1

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

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

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

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

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

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

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

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

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

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

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

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

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

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses
on Microfiche Service

Services des thèses canadiennes
sur microfiches

Ottawa, Canada
K1A 0N4

60001

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

PAUL RICHARD DOERR

Date of Birth — Date de naissance

DEC 8, 1958

Country of Birth — Lieu de naissance

CANADA

Permanent Address — Résidence fixe

APT 904, 470 CAMBRIDGE ST S
OTTAWA, ONT K1S 4H8

Title of Thesis — Titre de la thèse

RESIDUE NUMBER ARITHMETIC FOR VLSI
IMPLEMENTATION OF TRANSVERSAL FILTERS

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

M.ENG

Year this degree conferred — Année d'obtention de ce grade

1983

Name of Supervisor — Nom du directeur de thèse

JOHN KNIGHT

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication, ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur

Date

SEPT 27, 1983

Signature

Paul Doerr

RESIDUE NUMBER ARITHMETIC
FOR VLSI IMPLEMENTATION OF TRANSVERSAL FILTERS

by

© Paul R. Doerr

A thesis submitted to the Faculty of Graduate
Studies and Research in partial fulfillment
of the requirements for the degree of

Master of Engineering

Department of Electronics
Faculty of Engineering
Carleton University
Ottawa, Canada
September 1983

The undersigned recommend to the Faculty of Graduate Studies and Research the acceptance of the thesis:

"Residue Number Arithmetic for VLSI Implementation of Transversal Filters"

submitted by Paul R. Doerr in partial fulfillment of the requirements for the degree of Master of Engineering.

Prof. J.P. Knight
Thesis Supervisor

Prof. A.R. Boothroyd
Chairman
Department of Electronics

ABSTRACT

A residue number system based digital FIR filter is described. Advantages over conventional binary number filters include faster switching speeds and a linear increase in design complexity with respect to increasing word lengths (as compared to a squared increase of binary systems). VLSI implementation would normally be impractical, due to the excessive size of ROM stored arithmetic function tables. A new symmetric residue memory compression algorithm is presented that reduces table size by a factor of four, such that VLSI circuits containing sixteen or thirty-two taps become feasible. Considerations for VLSI implementation of four and five bit modulus filter circuits are discussed, including some rough chip area estimates.

ACKNOWLEDGEMENTS

I wish to thank Prof. J.P. Knight for his support and guidance in the preparation of this thesis. Relevant information provided by Dr. D. Roy of Interactive Circuits and Systems was useful in directing the course of this study.

Now that the experience of being a full-time student is (at least temporarily) behind me, I can look back and appreciate once again the help and advice I received from family and friends... and I wish to thank all of you for your patience.

TABLE OF CONTENTS

	PAGE
LIST OF SYMBOLS	(iii)
CHAPTER 1.0 INTRODUCTION	1.1
1.1 Summary	1.2
CHAPTER 2.0 THE RESIDUE NUMBER SYSTEM	2.1
2.1 Residue Arithmetic	2.2
2.2 Binary to Residue Conversion	2.6
2.2.1 Residue/Binary Conversion	2.7
2.3 Simplified Residue Multiplication Using Index Calculus	2.14
2.4 Summary	2.16
CHAPTER 3.0 METHODS FOR DIGITAL FILTERING	3.1
3.1 Filter Specifications	3.1
3.2 Methods of Filter Realization	3.2
3.2.1 Convolution in a Transform Domain	3.3
3.2.2 Direct Form Convolution	3.13
3.3 Residue Number Transversal Filtering	3.19
3.3.1 Residue Multiplication and Addition	3.22
3.4 Summary	3.24
CHAPTER 4.0 RESIDUE FILTER DESIGN	4.1
4.1 Memory Reduction in Stored Residue Function Tables	4.1
4.1.1 The Use of Diagonal Symmetry in Product Table Compression	4.4
4.1.2 An Additive Inverse Method for Table Compression	4.9
4.1.3 A New Compression Scheme Using Symmetric Residue Coding	4.10
4.1.4 Table Compression for Moduli Equal to 2^n	4.15
4.1.5 Memory Reduction in Residue Addition Tables	4.16
4.1.6 Designing a Symmetric Residue System	4.20
4.2 VLSI Design	4.23
4.2.1 Modulus Selection	4.24
4.2.2 Accumulation Circuitry	4.26
4.2.3 Fast Accumulation Using Random Logic Binary Addition	4.29
4.2.4 Block Diagram of a Residue Number Filter Chip	4.38
4.3 Conclusions	4.40
CHAPTER 5.0 TECHNOLOGY CONSIDERATIONS	5.1
5.1 Estimation of Circuit Complexity	5.1

5.2	CMOS Implementation	5.3
5.3	NMOS Implementation	5.6
5.4	Bipolar Implementation	5.8
5.5	Summary	5.9
CHAPTER 6.0	CONCLUSIONS	6.1
6.1	Recommendations	6.5
REFERENCES		R.1
APPENDIX		A.1

LIST OF SYMBOLS

ABS	Absolute value
M	Range of a residue number system, where $M = \prod_{i=0}^{N-1} m_i$
m_i	i'th modulus base
$\bar{m}_i$	$\bar{m}_i = M/m_i$, used in Chinese Remainder Theorem conversion
r_i	i'th residue digit
RND	Integer division rounded off to the nearest integer
$[x/m]$	integer division of x by m (truncated fraction)
$ x _m$	residue of x modulus m
$/x/_{\bar{m}}$	symmetric residue of x modulus m
$ 1/x _m$	multiplicative inverse of x modulus m
$\{r_1, \dots, r_{N-1}\}$	set of residue digits
$\oplus$	Exclusive OR

1.0 INTRODUCTION

Current signal processing systems are limited by the speed at which data can be manipulated. A critical requirement of these devices is the ability to process information in real time. For example, it is now possible to identify an iceberg from transmitted satellite image data, yet this information will only be of academic interest if it becomes available days after a Titanic is sunk. In the same fashion, it is of little use to recognize a sea-skimming missile (such as an Exocet missile) seconds after it has reached its destination.

Even in situations where processing time is not a limiting factor, analyzing data in less than real time can create a tremendous backlog of information. Unless the input is disabled, any storage medium, regardless of its maximum capacity will eventually overflow.

An integral part of these signal processing algorithms is the transversal or finite-impulse response filter. In the past, analog devices were often utilized to perform this operation, but they lacked the versatility available in a digital environment.

Unfortunately, a digital filter can be exceedingly complex, often involving more than a hundred SSI and MSI components. The obvious solution is to construct a single general purpose VLSI circuit. This has not proved to be a simple task, due to a wide range of filter specifications.

As digital word lengths increase (allowing greater signal to noise ratios), a flaw inherent in existing filter chips will become a

critical design factor. The number of gates in a cascadable binary circuit increases with the square of the word size; if this expansion is continued, many applications will require an unrealistic amount of circuitry.

An alternative to this approach is a filter based on residue or modular arithmetic. Such circuits have a linear relationship between size and digital word length. A circuit constructed in this fashion will have excellent adaptability, allowing fast accurate digital filtering with a minimum cost.

1.1 SUMMARY

Following a brief discussion of the properties of modular arithmetic, it will be shown that such arithmetic offers a practical alternative to some of the more 'standard' methods of digital FIR filtering. The cascadability of current single chip digital filters is shown to be limited when accurate (low quantization error) data manipulation is desired.

The need for fast processing combined with a minimum of design complexity indicates that stored function arithmetic tables offer the best result. However, integrating a complex filter onto a single chip would require an inordinate amount of memory, rendering the design impractical.

In this thesis a new method of table compression is described that can reduce memory by as much as a factor of four. When implemented as part of a single chip cascadable filter design, this

approach becomes not only practical, but better than many comparable commercial circuits. A filter of this type is easily expandable, to allow for future requirements of greater accuracy (greater bit length) or greater complexity (more poles).

Some rough calculations demonstrate that a residue circuit will require a large (but not impossible) chip area. These estimates are given for NMOS, CMOS, and bipolar versions, but the complexity of the design is such that the approximations may bear little resemblance to reality. In any case, they are only included for completeness, as the designs are intended to be technology independent. Indeed, many of the proposed techniques are especially suitable for hard-wired circuits, using standard digital components.

2.0 THE RESIDUE NUMBER SYSTEM

A residue digit r of an integer x is the smallest positive remainder of x when divided by the base or modulus m (another positive integer). The notation given below will be used to represent operations in residue arithmetic:

$[x/m]$ integer division of x by m .

$r = |x| \dots\dots\dots \text{residue of } x \text{ modulus } m.$
 m

A residue r is then calculated as:

$$r = x - m[x/m] \quad (2.1)$$

Derivation of residue number properties has been presented by Garner [2.1], Szabo and Tanaka [2.2], and Knuth [2.3]. More recently, McClellan and Rader [2.4] briefly discussed elementary number theory as a basis for residue arithmetic and number theoretic transforms (NTT's).

A residue number system is a method of uniquely representing an integer using a set of N residues based on a set of N different integer bases or moduli:

$$x = \{r_0, r_1, \dots, r_{N-1}\}$$

The maximum value of x that can be uniquely represented is limited by the number of different combinations or states of the residue digits. Since each residue digit r has m possible values, this 'dynamic range' M or maximum value is given by:

$$M = \prod_{i=0}^{N-1} m_i \quad (2.2)$$

If common factors exist between any of the modulus bases, then the maximum range is reduced to:

$$M = \left(\prod_{i=0}^{N-1} m_i \right) / (\text{product of all common factors}) \quad (2.3)$$

The following example should serve to illustrate dynamic range and conversion of an integer to residue notation.

Given: 3 modulus system, such that-

$$\begin{matrix} m_0 = 3, & m_1 = 5, & m_2 = 8 & \dots \text{all relatively prime} \end{matrix}$$

$$\text{Dynamic range } M = \prod_{i=0}^2 m_i = 120$$

Let $x = 56$

$$\text{From equation (2.2), } r_i = x - m_i [x/m_i]$$

$$\begin{matrix} r_0 = 56 - 3[56/3] = 2, & r_1 = 56 - 5[56/5] = 1, & r_2 = 56 - 8[56/8] = 0 \end{matrix}$$

Then $x = \{2, 1, 0\}$

Conversion from residue to binary notation is possible using the Chinese remainder theorem or with mixed radix conversion, as described in section 2.2.

2.1 RESIDUE ARITHMETIC

Using the definitions of the previous section, the following arithmetic identities can be developed. Some derivations and additional information on residue numbers are given in Appendix I.

$$\left| \frac{x+km}{m} \right| = \left| \frac{x}{m} \right| \quad (2.4)$$

$$|x+kM|_m = |x|_m \quad (2.5)$$

$$|kx|_{km} = k|x|_m \quad (2.6)$$

Equation (2.5) demonstrates that the residue system is cyclic with period M . Any operation on a residue number which results in a value outside of the range will cause an overflow. Detection of this state is impossible without additional (and complex) precautions, since the apparent result will be indistinguishable from a legitimate value inside the range. In designing a modular system, all possible results must be within the dynamic range of the moduli, or an erroneous output will occur.

Integer addition, subtraction and multiplication are performed efficiently using modular arithmetic. Some properties of these functions are:

$$||x|_m + |y|_m|_m = |x+y|_m \quad (2.7)$$

$$||x|_m |y|_m|_m = |xy|_m \quad (2.8)$$

With the set of moduli defined in the previous example, these identities can be illustrated:

$$\text{Given: } m_0 = 3, m_1 = 5, m_2 = 8$$

$$x = 17 = \{2, 2, 1\}, \quad y = 6 = \{0, 1, 6\}$$

$$x+y = \{|17+6|_3, |17+6|_5, |17+6|_8\} = \{2, 3, 7\}$$

$$xy = \{|17(6)|_3, |17(6)|_5, |17(6)|_8\} = \{0, 2, 6\}$$

The same results can be obtained in the residue system:

$$x+y = \{2,2,1\} + \{0,1,6\} = \{ |2+0|_3, |2+1|_5, |1+6|_8 \} = \{2,3,7\}$$

$$\begin{aligned} xy &= \{2,2,1\}\{0,1,6\} \\ &= \{ |(2)(0)|_3, |(2)(1)|_5, |(1)(6)|_8 \} = \{0,2,6\} \end{aligned}$$

In a modular system, there are M unique states, hence M integers can be represented without repetition. The integer range may start at any value, but the final value (to insure unique results) must be within M units of the initial integer, i.e.:

$$a \leq x < a+M \quad (2.9)$$

Commonly, the range either begins at zero, or is symmetrical about zero. These two cases are:

$$(1) \quad 0 \leq x < M \quad (2.10)$$

$$(11) \quad -[M/2] \leq x < [M/2] \quad (2.11)$$

The second equation is suitable for signed number systems such as two's complement.

The previously defined three modulus system ($M=3(5)(8)=120$) can uniquely represent integers from -60 to $+59$. For example, the residue value of $+23$ and -23 is given by:

$$23 = \{2,3,7\}$$

$$-23 = \{ |-23+120|_3, |-23+120|_5, |-23+120|_8 \} = \{1,2,1\}$$

The simplicity of multiplication and addition with residue numbers is in sharp contrast with the comparative difficulty of other operations such as scaling, sign detection and magnitude comparison. These functions usually require a partial or total conversion to

integer representation, a relatively difficult procedure when more than two moduli are involved.

Dividing two integers often gives a non-integer result. Since the residue number system does not represent fractions, division can be impossible without resorting to a conversion process. Scaling in a single modulus is calculated using the 'multiplicative inverse' of the denominator:

$$\text{Let } |(a)(b)|_m = 1 \quad (2.12)$$

Then, a is the multiplicative inverse of b , such that:

$$a = |1/b|_m \quad (2.13)$$

The multiplicative inverse of b will only exist if the highest common factor of b and m is equal to one. As an example, the multiplicative inverse of four exists in two of the three moduli used in the previous example:

$$\text{Let } |(a)(b)|_m = |4a|_m = 1$$

$$\text{mod } 3: |4a|_3 = 1$$

Since three and four are relatively prime, the inverse exists and by trial and error one finds that-

$$|1/4|_3 = 1$$

$$\text{Similarly, } |4a|_5 = 1, \quad |1/4|_5 = 4$$

Four and eight are not relatively prime ($8 = 2(4)$), and therefore the multiplicative inverse does not exist.

When integer division of two numbers results in a zero remainder, then this operation is possible without conversion. In this case:

$$\{ |a/b|_m^0, |a/b|_m^1, \dots, |a/b|_m^{N-1} \} = \{ |a(1//b)|_m^0, |a(1//b)|_m^1, \dots, |a(1//b)|_m^{N-1} \} \quad (2.14)$$

Of course, for the multiplicative inverse to exist, 'b' must be relatively prime to all moduli. It is unlikely that this equation will apply in general division, but it is used in mixed radix conversion. The identity given below is useful in simplifying the denominator:

$$|(1//a)(1//b)|_m = |1//(ab)|_m = |1//(|ab|_m)|_m \quad (2.15)$$

Appendix I contains a table of multiplicative inverses up to modulus 32.

2.2 BINARY/RESIDUE CONVERSION

It is a fairly simple operation to transform a binary number into a residue equivalent. Conversion in the other direction, from residue notation to binary is comparatively difficult, involving a complex structure of ROM tables and modulo addition.

A single ROM table is often sufficient for binary to residue conversion. When the length of a binary number is greater than about twelve bits, memory size becomes prohibitive in terms of both maximum propagation speed and number of memory cells. In this case,

it is easy to partition the integer into sets of bits - say the most significant and the least significant bit sets - followed by separate conversion and re-combination. This is illustrated in Figure 2.1, where a four bit residue value (for example, mod 15) is calculated from a sixteen bit binary number. The conversion itself is described by equation (2.16):

Let b_j be the j 'th bit of the binary number x

$$\text{Then } x = \sum_{j=0}^{15} b_j 2^j$$

$$\text{And } |x|_m = \left| \sum_{j=0}^7 b_j 2^j \right|_m + \left| \sum_{j=8}^{15} b_j 2^j \right|_m \quad (2.16)$$

2.2.1 RESIDUE/BINARY CONVERSION

The binary equivalent of a residue number is calculated using the Chinese remainder theorem (CRT) or mixed radix conversion (MRC). The CRT states that a unique solution exists in the range $a \leq x < a+M$, provided the moduli are relatively prime. The binary value is:

$$|x|_M = \left| \sum_{j=0}^{N-1} \bar{m}_j \left(|r_j|_{\bar{m}_j} \right) \right|_M \quad (2.17)$$

$$\text{Where } M = \prod_{i=0}^{N-1} m_i$$

$$\bar{m}_j = M/m_j = \prod_{\substack{i=0 \\ i \neq j}}^{N-1} m_i \quad (2.18)$$

The conversion procedure can be demonstrated with the following

FIGURE 2.1 - 16 bit binary to 4 bit residue conversion

example:

Given: $m_0 = 3, m_1 = 5, m_2 = 8 \dots M = 120$

Design a formula to convert a residue number to a normal integer.

Solution: $\bar{m}_0 = 120/3 = 40; \bar{m}_1 = 120/5 = 24; \bar{m}_2 = 120/8 = 15$

From the table of multiplicative inverses in Appendix I,

$$\left| \frac{1}{\bar{m}_0} \right|_m = \left| \frac{1}{40} \right|_{33} = \left| \frac{1}{1} \right|_3 = 1$$

$$\text{Similarly } \left| \frac{1}{\bar{m}_1} \right|_m = \left| \frac{1}{24} \right|_{55} = \left| \frac{1}{4} \right|_5 = 4$$

$$\left| \frac{1}{\bar{m}_2} \right|_m = \left| \frac{1}{15} \right|_{88} = \left| \frac{1}{7} \right|_8 = 7$$

$$\begin{aligned} \text{Hence: } |x|_M &= \left| \sum_{j=0}^{N-1} \bar{m}_j \left(\left| r_j / \bar{m}_j \right|_j \right) \right|_M \\ &= \left| 40 \left(\left| r_0 \right|_3 \right) (1) \right|_3 + 24 \left(\left| r_1 \right|_5 \right) (4) \right|_5 + 15 \left(\left| r_2 \right|_8 \right) (7) \right|_8 \Big|_{120} \end{aligned}$$

Thus if $x = \{0, 1, 1\}$,

$$\begin{aligned} \text{Then } |x|_M &= \left| 40 \left(\left| 0 \right|_3 \right) (1) \right|_3 + 24 \left(\left| 1 \right|_5 \right) (4) \right|_5 + 15 \left(\left| 1 \right|_8 \right) (7) \right|_8 \Big|_{120} \\ &= \left| 0 + 96 + 105 \right|_{120} = \left| 201 \right|_{120} = 81 \end{aligned}$$

This value is in the range $M/2 \leq x < M$, and could be expressed as a negative number:

$$x = |x|_M - M = -39$$

Another approach suggested by Soderstrand and Vernia trades

computational accuracy for simplicity of design [2.5]. The basic CRT equation is modified by dividing each term with the dynamic range M:

Let x' be the modified CRT result, where $x' = x/M - [x/M]$

(recall that "[]" are used to indicate integer division).

$$\begin{aligned} \text{Then } x' &= x/M - \left(\sum_{j=0}^{N-1} (\bar{m}_j / M) (|r_j // \bar{m}_j|) \right) \\ &= x/M - \left(\sum_{j=0}^{N-1} (1/\bar{m}_j) (|r_j // \bar{m}_j|) \right) \end{aligned} \quad (2.19)$$

The solution consists of the fractional portion of the x/M division result (and eliminates any integer components). Since the fraction is limited by a fixed number of bits, some roundoff or quantization error is introduced.

The range of the correlation output has now been modified from $0 \leq x < M-1$ to $0 \leq x' < 1$. In most cases, this transformation will be unimportant, since the relative magnitudes have remain unchanged. The zero to one range will be particularly useful when further manipulation of the filter output data is required.

Mixed radix conversion translates residue digits into a set of radix coefficients a_i . A radix number system is based on a set of radices R_i . In decimal number systems, all radices are equal to ten, while they are equal to two in binary. In mixed radix systems, the radices are not all the same, but are multi-valued, such as the bases used for residue conversion. In this case the radices are set equal

to the moduli, such that $R = m_i$.

An integer x can then be expressed as:

$$x = a_0 + a_1 R + a_2 R^2 + \dots + a_{N-1} \left(\prod_{i=0}^{N-2} R_i \right) \quad (2.20)$$

The a_i coefficients are unique for a given integer, and must lie within the range $0 \leq a_i < R_i$. Conversion of an integer to mixed radix form is demonstrated in the following example:

$$\text{Let } x = 43, \quad R_0 = 3, \quad R_1 = 5, \quad R_2 = 8$$

$$\text{Then } x = a_0 + a_1 R_0 + a_2 R_0 R_1 = a_0 + 3a_1 + 15a_2$$

$$\text{Hence } a_2 = 2, \quad a_1 = 4, \quad a_0 = 1 \quad \dots x = \langle a_2, a_1, a_0 \rangle = \langle 1, 4, 2 \rangle$$

$$\text{Here } \langle 1, 4, 2 \rangle = 1 + (3)(4) + (15)(2) = 43$$

Residue digits r_i are transformed into a_i coefficients sequentially, starting with a_0 . Each stage of the procedure calculates one more coefficient, while modifying the remaining residue digits for the next stage.

Starting with the initial stage, or stage 0, gives:

$$\text{Let } x(0) = a_0 + a_1 m_0 + a_2 m_0 m_1 + \dots + a_{N-1} \left(\prod_{i=0}^{N-2} m_i \right)$$

$$\text{Then } a_0 = x(0) - m_0 \lfloor x(0)/m_0 \rfloor = r_0 \quad \dots \text{by def'n of a residue}$$

$$\text{Let } x(1) = (x(0) - a_0)/m_0 = \lfloor x(0)/m_0 \rfloor$$

$$\text{Then } x(1) = a_1 + a_2 m_1 + a_3 m_1 m_2 + \dots + a_{N-1} \left(\prod_{i=1}^{N-2} m_i \right)$$

$$\text{And } a_1 = x(1) - m_1 [x(1)/m_1] = |x(1)|_{m_1}$$

This can be expanded to find the general case equations:

$$\text{Let } x(0) = a_0 + a_1 m_1 + a_2 m_1 m_2 + \dots + a_{N-1} \left(\prod_{i=0}^{N-2} m_i \right)$$

$$\text{Then } x(k+1) = (x(k) - a_k) / m_k \dots 0 \leq k \leq N-2 \quad (2.21)$$

$$\text{Where } a_k = x(k) - m_k [x(k)/m_k] = |x(k)|_{m_k} \dots 0 \leq k \leq N-1 \quad (2.22)$$

Residue to mixed radix conversion requires that all operations be performed using residue arithmetic. The term $(x(k) - a_k) / m_k$ will always have an integer result (i.e., evenly divisible), hence the residue division special case described by equation (2.14) can be applied ($|x(k)/m_k| = |x(1//k)|$, when x is evenly divisible by k).

$$\text{Let } |x(0)|_{m_1} = r_1 \quad (2.23)$$

$$\text{Then } |x(k+1)|_{m_1} = |(x(k) - a_k) / m_k|_{m_1}$$

$$\text{Or } |x(k+1)|_{m_1} = (|x(k)|_{m_1} - a_k) / m_k \quad (2.24)$$

$$\dots 0 < k \leq N-2, \dots k+1 \leq i \leq N-1$$

Mixed radix conversion, as described by equations (2.22) to (2.24) is illustrated in the following example:

$$\text{Given: } R_0 = m_0 = 3, R_1 = m_1 = 5, R_2 = m_2 = 8$$

Find the integer equivalent of the residue number $x = \{1, 3, 4\}$

Solution:

$$\text{Let } x(0) = a_0 + a_1 m + a_2 m^2 = a_0 + 3a_1 + 15a_2$$

$$|x(0)|_3 = r_0 = 1, \quad |x(0)|_5 = r_1 = 3, \quad |x(0)|_8 = r_2 = 4$$

$$\text{Then } a_0 = r_0 = 1$$

From equation (2.24),

$$|x(1)|_m = |(|x(0)|_m - a_0) / m| \quad \dots 1 \leq i \leq 2$$

$$|x(1)|_5 = |(|x(0)|_5 - a_0) / 3| = |(3-1)(2)|_5 = 4$$

$$|x(1)|_8 = |(|x(0)|_8 - a_0) / 3| = |(4-1)(3)|_8 = 1$$

$$\text{Then } x(1) = a_1 + 5a_2 = \{-, 4, 1\} \quad \dots a_1 = 4$$

$$|x(2)|_m = |(|x(1)|_m - a_1) / m| \quad \dots 2 \leq i \leq 2$$

$$|x(2)|_8 = |(|x(1)|_8 - a_1) / 5| = |(1-4)(5)|_8 = 1$$

$$\text{Then } x(2) = a_2 = \{-, -, 1\} \quad \dots a_2 = 1$$

$$\text{Finally } \langle a_0, a_1, a_2 \rangle = \langle 1, 4, 1 \rangle$$

$$x = a_0 + 3a_1 + 15a_2 = 1 + 3(4) + 15(1) = 28$$

$$\text{Check: } \{|28|_3, |28|_5, |28|_8\} = \{1, 3, 4\} \quad \text{as expected.}$$

Mixed radix conversion is a 'progressive' transformation, performed in a series of successive stages. In comparison, the

CRT requires a single transform stage, followed by addition of partial product terms.

While seemingly more complex than the Chinese remainder theorem, mixed radix conversion does not require the final mod M addition step of the CRT. It is also a simple procedure to add an extra modulus to an existing design, with the inclusion of another stage, and minor modification of the previous stages.

2.3 SIMPLIFIED RESIDUE MULTIPLICATION USING INDEX CALCULUS

A special property of residue arithmetic allows multiplication to be performed in a way similar to logarithmic addition. If a modulus p is prime, then mod p multiplication with zero valued elements deleted can be calculated using addition modulo $p-1$. Three steps are needed to perform this operation: transformation of modulo p integers into modulo $p-1$ indices, addition of these values, and final reversion to the mod p system.

One basis of this method is given by Fermats' theorem, which states:

Let 'p' be any prime number

'a' be an integer

$$\text{Then } \left| a^{p-1} \right|_p = 1 \quad (2.25)$$

The integer 'a' is called a generator, since it can be used to 'generate' integers in the range $1..p-1$ by raising it to some power less than $p-1$ (for example, $\left| 3^2 \right|_7 = 2$).

Multiplication of two numbers is now performed by conversion and modulo $p-1$ addition:

$$\text{Let } x = \left| a \right|_p^m, \quad y = \left| a \right|_p^n$$

$$\text{Then } xy = \left| a \right|_p^m \left(\left| a \right|_p^n \right) = \left| a \right|_p^{m+n} \quad (2.26)$$

Example: Multiply (6 by 5) mod 7, using 3 as a generator.

Solution: The generator look-up table is -

$$\text{Let } x = \left| 3 \right|_7^n$$

x	n
1	0
2	2
3	1
4	4
5	5
6	3

TABLE 2.1 - Index calculus values for mod 7, generator = 3.

$$\text{i.e., } 6 = \left| 3 \right|_7^3, \quad 5 = \left| 3 \right|_7^5$$

$$\text{And } \left| (6)(5) \right|_7 = \left| 3 \right|_7^3 \left| 3 \right|_7^5 = \left| 3 \right|_7^{3+5} = \left| 3 \right|_7^6 = \left| 3 \right|_7^2 = 2$$

One must pick the right number as a generator in order to express all integers in a given range. In the previous example, if two had been taken as the generator, it would be impossible to generate the

Thus if $x = |2^n|_7$

Then:

x	n
1	0
2	1
3	-
4	2
5	-
6	-

TABLE 2.2 - Index calculus values for mod 7, generator = 2

To fully utilize the modulus range (which is required in residue multiplication), the base must be a primitive root, i.e. one which generates all the possible residue values for a given modulus.

2.4 SUMMARY

Multiplication and addition in the residue number system is performed on a set of N different modulus bases. The operation of each modular arithmetic unit is independent of the other moduli, such that no carry bit propagation is required. Other functions, such as division (scaling) and overflow detection are more difficult, and can be performed more efficiently in the binary number system.

Binary to residue conversion is a relatively simple operation, and can be performed using a small number of ROM look-up tables. On the other hand, residue to binary conversion is comparatively complex. Algorithms for residue to binary conversion include mixed radix

conversion and the Chinese remainder theorem.

Fast multiplication similar to a logarithmic addition operation is possible using index calculus. However, these techniques are very modulus dependent, and may not be applicable to a general multiplication system.

3.0 METHODS FOR DIGITAL FILTERING

Discrete linear convolution of two sequences is denoted by the expression:

$$y(n) = c(n)*x(n) = \sum_{i=0}^{N-1} c(i)x(n-i) \quad (3.1)$$

A solution of (3.1) is obtained with either a direct summation of the product terms, or by converting each sequence to a transform domain (multiplication in a transform domain is equivalent to convolution in the original domain), and performing an inverse transform to find $y(n)$ (see section 3.2.1).

Several methods are available for computing a fast linear convolution of two discrete sequences. A set of design criteria will be presented in the following section to serve as a basis for determining the relative merits of each approach. It is then shown that use of a residue or modular number system allows greater adaptability with respect to coefficient word length and number of points, while maintaining a high filter throughput.

3.1 FILTER SPECIFICATIONS

In order to satisfy most filter specifications, the following criteria were defined:

- (1) HIGH SPEED - the filter should be able to process data in real time. To achieve this, the maximum clock frequency of the system must be as high as possible, with one convolution point calculated per clock cycle.

This implies a highly parallel structure, which may include an initial lag period of a number of clock cycles before the first convolution point becomes available.

- (ii) ADAPTABILITY - adaptable with respect to coefficient bit length and number of points. Smaller filter sections should be cascable to meet designer specifications, with a minimum of external hardware.
- (iii) ACCURATE - roundoff of partial product terms and final output should be kept to a minimum. Quantization will increase total noise and reduce the cascability of the system.
- (iv) LOW POWER CONSUMPTION - allow greater chip density with reduced cost.
- (v) SIMPLE INTEGRATION - structure of the design must be such as to allow easy layout with minimum chip area.

Selection of the best realization is determined by the circuit complexity necessary to meet these requirements.

3.2 METHODS OF FILTER REALIZATION

Multiplication has often been the most difficult function to complete in hardware. Operations such as addition and latching are possible with comparatively simple circuits.

Most systems will require a high degree of parallel operation or pipelining (dividing a function into two or more parts, which are then

performed in parallel) to increase throughput. In order to reduce much of the complexity in system diagrams, signal flow graphs similar to the system of Oppenheim and Schaffer [3.1] will be presented. A network of this type consists of nodes (summation, sources, sinks) and branches (multiplication, unit delays, special functions). Figure 3.1 contains the network symbols, including some new modular functions.

3.2.1 CONVOLUTION IN A TRANSFORM DOMAIN

Perhaps the most difficult decision to make in constructing a digital filter is to whether select a time domain (direct form) realization, or to convert to a transform domain and perform the convolution there. General transformation of an N-point sequence is given by the equation:

$$X(k) = \sum_{i=0}^{N-1} x(i)(q^{-ki}) \quad (3.2)$$

Multiplication in the transform domain is equivalent to 'circular' convolution in the time domain:

$$\text{If} \quad y(n) = x(n) * c(n)$$

$$\text{Then} \quad Y(k) = X(k)C(k) \quad (3.3)$$

Circular convolution is calculated by 'wrapping' the $x(n)$ sequence around a circle, so that the point $x(N-1)$ is followed by the first point $x(0)$ (in effect, forming a circle or ring instead of a 'linear' sequence of points). Commonly, an N-point linear convolution is derived using a $2N$ -point circular convolution, and placing an additional N zeroes on to the end of the original N-point

FIGURE 3.1 - Signal flow graph symbols

- (a) simple summation node
- (b) modulo summation (mod m) node
- (c) multiplication by 'a'; unit delay; modulo value (mod m)
- (d) typical operation

sequence.

Digital filtering involves the convolution of an 'infinite' input sequence with a fixed length coefficient function. Small convoluted blocks or sections (using the transformation described above) can be pieced together using overlap-add or overlap-save techniques (see Oppenheim and Schaffer [3.1], chapter 3).

The discrete Fourier transform (DFT) is commonly used for convolution or spectral analysis. One inherent problem of this function is finite accuracy, due to multiplication by quantized sine and cosine values. New transforms have been presented based on number theory and modular arithmetic [3.2, 3.3]. These functions (labelled number theoretic transforms or NTT's) are more exact, since they are performed in an integer number field, and use modular index calculus multiplication in place of trigonometric values (see section 2.3). They retain the property that multiplication in the transformed domain is equivalent to convolution in the original domain. However, the transform domain of the NTT does not have the useful physical interpretation that the Fourier transform has.

A reduction in the quantity of multiplications in a transformation is achieved using algorithms such as the fast Fourier transform or prime factor algorithm. The original N -point transform is subdivided into many smaller sections (two point transforms for the radix two FFT), which can be re-combined to calculate the complete function. In this fashion, the number of multiplications becomes approximately $N/2(\log N)$, much less than the original N .

Both the DFT and the NTT suffer from a lack of adaptability to design modifications (design changes usually involve either the word size - accuracy - or the number of taps). For the DFT, coefficient word length (and hence accuracy) is determined by the accuracy of sine and cos multipliers. Once the quantization has been fixed, expansion to a more accurate system would require an increase in the word length of each multiplier. Complexity of bit-slicing these multipliers is proportional to the square of the relative increase; i.e., a sixteen bit multiplication would require four eight bit multipliers in parallel, with separate addition of the partial products.

While this squared increase is avoided by employing a number theoretic transform, other difficulties arise that limit the suitability of this approach. These problems will be discussed later in the chapter.

An NTT is based on a set of finite or Galois fields (the finite fields used here are the numbers modulo a prime number or a prime number raised to some power; for example, five, seventeen and forty-nine are finite fields). Arithmetic operations are completed using residue arithmetic - there is a linear increase in complexity with increasing bit lengths, as compared to the squared increase of the DFT.

To illustrate this transform, let one of the finite fields have $m = p^n$ elements, p a prime number, n an integer. An N -point transform is given by the equation (N must evenly divide $m-1$):

$$\text{Let } m = p^n$$

$$\text{Then } |X(k)| = \sum_{m=0}^{N-1} |x(i)(q^k)| \quad (3.4)$$

The form of equation (3.4) is similar to that of the DFT, where q is the N 'th root of one:

$$X(k) = \sum_{i=0}^{N-1} x(i)(W_N^{ki}) \quad \dots W_N = \cos(2\pi/N) + j\sin(2\pi/N)$$

The ' q ' term of the NTT is a residue equivalent of the N 'th root of one, such that:

$$q^N = 1 \quad (3.5)$$

A practical design may be composed of a set of several different finite fields. Each field has a different q value, but they must be cyclic with a period equal to N , as described by the above equation.

In this case, the dynamic range of the system will be given by :

$$M = \prod_{i=0}^{L-1} p_i \quad (3.6)$$

Overflow is avoided by limiting the maximum output of the system to a value less than M :

$$[M/2] > \text{MAX}(\text{ABS}(y(n)) = \text{MAX}(\text{ABS}(x(n)))(\sum_{i=0}^{N-1} \text{ABS}(c(n))) \quad (3.7)$$

Equation (3.7) applies to all modular arithmetic FIR filters, including direct form realizations.

Since this is an residue number system, expansion to larger coefficient bit lengths is possible by increasing M . Additional moduli can be attached to the original system as needed to support

greater word lengths. Recombination at the last stage is accomplished using methods described in section 2.2.

The similarities and differences between the NTT and the DFT may best be illustrated by an example. In the case given below a (rather simple) three-point circular or periodic convolution is calculated in direct form and by converting to a transform domain. Of course, in this example the convolution length is too short to exploit the more efficient fast transform algorithms (FFT, prime number, etc).

Example:

Perform a circular or periodic convolution on the three-point sequences given below, using direct calculation of the convolution sum, a number theoretic transform calculation, and a discrete Fourier transform.

$$x(0) = -7; \quad x(1) = 3; \quad x(2) = 5; \quad \dots x(n+3k) = x(n), \quad k \text{ any integer}$$

$$c(0) = 6; \quad c(1) = -2; \quad c(2) = 1;$$

$$y(n) = c(n) * x(n)$$

Solution:

(i) Direct Calculation

The convolution equation is given by -

$$y(n) = \sum_{i=0}^{N-1} c(i)x(n-i) = \sum_{i=0}^2 c(i)x(n-i)$$

Expanding this equation gives -

$$\begin{aligned} y(0) &= c(0)x(0) + c(1)x(-1) + c(2)x(-2) \\ &= c(0)x(0) + c(1)x(2) + c(2)x(1) \\ &= (6)(-7) + (-2)(5) + (1)(3) = -49 \end{aligned}$$

$$\begin{aligned}
 y(1) &= c(0)x(1) + c(1)x(0) + c(2)x(2) \\
 &= (6)(3) + (-2)(-7) + (1)(5) = 37 \\
 y(2) &= c(0)x(2) + c(1)x(1) + c(2)x(0) \\
 &= (6)(5) + (-7)(3) + (1)(-7) = 17
 \end{aligned}$$

Thus, the solution to the convolution is:

$$y(0) = -49; \quad y(1) = 37; \quad y(2) = 17$$

(11) Number Theoretic Transform

In this example, a two modulus system will allow sufficient dynamic range for the solution. The moduli selected (both prime numbers) are $m_0 = 7$ ($q_0 = 2$) and $m_1 = 13$ ($q_1 = 3$).

$$\text{i.e.,} \quad \begin{matrix} 3 \\ |2|_7 = 1 \end{matrix} \quad \begin{matrix} 3 \\ |3|_{13} = 1 \end{matrix}$$

First, the coefficients must be converted to residue notation -

$$\begin{aligned}
 \{|x(0)|_7, |x(0)|_{13}\} &= \{0, 6\}; & \{|c(0)|_7, |c(0)|_{13}\} &= \{6, 6\} \\
 \{|x(1)|_7, |x(1)|_{13}\} &= \{3, 3\}; & \{|c(1)|_7, |c(1)|_{13}\} &= \{5, 11\} \\
 \{|x(2)|_7, |x(2)|_{13}\} &= \{5, 5\}; & \{|c(2)|_7, |c(2)|_{13}\} &= \{1, 1\}
 \end{aligned}$$

A number theoretic transform is given by the equation -

$$|X(k)|_m = \left| \sum_{i=0}^{N-1} x(i) q^{ik} \right|_m$$

The transformed values (as denoted by capital letters) are -

$$|X(0)|_m = \left| x(0)q^0 + x(1)q^0 + x(2)q^0 \right|_m = |x(0) + x(1) + x(2)|_m$$

$$|X(1)|_m = |x(0)q^0 + x(1)q^1 + x(2)q^2|_m = |x(0) + x(1)q + x(2)q^2|_m$$

$$|X(2)|_m = |x(0)q^0 + x(1)q^2 + x(2)q^4|_m = |x(0) + x(1)q^2 + x(2)q^4|_m$$

These equations can now be applied to the x and c coefficients -

$$\text{Thus } \{|X(k)|_7, |X(k)|_{13}\} = \left\{ \left| \sum_{i=0}^2 x(i)2^{ik} \right|_7, \left| \sum_{i=0}^2 x(i)3^{ik} \right|_{13} \right\}$$

$$\text{And } \{|X(0)|_7, |X(0)|_{13}\} = \{|0+3+5|_7, |6+3+5|_{13}\} = \{1, 1\}$$

$$\{|X(1)|_7, |X(1)|_{13}\} = \{|0+6+20|_7, |6+9+45|_{13}\} = \{5, 8\}$$

$$\{|X(2)|_7, |X(2)|_{13}\} = \{|0+12+10|_7, |6+27+15|_{13}\} = \{1, 9\}$$

In the same fashion,

$$\{|C(0)|_7, |C(0)|_{13}\} = \{5, 5\}$$

$$\{|C(1)|_7, |C(1)|_{13}\} = \{6, 9\}$$

$$\{|C(2)|_7, |C(2)|_{13}\} = \{0, 4\}$$

Convolution in the original domain is equivalent to multiplication in the transform domain. Therefore, the $Y(k)$ values are given by the equation -

$$Y(k) = C(k)X(k)$$

$$\{|Y(0)|_7, |Y(0)|_{13}\} = \{|C(0)X(0)|_7, |C(0)X(0)|_{13}\} = \{5, 5\}$$

$$\{|Y(1)|_7, |Y(1)|_{13}\} = \{|C(1)X(1)|_7, |C(1)X(1)|_{13}\} = \{2, 7\}$$

$$\{|Y(2)|_7, |Y(2)|_{13}\} = \{|C(2)X(2)|_7, |C(2)X(2)|_{13}\} = \{0, 10\}$$

The inverse transformation is given by the equation -

$$|y(n)|_m = |(-N') \left(\sum_{i=0}^{N-1} X(i) q^{-in} \right)|_{m_i} \quad \dots \text{where } N' = (m-1)/N$$

The N' coefficients in this example are -

$$N'_1 = (7-1)/3 = 2; \quad N'_2 = (13-1)/3 = 4;$$

Therefore, the $y(n)$ values can be calculated using the above inverse transform equation -

$$\{|y(0)|_7, |y(0)|_{13}\} = \{|-2(5+2+0)|_7, |-4(5+7+10)|_{13}\} = \{0, 3\}$$

$$\{|y(1)|_7, |y(1)|_{13}\} = \{|-2(5+8+0)|_7, |-4(5+63+30)|_{13}\} = \{2, 11\}$$

$$\{|y(2)|_7, |y(2)|_{13}\} = \{|-2(5+4+0)|_7, |-4(5+21+90)|_{13}\} = \{3, 4\}$$

The result can be checked by converting the correct solution (from the direct calculation) into residue equivalents -

Thus $-49 = \{0, 3\}$; $37 = \{2, 11\}$; $17 = \{3, 4\}$ as expected

(iii) Discrete Fourier Transform

The DFT is given by the equation -

$$X(k) = \sum_{i=0}^{N-1} x(i) (W_N^{ki}) \quad \dots W_N = \cos(2\pi/N) + j \sin(2\pi/N)$$

$$x(n) = (1/N) \left(\sum_{i=0}^{N-1} X(i) (W_N^{-ni}) \right)$$

For this calculation, the W_N sine and cosine values will be rounded off to seven bits - fairly accurate when compared to the magnitude of the $x(n)$ and $c(n)$ coefficients.

$$\text{Thus } W_N = \cos(2\pi/3) + j\sin(2\pi/3) = -.5 + j.859375$$

The calculation proceeds in the standard fashion to give the result (intermediate results were not rounded off) -

$$y(0) = -48.95; \quad y(1) = 36.824; \quad y(2) = 17.13$$

The quantization error of the DFT can be seen by comparing these answers to the exact solution. In comparison, the NTT gave the correct values, since no coefficient rounding was required.

The length of a DFT is not easily altered. For example, FFT algorithms are based on filter lengths equal to a power of two. Increasing the number of points will require a higher order root of one, and sequencing addresses will be different. This is impractical in a modular filter design, where each section must be cascadable. A NTT is even less flexible with respect to length modifications, as shown by equation (3.5). The generator or root q must be cyclic with a period equal to N . Changing N requires changing both the generator values and the modulus base. In effect, a totally new design is necessary. A limited amount of flexibility can be retained using residue number tables as described in [3.4], where the generator or modulus is altered in a new ROM.

An adaptable number theoretic transformer can be developed by converting a one dimensional convolution into two or more dimensions [3.5]. A two dimensional algorithm is used to calculate an N -point circular convolution, by performing the basic N -point transform $2N$ times. Computational efficiency in this case is reduced by a factor of two. Various transform lengths can be accommodated by controlling

the size of one dimension in the two dimensional equation. In this case, filter length is changed by completely replacing the second transform stage.

While convolution using a NTT is often better than using a DFT, neither approach offers the flexibility required for a cascadable filter design. A change in sequence length is only possible with new moduli and generator tables. Equally important, the control circuitry for a two dimensional transform (and block filter control) will be different for each specification, requiring in effect a programmable microsequencer. Unless the filter has a large number of points, the computational improvement will not be sufficient to compensate for the difficulty of implementing a system of this type. Of course, this argument is not valid in terms of mass produced devices with fixed coefficients, where adaptability is not required. In this case, the transform approach may offer the most efficient design.

3.2.2 DIRECT FORM CONVOLUTION

Convolution of two sequences (as illustrated in Figure 3.2(a)) is given by the expression:

$$y(n) = \sum_{i=0}^{N-1} c(i)x(n-i)$$

The simultaneous addition of N product terms is the most difficult function to implement digitally. Maximum throughput is

(a)

(b)

(c)

FIGURE 3.2 - (a) simultaneous summation FIR filter
 (b) eight tap pipelined simultaneous summation
 (c) progressive summation FIR filter

achieved with a tree structure and pipelining, as shown in Figure 3.2(b).

A better structure has been developed by applying the transposition theorem for digital networks [3.1]. This theorem states that when the direction of all branches in a single-input single-output network are reversed, the transfer function of the system will remain unchanged. The result of reversing the simultaneous summation circuit is shown in Figure 3.2(c).

Most fast direct form FIR filters are based on the progressive summation structure, since it is simple to realize digitally. Small filter sections are directly cascadable without external circuitry, provided that the dynamic range of the system is not exceeded.

One common filter realization is based on the simultaneous summation of product terms. This is the bit slice approach of Peled and Liu [3.6], where the convolution equation is written in single bit form:

Let $x_j(n)$ be the j 'th bit of $x(n)$ (where $x(n)$ is k bits long)

$$\text{Then } x(n) = \sum_{j=0}^{k-1} x_j(n) 2^j$$

And the convolution can be rearranged into the form:

$$y(n) = \sum_{i=0}^{N-1} c(i) x_j(n-i) 2^j = \sum_{j=0}^{k-1} 2^j \left(\sum_{i=0}^{N-1} c(i) x_j(n-i) \right) \quad (3.8)$$

An efficient realization of (3.8) is possible provided that the coefficients are fixed. The inner summation is stored in a 2^N by

log N bit memory, with an external shift and add circuit completing the outer summation. A general formulation for expansion in multiple dimensions (the bit-slice approach described above is a partial expansion) has been given by Burrus [3.7]. For adaptive filtering (i.e., variable coefficients), the coefficient value must also be brought into memory along with the data bits, requiring an excessive table size.

Another filter, using the progressive summation technique has a separate multiplier with accumulator for each point, such as the TRW eight tap (four bit coefficient) FIR filter chip [3.8]. Each tap is composed of a four bit multiplier with a thirteen bit binary adder (eight bit product, cascade up to thirty-two points without additional circuitry). Implementation of two filters (sixteen tap, four bit coefficients and eight tap, eight bit coefficients) is illustrated in Figure 3.3. The complexity of each circuit is linearly proportional to the number of points, but proportional to the square of the coefficient bit width. Thus, an eight bit coefficient filter requires four filter chips in parallel, with external addition circuitry. The multiplication in this case has been partially bit-sliced, such that:

$$\begin{aligned}
 \text{If} \quad y(n) &= \sum_{i=0}^7 c(i)x(n-i) \\
 \text{Then} \quad y(n) &= \sum_{i=0}^7 c_{0-3}(i)x_{0-3}(n-i) + \sum_{i=0}^7 c_{4-7}(i)x_{0-3}(n-i) \\
 &+ \sum_{i=0}^7 c_{0-3}(i)x_{4-7}(n-i) + \sum_{i=0}^7 c_{4-7}(i)x_{4-7}(n-i) \quad (3.9)
 \end{aligned}$$

(a)

(b)

FIGURE 3.3 - (a) sixteen tap, four bit coefficient FIR filter
(b) eight tap, eight bit coefficient FIR filter

FIGURE 3.4 - Stored square multiplier FIR filter

$$\text{Where } c(i) = \sum_{j=0}^7 c(j)2^j, \quad x(i) = \sum_{k=0}^7 x(k)2^k$$

Further expansion, up to sixteen bit coefficients would require sixteen chips with fifteen external pipelined adders (more efficient methods are possible [3.9], but the squared relationship is unavoidable). The maximum clock speed of this circuit is specified as twenty MHz. Higher clock frequencies are unlikely without an advance in technology. Larger order multipliers (such as eight by eight bit) are slower, and consume more power than is desirable when several units are to be integrated on a single chip. Therefore, while this approach has been developed, it does not meet the requirement of minimum complexity.

A second multiplication technique is based on a stored square approach [3.10, 3.11]:

$$\text{Since } (x+y)^2 = x^2 + 2xy + y^2$$

$$\text{And } (x-y)^2 = x^2 - 2xy + y^2$$

$$\text{Then } xy = ((x+y)^2 - (x-y)^2) / 4 \quad (3.10)$$

The squared terms are generated in a ROM look-up table, with an address size equal to the coefficient width, rather than the data width plus the coefficient width. Figure 3.4 describes a pipelined implementation of this filter. With M bit coefficients, each point requires two 2^M by $2M$ bit ROM's, two M -bit adders, a $2M$ bit adder and the final $2M+$ bit accumulator, along with the indicated pipelining circuitry. For example, an eight bit, eight tap filter would have 64K

bits in ROM alone. A practical realization would use smaller coefficient bit sizes, and cascade for wider specifications. This leads to the same squared relationship between bit size and chip quantity as in the parallel four bit multiplier.

Maximum clock speed of this design would be greater than the 20 MHZ of the previous example, assuming sufficient pipelining is included (i.e., four delays in the multiplier/accumulator, as compared to one delay). This approach is interesting, but it is still more complex than a residue number implementation, as described in the next section.

Other methods have been suggested for approximating a multiplication result. It is possible to substitute the addition of logarithms for multiplication [3.12, 3.13], although some quantization noise is induced due to conversion. A combination of ROM tables and approximation algorithms [3.14, 3.15] would give higher accuracy, but still not sufficient for greater word length cascading. Systems of this type would have to be fixed at a given input word size, losing flexibility in design.

3.3 RESIDUE NUMBER TRANSVERSAL FILTERING

The first system designed using residue arithmetic was proposed by Cheney [3.16]. In this paper, it was suggested that a modular architecture would simplify the operation of a digital correlator. Before the development of large scale memory arrays, arithmetic units were limited to magnetic core matrices or simple diode transistor

logic (DTL) [2.2]. The complexity of a binary multiplier constructed from gates will be proportional to the square of the number of bits in the word. In comparison, a residue number approach allows a linear increase with the word size, and a system with an equivalent number of gates will have higher switching speeds and greater dynamic range.

As fast ROM became available, new nonrecursive residue systems were designed for high speed filtering. In an early paper, Jenkins and Leon [3.17] used small conversion tables and index calculus for a three modulus system, similar to that shown in Figure 3.5. The difficulty of converting binary full adders to modulo addition made it necessary to multiplex these units over a number of calculations. Throughput is generally reduced by a factor equal to the number of multiplexed operations. In a more recent article, indexing was replaced by small moduli with tables for multiplication and addition [3.18]. Residue operations were still multiplexed, limiting the filter speed.

Soderstrand [3.19] proposed a parallel RNS filter, including pipelining to maximize the filter throughput. The cost of implementing this approach was shown to be much less than a similar circuit with binary arithmetic. A three modulus system based on the progressive summation structure is given in Figure 3.6. All multiplication, addition and conversion is completed in ROM, reducing the total chip count. This network also proved to be efficient for least mean square adaptive filtering [3.20].

FIGURE 3.5 - Residue number indexing FIR filter with multiplexed arithmetic units

FIGURE 3.6 - Progressive summation residue number FIR filter

3.3.1 RESIDUE MULTIPLICATION AND ADDITION

Several methods have been published for completing a residue multiplication. When dealing with large moduli, a quarter square method (as described in the previous section) can be used to reduce memory table size [3.21, 3.22]. A table look-up solution for large moduli addition has been described by Jullien [3.23]. The original modulus is replaced by two or more sub-moduli, with a dynamic range greater than the maximum result of the addition (usually twice the range of the original modulus). Sub-moduli addition is completed in small ROM tables. A final result is then constructed by applying the Chinese remainder theorem or mixed radix conversion.

An example of this method is given in Figure 3.7. Sub-moduli m_0 and m_1 are two four-bit integers, such that $(m_0)(m_1) > 2m$. The total memory of Figure 3.7(b) is approximately one-twentieth that of 3.7(a). By combining the sub-modular arithmetic tables with indexing conversion, the product of the two input values could be calculated with the same circuit (i.e., if modulus m is prime, the sub-moduli values would perform index addition mod $m-1$, with decoding and conversion completed in the final ROM).

Another method for multiplying in a very large modulus has been proposed based on commercial VLSI multipliers [3.24]. Partial product terms from each multiplier are combined in modulo converters to form the final result. Circuits of this type are limited to a specific moduli set ($2^n + 1$, 2^n , or $2^n - 1$), since it is difficult to modify the binary multiplier structure to accommodate modular arithmetic. They

(a)

(b)

FIGURE 3.7 - (a) mod m addition in single ROM (7-bit modulus)
 (b) mod m addition in two sub-moduli

are not suitable for a VLSI design where several multipliers must be integrated on a single chip.

Jenkins [3.25] has compared the efficiency of stored table residue multipliers with stored table two's complement multipliers. It was shown that when the result is not rounded (scaled), a residue circuit requires much less memory than an equivalent binary approach, and has a higher throughput rate without pipelining.

3.4 SUMMARY

An FIR filter designed as a highly parallel structure is bound to be an extremely complex device, regardless of the method of implementation. None of the techniques presented in this chapter can be defined as 'better' than the others in all situations; for example, a logarithmic approximation may be more than sufficient to meet accuracy requirements, allowing fast computation in a (relatively) simple design.

Yet a residue number filter can have some interesting features not available in other cases. A few of the more important advantages are listed here:

- (i) Exact multiplication can be performed more efficiently in the residue number system than is possible with other techniques.
- (ii) Complexity of a residue multiplier is linearly proportional to the word size, avoiding the squared increase of binary multipliers.

(iii) A high level of adaptability to design changes is achieved when operations are performed with stored memory tables. In this case, replacement of a ROM table will allow some minor design modifications.

(iv) VLSI implementation is simplified, due to the regular structure of memory.

(v) With bipolar technology, real time data processing is possible at clock speeds greater than 30 MHz, a major improvement over random logic multipliers.

Of course, a residue number filter will also suffer from what can be fairly major disadvantages, which may severely limit the suitability for some applications:

(i) Scaling a residue number (i.e., division by some integer) is a difficult operation and should be avoided if possible. Fortunately, this function will not be required in most transversal filtering applications.

(ii) Final residue to binary conversion can involve a significant amount of circuit overhead, which does not lend itself to VLSI implementation (due to the large number of bits that must be considered in each case).

However, it still seems probable that a residue number structure will prove to be more efficient than equivalent 'standard' (i.e., more commonly used) devices for digital FIR filtering.

4.0 RESIDUE FILTER DESIGN

Two designs are presented in this chapter for implementing a VLSI residue number transversal filter. The first system is based on moduli that can be uniquely represented by four binary digits. This results in a tradeoff between dynamic range and circuit complexity. In the second design, a larger range is provided with five bit moduli, at the expense of greater chip area.

Section 4.1 contains a discussion of techniques currently available for compression of stored function tables (such as multiplication and addition operations). A new approach is described that achieves equivalent reductions with relatively simple logic control. This algorithm permits the integration of a number of filter taps on a single chip.

A modular system was developed which uses a single one-modulus filter chip with variable ROM programming. The dynamic range of this device can be increased by operating several chips in parallel, with no loss in propagation speed. Section 4.2 examines the design of multiplier and accumulator circuits, with respect to limitations imposed by complexity and propagation delay. A block diagram is presented in the last part of the section, with a discussion of input/output organization suitable for compatibility of the two designs.

4.1 MEMORY REDUCTION IN STORED RESIDUE FUNCTION TABLES

A random logic circuit for residue multiplication will not have

the orderly structure of an ordinary binary multiplier. While the shorter bit length of a residue product is some compensation (the output length is equal to the input modulus bit length), gating will still be too complex for practical realization.

Adaptability is not a feature associated with residue multipliers. Using random logic, a given modulus design is unique, with gating and propagation delay dependent on the base. A better result is achieved with stored memory product tables [4.1]. The regular structure of this type of device allows greater adaptability and usually improves ~~switching~~ speed and power consumption. Only the ROM mask need be changed if the base modulus is changed.

The address width of a ROM table is twice the bit width of the modulus. Total memory size is given by equation (4.1), where n is the residue bit size:

$$\text{Memory capacity} = (2^{2n})(n) \text{ bits} \quad (4.1)$$

Table 4.1 lists the memory requirements for residue moduli of two to eight bits. The bit count increases exponentially with modulus size, until unrealistic values such as that for eight bit moduli are obtained.

MODULUS	ADDRESS WIDTH (BITS)	TOTAL BIT COUNT	COMMERCIAL MEMORY SIZE	PROPAGATION DELAY (APPROX)
2-4	4	32	32*8	15 nS
5-8	6	192	64*4	20 nS
9-16	8	1024	256*4	30-35 nS
17-32	10	5120	1024*8	35-40 nS
33-64	12	24576	4096*8	40-50 nS
65-128	14	1.1E5	16384*8	-
129-256	16	5.2E5	[65536*8]	-

([*] - Not commercially available as single memory)

TABLE 4.1 - Memory size for stored product tables.

The problem becomes more apparent when considering single chip FIR filters. Several multipliers must be integrated on a single chip, as well as circuitry for addition and pipeline control. Practical considerations therefore suggest that total memory be limited to about thirty-two thousand bits. A filter chip based on six bit moduli could then contain only one tap. A small improvement is obtained by resorting to five bit moduli. In this case, up to six taps can be accommodated on a single chip, without exceeding thirty thousand bits of memory.

A large number of taps is possible with a circuit based on four bit moduli, but the dynamic range of the system is limited:

Let $\{m_0, \dots, m_5\} = \{16, 13, 11, 9, 7, 5\}$ (for maximum range)

Then $M = \prod_{i=0}^5 m_i = 720,720 = 2^{19.46}$

Thirty-two tap filtering (with 32,768 bits of ROM) can result in

dynamic range overflow, unless the data and coefficients are limited to about seven bits. For example, to insure that no overflow occurs with eight bit coefficients and thirty-two taps, a dynamic range of 2^{8+8+5} , or 2^{21} bits is required. Note that when two's complement notation is employed, the necessary range is reduced by a factor of two, to twenty bits. The flexibility and accuracy of this system is thus constrained, and may not be suitable for some applications.

A table compression scheme would allow a greater number of taps on a single chip. By exploiting redundancy or symmetry in product tables, it is possible to replace much of the memory with simple logic functions to control addressing. This technique has been suggested by Szabo and Tanaka [2.2], and more recently by Huang and Taylor [4.2].

4.1.1 THE USE OF DIAGONAL SYMMETRY IN PRODUCT TABLE COMPRESSION

Figure 4.1 consists of a mod seven multiplication table. Symmetry exists along the diagonal (top left to bottom right) and along the anti-diagonal (excluding zeros, bottom left to top right). This 'reflection' is present because of properties inherent to modular arithmetic:

(i) diagonal symmetry

$$|xy|_m = |yx|_m \quad (4.2)$$

(ii) anti-diagonal symmetry

$$\text{Given } |(m-x)(m-y)|_m = |m^2 - my - mx + xy|_m$$

$|xy|_7$

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

FIGURE 4.1 - Mod 7 multiplication table, showing diagonal and anti-diagonal symmetry

$$\text{And} \quad |(m)(k)|_m = 0$$

$$\text{Then} \quad |(m-x)(m-y)|_m = |xy|_m \quad (4.3)$$

The four diagonal 'quadrants' of Figure 4.1 contain elements which are identical when reflected about the symmetry axes. With a new addressing procedure, it is possible to reduce the total memory (excluding zero elements) to one quadrant, as proposed in [4.2]. The protocol is given as the following algorithm (some minor corrections to the original paper have been included):

- (1) Define new variables x' and y' , and set control flags XFLAG and YFLAG according to the following rules:

If $x=0$ or $y=0$, then the result is $|xy|_m = 0$

Condition	New Variable	Flag
$x < [m/2]$	$x' = x$	XFLAG = 0
$x \geq [m/2]$	$x' = m-x$	XFLAG = 1
$y < [m/2]$	$y' = y$	YFLAG = 0
$y \geq [m/2]$	$y' = m-y$	YFLAG = 1

Each value of x and y is replaced by a new value between 0 and $[m/2]$ (x' and y'). Since magnitude has been reduced by a factor of two, one bit can be eliminated from the coding of each new variable. The flags XFLAG and YFLAG are set if the original value was modified. The ROM address width is now two bits smaller, reducing memory by close to a factor of four.

(iii) IF $x' = y'$ AND $XFLAG = YFLAG$, then the result is on the diagonal of the quadrant. These values are stored in ROM table D1, such that the result is $|xy| = D1(\text{address } x')$
m

(iii) IF $x' \neq y'$ OR $XFLAG \neq YFLAG$, then the result is somewhere in the remainder of the quadrant (including the anti-diagonal). The table given below is used to determine addressing of a second ROM table D2.

XFLAG	YFLAG	a	b
0	0	$\max(x', y')$	$\min(x', y')$
0	1	$\min(x', y')$	$\max(x', y')$
1	0	$\min(x', y')$	$\max(x', y')$
1	1	$\max(x', y')$	$\min(x', y')$

This step requires a comparator and simple logic control to determine position on the address bus. The symbol 'a' refers to the first half of the address bus, while 'b' refers to the last half. The result in this case is given by $|xy| = D2(\text{address } a, b)$
m

A circuit is given in Figure 4.2 for implementing this algorithm, with table values for mod seven multiplication.

The system described above is based on a large amount of random logic circuitry, such as comparators, data selectors and adders. Additional pipelining will be necessary to maintain a high throughput rate. Two separate memory tables are defined, as well as control for recognizing the special case of x or y equal to zero. With small moduli, it is more efficient to construct a complete product table

(a)

x'	
1	1
2	4
3	2

TABLE D1

	a		
	1	2	3
	1	6	2
	2	5	3
b	3	4	1
	3	4	1

TABLE D2

(b)

FIGURE 4.2 - (a) Circuit to exploit diagonal and anti-diagonal symmetry
(b) Product tables D1 and D2

than to implement this compression scheme. In the following section, a different symmetry is exploited to achieve an equivalent reduction.

4.1.2 AN ADDITIVE INVERSE METHOD FOR TABLE COMPRESSION

Another symmetry axis is present in stored product tables, as demonstrated in [2.2]. It is based on the residue property given by equation (4.4):

$$\left| \begin{matrix} xy \\ m \end{matrix} \right| = \left| \begin{matrix} m - (m-x)y \\ m \end{matrix} \right| = \left| \begin{matrix} m - (m-y)x \\ m \end{matrix} \right| \quad (4.4)$$

An algorithm for table reduction is listed below. As in the previous example, the original value is replaced by its additive inverse (i.e., $m-x$ for x), provided it is greater than $[m/2]$. Each operand is reduced by one bit, for an overall reduction in memory by a factor of four.

(i) Define new variables according to the table below:

CONDITION	NEW VARIABLE	FLAG
$x < [m/2]$	$x' = x$	XFLAG = 0
$x \geq [m/2]$	$x' = m-x$	XFLAG = 1
$y < [m/2]$	$y' = y$	YFLAG = 0
$y \geq [m/2]$	$y' = m-y$	YFLAG = 1

$$\text{FLAG} = \text{XFLAG} \oplus \text{YFLAG}$$

FLAG is set if either x or y (but not both) are modified.

(ii) $Z1 = \text{ROM}(\text{address } x', y')$

Each operand is always sent to the same address position. This

eliminates the data selector and comparator of the previous approach.

(iii) The final solution of $Z = |xy|$ is calculated from the ROM output $Z1$ according to the FLAG condition, as shown below:

2

CONDITION	OUTPUT VALUE
FLAG = 0	$Z = Z1$
FLAG = 1	$Z = m - Z1$

A circuit for performing this algorithm is depicted in Figure 4.3(a). The end result is achieved without much of the logic necessary to exploit diagonal symmetry, as in the first example (hence diagonal symmetry is still present in the reduced table of Figure 4.3(b)).

Propagation delay will be increased, causing throughput to decrease unless intermediate stages are pipelined. Control circuitry is necessary to determine an additive inverse, depending on a condition flag. Thus, the advantage obtained by reducing memory is partially offset by the increased amount of random logic.

4.1.3 A NEW COMPRESSION SCHEME USING SYMMETRIC RESIDUE CODING

It is possible to eliminate some of the complexity associated with normal residue coding by resorting to a modified coding scheme. The symmetric residue number system consists of digits ranging in

(a)

x

	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	2

y

ROM TABLE

(b)

FIGURE 4.3 - (a) Compression scheme (Szabo and Tanaka)
(b) Reduced table

value from $-[m/2]$ to $[m/2]$, as compared to the range $0 < x < m$ of normal residue notation. Binary to symmetric residue conversion differs slightly from the binary to residue conversion previously stated:

Let $\frac{x}{m}$ = symmetric residue of $x \bmod m$

$RND(x/m)$ = x/m rounded to the nearest integer

Then $\frac{x}{m} = x - (m)(RND(x/m))$ (4.5)

A table of symmetric residues is listed below:

x	$\frac{x}{3}$	$\frac{x}{5}$	$\frac{x}{7}$	$\frac{x}{8}$	x	$\frac{x}{3}$	$\frac{x}{5}$	$\frac{x}{7}$	$\frac{x}{8}$
-3	0	2	-3	-3	4	1	-1	-3	+/-4
-2	1	-2	-2	-2	5	-1	0	-2	-3
-1	-1	-1	-1	-1	6	0	1	-1	-2
0	0	0	0	0	7	1	2	0	-1
1	1	1	1	1	8	-1	-2	1	0
2	-1	2	2	2	9	0	-1	2	1
3	0	-2	3	3	10	1	0	3	2

TABLE 4.2 - Binary/symmetric residue equivalents

The additive inverse of a symmetric residue is obtained by complementing the sign; i.e., the additive inverse of 3 mod 7 is -3, and vice-versa. Properties presented in chapter two for normal residue numbers apply equally to symmetric residues.

A symmetric product table for mod seven multiplication is depicted in Figure 4.4. If one excludes the zero elements and neglects the sign, symmetry occurs along vertical and horizontal axes, as denoted by dotted lines. A simple design is possible with

		$/y/_7$						
		0	1	2	3	-3	-2	-1
$/x/_7$	0	0	0	0	0	0	0	0
	1	0	1	2	3	-3	-2	-1
	2	0	2	-3	-1	1	3	-2
	3	0	3	-1	2	-2	1	-3
	-3	0	-3	1	-2	2	-1	3
	-2	0	-2	3	1	-1	-3	2
	-1	0	-1	-2	-3	3	2	1

		0	1	2	3
	0	0	0	0	0
	1	0	1	2	3
	2	0	2	-3	-1
	3	0	3	-1	2

(a)
(b)

FIGURE 4.4 - Symmetric residue multiplication table mod 7

(a) Complete table
(b) Reduced version

FIGURE 4.5 - Memory Compression with symmetric residues

sign/magnitude coding of the symmetric residues:

Let $SIGN(x) = \text{sign bit of } x_m$

$MAG(x) = \text{magnitude bits of } x_m$

$Z = ((x_m)(y_m))_m$

And $Z1 = ROM(\text{address } MAG(x), MAG(y)) \quad (4.6)$

Then $SIGN(Z) = SIGN(x) \oplus SIGN(y) \oplus SIGN(Z1) \quad (4.7)$

$MAG(Z) = MAG(Z1) \quad (4.8)$

Equations (4.6) and (4.8) show that it is not necessary to bring the initial sign bits into memory to calculate the final magnitude. Thus, address width is decreased by two bits, reducing memory size by a factor of four. This simplification has not been used in previous table multiplication designs. However, it was discovered at a later time that the use of symmetric residues to simplify random logic multipliers has been suggested in an early internal Westinghouse report [4.3].

Equation (4.7) is written in a form suitable for hardware implementation. A circuit of this type is shown in Figure 4.5. Propagation time is increased by one gate delay (exclusive-or) over the original complete table circuit, but reducing the memory size decreases the addressing set-up time. Additional pipelining is not required, so that there is a minimal increase in random logic.

The bit coding scheme for mod six and seven is given in Table

4.3. With sign/magnitude notation, two states are defined as equivalent and equal to zero (000 and 100). In addition, when the moduli is even and not a power of two, there are two equivalent states defined for $[m/2]$ (see ± 3 in table below). Since not all states are unique, it is impossible to denote moduli equal to a power of two in a reduced form. In this case, it is necessary to include an extra magnitude bit, eliminating the memory compression advantage (for example, in mod eight, $4 = 100$. Since this is equivalent to zero, the system cannot be correctly applied).

x	/x/ 7	/x/ 6	x	/x/ 7	/x/ 6
-3	1 11	1 11/0 11	1	0 01	0 01
-2	1 10	1 10	2	0 10	0 10
-1	1 01	1 01	3	0 11	0 11/1 11
0	0 00/1 00	0 00/1 00	4	1 11	1 10

('/' indicates two equally valid states)

TABLE 4.3 - Symmetric residue sign/magnitude bit coding

4.1.4 TABLE COMPRESSION FOR MODULI EQUAL TO 2^n

Another compression technique was developed for the special case of moduli equal to a power of two:

Let $m = 2^n$

And
$$x = \sum_{i=0}^{n-1} x_i 2^i = \sum_{i=0}^{n-2} x_i 2^i + x_{n-1} 2^{n-1}$$

$$y = \sum_{i=0}^{n-2} y_i 2^i + y_{n-1} 2^{n-1}$$

$$\text{Then } Z_1 = \left| \left(\sum_{i=0}^{n-2} x_i 2^i \right) \left(\sum_{i=0}^{n-2} y_i 2^i \right) \right|_m \quad (4.9)$$

$$\text{And } |xy|_m = \sum_{i=0}^{n-2} Z_1 2^i + 2^{n-1} \left(\left(x_{n-1} \oplus y_{n-1} \right) \oplus Z_1 \right) \quad (4.10)$$

Equation (4.9) can be performed in a 2^{2n-2} by n bit ROM, one-quarter the size of a complete table. A logic diagram is depicted in Figure 4.6, along with a mod eight reduced product table.

The bit-slice approach described here will not have a greater propagation delay than the previous circuit, although two extra gates are required. Normal full adders can be used for accumulation, with a binary coded output (as compared to sign/magnitude notation).

4.1.5 MEMORY REDUCTION IN RESIDUE ADDITION TABLES

Techniques for compressing residue addition tables are not as efficient as those given for multiplication. With symmetric residues, a reduction by a factor of two can be achieved with little external circuitry, but further reduction is difficult.

A normal residue mod seven addition table is depicted in Figure 4.7(a). Diagonal symmetry is obvious, but anti-diagonal symmetry is less apparent. The symmetric residue equivalent in Figure 4.7(b) demonstrates this property:

$$\text{Since } -x/m = -x/m$$

(a)

	$x_0 \dots x_{N-2}$			
	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	1

$y_0 \dots y_{N-2}$

(b)

FIGURE 4.6 - Memory compression for modulus a power of two
 (a) Circuit diagram
 (b) ROM table

$|x+y|_7$ $|x|_7$

	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

$|y|_7$

(a)

$|x+y|_7$ $|x|_7$

	0	1	2	3	-3	-2	-1	0
0	0	1	2	3	-3	-2	-1	0
1	1	2	3	-3	-2	-1	0	1
2	2	3	-3	-2	-1	0	1	2
3	3	-3	-2	-1	0	1	2	3
-3	-3	-2	-1	0	1	2	3	-3
-2	-2	-1	0	1	2	3	-3	-2
-1	-1	0	1	2	3	-3	-2	-1
0	0	1	2	3	-3	-2	-1	0

$|y|_7$

(b)

FIGURE 4.7 - Mod 7 addition tables

(a) Normal residue

(b) Symmetric residue

$$\text{Then } \frac{x+y}{m} = -\left(\frac{-(x+y)}{m}\right) \quad (4.11)$$

This is not clearly shown in the first residue table, where additive inverses are used instead of negatives (an additive inverse x' of x obeys $|x'+x| = 0$, i.e. $x' = m-x$). Using additive inverses, the symmetry of 3.7(a) can be described as:

$$\frac{x+y}{m} = \frac{m - ((m-x) + (m-y))}{m}$$

Published techniques for exploiting diagonal symmetry are extremely complex, and are not suitable for VLSI design [4.2]. A new approach is suggested here, based on symmetry about the origin (defined where diagonal and anti-diagonal axes cross):

$$\frac{(-x)+y}{m} = -\frac{x+(-y)}{m} \quad (4.12)$$

$$\frac{(m-x)+y}{m} = \frac{m - (x + (m-y))}{m}$$

Equation (4.11) combined with (4.12) allows table reduction by a factor of two; i.e., since values are reflected about the origin, it is sufficient to store half of the addition table. Any contiguous half section of the original table may be selected, including top, bottom, left, right or diagonal (although designing address circuitry for this last option is more difficult). Random logic connections depend upon the compressed table chosen. The following algorithm is given assuming that the left hand plane is preserved (i.e., $-[m/2] \leq x < [x/2]$, $0 \leq y < [m/2]$), with elements in the right hand plane calculated by reflection.

Let $\text{SIGN}(x) = \text{sign bit of } x$

MAG(x) = magnitude bits of x

$$Z = \left\lfloor \frac{x + y}{m} \right\rfloor$$

$$(i) \quad \text{FLAG} = \text{SIGN}(x) \oplus \text{SIGN}(y)$$

$$(ii) \quad Z1 = \text{ROM}(\text{address FLAG}, \text{MAG}(x), \text{MAG}(y))$$

$$(iii) \quad \text{MAG}(Z) = \text{MAG}(Z1)$$

$$\text{SIGN}(Z) = \text{SIGN}(Y) \oplus \text{SIGN}(Z1)$$

A circuit that will perform this algorithm is shown in Figure 4.8, with the reduced mod seven table.

4.1.6 DESIGNING A SYMMETRIC RESIDUE SYSTEM

Since the operations of multiplication and addition can be performed efficiently in the symmetric residue number system, it is possible to eliminate normal residue coding in a given design. All operations required in a standard device can be implemented in a symmetrical system. Normal residue coding can either be converted to symmetric residue notation, or (preferably) avoided altogether. Final conversion can also be designed with compressed function tables. The Chinese remainder theorem is based on ROM tables that perform the function given below (see equation (2.17)).

The CRT is given by the equation:

$$|x|_M = \left| \sum_{i=0}^{N-1} \left(\overline{m_i} \right) \left(\frac{1}{r_i} \left(\frac{1}{m_i} \right) \right) \right|_M \quad \dots - \left\lfloor \frac{m}{2} \right\rfloor \leq r < \left\lfloor \frac{m}{2} \right\rfloor$$

(a)

MAG(x)

		EQFLAG = 0				EQFLAG = 1					
		0	1	2	3	3	2	1	0		
MAG(y)	0	0	1	2	3	-3	-2	-1	0	00	
	1	1	2	3	-3	-2	-1	0	1	01	
	2	2	3	-3	-2	-1	0	1	2	10	
	3	3	-3	-2	-1	0	1	2	3	11	
		000	001	010	011	111	110	101	100		

(b)

FIGURE 4.8 - (a) Compression circuit for modulo addition
 (b) Symmetric residue mod 7 reduced table

Where $\bar{m}_1 = M/m_1$

Define a table function for the CRT as follows:

$$T[r_j] = (\bar{m}_j) (/ (r_j) (1/\bar{m}_j) /) \dots - [m/2] \leq r_j < [m/2]$$

Now define a new table function for symmetric residue systems:

$$\text{Let } T_R(r) = T(r) \quad \dots 0 \leq r < [m/2] \quad (4.13)$$

$$\text{Since } /(-r_j) (1/\bar{m}_j) / = - / (r_j) (1/\bar{m}_j) /$$

$$\begin{aligned} \text{Then } \text{SIGN}(T(r_j)) &= (\text{SIGN}(r_j) \oplus \text{SIGN}(T_R(r_j))) \\ \text{MAG}(T(r_j)) &= \text{MAG}(T_R(r_j)) \end{aligned} \quad (4.14)$$

The new table will convert in the same fashion as the table defined for normal residues, while only using half of the original memory size. In the same fashion, the ROM needed in mixed radix conversion can be reduced. Two table functions that are used in this case are:

- (i) Residue to mixed-radix values (see equations (2.21) to (2.24)):

$$r / x / \bar{m}_1^{(k)} = / (/ x / \bar{m}_1^{(k-1)} - r_{k-1}^{(k-1)}) (1/\bar{m}_1^{(k-1)}) /$$

- (ii) Mixed-radix to binary value (see equation (2.20)):

$$x = a_0 + a_1 m + \dots + a_{N-1} \left(\prod_{i=0}^{N-2} m_i \right)$$

Reduction in the first equation is achieved with the addition compression algorithm. The radix coefficients ($a_i = r_i^{(1)}$) are in symmetric form ($-[m/2] \leq a_i < [m/2]$), which will simplify the final output. Normal mixed radix conversion results in a solution in the range $0 \leq x < M$, requiring further operations to convert to signed integer notation. With a symmetric mixed radix process, this final step is unnecessary as the result will already be in the range $-[M/2] \leq x < [M/2]$.

4.2 VLSI DESIGN

In this section, a modulus range is selected based on the techniques previously discussed. It is shown that with four bit moduli, a large number of taps can be contained on a single chip, although maximum dynamic range is limited. In comparison, five bit moduli are suitable for extremely accurate filtering, but cannot be packed as densely.

Progressive summation is efficiently performed in ROM tables, provided the modulus base is small (four bits or less). In the five bit modulus design, it is proposed that a random logic binary addition circuit will reduce chip area and increase propagation speed (as compared to a table approach). A conversion stage must be included at the filter output, to convert binary values to residue equivalents (a single 512 by 5 bit memory, or a combination of memory and random

logic). The cascading sum output is delayed by one or two clock cycles, with the result that an extra internal delay is added in the data cascading path. In either case, no external circuitry is needed for connecting filter sections.

4.2.1 MODULUS SELECTION

Large moduli allow greater dynamic range with fewer conversion steps than equivalent small moduli circuits. On the other hand, memory size increases exponentially with bit size, such that small moduli multipliers are easier to implement digitally. Table 4.4 lists the maximum dynamic range for various modulus bit lengths. It is assumed that total stored product table space was less than 32,768 bits, with the compression scheme of Section 4.1.3.

MODULUS BIT LENGTH	DYNAMIC RANGE (BITS)	NUMBER OF MODULI	MEMORY PER TAP (BITS)	MAXIMUM NUMBER OF TAPS
3	9.7	4	48	682.7
4	19.4	6	256	128.0
5	47.0	11	1280	25.6
6	89.9	18	6144	5.3
7	183.3	31	28672	1.1

(based on a stored product table space of 32,768 bits)

TABLE 4.4 - Modulus bit length and dynamic range

From the above table, it is clear that three bit moduli are impractical. The number of taps stated for the four bit example is optimistic, since the accumulation circuitry will be proportionally

more complex. Filter lengths on the order of sixteen or thirty-two taps are more practical. This is still very promising; a single filter (i.e., six modulus chips and conversion) can perform thirty-two tap filtering of seven bit data and coefficients. Larger dynamic ranges can be accommodated by several parallel sections (as in the TRW FIR filter chip).

The dynamic range of five bit moduli should be sufficient for most systems. With twenty bit coefficients, filter lengths of over one hundred taps can be cascaded together without the danger of a range overflow.

Memory reduction circuitry is identical for all five bit moduli less than thirty-two. Thus, a single filter chip need be designed for these values, with a final ROM masking level determining the base. It is preferable to perform operations mod sixteen on this chip, rather than designing a new circuit for mod thirty-two. The maximum dynamic range is reduced to forty-six bits, but this is still large enough for most applications.

Table 4.5 consists of four and five bit modulus sets with increasing dynamic range. Mod 2^{n-1} is listed first for simplified conversion procedures, but any order and selection of moduli is possible.

NUMBER OF MODULI USED	MODULUS VALUE (4 BIT)	DYNAMIC RANGE (BITS)	MODULUS VALUE (5 BIT)	DYNAMIC RANGE (BITS)
1	8	3.0	16	4.0
2	13	6.7	31	8.9
3	11	10.2	29	13.8
4	9	13.3	27	18.6
5	7	16.1	25	23.2
6	5	18.4	23	27.7
7	-	-	19	32.0
8	-	-	17	36.1
9	-	-	13	39.8
10	-	-	11	43.2
11	-	-	7	46.0

TABLE 4.5 - Four and five bit moduli dynamic range

With the memory constraint given above, a six bit residue filter chip is not practical. Eight tap filtering will require 49,152 bits of memory, plus accumulation, pipelining and addressing circuitry. As VLSI techniques improve (and as the demand for fast extremely accurate filtering increases), a filter of this type may be developed.

4.2.2 ACCUMULATION CIRCUITRY

A method was presented in section 4.1.5 for calculating the sum of two residue digits in ROM. Four bit moduli will require a 128 by 4 bit memory with appropriate address decoding, as depicted in Figure 4.9.

Each adder consists of a 512 bit ROM matrix (i.e., 512 transistors). Decoding can be performed by three or four input NOR gates, as shown in Figure 4.10. A simple circuit for a CMOS

FIGURE 4.9 - 32 row by 16 column 4-bit modulo ROM adder

(a)

(b)

FIGURE 4.10 - (a) Row decode circuitry;
(b) Typical CMOS D-type flip-flop (see ref. [4.4])

master-slave flip-flop is also depicted [4.4], although the exact topology will vary depending upon the device technology. In some cases, propagation delay can be decreased by simultaneously creating each signal with an inverse.

This method is not as efficient for five-bit moduli. Addition of two digits will necessitate a 512 by 5 bit memory, twice as large as the table needed for multiplication, and five times larger than that used in four bit addition. A random logic circuit may perform this function faster and take less chip area.

Designing a summation circuit for each modulus is possible; however, the flexibility of the previous method is lost. Instead of one integrated circuit with variable ROM programming, each modulus will require a unique design.

4.2.3 FAST ACCUMULATION USING RANDOM LOGIC BINARY ADDITION

A better solution can be obtained with the addition of some 'extension' logic (see Figures 3.11 to 3.13). Each accumulator will perform the operation in binary, to as many bits as the dynamic range requires. A binary to residue converter is included as a last stage in the filter chip, with a ROM program determining the modulus. This approach is based on the residue property:

$$\left| \underset{m}{|x|} + \underset{m}{|y|} + \underset{m}{|z|} \right| \underset{m}{=} \left| \underset{m}{|x|} + \underset{m}{|y|} + \underset{m}{|z|} \right| \quad (4.15)$$

The dynamic range of the summation result increases in proportion to the base two logarithm of the number of taps added; i.e., adding two taps will generate one extra bit, four taps will generate two

extra bits, while adding sixteen taps requires four extra bits. Expanding the accumulators in the circuit only as required will reduce the total number of gates. Similarly, full adder circuits are not necessary for these extra extension bits; circuits that propagate only a carry signal can be designed for higher speed with less chip area.

A block diagram of a sixteen point filter indicating the increasing accumulator length is shown in Figure 4.11. Each block contains a five bit adder/subtractor and sufficient carry propagation circuitry to avoid overflow. Propagation speed of this device can be a critical design factor. If addition time is greater than the memory addressing delay of the multiplier, the maximum filter speed is reduced unnecessarily. The use of look-ahead carry circuitry will allow extremely fast addition, at the expense of increased chip area.

In designing a circuit for addition/subtraction, two main options are available depending on the limits for propagation speed. A completely parallel structure can be designed with carry look-ahead for all five bits of the adder/subtractor, as shown in Figure 4.13 [4.5]. Each output is available after five gate delays, with a carry signal generated after four delays.

The carry signal is sent to an increment/decrement circuit containing the most significant bits (extension bits, see Figure 4.12). If the multiplication result is positive (i.e., after sign inversion, when $INV = 1$), then the sum is incremented for $CARRY = 1$. In the opposite case (negative multiplication result) a zero on the

FIGURE 4.11 - Five bit modulus accumulation structure

FIGURE 4.12 - Four bit increment/decrement one circuit

FIGURE 4.13 - five bit add/subtract circuit with look-ahead carry

carry signal will decrement one from the sum.

Depending on the device technology used in creating the chip, it may not be necessary to construct a completely parallel accumulator. Look-ahead carry circuitry may be sub-divided into two or more sections, reducing the total amount of random logic. For example, the first three bits and the next two bits can be designed with separate fast carry circuits. While propagation time through the accumulator is increased, the maximum filter throughput may remain unchanged, provided that the product table addressing delay is the critical factor. Determination of the best trade-off between chip area and maximum clock speed can be made with critical path analysis for a particular technology.

Figure 4.14 contains a single tap of a progressive summation FIR filter. The output of the product ROM has been changed from the sign/magnitude notation expressed in section 4.1.3 to a two's complement equivalent. This simplifies the accumulation circuit while eliminating a gate delay from the critical path. The sign bit is combined with the INV signal from the product result to control the increment/decrement function. INV also controls the add/subtract signal input of the accumulator. In the diagram, 'n' denotes the number of bits used in extending the dynamic range of the accumulator.

Two methods are suggested here for conversion of the nine bit summation result into residue notation. A simple circuit is illustrated in Figure 4.15(a), using a single 512*5 bit conversion ROM. In this case, an extra delay is inserted into the output stage,

FIGURE 4.14 - Five bit modulus single tap (product/accumulator) circuit

(a)

(b)

FIGURE 4.15 - Binary to residue conversion

(a) A single stage ROM conversion

(b) Two stage combination of ROM and binary addition

causing a short initial lag period before the first convolution point is produced (propagation speed is not reduced due to the additional pipelining circuitry).

A second approach is shown in Figure 4.15(b). The nine bit input is divided into two sections; the four least significant bits are added to the final multiplication result, while the five most significant bits are converted to residue notation in a 32×5 bit ROM.

The next stage calculates the sum of the conversion ROM output and the previous addition result. This value (labelled 'e' on the diagram) will be in the range $-2[m/2] \leq 'e' \leq 2[m/2]+15$, thus requiring a seven bit representation. Final conversion to an integer in the range $-15 \leq 'f' \leq +15$ (and hence requiring only a five bit representation) is performed by a controlled addition/subtraction of the modulus m. The value of m can be masked onto the inputs of five AND gates, such that when the value of 'e' is already in the correct range before conversion, it can be allowed to pass through the modulo converter unchanged.

Note that the final range is not in the usual form of $-[m/2] \leq y < [m/2]$. For example, a value of +15 (equal to 01111 in binary) may result in, perhaps, mod 23, where the maximum value should be limited to +11 (equal to 01011 in binary). This is, in effect an 'incomplete' conversion, where the correct answer is given by $15-23$, or -8. However, this incomplete conversion will have no effect on either cascading of further stages, or on final conversion in ROM.

e_6	e_5	e_4	e_3	e_2	e_1	e_0	VALUE OF 'e' ($-2 \lfloor m/2 \rfloor \leq e \leq 2 \lfloor m/2 \rfloor + 15$)	REQUIRED ACTION	MODULUS CONTROL	INV (ADD=1 SUB=0)
0	0	0	X	X	X	X	$0 \leq e \leq 15$	nothing	0	1
0	0	1	X	X	X	X	$16 \leq e \leq \lfloor m/2 \rfloor + 15$	sub. m	1	0
0	1	0	X	X	X	X	$\lfloor m/2 \rfloor + 16 \leq e \leq 2 \lfloor m/2 \rfloor + 15$	sub. m	1	0
0	1	1	X	X	X	X	impossible	---	X	X
1	0	0	X	X	X	X	impossible	---	X	X
1	0	1	X	X	X	X	impossible	---	X	X
1	1	0	X	X	X	X	$-2 \lfloor m/2 \rfloor \leq e \leq -17$	add m	1	1
1	1	1	X	X	X	X	$-16 \leq e \leq -1$	nothing	0	1

$$\text{MODULUS CONTROL} = e_5 + e_4$$

$$\text{INV} = e_6 + \overline{e_5} \overline{e_4}$$

(a)

(b)

FIGURE 4.16 - Modulo converter

(a) Converter Truth Table

4.2.4 BLOCK DIAGRAM OF A RESIDUE NUMBER FILTER CHIP

A four bit modulus filter chip is depicted in Figure 4.17, with labelled input and output signals. A five bit circuit would be slightly different from this, since it is necessary to include a conversion stage at the summation output. An extra delay would also be included in the data out path, to synchronize this output with the filter result. Filter sections can be directly cascaded to meet designer specifications without additional circuitry. Although not shown here, it is possible to include extra delays in the path of $x(n)$ in the four bit circuit, to make the timing compatible with the five bit example. If the four bit system is intended to stand alone, then this feature would not be included.

Four and five bit filters can be designed pin compatible (i.e., a given socket can be wired without prior knowledge of the filter type). In this case, the input/output connections are:

5	DATA IN
5	DATA OUT
5	SUM IN
5	SUM OUT
5	COEFFICIENT DATA
5	COEFFICIENT ADDRESS (assuming a 32 tap four-bit
1	COEFFICIENT ENABLE modulus filter chip)

31 PINS

A number of pins will also be required as power and clock inputs. For example, a bipolar TTL chip has two power supply pins (+5 volts and ground), and a single clock input (possibly two clock pins, if CP and $\overline{\text{CP}}$ are brought in together). If the circuit was packaged

FIGURE 4.17 - Four bit modulus filter block diagram

in a standard forty pin case, the extra pins could be used as additional enable inputs (for cascaded filter section coefficient storage) or programming of a PROM version.

4.3 CONCLUSIONS

Two residue filter designs have been presented that are suitable for fast digital filtering, using four bit and five bit moduli. Four bit moduli require less chip area, hence individual taps can be packed more densely. Dynamic range is limited in this case to 18.4 bits, but this is sufficient for many applications. Figure 4.18 depicts a six modulus, sixteen tap filter (assuming a sixteen tap filter chip is developed, as compared to a thirty-two tap version). Overflow is avoided by limiting coefficients to seven bits and data to eight bits.

Conversion to binary is performed here with the modified Chinese remainder theorem of equations (2.24) and (2.25). Accuracy is defined by the width of the conversion ROM, or in this case, twelve bits. A comparable system based on the TRW eight tap filter chip would be much more complex, involving sixteen filter chips with external addition and latching circuitry.

A five bit filter design would be similar to the four bit example, keeping in mind that the data signal is internally delayed to allow one clock cycle for final conversion. Cascading of four or five bit filters is demonstrated in Figure 4.19.

Computability is an important feature of these two filter designs. A four bit filter will be less complex and can be developed

FIGURE 4.18 - Four bit modulus, 32-tap adaptable FIR filter
(using a 32-tap filter chip)

FIGURE 4.19 - Cascading filter sections for greater number of taps

before the five bit circuit. When a second chip is produced, the dynamic range of the four bit system can be expanded to meet most requirements, by including some five bit moduli to the initial design.

22

OF / DE

5.0 TECHNOLOGY CONSIDERATIONS

The transversal filter design of the previous section was intended to be technology independent; i.e., the proposed techniques and structures are not specific to bipolar, CMOS or NMOS implementation. In this chapter, a brief (and extremely approximate) analysis will be made of these technologies, to obtain a rough estimate of chip size and propagation speed.

5.1 ESTIMATION OF CIRCUIT COMPLEXITY

The most accurate method of measuring chip area is to actually design the chip. Without resorting to this extreme, it is possible to obtain a rough estimate with a knowledge of the number and complexity of circuit elements.

Separate approximations will be made for ROM table space and random logic circuitry (binary adders, latches). It will be assumed that there is a reasonable correlation between the chip area required to integrate this circuit and packing densities of comparable commercial circuits designed using automatic or structured layout techniques.

In measuring the random logic circuit complexity, each element was first broken into more basic components. For example, binary addition is performed by a combination of relatively simple logic circuits, which are in turn composed of individual transistors.

LSI and VLSI data processing circuits were then used for comparison. These chips normally require a number of different

functions, with a significant portion of chip area allocated to data communication (bus lines). This problem is less severe in a filter chip consisting of cascaded binary latched adders. Hence, it is safe to assume that densities at least as great as those used in comparison are possible.

A slightly different approach was taken to estimate ROM chip area. In this case, published memory 'single cell' areas were examined, with an 'overhead' allowance ranging up to 60% of chip area for functions such as decoding and off-chip drivers. Note that in this design, multiplication and addition table outputs internal to the filter circuit are connected to a single data latch, eliminating the need for a driver stage. Thus, the approximations should be close to worst case estimates.

Table 5.1 contains a summary of the approximate number of elements in the four and five bit residue filter circuits.

FILTER	METHOD OF ACCUMULATION	ROM SIZE (bits)	LOGIC GATES	LOGIC INPUTS
4 BIT 32 TAP	ROM TABLE	8192 (PROD) 16384 (SUM)	3300 (4100)	6000
4 BIT 32 TAP	RANDOM LOGIC	8192 (PROD) 2048 (CONV)	6200 (7200)	12000
5 BIT 16 TAP	ROM TABLE	20480 (PROD) 40960 (SUM)	2300 (2800)	4100
5 BIT 16 TAP	RANDOM LOGIC	20480 (PROD) 2560 (CONV)	3400 (4000)	6700

TABLE 5.1 - Circuit element count

5.2 CMOS IMPLEMENTATION

Density of CMOS ROM circuits has increased rapidly in recent years, with some products containing up to 256K bits. Single cell areas range from 720 square μm^2 in 1977 [5.1], to something in the range of 60 square μm^2 . Memory elements often consume less than half of the total chip area. Other functions, such as decoding and output driving stages cannot be reduced proportionally, hence require large portions of space and power.

As density increases, greater 'utilization' ratios, or ratios of memory element area to total chip areas are possible. Most devices achieve ratios of twenty-five to fifty percent.

In the digital filter circuits of the previous section, three separate memory functions were defined. The product tables consist of a single large memory block, such that a fairly high utilization ratio

is possible. For area calculations expressed below, it will be assumed that this section has a ratio equal to that of the individual (and smaller) summation and conversion tables.

Random logic packing densities can vary by a large factor, ranging from greater than the 2800 square μm^2 /transistor of reference [5.2], to less than the 1290 square μm^2 of [5.3]. Both of the above examples are based on an average density of a variety of LSI and VLSI circuit elements, employing standard cell layout techniques.

Determination of the number of transistors in a filter circuit is based on the assumption of $2n$ transistors per n -input CMOS logic gate. Thus, the five bit filter circuit with random logic accumulation of Table 5.1 will consist of 13400 transistors.

Figures 5.1(a) and (b) contain a family of chip area curves for four and five bit filters. A memory utilization ratio of forty percent for product and summation tables has been assumed (changing the ratio will shift the curves in a fashion equivalent to changing the area per memory cell). Each graph contains two sets of parallel lines, each line denoting a specific chip area (50 and 100 square mm^2 ; 50 mm^2 = 77500 square mil's, 100 mm^2 = 155000 square mil's). The set of lines with a steeper slope are based on filters using stored function tables for both product and summation calculations. The second set of lines assumes that the sum is calculated with binary random logic adders.

The diagrams are suitable for approximating chip area and for determining the best design approach. For example, assume that the

FIGURE 5.1 - Chip area as a function of transistor and memory cell density.

layout and fabrication processes have the following characteristics:

Average memory cell area = 200 square μm^2 /cell

Average logic gate area = 3000 square μm^2 /transistor

From Figure 5.1, a four-bit modulus filter using product and summation tables can be constructed on a 60 square mm chip. The same chip designed with random logic accumulation will take approximately 100 square mm's.

On the other hand, a five-bit modulus filter chip has an area of about 65 square mm's with random logic or with ROM table summation; this is right on the border of where logic summation consumes less chip area than ROM summation. In both diagrams, the shaded region marks the area where logic accumulation will be more efficient.

It is very difficult to estimate propagation speed without experimental testing of a particular technology. Published results have indicated that it is possible to access CMOS ROM (with similar the same as the CMOS examples given above (while NMOS usually requires less chip area than CMOS, the nature of ROM memory using one transistor per cell negates some of the advantage of NMOS technology) Published examples range from 62 square μm^2 ([5.4], using electron beam data writing), to around 170 square μm^2 ([5.5], diffusion mask programmable). Access time is improved over comparable CMOS circuits, with values of 35 ns available in commercial devices [5.7].

Fewer transistors are required to construct a single logic gate. In calculating the total number of transistors to perform the random

logic functions, it was assumed that each n -input gate has $n+1$ transistors (as compared to $2n$ transistors in CMOS). Packing density can be slightly greater, since isolation wells or barriers are not required.

Figures 5.1(c) and (d) contain chip area curves for NMOS versions of the filter circuits. They are similar to the CMOS equivalents with a small increase in slope due to decreased logic complexity.

Using the same data as in the CMOS example (memory area = 200 square $\mu\text{m}^2/\text{cell}$, logic area = 3000 square $\mu\text{m}^2/\text{cell}$), smaller chip areas are obtained, as shown below:

Four bit/stored summation	50 square mm^2
Four bit/logic summation	80 square mm^2
Five bit/stored summation	60 square mm^2
Five bit/logic summation	55 square mm^2

The chip could probably be operated above ten MHz, with memory access time about 50 ns and latch set up time less than 50 ns. This is some improvement over the CMOS circuit, although power consumption is increased.

5.4 BIPOLAR IMPLEMENTATION

Area calculations will be much more approximate in this section, as it is more difficult to estimate logic gate densities (for example, TTL uses multiple emitter transistors for gate inputs). Circuits will be defined in terms of the number of logic gates, with an average area

assigned to each gate.

Bipolar ROM (utilizing Schottky diode memory cells) generally consumes more chip area than equivalent CMOS or NMOS circuits. In 1977, a bipolar 16K bit ROM was constructed with 420 square $\mu\text{m}^2/\text{cell}$ [5.7], while a 50K bit bipolar ROM was constructed in 1980 with only 165 square $\mu\text{m}^2/\text{cell}$ [5.8].

In 1978, an average gate area of 5800 square μm^2 was reported for a bipolar LSI processor [5.9]. More modern circuits will have reduced this quantity for implementation of VLSI functions.

Figures 5.1(e) and (f) consist of chip area graphs. In calculating these curves, no effort was made to allow for the greater power dissipation typical in bipolar circuits.

As an example of a bipolar filter, assume the following layout rules are observed:

memory cell area	300 square $\mu\text{m}^2/\text{cell}$
logic gate area	7500 square $\mu\text{m}^2/\text{gate}$

With these rules, the approximations given below were obtained:

Four bit/stored summation	50 square mm^2
Four bit/logic summation	70 square mm^2
Five bit/stored summation	65 square mm^2
Five bit/logic summation	45 square mm^2

The larger memory cell is a critical factor in the design of a five bit circuit. Area is increased by almost 20 square mm^2 , or

about 44 percent with stored summation tables. This may be countered by some advantages of bipolar memory, including lower power consumption and fast propagation speed.

A TTL filter circuit should easily achieve clock speeds of twenty MHz or more. When extremely fast filtering is required, ECL versions can be produced that have memory access times less than 10 ns, and latch delays less than 5 ns. Clock rates greater than 50 MHz are possible, provided there is sufficient heat dissipation.

5.5 CONCLUSIONS

Rough calculations were presented with graphical illustrations that demonstrate the trade-off between stored function table requirements and random logic circuit area. While these functions are over-simplistic, they allow a quick approximation of chip complexity suitable for determining the basic design approach (i.e., using stored summation tables or random logic binary addition circuits). Many other considerations (such as propagation speed, fabrication yield, and the difficulty of interfacing memory and logic on a single circuit) will have a significant influence on the final design approach.

6.0 CONCLUSIONS

Transversal filtering is one of the basic operations of digital signal processing. Typical applications may include pattern recognition, speech processing and radar data manipulation.

Real time performance is critical in many cases, both because results may only be meaningful if obtained immediately, and because data backlogs can rapidly fill any storage medium. Thus, highly parallel filter structures have evolved that perform many different calculations simultaneously.

A parallel structure, while being inherently faster, will consequently involve a large number of components. Digital systems that are already moderately complex (such as a single pipelined multiplier-accumulator unit) become perhaps unacceptably complex when a number of units must be connected together.

Systems that are based on normal binary arithmetic suffer from an additional limitation. The complexity of traditional multiplier structures is proportional to the square of the word length. As word lengths increase (i.e., greater number of bits to obtain more accurate results), this squared relationship can make the construction of even a single multiplier unit a difficult task.

In this thesis, a residue (or modular) based number system is applied to digital FIR filter design. A residue number system can separate operations on long word lengths into several smaller word operations, each of which can be performed independently. The number of smaller word arithmetic units is linearly proportional to the

original word length - thus, the squared increase of binary number systems is reduced to a linear increase. A second advantage of residue arithmetic units is the suitability of residue operations for implementation on stored function tables (i.e., ROM tables). Table look-up circuits will often be faster and consume less power than equivalent logic units.

Due to the wide variety of FIR filter applications, a flexible filter design is required that can be easily modified to meet a given set of specifications. While this may not be feasible with a binary system, a residue system can be designed in a completely modular fashion, allowing easy expansion for either greater word lengths or greater number of taps.

Two major disadvantages of residue number systems are the difficulty of scaling (division) operations, and the complexity of final residue to binary conversion. In a transversal filter design, the first disadvantage will not have a noticeable effect, since all operations consist of either addition or multiplication. The second disadvantage is more serious; however, the increase in hardware is roughly equivalent to the additional hardware needed for cascading binary number system filter chips in parallel, and hence is seen as not being a limiting factor.

While residue circuits can require significantly less hardware than equivalent binary filters (assuming a sufficiently large word length and number of taps), the elements of a typical design may still number in the hundreds on multiple circuit boards, consuming an

excessive amount of power. This is clearly unacceptable for many applications.

VLSI implementation has been largely impractical, due to the size of ROM look-up tables required for most residue operations. Fully expanded tables for five bit moduli take over 5,000 bits of memory each, such that a fully parallel sixteen tap modular filter chip needs over 160,000 bits of ROM memory, plus associated pipelining circuitry.

A new table compression scheme is described in this report that reduces the look-up table size by as much as a factor of four. Five bit moduli multiplication can now be performed in approximately 1,300 bits of memory, small enough to allow the integration of a number of parallel tables on a single VLSI chip.

The compression algorithm is based on exploiting symmetry present in tables of 'symmetric' residue functions, ('symmetric' residue notation is equivalent to a sign-magnitude method of expressing residue numbers). Addition and residue/binary conversion operations can also be performed in tables half the size of those previously required.

A novel architecture for a modular or flexible FIR filter design is described that is directly cascadable for greater word lengths or greater tap lengths. ROM programming is used to define the modulus base, such that a single VLSI design is sufficient for use with any modulus.

Two original methods of accumulating partial results were

described for four and five bit modulus circuits. The four bit modulus accumulation was performed in a compressed addition table consisting of about 500 bits per table. While larger modulus adders can be designed in this fashion, a random logic design was found to be more efficient.

This second approach used for five bit modulus addition combined residue products with normal binary addition, forming a non-modular summation. While seemingly more cumbersome, the speed and regular structure of binary adder circuits more than outweighed the added hardware cost of a final residue conversion of the non-residue summation. Two new conversion circuits were described suitable for a pipelined system design.

Chip area estimates are presented for a thirty-two tap, four bit modulus circuit and a sixteen tap, five bit modulus circuit. Using published packing density estimates of comparable digital signal processing chips, it was shown that the two designs will consume a large (but not unreasonable) amount of chip area (on the order of 100,000 square mils).

Although intended primarily as a VLSI design, the techniques discussed in this report are suitable for use in discrete component systems. Single chip memory sizes have limited designers in the past to six or seven bit moduli; the reduction algorithms described here can increase this limit by one bit without decreasing switching speed. Conversely, the modulus bit size can remain unchanged, allowing the use of smaller memory chips with higher switching

speeds. Number theoretic transforms, which are used to perform convolution in a fashion similar to the DFT can also be performed in a more efficient manner.

6.1 RECOMMENDATIONS

(i) The elements described as part of the residue filter chips are all standard, consisting of ROM cells, latches, binary adders and address decoding circuitry. Most VLSI computer-aided design systems will support standard cell layout techniques. The elements listed above should already be available as standard cells, such that a VLSI layout should be a fairly simple operation. Accurate chip area and power estimates obtained from this layout should be sufficient to determine the feasibility of the design as described. If the chip area is excessive, the number of taps per chip can be reduced until the design becomes manageable.

(ii) A single filter 'board' can be designed that has enough space for most filter systems. Sockets can be supplied that are pre-wired, such that construction of a filter is reduced to plugging in the correct ROM elements. Modularity is supported by allowing simple re-configuration of the original design, usually by adding more chips or by moving the existing chips to other sockets on the board. A general residue to binary conversion circuit should be a part of this board, easily expandable to any required bit width and number of moduli.

(iii) The compression algorithms and filter architectures are

equally applicable to systems designed with individual MSI and LSI components, although such a system will be much more complex. A modular design might segregate each modulus on a separate board, with a separate controller/residue to binary conversion board. A filter design will then consist of a number of modulus boards (note that larger modulus sizes could be supported on each board - for example, seven bit moduli are practical) that plug into a card cage as required. Each modulus board is identical, with sockets for ROM modulus tables. Thus a flexible design is still supported, such that the word length and number of taps can be varied in any desired fashion.

(iv) Currently, the difficulty of integrating residue to binary conversion circuits onto a single VLSI package lies in the number of pins required for parallel I/O, and not in the complexity of the required conversion circuit. New packaging techniques are being developed that allow as many as 144 pins or more per package. Thus, examination should be made of the possibility of integrating a conversion circuit onto such a package, reducing the complexity of a residue FIR filter circuit by a large factor.

(v) As VLSI techniques improve, the number of ROM memory cells that can be integrated onto a single chip will increase. When the technology reaches a sufficient level, a six or seven bit modulus circuit could be designed to expand the capabilities of current four and five bit modulus circuits.

REFERENCES

- [2.1] H.L. Garner, 'The residue number system', IRE Trans. Electron. Computers, Vol. EC-8, pp. 140-147, June, 1959.
- [2.2] N.S. Szabo and R.I. Tanaka, 'Residue Arithmetic and Its Applications to Computer Technology.' New York: McGraw-Hill Book Co., 1967.
- [2.3] D.E. Knuth, 'The Art of Computer Programming', Vol 2, Seminumerical Algorithms. Mass: Addison-Wesley Pub. Co. Inc., 1969, pp. 248-269.
- [2.4] J.H. McClellan and C.M. Rader, 'Number Theory in Digital Signal Processing.' New Jersey: Prentice-Hall, Inc., 1979, pp. 3-46.
- [2.5] M.A. Soderstrand and C. Vernia, 'An improved RNS digital-to-analog converter', Proc. IEEE Intern. Symp. Circuits Syst., Chicago, Illinois, pp.85-87, April 1981.
- [3.1] A.V. Oppenheim and R.W. Schaffer, 'Digital Signal Processing.' New Jersey: Prentice-Hall Inc., 1975.
- [3.2] J.M. Pollard, 'The fast Fourier transform in a finite field', Math. Comput., Vol 25, pp. 365-374, Apr. 1971.
- [3.3] R.C. Agarwal and C.S. Burrus, 'Fast convolution using Fermat number transforms with applications to digital filtering', IEEE Trans. Acoust, Speech, Signal Proc., Vol ASSP-22, pp. 87-97, Apr. 1974.
- [3.4] A. Baraniecka and G.A. Jullien, 'Hardware implementation of convolution using number theoretic transforms', Proceedings 1979 Int. Conf. Acoustics, Speech, Signal Proc., Wash. D.C., Apr. 1979, pp. 490-493.
- [3.5] R.C. Agarwal and C.S. Burrus, 'Fast one-dimensional digital convolution by multi-dimensional techniques', IEEE Trans. Acoustics, Speech, Signal Proc., Vol ASSP-22, pp. 1-10, Feb. 1974.
- [3.6] A. Peled and B. Liu, 'A new hardware realization of non-recursive digital filters', IEEE Trans. Acoustics, Speech, Signal Proc., Vol. ASSP-22, pp. 456-462, Dec. 1974.
- [3.7] C.S. Burrus, 'Digital filter structures described by

distributed arithmetic', IEEE Trans. Circ. Systems, Vol. CAS-24, pp. 674-680, Dec. 1977.

- [3.8] F.A. Williams, 'An expandable single-IC digital filter/correlator', IEEE Int. Conf. Acoustics, Speech, Signal Proc. 82, Paris France, pp. 1077-1080, May 1982.
- [3.9] W.J. Stenzel, W.J. Kubitz and G.H. Garcia, 'A compact high-speed parallel multiplication scheme', IEEE Trans. Comp., Vol C-26, pp. 948-957, Oct. 1977.
- [3.10] E.L. Johnson, 'A digital quarter square multiplier', IEEE Trans. Comp., Vol C-29, pp. 258-261, Mar. 1980.
- [3.11] T. Tjahjadi and W.J. Steenaart, 'On the accuracy of ROM stored square multipliers', IEEE Trans Circ. Systems, Vol CAS-29, pp. 441-447, July 1982.
- [3.12] N.G. Kingsbury and P.J.W. Rayner, 'Digital filtering using logarithmic arithmetic', Electronics Letters, Vol. 7, pp. 56-58, Jan. 28, 1971.
- [3.13] T.A. Brubaker and J.C. Becker, 'Multiplication using logarithms implemented with read-only memory', IEEE Trans. Comp., Vol C-24, pp. 761-765, Aug. 1975.
- [3.14] E.L. Hall, D.D. Lynch and S.J. Dwyer, 'Generation of products and quotients using approximate binary logarithms for digital filtering applications', IEEE Trans. Comp., Vol C-19, pp. 97-105, Feb. 1970.
- [3.15] D. Marino, 'New algorithms for the approximate evaluation in hardware of binary logarithms and elementary functions', IEEE Trans. Comp., Vol. C-21, pp. 1416-1421, Dec. 1972.
- [3.16] P.W. Cheney, 'A digital correlator based on the residue number system', IRE Trans. Electronic Comp., Vol. EC-10, pp. 63-70, Mar. 1961.
- [3.17] W.K. Jenkins and B.J. Leon, 'The use of residue coding in the design of hardware for non-recursive digital filters', Eighth Asil. Conf. Circ., Syst. and Comp., pp. 458-462, Pacific Grove, Calif., 1974.
- [3.18] W.K. Jenkins and B.J. Leon, 'The use of residue number systems in the design of finite impulse response digital filters', IEEE Trans. Circ. Syst., Vol. CAS-24, pp. 191-201, Apr. 1977.
- [3.19] M.A. Soderstrand, 'High-speed digital filters using residue

- number arithmetic', Ninth Annual Asil. Conf. Circ., Syst. Comp., pp. 416-420, Pacific Grove, Calif. 1975.
- [3.20] M.A. Soderstrand and J.K. Kelley, 'Design of a low-cost adaptive FIR filter with sampling rate in excess of 10 MHz', IEEE Int. Symp. Circ., Systems Proc., pp. 432-434, Chicago, Ill. 1981.
 - [3.21] F.J. Taylor, 'Large moduli multipliers', IEEE Int. Conf. Acoustics, Speech, Signal Proc. 80, pp. 792-795, Denver, Col. 1980.
 - [3.22] M.A. Soderstrand and C. Vernia, 'General modulo $P(i)$ multiplier with RNS arithmetic applications', 1980 IEEE Int. Symp. Circ., Systems Proc., pp. 375-378, Houston Tex. 1980.
 - [3.23] G.A. Jullien, 'Implementation of multiplication, modulo a prime number, with applications to number theoretic transforms', IEEE Trans. Comp., Vol. C-29, pp.899-905, Oct. 1980.
 - [3.24] F.J. Taylor, 'A VLSI arithmetic multiplier', IEEE Trans. Comp., Vol. C-31, pp. 540-546, June 1982.
 - [3.25] W.K. Jenkins, 'A comparison of residue number multipliers and 2's complement multipliers implemented by stored multiplication tables', 1978 IEEE Int. Symp. Circ. Systems Proc., pp. 297-301, New York, NY, 1978.
 - [4.1] G. A. Jullien, 'Residue number scaling and other operations using ROM arrays', IEEE Trans. Comp., Vol. C-27, pp.325-336, April 1978.
 - [4.2] C.H. Huang and F.J. Taylor, 'A memory compression scheme for modular arithmetic', IEEE Trans. Acoustics, Speech, Signal Proc., Vol. ASSP-27, pp. 608-611, Dec. 1979.
 - [4.3] Westinghouse Electric Corporation, Air Arm Division, 'Modular arithmetic computing techniques', ASD TDR 63-280, Baltimore, Md., May 1963.
 - [4.4] SimuTec, 'Interactive Logic Simulation Program, Users Manual', Version Two, Citrus Heights, Calif., 1982.
 - [5.1] R.G. Stewart, 'High density CMOS ROM arrays', ISSCC 77, pp. 20-21, Feb. 1977.
 - [5.2] R. Hoshikawa et al., 'A 10,000-gate CMOS LSI processor', ISSCC 80, pp. 106-107, Feb. 1980.

- [5.3] A.D. Lopez and H.S. Law, 'A dense gate matrix layout method for MOS VLSI', IEEE Jour. Solid-State Circ., Vol. SC-15, pp.736-740, Aug. 1980.
- [5.4] K. Kiuchi, N. Ieda, K. Takeya and T. Baba, 'A 65 mW 128K EB-ROM', ISSCC 79, pp. 12-13, Feb. 1979.
- [5.5] S. Kamuro, K. Sano, S. Kimura and Y. Aoki, '64K DSA ROM', IEEE Jour. Solid-State Circ., Vol. SC-15, pp. 253-254, Apr. 1980.
- [5.6] J.F. Gunn and R.L. Pritchett, 'A bipolar 16K ROM utilizing Schottky diode cells', ISSCC 77, pp. 118-119, Feb. 1977.
- [5.7] J.A. Ludwig, 'A 50K Schottky cell bipolar read-only memory', IEEE Jour. Solid-State Circ., Vol. SC-15, pp. 816-820, Oct. 1980.
- [5.8] Y. Akazawa, H. Kodama and T. Sudo, 'A high-speed 1600-gate bipolar LSI processor', ISSCC 78, pp. 208-209, Feb. 1978.

APPENDIX I
SOME DERIVATIONS OF RESIDUE NUMBER PROPERTIES

A1.0 RESIDUE NUMBER PROPERTIES

$$\begin{aligned}
 (1) \quad |x + km|_m &= |x|_m \\
 |x|_m &= x - m[x/m] \\
 |x + km|_m &= x + km - m[(x+km)/m] \\
 &= x + km - m[x/m] - m[k] \\
 &= x - m[x/m] \\
 &= |x|_m
 \end{aligned}$$

$$\begin{aligned}
 (11) \quad |x + kM|_m &= |x|_m \\
 |x + kM|_m &= x + kM - m[(x+kM)/m] \\
 &= x + kM - m[x/m] - m[k(\prod_{i=0}^{N-1} m_i)/m] \\
 &= x + kM - m[x/m] - m[k(\prod_{i=0}^{N-1} m_i)] \\
 &= |x|_m
 \end{aligned}$$

$$\begin{aligned}
 (111) \quad |kx|_{km} &= k|x|_m \\
 |kx|_{km} &= kx - km[kx/km] \\
 &= kx - km[x/m]
 \end{aligned}$$

$$\begin{aligned}
 |kx|_{km} &= k(x - m[x/m]) \\
 &= k|x|_m
 \end{aligned}$$

$$\begin{aligned}
 (iv) \quad | |x|_m \pm |y|_m | &= |x \pm y|_m \\
 | |x|_m \pm |y|_m | &= |x - m[x/m] \pm (y - m[y/m])|_m \\
 &= |x \pm y - m([x/m] \pm [y/m])|_m
 \end{aligned}$$

$$\text{But, from (i)... } (|x+km|_m = |x|_m)$$

$$| |x|_m \pm |y|_m | = |x \pm y|_m$$

$$\begin{aligned}
 (v) \quad | |x|_m |y|_m | &= |xy|_m \\
 | |x|_m |y|_m | &= |(x - m[x/m])(y - m[y/m])|_m \\
 &= |xy - m(x[y/m] + y[x/m]) + m^2 [x/m][y/m]|_m
 \end{aligned}$$

$$\text{But, from (i)... } (|x+km|_m = |x|_m)$$

$$| |x|_m |y|_m | = |xy|_m$$

(vi) Multiplicative Inverse

If $|ab|_m = 1$, then a is the multiplicative inverse of b , and the highest common factor of b and m is one.

Let the highest common factor of b and m equal d .

Then -

$$b = (k_1)d, \quad m = (k_2)d$$

$$\begin{aligned}
 |ab|_m &= |(a)(k1)d|_{(k2)d} \\
 &= d|(a)(k1)|_{(k2)} \quad \dots \text{from (iii)}
 \end{aligned}$$

Since $|ab|_m = 1$, then d must equal one.

(vii) Residue/ Binary Conversion with the Chinese Remainder Theorem

Since two numbers with the same residue values are equal (within the range $0 < x < N$), it is sufficient to show that the residue values of the CRT are equal to the original residues r .

The CRT states that:

$$|x|_M = \left| \sum_{j=0}^{N-1} \bar{m}_j \left(|r_j|_{\bar{m}_j} \right) \right|_M$$

$$\text{Where } M = \prod_{i=0}^{N-1} m_i, \quad \bar{m}_j = M/m_j$$

$$\text{Now, } x = [x/M] + |x|_M \quad \dots \text{by def'n of a residue}$$

$$\text{Since } | |x|_M |_{m_i} = |x|_{m_i}$$

$$\text{Then } | |x|_M |_{m_i} = |x|_{m_i}$$

From the CRT,

$$| |x|_M |_{m_i} = |x|_{m_i} = \left| \sum_{j=0}^{N-1} \bar{m}_j |r_j|_{\bar{m}_j} \right|_{m_i}$$

$$= \left| \sum_{j=0}^{N-1} (\bar{m}_j)(k_j) \right|_{m_i} \quad \dots k_j = \left| r // \bar{m}_j \right|_{m_j}$$

Now, $|(k)(m)|_m = 0$

$$\begin{aligned} \text{And } \left| \bar{m}_j \left| r // \bar{m}_j \right| \right|_{m_j} &= |(M/m_j)(k_j)|_{m_j} \\ &= |(m_j)(1/m_j)(M/m_j)(k_j)|_{m_j} \\ &= |(m_j)(k_j)(M/(m_j m_j))|_{m_j} \end{aligned}$$

Case (1)... If $i = j$, then -

$$M/(m_j m_j) = \prod_{\substack{s=0 \\ s \neq i, j}}^{N-1} (m_s)$$

$$\begin{aligned} \text{And } |(m_i)(k_j)(M/(m_j m_j))|_{m_i} &= |m_i(k'_j)|_{m_i} \\ &= 0 \text{ for all } k'_j \end{aligned}$$

Case (2)... If $i \neq j$, then -

$$\begin{aligned} |(m_i)(k_j)(M/(m_j m_j))|_{m_i} &= |m_i \left(\left| r // \bar{m}_j \right| \right) (M/(m_j m_j))|_{m_i} \\ &= \left| (r // (M/m_j)) (M/m_j) \right|_{m_i} \\ &= |r|_{m_i} = r_i \end{aligned}$$

Therefore $|x|_{\prod_{i=1}^m} = r$, and the CRT holds true.

(viii) Index Multiplication

Fermat's Theorem: $|a^{p-1}|_p = 1$

Let an integer n be written as:

$$n = (p-1)[n/(p-1)] + |n|_{p-1} \quad \dots \text{def'n of a residue}$$

$$\begin{aligned} |a^n|_p &= |a^{(p-1)[n/(p-1)] + |n|_{p-1}}|_p \\ &= |(a^{p-1})^{[n/(p-1)]} (a^{|n|_{p-1}})|_p \\ &= |(a^{p-1})^{[n/(p-1)]}|_p \cdot |a^{|n|_{p-1}}|_p \end{aligned}$$

But, $|a^{(p-1)[n/(p-1)]}|_p = 1 \quad \dots \text{from Fermat's theorem}$

$$\text{Therefore } |a^n|_p = |a^{|n|_{p-1}}|_p$$

MODULUS M

1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
1	1	0																														
2	1																															
3	1	2																														
4	1	0	3																													
5	1	3	2	4																												
6	1	0	0	5																												
7	1	4	5	2	3																											
8	1	0	3	0	5	0	7																									
9	1	5	0	7	2	0	4	8																								
10	1	0	7	0	0	0	3	0	9																							
11	1	6	4	3	9	2	8	7	5	10																						
12	1	0	0	0	5	0	7	0	0	0	11																					
13	1	7	9	10	8	11	2	5	3	4	6	12																				
14	1	0	5	0	3	0	0	0	11	0	9	0	13																			
15	1	8	0	4	0	0	13	2	0	0	11	0	7	14																		
16	1	0	11	0	13	0	7	0	9	0	3	0	5	0	15																	
17	1	9	6	13	7	3	5	15	2	12	14	10	4	11	8	16																
18	1	0	0	0	11	0	13	0	0	0	5	0	7	0	0	17																
19	1	10	13	5	4	16	11	12	17	2	7	8	3	15	14	6	9	18														
20	1	0	7	0	0	3	0	9	0	11	0	17	0	0	13	0	19															
21	1	11	0	16	17	0	0	8	0	19	2	0	13	0	0	4	5	0	10	20												
22	1	0	15	0	9	0	19	0	5	0	0	17	0	3	0	13	0	7	0	21												
23	1	12	8	6	14	4	10	3	18	7	21	2	16	5	20	13	19	9	17	15	11	22										
24	1	0	0	0	5	0	7	0	0	0	11	0	13	0	0	17	0	19	0	0	23											
25	1	13	17	19	0	21	18	22	14	0	16	23	2	9	0	11	3	7	4	0	6	8	12	24								
26	1	0	9	0	21	0	15	0	3	0	19	0	0	0	7	0	23	0	11	0	5	0	17	0	25							
27	1	14	0	7	11	0	4	17	0	19	5	0	25	2	0	22	0	10	23	0	16	20	0	13	26							
28	1	0	19	0	17	0	0	0	25	0	23	0	13	0	5	0	3	0	0	11	0	9	0	27								
29	1	15	10	22	6	5	25	11	13	3	8	17	9	27	20	12	21	26	16	19	4	24	23	7	19	14	28					
30	1	0	0	0	0	13	0	0	11	0	7	0	0	21	0	19	0	0	17	0	0	0	0	0	0	0	0	29				
31	1	16	21	8	25	26	9	4	7	28	17	13	12	20	29	2	11	19	10	14	3	24	27	22	5	6	23	10	15	10		
32	1	0	11	0	13	0	23	0	25	0	3	0	5	0	15	0	17	0	23	0	29	0	7	0	9	0	19	0	21	0	21	

TABLE A1.1 - Multiplicative inverse of X Modulus M (1//X)

END

0	6	1	0	9	H	8	4
---	---	---	---	---	---	---	---

FIN