 \cdot 13 \cdot 3 \cdot 2^3$	8	4
173	43	$43 \cdot 2^2+1$	$43 \cdot 29 \cdot 3 \cdot 2^3$	8	4
181	45	$45 \cdot 2^2+1$	$13 \cdot 7 \cdot 5 \cdot 3^2 \cdot 2^3$	8	4
193	48	$3 \cdot 2^6+1$	$97 \cdot 3 \cdot 2^7$	128	64
197	49	$49 \cdot 2^2+1$	$11 \cdot 7^2 \cdot 3^2 \cdot 2^3$	8	4
229	57	$57 \cdot 2^2+1$	$23 \cdot 19 \cdot 5 \cdot 3 \cdot 2^3$	8	4
233	58	$29 \cdot 2^3+1$	$29 \cdot 13 \cdot 3^2 \cdot 2^4$	16	8
241	60	$15 \cdot 2^4+1$	$11^2 \cdot 5 \cdot 3 \cdot 2^5$	32	16

TABLE 4.4. Table of primes $m_i = 4\xi + 1$ less than 257

m_i	ξ	Representation $q2^p - 1$	Factorization of $m_i^2 - 1$	Maximum Radix 2 Length in $GF(m_i^2)$	Maximum Radix 2 Length in $GF(m_i)$
3	0	$2^2 - 1$	2^3	8	2
7	1	$2^3 - 1$	$3 \cdot 2^4$	16	2
11	2	$3 \cdot 2^2 - 1$	$5 \cdot 3 \cdot 2^3$	8	2
19	4	$5 \cdot 2^2 - 1$	$5 \cdot 3^2 \cdot 2^3$	8	2
23	5	$3 \cdot 2^3 - 1$	$11 \cdot 3 \cdot 2^4$	16	2
31	7	$2^5 - 1$	$5 \cdot 3 \cdot 2^6$	64	2
43	10	$11 \cdot 2^2 - 1$	$11 \cdot 7 \cdot 3 \cdot 2^3$	8	2
47	11	$3 \cdot 2^4 - 1$	$23 \cdot 3 \cdot 2^5$	32	2
59	14	$15 \cdot 2^2 - 1$	$29 \cdot 5 \cdot 3 \cdot 2^3$	8	2
67	16	$17 \cdot 2^2 - 1$	$17 \cdot 11 \cdot 3 \cdot 2^3$	8	2
71	17	$9 \cdot 2^3 - 1$	$7 \cdot 5 \cdot 3^2 \cdot 2^4$	16	2
79	19	$5 \cdot 2^4 - 1$	$13 \cdot 5 \cdot 3 \cdot 2^5$	32	2
83	20	$21 \cdot 2^2 - 1$	$41 \cdot 7 \cdot 3 \cdot 2^3$	8	2
103	25	$13 \cdot 2^3 - 1$	$17 \cdot 13 \cdot 3 \cdot 2^4$	16	2
107	26	$27 \cdot 2^2 - 1$	$53 \cdot 3^3 \cdot 2^3$	8	2
127	31	$2^7 - 1$	$7 \cdot 3^2 \cdot 2^8$	256	2
131	32	$33 \cdot 2^2 - 1$	$13 \cdot 11 \cdot 5 \cdot 3 \cdot 2^3$	8	2
139	34	$35 \cdot 2^2 - 1$	$23 \cdot 7 \cdot 5 \cdot 3 \cdot 2^3$	8	2
151	37	$19 \cdot 2^3 - 1$	$19 \cdot 5^2 \cdot 3 \cdot 2^4$	16	2
163	40	$41 \cdot 2^2 - 1$	$41 \cdot 3^4 \cdot 2^3$	8	2
167	41	$21 \cdot 2^3 - 1$	$87 \cdot 7 \cdot 3 \cdot 2^4$	16	2
179	44	$45 \cdot 2^2 - 1$	$89 \cdot 5 \cdot 3^2 \cdot 2^3$	8	2
191	47	$3 \cdot 2^6 - 1$	$19 \cdot 5 \cdot 3 \cdot 2^7$	128	2
199	49	$25 \cdot 2^3 - 1$	$11 \cdot 5^2 \cdot 3^2 \cdot 2^4$	16	2
211	52	$53 \cdot 2^2 - 1$	$53 \cdot 7 \cdot 5 \cdot 3 \cdot 2^3$	8	2
223	55	$7 \cdot 2^5 - 1$	$37 \cdot 7 \cdot 3 \cdot 2^6$	64	2
227	56	$57 \cdot 2^2 - 1$	$113 \cdot 19 \cdot 3 \cdot 2^3$	8	2
239	59	$15 \cdot 2^4 - 1$	$17 \cdot 7 \cdot 5 \cdot 3 \cdot 2^5$	32	2
251	62	$63 \cdot 2^2 - 1$	$7 \cdot 5^3 \cdot 3^2 \cdot 2^3$	8	2

TABLE 4.5. Table of primes $m_i = 4\xi + 3$ less than 257

Table 4.6 shows the dynamic range associated with the moduli $\{m_i\}$ which allow appreciable radix 2 transform length over $GF(m_i^2)$ and are suitable for RNS implementation using either arrays of ROMs or arrays of microprocessors.

m_i	17	31	47	97	127	191	193
$\log_2 m_i$	4.08	4.95	5.55	6.59	6.98	7.57	7.59

TABLE 4.6. Effective wordlength in bits for primes suitable for RNS implementation of NTT

The moduli m_i are chosen in such a way that for the ROM implementation the ratio of the possible length for package count is maximized and for the microprocessor implementation, the transform length is maximized. The selection of transform parameters for those two RNS structures is discussed in the two following sections.

4.3.4. ROM array implementation considerations. Hardware features and selection of transform parameters

The RNS operations can be implemented by look-up tables stored in ROMs, as discussed in Chapter 3. This approach is particularly suited to the implementation of the basic computational element in the NTT processor for the convolution filter of Figure 4.1. Moreover, hardware savings can be obtained in the inverse transform unit T^{-1}, structurally identical to T. A conventional binary implementation requires separate hardware for multiplication by the multiplicative inverse $\left|N^{-1}\right|_{m_i}$. In the look-up table approach, the contents of the ROMs used for multiplying two transforms, or the contents of ROMs in the final conversion stage, can be premultiplied by this fixed operand. The fast NTT algorithm (FNTT) is identical to the FFT algorithm of which many forms have been described in the literature. For the FNTT, the twiddle factors W^n, $W = e^{-j\frac{2\pi}{N}}$, are

replaced by the powers of generator α_i.

The organization of an NTT processor (sequential, cascade, parallel or array structure) is usually dictated by performance and cost requirements [66]. For each of these structures, RNS implementation using ROM arrays is particularly advantageous because of the inherent simplicity in pipelining the array (see section 3.1.1). If the transform processor can be implemented as an array processor, then a mixed radix algorithm can be used; this results in very high speeds and large lengths such as large factors of $m_i^2 - 1$ given in the Tables 4.4 and 4.5. However at this point in time, the cost of this approach severely limits its application.

For sequential or parallel realizations, we require the same calculation at each stage, hence we implement a single radix algorithm. If the transform length is a power of 2, $N = 2^B$, then for a minimum hardware sequential realization, radix 2 is chosen. The basic computational unit (BCU) is a radix 2 butterfly.

For a radix 2 decimation in time (DIT) algorithm the flow chart of the butterfly is shown in Figure 4.3, where $\eta + \sqrt{r}\,\theta$ denotes the n'th power of generator $\alpha = \beta + \sqrt{r}\,\gamma$ and n depends on how many steps of the FNTT algorithm have been executed. The radix 2 DIT butterfly repeatedly performs the computation

$$A + \sqrt{r}\,B = a + \sqrt{r}\,b + (a' + \sqrt{r}\,b')\cdot(\eta + \sqrt{r}\,\theta)$$

$$C + \sqrt{r}\,D = a + \sqrt{r}\,b - (a' + \sqrt{r}\,b')\cdot(\eta + \sqrt{r}\,\theta) \qquad (4.33)$$

with addition and multiplication on elements in $GF(m_i^2)$ as defined in equation (4.17) and (4.18). Hence, the usual complex arithmetic is now replaced by finite field arithmetic. When a radix 2 decimination-in-frequency (DIF) is implemented, the only structural difference between

the DIT and DIF butterfly is whether the multiplication by α^n is before or after the 2 point transform.

Figure 4.4 illustrates the array of look-up tables storing the results of addition and multiplication modulo m_i for a radix 2 DIT BCU, when the modulus is of the form $4\xi + 3$. In this case $\alpha = \beta + \sqrt{-1}\,\gamma$, hence its power has the form $\eta + \sqrt{-1}\,\theta$. The number of look-up-tables in BCU (not counting the stored tables for α) is 10 for one modulus.

It has been shown in section 4.2.2, that for moduli of the form $4\xi + 1$, the generator α has simple form, $\alpha = \sqrt{r}$. Hence, the multiplication by even power of α requires only 2 general multiplications and by odd powers of α 2 general multiplications and one multiplication by a constant r. The BCU arrays for even and odd powers of α are illustrated in Figure 4.5 and 4.6 respectively. The look-up table requirement is 6 in both cases. It is evident that the hardware implementation of an NTT in $GF(m_i^2)$, $m_i = 4\xi + 1$, is almost as efficient as in $GF(m_i)$, whereas the attainable transform length is increased significantly.

A conservative data throughput of the BCU is four real samples per 70 nsec; (see section 3.1.1) this assumes that the pipeline is always full. For $m_i < 32$ the look-up table has maximum size 1024 x 5 bits and can be stored at one commercially available 8K ROM. For larger m_i, required for practical NTTs, efficient implementation of multiplication and addition using submodular approach has been developed by Jullien[11], as referred to in section 3.1.1.

The technique has been described in detail in [11] and also in [44]. Table 4.7 shows the package count required for multiplication or addition (not counting the stored tables for α) for several prime moduli. If addition follows multiplication, then savings can be made in the number

of tables so that the total package count is less than twice the package count obtained from Table 4.7.

Modulus	Sub-Modulus	ROM Size	Package Count
11, 13	direct	256 x 4	1
17, 19, 23, 29, 31	direct	1024 x 8	1
37, 41, 43, 47	15, 7	64 x 8	1
		64 x 4	1
		256 x 4	1
		128 x 8	1
53, 59, 61, 67 — 71	15, 14	128 x 8	1
73, 79, 83 — 89		256 x 4	2
97, 101, 103		256 x 8	1
107 —— 229	31, 15	256 x 8	1
		256 x 1	1
		1024 x 8	1
		256 x 4	1
		512 x 8	1

TABLE 4.7. Package count for multiplication or addition using submodular approach

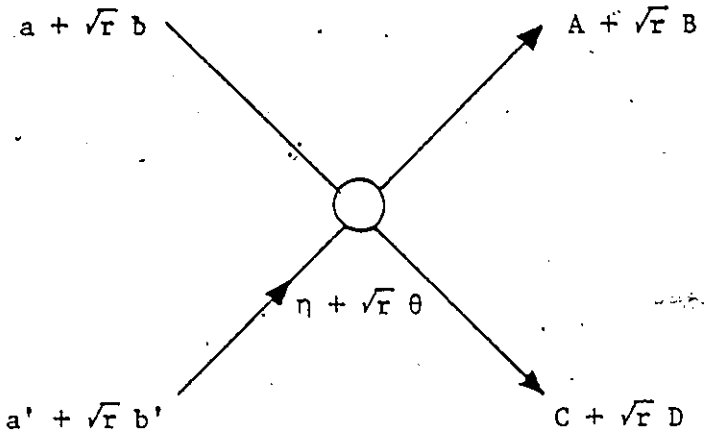


Figure 4.3. Radix-2 DIT Butterfly (BCU)

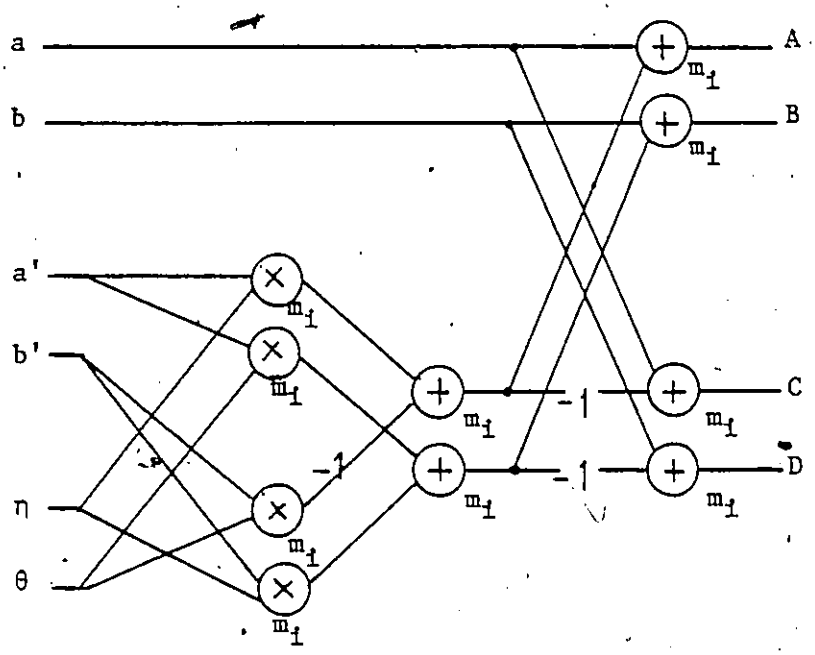


Figure 4.4. BCU Array for Primes $m_i = 4\xi + 3$

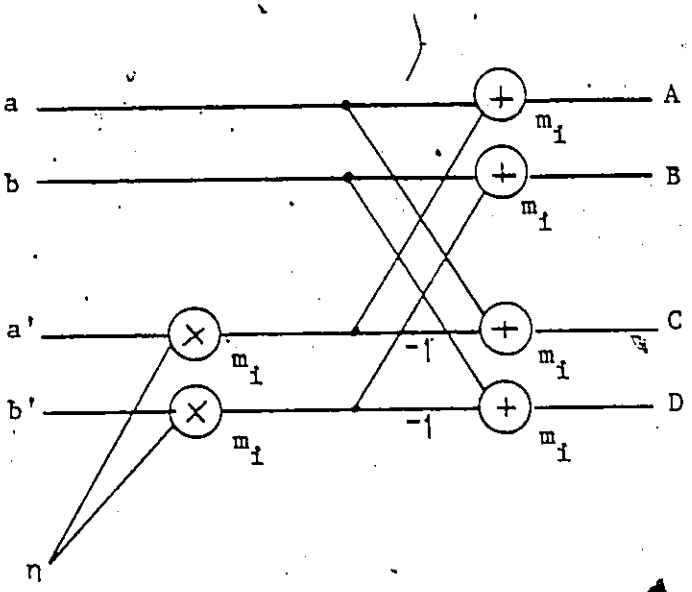


Figure 4.5. BCU Array for Primes $m_i = 4\xi + 1$ and even powers of α

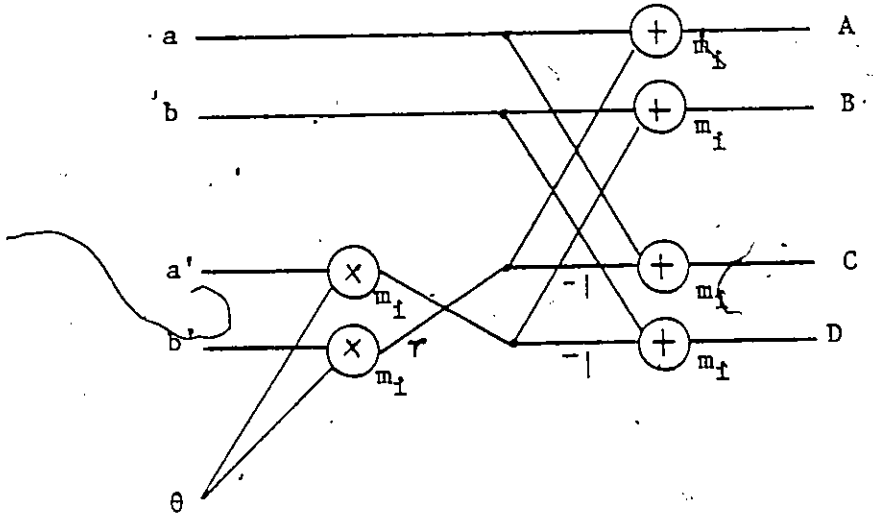


Figure 4.6. BCU Array for primes $m_i = 4\xi + 1$ and odd powers of α

Table 4.8 shows some possible choices of moduli for ROM look-up table implementation of NTTs over direct sum of $GF(m_i^2)$, together with associated dynamic range and transform factors.

Moduli m_i	Dynamic range $M = \prod_{i=1}^{L} m_i$	Length N	Generators α_i in $GF(m_i^2)$ respectively
17,31, 47	$2^{14.6}$	32	$\alpha_1 = \sqrt{5}$, $\alpha_2 = 2+\sqrt{-1}\ 11$, $\alpha_3 = 2+\sqrt{-1}\ 18$
31,97,127	$2^{18.54}$	64	$\alpha_1 = 25 + \sqrt{-1}\ 5$, $\alpha_2 = \sqrt{19}$, $\alpha_3 = 7+\sqrt{-1}\ 29$

TABLE 4.8. Examples of transform parameters for ROM array implementation

Of course, there are a number of other choices, for example

i) $M = 17 \times 31 \times 47 \times 79 \times 113$ provides $N = 32$ and generates a dynamic range of 27.72 bits

ii) $M = 31 \times 97 \times 127 \times 191$ provides $N = 64$ and generates a dynamic range of 26.12 bits

iii) $M = 127 \times 191 \times 193$ provides $N = 128$ and generates a dynamic range of 22.15 bits.

As an illustration of the ROM requirements for a typical implementation, consider the second example from Table 4.8, ie, the implementation of the transform over a ring isomorphic to the direct sum $GF(31^2) \oplus GF(97^2) \oplus GF(127^2)$. For $m_1 = 31$ only 10 ROMs are required for one BCU, each 1024 x 8, hardwired in the configuration shown in Figure 4.4. For $m_2 = 97$, a complete look-up table array,

implementing the structure of Figure 4.5, is shown in Figure 4.7. The adders are implemented in a sub-modular fashion, in which case the results of the previous multiplication can be obtained directly in the submodular system. The ROM count (not counting the stored tables for η) is 4 ROMs of size 256 x 4, 4 ROMs of size 128 x 8 and 10 ROMs, 256 x 8.

As a third parallel BCU, there is a butterfly array for modulus $m_3 = 127$. Using the same approach as above, for implementation of the structure of Figure 4.4, and package count given in Table 4.7, we obtain the following ROM count for the BCU in $GF(127^2)$:

ROM size	256 x 1	256 x 4	256 x 8	512 x 8	1024 x 8
Package count	4	10	4	10	10

4.3.5 Selection of transform parameters for parallel microprocessor implementation

For applications requiring only moderate speed but error free computation of convolution, arrays of microprocessors, operating in parallel, can be used to implement circular convolution on two blocks of real data using the NTT over direct sum of $GF(m_1^2)$. Both look-up tables and algorithms are stored in the paralled microprocessor systems. Separate microprocessors can be used to compute results in each modulus and the results can be combined using mixed radix conversion. A variety of cost versus speed trade-offs can be obtained using different microprocessor systems - eg., Shottky-Bipolar bit sliced system would provide a high speed implementation and the Intel 8048 system would provide a low speed, low cost implementation.

An efficient implementation of addition and multiplication, modulo

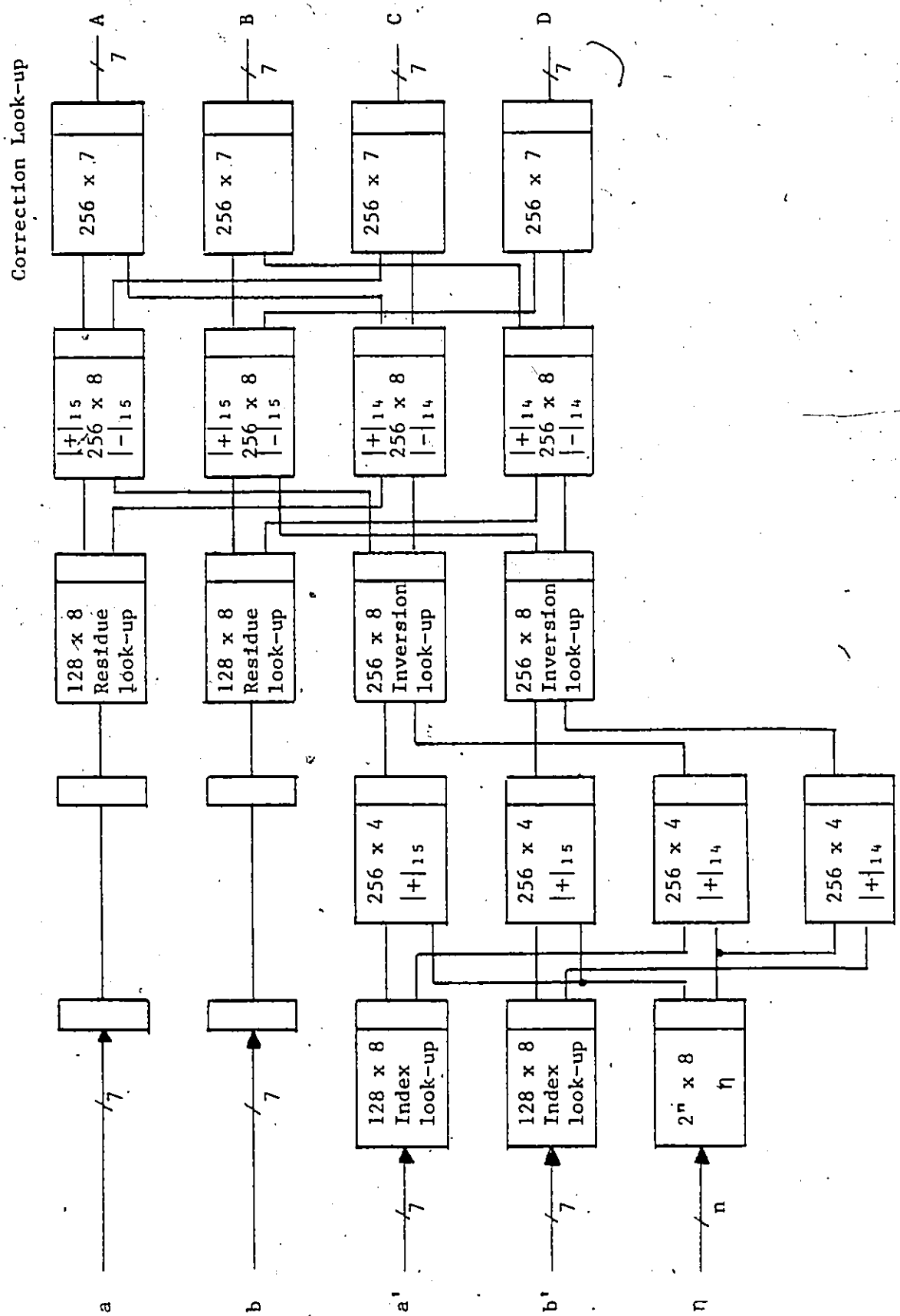


Figure 4.7. ROM array for radix-2 butterfly for $m_1 = 97$

a prime, m_i, has been described by Jullien [11], together with an example of a radix 2 routine for a single chip 8 bit microprocessor from the Intel 8048 series.

Here, a selection of transform parameters for such an implementation is given. If we use three 8 bit processors operating in parallel, then the moduli 127, 191 and 193 provide a dynamic range of 22.15 bits and a radix 2 transform length of 128 points. Since we use an NTT over a direct sum of $GF(m_i^2)$, 2 blocks of real data are transformed within one NTT.

The moduli 127 and 191 have the form $4\xi + 3$, so $\alpha_1 = 39 + \sqrt{-1}\,2$ and $\alpha_2 = 66 + \sqrt{-1}\,6$ are suitable generators of order 128 in the $GF(127^2)$ and $GF(191^2)$ respectively. For the modulus 193, the generator can be equal to $0 + \sqrt{r}$ if we choose for example $r = 125$ or $r = 158$.

4.4 NTT Convolution Filter For Complex Data

In many applications such as radar, sonar or communication, the sequences to be convolved consist of complex quantities. The convolution of complex-valued sequences can be performed naturally with NTTs defined in complex residue rings. The RNS implementation of such NTTs, presented here and published in [44] offers greater flexibility for the convolution with similar efficiency of computation compared to the conventional binary implementations.

The numbers of the form $A + jB$, $j = \sqrt{-1}$, A and B integers, are often called Gaussian integers. Hence, we will consider the residue class of Gaussian integers, ie, the complex residue ring $C(m_i)$ composed of the set:

$$C(m_i) = \{a + jb \mid a, b \in GF(m_i)\}$$

where j is a root of $x^2 + 1$.

If $j = \sqrt{-1}$ ($j \in C(m_i)$) does not belong to the field $GF(m_i)$, ie, $x^2 + 1$ is irreducible in $GF(m_i)[x]$, then the complex residue ring, $C(m_i)$, is composed of m_i^2 distinct elements, and the multiplicative group of invertible elements is of order $m_i^2 - 1$. Hence $C(m_i)$ is isomorphic to the Galois Field $GF(m_i^2)$. It has been shown in section 4.2 that this holds only for primes $m_i = 4\xi + 3$. For primes $4\xi + 1$, -1 is a quadratic residue, and $x^2 + 1$ has two solutions, λ_1 and λ_2, such that $\lambda_2 = m_i - \lambda_1$. It follows that when $m_i = 1 \pmod 4$, j may be considered as a member of $GF(m_i)$. The maximum order of any element in the multiplicative group of the complex ring $C(m_i)$ is $m_i - 1$; ie, the length of the transform is the same as in the real residue field modulo m_i. This is, for example, the case with Fermat Number Transforms. The generator α of order N_i, where N_i is a divisor of $m_i - 1$, can be a real number, $\alpha \in GF(m_i)$, or a complex number, $\alpha = \{\beta + j\gamma: \beta, \gamma \in GF(m_i)\}$. There are $\phi(N_i)$ real elements and $\phi(N_i)$ complex elements of order N_i in the ring $C(m_i)$.

The results can be summarized as follows:

A complex number theoretic transform with the dynamic range $M = \prod_{i=1}^{L} m_i$, m_i prime, can be implemented in the residue number system if the transform length N is a divisor of the gcd of the numbers N_i, $i = 1, \ldots, L$

where $N_i = m_i^2 - 1$, $m_i = 4\xi + 3$

$N_i = m_i - 1$, $m_i = 4\xi + 1$ (4.34)

The above is equivalent to implementing the transform over a complex ring C_M, which is isomorphic to a direct sum of rings

$$C_M \cong C_{m_1} \oplus C_{m_2} \oplus \cdots \oplus C_{m_L}$$

If $m_i = 4\xi + 3$, the ring C_{m_i} is isomorphic to the Galois Field $GF(m_i^2)$. If $m_i = 4\xi + 1$, the ring C_{m_i} is isomorphic to the direct sum, $GF(m_i) \oplus GF(m_i)$. Since, for this case, α can be chosen to be real, one possible implementation of the transform is to separately transform the real and imaginary parts in two Galois Fields $GF(m_i)$. The other approach is to use the mapping [34]

$$\psi: \psi(a + jb) = (a + \lambda b, a - \lambda b)$$

where λ denotes one of the square roots of -1.

Here, the former approach will be considered, because in ROM arrays the real and imaginary part can easily be handled separately; implementation of the mapping ψ and solving for a and b at the output would require extra hardware. The components of the complex circular convolution of sequences $W(t) = W_r(t) + j\, W_i(t)$ and $h(t) = h_r(t) = h_r(t) + j\, h_i(t)$, obtained from:

$$\begin{aligned} y(s) = y_r(s) + j\, y_i(s) = \sum_{t=0}^{N-1} \Big[& (h_r(|s-t|_N) \cdot W_r(|t|_N) \\ & - h_i(|s-t|_N) \cdot W_i(|t|_N)) + (j\, h_r(|s-t|_N) \cdot W_i(|t|_N) \\ & + h_i(|s-t|_N) \cdot W_r(|t|_N)) \Big] \end{aligned} \tag{4.35}$$

are required to have an upper bound M. Hence, the absolute upper bound on W and h is:

$$\max|W_r| \cdot \max|h_r| + \max|W_i| \cdot \max|h_i| \le \frac{M-1}{2N} \tag{4.36}$$

$$\max|W_r| \cdot \max|h_i| + \max|W_i| \cdot \max|h_r| \le \frac{M-1}{2N} \tag{4.37}$$

4.4.1. ROM array implementation considerations

As in section 4.3.4, we will consider a structure of a radix-2 DIT butterfly. For primes $4\xi + 3$ generator α is complex, $\alpha = \beta + j\gamma$, hence the array of look-up tables required for the radix-2 complex calculation

is the same as shown in Figure 4.4, where a, a' and b, b' now denote real and imaginary parts respectively. For primes $m_i = 4\xi + 1$, the generator is real, and the real and imaginary parts can be handled separately, as discussed in section 4.4. Hence, the number of look-up tables is 6 for one modulus if two identical arrays are constructed.

As an example of the ROM requirements for a typical implementation, consider a 32 point complex transform in the ring

$$C(31 \times 47 \times 97) = GF(31^2) \oplus GF(47^2) \oplus GF(97) \oplus GF(97)$$

For $m_i = 31$ only 10 ROMs are required, each 1024 x 8. For $m_i = 47$ each multiplier will require the ROMs shown in Table 4.7†, the adders will each require the following ROMs,

64 x 4, 256 x 4, 128 x 8

and the residue look-up for the input $\{a + jb\}$ will require two 64 x 8 ROMs.

For $m_i = 97$ two arrays can be constructed, each requiring two 128 x 8 ROMs, two 256 x 4 ROMs and five 256 x 8 ROMs. The total ROM count for this implementation, and two others, is shown in Table 4.9. The generator look-up tables are not included in the package count.

---

†The real and imaginary parts, a' and b', need only be passed once through the index look-up.

Dynamic Range M	Transform Length N	ROMs required for complex transform	
		Size	Package Count
$31 \times 47 \times 97 = 2^{17.09}$	32	64 x 4	10
		256 x 4	14
		64 x 8	4
		128 x 8	14
		256 x 8	10
		1024 x 8	10
$31 \times 47 \times 79 \times 97 = 2^{23.4}$	32	64 x 4	10
		256 x 4	34
		64 x 8	6
		128 x 8	18
		256 x 8	30
		1024 x 8	10
$31 \times 127 = 2^{11.94}$	64	256 x 1	4
		256 x 4	10
		256 x 8	4
		512 x 8	10
		1024 x 8	20

TABLE 4.9. Package count for some possible choices of moduli for a complex radix 2 computational unit

4.4.2 Parallel microprocessors implementation considerations

For applications requiring only moderate speed but error free computation of convolution, two microprocessors operating in parallel can be used to implement circular convolution using an NTT defined in a complex residue ring.

This idea has already been discussed in [40], but since the NTT was defined in real residue ring, the transform length was limited to a maximum value of m_i-1. If we choose the moduli in the form $4\xi + 3$, and we use an 8 bit processor, such as the Intel 8085 for Motorola 6800, arithmetic modulo the primes $m_1 = 239$ and $m_2 = 251$ can be carried out in a straightforward manner.

The dynamic range is 15.87 bits and the length of convolution can be increased significantly for a complex generator α. The maximum sequence length for $m_1 = 239$ is $N_1 = 57120 = 2^5 \cdot 3 \cdot 5 \cdot 7 \cdot 17$ and the maximum sequence length for $m_2 = 251$ is $N_2 = 63000 = 2^3 \cdot 3^2 \cdot 5^3 \cdot 7$. When the radices $\{2,3,5,7\}$ are used, the transform length is $N = 840 = 2^3 \cdot 3 \cdot 5 \cdot 7$ for complex input samples. Such an NTT would be useful for convolving data with short duration impulse responses. The number of elements of order 840 in $GF(239^2)$ and $GF(251^2)$ is $\phi(840) = 192$.

A computer program has been written for searching for elements of order N. $\alpha_1 = (34 + j1)$ and $\alpha_2 = (42 + j5)$ are suitable generators of order 840 in the $GF(239^2)$ and $GF(251^2)$ respectively. The multiplications (mod m_i) of two integers can be replaced by additions $(\text{mod}(m_i-1))$ and index table look-up operations. If we choose generators such that some of their powers have unity for real or imaginary parts, then we can reduce the computation time a little. The full precision final result of the convolution can be obtained from:

$$y = m_1 \cdot a_2 + a_1$$

where $a_1 = |y|_{m_1}$

$$a_2 = \left|\left|m_1^{-1}\right|_{m_2} \cdot \left|\left|y\right|_{m_2} - a_1\right|_{m_2}\right|_{m_2} \qquad (4.38)$$

For $m_1 = 239$ and $m_2 = 251$, $\left|m_1^{-1}\right|_{m_2} = 230$, so that the output y can be obtained from

$$y = 239 \cdot \left| 230 \quad (y_2 - y_1)\right|_{251} + y_1$$

If the precision at the output can be reduced to 8 bits, the scaled output, y_s, can be computed from

$$y_s = \left|\frac{y}{m_1}\right| = \left| 230 \cdot (y_2 - y_1)\right|_{251} + y_1$$

The error of scaling is bounded by $\left(1 - \frac{1}{m_i}\right)$, ie, is less than one quantization level. This latter case yields a much smaller computation time than the former, because the result can be computed using index table look-ups.

4.5 Transforms over Galois fields of higher degree

When the degree n of Galois field $GF(m^n)$ increases, the transform length, a divisor of m^n-1, increases significantly. However, the number of operations required for the multiplication of two field elements grows very rapidly and may overshadow the advantages of larger attainable length. Even if there exists the possibility of finding a generator with a simple form, there is still a general multiplication in the transform domain to be performed. Addition of field elements is componentwise, hence the number of binary additions is proportional to n. To minimize the number of binary operations required for general multiplication of two elements in $GF(m^n)$, we will consider the simplest form

of the irreducible polynomial of degree n, ie, a monic binomial x^n-r.

In this case general multiplication requires n^2 binary multiplications, $n(n-1)$ binary additions and $n \cdot \left\lceil \frac{n-1}{2} \right\rceil$ multiplications by a constant r.

Consider the case $n = 4$. Assume that x^4-r is irreducible. Hence, the multiplication of two elements in $GF(m^4)$, where x^4-r has a root, λ, is as follows:

$$(a + \lambda b + \lambda^2 c + \lambda^3 d)(a' + \lambda b' + \lambda^2 c' + \lambda^3 d') =$$

$$|aa' + rbd' + rcc' + rdb'|_m + \lambda|ab' + ba' + rcd' + rdc'|_m +$$

$$\lambda^2|ac' + bb' + ca' + rdd'|_m + \lambda^3|ad' + bc' + cb' + da'|_m$$

Hence, a total of 16 binary multiplications, 12 binary additions and 6 multiplications by a constant r is required. It is clear, from the factorization of m^4-1 that radix-2 transform lengths in $GF(m^4)$ are increased twice over those in $GF(m^2)$.

However, if the increased transform length is of primary importance, the search for computationally simple Galois fields of higher degree is worth attempt.

4.6 Summary

Techniques have been developed for indirect filtering of real and complex data, using fast Number Theoretic Transforms. Transforms have been defined over a direct sum of Galois fields, $\sum_{i=1}^{L} \oplus GF(m_i^2)$ and it has been demonstrated that such NTTs provide greatly increased power of 2 transform length over those defined over $\sum_{i=1}^{L} \oplus GF(m_i)$. If sequences to be convolved are real, the transform length is further increased by a factor of 2 (in general by a factor equal to the degree of extension).

We have restricted the ring modulus, M, to have a prime power factorization where the power is unity. This restriction does not limit the transform length and represents the most efficient factorization as far as implementation is concerned. We are therefore restricting the ring to be isomorphic to a direct sum of Galois fields rather than the more general isomorphism to a direct sum of Galois rings.

Within each Galois field, the most efficient implementation is obtained when the irreducible polynomial (which is used to build the extension field) has the form of the monic binomial, $f(x) = x^2 - r$.

The result has been obtained, and proved, that for primes of the form $4\xi + 1$ the generator of the multiplicative group of order $N = 2^B$, where B is the maximum power of two transform length in $GF(m_i^2)$, can have the simple form, $\alpha = 0 + \sqrt{r}$. Techniques for finding such generators have been provided. This result allows hardware implementations of an NTT over $GF(m_i^2)$ which are as efficient as those over $GF(m_i)$; the transform length, however, is increased significantly. The selection of transform parameters has been provided for two distinct RNS implementations, are using arrays of ROMs and the other using arrays of microprocessors (or similar computational elements). Initial results obtained for transforms defined over a direct sum of Galois fields of degree higher than 2, $\sum_{i=1}^{L} \oplus GF(m_i^n)$, indicate that the number of binary operations, required for general multiplication in the transform domain, increase rapidly with the degree, whereas the power of 2 length is only doubled for each successive even value of n. However, if the increased transform length is of primary importance, a study could be undertaken to determine simple forms for generator α. Perhaps consider-

ing radix 4 implementations, ie, searching for a simple form generator of power of 4 order, will lead in some way to improving the efficiency. This problem was not addressed in this research work, and indicates a definite area for future research.

CHAPTER 5

READ-ONLY-MEMORY IMPLEMENTATIONS OF RNS CODED RECURSIVE DIGITAL FILTERS

5.1. Introduction

Recent applications of RNS structures have concentrated, in the main, on digital signal processing functions. As one of the building blocks for linear digital processing, the recursive digital filter has received some attention from workers in the field. Jullien [4] has discussed applications of residue coding to the implementation of a recursive filter second order canonic section, which is one of the basic building blocks for any one dimensional recursive filter.

Jenkins [5] has proposed hardware architectures combining residue number concepts and combinatorial techniques. Combinatorial digital filter architectures [17] eliminate general multiplication through the use of precomputed partial sums stored in ROMs. Residue combinational structures, consisting of several short wordlength subfilters operating in parallel [5] provide the capability for efficient, high-speed and high-precision implementation of recursive filters.

Soderstrand [50] 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 [51]. The need for scaling is avoided by storing complete look-up tables for multiplication by fractions. The RNS-LDI filter has been shown to offer substantial cost savings and speed advantages over the currently available structures based on binary arithmetic. However, the stored table implementation of multiplication by a fraction excludes a

straightforward application of RNS-LDI filters for adaptive filtering or multiplexing schemes, where the filter coefficients are to be changed dynamically. This architecture is also not attractive for standard RNS integer multiplication because twice as many scaling arrays are required compared to the previously discussed forms.

The combinational algorithm, residue coded or not, is not feasible for adaptive filters because it is based on storing precomputed partial sums. For a second order section, 32 linear combinations of coefficients would have to be computed and stored each time the coefficients were updated, every combination requiring 4 additions. This procedure prohibitively slows down the filtering process.

For the cases where the filter should handle adaptively varying coefficients, the structures with general multiplication should be considered. In this case, scaling is necessary to keep the data within the limited dynamic range.

The Residue Number System provides the potential for high-speed, efficient implementation if the structures used as basis for RNS implementation have an abundance addition and multiplication operations, and as few as possible scaling operations. Moreover, since in the residue number system, addition, subtraction and multiplication are performed with extended precision and the only errors occur during the scaling process, such structures will also exhibit low sensitivity to quantization noise. Therefore, we will consider direct and canonic forms of recursive filter second order sections, which require the computation of five multiplications and four additions and can be implemented with only one scaling process. We will present an analysis of quantization noise and limit cycle effects in the proposed RNS implementations.

5.2. Recursive Digital Filters

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

$$y(n) = \sum_{i=0}^{M} a_i\, u(n-i) - \sum_{i=1}^{N} b_i\, y(n-i) \tag{5.1}$$

where $\{u(n)\}$ is the input sequence, $\{y(n)\}$ the output sequence and $\{a_i\}$, $\{b_i\}$ are the filter coefficients.

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 in terms of computation time and memory than nonrecursive filters in producing similar magnitude-frequency response characteristics.

For hardware realizations, it is convenient to consider the z-domain transfer function of the digital filter H(z), given [18] by:

$$H(z) = \frac{\sum_{i=0}^{M} a_i\, z^{-i}}{\sum_{i=0}^{N} b_i\, z^{-i}} \tag{5.2}$$

where $b_o \triangleq 1$ and z^{-1} is the unit delay operator.

A digital filter transfer function can often be realized in a variety of ways. Noise and inaccuracies caused by the quantization of digital filter parameters are very dependent on the precise digital filter structure.

It has been long recognized ([53], [54], [18]) that direct implementation of (5.2) is usually to be avoided because the accuracy

requirements on the coefficients $\{a_i\}$ and $\{b_i\}$ are often severe, since small errors in the coefficients (due to rounding) result in unacceptable large errors in pole positions. In most instances it proves desirable to realize a given network by means of either cascade or parallel combinations of second order sections because these realizations are the least sensitive to the adverse effects associated with finite register length.

The cascade form corresponds to a factorization of the numerator and denominator polynomials of (5.2) so that H(z) is expressed as a product of second order sections:

$$H(z) = a_o \prod_{i=1}^{N_1} H_i(z) \qquad (5.3)$$

where N_1 is the integer part of (N+1)/2.

If the transfer function of the filter is written as a partial fraction expansion of first and second order terms,

$$H(z) = c + \sum_{i=1}^{N_1} H_i(z) \qquad (5.4)$$

where $c = a_N/b_N$, the entire filter may be visualized as a parallel connection of the simpler filters $H_i(z)$.

The individual second order sections

$$H_i(z) = \frac{L_0 + L_1 Z^{-1} + L_2 Z^{-2}}{1 + K_1 Z^{-1} + K_2 Z^{-2}} \qquad (5.5)$$

where $\{L_i\}$, $\{K_i\}$ are real coefficients (L_2 is not required for parallel form) are usually realized using one of the following forms:

i) direct form 1, shown in Fig.5.3, implementing the difference equation:

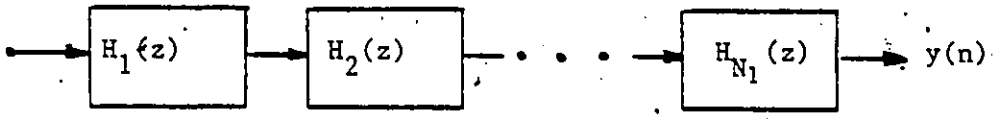


Fig.5.1 Cascade form

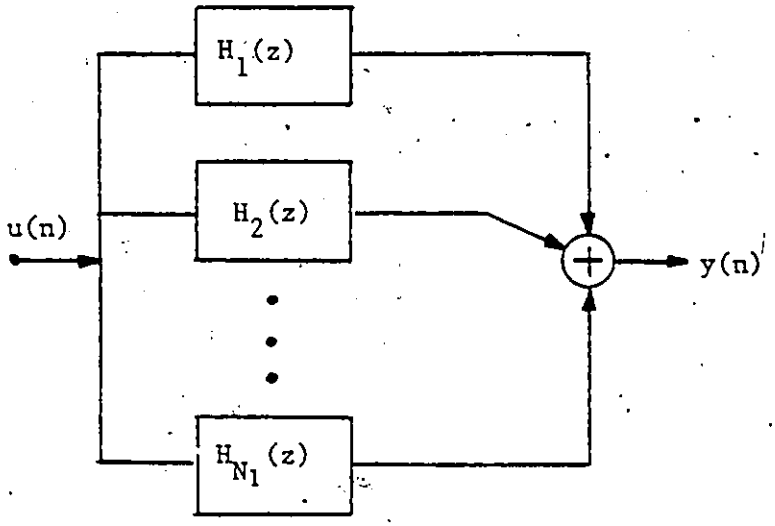


Fig.5.2 Parallel form

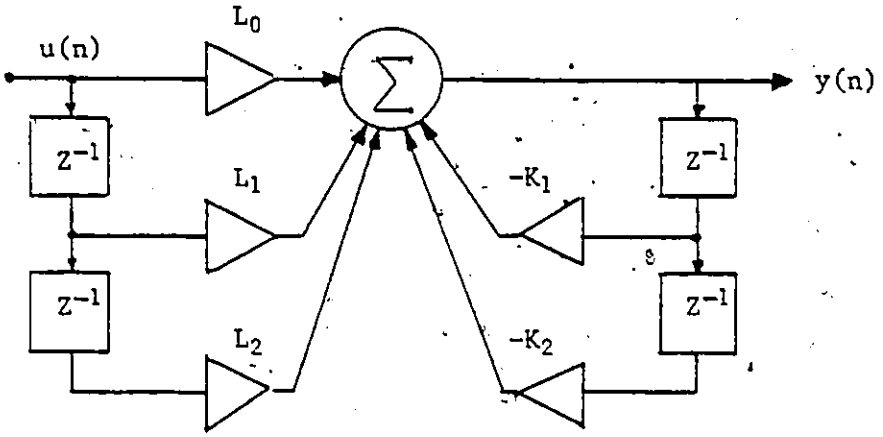


Fig.5.3 Block diagram representation of direct form 1 second order section of a digital filter

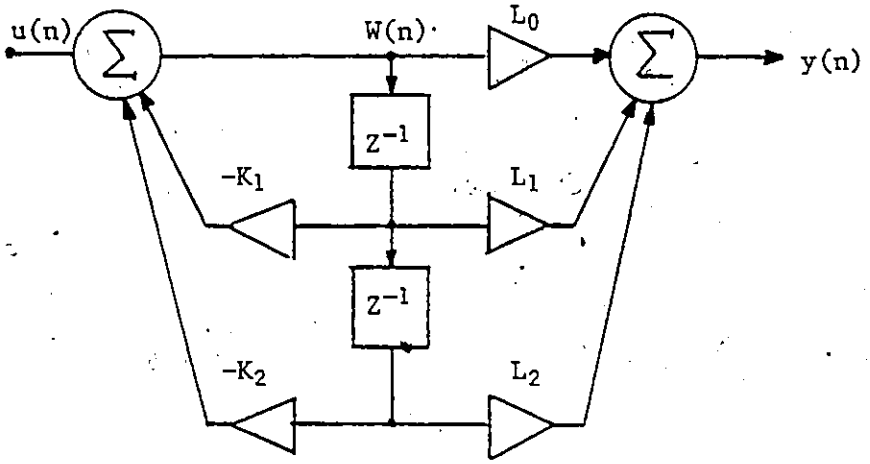


Fig.5.4 Block diagram representation of canonical form second order section

$$y(n) = L_0u(n) + L_1u(n-1) + L_2\,u(n-2) - K_1y(n-1) - K_2\,y(n-2) \quad (5.6)$$

ii) direct form 2, also called canonic form, shown in Fig.5.4, implementing the pair of difference equations:

$$y(n) = L_0w(n) + L_1w(n-1) + L_2w(n-2)$$
$$w(n) = u(n) - K_1w(n-1) - K_2w(n-2) \quad (5.7)$$

where w(n) is internal to the filter.

Alternative second order section realizations, such as coupled form [18] or specific nonminimal realization proposed in [65], require substantially more computations than direct or canonic forms and are therefore used only when the quantization of parameters is a problem.

The implementation of a second order section in direct or canonic form would require four adders, five multipliers and the number of delay units shown in Fig.5.3 and 5.4.

Since digital filter arithmetic is carried out with finite word-length representation, the output of the digital filter deviates from the desired characteristics. A great deal of study ([53-57]) 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). In this chapter, quantization effects associated with the RNS implementation, will be discussed.

5.3. RNS Coded Recursive Filters

Since the RNS is an integer number system, fractional filter coefficients must be converted to integers by multiplying by an appropriate conversion factor, p, and rounding the result to the nearest integer. Let the superscript "^" denote an integer value, hence:

$$\hat{L}_i = [L_i \cdot p]_{RO}$$
$$\hat{K}_i = [K_i \cdot p]_{RO} \qquad (5.8)$$

where $[\cdot]_{RO}$ represents the closest integer value function.

Scaling is necessary to keep the data within the limited dynamic range. Dynamic range requirements are discussed in later sections.

Figure 5.5 - 5.8 illustrate cascade and parallel realizations of a sixth order recursive filter, where second order sections are implemented either in the direct form 1 or in canonic form; $|u(n)|_{m_i}$ and $|y(n)|_{m_i}$ denote the residues modulo m_i of the input and output respectively.

It can be seen from the figures that we have to scale only once in each second order section. An immediate observation can be made that the RNS implementation with one noise injection source, due to scaling in each second order section, can provide better quantization error signal to noise ratio than the conventional binary implementations, where the wordlength reduction is usually performed after every multiplication. The error introduced by residue scaling is discussed in the following sections.

5.3.1. Quantization error sources

When a recursive filter is implemented using the RNS, three forms of quantization error are present:

i) errors due to quantizing the input signal in the A/D converter

ii) coefficient rounding error

iii) error generated by the scaling operation.

In order to determine the effects of quantization errors on filter performance, it is first necessary to establish the error models.

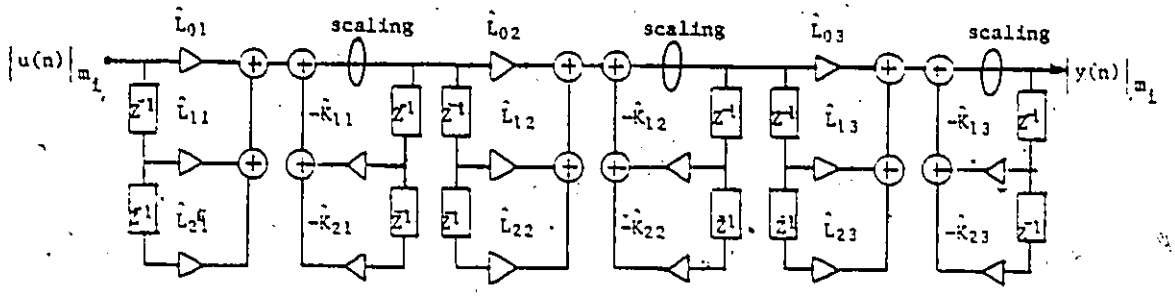


Fig.5.5 Cascade realization of sixth order recursive residue filter. Second order sections implemented in direct form I

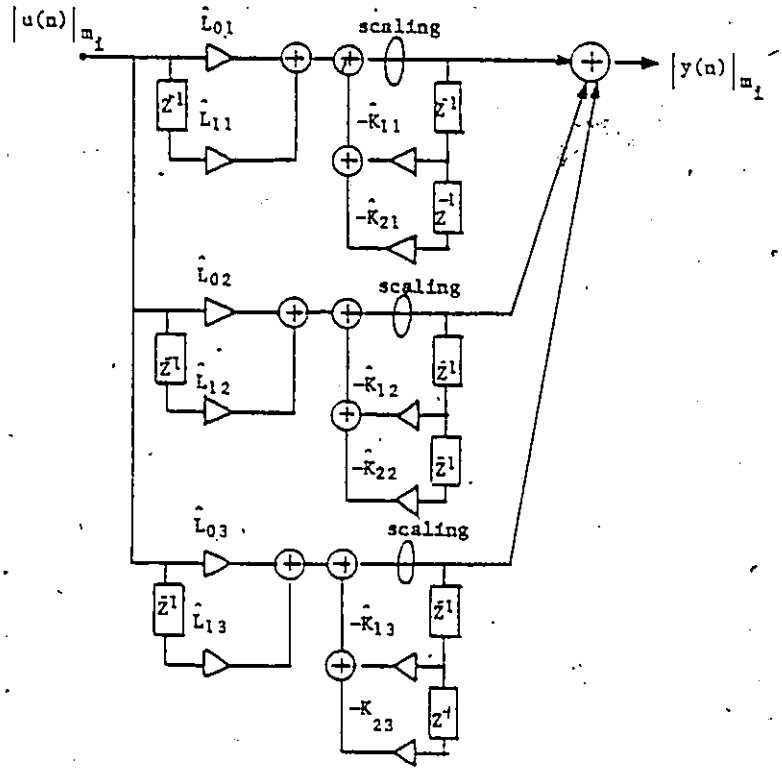


Fig.5.6 Parallel implementation of sixth order recursive residue filter. Second order sections implemented in direct form I

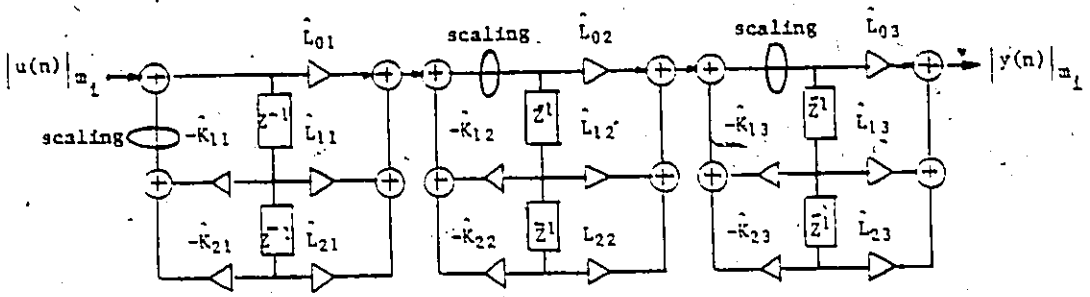


Fig.5.7 Cascade realization of sixth order recursive residue filter. Second order sections implemented in canonic form

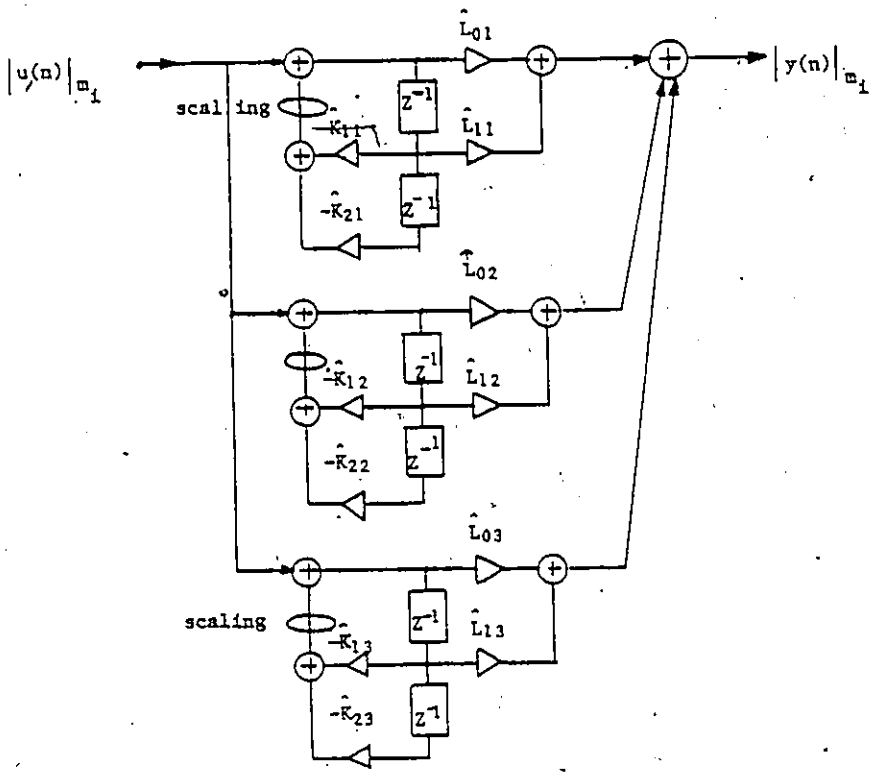


Fig.5.8 Parallel realization of sixth order recursive residue filter. Second order sections implemented in canonic form

i) A/D quantization noise

A quantization error is inherent in any A/D conversion process. The exact input samples are rounded to the nearest quantization level, so that the quantized-input to the filter can be expressed as $u(n) + e_Q$; $u(n)$ is the exact sample and e_Q is a quantization error bounded by $-Q/2 < e_Q \leq Q/2$, Q-converter step size.

Assume that the dynamic range of the A/D converter is 2^B bits. In the RNS we are dealing with integers, therefore the converter quantization step size, Q, is equal to 1.

Usually, it is assumed that the errors are equiprobable in the range (-0.5, 0.5), with this assumption the mean and variance of the converter quantization error can be shown to be.

$$E[e_Q] = 0$$

$$\sigma^2 e_Q = \frac{Q^2}{12} = \frac{1}{12} \tag{5.9}$$

The usual approach for treating the effect of input quantization in digital filters ([55], [58]) is based on the assumption that the following statistical model holds, i.e.

- the error sequence is uncorrelated with the input sequence
- the random variables of the error process are uncorrelated
- the probability distribution of the error is uniform over the quantization error range.

Experimental results[57] support the validity of these assumptions for signals with a reasonably large amplitude variance and wide spectral bandwidth. The steady state output component due to $\{e_Q\}$ has zero mean and its variance is computed [58] from

$$\sigma^2 oe_Q = \sigma^2 e_Q \frac{1}{2\pi j} \oint_{|z|=1} H(z)\, H(z^{-1}) \left[\frac{1}{z}\right] dz \qquad (5.10)$$

The contour integral can be evaluated analytically (a tedious task for high order filters) or, if the filter coefficients are given numerically, it can be evaluated by computer program or tables [60].

For the RNS implementation, the output variance of the converter quantization error is

$$\sigma^2 oe_Q = \frac{1}{12} \frac{1}{2\pi j} \oint_{|z|=1} H(z)\, H(z^{-1}) \left[\frac{1}{z}\right] dz \qquad (5.11)$$

To compare the effects of input quantization error in the RNS and conventional fixed point implementations, the relative signal to noise ratio, $\frac{\sigma_u^2}{\sigma^2 oe_Q}$ can be determined. If we assume the input signal to be a uniformly distributed, zero mean sequence, then the mean square value of the input is

$$\sigma_u^2 = \frac{2^{2B}}{12} \qquad (5.12)$$

In the conventional fixed point implementation of a filter, the input is usually scaled such that $|u(n)| \leq 1$. For a B bit quantizer (we assume the B bits of the A/D converter include the sign bit) the quantization step is 2^{-B+1}. Hence

$$\sigma^2 oe_Q = \frac{2^{-2B}}{3} \frac{1}{2\pi j} \oint_{|z|=1} H(z)\, H(z^{-1}) \left[\frac{1}{z}\right] dz \qquad (5.13)$$

The mean square value of the input is

$$\sigma_u^2 = \frac{1}{3} \qquad (5.14)$$

Hence, the output signal to noise ratio due to input quantization is the same in the RNS and conventional implementation.

ii) Effect of coefficient inaccuracy

The error associated with the truncation and integer conversion of filter coefficients is defined as

$$e_{LI} = L_i \cdot p - \hat{L}_i \qquad i = 0, 1, 2$$
$$e_{KI} = K_i \cdot p - \hat{K}_i \qquad i = 1, 2 \tag{5.15}$$

Using a round-off procedure to determine the integer representation of coefficients (eq.5.8), the errors $\{e_{LI}\}$ and $\{e_{KI}\}$ are uniformly distributed, zero mean sequences, with variance $\sigma^2_I = \frac{1}{12}$. The effects of coefficient error on the filter characteristic have received extensive treatment in the literature [55], [18], [59] and they will not be discussed here.

iii) Error due to scaling

The effects of RNS scaling on the performance of second order recursive sections is discussed in the following sections.

5.3.2. Scaling in the RNS

In order to keep data within a given dynamic range, scaling will have to be performed. The fundamental problems with the scaling operation in the RNS are that, unlike addition and multiplication, the number system is not closed under the scaling operating and that the residue digits do not convey any immediate information about the magnitude of the number.

Scaling in the residue number system is most easily implemented when the scaling factor is a product of some of the moduli.

Several techniques are available for scaling within the RNS. The following discussion briefly outlines these techniques. A complete discussion with algorithms can be found in the literature [4],[5].

i) Exact Division Scaling

This technique uses multiplicative inverses to perform division. The complete algorithm for scaling by the first S moduli in the RNS with the range $M = \prod_{i=1}^{L} m_i$ has been given by Jullien in [4].

The output from this scaling procedure is

$$W = \left\lfloor \frac{X}{D} \right\rfloor = \frac{X - |X|_D}{D} \tag{5.16}$$

where X is the input and D is a scale factor, $D = \prod_{i=1}^{S} m_i$. The result is meaningful only if the number X is exact multiple of D. If we choose a divisor that is equal to one of the moduli, eg. $D = m_1$, then $|X|_D$ is directly available from the first residue and

$$|W|_{m_i} = \left| |X - |X|_{m_1}| \cdot \left| \frac{1}{m_1} \right|_{m_i} \right|_{m_i} \tag{5.17}$$

If $S > 1$ we do not have direct access to $|X|_D$, rather we use an iterative scaling process [4]:

$$\left|\phi^{(k+1)}\right|_{m_i} = \left| |\phi^{(k)} - \phi_k| \cdot \left| \frac{1}{m_k} \right| \right|_{m_i} \tag{5.18}$$

with $\phi^{(1)} = X$, $\phi^{(S+1)} = W$ and $\phi_k = |\phi^{(k)}|_{m_k}$ for $1 \le k \le S$; $k < i \le L$.

This technique will only generate residues for $m_i : i \geq S+1$ (for $1 \leq i \leq S, (D, m_i) = m_i$ and so we cannot determine $\left|\frac{1}{D}\right|_{m_i}$ or $\left|W\right|_{m_i}$). If we wish to continue calculations in the RNS after such a scaling algorithm, as in the case of recursive filtering, we have to reconstruct the first S residues of the scaled number. This is referred to as base extension [2] and employs a partial conversion to a mixed radix weighted magnitude representation.

This scaling algorithm yields a rounded down (truncated) estimate. The algorithm can generate a zero mean error by the addition of $[\frac{D}{2}]$ to the input [2]. The number of tables and look-up cycles required for the scaling array can be obtained from the following

$$\begin{aligned} &\text{No. of Tables} = (L-1)(S + \tfrac{L}{2}) - S^2 \\ &\text{No. of look-up cycles} = L \end{aligned} \tag{5.19}$$

The program simulating 'Exact Division' scaling is given in Appendix C.

ii) Metric Vector Estimate Scaling

This algorithm uses the notion of scaled metric vector estimates. Associated with each residue x_i in the L tuple is a metric vector [1]

$$V_i = x_i \cdot \hat{m}_i \cdot \left|\frac{1}{\hat{m}_i}\right|_{m_i} \tag{5.20}$$

where $\hat{m}_i = \frac{M}{m_i}$ and the magnitude of the unit metric vector is given by $\hat{m}_i \cdot \left|\frac{1}{\hat{m}_i}\right|_{m_i}$.

We can rewrite the Chinese Remainder Theorem (section 3) in terms of a summation of metric vectors:

$$X = \left|\sum_{i=1}^{L} V_i\right|_M$$

An estimate of W can be found by summing the nearest integer values of the metric vector scaled by D.

$$\left|W'\right|_{m_k} = \left|\sum_{i=1}^{L}\left[\frac{V_i}{D} + \frac{1}{2}\right]\right|_{m_k} = \left|\sum_{i=1}^{L}\left|\left[\frac{\left(\prod_{j=s+1}^{L} m_j\right)\cdot\left|\hat{\frac{x_i}{m_i}}\right|_{m_i}}{m_i} + \frac{1}{2}\right]\right|_{m_k}\right|_{m_k}$$

for $s+1 \le k \le L$. (5.21)

This procedure is followed by the base extension algorithm, as in the exact division case.

The ROM implementation of this scaling algorithm has been covered in depth in[4]. Also it has been shown that the upper bound on the error due to the metric vector estimate scaling is $|\epsilon| < S/2$ and so is larger than for the exact division algorithm. If the number of moduli for scaling, S, is fixed a priori, the efficiency of realization of the estimate technique is either equal to or greater than the exact division technique. The latter, however, is more efficient if flexibility in S is required.

iii) Specialized Residue Systems for Efficient Scaling

Jenkins [5], [49] identified four special classes of residue systems in which the scaling algorithm, given by equation (5.22), is simple to implement with minimal quantization error

$$W'' = \left|\sum_{i=1}^{L} \eta_i \cdot \left|\hat{\frac{x_i}{m_i}}\right|_{m_i}\right|_{\frac{M}{D}} \qquad (5.22)$$

where $D = \prod_{i=1}^{S} m_i$, η_i is an integer $\frac{\hat{m}_i}{D}$ for $S < i \le L$ and η_i is replaced by $\left[\frac{\hat{m}_i}{D} + \frac{1}{2}\right]$ for $1 \le i \le S$.

Equation (5.22), as with the scaled metric vector summation,

represents the scaled version of the Chinese Remainder Theorem. Although, in general for S > 1 the upper bound on the error is large in this case, a judicious choice of moduli [5] leads to minimal quantization error accummulation. These classes of residue systems have moduli, scale factors and quantization errors as follows:

Class I

$m_1 = m,\ m_2 = m-1;\ D = m-1;\ -(1 - 1/(m-1)) \leq \varepsilon \leq 0$

Class II

$m_1 = 2^{k+\ell} - 1,\ m_2 = 2^k;\ D = 2^k;\ 0 \leq \varepsilon \leq (1 - 1/2^k)$

Class III

$$m_1 = m-1,\ m_2 = m,\ m_3 = m+1;\ D = (m-1)(m+1);\ -\left[1 - \frac{1}{(m-1)}\right] \leq \varepsilon \leq \left[1 - \frac{1}{(m+1)}\right]$$

Class IV

$$m_1 = 2^k,\ m_2 = 2^k-1,\ m_3 = 2^{k-1} - 1;\ D = 2^k(2^{k-1} - 1);$$

$$-\left[1 - \frac{1}{(2^{k-1}-1)}\right] \leq \varepsilon \leq \left[1 - \frac{1}{2^{k-1}}\right]$$

In case of scaling by two moduli (Class III and IV) the upper bounds on the quantization error are larger than the upper bounds for the exact division algorithm, derived in the following section.

The upper bounds on the quantization error and the efficiency of implementation of scaling expressed by equations (5.22) and (5.21) are comparable for small number of moduli in the RNS system.

5.3.3. Quantization error analysis for second order recursive section and exact division scaling.

Here, the most general case will be considered where the relatively prime moduli $\{m_i\}$, are chosen to provide the required dynamic range

$M = \prod_{i=1}^{L} m_i$. The exact division scaling algorithm will be considered for RNS implementation of a second order recursive section either in direct form 1 (Fig.5.9) or in canonic form (Fig.5.10).

The difference equations for direct and canonic forms respectively are given by equations (5.23) and (5.24).

$$y(n) = \left[\frac{1}{D}\left(\hat{L}_0\, u(n) + \hat{L}_1\, u(n-1) + \hat{L}_2\, u(n-2) - \hat{K}_1\, y(n-1) - \hat{K}_2\, y(n-2)\right)\right] \tag{5.23}$$

$$y(n) = \hat{L}_0\, w(n) + \hat{L}_1\, w(n-1) + \hat{L}_2\, w(n-2)$$

$$w(n) = u(n) + \left[\frac{1}{D}\left(-\hat{K}_1\, w(n-1) - \hat{K}_2\, w(n-2)\right)\right] \tag{5.24}$$

or:

$$w(n) = \left[\frac{1}{D}\left(u'(n) - \hat{K}_1\, w(n-1) - \hat{K}_2\, w(n-2)\right)\right]$$

where D is the scaling factor, $D = \prod_{i=1}^{S} m_i$, and $u'(n)$ is the normalized input for scaling placed as shown in Fig.5.10b.

Let the dynamic range of input data and filter coefficients be $\leq 2^B$ and $\leq 2^C$ respectively. The scaling factor can be chosen as p (the coefficient integer normalization factor, p, has been defined in equation (5.8)), to that scaling does not change the overall filter function. Usually the coefficients are in the range (-2, +2), hence $D \simeq 2^{C-2}$ is desirable.

The minimum requirement for RNS dynamic range is $M = 3 \cdot 2^{B+C}$ for canonic realization and $M = 5 \cdot 2^{B+C}$ for direct form 1 realization. Increasing M will increase the signal to noise ratio, however it will increase cost.

In standard fixed point realizations of digital filters, the quantization errors associated with the realization of multipliers are

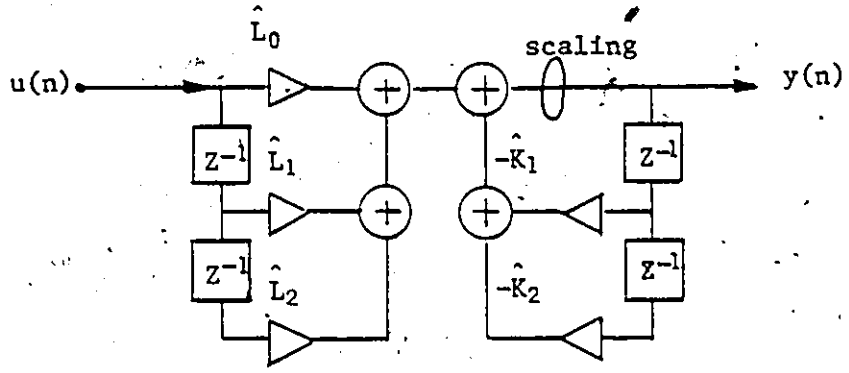
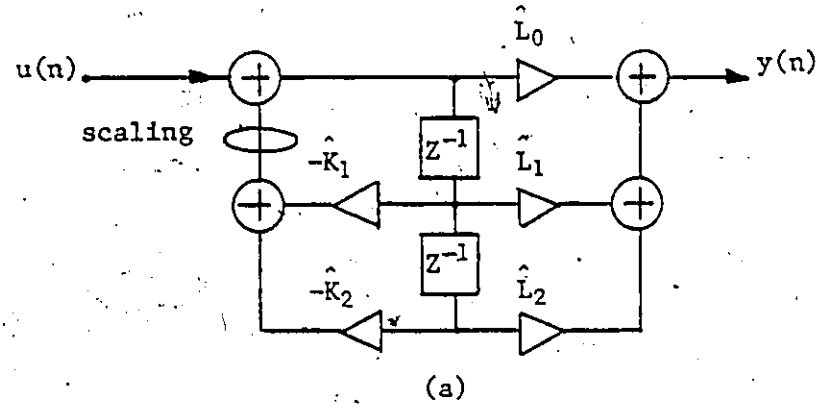
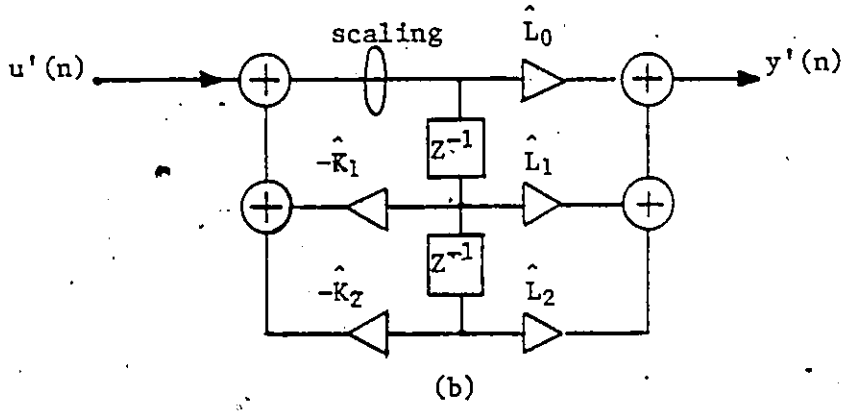


Fig.5.9 RNS implementation of direct form 1 second order section



(a)



(b)

Fig.5.10 RNS implementation of second order canonic section

(a) $|u(n)| \leq 2^{B-1}$

(b) input normalized to $|u'(n)| \leq 2^{B-1} \cdot D$

usually modelled as random uniformly distributed processes, uncorrelated with the input. This model, based on the fundamental work by Bennet [57], has been shown to give very reliable results [55], [54], [56] when the filter is driven by a non-zero input signal.

In the case of the RNS implementation the binary operations of multiplication and addition are performed in an extended precision. The only errors occur during the scaling process.

We will perform a noise analysis based upon the same assumptions, ie,

i) the scaling noise probability density function is uniformly distributed

ii) the scaling noise is uncorrelated with the input

iii) the scaling noise is uncorrelated from sample to sample.

For the case of round-off scaling, the mean of the error is zero and the maximum bound on the error can be obtained as follows:

$$e_s = \frac{X}{D} - \left[\frac{X + [\frac{D}{2}]}{D}\right] = \frac{X}{D} - \frac{X + [\frac{D}{2}] - \left| X + [\frac{D}{2}]\right|_D}{D} =$$

$$- \frac{D - |D|_2}{2D} + \frac{\left|X + [\frac{D}{2}]\right|_D}{D} \leq - \frac{D-1}{2D} + \frac{D-1}{D} = \frac{D-1}{2D} \qquad (5.25)$$

Hence $- \frac{D-1}{2D} \leq e_s \leq \frac{D-1}{2D}$

The assumed scaling noise probability density function is shown in Fig.5.11a. The error due to round-down scaling $[\frac{X}{D}]$ can easily be shown to $0 \leq e_s < \frac{D-1}{D}$.

In both cases the variance of the scaling noise can be shown to be:

$$\sigma_s^2 = \frac{1}{12}\left(\frac{D-1}{D}\right)^2 \qquad (5.26)$$

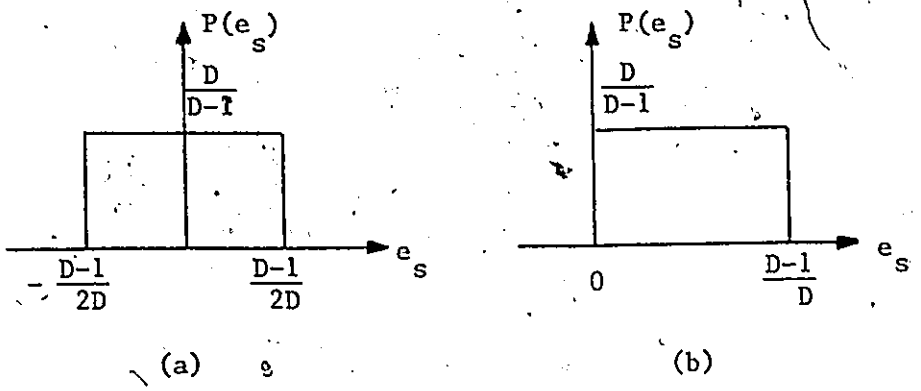


Fig.5.11 Scaling noise probability density functions for RNS round-off (a) and round-down (b) scaling

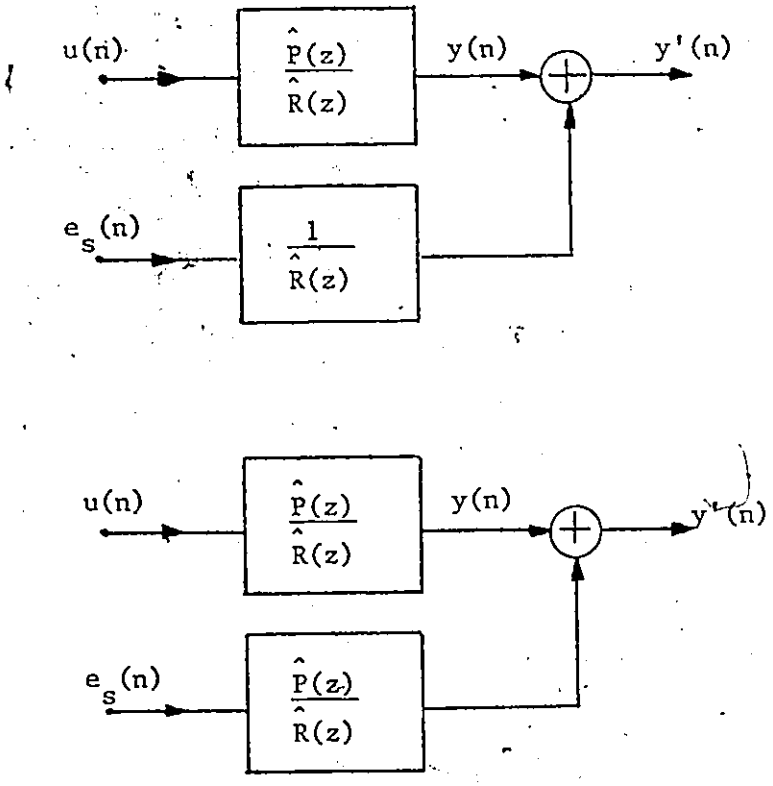


Fig.5.12 Contribution of error due to scaling for RNS second order section realized in direct form 1 (a) and canonic form (b) $\hat{P}(z) = \hat{L}_0 + \hat{L}_1 Z^{-1} + \hat{L}_2 Z^{-2}$; $\hat{R}(z) = p + \hat{K}_1 Z^{-1} + \hat{K}_2 Z^{-2}$

The scaling error contribution in the second order section realized in direct form 1 and canonic form is shown in Fig.5.12.

The output noise variance can be obtained from:

$$\sigma^2_{os_1} = \frac{1}{12}\left[\frac{D-1}{D}\right]^2 \cdot \frac{1}{2\pi j} \oint_{|Z|=1} \frac{\hat{P}(z)\,\hat{P}(z^{-1})}{\hat{R}(z)\,R(z^{-1})}\left[\frac{1}{z}\right] dz \qquad (5.27)$$

$$\sigma^2_{os_2} = \frac{1}{12}\left[\frac{D-1}{D}\right]^2 \cdot \frac{1}{2\pi j} \oint_{|Z|=1} \frac{1}{\hat{R}(z)\cdot\hat{R}(z^{-1})}\left[\frac{1}{z}\right] dz \qquad (5.28)$$

for canonic and direct forms respectively.

The integral can be evaluated using algorithms presented in [60]. In particular, for the second order section,

$$\frac{1}{2\pi j} \oint_{|Z|=1} \frac{P(z)\,P(z^{-1})}{R(z)\,R(z^{-1})}\frac{1}{z}\,dz = \frac{(L_0^0)^2}{K_0^0} + \frac{(L_1^1)}{K_0^1} + \frac{L_2^2}{K_0} \qquad (5.29)$$

where $P(z) = L_0 + L_1\, z^{-1} + L_2\, z^{-2}$

$R(z) = K_0 + K_1\, z^{-1} + K_2\, z^{-2}$

$$K_1^1 = \frac{K_0K_1 - K_1K_2}{K_0}$$

$$L_1^1 = \frac{K_0L_1 - L_2K_1}{K_0}$$

$$K_0^1 = \frac{(K_0)^2 - (K_2)^2}{K_0}$$

$$L_0 = \frac{K_0L_0 - L_2K_2}{K_0}$$

$$K_0^0 = \frac{(K_0^1)^2 - (K_1^1)^2}{K_0^1}$$

$$L_0^0 = \frac{K_0^1L_0^1 - L_1^1K_1^1}{K_0^1}$$

It is obvious from the above that either $\sigma^2_{os_1}$ or $\sigma^2_{os_2}$ is the smaller depending on the particular values of filter coefficients.

Consider the second order canonic section. Using the same approach as in section 5.3.1, the theoretical signal to noise ratio for the RNS implementation can be shown to be

$$\left(S/N\right)_{RNS} = \frac{\sigma_{ou}^2}{\sigma_{os}^2} = 2^{2B} \cdot \left(\frac{D}{D-1}\right)^2 \tag{5.30}$$

whereas, for the conventional implementation

$$\left(S/N\right)_C = \frac{\frac{1}{3} \cdot \frac{1}{2\pi j} \oint H(z) \cdot H(z^{-1}) \left[\frac{1}{z}\right] dz}{2 \cdot \frac{2^{-2B}}{3} \cdot \frac{1}{2\pi j} \oint H(z) \cdot H(z^{-1}) \left[\frac{1}{z}\right] dz + 3 \cdot \frac{2^{-2B}}{3}} = 2^{2B} \cdot \frac{J}{2J+3} \tag{5.31}$$

where J denotes the closed contour integral.

Figure 5.13 shows the plot of $\left(S/N\right)_{RNS}$ given by equation (5.30) versus scaling factor D.

Since $D \simeq p = 2^{C-2}$, for practical dynamic range of filter coefficients, 2^C, $\left(S/N\right)_{RNS} \simeq 2^{2B}$ and the relative signal to noise ratio for RNS and conventional implementation for canonic form is

$$\frac{\left(S/N\right)_{RNS}}{\left(S/N\right)_C} = \frac{2J + 3}{J} \tag{5.32}$$

The relative signal to noise ratio for direct form 1 realization is given by

$$\frac{\left(S/N\right)_{RNS}}{\left(S/N\right)_C} = 5 \tag{5.33}$$

showing that RNS implementation is far superior to the conventional fixed point implementation when the error due to internal arithmetic is a criterion of performance.

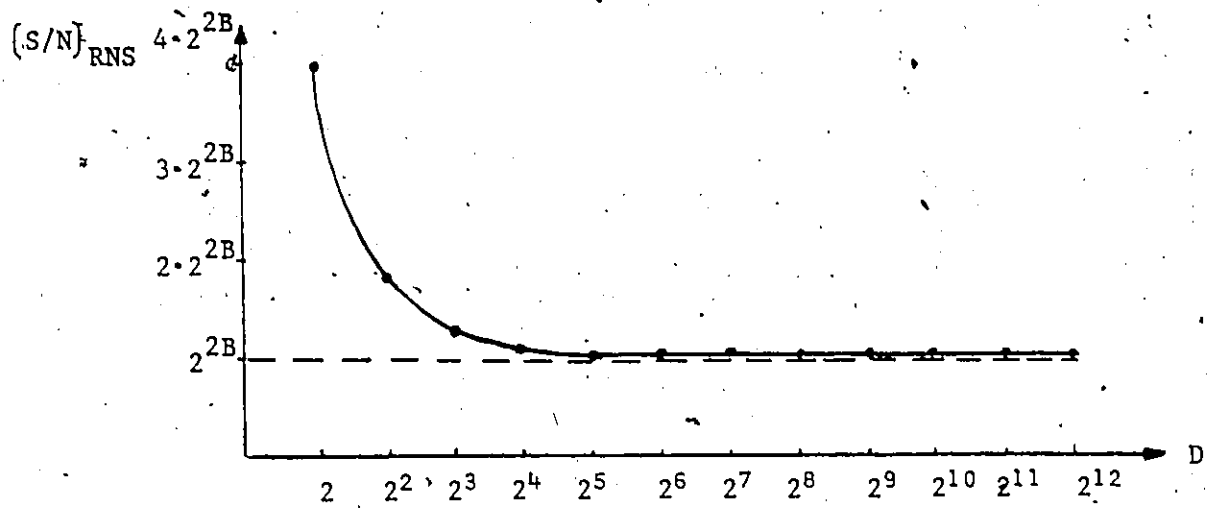


Fig.5.13 Plot of signal to noise ratio versus scaling factor

Experimental simulations of estimated mean-squared output noise to signal ratio have shown that the theoretical value is very pessimistic.

As an example, consider a second order lowpass Butterworth filter whose transfer function is given by:

$$H(z) = \frac{1 + 2\ z^{-1} + z^{-2}}{1 + 1.5610\ z^{-1} + 0.6414\ z^{-2}} \qquad (5.34)$$

Assume an RNS implementation with C = 9 bits. Hence:

$\hat{L}_0 = 256$, $\hat{L}_1 = 512$, $\hat{L}_2 = 256$, $\hat{K}_1 = 400$, $\hat{K}_2 = 164$.

Assume the input signal to be uniformly distributed with zero mean and B = 8 bits. The theoretical value for the signal to noise ratio is in this case 66572.

The experimental ratio of estimated mean-squared output signal for a pseudo-random input to the mean-squared output noise due to residue scaling is 94966 over a period of 1000 samples.

The theoretical value of the signal to noise ratio for the conventional binary implementation is 17904, since for the transfer function given by equation (5.34), the integral in the equation (5.31) evaluates to 1.807.

The relative signal to noise ratio defined by (5.32) is therefore 3.718, showing that the performance of the RNS realization is also superior to the conventional realization for the canonic form second order section.

5.4. Limit cycles in the RNS implementation of second order recursive section

In section 5.3.3 the error due to scaling has been modelled as a

random process, uncorrelated with the processed signal. This assumption leads to accurate results for most applications with high signal level and sufficiently wide spectral contents.

However, if the input to the filter is constant, eg, zero, quantization nonlinearities in digital filters implemented with fixed-point arithmetic give rise to small amplitude limit cycle oscillations, ie, the output of a filter remains periodic and non-zero after the input has been set to zero. Recently, specific structures have been derived [65] that are free of limit cycles. They also can be used as a basis for RNS design, however, they require more multiplications than the standard direct form 1 and canonic forms. A recent review of limit cycles in digital filters, with an extensive bibliography, has been provided by Claasen et al [56].

The problem of the stability of RNS coded second order sections is different from most of the previously reported work, because the quantization is performed once per iteration rather than after each multiplication. Fig.5.14 shows the model of a zero input second order section with two poles and no zeros. The nature of the nonlinearities due to round-off and round-down RNS scaling are shown in Fig.5.15. The output, y(n) is described by

$$y(n) = \left[\frac{-\hat{K}_1\, y(n-1) - \hat{K}_2\, y(n-2)}{p} \right] \tag{5.35}$$

and

$$y(n) = \left[\frac{-\hat{K}_1\, y(n-1) - \hat{K}_2\, y(n-2) + \left[\frac{p}{2}\right]}{p} \right] \tag{5.36}$$

for round-down and round-off scaling respectively, where it is assumed that D = p.

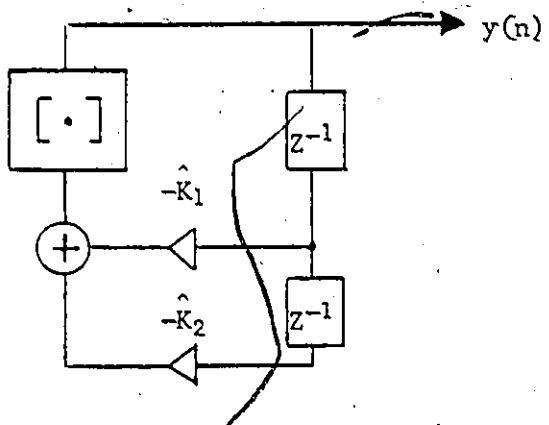


Fig.5.14 Second order section model for zero input limit cycles.

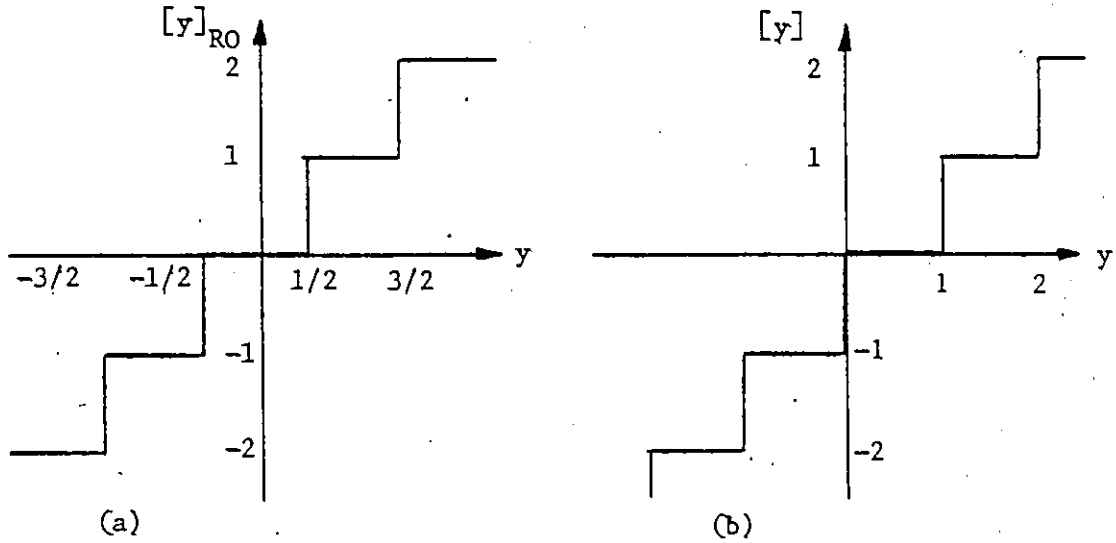


Fig.5.15 Scalor characteristic (a) round-off (b) round-down

As is well known [64], the area of absolute stability of the linear filter is given by $|K_2| < 1$ and $|K_1| < 1 + K_2$. This is the triangle in the (K_1, K_2) plane, shown in Fig.5.16. In the RNS implementation, the filter coefficients are converted into integers,

$$\hat{K}_1 = [K_1 \cdot p] \quad , \quad \hat{K}_2 = [K_2 \cdot p]$$

In the rest of this section, limit cycles of period one and two will be examined and necessary and sufficient conditions for the existence of such limit cycles will be presented.

(A) Round-off scaling

i) Limit cycles of period one.

In this case we have

$$y(n) = F$$

$$\left[\frac{-\hat{K}_1 \cdot F - \hat{K}_2 \cdot F + \frac{p}{2}}{p}\right] = F \tag{5.37}$$

where we assume p even for simplicity

$$\frac{-\hat{K}_1 \cdot F - \hat{K}_2 \cdot F + \frac{p}{2}}{p} = F + \varepsilon \tag{5.38}$$

where $0 \leq \varepsilon \leq \frac{p-1}{p} < 1$

From the above we have

$$F(\hat{K}_1 + \hat{K}_2 + p) - p/2 = -p \cdot \varepsilon \tag{5.39}$$

therefore

$$-p < F(\hat{K}_1 + \hat{K}_2 + p) - p/2 \leq 0$$

or

$$-\frac{3}{2}p < F(\hat{K}_1 + \hat{K}_2 + p) \leq p/2 \tag{5.40}$$

F is an integer, hence limit cycles of period 1 exist only if the values of $\hat{K}_1$ and $\hat{K}_2$ satisfy

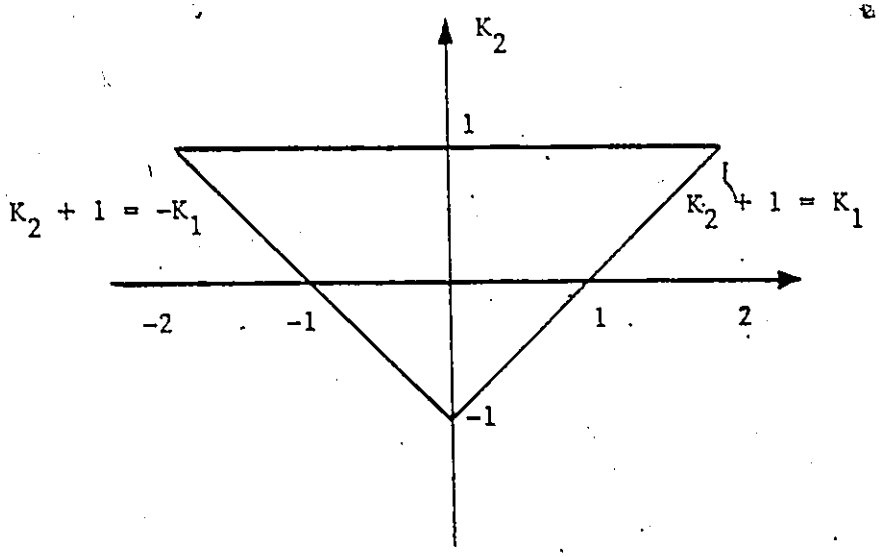


Fig.5.16 Stability diagram for filter of Fig.5.14

$$-\frac{5}{2}p < \hat{K}_1 + \hat{K}_2 \leq -\frac{p}{2} \tag{5.41}$$

or, since inside the triangle of stability $\hat{K}_1 + \hat{K}_2 > -p$

$$-p < \hat{K}_1 + \hat{K}_2 \leq -p/2 \tag{5.42}$$

Cross-hatched region in Fig.5.17 indicates the absence of zero input limit cycles of period 1. In particular, from (5.40), limit cycles of constant amplitude

$F = \pm 1$ can only exist if $\hat{K}_1 + \hat{K}_2 \leq -p/2$

$F = \pm 2$ if $\hat{K}_1 + \hat{K}_2 \leq -\frac{3p}{4}$

$F = \pm 3$ if $\hat{K}_1 + \hat{K}_2 \leq -\frac{5}{6}p$ etc.

The maximum amplitude of constant value limit cycles is given by

$$F_{max} \leq \left[\frac{p/2}{p + \hat{K}_1 + \hat{K}_2}\right] \tag{5.43}$$

ii) Limit cycles of period two

In this case we have

$$y(n) = F_1$$

$$y(n+1) = F_2 \tag{5.44}$$

F_1, F_2 integers.

From (5.36)

$$\left[\frac{-F_1 \cdot \hat{K}_1 - F_2 \cdot \hat{K}_2 + p/2}{p}\right] = F_2 \tag{5.45}$$

$$\left[\frac{-F_2 \cdot \hat{K}_1 - F_1 \cdot \hat{K}_2 + p/2}{p}\right] = F_1 \tag{5.46}$$

or:

$$-F_1 \cdot \hat{K}_1 - F_2 \cdot \hat{K}_2 + p/2 = p \cdot F_2 + p \cdot \varepsilon_1 \tag{5.47}$$

$$-F_2 \cdot \hat{K}_1 - F_1 \cdot \hat{K}_2 + p/2 = p \cdot F_1 + p \cdot \varepsilon_2 ; 0 \leq \varepsilon_1, \varepsilon_2 \leq \frac{p-1}{p} < 1$$

Assume, first that $F_2 = -F_1$, hence:

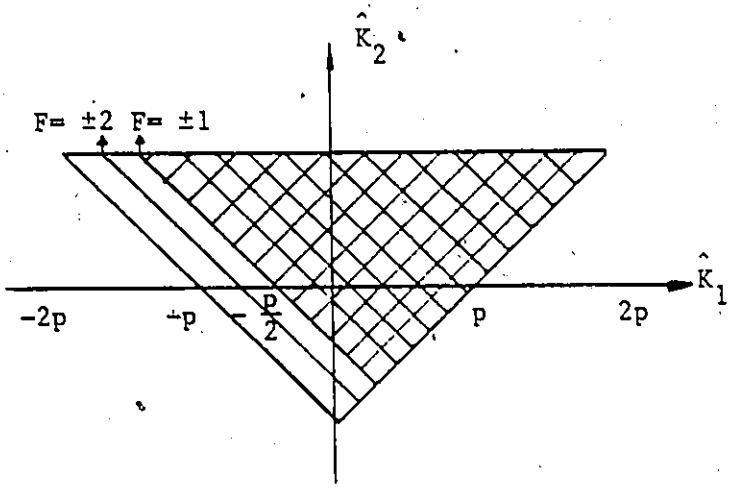


Fig.5.17 Stability diagram for recursive residue filter. Necessary and sufficient conditions for existence of dc limit cycles for round-off scaling

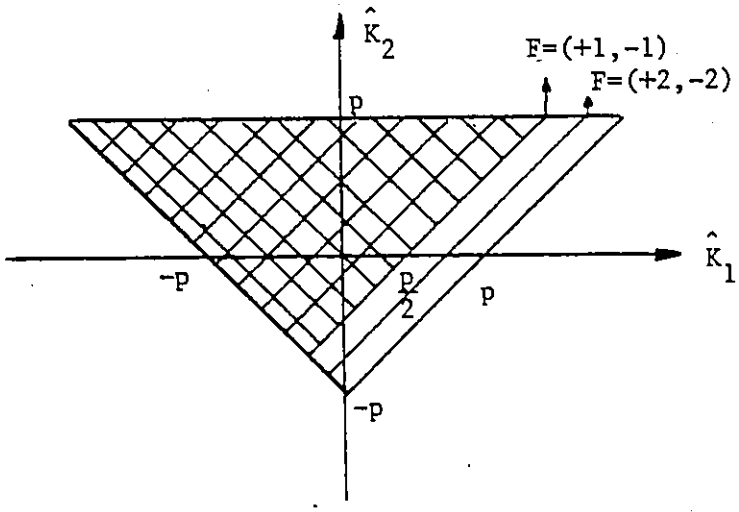


Fig.5.18 Necessary and sufficient conditions for existence of limit cycles of period 2 and $F_1 = F_2$ for round-off scaling

$$F_1 \cdot \hat{K}_1 - F_1 \cdot \hat{K}_2 + p/2 = p \cdot F_1 + p \cdot \varepsilon \tag{5.48}$$

and we obtain the inequality

$$-p/2 \leq F_1(\hat{K}_1 - \hat{K}_2 - p) < p/2 \tag{5.49}$$

Therefore limit cycles of period 2, $F_1 = -F_2$, can exist only if

$$p/2 < \hat{K}_1 - \hat{K}_2 < 3/2\ p \tag{5.50}$$

or, since inside the triangle of stability, $\hat{K}_1 - \hat{K}_2 < p$, then $p/2 < \hat{K}_1 - \hat{K}_2 < p$.

In particular, limit cycles with amplitude

$F_1 = \pm 1$, $F_2 = \mp 1$ exist for $\hat{K}_1 - \hat{K}_2 > p/2$

$F_1 = \pm 2$, $F_2 = \pm 2$ exist for $\hat{K}_1 - \hat{K}_2 > 3p/4$

$$F_{1,2\ max} \leq \left[\frac{p/2}{K_1 - K_2 - p} \right] \tag{5.51}$$

Consider now $F_2 \neq -F_1$ and let $G = -F_1 - F_2$

From (5.47) we get

$$G \cdot \hat{K}_1 + G \cdot \hat{K}_2 + p = -G \cdot p + p(\varepsilon_1 + \varepsilon_2) \tag{5.52}$$

which can be rewritten as

$$0 \leq G(\hat{K}_1 + \hat{K}_2 + p) + p < 2p$$

$$\text{or} \quad \frac{-p}{\hat{K}_1 + \hat{K}_2 + p} \leq G < \frac{p}{\hat{K}_1 + \hat{K}_2 + p} \tag{5.53}$$

Therefore a necessary condition for the existence of limit cycles of period 2, for $G = \pm 1$, is that

$$-p \leq \hat{K}_1 + \hat{K}_2 + p < p$$

or

$$-2p \leq \hat{K}_1 + \hat{K}_2 < 0 \tag{5.54}$$

If $G = \pm 2$, then $3/2\ p \leq K_1 + K_2 < -p/2$

Consider $G = 1$ and $F_1 = -1$ so that $F_2 = 0$.

Substituting these values into (5.47) a sufficient condition for the existence of the limit cycle (-1, 0) is that

$$-p/2 \leq \hat{K}_1 < p/2$$

$$-\frac{3}{2}p \leq \hat{K}_2 < -p/2 \qquad (5.55)$$

Combining results of (5.54) and (5.55), the necessary and sufficient conditions for the existence of the limit cycle (-1, 0) are illustrated in Fig.5.19.

Similar analyses can be carried out for other values of F_1 and F_2. No limit cycles of period 2 and $F_1 \neq -F_2$ have been observed to occur for $|\hat{K}_2| < p/2$.

(B) Round-down Scaling

The only difference in the analysis of the occurence of limit cycles is that the addition of one half of the scale factor is omitted.

i) Limit cycles of period 1.

In this case we have, from (5.35):

$$\left[\frac{-\hat{K}_1 \cdot F - \hat{K}_2 \cdot F}{p}\right] = F \qquad (5.56)$$

or

$$\frac{-\hat{K}_1 \cdot F - \hat{K}_2 \cdot F}{p} = F + \varepsilon \quad 0 \leq \varepsilon < 1$$

Therefore we obtain

$$-p < F(\hat{K}_1 + \hat{K}_2 + p) \leq 0 \qquad (5.57)$$

Since inside the triangle of stability $\hat{K}_1 + \hat{K}_2 + p > 0$ observation can be made that in this case the limit cycles of period one will have negative magnitude, with

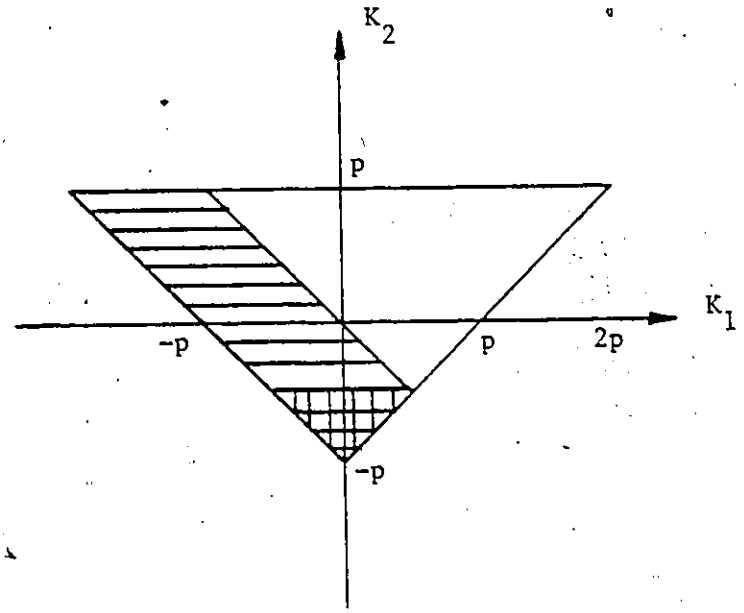


Fig.5.19 Limit cycles of period two and round-off scaling
Horizontally hatched region indicates necessary condition for existence of limit cycles of period two. Vertically matched region indicates sufficient conditions for existence of limit cycle (-1, 0)

$$|F_{max}| < \frac{p}{p + \hat{K}_1 + \hat{K}_2} \tag{5.58}$$

From (5.57) we obtain that, in particular,

for $F = -1$

$$-(\hat{K}_1 + \hat{K}_2 + p) > -p$$

or $\hat{K}_1 + \hat{K}_2 + p < p$

and $\hat{K}_1 + \hat{K}_2 < 0$ (5.59)

for $F = -2$

$$\hat{K}_1 + \hat{K}_2 < -p/2 \tag{5.60}$$

for $F = -3$

$$\hat{K}_1 + \hat{K}_2 < -\frac{3}{2}p \tag{5.61}$$

The regions in the $(\hat{K}_1, \hat{K}_2)$ plane where limit cycles of constant magnitude, F, exist are shown in Fig.5.20.

ii) Limit cycles of period two

Assume first $F_1 = -F_2$

then

$$F_1 \cdot \hat{K}_1 - F_1 \cdot \hat{K}_2 = pF_1 + p\varepsilon \tag{5.62}$$

and using the inequality $0 \leq \varepsilon < 1$

$$0 \leq F_1(\hat{K}_1 - \hat{K}_2 - p) < p \tag{5.63}$$

Limit cycles of period two and $F_1 = -F_2$ can exist only if

$$p \leq \hat{K}_1 - \hat{K}_2 < 2p \tag{5.64}$$

Inequality (5.64) specifies the coefficients $\hat{K}_1$, $\hat{K}_2$ outside the triangle of stability; therefore, by contradiction, there can be no limit cycles of period two with equal magnitude and opposite sign.

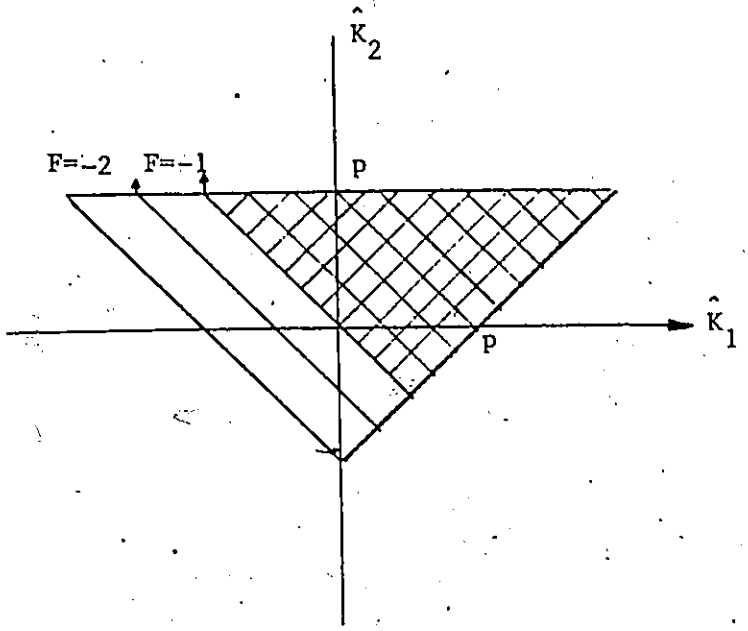


Fig.5.20 Necessary and sufficient conditions for existence of dc limit cycles for round down scaling

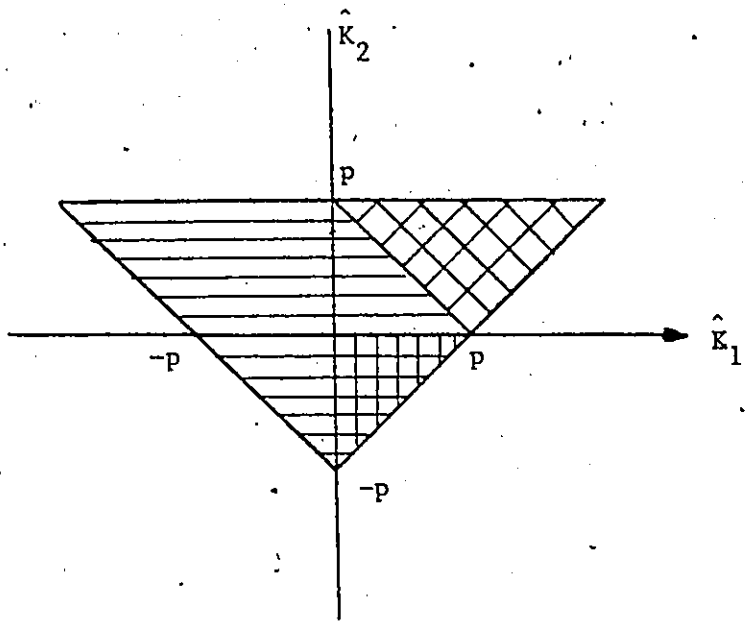


Fig.5.21 Limit cycles of period 2 and round down scaling. Horizontally hatched area indicates necessary condition for existence of limit cycles of period 2. Vertically hatched area indicates sufficient conditions for existence of limit cycle (-1, 0).

For $F_2 \neq -F_1$ let $G = -F_1 - F_2$

In this case we have

$$-F_1 \cdot \hat{K}_1 - F_2 \cdot \hat{K}_2 = pF_2 + p\varepsilon_1 \tag{5.65}$$
$$-F_2 \cdot \hat{K}_1 - F_1 \cdot \hat{K}_2 = pF_1 + p\varepsilon_2 \qquad 0 \leq \varepsilon_1, \ \varepsilon_2 < 1$$

and

$$0 \leq G\,(\hat{K}_1 + \hat{K}_2 + p) < 2p$$

Since $\hat{K}_1 + \hat{K}_2 + p > 0$, therefore G can take on only positive values. The necessary condition for existence of limit cycles of period 2 is, for G = 1

$$\hat{K}_1 + \hat{K}_2 < p \tag{5.66}$$

and, for G = 2

$$\hat{K}_1 + \hat{K}_2 < 0 \tag{5.67}$$

In particular for G = 1, where $F_1 = -1$, $F_2 = 0$, a sufficient and necessary condition is

$$\left.\begin{aligned} 0 \leq \hat{K}_1 < p \\ -p \leq K_2 < 0 \end{aligned}\right\} \tag{5.68}$$

Combined equations (5.68) and (5.66) are illustrated in Fig.5.21. Similar analyses for other values of F_1 and F_2 lead to the conclusion that no limit cycles of period two will occur for $\hat{K}_2 > 0$.

iii) Longer period cycles

Limit cycles of period three and longer are due to "effective" pole pairs [61] on the unit circle. It has been demonstrated[61], [62] that for roundoff quantization these limit cycles can occur for $|K_2| > 0.5$, irrespective of whether one or two quantizers are used. The exact analyses of long period limit cycles were beyond the scope of thia research work. Computer simulations of second order recursive

sections with a set of 100 randomly chosen initial conditions have confirmed the absence of long period limit cycles for $K_2 < 0.5$ ($\hat{K}_2 < p/2$) using round-off scaling. In the case of round-down scaling, no limit cycles of period longer than two have been found for $K_2 < 0.7$ ($\hat{K}_2 < 0.7\ p$).

The following examples illustrate the occurrence of limit cycles in the RNS implementation of a second order section. Most of the cycles have only appeared if the filter was started with initial conditions pertaining to that cycle. Only same of the DC limit cycles have been reached from randomly chosen initial conditions.

Filter Coefficients	RNS Filter Coefficients for C = 9 bits	LIMIT CYCLES IN RNS FILTER	
		Round-down scaling	Round-off scaling
$K_1 = -1.8437$	$\hat{K}_1 = -230$	DC limit cycles with magnitude	DC limit cycles with magnitude
$K_2 = 0.9375$	$\hat{K}_2 = 120$	- 1	+ 1
		- 2	+ 2
		- 3	+ 3
		- 4	- 1
		- 5	- 2
		- 6	- 3
		- 7	
$K_1 = 1.8437$	$\hat{K}_1 = 230$	Limit cycle of period 7	Limit cycle of period 2
$K_2 = 0.9375$	$\hat{K}_2 = 120$	(3, -3, 2, -1, -1, 2, -3)	(+3, -3)

TABLE 5.1. Examples of limit cycles in RNS recursive digital filters.

5.5 ROM oriented RNS based second order section

One important feature of using ROM arrays for building signal processing structures is the ease with which the array can be pipelined. Consider a moduli set for the RNS with a dynamic range $M = \prod_{i=1}^{L} m_i$, $16 < m_i \leq 32$. The second order RNS coded recursive filter will consist of L parallel subfilters, operating in each modulus. The block diagram of a canonic realization second order subfilter is shown in Fig. 5.22.

The package count required for the RNS coded recursive filter is L x 9 ROMs plus the package count required for the scaling array (equation 5.19), not counting stored tables for the filter coefficients. Since the 'Exact Division' scaling requires L cycles, the throughput rate of the second order section is given by

$$f_t = \frac{1}{(L+3)\ T} \tag{5.69}$$

where T is the memory access time, or in the case of the pipelined structure, the memory access time plus latch settling time (section 3). In the pipelined structure, the data can be accepted at a rate equal to the inverse of T, conservatively in excess of 14 MHz. Therefore, we can consider, for example, the parallel configuration of L + 3 second order sections, implemented with one multiplexed [18] second order section. The filter coefficients are changed each clock pulse to correspond to the appropriate section in the parallel configuration. Alternatively, the multiplexed second order section can operate upon L + 3 inputs simultaneously, ie, the input samples from L + 3 sources are interleaved sample by sample and fed (serially) into the filter. The combination of these two types of multiplexins is also possible. If the filter coefficients are allowed to be fixed, the package count of the RNS

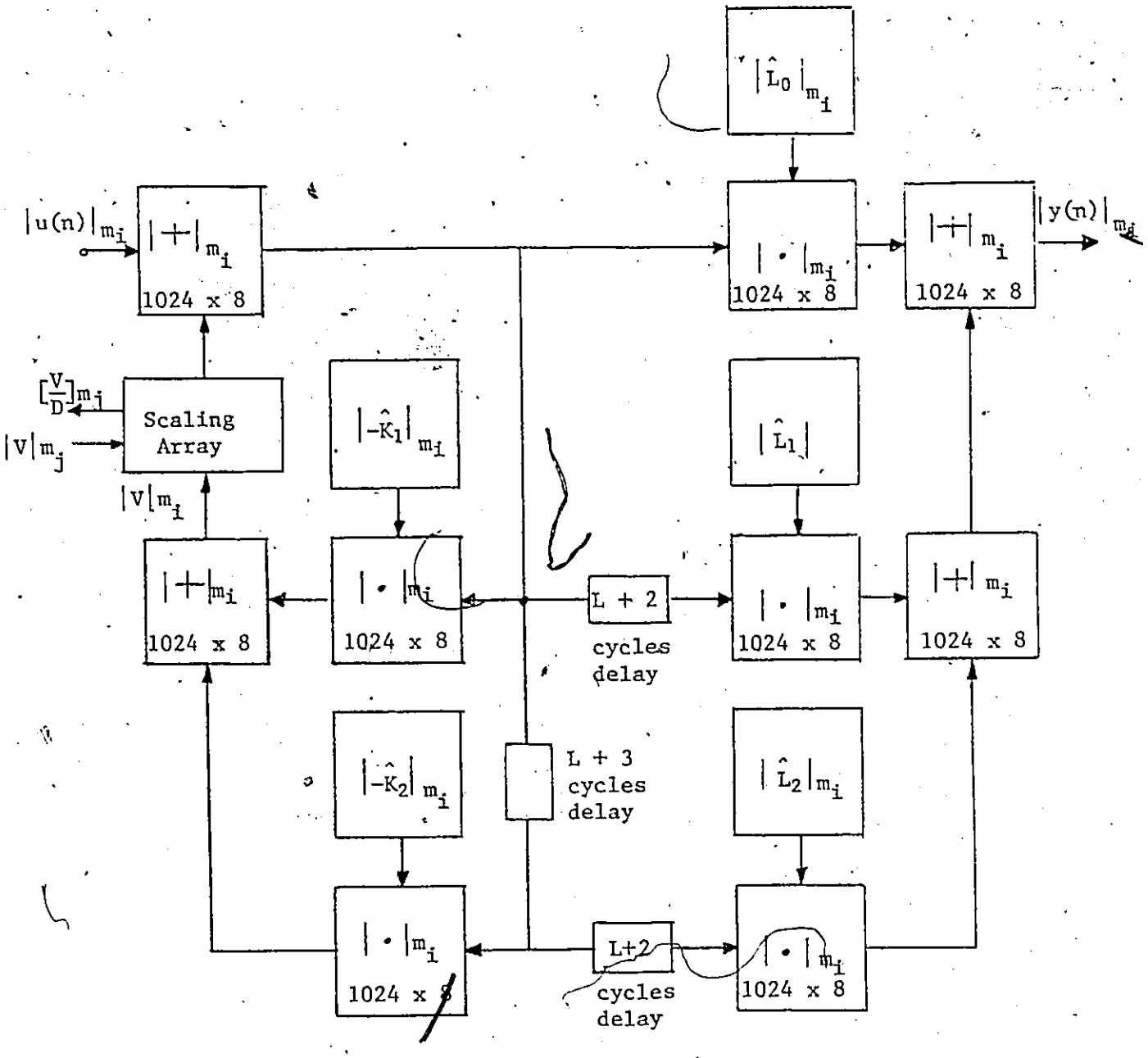


Fig.5.22 ROM implementation of canonic realization second order subfilter for $m_i \leq 32$ and variable filter coefficients

second order section is reduced to 4 x L look-up tables plus the package count for the scaling array. The throughput rate is increased to

$$f_t' = \frac{1}{(L+2)T} \tag{5.70}$$

In this case, the second type of multiplexing mentioned above can be applied.

Assume that the dynamic range of input data and filter coefficients is 2^{10} and 2^{12} respectively. One possible choice for M is, M = 32 x 31 x 29 x 19 x 17, which provides a dynamic range of $\sim$ 23.2 bits. With the scaling factor equal to the produce of two moduli, D = 17 x 19 $\simeq$ $1.2 \cdot 2^{C-2}$, the package count for the 'Exact Division' scaling array is 14 ROMs (equation 5.19). The total package count of the RNS filter is 9 x L + 14 = 59 ROMs, not counting stored tables of filter coefficients. The wordrate of such a five moduli second order section is equal to $\sim$ 1.8 MHz.

When the second order section is multiplexed to generate the parallel configuration of second order sections, the throughput rate is 14 MHz, the package count is reduced to 49 ROMs (since $\hat{L}_2$ is not required for the parallel form).

If the filter coefficients are allowed to be fixed, the total package count of a 5 moduli second order section is 34 ROMs and the wordrate is in excess of 2 MHz. Such a section could be multiplexed between seven signals, having a bandwidth of 1 MHz each.

A variety of cost versus speed options exist, and all of the options provide a viable state-of-the art solution to recursive filtering.

5.6 Summary

This section has covered aspects of RNS implementation of infinite impulse response (recursive) filters suitable for handling adaptively varying coefficients. An analysis of the quantization error associated with scaling in the proposed realization of second order sections has been presented.

The conditions for existence of limit cycles in residue coded recursive sections have been derived for certain cases. Theoretical expressions for predicting relative error due to scaling, and bounds on existence of zero-input limit cycles at the output of second order sections were verified using simulation techniques.

CHAPTER 6

CONCLUSIONS

The objective of the research work described in this dissertation was to develop techniques and algorithms for implementing digital filters using Residue Number System principles. Both finite impulse response and infinite impulse response (recursive) filtering algorithms have been investigated.

The major results, conclusions and contributions may be summarized as follows.

6.1. Finite Impulse Response Digital Filtering via Number Theoretic Transforms

Techniques have been developed for RNS based implementation of Number Theoretic Transforms, as a way of fast, error-free, indirect filtering of real and complex data. By performing arithmetic in the RNS and not using binary arithmetic elements we are free to choose the moduli $\{m_i\}$ and generators $\{\alpha_i\}$ to maximize transform length, restricted only by algebraic considerations.

Defining Number Theoretic Transforms over Galois Fields of second degree, $GF(m_i^2)$, rather than over simple finite field of integers, $GF(m_i)$, leads to greatly increased power of 2 transform length, allowing application of fast algorithms. In the case of filtering real data, two sequences may be simultaneously convolved with the same impulse response.

Computing Transforms over a ring which is isomorphic to a direct sum of L Galois Fields, $\sum_{i=1}^{L} \oplus GF(m_i^2)$, ie, using L parallel NTTs

provides dynamic range $M = \prod_{i=1}^{L} m_i$, where M is greater than the upper bound on the convolution sum.

Within each Galois Field, $GF(m_i^2)$, the most efficient implementation is obtained when the irreducible polynomial (which is used to build the extension field) has the form of the monic binomial.

The result has been obtained, and proved, that for primes of the form $4\xi + 1$ the generator of the cyclic group, α_i, which has maximal power of 2 order, can have a simplified form which allows a significant reduction in the number of binary operations. Techniques for obtaining such generators have been provided. This result allows hardware implementations of NTTs over $GF(m_i^2)$ which are as efficient as those over $GF(m_i)$; the transform length, however, is increased significantly.

A selection of transform parameters has been provided for hardware implementation of Number Theoretic Transform filters for real data, when the component fields are isomorphic to the finite quadratic integers, and for complex data, when the component fields or rings are isomorphic to the residue classes of Gaussian integers.

6.2. Infinite Impulse Response Digital Filtering

Some of the design problems associated with a ROM implementation of Residue Number System coded second order sections, which can be used as a building block for any order, one dimensional recursive filters, have been considered.

Based on the requirement that the filter should handle varying coefficients, as required in multiplexing schemes or adaptive filtering, and also that for efficient realization the number of scaling operations

should be minimized, we have found that the canonic and direct form 1 second order sections are the most viable choice. These forms require the computation of five multiplications and four additions with only one scaling process.

Analysis of the quantization error associated with 'Exact Division' scaling has been presented. In the case of scaling by two or more moduli, this scaling technique has the best error performance (smallest bounds on the scaling error). The signal to noise ratio of the proposed RNS implementation has been shown to be always higher than the signal to noise ratio of conventional binary implementations.

An analysis of zero-input limit cycles (autonomous oscillations) in Residue Number System second order section realizations has been presented. Regions on the coefficient space which allow limit cycles of period 1 (dc) and period 2 have been determined and the bounds on their magnitude have been derived.

6.3. Interface of Residue Number System Structures With Conventional Binary Hardware

A comprehensive review of interface techniques have been presented and a new scheme for translation of residue coded number into a binary form, based on a mixed radix conversion and bit slice technique, has been introduced.

APPENDIX A

LIST OF SYMBOLS

$a \mid b$ a divides b

$(a,b]$ or $\gcd(a,b]$ greatest common divisor of the elements a and b

$a \equiv b \pmod{m}$ integer a is congruent to integer b modulo m

a^{-1} multiplicative inverse of a

$\left| \frac{1}{a} \right|_m$ multiplicative inverse of a modulo m

$C(m) = \{a + \sqrt{-1}\, b \mid a, b \in GF(m)\}$ finite field of Gaussian integers

$F[x]$ set of polynomials in the element x

$F(\lambda)$ the extension field obtained by adjoining λ to field F.

$GF(m^n]$ Galois Field with m^n elements

$GF(m^n) - \{0\}$ multiplicative group of $GF(m^n]$

$GF(m^2] = \{a + \sqrt{r}\, b \mid a, b \in GF(m)\}$ finite field of quadratic integers

I ideal

$H(z]$ transfer function (z transform of the impulse response)

ind a index of a

$\text{ind}_\alpha a$ index of a, relative to α, α a generator of the multiplicative group

L number of moduli

m_i i-th modulus

M product of moduli m_i

r quadratic nonresidue

$T_{GF(m^2]}$ Number Theoretic Transform over $GF(m^2]$

Z_m ring of integers modulo m

$|x|_{m_i}$ the residue of x modulo m_i

$\phi(a)$ Euler's phi function

α generator of the multiplicative group

λ root of an irreducible polynomial $f(x)$

σ^2 variance

$\sum \oplus R_i$ direct sum of rings

$\{a_i\}$ set consisting of the elements a_i

$[\cdot]$ integer part of the term enclosed

$[\cdot]_{R_o}$ the closest integer to the term enclosed

$\left\{ \begin{matrix} m \\ k \end{matrix} \right\}$ binomial coefficients

$h(s) * u(s)$ convolution

$h(s) \circledast u(s)$ cyclic (circular) convolution

APPENDIX B

APPLICATION OF THE BINOMIAL THEOREM TO ELEMENTS OF THE GALOIS FIELD $GF(m^2)$

Let a and b be two elements of the ring R. If m is any positive integer, we have the binomial expansion:

$$(a + b)^m = a^m + \left\{ \begin{matrix} m \\ 1 \end{matrix} \right\} a^{m-1} b + \cdots + \left\{ \begin{matrix} m \\ k \end{matrix} \right\} a^{m-k} b^k + \cdots$$

$$+ \left\{ \begin{matrix} m \\ m-1 \end{matrix} \right\} ab^{m-1} + b^m \qquad (1B)$$

where $\left\{ \begin{matrix} m \\ k \end{matrix} \right\} = \frac{m!}{k!\ (m-k)!} = \frac{m(m-1)\ \cdots\ (m-k+1)}{k!}$

is the usual binomial coefficient.

Binomial coefficients $\left\{ \begin{matrix} m \\ k \end{matrix} \right\}$, $0 \leq k \leq m$, are always integers (e.g., [45], Th.73).

If m is a prime, then all the binomial coefficients in the expansion 1B are divisible by m, because from the definition of a prime no term in the denominator divides m in the numerator.

Equivalently, we can say that $\left\{ \begin{matrix} m \\ k \end{matrix} \right\}$, $k = 1, \ldots, m-1$, is congruent to zero (mod m).

Hence 1B becomes:

$$(a + b)^m \equiv (a^m + b^m) \bmod m \qquad (2B)$$

If a and b are integers, from Fermat's theorem [47] it follows that for prime m

$$(a + b)^m \equiv (a + b) \bmod m \qquad (3B)$$

Let $u = a + \sqrt{r}\, b \in GF(m^2)$, where $a, b \in GF(m)$.

Hence, from eq.2B and 3B

$$(a + \sqrt{r}\, b)^m \equiv (a + \sqrt{r}^m\, b) \bmod m \tag{4B}$$

Since r has to be a quadratic non-residue for $a + \sqrt{r}\, b \in GF(m^2)$, hence [42]

$$r^{\frac{m-1}{2}} \equiv -1 \bmod m \tag{5B}$$

so that $\sqrt{r}^m = r^{\frac{m-1}{2}} \cdot \sqrt{r} \equiv -\sqrt{r} \bmod m$.

Thus, 4B becomes:

$$(a + \sqrt{r}\, b)^m \equiv a - \sqrt{r}\, b \bmod m.$$

APPENDIX C

SIMULATION DETAILS

The digital filtering algorithms described in this thesis were simulated on an IBM / 3031 computer. Listings of four programs are given here.

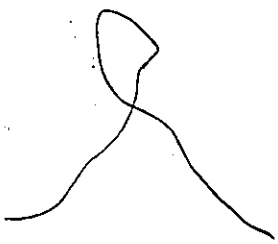
The first program simulates indirect filtering of real or complex data, via Number Theoretic Transform defined over Galois Field $GF(m_i^2)$. The value of the generator α of order N, required for the NTT, can be obtained from the second program for primes $4\xi + 3$; generator α for primes $4\xi + 1$ can be determined from Theorem 4.2 (Chapter 4).

Program 3 simulates second order recursive canonic section incorporating "Exact Division" scaling.

Program 4 was written to simulate the Residue Number System Arithmetic "Exact Division" scaling.

```
C     ***************PROGRAM 1***************
C     ***************************************
C     THIS PROGRAM COMPUTES CONVOLUTION
C     USING NUMBER THEORETIC TRANSFORM
C     OVER GALOIS FIELD OF SECOND DEGREE
C     ***************************************
C     SUBROUTINE REQUIRED - NTT
C     ***************************************
C     MO -MODULUS
C     N -TRANSFORM LENTH
C     M -NUMBER OF STAGES(N=2**M)
C     IR -QUADRATIC NONRESIDUE
C     IBETA -'REAL' PART OF GENERATOR ALPHA
C     IGAMA -'IMAGINARY' PART OF GENERATOR ALPHA
C     IX1, IY1 -'REAL' AND 'IMAGINARY' PARTS OF FIRST SEQUENCE
C     IX2, IY2 -'REAL' AND 'IMAGINARY' PARTS OF SECOND SEQUENCE
C     ***************************************
      INTEGER IX1(128), IY1(128), IX2(128), IY2(128), ITR(128), ITI(128)
      READ, IR, MO, N, M
      READ, IBETA, IGAMA
      READ, (IX1(I), I=1, N)
      READ, (IY1(I), I=1, N)
      READ, (IX2(I), I=1, N)
      READ, (IY2(I), I=1, N)
      NA=IBETA
      NB=IGAMA
      CALL NTT(IX1, IY1, NA, NB, M, N, IR, MO)
      CALL NTT(IX2, IY2, NA, NB, M, N, IR, MO)
C     MULTIPLYING TWO TRANSFORMS
      DO 30 I=1, N
      ITR(I)=IX1(I)*IX2(I)+IR*IY1(I)*IY2(I)
      ITR(I)=MOD(ITR(I), MO)
      ITI(I)=IX1(I)*IY2(I)+IX2(I)*IY1(I)
   30 ITI(I)=MOD(ITI(I), MO)
C     COMPUTING MULTIPLICATIVE INVERSE OF ALPHA
      NA=IBETA
      NB=IGAMA
      NN=N-1
      DO 50 J=2, NN
      IA=NA*IBETA+IR*NB*IGAMA
      IB=NA*IGAMA+NB*IBETA
      IA=MOD(IA, MO)
      IF(IA.LT.0) IA=IA+MO
      IB=MOD(IB, MO)
      IF(IB.LT.0) IB=IB+MO
      NA=IA
   60 NB=IB
C     COMPUTING MULTIPLICATIVE INVERSE OF N
      INV=0
   35 INV=INV+1
      IF(MOD(INV*N, MO).NE.1) GO TO 35
      DO 40 I=1, N
      ITR(I)=MOD(ITR(I)*INV, MO)
   40 ITI(I)=MOD(ITI(I)*INV, MO)
C     COMPUTING INVERSE TRANSFORM
      CALL NTT(ITR, ITI, NA, NB, M, N, IR, MO)
```

```
      DO 50 I=1,N
   50 PRINT, I, IX1(I), IY1(I), IX2(I), IY2(I)
      PRINT 55
   55 FORMAT(' ',7X,'CONVOLUTION')
      DO 60 I=1,N
   60 PRINT, ITR(I), ITI(I)
      STOP
      END
      SUBROUTINE NTT(IX,IY,NA,NB,M,N,IR,MO)
C     ************************************************
C     THIS SUBROUTINE COMPUTES RADIX-2 DIT NTT
      INTEGER IX(128),IY(128)
      NV2=N/2
      NM1=N-1
      J=1
C     SCRAMBLING OF INPUT DATA
      DO 7 I=1,NM1
      IF(I.GE.J) GO TO 5
      IT=IX(J)
      IX(J)=IX(I)
      IX(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
      NA1=NA
      NB1=NB
C     PERFORMING RADIX-2 NTT
      DO 20 L=1,M
      LE=2**L
      LE1=LE/2
      NLE=N/LE
      IUI=0
      IUR=1
      IWR=1
      IWI=0
      DO 15 LI=1,NLE
      IRR=NA1*IWR+IR*NB1*IWI
      IRI=NA1*IWI+NB1*IWR
      IWR=MOD(IRR,MO)
   15 IWI=MOD(IRI,MO)
      NA=IUR
      NB=IUI
      DO 20  J=1,LE1
      DO 10 I=J,N,LE
      IP=I+LE1
      IT1=IX(IP)*NA+IR*IY(IP)*NB
      IT1=MOD(IT1,MO)
      IT2=IY(IP)*NA+IX(IP)*NB
      IT1=MOD(IT1,MO)
      IF(IT2.LT.0) IT2=IT2+MO
      IX(IP)=IX(I)-IT1
      IY(IP)=IY(I)-IT2
      IIX=IX(IP)
      IX(IP)=MOD(IIX,MO)
```



```
      IF(IX(IP).LT.0) IX(IP)=IX(IP)+MO
      IIY=IY(IP)
      IY(IP)=MOD(IIY,MO)
      IF(IY(IP).LT.0) IY(IP)=IY(IP)+MO
      IX(I)=IX(I)+IT1
      IIX=IX(I)
      IX(I)=MOD(IIX,MO)
      IY(I)=IY(I)+IT2
      IIY=IY(I)
      IF (IX(I).LT.0) IX(I)=IX(I)+MO.
      IY(I)=MOD(IIY,MO)
   10 IF(IY(I).LT.0) IY(I)=IY(I)+MO
      IU=IWR*IUR+IR*IWI*IUI
      II=IWR*IUI+IWI*IUR
      IUR=MOD(IU,MO)
      IUI=MOD(II,MO)
      NA=IUR
   20 NB=IUI
      RETURN
      END
```

```
C      ****************PROGRAM 2*****************
C      PROGRAM FOR FINDING COMPLEX PRIMITIVE ROOT OF UNITY ALPHA
C      IN THE GALOIS FIELD
C      ******************************************
C      NA -REAL PART OF ALPHA
C      NB -IMAGINARY PART OF ALPHA
C      N -ORDER OF ALPHA
C      MO-MODULUS
C      ******************************************
       COMPLEX X, W, Y, Z, CMPLX
       READ(5, 1) N, MO
    1  FORMAT(2I3)
       MI=MO-1
       B=0.
       DO 7 M=1, MI
       A=0.
       B=B+1
       DO 7 J=1, MI
       A=A+1
       X=CMPLX(A, B)
       W=CMPLX(A, B)
       DO 3 I=1, N
       W=X*W
       AA=REAL(W)
       BB=AIMAG(W)
       K=INT(AA)
       L=INT(BB)
       KK=MOD(K, MO)
       LL=MOD(L, MO)
       II=I+1
       IF(KK. LT. 0) KK=KK+MO
       IF(LL. LT. 0) LL=LL+MO
       IF((LL. EQ. 0). AND. (KK. EQ. (MO-1))) GO TO 5
       C=KK
       D=LL
       W=CMPLX(C, D)
    3  CONTINUE
    5  IF(II. EQ. N/2) GO TO 10
       IF(II. EQ. N) GO TO 9
    7  CONTINUE
    9  AA=A
       A=A**2-B**2
       B=2*AA*B
       NA=INT(A)
       NB=INT(B)
       NA=MOD(NA, MO)
       NB=MOD(NB, MO)
       IF(NA. LT. 0) NA=NA+MO
       IF(NB. LT. 0) NB=NB+MO
       GO TO 11
   10  NA=INT(A)
       NB=INT(B)
   11  PRINT 12, NA, NB
   12  FORMAT(' ', 'GENERATOR ALPHA', 7X, I3, 9X, I3)
```

```
C     ***********************************************
C     GENERATING POWERS OF ALPHA
      PRINT 13
   13 FORMAT(' ',3X,'POWER',7X,'REAL',7X,'IMAGINARY')
      Z=CMPLX(A,B)
      Y=CMPLX(A,B)
      NI=N-1
      DO 15 I=1,NI
      Z=Y*Z
      E=REAL(Z)
      F=AIMAG(Z)
      NE=INT(E)
      NF=INT(F)
      NEM=MOD(NE,MO)
      NFM=MOD(NF,MO)
      IF(NFM.LT.0) NFM=NFM+MO
      IF(NEM.LT.0) NEM=NEM+MO
      II=I+1
      PRINT,II,NEM,NFM
      EE=NEM
      FF=NFM
      Z=CMPLX(EE,FF)
   15 CONTINUE
      STOP
      END
```

```
C     ***********************PROGRAM 3***********************
C     THIS PROGRAM SIMULATES THE RESIDUE CODED
C     SECOND ORDER CANONIC SECTION
C     AND ALSO COMPUTES RELATIVE MEAN SQUARE ERROR
C     DUE TO 'EXACT DIVISION' SCALING
C     AA1, AA2, BB1, BB2 -FILTER COEFFICIENTS
C          1+AA1*Z**(-1)+AA2*Z**(-2)
C     H(Z)=-----------------------------------
C          1+BB1*Z**(-1)+BB2*Z**(-2)
C     D -SCALE FACTOR
C     P -INTEGER NORMALIZATION FACTOR FOR FILTER COEFFICIENTS
C     FACT -MAXIMUM  ABSOLUTE VALUE OF COEFFICIENTS
C     B -NUMBER OF BITS TO REPRESENT INPUT
C     C -NUMBER OF BITS TO REPRESENT FILTER COEFFICIENTS
C     X1, X2 -INTIAL CONDITIONS(W(N-1) AND W(N-2))
C     SX -INPUT WITH THE RANGE(-1, +1)
C     X -INPUT WITH THE RANGE (-(2**(B-1)), +(2**(B-1)))
C     SYR -RELATIVE MEAN-SQUARE ERROR
C     N - NUMBER OF SAMPLES
C     II -NUMBER OF DIFFERENT SETS OF FILTER COEFFICIENTS
C     ***********************************************
      DOUBLE PRECISION AA1(20), AA2(20), BB1(20), BB2(20)
      INTEGER X
      DOUBLE PRECISION HOLI, HOLS, HOLD
      DOUBLE PRECISION X1(20), X2(20)
      DOUBLE PRECISION X3(20), X4(20)
      DOUBLE PRECISION A1(20), A2(20), B1(20), B2(20)
      DOUBLE PRECISION YY(1000), YYS(1000), E(1000), SUM
      DIMENSION IA1(20), IA2(20), IB1(20), IB2(20)
      DOUBLE PRECISION Y, YS
      FACT=2
      READ(5, 1) II, N, B, C
    1 FORMAT(4I4)
      READ, (AA1(I), I=1, II)
      READ, (AA2(I), I=1, II)
      READ, (X1(I), I=1, II)
      READ, (X2(I), I=1, II)
      READ, (BB1(I), I=1, II)
      READ, (BB2(I), I=1, II)
      P=2. D0**(C-1. D0)/FACT
      DO 100 I=1, II
      IA1(I)=AA1(I)*P+DSIGN(0. 5D0, AA1(I))
      IA2(I)=AA2(I)*P+DSIGN(0. 5D0, AA2(I))
      IB1(I)=BB1(I)*P+DSIGN(0. 5D0, BB1(I))
      IB2(I)=BB2(I)*P+DSIGN(0. 5D0, BB2(I))
      PRINT, IA1(1), IA2(1), IB1(1), IB2(1)
      A1(1)=IA1(1)
      A2(1)=IA2(1)
      B1(1)=IB1(1)
      B2(1)=IB2(1)
      FNOM=P*(22**(C-1))
      IX=35611
      DO 5 J=1, N
```

```
C       DEFINE INPUT SEQUENCE
        CALL RANDU(IX, IY YFL)
        IX=IY
        SX=2*(YFL-0.5)
        X=2.D0**(B-1.D0)*SX
        CALL CANON(A1, A2, B1, B2, X1, X2, X, Y, I, P)
        CALL CANOS(A1, A2, B1, B2, X1, X2, X, YS, I, P)
        YYS(J)=YS
        YY(J)=Y
    5   CONTINUE
        SUM=0
        SY=0
        DO 20 KJ=1, N
        E(KJ)=(YYS(KJ)-YY(KJ))**2
        SY=SY+YYS(KJ)**2
   20   SUM=SUM+E(KJ)
        SYR=SUM/SY
  100   CONTINUE
        STOP
        END
        SUBROUTINE CANON (A1, A2, B1, B2, X3, X4, X, Y, I, P)
C       ***********************************************
C       THIS SUBROUTINE IMPLEMENTS SECOND ORDER CANONIC SECTION
C       USING DOUBLE PRECISION FLOATING POINT ARITHMETIC
        INTEGER X
        DOUBLE PRECISION Y
        DOUBLE PRECISION A1(20), A2(20), B1(20), B2(20), X1(20), X2(20)
        DOUBLE PRECISION X3(20), X4(20)
        DOUBLE PRECISION HOLI, HOLS, HOLD
        D=P
        HOLI=-B1(I)*X3(I)-B2(I)*X4(I)
        HOLS=HOLI/D
        HOLD=X+HOLS
        Y=HOLD*P+A1(I)*X3(I)+A2(I)*X4(I)
        X4(I)=X3(I)
        X3(I)=HOLD
        RETURN
        END
        SUBROUTINE CANOS(A1, A2, B1, B2, X1, X2, X, YS, I, P)
C       ***********************************************
C       THIS SUBROUTINE SIMULATES THE RESIDUE CODED SECOND ORDER
C       CANONIC SECTION WITH 'EXACT DIVISION' ROUNDOFF SCALING
        INTEGER X
        DOUBLE PRECISION YS
        DOUBLE PRECISION A1(20), A2(20), B1(20), B2(20), X1(20), X2(20)
        DOUBLE PRECISION HOLI, HOLS, HOLD
        D=P
        X=X*P
        HOLI=-B1(I)*X1(I)-B2(I)*X2(I)+X
        HOLS=HOLI/D+DSIGN(0.5D0, HOLI)
        HOLS=IDINT(HOLS)
        IHOL=HOLS
        HOLD=IHOL
        IHOL=IHOL-1
        YS=HOLD*P+A1(I)*X1(I)+A2(I)*X2(I)
        X2(I)=X1(I)
        X1(I)=HOLD
        RETURN
        END
```

```
C     ***************PROGRAM 4 ***********************
C     THIS PROGRAM IMPLEMENTS RESIDUE NUMBER SYSTEM
C     'EXACT DIVISION ' SCALING
C     BY THE PRODUCT OF FIRST 'N1' MODULI IN 'N'MODULI RNS SYSTEM
C     ALGORITHM FOR 'EXACT DIVISION' SCALING IS GIVEN IN REF(4)
C     N -NUMBER OF MODULI
C     N1 -NUMBER OF MODULI FOR SCALING
C     M(I) -MODULI
C     X(I) -RESIDUES MODULO M(I) OF NUMBER TO BE SCALED
C     Y(I) -RESIDUES MODULO M(I) OF SCALED NUMBER
      INTEGER P,W,FSUM,SUM,M(10),X(10),TI(10),TJ(10),F(10),Y(10),R(10)
      INTEGER D
      READ(5,11) N,N1
   11 FORMAT(2I2)
      READ,(M(I),I=1,N)
      READ,(X(I),I=1,N)
      PRINT 1
    1 FORMAT(' ','CALCULATION OF T1(I,I)T1(I,J),F(I)')
      M1=2
      DO 30 I=M1,N
      F(1)=X(1)
      IF(I-N1) 55,55,65
   55 L=I-1
      SUM=0
      DO 22 J=1,L
      P=1
      DO 20 K=J,L
C     COMPUTING MULTIPLICATIVE INVERSE OF M(K) MODULO M(I)
      MULTIN=M(K)**(M(I)-2)-M(I)*(M(K)**(M(I)-2)/M(I))
      P=MULTIN*P
   20 CONTINUE
      W=-P*F(J)
      IF(W) 10,40,40
   10 W=M(I)+W
      IF(W) 10,40,40
   40 TJ(J)=MOD(W,M(I))
   17 PRINT 3,I,J,TJ(J)
    3 FORMAT(' ','I=',I3,3X,'J=',I3,3X,'T1(I,J)=',I4)
   18 SUM=SUM+TJ(J)
   22 CONTINUE
      P=1
      DO 25 K=1,L
      MULTIN=M(K)**(M(I)-2)-M(I)*(M(K)**(M(I)-2)/M(I))
      P=MULTIN*P
   25 CONTINUE
      W=P*X(I)
      IF(W) 50,60,60
   50 W=M(I)+W
      IF(W) 50,60,60
   60 TI(I)=MOD(W,M(I))
      FSUM=SUM+TI(I)
      F(I)=FSUM-M(I)*(FSUM/M(I))
   27 PRINT 4,I,TI(I)
    4 FORMAT(' ','I=',I3,11X,'T1(I,I)=',I4)
```

```
      PRINT 5, I, F(I)
    5 FORMAT(' ', 'I=', I3, 11X, 'F(I)=', I4)
      GO TO 30
   65 L=N1
      L1=I-1
      PRINT 2
    2 FORMAT(' ', 'CALCULATION OF T2(I, I), T2(I, J), Y(I)')
      SUM=0
      DO 32 J=1, L
      P=1
      DO 33 K=J, L
      MULTIN=M(K)**(M(I)-2)-M(I)*(M(K)**(M(I)-2)/M(I))
      P=MULTIN*P
   33 CONTINUE
      W=-P*F(J)
      IF(W) 35, 45, 45
   35 W=M(I)+W
      IF(W) 35, 45, 45
   45 TJ(J)=MOD(W, M(I))
      PRINT 7, I, J, TJ(J)
    7 FORMAT(' ', 'I=', I3, 3X, 'J=', I3, 3X, 'T2(I, J)=', I4)
      SUM=SUM+TJ(J)
   32 CONTINUE
      P=1
      DO 36 K=1, L
      MULTIN=M(K)**(M(I)-2)-M(I)*(M(K)**(M(I)-2)/M(I))
      P=MULTIN*P
   36 CONTINUE
      W=P*X(I)
      IF(W)80, 90, 90
   80 W=M(I)+W
      IF(W) 80, 90, 90
   90 TI(I)=MOD(W, M(I))
      PRINT 8, I, TI(I)
    8 FORMAT(' ', 'I=', I3, 11X, 'T2(I, I)=', I4)
      FSUM=SUM+TI(I)
      Y(I)=FSUM-M(I)*(FSUM/M(I))
      PRINT 9, I, Y(I)
    9 FORMAT(' ', 'I=', I3, 11X, 'Y(I)=', I4)
      R(1)=Y(N1+1)
      IF(I-N1-2) 30, 100, 100
  100 L=I-N1-1
      PRINT 120
  120 FORMAT(' ', 'CALCULATION OF T3(I, I), T3(I, J), R(I-N1)')
      SUM=0
      DO 105 J=1, L
      P=1
      L2=J+N1
      DO 110 K=L2, L1
      MULTIN=M(K)**(M(I)-2)-M(I)*(M(K)**(M(I)-2)/M(I))
      P=MULTIN*P
  110 CONTINUE
      W=-P*R(J)
      IF(W)115, 125, 125
  115 W=M(I)+W
      IF(W)115, 125, 125
  125 TJ(J)=MOD(W, M(I))
```

```
      PRINT 127, I, J, TJ(J)
  127 FORMAT(' ', 'I=', I3, 3X, 'J=', I3, 3X, 'T3(I, J)=', I4)
      SUM=SUM+TJ(J)
  105 CONTINUE
      P=1
      L3=N1+1
      DO 126 K=L3, L1
      MULTIN=M(K)**(M(I)-2)-M(I)*(M(K)**(M(I)-2)/M(I))
      P=MULTIN*P
      W=P*Y(I)
  126 CONTINUE
      IF(W) 130, 140, 140
  130 W=M(I)+W
      IF(W) 130, 140, 140
  140 TI(I)=MOD(W, M(I))
      PRINT 155, I, TI(I)
  155 FORMAT(' ', 'I=', I3, 11X, 'T3(I, I)=', I4)
      FSUM=SUM+TI(I)
      N3=I-N1
      R(N3)=FSUM-M(I)*(FSUM/M(I))
      PRINT 170, I, R(N3)
  170  FORMAT (' ', 'I=', I3, 11X, 'R(I-N1)=', I4)
      IF(I-N) 30, 300, 300
  300 PRINT 202
  202 FORMAT(' ', 'CALCULATION OF T4(I, J)Y(I)')
      DO 200 I2=1, N1
      SUM=0
      J=1
      W=R(1)
      TJ(J)=MOD(W, M(I2))
      PRINT 201, I2, J, TJ(J)
  201  FORMAT (' ', 'I=', I3, 3X, 'J=', I3, 3X, 'T4(I, 1)=', I4)
      L3=N-N1-1
      L4=N1+1
      IF(L3. LT. 2) GO TO 225
      DO 210 J=2, L3
      P=1
      L5=J+N1-1
      DO 220 K=L4, L5
      P=M(K)*P
  220 CONTINUE
      W=P*R(J)
      IF(W) 230, 240, 240
  230 W=M(I2)+W
      IF(W) 230, 240, 240
  240 TJ(J)=MOD(W, M(I2))
      PRINT 250, I2, J, TJ(J)
  250  FORMAT (' ', 'I=', I3, 3X, 'J=', I3, 3X, 'TJ(I, J)=', I4)
      SUM=SUM+TJ(J)
  210 CONTINUE
  225 J=N-N1
      L6=N-1
      P=1
      DO 260 K=L4, L6
      P=M(K)*P
```

```
      D=0
      IF(R(J).GE.(M(N)/2)) D=1
      W=P*(R(J)-D*M(N))
  260 CONTINUE
      IF(W) 270,280,280
  270 W=M(I2)+W
      IF(W) 270,280,280
  280 TJ(J)=MOD(W,M(I2))
      PRINT 235,I2,J,TJ(J)
  235  FORMAT (' ','I=',I3,3X,'J=',I3,3X,'T4(I,N-N1)=',I4)
      FSUM=SUM+TJ(1)+TJ(J)
      Y(I2)=FSUM-M(I2)*(FSUM/M(I2))
      PRINT 290,I2,Y(I2)
  290  FORMAT (' ','I=',I3,11X,'Y(I)=',I4)
  200 CONTINUE
   30 CONTINUE
      STOP
      END
```

BIBLIOGRAPHY

[1] A. Svoboda, "Rational Numerical Systems of Residual Classes", Stroje Na Zpracovani Informaci, pp.1-29, Sbornik V, Prague, Czechoslovakia, 1957.

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

[3] W.K. Jenkins and B.J. Leon, "The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters", IEEE Trans. on Circuits and Systems, Vol.CAS-24, No.4, April 1977, pp.191-201.

[4] 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.

[5] W.K. Jenkins, "Recent Advances in Residue Number System Techniques for Recursive Digital Filtering", IEEE Trans. on ASSP-27, No.1, Feb.1979, pp.19-30.

[6] 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.

[7] G.A. Jullien, W.C. Miller, J.J. Soltis, A. 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.

[8] A. Baraniecka and G.A. Jullien, "Hardware Implementation of Convolution Using Number Theoretic Transforms", Proc. of IEEE International Conf. on Acoustics, Speech and Signal Processing, Washington, D.C., April 1979.

[9] W.K. Jenkins, "Techniques for High Precision Filtering with Multiple Microprocessors", Proc. 20th Midwest Symp. Circuits Syst., pp. 58-62, 1977.

[10] M.A. Soderstrand, "A Universal Digital Filter for Use with 8-Bit Microprocessors", 20th Midwest Symp. on Circuits and Syst., 1978.

[11] G.A. Jullien, "Implementation of Multiplication Modulo a Prime Number with Application to Number Theoretic Transforms", accepted for publication in IEEE Trans. on Computers.

[12] D.K. Banerji, "A Novel Implementation Method for Addition and Subtraction in Residue Number Systems", IEEE Trans. on Computers, Vol.C-23, No.1, January 1974, pp.106-109.

[13] S. You and J. Chung, "On the Design of Modulo Arithmetic Units Based on Cyclic Groups", IEEE Trans. on Computers, Vol.C-25, No.11, Nov.1976, pp.1057-1067.

[14] "Bipolar LSI Data Book", Monolitic Memories, 1978.

[15] A. Baraniecka and G.A. Jullien, "On Decoding Techniques for Residue Number System Realizations of Digital Signal Processing Hardware", IEEE Trans. on Circuits and Systems, Vol.CAS-25, No.10, Nov.1978, pp.935-936.

[16] W.K. Jenkins, "Techniques for Residue-to-Analog Conversion for Residue-Encoded Digital Filters", IEEE Trans. on Circuits and Systems (Special Issue on Analog/Digital Conversion), Vol.CAS-27, No.7, July 1978, pp.555-562.

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

[18] L.R. Rabiner and B. Gold, "Theory and Application of Digital Signal Processing", Prentice-Hall, 1975.

[19] J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Math. Comp., V.19, 1965, pp.297-301.

[20] G.D. Bergland, "A Guided Tour of the Fast Fourier Transform", IEEE Spectrum, V.6, No.7, 1969, pp.41-53.

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

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

[23] C.M. Rader, "The Number Theoretic DFT and Exact Discrete Convolution", IEEE Arden House Workshop on Digital Signal Processing, Jan.11, 1972.

[24] R.C. Agarwal and C.S. Burrus, "Fast Convolution Using Fermat Number Transforms with Application to Digital Filtering", IEEE Trans. Acoust., Speech, Signal Processing, Vol.ASSP-22, April 1974, pp.87-97.

[25] R.C. Agarwal and C.S. Burrus, "Number Theoretic Transforms to Implement Fast Digital Convolution", Proc.IEEE, Vol.63, April 1975, pp.550-560.

[26] E. Dubois and A.N. Venetsanopoulos, "The Discrete Fourier Transform Over Finite Rings with Application to Fast Convolution", IEEE Trans. Comput., Vol.C-17, No.7, July 1978, pp.586-593.

[27] J.H. McClellan, "Hardware Realization of a Fermat Number Transform" IEEE Trans. Acoust., Speech, Signal Processing, Vol.ASSP-24, June 1976, pp.216-225.

[28] C.M. Rader, "Discrete Convolutions via Mersenne Transforms", IEEE Trans. Comput., Vol.C-21, Dec.1972, pp.1269-1273.

[29] H. Nussbaumer, "Digital Filtering Using Pseudo-Fermat Number Transforms", IEEE Trans. Acoust., Speech, Signal Processing, Vol.ASSP-25, Feb.1977, pp.79-83.

[30] H. Nussbaumer, "Digital Filtering Using Complex Mersenne Transforms" IBM J. Res. Develop., Vol.20, September 1976, pp.498-504.

[31] J.M. Pollard, "Implementation of Number Theoretic Transforms", Electronic Letters, Vol.12, July 22, 1976, pp.378-379.

[32] K.Y. Liu, I.S. Reed and T.K. Truong, "Fast Number Theoretic Transforms for Digital Filtering", Electronic Letters, Vol.12, Nov.1976, pp.644-646.

[33] 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.

[34] 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.

[35] 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.

[36] I.S. Reed and T.K. Truong, "Complex Integer Convolution over a Direct Sum of Galois Fields", IEEE Trans. Acoust., Speech, Signal Processing, Vol.ASSP-23, Dec.1975, pp.657-661.

[37] K.Y. Liu, I.S. Reed and T.K. Truong, "Fast Algorithms for Complex Integer Transforms", IEEE Trans. Acoust., Speech, Signal Processing,

Vol.ASSP-25, October 1977, pp.450-452.

[38] H.J. Nussbaumer, "Complex Convolutions via Fermat Number Transforms" IBM J. Res. Develop., Vol.20, May 1976, pp.282-284.

[39] I.S. Reed and J.K. Truong, "Convolutions over Residue Classes of Quadratic Integers", IEEE Trans. Inform. Theory, Vol.IT-22, July 1976, pp.468-475.

[40] W.K. Jenkins, "Composite Number Theoretic Transforms for Digital Filtering", Conf. Record of the Ninth Annual Asilomar Conf. on Circuits, Systems and Computers, November 1975, pp.458-462.

[41] L. Gaal, "Classical Galois Theory", Chelsea Publishing Co., New York, 1971.

[42] I.M. Vinogradov, "Elements of Number Theory", New York, Dover, 1954.

[43] U. Dudley, "Elementary Number Theory", W.H. Freeman and Co., 1969.

[44] A. Baraniecka and G.A. Jullien, "Residue Number System Implementations of Number Theoretic Transforms in Complex Residue Ring", accepted for publication in IEEE Trans. on Acoust., Speech Signal Processing, June 1980.

[45] G.H. Hardy and E.M. Wright, "An Introduction to the Theory of Numbers", 4th ed., Oxford Clarendon Press, 1960.

[46] D.M. Burton, "A First Course in Rings and Ideals", Addison-Wesley Publ.Co., 1970.

[47] J.E. Shockley, "Introduction to Number Theory", Holt, Rinehart and Winston Inc., 1967.

[48] W.K.Jenkins and B.J. Leon, "Algebraic Techniques for the Analysis and Design of Digital Filters", Technical Report TR-EE 74-27, School of Electrical Engineering, Purdue University, West Lafayette, Indiana, August 1974.

[49] W.K. Jenkins, "A New Algorithm for Scaling in Residue Number Systems with Applications to Recursive Digital Filtering", in Proc.1977 IEEE International Symp.Circuits and Systems, New York 1977, pp.56-59.

[50] G.A. Jullien and W.K. Jenkins, "The Application of Residue Number Systems to Digital Signal Processing", submitted for publication to Trans. ASSP.

[51] L.T. Bruton, "Low Sensitivity Digital Ladder Filters", IEEE Trans. on Circuits and Systems, Vol.CAS-22, No.3, March 1975, pp.168-176.

[52] F. Barsi and P. Maestrini, "Error Correcting Properties of Redundant Residue Number Systems", IEEE Trans. on Computers, Vol.C-22, No.3, March 1973, pp.307-315.

[53] 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.

[54] 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.

[55] B. Liu, "Effects of Finite Wordlength on the Accuracy of Digital Filters - A Review", IEEE Trans. Circuit Theory (Special Issue on Active and Digital Networks), Vol.CT-18, Nov.1971, pp.670-677.

[56] T.A. Claasen, W. Mecklenbräuker and J.B. Peck, "Effects of Quantication and Overflow in Recursive Digital Filters", IEEE Trans. Acoust., Speech, Signal Proc., Vol.ASSP-24, Dec.1976, pp.517-529.

[57] W.R. Bennet, "Spectra of Quantized Signals", Bell Syst.Tech.J., Vol.27, July 1948, pp.446-472

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

[59] A.V. Oppenheim and R.W. Schafer, "Digital Signal Processing", Englewood Cliffs, N.J., Prentice-Hall, 1975.

[60] K.J. Astrom, E.I. Jury and R.G. Agniel, "A Numerical Method for the Evaluation of Complex Integrals", IEEE Trans. Automat. Contr., Vol.AC-13, Aug.1970, pp.468-471.

[61] L.B. Jackson, "An Analysis of Limit Cycles Due to Multiplication Rounding in Recursive Digital Filters", in Proc.7th Annual Allerton Conf., October 1969.

[62] T.A. Claasen, W. Mecklenbrauker and J. Peck, "Some Remarks on the Classification of Limit Cycles in Digital Filters", Philips Res. Rep., Vol.28, Aug.1973, pp.297-305.

[63] C.W. Barnes and A.T. Fam, "Minimum Norm Recursive Digital Filters That are Free of Overflow Limit Cycles", IEEE Trans. Circuits Syst. Vol.CAS-24, Oct.1977, pp.569-574.

[64] E. Jury, "Theory and Application of the Z-Transform Method", Wiley, 1964.

[65] A. Famm and C. Barnes, "Non-minimal Realizations of Fixed-Point Digital Filters that are Free of All Finite Wordlength Limit Cycles" IEEE Trans. ASSP, Vol.ASSP-17, April 1979, pp.149-153.

[66] B. Gold and T. Bially, "Parallelism in Fast Fourier Transform Hardware", IEEE Trans. on Audio & Electroacoustics, Vol. AU-21, No.1, Feb.1973, pp.5-16.

[67] D.J. Britten and F.W. Lemire, "A Structure Theorem for Rings Supporting a Discrete Fourier Transform", Math.Dept., University of Windsor.

[68] W.K. Jenkins, "A Highly Efficient Residue-Combinational Architecture for Digital Filters", Proc. IEEE (Lett.) Vol.66, No.6, June 1978, pp.700-702.

[69] N. McCoy, "Fundamentals of Abstract Algebra", Allyn and Bacon Inc., 1972.

[70] G. Birkhoff, "A Survey of Modern Algebra", The MacMillan Co., 1968.

VITA AUCTORIS

Anna Z. Baraniecka

1947 Born on the 25th of March in Lublin, Poland.

1965 Completed High School "Sempolowska", Warsaw, Poland.

1970 Graduated from the Technical University of Warsaw, Faculty of Electronics, Warsaw, Poland, with the degree of M.Sc. and Dipl.Ing.

1970-75 Working as Research and Teaching Assistant at Technical University of Warsaw, Poland.

1980 Candidate for the degree of Doctor of Philosophy in Electrical Engineering, University of Windsor, Windsor, Ontario, Canada.