

APPLICATION OF THE RESIDUE NUMBER SYSTEM
TO THE MATRIX MULTIPLICATION PROBLEM

A Thesis

by

GARY FRANKLIN CHARD

Submitted to Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 1989

Major Subject: Electrical Engineering

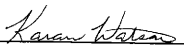
APPLICATION OF THE RESIDUE NUMBER SYSTEM
TO THE MATRIX MULTIPLICATION PROBLEM

A Thesis
by
GARY FRANKLIN CHARD

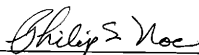
Approved as to style and content by:



Yu-Ying Jackson Leung
(Chair of Committee)



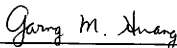
Karan L. Watson
(Member)



Philip S. Noe
(Member)



Donald K. Friesen
(Member)



Jo W. Howze
(Head of Department)

December 1989

ABSTRACT

Application of the Residue Number System
to the Matrix Multiplication Problem. (December 1989)

Gary Franklin Chard, B. S., Texas A&M University
Chairman of Advisory Committee: Dr. Yu-Ying Jackson Leung

The primary objective of this research is to evaluate a residue implementation of the matrix multiplication algorithm by comparison to a more conventional binary approach. Included in this research is a proposed method of concurrent residue multiplication and addition, as well as methods of input and output translation. Common building blocks are used repetitively throughout the design process, in an effort to minimize the design time of such a residue system. Logical simulation of the residue design was conducted for functional verification, as well as for a means of timing comparison to a more conventional design. It was found that the residue design was 2.73 times larger, and 3.18 times faster than the typical binary comparison structure. Many comments are presented throughout this thesis pertaining to considerations that must be made when contemplating the design of a residue system. The matrix multiplication algorithm is also simulated, such that exact timing information is given for both input and output matrix coefficients.

To my parents

ACKNOWLEDGEMENT

I would especially like to thank Dr. Leung for his advice and continual support throughout this research. I also thank him for helping me make graduate school a positive experience. I would also like to thank Dr. Watson, Dr. Noe, and Dr. Friesen for serving on my committee. Finally, I would like to thank my girlfriend Sherry for her moral support, and for her help in preparing the manuscript.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
DEDICATION	iv
ACKNOWLEDGMENT	v
LIST OF TABLES	ix
LIST OF FIGURES	x
CHAPTER	
I INTRODUCTION	1
1.1 Problem Statement	4
1.2 Approach	4
II BACKGROUND	7
2.1 The Residue Number System	8
2.1.1 Properties of the RNS	8
2.1.2 Basic RNS Identities	13
2.2 Basic Operations in The RNS	15
2.2.1 RNS Addition	16
2.2.2 RNS Multiplication	18
2.3 Translation from Binary to Residue	19
2.4 The Chinese Remainder Theorem	21
2.5 Sign Representation of a Residue Number	22
2.6 Introduction to Matrix Multiplication	24
2.7 The Matrix Multiplication Algorithm	26
2.7.1 The Multiply and Add Cell	27
2.7.2 Formulation of the Matrix Multiplication Algorithm	28
III APPLICATION OF THE RNS TO MATRIX MULTIPLICATION	31
3.1 Error Free Design	32
3.2 System Dynamic Range Determination	35

TABLE OF CONTENTS (Continued)

	Page
IV MATRIX MULTIPLICATION ALGORITHM SIMULATION	39
4.1 MAC Computing Structure	39
4.2 Input Matrix Coefficient Timing	41
4.3 Algorithm Simulation Development	44
4.4 Algorithm Simulation	45
V DESIGN DEVELOPMENT	49
5.1 Residue System Specifications	49
5.2 Multiply and Add Cell	52
5.2.1 MAC Functional Configuration	55
5.2.2 Modified Braun Array	57
5.2.3 Lower Truth Table Modulo m	61
5.2.4 Upper Truth Table Modulo m	66
5.2.5 Four-Bit Binary Adder	67
5.3 Input Translation	68
5.3.1 Input Operand Adjustment	70
5.3.2 Residue Digit Generation	72
5.4 Output Translation	75
5.4.1 Controlled Addition/Subtraction	78
5.4.2 Multiplication by Inverses	78
5.4.3 Correct Sign Determination	83
VI SIMULATION RESULTS AND COMPARISON	87
6.1 Simulation Development	87
6.2 Simulation Results	88
6.2.1 MAC Simulation	89
6.2.2 Input Simulation	91
6.2.3 Output Simulation	93
6.2.4 Global Considerations	95
6.3 Residue Design Area Calculations	96
6.3.1 MAC Area	97
6.3.2 Input Translation Area	98
6.3.3 Output Translation Area	98
6.3.4 Global Considerations	100
6.4 Design Comparison	101
6.4.1 Comparison Structure	102

TABLE OF CONTENTS (Continued)

	Page
6.4.2 Time and Area Comparison	103
VII CONCLUSION	105
7.1 Contributions	106
7.2 Future Research	108
REFERENCES	110
APPENDIX A MATRIX MULTIPLICATION ALGORITHM SIMULATION RESULTS	112
APPENDIX B TRUTH-TABLES AND KARNAUGH MAPS	116
APPENDIX C SCHEMATIC PLOTS	125
APPENDIX D SIMULATION RESULTS	136
VITA	150

LIST OF TABLES

Table	Page
2.1 Residue Representation of the Numbers from -4 to +32 for Moduli 2,3,5	12
2.2 Multiplicative Inverses	15
2.3 Partitioned Interval of Definition	23
3.1 Determination of Dynamic Input Range	36
4.1 Matrix A and B Input Coefficient Timing	43
4.2 Algorithm Simulation for Arbitrary Input Matrices	46
4.3 Output Coefficient Timing	47
5.1 Modulo 15 Truth Table	62
5.2 Multiplicative Inverses	80
6.1 Primitive Component Models	88
6.2 MAC Simulation Results	90
6.3 Input Translation Simulation Results	92
6.4 Output Translation Simulation Results	94
6.5 Processing Time Comparison	103
A.1 MAC Simulation Data	137
A.2 Input Translation Simulation Data	143
A.3 Output Translation Simulation Data	149

LIST OF FIGURES

Figure	Page
2.1 Multiply Add Cell	27
2.2 Hexagonal Array for Matrix Multiplication	29
2.3 Banded Matrix Multiplication	30
3.1 System Configuration	31
4.1 Computing Array for Matrix of Bandwidth Five	40
4.2 MAC Input/Output Naming Convention	41
4.3 Input Matrices of Bandwidth Five	42
5.1 Residue MAC Configuration	52
5.2 Proposed MAC Configuration of Each Modulus	56
5.3 Modified Braun Array	58
5.4 Full Adder Cell Design	60
5.5 Modified Braun Array Hardware	60
5.6 Modulo 15 Karnaugh Maps	63
5.7 Modulo 15 Truth Tables	65
5.8 Four-Bit Binary Adder	68
5.9 Input Translation Functional Configuration	71
5.10 Input Operand Adjustment	73
5.11 Mixed Radix Coefficient Determination	77
5.12 Conditional Adder	79
5.13 Non-Conditional Adder	79
5.14 Four-Bit Braun Array	81
5.15 Multiplication and Addition of the Mixed-Radix Coefficients	84
A.1 Modulo 7 Truth Table Hardware	126

LIST OF FIGURES (Continued)

	Page
A.2 Modulo 11 Truth Table Hardware	127
A.3 Modulo 13 Truth Table Hardware	128
A.4 8X4 Multiplier	129
A.5 Twelve-Bit Multiple Generator	130
A.6 Fourteen-Bit Carry Save Adder	131
A.7 Fourteen-Bit Binary Adder	132
A.8 Modified Adder A	133
A.9 Modified Adder B	134
A.10 Seventeen-Bit Two's Complementer	135
A.11 Modulo 7 Trial #1 MAC Simulation	138
A.12 Modulo 11 Trial #1 MAC Simulation	139
A.13 Modulo 13 Trial #1 MAC Simulation	140
A.14 Modulo 15 Trial #1 MAC Simulation	141
A.15 Modulo 16 Trial #1 MAC Simulation	142
A.16 Modulo 7 Trial #1 Input Translation Simulation	144
A.17 Modulo 11 Trial #1 Input Translation Simulation	145
A.18 Modulo 13 Trial #1 Input Translation Simulation	146
A.19 Modulo 15 Trial #1 Input Translation Simulation	147
A.20 Modulo 16 Trial #1 Input Translation Simulation	148

CHAPTER I
INTRODUCTION

Digital signal processing is a rapidly emerging technical area, where speed of computation is of prime importance, as well as practical considerations such as component packaging, silicon area, and cost. Some of the newest areas of interest in digital signal processing are real-time image processing, satellite communications, pattern recognition, and vector calculations. For these applications, parallelism has recently proven to be the key to faster processing of data. Parallelism may be achieved on mathematical, architectural, and realizational levels [1]. The residue number system, as will be seen shortly, achieves parallelism on a mathematical level.

Around 100 A.D., a Chinese mathematician named Sun Tzu authored a book containing a poem called t'ai-yen (great generalization). This poem was a puzzle, which challenged the reader to determine an integer number having a remainder of two, three, and two, when divided by three, five, and seven, respectively. The answer to the poem being the integer twenty-three. Although Sun Tzu did not know it at the time, he formed the basis of the Residue Number System (RNS), which would be studied in detail twenty centuries later. His poem is the equivalent of a

three-modulus RNS with three prime moduli $\{3,5,7\}$. This poem also stated a rule, refined by scholarly people over many centuries, called the Chinese Remainder Theorem [2]. It is the Chinese Remainder Theorem that allows conversion of the residue remainder digits back to an integer.

Between twenty and thirty years ago, a renewed interest in the Residue number system began. Szabo and Tanaka published a comprehensive book on the basic theorems and properties of the RNS [2]. Their primary interest in this number system was its application to the design and organization of digital computing machines. Without the invention of digital computers, the residue number system would most likely be as underdeveloped today as it was centuries ago. The techniques of addition, subtraction, multiplication, and division, as well as the fundamental properties and theorems of residue arithmetic were presented in [2]. Szabo and Tanaka concluded that operations such as addition, subtraction, and multiplication are simple operations to perform. Division, sign determination, and overflow detection were found to be difficult operations in both concept and implementation.

Since the renewed interest occurring in the mid 1960's, scientists have been studying and contributing to the topic of residue arithmetic. Industry has never adopted the residue number system as a viable alternative to the conventional binary number system. Several changing factors, such as the need for increased parallelism in

algorithms, new hardware capabilities, and semiconductor technology evolution, will soon cause the characteristics of the residue number system to be more closely examined. The number system has many inherent advantages over conventional number systems, as well as a few shortcomings, which subsequently have greatly limited the acceptance and use of the residue number system. Typical processors implemented today are unable to do matrix multiplication without careful programming by the user. Thus by implementation of a dedicated processor for the specific task of matrix multiplication, using current Very Large Scale Integration (VLSI) techniques, a cost and performance effective solution to the problem of matrix multiplication can be achieved. The approach that will be used in designing this dedicated processor, will be that of systematically connecting local processing elements in a parallel-pipelined fashion. In [3], an algorithm is proposed for matrix-matrix multiplication using a systolic array concept. Designs using the systolic array concept (simple and regular interconnections, parallel algorithms, and pipelining), have been proven to achieve a higher chip density, resulting in both a cheaper and a higher performance implementation. It is possible that further time enhancements may be made by the RNS, which has an inherent parallel nature, as compared to the conventional binary number system. The intent of this research is to

show that by applying the residue number system to a computationally intense problem, enhancements can be made over a comparable problem using the binary number system.

1.1 Problem Statement

Significant amounts of research effort have been expended investigating the properties of the residue number system, and its applications to current computer technology. The aim of this research is to determine if through applying the residue number system to the matrix multiplication problem, the solution time can be effectively reduced. Also, this research hopes to express several practical considerations to be dealt with when contemplating the use of a residue type design. Furthermore, the exact timing information of the matrix multiplication algorithm will be studied, in hope that a generalized timing equation can be derived.

1.2 Approach

The goal of this research is to quantify the processing time of matrix multiplication, using the residue number system. It is expected that the exact timing specifications of the system, as well as the exact chip area such a design would occupy will be determined. Moreover, the results of this research will allow the

comparison of the residue number system approach to that of the binary system. In the event that significant improvements over the binary number system are made, the use of the residue number system will greatly be promoted. Details of the approach towards the above stated goals will now be described.

There are several tasks to be considered in the design and simulation of a system as mentioned above. First, a method of translation to the residue number system which is suitable to a pipelined operation will have to be considered. There are currently several papers making comments on the translation problem from binary to the residue number system. In [4], ROM's (Read Only Memories) are used to accomplish part of the translation task. In a VLSI design, it is very desirable to avoid using ROM's from the aspect of their slow speed and area requirements. Thus, it will be important to develop a method of translation, avoiding the use of ROM's.

Also, on a larger architectural level, a systolic array method of matrix multiplication will be necessary [5]. The method of matrix multiplication, proposed in [3], uses a systolic array approach. This algorithm will be developed, and tailored to accommodate the RNS. The exact timing information will also be given.

Next, a method for implementation of the basic residue number operations such as addition and multiplication will

be investigated [6-8]. Once again, the design will avoid using the ROM approach, in search of a higher performance solution. Considerable time will be spent on the optimization of addition and multiplication processes, since the MAC (Multiply and Add Cell) will be the most prevalent processing element in the design.

Just as it was necessary to translate from the binary to the RNS, it will be necessary to translate from the RNS back to the binary number system [4, 9-12].

As a verification on the design process, and as a check on the timing information, the design will be simulated on an Apollo workstation using Mentor Graphics' Neted and Quicksim design tools.

CHAPTER II

BACKGROUND

During the 1950's, fabrication of transistors on crystalline silicon was developed. The integrated circuit plays a large role in society today. It has applications ranging from components in home stereos, to the electronic ignition control computer module in automobile engines. The integrated circuit is also fundamentally important to computers as we know them today.

Over a short time in the span of history, the integrated circuit (IC) has evolved from containing several transistors, to present technology of a million transistors on a single silicon chip.

The photolithographic process of the mid 1980's allows for the fabrication of integrated circuits very large in size, not previously possible, to be placed inside a single component package. Even more important than this, current research in component packaging includes effort in the area of multichip modules, where it will be possible to implement large circuits in several pieces, and combine them onto a silicon substrate, and encase them in one package [13].

Therefore, as technology progresses, the capability of fabricating complex systems such as the matrix multiplication algorithm, requiring significant amounts of

hardware, becomes more viable. It should be noted that the desire to implement such a specialized algorithm would be primarily for that of an increase in computational speed. The host processor could not possibly multiply two matrices of such a complexity in a comparable amount of time. The matrix multiplication algorithm will be responsible for a certain increase in speed, which is further enhanced through the application of the residue number system. The properties of the residue number system will now be investigated.

2.1 The Residue Number System

The notation used in introducing the properties and various aspects of the Residue Number System will be consistent with that of Szabo and Tanaka [2]. In cases where theorems are stated, the proofs will be omitted.

2.1.1 Properties of the RNS

Every number system has several characteristics allowing it to be distinguished from other number systems. Among these characteristics are the range, uniqueness in representation, and the base (radix) of the number system. The decimal number system and the binary number system are both fixed radix number systems. The decimal number system has a fixed radix of ten, the binary number system has a

fixed radix of two. The following illustrates the idea of a fixed radix system using the decimal number system as an example.

Example 2.1:

$$(12793)_{10} = 1 \cdot 10^4 + 2 \cdot 10^3 + 7 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0$$

Thus, we note that any decimal number can be expressed as a sum of its individual digits multiplied by the base raised to the appropriate power of the digit being expressed. In this case the digits are multiplied by powers of 10. The residue number is not a fixed radix number system. In fact, the residue system has more than one radix and is described by an N-tuple of integers $\{m_1, m_2, m_3, \dots, m_N\}$ where each of the integers m_i is called a modulus. This N-tuple of integers is often referred to as *moduli*, which is the plural form of the word modulus. Any number x in the residue number system can be expressed as an N-tuple of integers defined by a set of N equations:

$$x = q_i m_i + r_i \quad i = 1, 2, 3, \dots, N$$

where q_i is an integer chosen to ensure that r_i has a value equal to or greater than zero and less than the modulus m_i . The integer number r_i is the least positive remainder of the division of x by m_i . This value, r_i , is called the *residue of x modulo m_i* , often denoted by x/m_i . The quotient term, q_i , is often represented as $[x/m_i]$. A

commonly used form of the above equation is often expressed as:

$$x = m_i[x/m_i] + /x/m_i$$

where $/x/m_i$ is always a positive integer. The following example illustrates both the idea of an N-tuple of moduli, and the idea of an N-tuple of r_i .

Example 2.2:

For a three modulus system, with moduli given by an N-tuple ($N=3$) $\{m_1, m_2, m_3\} = \{3, 5, 7\}$, given an integer in the decimal number system $x = (37)_{10}$, a representation in the residue number system can be found as follows:

$$r_1 = /x/m_1 = /37/3 = x - m_1[x/m_1] = 37 - 3[12] = 1$$

$$r_2 = /x/m_2 = /37/5 = x - m_2[x/m_2] = 37 - 5[7] = 2$$

$$r_3 = /x/m_3 = /37/7 = x - m_3[x/m_3] = 37 - 7[5] = 2$$

Thus, the RNS representation of 37 is $\{r_1, r_2, r_3\} = \{1, 2, 2\}$.

Example 2.3:

Similarly, we can find the residue representation of the decimal number 142.

$$r_1 = 142 - 3[47] = 1$$

$$r_2 = 142 - 5[28] = 2$$

$$r_3 = 142 - 7[20] = 2$$

Thus, the RNS representation of 142 is given by the set $\{1, 2, 2\}$. It should be noticed that this is the exact

result obtained in the previous example when a RNS representation of 37 was found. It seems as though a contradiction has been made, implying that as far as the residue number system is concerned, the numbers 37 and 142 are identical. The following theorem will help resolve this paradox [2].

Theorem 2.1:

Two integers x and y have the same representation for a given set of moduli m_1, m_2, \dots, m_N if and only if $(x-y)$ is an integer multiple of the least common multiple of the moduli denoted by M .

The least common multiple of the moduli for the above examples is $M = (3)*(5)*(7) = 105$. If we denote $x = 37$, and $y=142$, then $(x-y) = 105$, which is an integer multiple of M . Thus as the theorem predicts, the numbers 37 and 142 should have the same residue representation. Further insight into the RNS can be obtained by examining Table 2.1. First we notice that $M = (2)*(3)*(5) = 30$. Also noticing that the residue representation of 0 is the same as the residue representation of 30, and that all the numbers between 0 and 30 have a unique residue representation. It is also true that any arbitrary interval of exactly 30 numbers denotes a unique mapping from decimal to a residue representation. The residue number system is periodic, and must be restricted to a single period, denoted by an interval of definition. It is

Table 2.1 Residue Representation of the Numbers from -4 to +32 for Moduli 2,3,5

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

fairly apparent that once a number is converted into its residue representation that it is not easy to determine the sign of a number, nor is it easy to perform any type of magnitude comparison among the residue digits. Consequently, this is one of the most serious disadvantages of the Residue Number System.

2.1.2 Basic RNS Identities

The following are several pertinent identities of the RNS, with the proofs omitted for the sake of brevity. Interested readers are urged to consult [2] for further details.

- (1) $0 \leq x/m < (m-1)$
- (2) $Km/m = 0$ for K an integer
- (3) $(x/m)/m = x/m$
- (4) $(x+mK)/m = (x-mK)/m = x/m$
- (5) $(-x)/m = ((m-1)x)/m = (m-x)/m$

Addition for a single modulus in the residue number system is now formulated by the following equation:

$$(x + y)/m = ((x/m) + (y/m))/m = (x/m + y/m) = (x + y)/m/m$$

$(x+y)/m$ is often referred to as the sum modulo m of x and y .

Multiplication for a single modulus in the residue number system is formulated by the following equation:

$$(x)(y)/m = ((x/m)(y/m))/m = (x)(y)/m/m$$

The properties of addition and multiplication modulo m will now be demonstrated in Example 2.5.

Example 2.5:

Let $m = 7$, $x = 32$, $y = 26$

$$/32/_{7} = 4 \quad /26/_{7} = 5$$

Addition

$$/x + y/_{7} = /32 + 26/_{7} = /58/_{7} = /4 + 26/_{7} = 2$$

Multiplication

$$/xy/_{7} = /(32)(26)/_{7} = //32/_{7}26/_{7} = /(4)(26)/_{7} = 6$$

The following multiplicative inverse theorem [2] is very useful in solving linear equations of the form $/ax/_{m} = /b/_{m}$. If $0 \leq a < m$ and $/ab/_{m} = 1$, then a is called the multiplicative inverse of b modulo m , and is denoted by $a = /1/b/_{m}$.

Theorem 2.2:

The quantity $/1/b/_{m}$ exists if and only if the greatest common divisor between b and m is equal to one and $/b/_{m}$ does not equal zero.

If the above theorem holds, then $/1/b/_{m}$ is unique. From Table 2.2 it is apparent that for a given number, a multiplicative inverse does not always exist.

Table 2.2 Multiplicative Inverses

Modulo 14

X	$ X _{14}$
1	1
2	None
3	5
4	None
5	3
6	None
7	None
8	None
9	11
10	None
11	9
12	None
13	13

Example 2.5:

The following equation may be solved using the multiplicative inverse theorem:

$$3x/7 = 4$$

$$/1/3/7 = 5 \quad \text{because } (a,b) = (3,5) = /(3)(5)/7 = 1$$

$$\text{Thus, } /x/7 = /(5)(4)/7 = 6$$

2.2 Basic Operations in the RNS

The previous section discussed the fundamental theorems and identities of the RNS. This section will

emphasize the operations on a complete residue representation, rather than on a system with a single modulus. For the discussion following, the reader should assume that we have a moduli set that is pairwise relatively prime. The assumption of pairwise relatively prime moduli will be commented on later.

2.2.1 RNS Addition

The basic identity for addition modulo m was defined for individual moduli. This basic definition can be extended to include systems where multiple moduli are to be used. Theorem 2.3 allows addition in systems with multiple moduli [2].

Theorem 2.3:

For a given residue system consisting of moduli $m_1, m_2, m_3, \dots, m_N$, let x and y be defined to be in the residue form. This residue form is denoted by $/x + y/M$.

$$\begin{array}{rcl}
 x & \langle \text{-----} \rangle & \{ /x/m_1, /x/m_2, \dots, /x/m_N \} \\
 + y & \langle \text{-----} \rangle & \{ /y/m_1, /y/m_2, \dots, /y/m_N \} \\
 \hline
 /x+y/M & \langle \text{-----} \rangle & \{ /x+y/m_1, /x+y/m_2, \dots, /x+y/m_N \}
 \end{array}$$

Also important, there exists one and only one integer, namely $/x+y/M$, with such a representation on the interval $\{0, M-1\}$. The following example illustrates the process of addition for a given set of moduli.

Example 2.6:

For the moduli 3,4,5, and 13, ($M = 780$) add

$$x = 124 \text{ <----> } \{ 1, 0, 4, 7 \}$$

$$y = 79 \text{ <----> } \{ 1, 3, 4, 1 \}$$

$$\text{Moduli: } \underline{3 \quad 4 \quad 5 \quad 13}$$

$$\begin{array}{r} 124 \quad \text{<----->} \quad \{ 1, 0, 4, 7 \} \\ + 79 \quad \text{<----->} \quad \{ 1, 3, 4, 1 \} \\ \hline /203/M = 203 \text{ <---->} \{ 2, 3, 3, 8 \} \end{array}$$

From Example 2.6, several comments can be made. First, the process of residue addition has no intermodular carries. Each residue digit of the result is only dependent upon the corresponding digit of the operands. Typical fixed radix number systems are not defined in such a way. The binary number system is used to illustrate this in Example 2.7.

Example 2.7:

Let x and y be binary numbers given by $x = (13)_{10} = (1101)_2$ and $y = (11)_{10} = (1011)_2$. The binary addition of x and y is shown below:

$$\begin{array}{r} 1101 \\ + 1011 \\ \hline 11000 \end{array} \quad \begin{array}{r} 111 \text{ <--- carry digits} \\ 1101 \\ + 1011 \\ \hline 11000 \end{array}$$

Note that in order to obtain the result, it is necessary to

generate carries from the least significant bit position towards the most significant bit position (left to right) so that the higher order resultant bits may be determined. It is this absence of interdigit carries that result in an inherent speed advantage over fixed radix systems. Also, notice that the result is obtained modulo M , such that if the result exceeds the value $M-1$, an ambiguity arises. This is a result of a previous identity, stating that $/x/m$ and $/x + mK/m$ will have the same residue representation.

2.2.2 RNS Multiplication

The basic identity for multiplication modulo m was defined previously for a system with a single modulus. This definition can be extended to include systems with multiple moduli, as was the case in addition [2].

Theorem 2.4:

For a given residue system consisting of moduli $m_1, m_2, m_3, \dots, m_N$, residue multiplication is defined for x and y by the following:

$$\begin{array}{l} x \text{ <-----> } \{ /x/m_1 \ , /x/m_2 \ , /x/m_3 \ , \dots \ , /x/m_N \} \\ x \ y \text{ <-----> } \{ /y/m_1 \ , /y/m_2 \ , /y/m_3 \ , \dots \ , /y/m_N \} \\ \hline /xy/M \qquad \qquad \qquad \{ /xy/m_1 \ , /xy/m_2 \ , /xy/m_3 \ , \dots \ , /xy/m_N \} \end{array}$$

Within the interval $(0, M-1)$ only one integer will have the above residue representation, namely $/xy/M$.

Example 2.8:

For the moduli 3, 4, 5, and 13, ($M=780$) multiply

$$x = 122 \quad \langle \text{-----} \rangle \quad \{ 2, 2, 2, 5 \}$$

$$y = 5 \quad \langle \text{-----} \rangle \quad \{ 2, 1, 0, 5 \}$$

$$122 \quad \langle \text{-----} \rangle \quad \{ 2, 2, 2, 5 \}$$

$$\times \quad 5 \quad \langle \text{-----} \rangle \quad \{ 2, 1, 0, 5 \}$$

$$\hline /610/_{780} = 610 \quad \langle \text{----} \rangle \quad \{ 1, 2, 0, 12 \}$$

The same comments apply to multiplication that applied to addition. Specifically, that multiplication is carry free between moduli, and results in ambiguity if xy exceeds $M-1$.

2.3 Translation from Binary to Residue

A vast majority of digital computers today use a form of the binary number system for computation. In order to use the RNS, it is necessary to translate from the conventional binary number system to the residue representation of a number. An integer x in the binary number system is described as follows:

$$x = 2^n b_n + 2^{n-1} b_{n-1} + \dots + 2^2 b_2 + 2^1 b_1 + b_0$$

Szabo and Tanaka observed that if the powers of 2 modulo m are stored in computer memory, that x/m could be computed by adding modulo m the powers of two which have a non-zero b_i [2].

Example 2.9:

$$\text{Let } x = (26)_{10} = (11010)_2$$

To compute $/x/3$, the following values should be computed prior to $/x/3$:

$$/2^4/3 = 1 \quad /2^3/3 = 2 \quad /2^2/3 = 1$$

$$/2/3 = 2 \quad /1/3 = 1$$

Computing $/x/3$:

$$\begin{aligned} /x/3 &= /(1)(1) + (2)(1) + (1)(0) + (2)(1) + (1)(0)/3 \\ &= /5/3 = 2 \end{aligned}$$

$$/26/3 = 2$$

Another method of input translation has been proposed that uses a variation on the idea presented above. The method uses $n/2$ processing elements (n = word length of weighted number), with each processing element responsible for storing the two residues of two consecutive bits in the input word [10]. Depending on whether a one or a zero is present for the specified input bit, the residue of the 2^n bit position is either added modulo m to the resultant of previous bits or zero added to the previous bits respectively. This design would be very suitable for pipelined operation, with the computation of each pair of residue digits for each clock stage. The matrix multiplication algorithm used in this research does not allow any speed increase with the application of pipelining.

2.4 The Chinese Remainder Theorem

The Chinese Remainder Theorem allows conversion out of a residue representation into a weighted number system [14]. Given a residue representation $\{r_1, r_2, r_3, \dots, r_N\}$, the Chinese Remainder Theorem makes it possible to determine x/M , provided that the greatest common divisor of any pair of moduli is one. A moduli set obeying this property is called *pairwise relatively prime*. The following theorem fails to hold if the requirement of pairwise relatively prime does not hold [2].

Theorem 2.4: Chinese Remainder Theorem

$$x/M = / (z_j / r_j | z_j / m_j) / M$$

where $z_j = M/m_j$, $M = (m_1)(m_2)\dots(m_N)$, and the greatest common divisor between any two moduli is one.

Example 2.10:

For the moduli $m_1 = 13$, $m_2 = 11$, $m_3 = 7$, and $m_4 = 9$, the number given by the residue representation $\{4, 2, 4, 7\}$ can be found as follows:

$$M = (13)(11)(7)(9) = 9009$$

$$z_1/13 = /(11)(7)(9)/13 = /693/13 = 4$$

$$z_2/11 = /(13)(7)(9)/11 = /819/11 = 5$$

$$z_3/7 = /(13)(11)(9)/7 = /1287/7 = 6$$

$$z_4/9 = /(13)(11)(7)/9 = /1001/9 = 2$$

$$x/9009 = /693/(10)(4)/13 + 819/(9)(2)/11 + 1287/(6)(4)/7 + 1001/(5)(7)/9/9009$$

$$\begin{aligned}
 &= / (693) (1) + (819) (7) + (1287) (3) + (1001) (8) /_{9009} \\
 &= / 18295 /_{9009} = 277
 \end{aligned}$$

There exists a modified form of the Chinese Remainder Theorem in the event that moduli are chosen such that they are not pairwise relatively prime. Interested readers should consult [2] for details of this modified Chinese Remainder Theorem.

2.5 Sign Representation of a Residue Number

Explicit sign representation of a number defines the case where the sign of a number can be determined by inspection. Such is the case with a signed magnitude representation of a binary number, where the most significant bit position gives the sign of the operand.

Implicit sign representation of a number defines the case where the sign information is not readily apparent upon inspection of a number. Implicit representation is the case when a number is in residue representation. It should be apparent from Table 2.1 that immediate determination of operand sign is virtually impossible upon inspection.

It is common practice to consider numbers in the range of $[0, M/2 - 1]$ as positive, and numbers in the range $[M/2, M - 1]$ as negative. This assignment is made assuming that the

dynamic range of the system will remain within the specified range of $[0, M-1]$, otherwise the actual resulting number, not to mention the sign, will be lost. The following example illustrates the partitioning of a residue system into positive and negative parts. Table 2.3 illustrates what is meant by dividing the interval of definition for a given set of moduli.

Table 2.3 Partitioned Interval of Definition

A = Actual Number
B = Partitioned Number

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

Example 2.11:

Let $x = 5$, and $y = -9$, from Table 2.3 the residue representation of x and y are as follows:

$$\begin{aligned}x &= \{ 1, 2, 0 \} \\y &= \{ 1, 0, 1 \} \\/ x + y /_M &= \{ 0, 2, 1 \}\end{aligned}$$

Since $x + y = 5 + -9 = -4$, we would expect in Table 2.3 for the residue representation of -4 to be $\{ 0, 2, 1\}$, this is exactly the case.

This concludes the introduction of the Residue Number System. There are many other theorems and identities which have not been presented. Division is presented in [2], but it is a complex operation when compared to addition and multiplication. This research will not need to implement a method of division, hence it will not be discussed. Next the basics of matrix multiplication, as well as the matrix multiplication algorithm will be examined.

2.6 Introduction to Matrix Multiplication

A matrix is simply an array of numbers denoted by A , in the form of:

$$A = \begin{array}{|cccc|} \hline a_{11} & a_{12} & \dots & a_{1n} \\ \hline a_{21} & a_{22} & \dots & a_{2n} \\ \hline \dots & \dots & \dots & \dots \\ \hline a_{m1} & a_{m2} & \dots & a_{mn} \\ \hline \end{array}$$

A matrix has m rows, and n columns. If A is a matrix, and has m rows and n columns, then in order to multiply B by A , we require that B be a matrix of n rows and p columns. The multiplication of two matrices is defined as follows [15]:

Let $C = AB = |c_{ij}|$, then C is known to be a matrix of m rows and p columns such that

$$C = a_{11}b_{1j} + a_{12}b_{2j} + \dots + a_{1n}b_{nj} \quad \begin{matrix} i = 1, 2, \dots, m \\ j = 1, 2, \dots, p \end{matrix}$$

Example 2.13:

$$\text{Let } A = \begin{vmatrix} 1 & 2 \\ 3 & 1 \end{vmatrix} \quad \text{and} \quad B = \begin{vmatrix} -2 & 5 \\ 4 & -3 \end{vmatrix}$$

Then

$$AB = \begin{vmatrix} (1)(-2) + (2)(4) & (1)(5) + (2)(-3) \\ (3)(-2) + (1)(4) & (3)(5) + (1)(-3) \end{vmatrix}$$

$$AB = \begin{vmatrix} 4 & -1 \\ -2 & 12 \end{vmatrix}$$

It should be noted from a computational standpoint, it takes a total of 8 multiplications and 4 additions to multiply simple 2-by-2 matrices together. In general, $[m \times n] \times [n \times m]$ requires $(m-1)n^2$ additions, and mn^2 multiplications. Clearly for large matrices, the number of multiply and addition steps increase rapidly. The following algorithm, which forms the foundation to which this research was applied, greatly reduces the amount of time required to multiply two matrices together.

2.7 The Matrix Multiplication Algorithm

One of the most successful applications of pipeline processing has been in the execution of arithmetic operations [3]. Pipelined operation allows for small portions of an overall task to occur at each position in the "pipe". This type of setup is extremely valuable when successive operations of the same type occur (e.g. an operation operates on a vector). It takes a certain amount of time to fill up the "pipe", which is known as the start-up or initialization time. If the successive operations on the vector are very long, the start-up time becomes very insignificant. A majority of supercomputers use multi-stage pipelining to achieve very fast operating speeds. A pipeline arithmetic unit can be visualized as a systolic array of linearly connected processors. Where each processor (processing element) is capable of performing a small portion of a global task. Kung found that the multiplication of two matrices could be done by using an array of hexagonally shaped processing elements [3]. This algorithm is suitable to Very Large Scale Integration (VLSI) where it is essential that processors are regular (in this case identical) and only locally connected. The basic processing element used by this algorithm is called an inner-product step processor. In this research, the inner-product step processor will be

referred to as a Multiply and Add Cell (MAC).

2.7.1 The Multiply and Add Cell

Figure 2.1 illustrates the shape of the multiply and add cell (inner product step processor), which is the most basic element in the matrix multiplication algorithm.

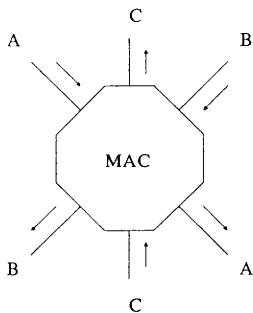


Figure 2.1 Multiply Add Cell

The MAC contains three registers R_A , R_B , and R_C , and six connections crossing the MAC boundary. Of the six connections, three are inputs and three are outputs. At each time interval, the processor transfers the data on its input lines denoted by A, B, and C into R_A , R_B , and R_C , respectively, then computes the value of $(R_A)(R_B) + (R_C)$, and

transfers the old values of R_A and R_B , along with the new value of R_C ($(R_A)(R_B)+(R_C)$) to the output lines, denoted A, B, and C, respectively. Since the inputs of each of the MAC's are latched, changing outputs will not interfere with the input of another MAC until the following clock cycle. It is this described cell, that will allow the multiplication of two matrices together by the following algorithm.

2.7.2 Formulation of the Matrix Multiplication Algorithm

The matrix product $C = (c_{ij})$ of $A = (a_{ij})$ and $B = (b_{ij})$, can be computed by the following relationships [3]:

$$\begin{aligned}
 c_{ij}^{(1)} &= 0 \\
 c_{ij}^{(k+1)} &= c_{ij}^{(k)} + a_{ik}b_{kj} \quad k = 1, 2, \dots, n \\
 c_{ij} &= c_{ij}^{(n+1)}
 \end{aligned}$$

Figure 2.2 illustrates the algorithm using a diamond-shaped array of linearly connected hexagonally shaped multiply and add cells.

The configuration of Figure 2.2 could be used to multiply the following matrices: $A \times B = C$

$$\begin{array}{|c|c|c|} \hline a11 & a12 & | \times | \\ \hline a21 & a22 & | \\ \hline \end{array} \begin{array}{|c|c|} \hline b11 & b12 \\ \hline b21 & b22 \\ \hline \end{array} = \begin{array}{|c|c|} \hline c11 & c12 \\ \hline c21 & c22 \\ \hline \end{array}$$

The algorithm is easily applied to larger matrices with the addition of more MAC's configured in a similar manner.

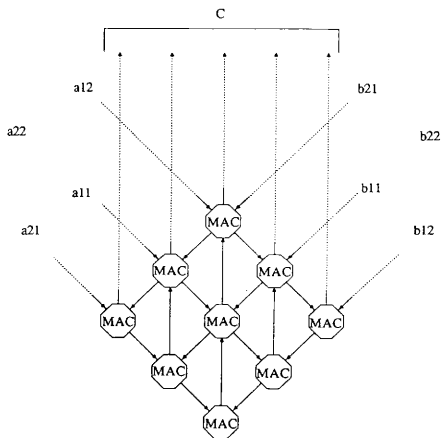


Figure 2.2 Hexagonal Array for Matrix Multiplication

The pattern for the input coefficients, as well as the timing constraints will be examined in Chapter IV.

The exact configuration of Figure 2.2 could be used to multiply two band matrices of larger dimension. The multiplication of two matrices with bandwidth $w_1 = p_1+q_1-1$ and $w_2 = p_2+q_2-1$, respectively, is shown in Figure 2.3.

$$\begin{array}{c} \uparrow \\ p_1 \\ \downarrow \end{array} \left[\begin{array}{ccccc} \xrightarrow{q_1} & & & & \\ a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{array} \right] \begin{array}{c} \uparrow \\ p_2 \\ \downarrow \end{array} \left[\begin{array}{ccccc} \xrightarrow{q_2} & & & & \\ b_{11} & b_{12} & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & 0 & 0 \\ 0 & b_{32} & b_{33} & b_{34} & 0 \\ 0 & 0 & b_{43} & b_{44} & b_{45} \\ 0 & 0 & 0 & b_{54} & b_{55} \end{array} \right] = \left[\begin{array}{ccccc} c_{11} & c_{12} & c_{13} & 0 & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & 0 \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ 0 & c_{42} & c_{43} & c_{44} & c_{45} \\ 0 & 0 & c_{53} & c_{54} & c_{55} \end{array} \right]$$

Figure 2.3 Banded Matrix Multiplication

From Figure 2.3, the bandwidth of A and B can be calculated to be $w_1 = 2+2-1 = 3$ and $w_2 = 2+2-1 = 3$ respectively. It should be noted that this is exactly the bandwidth of a matrix which has two columns and two rows. Thus the matrices given in Figure 2.3 could be multiplied using the MAC configuration of Figure 2.2. In general, if A and B are matrices of bandwidth w_1 and w_2 , then it takes $w_1 w_2$ hex-connected processors to compute the multiplication of A and B to obtain the resultant matrix C [3]

CHAPTER III
APPLICATION OF THE RNS TO MATRIX MULTIPLICATION

The matrix multiplication algorithm lends itself to an application requiring high speed multiplication. In addition to the requirement of high speed multiplication, the application must also have a need to multiply matrices with a very large dimension or very frequently. In most applications, the algorithm will ideally be implemented on a single chip. It is expected that this chip will be attached to a host processor, exchanging the various input and output operands through the system bus, as shown in Figure 3.1. [16].

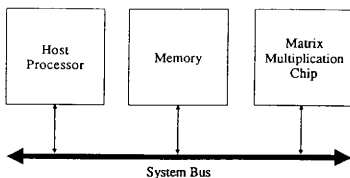


Figure 3.1 System Configuration

Applications without the need for high speed matrix multiplication, or without the need to multiply large matrices successively, can not efficiently use the algorithm. It should be clear that since the algorithm

will most likely be implemented on a separate chip, and that any further increases in speed, attributed to the Residue Number System, would be worth extra design time. While the amount of time saved for one multiplication of small matrices is not very significant, the amount of time saved for many successive multiplications adds up to be very relevant. As previously mentioned, certain residue operations are much more complex than other operations. The operations of multiplication and addition are among the simplest, while magnitude comparison, sign determination, and division have proved to be more difficult. The matrix multiplication algorithm is simplified into successive multiply and add operations. For this reason, it should be apparent that this algorithm is a prime candidate for operations of the residue type. It is also expected that if division was necessary, that the overhead required to do this might be fatal to the application of the Residue Number System. The following section discusses the considerations necessary to successfully implement the matrix multiplication algorithm using the RNS.

3.1 Error Free Design

The residue number system has a very unique property, being that it does not suffer from round-off error. This can be used to the systems advantage or disadvantage

depending on the application. In an application where exclusively integers will be manipulated, this is a very highly desired feature. In applications where fractions are being manipulated, error of some magnitude is both tolerated and expected. In the conventional binary number system, when two numbers of arbitrary word length n are multiplied together, it is possible to get resultant word lengths of $2n$. It is common practice to truncate the lower n bits when dealing with fractions. Conversely, when dealing with integers, it is common to designate a certain upper limit number of bits for the system. Any time this system upper limit is exceeded, overflow is said to have occurred. Upon the occurrence of overflow, the result of the calculation may be only partially complete, or completely incorrect. At any rate, the answer is inadequate, and should never be used for any further calculation.

It should be clear that in a residue system design, it is not important to designate at the onset whether the input operand will be a fraction or an integer. The system will produce the entire output length, depending on the application, the designer can truncate the upper or lower portion of unused bit positions, for integer or fractional designs, respectively.

One consequence of the above mentioned error free property, is that the overall system dynamic range must be determined. Considering an ordinary binary system, where

successive multiplication and addition processes occur, it can be shown that if the dynamic range of the system (i.e. maximum word length of the binary operands) is exceeded before any of the above mentioned processes are complete, that the result will be both incorrect and unusable. In a residue system equivalent, if the dynamic range is exceeded during some calculation internal to the overall process, there is still hope. It is only mandatory that the end result remain in the dynamic range of the given system. The following example should clarify this issue.

Example 3.1:

Let the moduli of a system be $m_1 = 3$, and $m_2 = 5$.

$$M = (3)(5) = 15$$

Thus, the system interval of definition is $(0,14)$.

Suppose $z = (a)(b) + (c)$ is to be calculated, where

$$a = (4)_{10} \quad \langle \text{-----} \rangle \quad /a/M = \{ 1, 4 \}$$

$$b = (6)_{10} \quad \langle \text{-----} \rangle \quad /b/M = \{ 0, 1 \}$$

$$c = (-10)_{10} \quad \langle \text{-----} \rangle \quad /c/M = \{ 2, 0 \}$$

$$\begin{array}{r} a \qquad \qquad \qquad \{ 1, 4 \} \\ \times b \qquad \qquad \times \{ 0, 1 \} \\ \hline (24)_{10} \qquad \qquad \{ 0, 4 \} \end{array}$$

Note: $(a)(b)$ has already exceeded the interval of definition, even so the calculation is continued.

$$\begin{array}{r} 24 \qquad \qquad \qquad \{ 0, 4 \} \\ + -10 \qquad \qquad \times \{ 2, 0 \} \\ \hline (14)_{10} \quad \langle \text{----} \rangle \quad \{ 2, 4 \} \end{array}$$

$ab + c = (24)_{10} + (-10)_{10} = (+14) \leftarrow$ desired result
Since $\{ 2, 4 \}$ is the residue representation of 14,
the calculation is exactly correct.

This property has no parallel in the binary number system. If at any point in a binary calculation overflow occurs, the resulting calculation has no predictable chance of being correct.

3.2 System Dynamic Range Determination

There are two ways of determining the dynamic range of a residue system design. It is possible to examine the input word length, and make a calculation to determine the maximum possible value at the output, assuming worst case (largest valued) numbers at the input, for all inputs. The second approach is to agree on a maximum allowable output, and hope that this range is never exceeded. This particular method would be particularly useful if a designer knows ahead of time that a certain output value will never be exceeded. In this case, the design would be simplified accordingly. In this research, the first approach is used.

The assumed input operand format in this design is presumed to be that of a signed magnitude number. This format is typical in floating point processors, although this is not a floating point processor. It would allow

easier communication with a floating point processor, since the operands will be input and output in the same format. It may be advantageous to place an intermediate processor between the host and matrix multiplier, for the purpose of pre-adjusting the mantissa of an input floating point number, also for the purpose of readjusting the mantissa on return to the host processor [16]. This process should be pipelined, so that the time required to adjust the operands does not affect the performance of the algorithm.

The actual dynamic range of this system will be directly determined by two factors. The first factor is the input operand word length. The second factor depends on the size of the input matrices to be multiplied. Table 3.1 shows the value determining the maximum value possible in any position of the output matrix.

Table 3.1 Determination of Dynamic Input Range

		Square Input Matrix Dimension					
A	B	3	4	5	6	7	8
2	1	6	8	10	12	14	16
3	3	54	72	90	108	126	144
4	7	294	392	490	588	686	784
5	15	1350	1800	2250	2700	3150	3600
6	31	5766	7688	9610	11532	13454	15376
7	63	23814	31752	39690	47628	55566	63504
8	127	96774	129032	161290	193548	225806	258604

A = Signed Magnitude Input Word Length

B = Maximum Possible Magnitude of Input

From Table 2.1, for signed magnitude input word length, the maximum magnitude of the input can be calculated by $2^{n-1}-1$. For an input word length of eight bits ($n=8$), the maximum input value can be calculated as $2^{(8-1)}-1 = 2^7-1 = 127$. The maximum value of any number in the output matrix can be determined by:

$$X = (\text{dimension of input matrix}) (\text{maximum input value})^2$$

The above formula is assuming square input matrices.

Example 3.2 demonstrates the calculation of this value.

Example 3.2:

The maximum value of any one value in the output matrix can be calculated given both the square input matrix size, and the input word length. If the matrix input size is 4, and the input word length is 7, we can calculate the upper bound of any entry in the output matrix as follows:

$$x = (4) ((2^{(7-1)}-1))^2 = (4) (63)^2 = 15876$$

Consulting Table 3.1, for an input word length of 7, and a matrix size of 4, we do not get the same result as Example 3.2. This is because in Example 3.2, only the positive output range was considered, so that the total range can be found by doubling the positive dynamic range. In the case of Example 3.1, the total dynamic output range is given by $(2)(15876) = 31752$.

The motivation for finding the output range so

meticulously is due to the nature of the residue system. Remembering that the result will only be correct in the case where it is enclosed by the interval of definition. The approach taken in such a design might be that the output must be correct for all possible inputs. Another approach could be that of a defined interval of definition, with some sort of assurance that the defined interval will never be exceeded. Once again, the philosophy behind this design is that the correct result will be achieved for all possible input combinations of a given word length.

In this design it was decided that an eight-bit input operand, and an input matrix of bandwidth five would be sufficient to demonstrate the advantages and disadvantages of a residue system design. Specifying a bandwidth of five also specifies the maximum allowable output operand value, in exact accordance with Table 3.2. This is very convenient from the standpoint of a general design. For an input matrix of any arbitrary size, the algorithm works as long as the bandwidth of the arbitrary matrix is less than or equal to five. In the event that the matrix has a bandwidth less than five, zeros should be input at the unused input ports.

CHAPTER IV
MATRIX MULTIPLICATION ALGORITHM SIMULATION

The reference presenting this algorithm fails to adequately introduce the necessary timing information to successfully implement the algorithm [3]. It will be the purpose of this chapter to develop and demonstrate the application of the algorithm itself. Specifically, the algorithm will be demonstrated for input matrices with an input bandwidth of five. The timing parameters obtained from this simulation will be needed later.

4.1 MAC Computing Structure

The computing structure will contain w_1w_2 multiply and add cells. Therefore, a diamond shaped array of twenty-five processors will be necessary to implement the algorithm. Figure 4.1 shows the structure to be used for the simulation. A MAC referencing system is necessary, so that each separate MAC can be identified individually from the surrounding processors. As shown in Figure 4.1, the numbering convention is that of starting at the top, and numbering each MAC from left to right, consecutively, in a row-wise fashion. With the exception of the lower-most MAC in each vertical column, each MAC has six external boundary

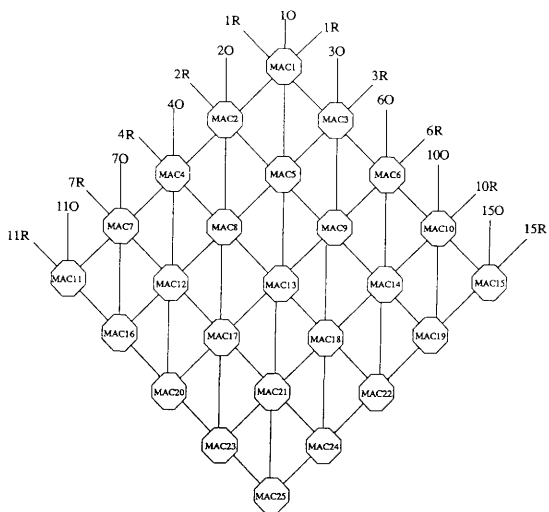


Figure 4.1 Computing Array for Matrix of Bandwidth Five

connections. These connections must also have distinguishable names. The naming convention for MAC #4 is shown in Figure 4.2.

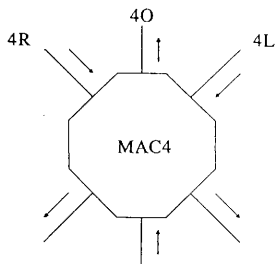


Figure 4.2 MAC Input/Output Naming Convention

It should be noted that all inputs to the MAC traveling from left to right, are labeled 4R. All inputs to the MAC traveling from right to left are labeled 4L. The upwards going input is labeled 4I, while the upwards going output is labeled 4O. All other cells are named in a similar convention.

4.2 Input Matrix Coefficient Timing

Crucial to the success of this algorithm is the pattern of input coefficients. The pattern is somewhat

regular in structure. After simulation of the algorithm is complete, the output port timing coefficients will be apparent. Timing patterns for $A = (a_{ij})$ and $B = (b_{ij})$ will be examined. Figure 4.3 shows the exact format of the input matrices A and B, each having a bandwidth of five.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & a_{33} & a_{43} & a_{44} & a_{45} \\ 0 & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & 0 & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} & 0 \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ 0 & b_{33} & b_{43} & b_{44} & b_{45} \\ 0 & 0 & b_{53} & b_{54} & b_{55} \end{bmatrix}$$

Figure 4.3 Input Matrices of Bandwidth Five

The A matrix input coefficients will march into the computing array of Figure 4.2 from the left hand side towards the right. The B matrix input coefficients will march into the computing array from the right hand side towards the left. There are five input ports into which the A matrix coefficients will go, namely 11R, 7R, 4R, 2R, and 1R. There are also five input ports into which the B matrix coefficients will go, namely 1L, 3L, 6L, 10L, and 15L. Table 4.1 shows the input port coefficient timing table. The input timing was found by trial and error. All coefficients on a horizontal row are input into the same

Table 4.1 Matrix A and B Input Coefficient Timing

Matrix A Input Timing Table

		Coefficient Movement									
		←									
Input Port	11R		a31		a42		a53				
	7R	a21		a32		a43		a54			
	4R	a11		a22		a33		a44		a55	
	2R		a12		a23		a34		a45		
	1R			a13		a24		a35			
		Increasing Time →									

Matrix B Input Timing Table

		Coefficient Movement									
		←									
Input Port	1L				b31		b42		b53		
	3L		b21		b32		b43		b54		
	6L	b11		b22		b33		b44		b55	
	10L	b12		b23		b34		b45			
	15L		b13		b24		b35				
		Increasing Time →									

input port. All coefficients on a vertical segment are input at different ports, during the same clock cycle. Going in increments of one from left to right corresponds to one clock cycle. This means that input coefficient b_{11} is input one clock cycle before coefficient b_{12} , and that input coefficient b_{24} is input three clock cycles before input coefficient b_{35} . With the input coefficients timing established, the development of the simulation may be examined.

4.3 Algorithm Simulation Development

In order for the development to be clear, the reader is recommended to consult Figure 4.1 and Figure 4.2 as necessary. The algorithm may be broken down into three basic transfers at the MAC level. Each of these transfers occurs at the onset of a clock cycle. The first basic transfer is an operand traveling from left to right across the array. An example of this is the operand at 5R being transferred to 9R. The second basic transfer is similar to the first basic transfer, except instead of moving from left to right, an operand moves from right to left. An example of this is the operand at 8L being transferred to 12L. The third basic transfer is only slightly more complex than the first two. During any given clock cycle, the MAC computes the product and sum of the three inputs, which appears at the output of the mac before the next

clock cycle begins. Using MAC9 from Figure 4.1 as an example, in a given clock cycle, MAC9 multiplies (9R) (9L), and adds to this (9I), and places this result on 9O before the clock cycle is finished. At the onset of the next clock cycle, the operand at 9O will be stored in the register in front of MAC3. In other words, the value at 9O is transformed into 3I when the clock pulse occurs. A program was written modeling this transfer level description of the MAC array. The complete output of the simulation may be found in Appendix A of this thesis, but the results are discussed here.

4.4 Algorithm Simulation

Simulation of this algorithm is important for several reasons. First, the determination of the input coefficient timing is necessary to implement this array structure. Of equal importance, the output coefficient timing will be determined by examining the simulation results. Finally, it is important in this research to determine exactly how many clock cycles it takes to complete a complete matrix multiplication process, from start to finish.

The simulation input matrices, as well as the resultant matrix is shown in Table 4.2. The input matrices A and B were chosen arbitrarily to demonstrate how the algorithm works. Table 4.2 shows the actual A and B input

Table 4.2 Algorithm Simulation for Arbitrary Input Matrices

Input Matrix A	Input Matrix B	Resultant Matrix C
2 3 9 0 0	1 4 8 0 0	-25 -40 -62 45 -36
1 2 3 -3 0	3 2 -5 3 0	-5 -31 -32 6 9
4 5 7 -2 2	-4 -6 -7 4 -4	-9 -30 -66 33 -24
0 6 -2 1 5	0 7 3 4 -7	26 31 -58 9 -24
0 0 -9 9 3	0 0 -9 -1 -5	36 117 63 -3 -42

Matrix A Input Timing															
11R	0	0	4	0	0	6	0	0	-9	0	0	0	0	0	0
7R	0	1	0	0	5	0	0	-2	0	0	9	0	0	0	0
4R	2	0	0	2	0	0	7	0	0	1	0	0	3	0	0
2R	0	0	3	0	0	3	0	0	-2	0	0	5	0	0	0
1R	0	0	0	0	9	0	0	-3	0	0	2	0	0	0	0
Matrix B Input Timing															
1L	0	0	0	0	-4	0	0	7	0	0	-9	0	0	0	0
3L	0	0	3	0	0	-6	0	0	3	0	0	-1	0	0	0
6L	1	0	0	2	0	0	-7	0	0	4	0	0	-5	0	0
10L	0	4	0	0	-5	0	0	4	0	0	-7	0	0	0	0
15L	0	0	8	0	0	3	0	0	-4	0	0	0	0	0	0
Output Matrix Timing															
11O	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0
7O	0	0	0	0	0	0	0	26	0	0	117	0	0	0	0
4O	0	0	0	0	0	0	-9	0	0	31	0	0	63	0	0
2O	0	0	0	0	0	-5	0	0	-30	0	0	-58	0	0	-3
1O	0	0	0	0	-25	0	0	-31	0	0	-66	0	0	9	0
3O	0	0	0	0	0	-40	0	0	-32	0	0	33	0	0	-24
6O	0	0	0	0	0	0	-62	0	0	6	0	0	-24	0	0
10O	0	0	0	0	0	0	0	45	0	0	9	0	0	0	0
15O	0	0	0	0	0	0	0	0	-36	0	0	0	0	0	0

coefficients as they are input into the array. Also shown are the output coefficients, which will be used to generalize the output matrix coefficient timing sequence.

The output coefficient timing can now be generalized for the C matrix, and is shown in Table 4.3.

Table 4.3 Output Coefficient Timing

110					C51								
70				C41			C51						
40			C31			C41			C53				
20		C21			C31			C43			C54		
10	C11			C22			C33			C44			C55
30		C12			C23			C34			C45		
60			C13			C24			C35				
100				C14			C25						
150					C15								

The amount of time for one complete matrix multiplication can now be extracted. The reader should note the regularity of the wedge shaped output coefficient pattern in Table 4.3.

Several comments can now be made about the overall timing of the algorithm. It takes five clock cycles before any output coefficient appears at an output port. It takes an additional twelve clock cycles before the last coefficient is output. Thus, it takes a total of

seventeen clock cycles to completely multiply two matrices of bandwidth five together.

In general, the processing time to multiply two matrices of an arbitrary bandwidth w ($w = w_1 = w_2$) is given by the following equation:

$$T_p = (3w - 4) + 3(S)$$

where:

T_p = Overall Processing Time

S = Dimension amount larger than a minimum sized matrix of the same bandwidth

For example, a minimum sized matrix of bandwidth 3 is a 2X2 matrix, while a minimum sized matrix of bandwidth 5 is a 3X3 matrix. The amount of time to multiply two matrices of bandwidth five, and dimension 4X4 is calculated from the above equation $T_p = (3(5) - 4) + 3(4-3) = 14$ clock cycles.

This concludes the algorithm simulation discussion. The whole simulation output, rather than the summarized results presented in this chapter, can be found in Appendix A of this thesis.

CHAPTER V

DESIGN DEVELOPMENT

This chapter introduces the design to be simulated in this research. The approach of this chapter will be that of a detailed presentation, such that this design could be duplicated by the reader. Some of the smaller details will be presented as they are important in the design of a residue system. The following section describes the design at a system level. Subsequent sections examine the steps in designing the major blocks of the system.

5.1 Residue System Specifications

It was agreed upon that the design to be simulated in this research should be large enough to demonstrate the application of the residue system to a problem of useful complexity. The design presented in this chapter assumes an input operand of eight bits, in signed magnitude format. This allows an input range from -127 to +127. An upper limit bandwidth of five is placed on the input matrices A and B. There are many scientific algorithms requiring the multiplication of banded form matrices with bandwidth dimensions of five and smaller.

With the global system requirements specified, the process of moduli selection may begin. First the dynamic

range of the system must be determined. From Table 3.1, for an input of eight bits, and a matrix size of 5, we see that the overall dynamic range of the system is 161290. Although the selection of moduli is arbitrary, it is beneficial to choose pairwise relatively prime moduli. This is done so that the Chinese Remainder Theorem can be implemented, rather than the alternate form of the Chinese Remainder Theorem. Before the moduli set chosen for this design are presented, several comments should be made about moduli selection. It is important to have as few moduli as possible, yet it is also true that hardware complexity increases as the moduli size increases. There exists a set of equations to generate moduli sets that are pairwise relatively prime. It is not convenient to use these equations, since they tend to select small numbers initially, with the moduli size growing very rapidly. The design of the system presented here chose a set of five moduli, although a set of four moduli of the proper magnitude would successfully satisfy the system requirements. The reason five moduli were presented instead of four will be explained shortly. The moduli set for this research is given by $\{m_1, m_2, m_3, m_4, m_5\} = \{7, 11, 13, 15, 16\}$. First it should be noted that the moduli are pairwise relatively prime. The moduli seven, eleven, and thirteen are prime numbers themselves, so there is no concern that these moduli are not pairwise relatively prime. Fifteen and sixteen have a greatest common divisor

of one, so they are pairwise relatively prime with respect to themselves, and with respect to the other moduli as well. The reason for selecting five moduli, instead of four, is due to the convenience of implementing modulo 2^n addition and multiplication. It was found that modulo 2^n addition and multiplication are identical to conventional binary addition and multiplication. Although it is not apparent to the reader at the present time, it will be demonstrated that very little effort will be required to implement the modulo 16 operations. Thus, designing the MAC portion of the system will be equivalent to the design of a four moduli system. The primary advantage of having five smaller moduli, rather than four larger, is the speed of the computation involved. Remembering that the modulus of a number is governed by the following equation:

$$0 \leq x/m_i < m_i$$

Thus the range of the residue representations for all the moduli will be between zero and fifteen. All of which can be expressed by four binary digits. This will not be a critical component of the design, but will contribute to the performance of the residue design. The choice of the moduli set is a task left to the designer. Assuming the set of moduli will satisfy the system requirements, there is no simple and clear cut way to arrive upon the optimum set. There is no guarantee that the moduli set chosen in this research is optimum. It is not even clear what the

word optimum means, since in one design it may be necessary to optimize the circuit area, the speed of operation, or a combination of both.

With the moduli set chosen above, the various facets of the design may now be investigated. The most prevalent portion of the design is the multiply and add cell, which will be presented first. Also, the approach used to translate into and out of the residue representation will be presented.

5.2 Multiply and Add Cell

The multiply and add cell is the most basic building block of the matrix multiplication algorithm. The MAC of the residue design will still be referred to as a MAC, although it will consist of five smaller blocks. The residue MAC is shown in Figure 5.1.

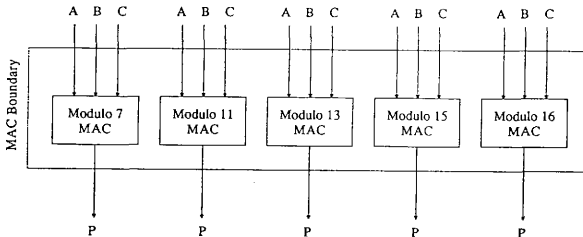


Figure 5.1 Residue MAC Configuration

It should be noted that each MAC independently processes the corresponding residue digits of the input operands A, B, and C.

There are several approaches that can be made when implementing modulo addition and multiplication. There has previously not been any research effort in concurrent addition and multiplication modulo m . It would be easy to implement modulo m multiplication, and then implement modulo m addition, but this method was found to be very time consuming. There are very few if any papers on implementing residue multiplication without using the ROM approach, which is not a viable solution to this problem. There are several methods of implementing residue addition. One method proposed adds two numbers together, then subtracts the modulus from the result of addition repetitively until the sum changes from a positive sign to a negative sign, when this occurs, the modulus is then added back to the current sum, which then becomes the result modulo m [7]. This method is acceptable when the addition process can be pipelined, but is very time consuming to implement sequentially due to the large number of subtractions necessary. Another proposed method recognizes that residue addition is cyclic, and as a consequence of this, uses shift and rotate logic to correctly select the desired result [8]. This method grows very large in complexity, even for moduli of modest size. It was reported that implementation of modulo fifteen

addition requires over seven hundred logic gates. This method will not be acceptable in this research either. It must be remembered that this does not include hardware requirements to implement modulo m multiplication.

An alternative approach to the problem is to examine the two processes that must occur simultaneously, namely modulo m multiplication and addition. Each of the five digits of a residue representation can be expressed in four binary digits, with the exception of the first modulus, which is 7, where its residue digits can be expressed in three binary bits. The method proposed in this research is a hybrid approach to the problem in the sense that operations of both binary and residue types will be used. It was found that after the binary multiplication and addition of A , B , and C for each modulus has occurred, the result could be taken modulo m_i . This approach requires the implementation of two truth tables for each modulus, requiring a total of eight truth tables for the entire design. Although it is undesirable to have large amounts of truth tables in a design due to their irregular structure, in this case it will be acceptable, because the same structure will be used repetitively. It will be found later that both the input and output translation problems will use the same modular truth table blocks. Another important factor in allowing the use of truth tables is their simplicity in design. Each of the truth tables will

only be required to have five inputs, and four outputs. A generalization of this approach to designs of larger complexity will be made at later point in this research.

5.2.1 MAC Functional Configuration

The MAC configuration proposed in this research is shown in Figure 5.2. It should be noted that this configuration is for each modulus inside the MAC boundary shown, thus there will be four such designs inside the MAC. There would be five, except modulo 16 operations are simplified and will not need the full configuration as shown in Figure 5.2. Also, the modulo 7 design will not have as many input and output bits, but the overall structure will be identical. Before the design of the individual blocks of the MAC are discussed, an example will be used to illustrate operation at a functional level for an individual modulus.

Example 5.1:

For the modulus 15, given the following inputs:

$$A = (12)_{10} = (1100)_2$$

$$B = (9)_{10} = (1001)_2$$

$$C = (5)_{10} = (0101)_2$$

the result $P = Ax + B + C$ can be computed using the proposed structure as follows:

From the Modified Braun Array,

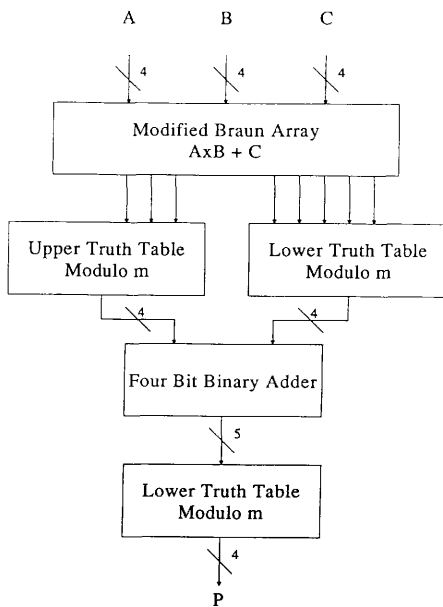


Figure 5.2 Proposed MAC Configuration for Each Modulus

$$A \times B + C = (12)(9) + (5) = (113)_{10} = (1110001)_2$$

From the Upper Truth Table,

$$/(1100000)_2 /_{15} = /(96) /_{15} = (6)_{10}$$

From the Lower Truth Table:

$$/(10001)_2 /_{15} = /(17) /_{15} = (2)_{10}$$

From the Four-Bit Binary Adder,

$$(2) + (6) = (0010)_2 + (0110)_2 = (1000)_2 = (8)_{10}$$

From the second Lower Truth table,

$$/(8)_{10} /_{15} = /(1000) /_2 = (8)_{10} = P$$

Thus, the modulo 15 result $P = A \times B + C$ is:

$$/(113)_{10} /_{15} = P = (8)_{10}$$

This is the exact answer obtained by the MAC configuration, hence the configuration is functionally correct.

This concludes the description of the MAC configuration at the functional level. The individual blocks of the functional configuration of Figure 5.2 will now be examined closer at the design level.

5.2.2 Modified Braun Array

A modified form of the braun array is used for the binary multiplication and addition of input operands A, B, and C. The structure to be used in this design is shown in Figure 5.3. This structure is the common input stage to all moduli of the MAC.

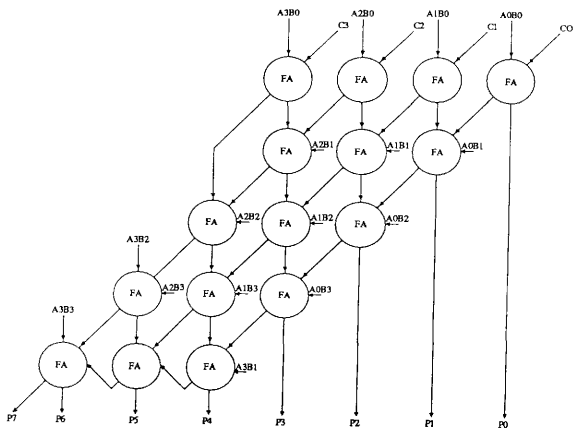


Figure 5.3 Modified Braun Array
 $P = Ax + B + C$

A typical braun array consists of the lower portion of the array of full adders. A top row of full adder cells was added to the top of the array in order to accommodate the additive input C. Thus the multiplication of A and B, and the addition of C to the result of A and B occurs simultaneously. Full adder cells will appear throughout this thesis, and it is appropriate at this point to introduce the repetitively used full adder cell. The unmodified full adder cell is shown in Figure 5.4a. The full adder cell in Figure 5.4b is modeled after the cell in Figure 5.4a, with several "cosmetic" differences. The full adder cell in Figure 5.4b is actually identical to the cell in Figure 5.4a. First, the plotter resolution does not include the bubble at the output of NAND gates, so that NAND gates appear to be AND gates, although this is not the case. Secondly, the very last gate in Figure 5.4b really is a hardwired AND gate, but Neted does not contain a hardwired AND gate in the component library. Thus, a regular AND gate with an area and time delay of zero was inserted for functional and simulation purposes. Thirdly, the vertical dimension of the actual implementation is reduced such that more full adder cells could be placed on the schematic editor screen. It is for this reason that the NAND gates are offset from one another rather than in a straight line. The hardware implementation of the modified Braun Array is shown in Figure 5.5. This array will be used at several places, and will be referred to regularly.

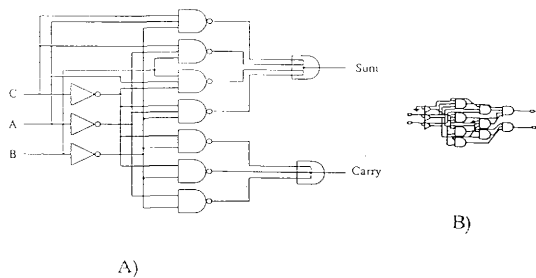


Figure 5.4 Full Adder Cell Design

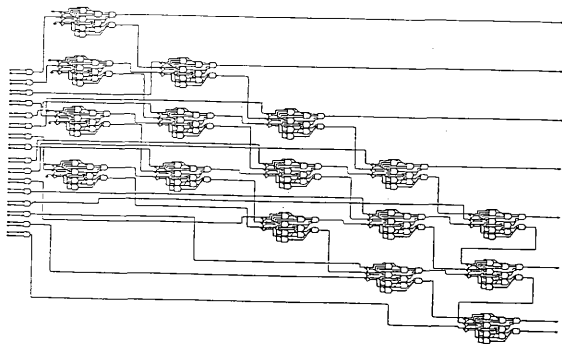


Figure 5.5 Modified Braun Array Hardware

5.2.3 Lower Truth Table Modulo m

The lower truth table block has five inputs, and four outputs. The input bits are the five lowest order bits of the result from the braun array. The output bits represent the input modulo m (/input/_m). There are five different moduli in this design, but only four of them have hardware requirements. Modulo 16 operations are equivalent to typical binary operations, thus no further manipulation of the result from the braun array is necessary (the lower four bits are the needed result). The detailed and complete design procedure of the lower modulo 15 truth table will be presented, as well as the results of the modulo 7, modulo 11, and modulo 13 truth tables.

Table 5.1 shows the five-bit binary input decimal equivalent, as well as the result modulo 15, which is the desired output of the truth table. The four truth table outputs are labeled W, X, Y, and Z, in order of decreasing significance. For example, a decimal equivalent input of 23, gives a result of $8 = \text{/23/}_{15}$. From Table 5.1, a Karnaugh Map may be formed for each individual output bit, W, X, Y, and Z. Karnaugh Maps allow output variables to be simplified into logical equations, as functions of their inputs. The Karnaugh Maps for the Modulo 15 truth table are shown in Figure 5.6. The simplified output equations derived from the Karnaugh Maps are as follows:

Table 5.1 Modulo 15 Truth Table

Five Bit Result	Result Mod 15	Truth Table Outputs			
		W	X	Y	Z
0	0	0	0	0	0
1	1	0	0	0	1
2	2	0	0	1	0
3	3	0	0	1	1
4	4	0	1	0	0
5	5	0	1	0	1
6	6	0	1	1	0
7	7	0	1	1	1
8	8	1	0	0	0
9	9	1	0	0	1
10	10	1	0	1	0
11	11	1	0	1	1
12	12	1	1	0	0
13	13	1	1	0	1
14	14	1	1	1	0
15	0	0	0	0	0
16	1	0	0	0	1
17	2	0	0	1	0
18	3	0	0	1	1
19	4	0	1	0	0
20	5	0	1	0	1
21	6	0	1	1	0
22	7	0	1	1	1
23	8	1	0	0	0
24	9	1	0	0	1
25	10	1	0	1	0
26	11	1	0	1	1
27	12	1	1	0	0
28	13	1	1	0	1
29	14	1	1	1	0
30	0	0	0	0	0
31	1	0	0	0	1

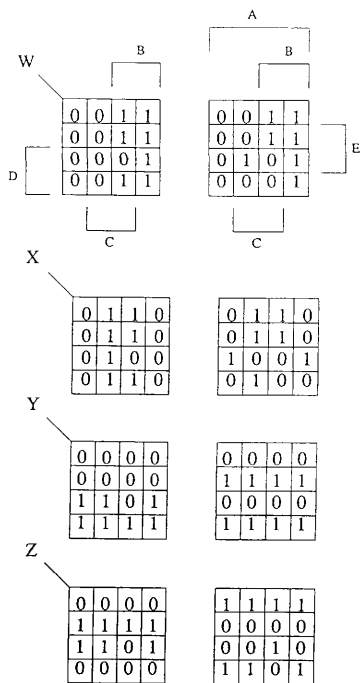


Figure 5.6 Modulo 15 Karnaugh Maps

$$W = \overline{BD} + \overline{BC} + \overline{ABDE} + \overline{ABCDE}$$

$$X = \overline{CD} + \overline{ABC} + \overline{ACDE} + \overline{ACE} + \overline{BCE}$$

$$Y = \overline{ADE} + \overline{ADE} + \overline{ABD} + \overline{ACD} + \overline{BDE} + \overline{CDE}$$

$$Z = \overline{ADE} + \overline{ADE} + \overline{ABE} + \overline{ABE} + \overline{ABCE} + \overline{ACE} + \overline{ABCDE}$$

These equations can be implemented with multi-input NAND gates and inverters. The hardware implementation of the Modulus 15 truth tables is shown in Figure 5.7. Once again, this is an all NAND realization even though plotter resolution does not allow for bubbles on the outputs of NAND gates. The circuitry on the left hand side is the lower truth table implementation, the circuitry on the right hand side is the upper truth table implementation (discussion in the following section). Similar truth tables and Karnaugh Maps can be formed for the remaining moduli (found in Appendix B). The results are given below:

Modulus 7:

$$W = 0$$

$$X = \overline{ACD} + \overline{ABCE} + \overline{BCDE} + \overline{ACD} + \overline{ABCE} + \overline{BCDE}$$

$$Y = \overline{ABDE} + \overline{ABCD} + \overline{ABDE} + \overline{ACDE} + \overline{ACDE} + \overline{ABCD} + \overline{ABDE} + \overline{ABCD} + \overline{ABDE} + \overline{ACDE}$$

$$Z = \overline{BCE} + \overline{BCE} + \overline{ABCE} + \overline{ABDE} + \overline{ABDE} + \overline{BCDE} + \overline{ABCDE}$$

Modulus 11:

$$W = \overline{ABCD} + \overline{ABCE} + \overline{ABCD} + \overline{ABCD} + \overline{ABCDE}$$

$$X = \overline{ABC} + \overline{ACDE} + \overline{ABCD} + \overline{ABCD} + \overline{ABCD} + \overline{ACDE}$$

$$Y = \overline{ADE} + \overline{ABD} + \overline{BCDE} + \overline{ADE} + \overline{ABD} + \overline{BCDE}$$

$$Z = \overline{ABE} + \overline{ABCE} + \overline{BCDE} + \overline{ABDE} + \overline{ABE} + \overline{ACDE} + \overline{ABCE}$$

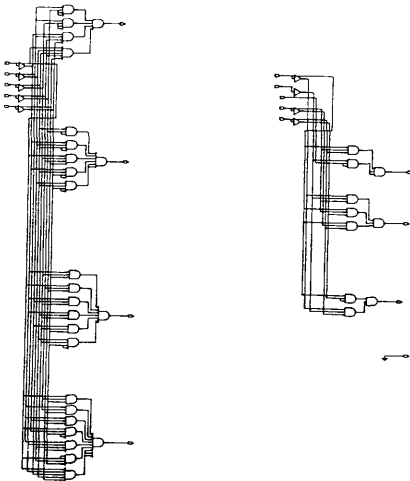


Figure 5.7 Modulo 15 Truth Tables

Modulus 13:

$$W = \overline{A}\overline{B}\overline{D}\overline{E} + \overline{A}\overline{B}C + \overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{E} + \overline{A}\overline{B}\overline{C}\overline{D}$$

$$X = \overline{A}\overline{B}C + \overline{A}\overline{C}\overline{D}\overline{E} + \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}\overline{C}\overline{D}\overline{E} + \overline{B}\overline{C}\overline{D}\overline{E}$$

$$Y = \overline{A}\overline{B}\overline{D} + \overline{A}\overline{D}\overline{E} + \overline{A}\overline{C}\overline{D} + \overline{B}\overline{D}\overline{E} + \overline{A}\overline{D}\overline{E} + \overline{A}B\overline{C}\overline{D}$$

$$Z = \overline{A}\overline{B}\overline{E} + \overline{A}\overline{C}\overline{E} + \overline{A}\overline{B}\overline{E} + \overline{A}\overline{C}\overline{D}\overline{E} + \overline{A}B\overline{C}\overline{E} + \overline{A}B\overline{D}\overline{E} + \overline{A}\overline{B}\overline{C}\overline{D}\overline{E}$$

5.2.4 Upper Truth Table Modulo m

The upper truth table is developed in a manner similar to the lower truth table. Each upper truth table has five inputs, and four outputs, but the significance of the input bit positions are greater than those of the lower truth tables. From Figure 5.2, it is seen that the Modified Braun Array structure has eight outputs, five of them connected to the lower truth table as inputs, and three of them connected to the upper truth table. The reader should note that only the three lower order bits are necessary for the MAC, but it will be shown shortly that the input translational problem will have a necessity for the uppermost two bits. The truth tables and Karnaugh Maps are very similar to those previously presented, except for the relative magnitude of each input bit position. Previously a binary input of 00011 represented the decimal value of three. For the upper truth tables this is not the case, a similar input in this case yields a decimal value of ninety-six $(00011XXXXX)_2$, remembering that the least significant bit is really the sixth significant position

from the Modified Braun Array. The results of the Karnaugh Maps for the various moduli are presented below:

Modulus 7:

$$W = 0$$

$$X = \bar{D}\bar{E} + BD + \bar{C}E$$

$$Y = \bar{C}\bar{D} + \bar{C}\bar{E} + AD$$

$$Z = AD + ADE + ACD$$

Modulus 11:

$$W = \bar{A}\bar{C}E + \bar{A}\bar{C}\bar{D}$$

$$X = C + \bar{A}\bar{D} + \bar{A}\bar{E}$$

$$Y = \bar{A}\bar{D}E + \bar{A}\bar{C}\bar{D} + \bar{A}BD + ADE + A\bar{D}\bar{E}$$

$$Z = \bar{A}E + \bar{A}\bar{D}E + B + CE$$

Modulus 13:

$$W = \bar{A}\bar{D}E + \bar{C}\bar{E} + B + \bar{A}\bar{E}$$

$$X = \bar{A}\bar{E} + \bar{A}\bar{D}E + B + \bar{A}\bar{C}E + \bar{A}\bar{C}\bar{D}$$

$$Y = \bar{A}\bar{E} + \bar{B}\bar{C}\bar{D} + \bar{C}\bar{D}E + \bar{C}\bar{E}$$

$$Z = \bar{A}\bar{D} + \bar{C}\bar{D}E + \bar{A}\bar{B}\bar{D}E$$

Modulus 15:

$$W = ADE + \bar{B}C$$

$$X = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{D}E + \bar{A}\bar{D}\bar{E}$$

$$Y = \bar{A}\bar{E} + \bar{A}\bar{B}\bar{E}$$

$$Z = 0$$

The hardware implementation of the modulo 15 upper truth table was shown previously in Figure 5.7.

5.2.5 Four-Bit Binary Adder

The binary adder is used to add the two four-bit outputs from the upper and lower truth tables. A serial ripple carry adder is used in this case, which is simply a one-dimensional array of full adder cells. To form the four-bit binary adder, four of the cells in Figure 5.4 are repeated, and connected such that the carry out from lower bit positions becomes the carry in for higher bit positions. The complete four-bit binary adder is shown in Figure 5.8.

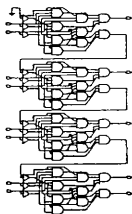


Figure 5.8 Four-Bit Binary Adder

5.3 Input Translation

Input translation must occur before the various operands may enter the MAC computing array. Globally, the matrix multiplication chip is composed of three major

parts, the input translator, the MAC array, and the output translator. This section will discuss input translation.

The goal of the input translational portion is to convert an eight-bit input operand, in signed magnitude format, to the proper residue system representation. The method used for input translation uses several functional blocks common to the Multiply and Add Cell, reducing the design time. It was found that if the input operand is a negative number, $/x/_{m_i} = /M-b/_{m_i}$ must be calculated (where $M = 7*11*13*15*16$, and b is the absolute value of the input operand) to obtain the correct residue representation, rather than $/x/_{m_i}$ when the input operand is positive. Example 5.2 gives several examples of the input conversion process.

Example 5.2:

$$m_i = \{ 7, 11, 13, 15, 16 \}$$

$$M = 7*11*13*15*16 = 240240$$

Given the following input operands:

$$A = (10011010)_2 = (-26)_{10}$$

$$B = (01101001)_2 = (+105)_{10}$$

$$C = (10001001)_2 = (-9)_{10}$$

the residue representations are found as follows:

$$/A/_{m_i} = /240240 - 26/_{m_i} \quad i = 1, 2, 3, 4, 5$$

$$/B/_{m_i} = \quad \quad \quad /105/_{m_i} \quad i = 1, 2, 3, 4, 5$$

$$/C/_{m_i} = /240240 - 9/_{m_i} \quad i = 1, 2, 3, 4, 5$$

The residue representations are:


```

A = { 2, 7, 0, 4, 6 }
B = { 0, 6, 1, 0, 9 }
C = { 5, 2, 4, 6, 7 }

```

The block diagram for the input translation of an eight-bit signed magnitude input operand is shown in Figure 5.9. The various hardware aspects of input translation will now be examined in detail.

5.3.1 Input Operand Adjustment

The input operand adjustment portion of the input translational process examines the most significant bit of the input, if this value is a logical "1", then the input is negative, and the absolute value of the input must be subtracted from M ($M=240240$). If the most significant bit of the input is a logical "0", then the input is positive, and the input value should be transferred through the input operand adjustment block. It should be noted that eighteen bits are required to express M in binary form. This implies that eighteen-bit addition will have to be performed when the input operand is negative ($M - \text{absolute value of } x$). A great simplification can be made at this point due to the limitation of the input range. It should be noted that for an eight-bit signed magnitude number, the largest positive or negative number is one hundred and twenty-seven. When this value is subtracted from 240240 in binary form, only the ten least significant

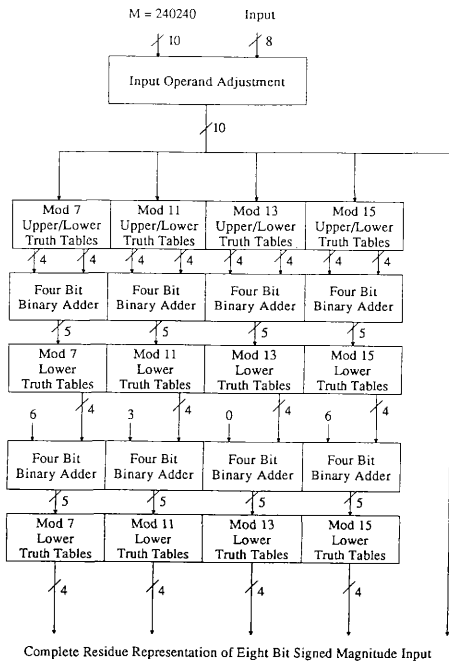


Figure 5.9 Input Translation Functional Configuration

bit positions are altered. Therefore, it is useless to carry out the full eighteen-bit addition process, when only the lowest ten bit positions have the possibility of changing. In other words, the top eight bit positions remain constant throughout the addition process. The necessary manipulation of the upper eight bit positions will be examined in the following section.

The hardware implementation to accomplish the input operand adjustment portion of the input translation process is shown in Figure 5.10. It is fairly simple in design, a ten-bit binary adder, with a row of exclusive-or gates at the input. It should be stated that whenever the input is positive, the circuitry allows the input operand to pass through unchanged, but when negative, the absolute value of the input operand is subtracted from the lower ten bit positions of 240240. From the above simplification, the result of the subtraction yields a ten-bit number. The rest of the input translation of Figure 5.9 will now be examined.

5.3.2 Residue Digit Generation

The tight dynamic range of the eight-bit input, as compared to the large value of M , allows a simplification in the upper truth tables of the individual Karnaugh maps. Since the input number subtracted from M must be less than 127, the dynamic range of resultant is a ten-bit

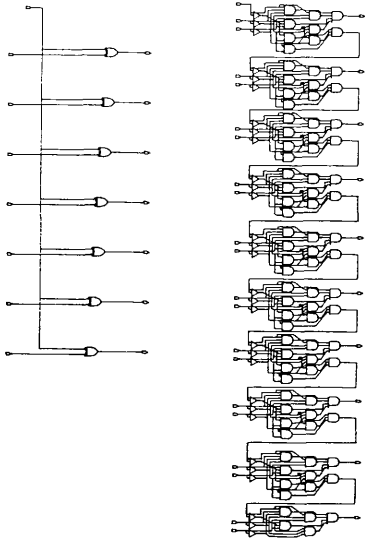


Figure 5.10 Input Operand Adjustment

number between 01111XXXXX and 10011XXXXX. Remembering that the Multiply and Add Cell only requires values between 00000 and 00111, a great simplification is made in the upper truth tables for all of the moduli. The reason for this simplification was not explained when the upper truth tables were introduced earlier in this chapter.

As was mentioned previously in the operand adjustment section, the upper eight bits of the eighteen-bit representation of 240240 are not involved in the subtraction process in the operand adjustment portion of Figure 5.9. However, they must be taken into consideration to produce a correct result. The upper eight bits of M corresponds to a decimal value of 239616. It was found that if $/239616/m_i$ was added to the result of the previous steps for each corresponding moduli, that the resulting residue representation was correct. The calculation of the values to be added to each moduli are calculated below:

$$/239616/_7 = 6$$

$$/239616/_11 = 3$$

$$/239616/_13 = 0$$

$$/239616/_15 = 6$$

$$/239616/_16 = 0$$

These values are shown to be added in at the appropriate step in Figure 5.9. It is expected that an engineer will spend more time designing a residue system than he or she would a more conventional system. It should be noted that

there are quite a few common blocks that are repetitively used in this proposed design. More importantly, the common building blocks may be optimized separately, then combined in an orderly fashion. This methodology both improves circuit performance and saves silicon area.

5.4 Output Translation

The Chinese Remainder Theorem was examined first when considering an appropriate method of converting from residue representation back to a binary or fixed weight representation. Implementation of the Chinese Remainder Theorem requires addition modulo M , which in this case means several addition or subtractions with word lengths greater than eighteen. Another method of conversion from residue to a weighted system is called the mixed-radix conversion process [2]. It was found that the mixed-radix conversion process has two main advantages over an implementation of the Chinese Remainder Theorem. First, a significantly larger amount of hardware is required to directly implement the Chinese Remainder Theorem. Second, the Chinese Remainder Theorem does not allow any type of intermediate magnitude comparison or sign determination. It will be shown that at an intermediate step of the mixed-radix conversion process, enough information exists to compare the magnitude of two residue numbers, or to determine the sign of a residue number. Using the Chinese

Remainder Theorem, it is impossible to obtain any of the above mentioned information without completely converting to the binary representation.

The mixed radix conversion process is governed by the following equation:

$$x = A_5(m_1m_2m_3m_4) + A_4(m_1m_2m_3) + A_3(m_1m_2) + A_2(m_1) + A_1$$

where $\{ m_1, m_2, m_3, m_4, m_5 \} = \{ 16, 15, 13, 11, 7 \}$, and x is the result of converting a number from residue to binary representation. It should be noted that the mixed-radix system is a weighted system, hence magnitude and sign determination is relatively easy. The mixed radix representation of x is given by $\langle A_1, A_2, A_3, A_4, A_5 \rangle$. The A_i values may be determined by the following:

$$A_1 = r_1$$

$$A_2 = \text{floor}((x - r_1)/m_1)/m_2$$

$$A_3 = \text{floor}((A_2 - r_2)/m_2)/m_3$$

$$A_4 = \text{floor}((A_3 - r_3)/m_3)/m_4$$

$$A_5 = \text{floor}((A_4 - r_4)/m_4)/m_5$$

Residue division is not really occurring even though the above equations imply it is necessary. Multiplying by the multiplicative inverse is the same as division, which will be the approach taken. The functional diagram of the mixed radix conversion process is shown in Figure 5.11. The individual portions of Figure 5.11 will now be examined. Also included in this section on output translation will be a comprehensive example unifying the individual processes.

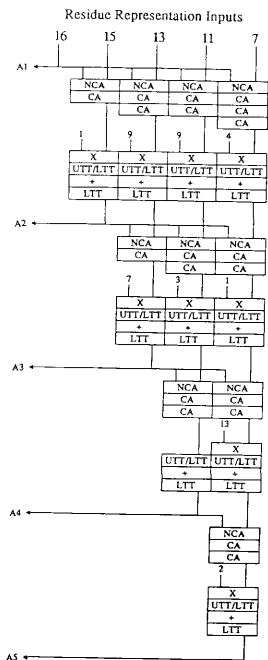


Figure 5.11 Mixed Radix Coefficient Determination Output Translation

5.4.1 Controlled Addition/Subtraction

Two of the blocks in Figure 5.11 are called "CA" and "NCA". These notations stand for conditional and non-conditional addition. The non-conditional addition block has two four-bit numbers as inputs and a five-bit number as an output. The block subtracts the one four-bit input from the other. This subtraction performs the $x-r_1$ portion of the mixed-radix process. Since a residue representation must not be negative, the subsequent conditional adder blocks sample the most significant bit of the five-bit input, if this is a zero, then no computation occurs. If the most significant bit is a one, then the input is negative, and thus the modulus (m_1) for the appropriate digit of the residue representation (r_1) must be added to the negative number. This non-conditional addition process must be repeated until it is assured that the residue representation at each of the stages in determining the A_1 contains only non-negative (or zero) residue digits. The hardware implementation of both the conditional and non-conditional adders are shown in Figure 5.12 and 5.13.

5.4.2 Multiplication by Inverses

After the subtraction of the most significant remaining residue digits, and the conditional addition processes are complete, multiplication of the residue

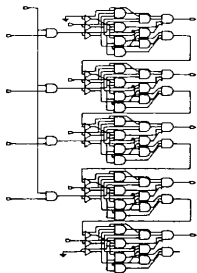


Figure 5.12 Conditional Adder

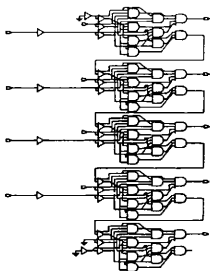


Figure 5.13 Non-Conditional Adder

representations by the respective multiplicative inverse occurs. The multiplicative inverses are shown in Table 5.2.

Table 5.2 Multiplicative Inverses

	Modulus			
	15	13	11	7
$1 16$	1	9	9	4
$1 15$	-	7	3	1
$1 13$	-	-	17	13
$1 11$	-	-	-	2

For example, $1|16/7 = 4$. All multiplicative inverses can be represented in four binary bits except for 17, which requires five bits. Multiplication by 17, when using binary arithmetic, is simply the digits of the multiplicand repeated twice. For example, $(17)(9) = (10011001)_2$, and $(17)(13) = (11011101)_2$. Thus, the multiplication of all residue digits by their respective multiplicative inverses will only require a Braun array capable of four-bit multiplication. The desired Braun array has been previously designed, with the exception that the top row of full adders needed in the MAC has been deleted, and is shown in Figure 5.14. The blocks in Figure 5.11 labeled "UTT/LTT", "+", "LTT", are the same blocks that were presented in the MAC section. This sequence of blocks simply converts the larger input number into residue representation for each of the moduli. The following

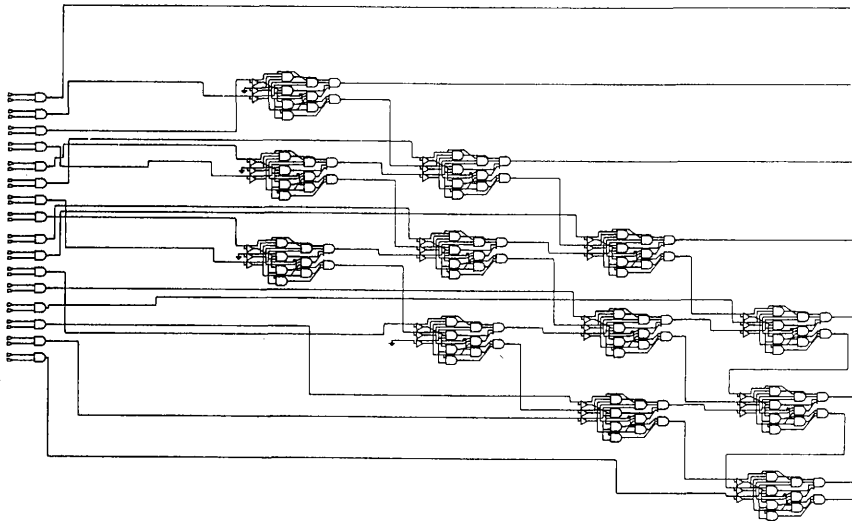


Figure 5.14 Four Bit Braun Array

example should help clarify some of the items presented in this section.

Example 5.3:

Given $m_1=16$, $m_2=15$, $m_3=13$, $m_4=11$, and $m_5=7$, the mixed radix coefficients (\bar{A}_i) can be computed for a residue representation of $x = (2, 3, 4, 2, 6)$ as follows:

(reference Figure 5.11 as necessary)

Modulus 16 15 13 11 7

$A_1=2$

$$\begin{array}{r}
 (-2) \quad 2 \quad 3 \quad 4 \quad 2 \quad 6 \\
 \quad \quad \quad \underline{-2 \quad -2 \quad -2 \quad -2 \quad -2} \\
 \quad \quad \quad 1 \quad 2 \quad 0 \quad 4 \quad 4 \\
 \\
 x(/1|16/m_1) \quad \quad \quad \underline{x1 \quad x9 \quad x9 \quad x4} \\
 (/r_1/m_1) \quad \quad \quad 1 \quad 18 \quad 0 \quad 16 \\
 \quad \quad \quad \underline{1 \quad 5 \quad 0 \quad 2}
 \end{array}$$

$A_2=1$

$$\begin{array}{r}
 (-1) \quad 1 \quad 5 \quad 0 \quad 2 \\
 \quad \quad \quad \underline{-1 \quad -1 \quad -1} \\
 \quad \quad \quad 4 \quad -1 \quad 1 \\
 (+m_1) \quad \quad \quad \underline{0 \quad +11 \quad 0} \\
 \quad \quad \quad 4 \quad 10 \quad 1 \\
 \\
 x(/1|15/m_1) \quad \quad \quad \underline{x7 \quad x3 \quad x1} \\
 (/r_1/m_1) \quad \quad \quad 28 \quad 30 \quad 1 \\
 \quad \quad \quad \underline{2 \quad 8 \quad 1}
 \end{array}$$

$A_3=2$

$$\begin{array}{r}
 (-2) \quad 2 \quad 8 \quad 1 \\
 \quad \quad \quad \underline{-2 \quad -2} \\
 \quad \quad \quad 6 \quad -1 \\
 (+m_1) \quad \quad \quad \underline{0 \quad +7} \\
 \quad \quad \quad 6 \quad 6 \\
 \\
 x(/1|15/m_1) \quad \quad \quad \underline{x17 \quad x13} \\
 (/r_1/m_1) \quad \quad \quad 102 \quad 78 \\
 \quad \quad \quad \underline{3 \quad 1}
 \end{array}$$

$A_4=3$

$$\begin{array}{r}
 (-3) \\
 (+m_1) \\
 \\
 x(/1111/m_1) \\
 (/r_1/m_1)
 \end{array}
 \begin{array}{r}
 3 \quad 1 \\
 \underline{-3} \\
 -2 \\
 \underline{+7} \\
 5 \\
 \\
 \underline{x2} \\
 10 \\
 3
 \end{array}$$

A₅=3

In summary, $A_1=2$, $A_2=1$, $A_3=2$, $A_4=3$, $A_5=3$.

Checking the Result:

$$\begin{aligned}
 x &= 3*16*15*13*11 + 2*16*15*13 + 2*16*15 + 1*16 + 2 \\
 &= 112818
 \end{aligned}$$

$x = 112818 = \{ 2, 3, 4, 2, 6 \}$ in residue representation

The resulting coefficients are exactly correct.

5.4.3 Correct Sign Determination

The conversion is almost complete. The multiplication and addition of the moduli and the mixed-radix coefficients (A_i) can begin as soon as the first A_1 is determined. The multiplication and addition begins and occurs simultaneously with the determination of the latter mixed-radix coefficients (A_3 , A_4 , A_5). Figure 5.15 shows the functional description of the addition and multiplication of the appropriate moduli and mixed-radix coefficients. The hardware implementations of the individual blocks are shown in Appendix C of this document. The last three blocks of Figure 5.15 will be discussed now as they are important to a residue type design. It should be noted

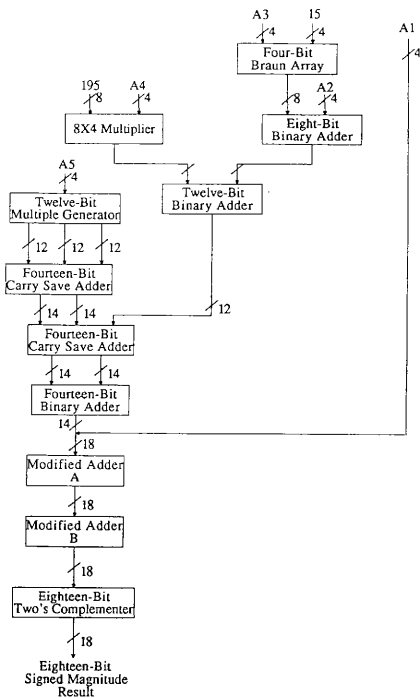


Figure 5.15 Multiplication and Addition of the Mixed-Radix Coefficients

that the output before the Modified Adder A block will be an eighteen-bit number between 0 and 240240. If the number is in the range [0,120119], then the result is a positive number and is correct in present form. If the number is in the range [120120,240239], then the number is a negative number, and 240240 must be subtracted from it. The determination of sign must occur in two distinct stages. First $M/2=120120$ must be subtracted from the eighteen-bit result. If this number is negative, then the original number was in the range of [0,120119], which was a positive number. In the very next stage it is necessary to add 120120 back to this number, because it needs to be a positive number as it was originally. If the subtraction of $M/2$ is not a negative number, then the original number was in the range [120120,240239], and must be a negative number. In the very next stage it is necessary to subtract another 120120 from this number, so that a total of 240240 has been subtracted from it. At this point, the result is the completely correct result in two's complement representation. If the most significant bit is a "one", then the lower seventeen bits are complemented, and the result will be in the desired signed magnitude form. If the most significant bit is a "zero", then the result is in correct signed magnitude form, and should bypass the complementer stage. The hardware implementation of the Modified Adder A and B, as well as the two's complementer, are shown in Appendix C of this thesis.

This concludes the introduction of the design of this research. The following chapter presents the simulation of this design, as well as a performance comparison to a more conventional approach.

CHAPTER VI

SIMULATION RESULTS AND COMPARISON

The purpose of this chapter is to present the simulation results, as well as compare the timing and area constraints of the residue design to a more conventional binary approach. Mentor Graphics Neted and Quicksim were used for schematic capture and logic simulation.

6.1 Simulation Development

Chapter IV presented a detailed simulation of the matrix multiplication algorithm. Rather than simulate the design as a whole, the three fundamental portions were simulated. Specifically, the Multiply and Add Cell for all moduli, and the Input and Output translational portions were simulated. Simulation of the above portions, including the matrix multiplication algorithm simulation, will give all the required timing information. It was found that the schematic capture of the full design would be a very large task, and would not be beneficial to this research.

Before any simulation or comparison begins, it is appropriate to present the primitive component timing and area models [17]. These parameters will be used consistently throughout this chapter for comparison and

simulation purposes. Table 6.1 gives the proportional delay time of an individual gate, as well as the proportional area each component occupies. Actual timing and area information is strongly dependent on semiconductor processing. To obtain actual timing or area information, the values must be multiplied by a scaling factor. The scaling factor for the gate delays is T_g , and typically ranges from .25 ns to 1 ns. The area scaling factor A_g is typically around 25X25 square microns, with a strong dependence on the lithographic linewidth of the process used for fabrication. The simulation results from Quicksim will now be summarized. The more detailed simulation output in raw data form, as well as graphical form, can be found in Appendix D of this thesis.

Table 6.1 Primitive Component Models

Component	Delay (T_g)	Area (A_g)
n-input NAND Gate ($n < 10$)	1	1
Inverter	1	1
n-input AND Gate ($n < 10$)	2	2
n-input OR Gate ($n < 10$)	2	2
XOR Gate	2	3
One-Bit Full Adder	2	10
D Flip-Flop	3	5

6.2 Simulation Results

This section will present the simulation results of

each of the three major portions of this design. The most important portion of the simulation is the MAC. The timing information of the MAC will determine the overall system clock speed. The overall clock speed will be crucial when the comparison is made later in the chapter. A portion of this section also deals with the global timing information of the design.

6.2.1 MAC Simulation

The more detailed simulation output for the MAC can be found in Table A.1 and Figures A.11 through A.15 in the Appendix D. Table 6.2 shown in this section is to summarize the MAC simulation results. Since each MAC contains five independent residue multiply and add cells, there are five different sub-tables (one for each modulus) shown in Table 6.2. Also, note that numbers shown in the table have been converted to decimal for convenience. The worst case simulated delay from Table 6.2 is $23 T_g$. The following calculations show how the worst case "predicted" gate delay for the MAC is obtained:

Modified Braun Array:

$$\begin{aligned} \text{Delay} &= 1 \text{ AND Gate} + 7 \text{ Full Adders} \\ &= (1)(2) + (7)(2) \\ &= 16 T_g \end{aligned}$$

Upper/Lower Truth Table:

$$\begin{aligned} \text{Delay} &= 1 \text{ inverter} + 2 \text{ nand gates} \\ &= (1)(1) + (2)(1) \end{aligned}$$

Table 6.2 MAC Simulation Results

Modulo 7

	Input A	Input B	Input C	Result	Delay
Trial #1	4	3	5	3	23
Trial #2	6	3	5	2	13
Trial #3	3	6	1	5	23
Trial #4	3	4	3	1	14

Modulo 11

	Input A	Input B	Input C	Result	Delay
Trial #1	8	9	4	10	21
Trial #2	3	7	9	8	21
Trial #3	4	8	2	1	22
Trial #4	1	3	10	2	15

Modulo 13

	Input A	Input B	Input C	Result	Delay
Trial #1	8	7	10	1	23
Trial #2	4	9	2	12	19
Trial #3	12	4	8	4	19
Trial #4	2	11	5	1	16

Modulo 15

	Input A	Input B	Input C	Result	Delay
Trial #1	13	10	3	13	23
Trial #2	3	9	12	9	18
Trial #3	8	9	4	1	16
Trial #4	9	3	7	4	20

Modulo 16

	Input A	Input B	Input C	Result	Delay
Trial #1	1	4	9	13	9
Trial #2	3	5	2	1	9
Trial #3	4	12	10	10	8
Trial #4	1	11	12	7	6

$$= 3 T_g$$

Four-Bit Binary Adder:

$$\begin{aligned} \text{Delay} &= 4 \text{ Full Adders} \\ &= (4)(2) \\ &= 8 T_g \end{aligned}$$

Lower Truth Table:

$$\begin{aligned} \text{Delay} &= 1 \text{ inverter} + 2 \text{ NAND gates} \\ &= (1)(1) + (2)(1) \\ &= 3 T_g \end{aligned}$$

Summing the above gate delays gives the MAC worst case gate delay, which is $16+3+8+3 = 30 T_g$. Note that the simulated worst case delay should always be less than or equal to the worst case predicted delay, as is the case in Table 6.2.

6.2.2 Input Simulation

The results of the input translation process are summarized in Table 6.3. The more detailed simulation output can be found in Table A.2 and Figures A.16 through A.20. The reader should notice that some of the delays are larger than 30. This means that the input translation process must be broken up into two stages since the pipeline segment time (governed by the MAC) is the maximum time any one segment should take to execute. Examining the worst case delays for the input translation process will help determine where the latches will need to be placed, ensuring that no operation in the input translational process exceeds the maximum of $30 T_g$. The worst case delays for the input translation process are given as

Table 6.3 Input Translation Simulation Results

Modulo 7

	Input	Result	Delay
Trial #1	-44	5	25
Trial #2	+77	0	26
Trial #3	-114	5	23
Trial #4	+57	1	15

Modulo 11

	Input	Result	Delay
Trial #1	-82	6	29
Trial #2	+60	5	21
Trial #3	-41	3	28
Trial #4	+114	4	23

Modulo 13

	Input	Result	Delay
Trial #1	-36	3	22
Trial #2	+109	5	25
Trial #3	-51	1	24
Trial #4	+71	6	20

Modulo 15

	Input	Result	Delay
Trial #1	-115	5	35
Trial #2	+36	6	15
Trial #3	-43	2	25
Trial #4	+28	13	19

Modulo 16

	Input	Result	Delay
Trial #1	-51	13	7
Trial #2	+49	1	4
Trial #3	-87	9	6
Trial #4	+104	8	2

follows:

Input Operand Adjustment:

$$\begin{aligned} \text{delay} &= 1 \text{ XOR Gate} + 10 \text{ Full Adder} \\ &= (1)(2) + (10)(2) \\ &= 22 T_g \end{aligned}$$

Upper/Lower Truth Tables:

$$\begin{aligned} \text{delay} &= 1 \text{ Inverter} + 2 \text{ NAND Gates} \\ &= (1)(1) + (2)(1) \\ &= 3 T_g \end{aligned}$$

Four-Bit Binary Adder:

$$\begin{aligned} \text{delay} &= 4 \text{ Full Adders} \\ &= (4)(2) \\ &= 8 T_g \end{aligned}$$

Lower Truth Table:

$$\begin{aligned} \text{delay} &= 1 \text{ Inverters} + 2 \text{ NAND Gates} \\ &= (1)(1) + (2)(1) \\ &= 3 T_g \end{aligned}$$

Four-Bit Binary Adder:

$$\text{delay} = 8 T_g$$

Lower Truth Table: (Same)

This gives a worst case gate delay of $47 T_g$ for the input translation process. If a row of latches is placed between the Upper/Lower Truth Tables block and the first Four-Bit Binary Adder block, the gate delays for the two different stages are $25 T_g$ and $22 T_g$ for the first and second stages, respectively.

6.2.3 Output Simulation

The simulation results of the output translational process are summarized in Table 6.4. Again, the

Table 6.4 Output Translation Simulation Results

	Mod 16 Input	Mod 15 Input	Mod 13 Input	Mod 11 Input	Mod 7 Input	Result	Delay
Trial #1	7	13	2	4	3	-104297	173
Trial #2	13	13	4	8	6	107533	125
Trial #3	12	5	10	5	4	12380	118
Trial #4	6	2	3	7	2	-1258	160

more detailed simulation output can be found in Table A.3 in the Appendix D. There is no timing diagram for the output portion because the large number of signals would not fit with clarity on one page. Output translation is the most complex of the operations presented thus far. There are many design options, depending on the complexity of circuitry used in the addition and multiplication of the mixed-radix coefficients. This design uses only simple binary adders and multipliers. Much faster methods are available, but they consume a much larger area. Faster methods, as will be proved shortly, are not beneficial. Output translation is a pipelined process. As soon as the first matrix multiplication is out of the MAC array, the next multiplication may begin. In light of this, one pair of matrices are being multiplied together at the same time the result of the prior pair of matrices is going through output translation. Thus, for the above stated reason, the time required for output translation is negligible after the first matrix multiplication. It can be shown that the output translational process must be broken into fifteen

stages in order to operate at the same clock cycle as the rest of the matrix multiplier. From Figure 5.11 it takes eight clock cycles to generate the mixed-radix coefficient A5. The remaining seven ($8+7 = 15$) are a result of the Twelve-Bit Multiplication, the Fourteen-Bit Carry Save Adders, the Modified Adders A and B, and the Two's Complementer (all shown in Figure 5.15). The resulting output matrix values will most likely be transferred off of the matrix multiplier chip at the same clock rate as the system bus. Also, it is very unlikely that the system bus will be operating at the same speed as the matrix multiplier. Nonetheless, for comparison purposes, this research will use a worst case of 15 clock cycles to convert from residue representation to signed magnitude format. The following section examines the the global timing information from the above simulation results, as well as the results of the matrix multiplication algorithm simulation.

6.2.4 Global Considerations

Although a comprehensive global simulation has not been performed, there exists enough information to precisely predict the overall performance of the design. As derived in Chapter IV, it takes seventeen clock cycles to completely multiply two matrices together. From the

previous section, it takes 2 clock cycles for input translation, and 15 clock cycles for output translation. Therefore, it takes 34 clock cycles to complete the first matrix multiplication. It takes 17 additional clock cycles for each successive matrix multiplication. The following equation gives the total processing time (T_p) in gate delays for a certain number of successive matrix multiplications (N):

$$T_p = t_A + (N-1)(t_{mult})(t_B) \quad N \geq 1$$

$$\text{where } t_A = (17 \text{ cycles} + 17 \text{ cycles})t_B$$

$$\begin{aligned} t_B &= t_{seq} + t_{diff} \\ &= 30 + 3 \\ &= 33 T_g/\text{cycle} \end{aligned}$$

$$t_{mult} = \text{clock cycles to complete MAC array portion}$$

Simplifying:

$$\begin{aligned} T_p &= 1122 + (N-1)(17)(33) \\ &= 1122 + (N)(561) - 561 \\ &= 561 + (N)(561) \\ &= (N+1)(561) T_g \end{aligned}$$

For example, the total processing time to multiply two pairs of matrices in succession is $1683 T_g$. The following section calculates the area such a design occupies on silicon.

6.3 Residue Design Area Calculations

This section will briefly show the calculations made in determining the area this design will occupy on silicon. Also in this section will be a discussion on global area

issues.

6.3.1 MAC Area

The MAC area calculations are shown below, it may be necessary for the reader to refer back to figures in the previous chapter dealing specifically with the MAC. The calculations for one MAC are as follows:

Modified Braun Array:

$$\begin{aligned} \text{Area} &= 16 \text{ Full Adders} + 16 \text{ And Gates} \\ &= (16)(10) + (16)(2) \\ &= 192 A_g \\ &\quad (\text{6 Less Full Adders for Mod 16 Braun Array Only}) \end{aligned}$$

Lower and Upper Truth Tables:

Modulo 7:

Upper

$$\begin{aligned} \text{Area} &= 12 \text{ NAND Gates} + 5 \text{ Inverters} \\ &= (12)(1) + (5)(1) \\ &= 17 A_g \end{aligned}$$

Lower

$$\begin{aligned} \text{Area} &= 27 \text{ NAND Gates} + 5 \text{ Inverters} \\ &= (27)(1) + (5)(1) \\ &= 32 A_g \end{aligned}$$

Modulo 11:

Upper

$$\begin{aligned} \text{Area} &= 18 \text{ NAND Gates} + 5 \text{ Inverters} \\ &= (18)(1) + (5)(1) \\ &= 23 A_g \end{aligned}$$

Lower

$$\begin{aligned} \text{Area} &= 28 \text{ NAND Gates} + 5 \text{ Inverters} \\ &= (28)(1) + (5)(1) \\ &= 33 A_g \end{aligned}$$

Modulo 13:

Upper

$$\begin{aligned} \text{Area} &= 20 \text{ NAND Gates} + 5 \text{ Inverters} \\ &= (20)(1) + (5)(1) \\ &= 25 A_g \end{aligned}$$

Lower

$$\begin{aligned} \text{Area} &= 28 \text{ NAND Gates} + 5 \text{ Inverters} \\ &= (28)(1) + (5)(1) \\ &= 33 A_g \end{aligned}$$

Modulo 15

Upper

$$\begin{aligned}
 \text{Area} &= 10 \text{ NAND Gates} + 5 \text{ Inverters} \\
 &= (10)(1) + (5)(1) \\
 &= 15 A_g \\
 \text{Lower} \\
 \text{Area} &= 26 \text{ NAND Gates} + 5 \text{ Inverters} \\
 &= (26)(1) + (5)(1) \\
 &= 31 A_g
 \end{aligned}$$

$$\begin{aligned}
 \text{Four-Bit Binary Adder:} \\
 \text{Area} &= 4 \text{ Full Adders} \\
 &= (4)(10) \\
 &= 40 A_g
 \end{aligned}$$

$$\begin{aligned}
 \text{TOTAL MAC AREA} &= 5(192) + 80 + 2(129) + 4(40) - 6(10) \\
 &= 1398 A_g
 \end{aligned}$$

Thus the total Multiply and Add Cell Area is 1398 A_g .

6.3.2 Input Translation Area

The input translation area calculations are as follows:

Input Operand Adjustment:

$$\begin{aligned}
 \text{Area} &= 10 \text{ Full Adders} + 7 \text{ XOR Gates} \\
 &= (10)(10) + (7)(3) \\
 &= 121 A_g
 \end{aligned}$$

Upper and Lower Truth Tables:
(Same as Above)

Four-Bit Binary Adder:
(Same as Above)

$$\begin{aligned}
 \text{TOTAL AREA} &= 1(121) + 80 + 3(129) + 8(40) \\
 &= 908 A_g
 \end{aligned}$$

Thus the total Input Translation Area is 908 A_g .

6.3.3 Output Translation Area

The output translation area calculations are as

follows:

NCA (Non-Conditional Adder):

$$\begin{aligned} \text{Area} &= 5 \text{ Full Adders} + 4 \text{ Inverters} \\ &= (5)(10) + (4)(1) \\ &= 54 A_g \end{aligned}$$

CA (Conditional Adder):

$$\begin{aligned} \text{Area} &= 5 \text{ Full Adders} + 4 \text{ AND Gates} \\ &= (5)(10) + (4)(2) \\ &= 58 A_g \end{aligned}$$

Four-Bit Braun Array:

$$\begin{aligned} \text{Area} &= 12 \text{ Full Adders} + 16 \text{ AND Gates} \\ &= (12)(10) + (16)(2) \\ &= 152 A_g \end{aligned}$$

Truth Tables:

(Same as Above)

Four-Bit Binary Adder:

(Same as Above)

Eight-Bit Binary Adder:

$$\begin{aligned} \text{Area} &= 8 \text{ Full Adders} \\ &= (8)(10) \\ &= 80 A_g \end{aligned}$$

Eight by Four Multiplier:

$$\begin{aligned} \text{Area} &= 32 \text{ Full Adders} + 32 \text{ And Gates} \\ &= (32)(10) + (32)(2) \\ &= 384 A_g \end{aligned}$$

Twelve-Bit Binary Adder:

$$\begin{aligned} \text{Area} &= 12 \text{ Full Adders} \\ &= (12)(10) \\ &= 120 A_g \end{aligned}$$

Multiple Generator:

$$\begin{aligned} \text{Area} &= 36 \text{ And Gates} \\ &= (36)(2) \\ &= 72 A_g \end{aligned}$$

Fourteen-Bit Carry Save Adder:

$$\begin{aligned} \text{Area} &= 14 \text{ Full Adders} \\ &= (14)(10) \\ &= 140 A_g \end{aligned}$$

Modified Binary Adder A:

$$\begin{aligned} \text{Area} &= 18 \text{ Full Adders} \\ &= (18)(10) \\ &= 180 A_g \end{aligned}$$

Modified Binary Adder B:

$$\begin{aligned} \text{Area} &= 36 \text{ Inverters} + 54 \text{ NAND Gates} + 18 \text{ Full} \\ &\quad \text{Adders} \\ &= (36)(1) + (54)(1) + (18)(10) \\ &= 270 A_g \end{aligned}$$

Twos Complementer:

$$\begin{aligned} \text{Area} &= 17 \text{ AND Gates} + 17 \text{ OR Gates} + 17 \text{ XOR Gates} \\ &= (17)(2) + (17)(2) + (17)(3) \\ &= 119 A_g \end{aligned}$$

TOTAL OUTPUT

$$\begin{aligned} \text{TRANSLATION AREA} &= 10(54) + 19(58) + 6(152) + 850 + 400 \\ &\quad + 80 + 404 + 120 + 72 + 2(140) \\ &\quad + 140 + 180 + 270 + 119 \\ &= 5469 A_g \end{aligned}$$

Thus the total Output Translation Area is 5469 A_g .

6.3.4 Global Considerations

There are several global options to be considered when implementing such a design, depending strongly upon the total amount of silicon area available. One may examine the output coefficient pattern, and note that each output port only has a non-zero coefficient every three clock cycles, such that three output ports could share an output translator. It is possible to build an array of data latches to accumulate the three resultant matrix operands at each clock cycle. In this method, the resulting operands wait their turn to enter the single output translator, and are then transferred off the of the matrix multiplier chip. Another scheme could be to implement three different output translators, with each translator transferring an output matrix coefficient off chip every

clock cycle. The latter scheme is much more efficient time-wise, but requires three output translators as opposed to one. The latter scheme also requires more component package pins.

The input translator is not as large, and does not require as much consideration. Since each input port only requires an input operand every three clock cycles, and there are 10 input ports, only four input translators are necessary.

The global area calculation, for both methods of output translation is shown below:

$$\begin{aligned} \text{TOTAL AREA} &= 4(908) + 25(1398) + (1 \text{ or } 3)(5469) \\ &= 44051 \text{ or } 54989 A_g \end{aligned}$$

The following calculation should give the reader an idea of how large such a circuit is on silicon.

$$(A_g = 25 \times 25 \text{ square microns})$$

$$\text{Chip Area} = 54989 \times (25 \times 10^{-6}) \times (25 \times 10^{-6}) = .344 \text{ cm}^2$$

6.4 Design Comparison

It is the purpose of this section to compare the residue design to a more conventional implementation of the matrix multiplication algorithm. As previously stated, the residue system is error free, in the sense that the output is correct for all possible eight-bit inputs. It was assumed that the input operand was an eight-bit integer value.

6.4.1 Comparison Structure

The approach of the comparison structure will be that of cascading a signed magnitude multiplier with a signed magnitude adder. Signed magnitude multipliers are the same as a Braun array, with the addition of one XOR Gate to determine the resulting sign. The timing and area calculations for the eight-bit multiplier are shown below:

Eight-Bit Multiplier:

$$\begin{aligned} \text{Delay} &= 12(2) + 2 \\ &= 26 T_g \end{aligned}$$

$$\begin{aligned} \text{Area} &= 49(2) + 42(10) \\ &= 518 A_g \end{aligned}$$

The calculations for a eighteen-bit signed magnitude adder are shown below [17]:

Eighteen-Bit Adder:

$$\begin{aligned} \text{Delay} &= 3(2) + 17(2) + 17(2) + 2 \\ &= 76 T_g \end{aligned}$$

$$\begin{aligned} \text{Area} &= 16(18) \\ &= 288 A_g \end{aligned}$$

Thus, the comparison structure has a total of $102 T_g$, and an area of $806 A_g$. As with the residue design, the following global area calculation will allow comparison to the residue design:

$$\text{Total Area} = 25(806) = 20150 A_g$$

The total processing time (T_p) to multiply N successive pairs of matrices together is given by the following equation:

$$\begin{aligned}
 T_p &= N(t_g) (17) & N \geq 1 \\
 &= N(t_{seg} + t_{diff}) (17) \\
 &= N(102 + 3) (17) \\
 &= 1785N T_g
 \end{aligned}$$

The total time required to multiply two pairs of matrices in succession is $3570 T_g$, the time required to multiply three pairs of matrices together in succession is $5355 T_g$.

6.4.2 Time and Area Comparisons

The global area required to implement the residue design is $44051 A_g$ or $54989 A_g$ depending on the output strategy used. The global area required for the conventional binary approach is $20150 A_g$. The residue design is 2.18 or 2.73 times larger than the binary approach.

The timing comparisons yield different results depending upon the assumed number of successive multiplications occurring. Table 6.5 shows the time required to execute a given number of matrix multiplications for the residue and conventional method, as well as the ratio of the two processing times.

Table 6.5 Processing Time Comparison

N	Residue Tp	Binary Tp	Tp Ratio
1	1122	1785	1.59
5	3366	8925	2.65
10	6171	17850	2.89
50	28611	89250	3.12
100	56661	178500	3.15
500	281061	892500	3.18

The total processing time T_p ratio converges to 3.18. In an ideal application, the matrix multiplier is constantly in operation. It must be remembered that the residue design is capable of multiplying over 1.5 million pairs of matrices in one second (assuming a relative gate delay of one nano-second). For this reason, 500 successive multiplications (as assumed in Table 6.5) is a very small number compared to actual hardware capabilities.

CHAPTER VII
CONCLUSION

This research applied the Residue Number System to a specific digital signal processing problem, that of matrix multiplication. The mathematical operations of addition and multiplication are simpler than residue division and sign determination. The matrix multiplication algorithm was an ideal candidate, since it only requires multiplication and addition.

The proposed design used common building blocks in the multiply and add cell, the input translator, and the output translator. Using the common building block approach to VLSI design greatly reduces design time. As a result of this, any extra design time spent optimizing the layout of these modules should be very beneficial to the performance of the overall design. This design methodology also achieves a higher chip density, resulting in both a cheaper and a higher performance implementation.

The system presented in this research was designed to interface with a system using the signed magnitude number system. If this design is attached to a purely residue processor, neither the input nor the output translators are necessary. This would greatly affect the area comparison calculations, beneficial to the residue number system. The input and output translators were designed and included

because there are no commercially available residue processors, hence input and output translators are essential at the present time.

7.1 Contributions

In this research, a design was formulated for a specific input word length and matrix bandwidth. It should be noted that there is no limitation when extending this methodology to either larger word lengths or matrix bandwidths. The number of truth table inputs is not dependent on the specific problem. In this research, two four-bit truth tables, and one two-bit truth table could have been implemented rather than two five-bit truth tables. In light of this, as many truth tables as necessary can be placed in parallel for larger word lengths. All other portions of the design may easily be extended to larger problems, although it may be necessary to add another modulus to satisfy dynamic range requirements.

Typical methods of residue addition, and especially multiplication, require the use of ROM's. ROM's tend to be very slow, particularly in this case, where the global clock speed (determined by the MAC) is of prime importance. A very regular and modular approach to residue multiplication and addition was presented. The fact that addition and multiplication occurs simultaneously in this

research is irrelevant, as each could occur alone with similar hardware. The proposed method of residue addition and multiplication is an excellent option to the VLSI designer. Along with residue addition and multiplication, this research also presented methods of input and output translation, which modified current methods of input and output translation. More importantly, these methods use the same building blocks as the MAC, which is essential to VLSI design.

This research also provided a comparison of the residue design to a more conventional approach. Although a larger amount of area is necessary to implement the residue design, it is still easily implemented on a single chip. The timing performance is very significant. The residue design is capable of a throughput greater than three times that of the binary design. The use of the residue system, through this comparison, should be greatly promoted. Also presented in this research were several practical design considerations, essential to a system designer considering a design of the residue type. Several comments were made on the less apparent properties of the RNS, as well as the process of moduli selection, and dynamic range determination.

The exact input and output coefficient timing was derived for the matrix multiplication algorithm, due to inadequate presentation in prior literature. Also, a

simulation of the algorithm was conducted such that timing information could be obtained. From the algorithm simulation and Quicksim logic simulation, the comparison to a conventional binary approach was made. The area required for the residue design was found to be 2.73 times larger than the conventional design in the worst case. A large portion of the difference is found in the input and output translation processes. The residue design is much faster. The residue design processes input matrices of bandwidth five 3.18 times faster than the conventional design. This proves that speed enhancements over conventional methods can be obtained through the application of the Residue Number System.

7.2 Future Research

There are many areas within the Residue Number System which would benefit from further research. In order for the RNS to find its way into the commercial market, several shortcomings must be overcome. Residue division, is very difficult, as well as sign determination and magnitude comparison. It is the above mentioned limitations that currently prevent the possibility of an all-RNS workstation.

It is very likely that many already developed algorithms, like the matrix multiplication algorithm, could also benefit greatly from the RNS. It is also likely

basic algorithms can be modified to exploit the characteristics and unique properties of the RNS. Especially algorithms which fail to converge from round off error, since the RNS does not allow this type of error.

The residue number system has many interesting properties to offer, but requires future designers to examine their individual applications, and to objectively evaluate the advantages and disadvantages of the RNS.

REFERENCES

- [1] Bayoumi, M. A., G. A. Jullien, and W. C. Miller, "Highly Parallel Architectures for DSP Algorithms using RNS", *Proceedings of ISCAS 85*, pp. 1395-1398, 1985.
- [2] Szabo, N. S. and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*, New York: McGraw-Hill, 1967.
- [3] Kung, H. T., "Structure of Parallel Algorithms", *Advances in Computers*, Vol. 19, pp. 65-112, 1980.
- [4] Baraniecka, A. and G. A. Jullien, "On Decoding Techniques for Residue Number System Realizations of Digital Signal Processing Hardware", *IEEE Transactions on Circuits and Systems*, Vol. CAS-25, No. 11, pp. 935-936, November 1978.
- [5] Leung, Y.-Y. J. and M. A. Shanblatt, *Performance Tradeoffs in the Hierarchical Design of Regular VLSI Structures*, Technical Report No. MSU-ENGR-86-001, Michigan State University, East Lansing, MI, January 1986.
- [6] Taylor, F. J., "A VLSI Residue Arithmetic Multiplier", *IEEE Transactions on Computers*, Vol. C-31, No. 6, pp. 540-546, June 1982.
- [7] Bayoumi, M. A., G. A. Jullien, and W. C. Miller, "A VLSI Implementation of Residue Adders", *IEEE Transactions on Circuits and Systems*, Vol. CAS-34, No. 3, pp. 284-288, March 1986.
- [8] Banerji, D., "A Novel Implementation for Addition and Subtraction in Residue Number Systems", *IEEE Transactions on Computers*, Vol. C-23, No. 1, pp. 106-109, January 1974.

- [9] Banerji, D. K. and J. A. Brzozowski, "On Translation Algorithms in Residue Number Systems", *IEEE Transactions on Computers*, Vol. C-21, No. 12, pp. 1281-1285, December 1972.
- [10] Alia, G. and E. Martinelli, "A VLSI Algorithm for Direct and Reverse Conversion from Weighted Binary Number System to Residue Number System", *IEEE Transactions on Circuits and Systems*, Vol. CAS-31, No. 12, pp. 1033-1039, December 1984.
- [11] Capocelli, R. M. and R. Giancarlo, "Efficient VLSI Networks for Converting an Integer from Binary System to Residue Number System and Vice Versa", *IEEE Transactions on Circuits and Systems*, Vol. CAS-35, No. 11, pp. 1425-1430, November 1988.
- [12] Bayoumi, M. A., G. A. Jullien, and W. C. Miller, "I/O Strategies for Residue Number System Architectures for Digital Signal Processing Applications", *International Symposium on Circuits and Systems 1984*, pp. 1069-1072, 1984.
- [13] Lyman, J., "Components and Packaging", *Electronic Design*, Vol. 37, No. 1, pp. 50-63, January 12, 1989.
- [14] Soderstrand, M. A., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, New York: IEEE Press, 1986.
- [15] Agnew, J., and R. C. Knapp, *Linear Algebra With Applications*, Monterrey, CA: Brooks/Cole Publishing Co., 1983.
- [16] Leung, Y.-Y. J. and M. A. Shanblatt, "Systolic Array Simulation for Quantification of Speed/Area Parameters", *Simulation*, Vol. 44, No. 6, pp. 295-300, June 1985.
- [17] Hwang, K., *Computer Arithmetic*, New York: John Wiley and Sons, 1987.

APPENDIX A

MATRIX MULTIPLICATION ALGORITHM SIMULATION RESULTS

5X5 MATRIX MULTIPLICATION SIMULATION

A					B					C				
2	3	9	0	0	1	4	8	0	0	-25	-40	-62	45	-36
1	2	3	-3	0	3	2	-5	3	0	-5	-31	-32	6	9
4	5	7	-2	2	-4	-6	-7	4	-4	-9	-30	-66	33	-24
0	6	-2	1	5	0	7	3	4	-7	26	31	-58	9	-24
0	0	-9	9	3	0	0	-9	-1	-5	36	117	63	-3	-42

11R	0	0	4	0	0	6	0	0	-9	0	0	0	0	0
7R	0	1	0	0	5	0	0	-2	0	0	9	0	0	0
4R	2	0	0	2	0	0	7	0	0	1	0	0	3	0
2R	0	0	3	0	0	3	0	0	-2	0	0	5	0	0
1R	0	0	0	0	9	0	0	-3	0	0	2	0	0	0

1L	0	0	0	0	-4	0	0	7	0	0	-9	0	0	0
3L	0	0	3	0	0	-6	0	0	3	0	0	-1	0	0
6L	1	0	0	2	0	0	-7	0	0	4	0	0	-5	0
10L	0	4	0	0	-5	0	0	4	0	0	-7	0	0	0
15L	0	0	8	0	0	3	0	0	-4	0	0	0	0	0

OUTPUTS

11	0	0	0	0	0	0	0	0	36	0	0	0	0	0
7	0	0	0	0	0	0	0	26	0	0	117	0	0	0
4	0	0	0	0	0	0	-9	0	0	31	0	0	63	0
2	0	0	0	0	0	-5	0	0	-30	0	0	-58	0	-3
1	0	0	0	0	-25	0	0	-31	0	0	-66	0	9	0
3	0	0	0	0	0	-40	0	0	-32	0	0	33	0	-24
6	0	0	0	0	0	0	-62	0	0	6	0	0	-24	0
10	0	0	0	0	0	0	0	45	0	0	9	0	0	0
15	0	0	0	0	0	0	0	0	-36	0	0	0	0	0

1 I	0	0	0	11	0	0	0	-10	0	0	-48	0	0	-42
L	0	0	0	0	-4	0	0	7	0	0	-9	0	0	0
R	0	0	0	0	9	0	0	-3	0	0	2	0	0	0
O	0	0	0	0	-25	0	0	-31	0	0	-66	0	9	0
2 I	0	0	0	0	7	0	0	-16	0	0	-13	0	0	-3
L	0	0	0	0	-4	0	0	7	0	0	-9	0	0	0
R	0	0	3	0	0	3	0	0	-2	0	0	5	0	0
O	0	0	0	0	-5	0	0	-30	0	0	-58	0	0	-3
3 I	0	0	0	0	14	0	0	-23	0	0	35	0	0	-24
L	0	0	3	0	0	-6	0	0	3	0	0	-1	0	0
R	0	0	0	0	9	0	0	-3	0	0	2	0	0	0
O	0	0	0	0	0	-40	0	0	-32	0	0	33	0	-24
4 I	0	0	0	0	0	19	0	0	24	0	0	90	0	0
L	0	0	0	0	0	-4	0	0	7	0	0	-9	0	0
R	2	0	0	2	0	0	0	7	0	0	1	0	0	3
O	0	0	0	0	0	-9	0	0	31	0	0	63	0	0
5 I	0	0	2	0	0	8	0	0	-42	0	0	14	0	-42
L	0	0	3	0	0	-6	0	0	3	0	0	-1	0	0
R	0	0	3	0	0	3	0	0	-2	0	0	5	0	0

	O	0	0	0	11	0	0	-10	0	0	-48	0	0	9	0	0	-42	0
6	I	0	0	0	0	0	0	1	0	0	18	0	0	-14	0	0	0	0
	L	1	0	0	2	0	0	-7	0	0	4	0	0	-5	0	0	0	0
	R	0	0	0	0	0	0	9	0	0	-3	0	0	2	0	0	0	0
	O	0	0	0	0	0	0	-62	0	0	6	0	0	-24	0	0	0	0
7	I	0	0	0	0	0	0	0	18	0	0	54	0	0	0	0	0	0
	L	0	0	0	0	0	0	-4	0	0	7	0	0	-9	0	0	0	0
	R	0	1	0	0	5	0	0	-2	0	0	9	0	0	0	0	0	0
	O	0	0	0	0	0	0	0	26	0	0	117	0	0	0	0	0	0
8	I	0	0	0	0	1	0	0	26	0	0	-16	0	0	0	0	0	0
	L	0	0	0	0	3	0	0	-6	0	0	3	0	0	-1	0	0	0
	R	0	2	0	0	2	0	0	7	0	0	1	0	0	3	0	0	0
	O	0	0	0	0	7	0	0	-16	0	0	-13	0	0	-3	0	0	0
9	I	0	0	0	0	8	0	0	-2	0	0	43	0	0	1	0	0	0
	L	1	0	0	0	2	0	0	-7	0	0	4	0	0	-5	0	0	0
	R	0	0	0	0	3	0	0	3	0	0	-2	0	0	5	0	0	0
	O	0	0	0	0	14	0	0	-23	0	0	35	0	0	-24	0	0	0
10	I	0	0	0	0	0	0	0	9	0	0	-12	0	0	0	0	0	0
	L	0	4	0	0	-5	0	0	4	0	0	-7	0	0	0	0	0	0
	R	0	0	0	0	0	0	0	9	0	0	-3	0	0	2	0	0	0
	O	0	0	0	0	0	0	0	45	0	0	9	0	0	0	0	0	0
11	I																	
	L	0	0	0	0	0	0	0	-4	0	0	7	0	0	-9	0	0	0
	R	0	0	4	0	0	0	6	0	0	-9	0	0	0	0	0	0	0
	O	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	0
12	I	0	0	0	0	0	4	0	0	12	0	0	63	0	0	0	0	0
	L	0	0	0	0	0	3	0	0	-6	0	0	3	0	0	-1	0	0
	R	0	1	0	0	0	5	0	0	-2	0	0	9	0	0	0	0	0
	O	0	0	0	0	0	19	0	0	24	0	0	90	0	0	0	0	0
13	I	0	0	0	0	0	4	0	0	7	0	0	10	0	0	-27	0	0
	L	0	1	0	0	0	2	0	0	-7	0	0	4	0	0	-5	0	0
	R	0	2	0	0	0	2	0	0	7	0	0	1	0	0	3	0	0
	O	0	0	2	0	0	8	0	0	-42	0	0	14	0	0	-42	0	0
14	I	0	0	0	0	0	16	0	0	6	0	0	-28	0	0	0	0	0
	L	0	4	0	0	0	-5	0	0	4	0	0	-7	0	0	0	0	0
	R	0	0	0	0	0	3	0	0	3	0	0	-2	0	0	5	0	0
	O	0	0	0	0	0	1	0	0	18	0	0	-14	0	0	0	0	0
15	I																	
	L	0	0	8	0	0	0	3	0	0	-4	0	0	0	0	0	0	0
	R	0	0	0	0	0	0	0	0	9	0	0	-3	0	0	2	0	0
	O	0	0	0	0	0	0	0	0	-36	0	0	0	0	0	0	0	0
16	I																	
	L	0	0	0	0	0	0	3	0	0	-6	0	0	3	0	0	-1	0
	R	0	0	0	4	0	0	0	6	0	0	-9	0	0	0	0	0	0
	O	0	0	0	0	0	0	18	0	0	54	0	0	0	0	0	0	0
17	I	0	0	0	0	0	0	16	0	0	-30	0	0	-36	0	0	0	0
	L	0	0	1	0	0	0	2	0	0	-7	0	0	4	0	0	-5	0
	R	0	0	0	1	0	0	0	5	0	0	-2	0	0	9	0	0	0
	O	0	0	0	1	0	0	26	0	0	-16	0	0	0	0	0	0	0
18	I	0	0	0	0	0	0	8	0	0	15	0	0	8	0	0	0	0
	L	0	0	0	4	0	0	-5	0	0	4	0	0	-7	0	0	0	0
	R	0	0	2	0	0	0	2	0	0	7	0	0	1	0	0	3	0
	O	0	0	0	8	0	0	-2	0	0	43	0	0	1	0	0	0	0

19	I																
	L	0	0	8	0	0	3	0	0	-4	0	0	0	0	0	0	0
	R	0	0	0	0	0	3	0	0	3	0	0	-2	0	0	5	0
	O	0	0	0	0	0	9	0	0	-12	0	0	0	0	0	0	0
20	I																
	L	0	0	0	1	0	0	2	0	0	-7	0	0	4	0	0	-5
	R	0	0	0	4	0	0	6	0	0	-9	0	0	0	0	0	0
	O	0	0	0	4	0	0	12	0	0	63	0	0	0	0	0	0
21	I																
	L	0	0	0	0	0	0	32	0	0	18	0	0	36	0	0	0
	R	0	0	0	4	0	0	-5	0	0	4	0	0	-7	0	0	0
	O	0	0	0	1	0	0	5	0	0	-2	0	0	9	0	0	0
	O	0	0	0	4	0	0	7	0	0	10	0	0	-27	0	0	0
22	I																
	L	0	0	0	8	0	0	3	0	0	-4	0	0	0	0	0	0
	R	0	0	0	2	0	0	2	0	0	7	0	0	1	0	0	3
	O	0	0	0	0	16	0	0	6	0	0	-28	0	0	0	0	0
23	I																
	L	0	0	0	0	4	0	0	-5	0	0	4	0	0	-7	0	0
	R	0	0	0	0	4	0	0	6	0	0	-9	0	0	0	0	0
	O	0	0	0	0	16	0	0	-30	0	0	-36	0	0	0	0	0
24	I																
	L	0	0	0	0	8	0	0	3	0	0	-4	0	0	0	0	0
	R	0	0	0	0	1	0	0	5	0	0	-2	0	0	9	0	0
	O	0	0	0	0	0	8	0	0	15	0	0	8	0	0	0	0
25	I																
	L	0	0	0	0	0	8	0	0	3	0	0	-4	0	0	0	0
	R	0	0	0	0	0	4	0	0	6	0	0	-9	0	0	0	0
	O	0	0	0	0	0	32	0	0	18	0	0	36	0	0	0	0

APPENDIX B
TRUTH TABLES AND KARNAUGH MAPS

Modulo 7 Lower Truth Table

5 BIT RESULT		TRUTH TABLE OUTPUT										
RESULT	MOD 7	W	X	Y	Z							
0	0	0	0	0	0	W						
1	1	0	0	0	1	0	0	0	0	0	0	0
2	2	0	0	1	0	0	0	0	0	0	0	0
3	3	0	0	1	1	0	0	0	0	0	0	0
4	4	0	1	0	0	0	0	0	0	0	0	0
5	5	0	1	0	1	X						
6	6	0	1	1	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	1	1	0	0	0	0
8	1	0	0	0	1	0	0	1	1	0	0	1
9	2	0	0	1	0	0	0	0	1	1	0	1
10	3	0	0	1	1	0	0	1	0	0	1	1
11	4	0	1	0	0	Y						
12	5	0	1	0	1	0	0	0	0	1	1	0
13	6	0	1	1	0	0	0	0	1	1	0	1
14	0	0	0	0	0	0	0	1	1	1	0	0
15	1	0	0	0	1	0	1	0	0	0	1	1
16	2	0	0	1	0	1	1	0	1	0	0	1
17	3	0	0	1	1	Z						
18	4	0	1	0	0	0	0	1	1	0	0	1
19	5	0	1	0	1	1	1	0	0	1	0	1
20	6	0	1	1	0	1	1	0	0	1	0	1
21	0	0	0	0	0	1	0	1	0	1	0	1
22	1	0	0	0	1	0	0	0	1	0	1	0
23	2	0	0	1	0							
24	3	0	0	1	1							
25	4	0	1	0	0							
26	5	0	1	0	1							
27	6	0	1	1	0							
28	0	0	0	0	0							
29	1	0	0	0	1							
30	2	0	0	1	0							
31	3	0	0	1	1							

Modulo 11 Lower Truth Table

5 BIT RESULT	RESULT MOD 11	TRUTH TABLE OUTPUT				W	X	Y	Z
		W	X	Y	Z				
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
2	2	0	0	1	0	0	0	1	0
3	3	0	0	1	1	0	0	0	1
4	4	0	1	0	0	0	0	0	0
5	5	0	1	0	1	0	0	0	0
6	6	0	1	1	0	0	1	0	0
7	7	0	1	1	1	0	1	0	0
8	8	1	0	0	0	0	1	0	0
9	9	1	0	0	1	0	1	0	0
10	10	1	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	1	1
12	1	0	0	0	1	0	0	1	1
13	2	0	0	1	0	1	1	0	0
14	3	0	0	1	1	1	1	0	0
15	4	0	1	0	0	0	0	0	0
16	5	0	1	0	1	0	0	1	0
17	6	0	1	1	0	1	1	0	0
18	7	0	1	1	1	1	0	0	1
19	8	1	0	0	0	1	1	1	1
20	9	1	0	0	1	0	0	0	0
21	10	1	0	1	0	0	0	1	0
22	0	0	0	0	0	0	0	0	0
23	1	0	0	0	1	0	0	1	0
24	2	0	0	1	0	1	0	0	1
25	3	0	0	1	1	0	0	1	1
26	4	0	1	0	0	1	0	0	0
27	5	0	1	0	1	0	0	1	0
28	6	0	1	1	0	0	1	0	0
29	7	0	1	1	1	0	0	1	0
30	8	1	0	0	0	1	1	1	1
31	9	1	0	0	1	0	0	0	0

Modulo 13 Lower Truth Table

5 BIT RESULT	MOD 13	TRUTH TABLE OUTPUT				W	X	Y	Z
		W	X	Y	Z				
0	0	0	0	0	0				
1	1	0	0	0	1	0	0	1	
2	2	0	0	1	0	0	0	0	
3	3	0	0	1	1	0	0	0	
4	4	0	1	0	0	0	0	0	
5	5	0	1	0	1				
6	6	0	1	1	0	X			
7	7	0	1	1	1	0	1	1	
8	8	1	0	0	0	0	1	0	
9	9	1	0	0	1	0	1	0	
10	10	1	0	1	0	0	1	0	
11	11	1	0	1	1				
12	12	1	1	0	0	Y			
13	0	0	0	0	0	0	0	0	
14	1	0	0	0	1	0	0	0	
15	2	0	0	1	0	1	1	1	
16	3	0	0	1	1	1	1	0	
17	4	0	1	0	0				
18	5	0	1	0	1	Z			
19	6	0	1	1	0	0	0	0	
20	7	0	1	1	1	1	1	1	
21	8	1	0	0	0	1	1	0	
22	9	1	0	0	1	0	0	0	
23	10	1	0	1	0	0	1	0	
24	11	1	0	1	1				
25	12	1	1	0	0				
26	0	0	0	0	0				
27	1	0	0	0	1				
28	2	0	0	1	0				
29	3	0	0	1	1				
30	4	0	1	0	0				
31	5	0	1	0	1				

Modulo 7 Upper Truth Table

5 BIT RESULT	ACTL. VALUE	RESULT MOD 7	TRUTH TABLE OUTPUT				W	X	Y	Z	W	X	Y	Z		
			W	X	Y	Z										
0	0	0	0	0	0	0	0	0	0	0	*	*	*	*		
1	32	4	0	1	0	0	0	0	0	0	*	*	*	*		
2	64	1	0	0	0	1	0	0	0	0	*	*	*	*		
3	96	5	0	1	0	1	0	0	0	0	*	*	*	*		
4	128	2	0	0	1	0	0	0	0	0	*	*	*	*		
5	160	6	0	1	1	0	0	0	0	0	*	*	*	*		
6	192	3	0	0	1	1	0	0	0	0	*	*	*	*		
7	224	0	0	0	0	0	0	0	0	0	*	*	*	*		
8	256	*	*	*	*	*	*	*	*	1	0	1	*	*	*	
9	288	*	*	*	*	*	*	*	*	0	0	*	*	*	*	
10	320	*	*	*	*	*	*	*	*	0	0	*	*	*	*	
11	352	*	*	*	*	*	*	*	*	0	0	*	*	*	*	
12	384	*	*	*	*	*	*	*	*	0	1	*	*	*	*	
13	416	*	*	*	*	*	*	*	*	0	1	*	*	*	*	
14	448	*	*	*	*	*	*	*	*	0	0	0	*	*	*	
15	480	4	0	1	0	0	0	0	0	0	1	*	*	*	*	
16	512	1	0	0	0	1	0	0	0	0	1	*	*	*	*	
17	544	5	0	1	0	1	0	0	0	0	0	*	*	*	*	
18	576	2	0	0	1	1	0	0	0	0	0	*	*	*	*	
19	608	6	0	1	1	0	0	0	0	0	0	*	*	*	*	
20	640	*	*	*	*	*	*	*	*	1	0	0	*	*	*	*
21	672	*	*	*	*	*	*	*	*	1	1	*	*	*	*	*
22	704	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
23	736	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
24	768	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
25	800	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
26	832	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
27	864	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
28	896	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
29	928	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
30	960	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*
31	992	*	*	*	*	*	*	*	*	0	0	*	*	*	*	*

Modulo 11 Upper Truth Table

5 BIT RESULT	ACTL. VALUE	RESULT MOD 11	TRUTH TABLE OUTPUT				W	X	Y	Z
			W	X	Y	Z				
0	0	0	0	0	0	0				
1	32	10	1	0	1	0				
2	64	9	1	0	0	1				
3	96	8	1	0	0	0				
4	128	7	0	1	1	1				
5	160	6	0	1	1	0				
6	192	5	0	1	0	1				
7	224	4	0	1	0	0				
8	256	*	*	*	*	*				
9	288	*	*	*	*	*				
10	320	*	*	*	*	*				
11	352	*	*	*	*	*				
12	384	*	*	*	*	*				
13	416	*	*	*	*	*				
14	448	*	*	*	*	*				
15	480	7	0	1	1	1				
16	512	6	0	1	1	0				
17	544	5	0	1	0	1				
18	576	4	0	1	0	0				
19	608	3	0	0	1	1				
20	640	*	*	*	*	*				
21	672	*	*	*	*	*				
22	704	*	*	*	*	*				
23	736	*	*	*	*	*				
24	768	*	*	*	*	*				
25	800	*	*	*	*	*				
26	832	*	*	*	*	*				
27	864	*	*	*	*	*				
28	896	*	*	*	*	*				
29	928	*	*	*	*	*				
30	960	*	*	*	*	*				
31	992	*	*	*	*	*				

W	0	0	*	*	0	*	*	*	*
	1	0	*	*	0	*	*	*	*
	1	0	0	*	0	*	*	*	*
	1	0	*	*	0	*	*	*	*
X	0	1	*	*	1	*	*	*	*
	0	1	*	*	1	*	*	*	*
	0	1	1	*	0	*	*	*	*
	0	1	*	*	1	*	*	*	*
Y	0	1	*	*	1	*	*	*	*
	1	1	*	*	0	*	*	*	*
	0	0	1	*	1	*	*	*	*
	0	0	*	*	0	*	*	*	*
Z	0	1	*	*	0	*	*	*	*
	0	0	*	*	1	*	*	*	*
	0	0	1	*	1	*	*	*	*
	1	1	*	*	0	*	*	*	*

Module 13 Upper Truth Table

5 BIT RESULT	ACTL. VALUE	RESULT MOD 13	TRUTH TABLE OUTPUT				W	X	Y	Z			
			W	X	Y	Z							
0	0	0	0	0	0	0							
1	32	6	0	1	1	0	0	1	*	*	*		
2	64	12	1	1	0	0	0	0	1	*	*	*	
3	96	5	0	1	0	1	1	1	*	*	*		
4	128	11	1	0	1	1	1	1	*	*	*		
5	160	4	0	1	0	0	0						
6	192	10	1	0	1	0	0	0	0	*	*	*	
7	224	3	0	0	1	1	1	1	1	*	*	*	
8	256	*	*	*	*	*	*	1	0	1	*	*	*
9	288	*	*	*	*	*	*	1	0	*	*	*	
10	320	*	*	*	*	*	*						
11	352	*	*	*	*	*	*						
12	384	*	*	*	*	*	*						
13	416	*	*	*	*	*	*	0	1	*	*	*	
14	448	*	*	*	*	*	*	1	0	*	*	*	
15	480	12	1	1	0	0	0	0	1	0	*	*	*
16	512	5	0	1	0	1	1						
17	544	11	1	0	1	1	1						
18	576	4	0	1	0	0	0						
19	608	10	1	0	1	0	0	0	0	*	*	*	
20	640	*	*	*	*	*	*	1	1	0	*	*	*
21	672	*	*	*	*	*	*	0	0	*	*	*	
22	704	*	*	*	*	*	*						
23	736	*	*	*	*	*	*						
24	768	*	*	*	*	*	*						
25	800	*	*	*	*	*	*						
26	832	*	*	*	*	*	*						
27	864	*	*	*	*	*	*						
28	896	*	*	*	*	*	*						
29	928	*	*	*	*	*	*						
30	960	*	*	*	*	*	*						
31	992	*	*	*	*	*	*						

APPENDIX C
SCHEMATIC PLOTS

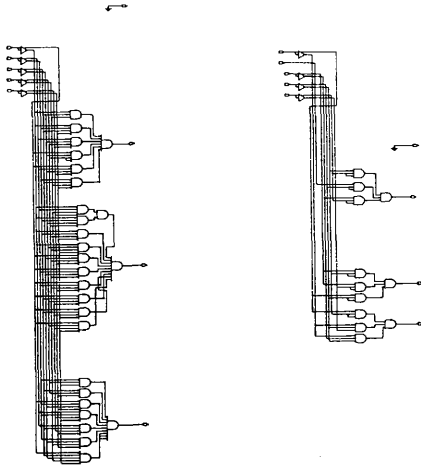


Figure A.1 Modulo 7 Truth Table Hardware

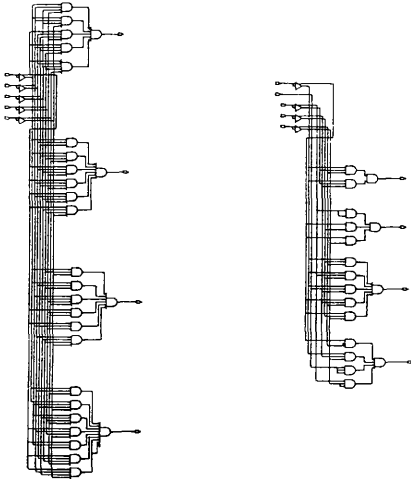


Figure A.2 Modulo 11 Truth Table Hardware

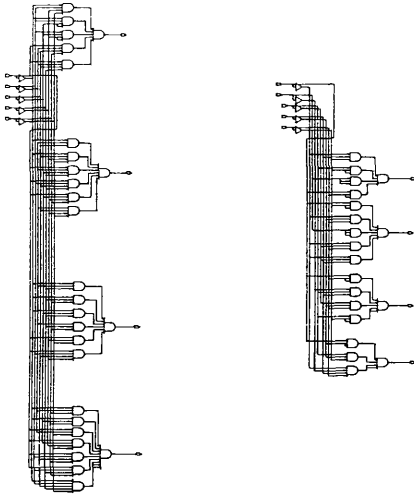


Figure A.3 Modulo 13 Truth Table Hardware

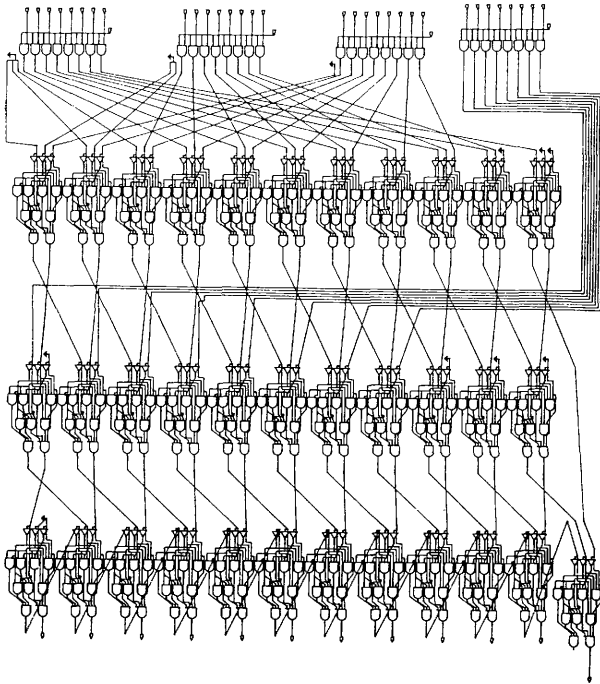


Figure A.4 8X4 Multiplier

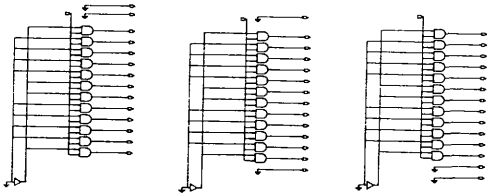


Figure A.5 Twelve-Bit Multiple Generator

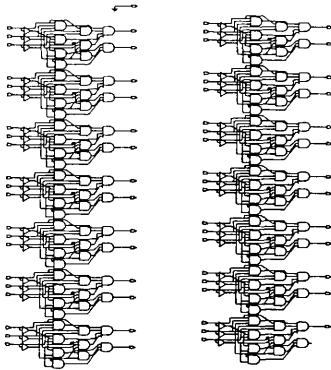


Figure A.6 Fourteen-Bit Carry Save Adder

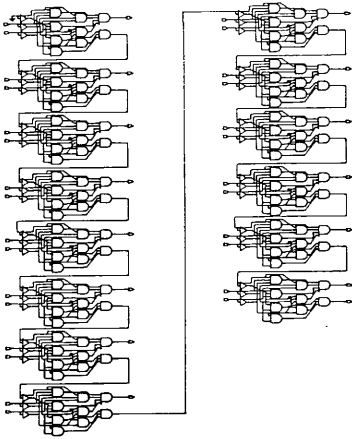


Figure A.7 Fourteen-Bit Binary Adder

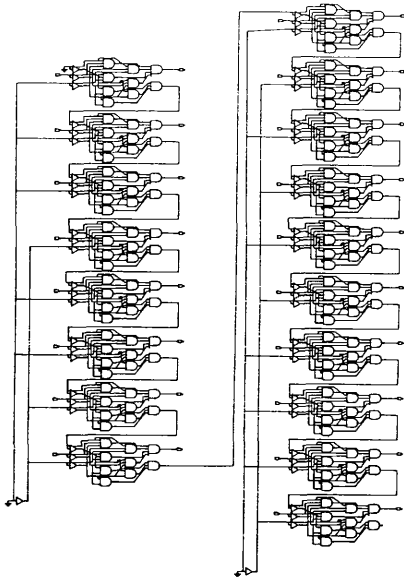


Figure A.8 Modified Adder A

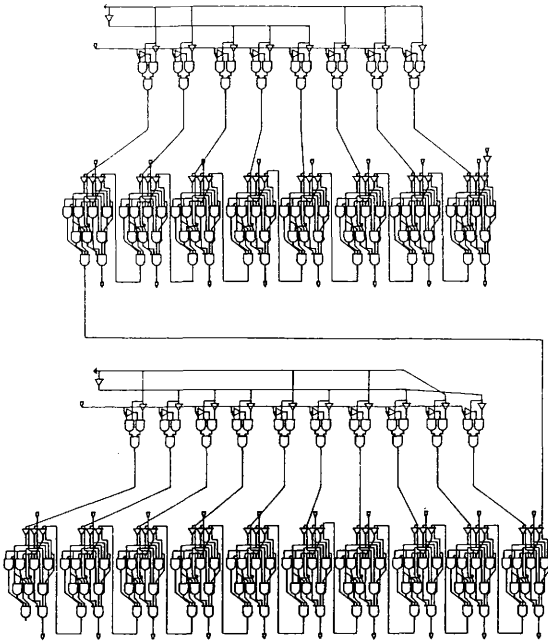


Figure A.9 Modified Adder B

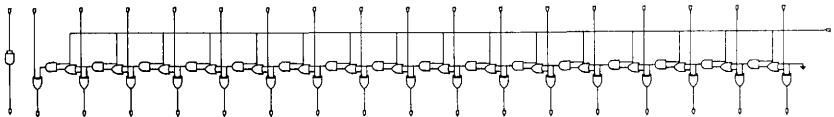


Figure A.10 Seventeen-Bit Two's Complementer

APPENDIX D
SIMULATION RESULTS

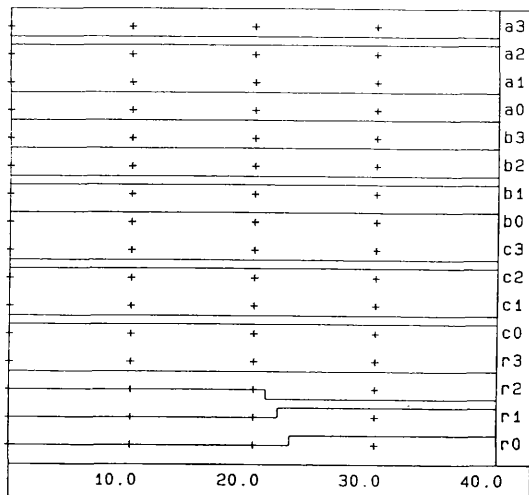


Figure A.11 Modulo 7 Trial #1 MAC Simulation

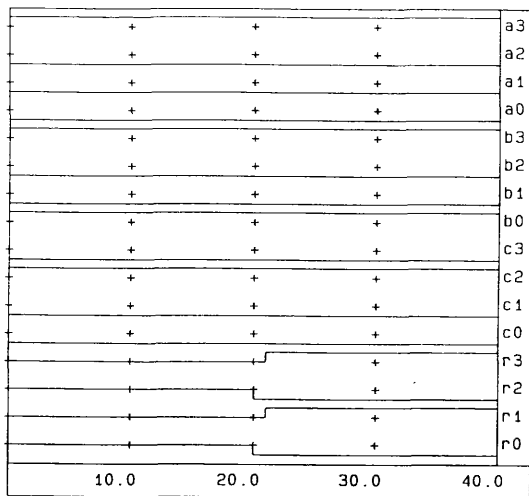


Figure A.12 Modulo 11 Trial #1 MAC Simulation

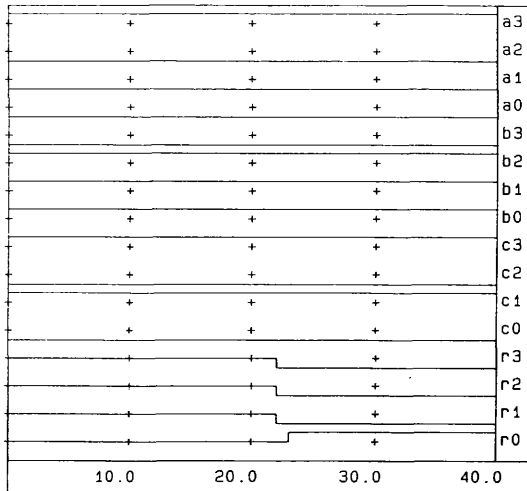


Figure A.13 Modulo 13 Trial #1 MAC Simulation

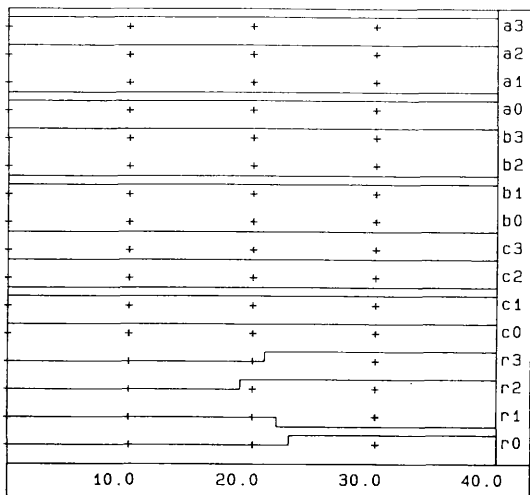


Figure A.14 Modulo 15 Trial #1 MAC Simulation

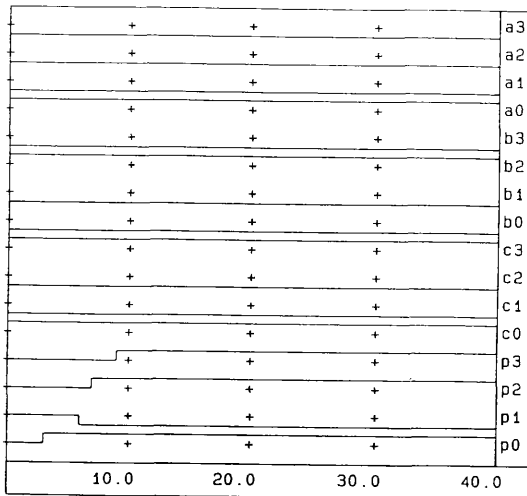


Figure A.15 Modulo 16 Trial #1 MAC Simulation

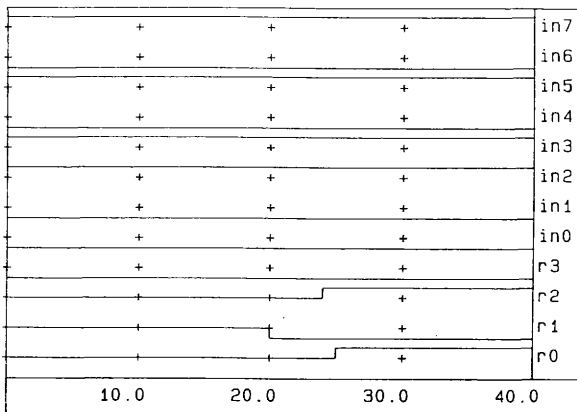


Figure A.16 Modulo 7 Trial #1 Input Translation Simulation

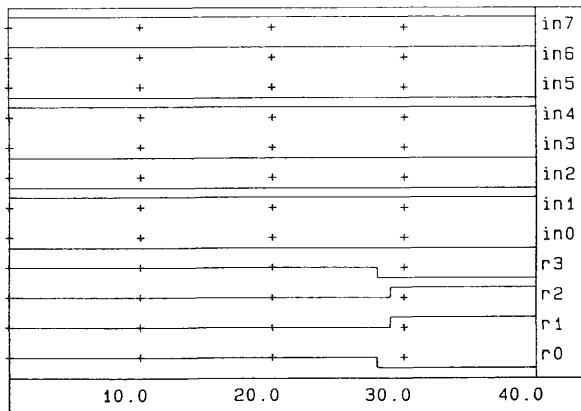


Figure A.17 Modulo 11 Trial #1 Input Translation Simulation

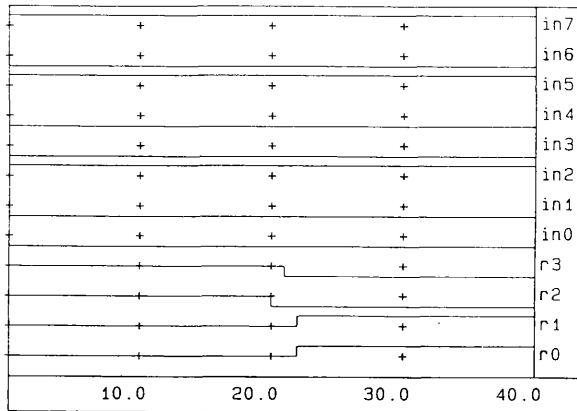


Figure A.18 Modulo 13 Trial #1 Input Translation Simulation

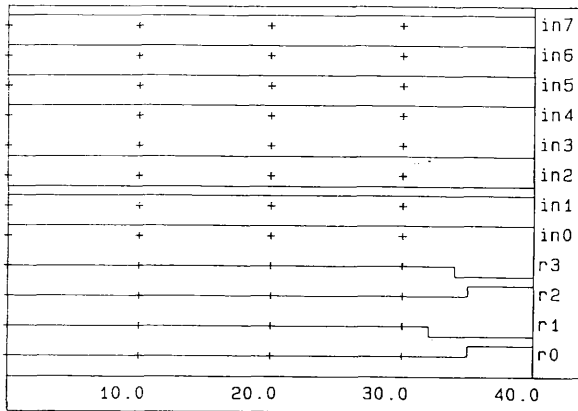


Figure A.19 Modulo 15 Trial #1 Input Translation Simulation

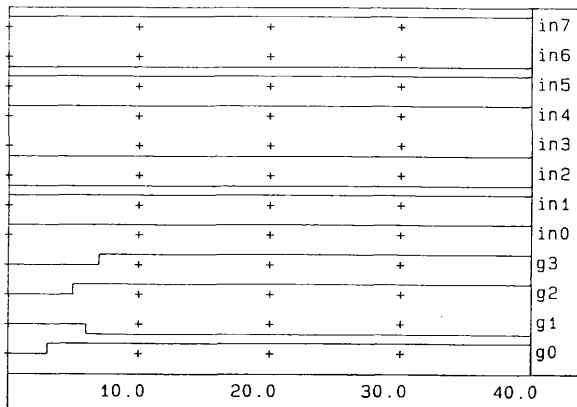


Figure A.20 Modulo 16 Trial #1 Input Translation Simulation

VITA

Gary Franklin Chard received his B. S. Degree in Electrical Engineering in May of 1988 from Texas A&M University, College Station, Texas. He is currently completing his M. S. Degree in Electrical Engineering also from Texas A&M University. He has worked for Texas Instruments Incorporated during the summer since 1985. He is a member of Eta Kappa Nu and Tau Beta Pi. His permanent address is 2503 Grandview Dr., Richardson, Texas 75080.