Modular Multiplication in the Residue Number System

A DISSERTATION SUBMITTED TO

THE SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

OF THE UNIVERSITY OF ADELAIDE

BY

Yinan KONG

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY

July 2009

Declaration of Originality

Name: Yinan KONG   Program: Ph.D.

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

The author acknowledges that copyright of published works contained within this thesis (as listed below) resides with the copyright holder/s of those works.

Signature: Date:

Acknowledgments

My supervisor, Dr Braden Jace Phillips, is an extremely hard working and dedicated man. So, first and foremost, I would like to say “thank you” to him, for his critical guidance, constant sustainment and role modelling as a supervisor. It is my true luck that I have been able to work with him for these years. This has been a precious experience which deserves my cherishing throughout my whole life.

I am also grateful to Associate Professor Cheng-Chew Lim and Dr Alison Wolff for their guidance through important learning phases of this complex technology. Throughout the course of my study I have received considerable help from my colleagues Daniel Kelly and Zhining Lim, who have been, and will remain, my great friends.

Acknowledgement is given to the Australian Research Council (ARC) as this work has been supported by the ARC Discovery Project scheme.

Thanks to the support from my family and friends. I have been relying on you throughout my candidature. Thanks are due to Mum, Dad, Ranran, Zhaozhao and Jingdong JU, who flew over 5000 miles to take care of me. I would definitely not have been able to get to this point without your encouragement. You are the real pearls lying on the bottom of my mind.

Mother, thank you for giving birth to me as well as cultivating me through those tough years. This thesis has your sweat in it.

My love, BEN YA, you are my greatest inspiration. Thank you for all you have done for me.

Yinan KONG
November 2008

Abstract

Public-key cryptography is a mechanism for secret communication between parties who have never before exchanged a secret message. This thesis contributes arithmetic algorithms and hardware architectures for the modular multiplication $Z = A \times B \mod M$. This operation is the basis of many public-key cryptosystems including RSA and Elliptic Curve Cryptography. The *Residue Number System* (RNS) is used to speed up long word length modular multiplication because this number system performs certain long word length operations, such as multiplication and addition, much more efficiently than positional systems.

A survey of current modular multiplication algorithms shows that most work in a positional number system, e.g. binary. A new classification is developed which classes these algorithms as Classical, Sum of Residues, Montgomery or Barrett. Each class of algorithm is analyzed in detail, new developments are described, and the improved algorithms are implemented and compared using FPGA hardware.

Few modular multiplication algorithms for use in the RNS have been published. Most are concerned with short word lengths and are not applicable to public-key cryptosystems that require long word length operations. This thesis sets out the hypothesis that each of the four classes of modular multiplication algorithms possible in positional number systems can also be used for long word length modular multiplication in the RNS; moreover using the RNS in this way will lead to faster implementations than those which restrict themselves to positional number systems. This hypothesis is addressed by developing new Classical, Sum of Residues and Barrett algorithms for modular multiplication in the RNS. Existing Montgomery RNS algorithms are also discussed.

The new Sum of Residues RNS algorithm results in a hardware implementation that is novel in many aspects: a highly parallel structure using short arithmetic operations within the RNS; fully scalable hardware; and the fastest ever FPGA implementation of the 1024-bit RSA cryptosystem at 0.4 ms per decryption.

Publications

1. Yinan Kong and Braden Phillips, “Fast Scaling in the Residue Number System”, accepted by IEEE Transactions on VLSI Systems in December 2007.

2. Yinan Kong and Braden Phillips, “Simulations of modular multipliers on FPGAs”, Proceedings of the IASTED Asian Conference on Modelling and Simulation, Beijing, China, Oct. 2007, pp. 11281131.

3. Yinan Kong and Braden Phillips, “Comparison of Montgomery and Barrett modular multipliers on FPGAs”, 40th Asilomar Conference on Signals, Systems and Computers. Pacific Grove, CA, USA: IEEE, Piscataway, NJ, USA, Oct. 2006, pp. 16871691.

4. Yinan Kong and Braden Phillips, “Residue number system scaling schemes”, in Smart Structures, Devices, and Systems II, ser. Proc. SPIE, S. F. Al-Sarawi, Ed., vol. 5649, Feb. 2005, pp. 525536.

5. Yinan Kong and Braden Phillips, “A classical modular multiplier for RNS channel operations”, The University of Adelaide, CHiPTec Tech. Rep. CHIPTEC-05-02, November 2005.

6. Yinan Kong and Braden Phillips, “A Montgomery modular multiplier for RNS channel operations”, The University of Adelaide, CHiPTec Tech. Rep. CHIPTEC-05-02, November 2005.

Publications in Submission

1. Yinan Kong and Braden Phillips, "Modular Reduction and Scaling in the Residue Number System Using Multiplication by the Inverse", submitted to IEEE Transactions on VLSI Systems in November 2008.

2. Yinan Kong and Braden Phillips, "Low latency modular multiplication for public-key cryptosystems using a scalable array of parallel processing elements", submitted to 19th IEEE Computer Arithmetic in October 2008.

3. Braden Phillips and Yinan Kong, "Highly Parallel Modular Multiplication in the Residue Number System using Sum of Residues Reduction", submitted to Journal of Applicable Algebra in Engineering, Communication and Computing in June 2008.

4. Yinan Kong and Braden Phillips, "Revisiting Sum of Residues Modular Multiplication", submitted to International Journal of Computer Systems Science and Engineering in May 2008.

Nomenclature

$\langle X \rangle_M$	The operation X mod M.
D	The dynamic range of a RNS.
M	The modulus of a modular multiplication, typically n bits.
m_i	The ith RNS channel modulus.
N	The number of RNS channels.
n	The wordlength of M.
w	The RNS channel width.
$\lceil X \rceil$	The ceiling of X. The smallest integer greater than or equal to X.
$\lfloor X \rfloor$	The floor of X. The largest integer smaller than or equal to X.
BE	Base Extension.
CRT	Chinese Remainder Theorem.
DSP	Digital Signal Processing.
ECC	Elliptic Curve Cryptography.
LUC	Look-Up Cycle.
LUT	Look-Up Table.
LUT	Look-Up Table

MRS	Mixed Radix Number System.
QDS	Quotient Digit Selection.
RNS	Residue Number System.
RSA	RSA Cryptography.

Contents

1 Introduction **1**

1.1 Thesis Outline 2

1.2 Contribution 4

2 Background **7**

2.1 Residue Number Systems 8

2.1.1 RNS Representation 8

2.1.2 Conversion between RNS and Positional Number Systems 9

2.1.3 RNS Arithmetic 14

2.1.4 Moduli Selection 15

2.1.5 Base Extension 16

2.2 RSA Public-Key Cryptography 18

2.2.1 Public-Key Cryptography 18

2.2.2 The RSA Cryptosystem 19

2.2.3 Exponentiation 20

3 Four Ways to Do Modular Multiplication in a Positional Number System **25**

3.1 Introducing Four Ways to Do Modular Multiplication 26

3.1.1 Classical Modular Multiplication 27

3.1.2 Sum of Residues Modular Multiplication 29

3.1.3 Barrett Modular Multiplication 30

3.1.4 Montgomery Modular Multiplication 31

3.2 Reinvigorating Sum of Residues 33
3.2.1 Tomlinson's Algorithm 33
3.2.2 Eliminating the Carry Propagate Adder 35
3.2.3 Further Enhancements 37
3.2.4 High Radix 40
3.2.5 Summary of the Sum of Residues Modular Multiplication 40
3.3 Bounds of Barrett Modular Multiplication 42
3.3.1 Bound Deduction 42
3.3.2 Performance at Different Word Lengths 45
3.4 Montgomery Modular Multiplication on FPGA 47
3.4.1 Separated Montgomery Modular Multiplication Algorithm 47
3.4.2 High Radix Montgomery Algorithm 47
3.4.3 Bound Deduction 54
3.4.4 Interleaved vs. Separated Structure 57
3.4.5 Trivial Quotient Digit Selection 58
3.4.6 Quotient Digit Pipelining 62
3.5 Modular Multiplications within RNS Channels 69

4 Four Ways to Do Modular Multiplication in the Residue Number System 71
4.1 Short Word Length RNS Modular Multiplications using Look-Up Tables 73
4.1.1 RNS Modular Reduction, Scaling and Look-Up Tables 74
4.1.2 Existing RNS Scaling Schemes using Look-Up Tables . 75
4.1.3 A New RNS Scaling Scheme using Look-Up Tables . . 88
4.2 RNS Classical Modular Multiplication 93
4.2.1 The Core Function 93
4.2.2 A Classical Modular Multiplication Algorithm in RNS using the Core Function 97
4.2.3 Examples 105

4.3 RNS Sum of Residues Modular Multiplication 109
4.3.1 Sum of Residues Reduction in the RNS 109
4.3.2 Approximation of α 110
4.3.3 Bound Deduction 115
4.3.4 The RNS Sum of Residues Modular Multiplication Algorithm 117
4.4 RNS Barrett Modular Multiplication 121
4.4.1 RNS Modular Reduction using Barrett Algorithm . . . 121
4.4.2 The Algorithm 127
4.5 RNS Montgomery Modular Multiplication 133
4.5.1 Montgomery Modular Reduction in RNS 133
4.5.2 The Algorithm 136
4.5.3 A Comparison between RNS Barrett and Montgomery Modular Multiplication 136

5 Implementation of RNS Sum of Residues Modular Multiplication 141
5.1 A Scalable Structure for Sum of Residues Modular Multiplication in RNS 142
5.1.1 A 4-Channel Architecture 143
5.1.2 The Scaled Architecture for Modular Multiplication . . 144
5.2 Implementation Results 149

6 Conclusion and Future Perspectives 153
6.1 Conclusion 154
6.2 Future Perspectives 154

Bibliography 157

List of Figures

2.1 Conversion from the RNS to the Mixed Radix Number System (MRS) . . . 12

3.1 An Architecture for Tomlinson's Modular Multiplication Algorithm [Tomlinson89] . . . 34

3.2 Modified Sum of Residues Modular Multiplier Architecture . . 36

3.3 n-bit Carry Save Adders . . . 37

3.4 New Sum of Residues Modular Multiplier Architecture 38

3.5 An Example for New Sum of Residues Modular Multiplication $n = 4, A = (1111)_2, B = (1011)_2$ and $M = (1001)_2$ . . . 39

3.6 New High-Radix Sum of Residues Modular Multiplier Architecture . . . 41

3.7 Delays and Areas of Improved Barrett Modular Multiplication Algorithm . . . 46

3.8 Delays of Montgomery Modular Multiplication at Different Radices. . . . 54

3.9 Delays and Areas of Interleaved and Separated Montgomery Modular Multiplication at Different Radices . . . 59

3.10 Shortest Delays of Separated Montgomery Multiplication Algorithm using Trivial QDS at n from 12 to 32 . . . 62

3.11 Shortest Delays of Separated Montgomery Multiplication Algorithm using Quotient Digit Pipelining at n from 13 to 32 . . 67

3.12 Shortest Delays of Separated Montgomery Multiplication Algorithm with & without Quotient Digit Pipelining at n from 13 to 32 68
3.13 Delays and Areas of Four Classes of Modular Multipliers in Binary 70

4.1 RNS Scaling using Look-Up Tables. 75
4.2 The 3 Modulus RNS Scaler 77
4.3 RNS Scaling using Estimation for Channels $S+1 \leq i \leq N$ . . 80
4.4 RNS Scaling using a Decomposition of the CRT (two redundant channels with channel m_i) 83
4.5 RNS Scaling using Parallel Base Extension (BE) Blocks 85
4.6 Parallel Architecture to Perform $Y = \left\lfloor \frac{X}{M} \right\rfloor \approx \sum_{i=1}^{N} f(x_i)$. . . . 86
4.7 One Channel of a Parallel Residue Arithmetic Process using Memories with Addressing Capacity $= r$. 87
4.8 A New Architecture to Perform Short Word length Scaling in RNS 90
4.9 A Typical Core Function C(X): $D = 30030$; $C(D) = 165$ . . . 95
4.10 The h Least Significant Bits to be Maintained during the Modular Multiplication $\langle x_i C(\sigma_i) \rangle_{2^h}$ 102
4.11 An End-Around Adder to Compute $\langle \theta_i + \theta_{i+1} \rangle_{2^k-1}$. 126
4.12 Architecture for Scaling by 2 in RNS 128
4.13 High-Radix Architecture for Scaling by 2^l in RNS 129
4.14 Delay of Channel-Width Multiplication and Addition modulo a General Modulus and Multiplication and Addition modulo $2^k - 1$ used for RNS Barrett Algorithm against RNS Channel Width w 132
4.15 Delay of Barrett and Montgomery RNS Modular Multiplication against RNS Channel Width w 138

5.1 A 4-Channel RNS Modular Multiplier 145

5.2 A 4-Channel Modular Multiplier Block Suitable for a Scalable Array of Blocks 146
5.3 A 12-Channel Modular Multiplier Built from a 3×3 Array of 4-Channel Blocks 148

List of Tables

2.1 The Pre-Computed Constants of RNS $\{3,5,7\}$ 9

2.2 An Example of Right-to-Left Modular Exponentiation 23

3.1 Montgomery Modular Reduction Process at Radix 2 49

3.2 Montgomery Modular Reduction Process at Radix 4 50

3.3 Montgomery Modular Reduction Process at Radix 8 ($l = \lceil \frac{n}{k} \rceil$) 52

3.4 Montgomery Modular Reduction Process at Radix 8 ($l = \lfloor \frac{n}{k} \rfloor$) 53

3.5 The k-boundary of Montgomery Reduction for n from 12 to 32 61

3.6 Shortest Delays of Separated Montgomery Multiplication Algorithm using Trivial QDS at n from 12 to 32 61

3.7 k and d in Quotient Digit Pipelining of Montgomery Algorithm for n from 12 to 25 66

3.8 Shortest Delays of Separated Montgomery Multiplication Algorithm using Quotient Digit Pipelining at n from 13 to 32 . . 67

4.1 A Comparison of 6 Short Word length RNS Scaling Schemes using LUTs 92

4.2 An Example of Pre-Computed Parameters for the Classical Modular Reduction Algorithm in RNS using the Core Function 106

4.3 An Example of the Classical Modular Reduction Algorithm in RNS using the Core Function for a case in which $\lfloor \text{Mag}(X) \rfloor \geq 1$ 107

4.4 An Example of the Classical Modular Reduction Algorithm in RNS using the Core Function for a case in which $\lfloor \text{Mag}(X) \rfloor < 1$ 108

4.5 The Maximum Possible N against w in RNS Sum of Residues Reduction 113

5.1 The Pre-Computed Constants for the RNS Sum of Residues Modular Multiplication Algorithm in Algorithm 4.2 144

Chapter 1

Introduction

The purpose of this chapter is to outline the thesis by chapters and present the contribution this thesis makes.

1.1 Thesis Outline

In the information age, *cryptography* has become a cornerstone for information security. Cryptography allows people to carry established notions of trust from the physical world to the electronic world; it allows people to do business electronically without worries of deceit and deception; and it establishes trust for people through open, standards-based, security technology that withstands the test of time.

In the distant past, cryptography was used to assure only secrecy. Wax seals, signatures, and other physical mechanisms were typically used to assure integrity of the message and authenticity of the sender. When people started doing business online and needed to transfer funds electronically, the use of cryptography for integrity began to surpass its use for secrecy. Hundreds of thousands of people interact electronically every day, whether it is through e-mail, e-commerce, ATM machines, or cellular phones. The constant increase of information transmitted electronically has led to an increased reliance on the transport of trust made possible by *public-key cryptography*, a mechanism for secret communication between parties who have never before exchanged a secret message.

One can argue that *public-key cryptosystems* become more secure as the hardware used to perform cryptography increases in speed. Take the *RSA cryptosystem* as an example [Rivest78]. The effort of cracking the RSA code, via factorization of the product of two large primes, approximately doubles for every 35 bits at key lengths around 2^{10} bits [Crandall01]. However, adding 35 bits to the key increases the work involved in decryption by only 10%! Thus speeding up the cryptography hardware by just 10% enables the use of a cryptosystem that is twice as strong with no compromise in performance [Koblitz87]. Speed, therefore, is an important goal for public-key cryptosystems. Indeed it is essential not just for cryptographic strength but also to clear the large number of transactions performed by central servers in electronic commerce systems.

This thesis contributes to the modular multiplication operation $Z = A \times B \mod M$, the basis of many public-key cryptosystems including *RSA* [Rivest78] and *Elliptic Curve Cryptography* (ECC) over a prime finite field [Hankerson04]. The *Residue Number System* (RNS) [Omondi07] is used to speed up long word length modular multiplication.

Chapter 2 provides the background to the Residue Number System and the RSA cryptosystem.

Chapter 3 defines a new classification of current modular multiplication algorithms in a positional number system. The four classes in this definition are Classical, Sum of Residues, Barrett and Montgomery. Further developments are made to these algorithms and implementations are prepared for the modular multiplications within RNS channels that will appear in Chapter 5.

Chapter 4 is the core of this thesis. Firstly it surveys existing modular multiplication algorithms in the RNS. Most of these schemes are designed for short word length operands and hence are not applicable in public-key cryptosystems that require long word length moduli. One reason for this incompatibility is that their architectures are typically based on Look-Up Tables (LUT). For long word length operands, these tables become infeasibly large. However, they are still useful for other applications of RNS, e.g. Digital Signal Processing (DSP). A new scheme of this kind is then derived and decreases hardware complexity compared to previous schemes without affecting time complexity.

Chapter 4 then explores the possibility that each of the four classes of modular multiplication algorithms for positional number systems can be adopted for modular multiplication in the RNS. The goal is a long word length operation that is faster than working in a positional number system. Ideally all intermediate processes will be short word length operations entirely within RNS channels.

Chapter 5 illustrates a highly parallel and scalable architecture for the RNS sum of residues modular multiplication algorithm discussed in Chapter 4. This architecture is then implemented on a Xilinx Virtex5 FPGA platform using results from Chapter 3 for RNS channel operations. The delay for a 64-bit RNS modular multiplier is 81.7 ns. 0.4 μs and 0.73 μs are required to perform 1024-bit and 2048-bit modular multiplication respectively. The corresponding speed of a 1024-bit RSA system is 0.4 ms per decryption.

Chapter 6 concludes the work and suggests some areas for further study.

1.2 Contribution

The following contributions are presented in this thesis:

- A new classification of existing modular multiplication algorithms in a positional number system. New developments are made and improved algorithms are implemented on FPGA platforms.
- Existing algorithms performing short word length modular multiplication in RNS are characterized in terms of Look-Up Cycles (LUC) and Look-Up Tables (LUT) within a consistent framework. A new LUT-based algorithm that is applicable in DSP achieves less hardware complexity compared to existing algorithms.
- New algorithms are developed for Classical, Sum of Residues and Barrett classes for a long word length modular multiplication in the Residue

Number System. All the intermediate operations are kept within RNS channels, which leads to a highly parallel architecture performing a fast long word length modular multiplication in the RNS in hardware.

- The fastest ever FPGA implementation of the 1024-bit RSA cryptosystem. It achieves 0.4 ms per decryption.

Chapter 2

Background

The purpose of this chapter is to introduce the Residue Number System and RSA cryptography. The Residue Number System is the main mathematical tool used to improve the performance of public-key cryptosystems in this thesis. The background on the Residue Number System is the basis of technical developments made in later chapters, particularly in Chapter 4. The introduction of RSA cryptography aims to show the significance of improving the efficiency of the long word length modular multiplication operation.

2.1 Residue Number Systems

2.1.1 RNS Representation

What number has the remainders of 2, 3 and 2 when divided by the numbers 3, 5 and 7 respectively? This question is probably the first documented use of residue arithmetic in representing numbers, recorded by a Chinese Scholar Sun Tsu over 1500 years ago [Jenkins93]. The question basically asks us to convert the coded representation $\{2, 3, 2\}$ in a residue number system, based on the moduli $\{3, 5, 7\}$, back to its normal decimal format.

A Residue Number System [Szabo67] is characterized by a set of N co-prime moduli $\{m_1, m_2, \ldots, m_N\}$ with $m_1 < m_2 < \cdots < m_N$. In the RNS a number X is represented in N channels: $X = \{x_1, x_2, \ldots, x_N\}$, where x_i is the residue of X with respect to m_i, i.e. $x_i = \langle X \rangle_{m_i} = X \mod m_i$. Hence a RNS has digit sets $\{[0, m_1 - 1]$, $[0, m_2 - 1]$, $[0, m_3 - 1]$, $\ldots$,$[0, m_N - 1]\}$ in its N channels. For example, in the RNS with the modulus set $\{3, 5, 7\}$ above, $X = \{2, 3, 2\}$. The word length of the largest channel m_N is defined as the *channel width* w.

Within a RNS: there is a unique representation of all integers in the range $0 \leq X < D$ where $D = m_1 m_2 \ldots m_N$. Also, D is the number of different representable values in the RNS and is therefore known as the *dynamic range* of the RNS. Hence $D = 3 \times 5 \times 7 = 105$ is the total number of distinct values that are representable in the example above. These 105 available values can be used to represent numbers 0 through 104, -52 through +52 or any other interval of 105 consecutive integers. This thesis considers the non-negative numbers from 0 to $D - 1$ only, as the RNS is used for positive modular arithmetic.

There are two other important values in a RNS, D_i and $\langle D_i^{-1} \rangle_{m_i}$. $D_i =$

Table 2.1: The Pre-Computed Constants of RNS $\{3, 5, 7\}$

N	3		
D	105		
i	1	2	3
m_i	3	5	7
D_i	35	21	15
$\langle D_i \rangle_{m_i}$	2	1	1
$\langle D_i^{-1} \rangle_{m_i}$	2	1	1

$\frac{D}{m_i}$ and $\langle D_i^{-1} \rangle_{m_i}$ is its multiplicative inverse with respect to m_i such that $\langle D_i \times D_i^{-1} \rangle_{m_i} = 1$. These values are constants for a particular modulus set and can be pre-computed before any actual computations are performed in a RNS. For example, in RNS $\{3, 5, 7\}$, the D_i set is $\{35, 21, 15\}$ and the $\langle D_i^{-1} \rangle_{m_i}$ set is $\{2, 1, 1\}$. Thus, each RNS has a table of known constants that might be useful in future computations. Table 2.1 lists some of the constants of RNS $\{3, 5, 7\}$.

2.1.2 Conversion between RNS and Positional Number Systems

Conversion from Binary to RNS

The binary-to-RNS conversion is quite a simple problem compared with the conversion in the opposite direction. All that is required is to find the residue of $X \mod m_i$ for each modulus. Computing $X \mod m_i$ for binary values X and m is a typical modular reduction problem and will be discussed in detail in Chapter 3.

For example, if $X = 23$ is to be converted into its RNS format based on the moduli $\{3, 5, 7\}$, then $x_1 = \langle 23 \rangle_3 = 2$, $x_2 = \langle 23 \rangle_5 = 3$ and $x_3 = \langle 23 \rangle_7 = 2$.

Conversion from RNS to Positional Number Systems using the Mixed Radix Number System (MRS)

Associated with any RNS $\{m_1, m_2, \ldots, m_N\}$ is a *Mixed Radix Number System* (MRS) $\{m_1, m_2, \ldots, m_N\}$ [Soderstrand86], which is an N-digit positional number system with position weights

$$1,\ m_1,\ m_1m_2,\ m_1m_2m_3,\ \ldots\ , m_1m_2m_3\ldots m_{N-1}$$

and digit sets $\{[0, m_1-1],\ [0, m_2-1],\ [0, m_3-1],\ \ldots,\ [0, m_N-1]\}$ in its N digit positions. Hence, the MRS digits are in the same ranges as the RNS residues. For example, the MRS $\{3, 5, 7\}$ has position weights 1, 3 and $3 \times 5 = 15$.

The RNS-to-MRS conversion is to determine the g_i digits of MRS, given the x_i digits of RNS, so that $\{g_1, g_2, \ldots, g_N\} = \{x_1, x_2, \ldots, x_N\}$. From the definition of MRS, we have

$$X = g_1 + g_2m_1 + g_3m_1m_2 + \cdots + g_Nm_1m_2m_3\ldots m_{N-1}.$$

A modular reduction, with respect to m_1, of both sides of this equation yields

$$\langle X\rangle_{m_1} = g_1.$$

Hence $g_1 = x_1$. Subtracting x_1 and dividing both sides by m_1 yields

$$\frac{X - g_1}{m_1} = g_2 + g_3m_2 + \cdots + g_Nm_2m_3\ldots m_{N-1}.$$

Now reduction modulo m_2 yields g_2 as

$$g_2 = \langle (X - g_1)m_1^{-1}\rangle_{m_2} = \langle (x_2 - g_1)m_1^{-1}\rangle_{m_2}$$

because the equality $\langle \frac{X-g_1}{m_1}\rangle_{m_2} = \langle (X-g_1)m_1^{-1}\rangle_{m_2}$ holds according to a property in number theory [Richman71]. A prerequisite for this is that m_1 and

m_2 must be co-prime so that $\langle m_1^{-1} \rangle_{m_2}$ exists. This is exactly what happens in a RNS.

Following a similar procedure,

$$\frac{\frac{X-g_1}{m_1} - g_2}{m_2} = g_3 + g_4 m_3 + \cdots + g_N m_3 \ldots m_{N-1},$$

g_3 can be obtained as

$$g_3 = \langle ((X - g_1)m_1^{-1} - g_2)m_2^{-1} \rangle_{m_3} = \langle ((x_3 - g_1)m_1^{-1} - g_2)m_2^{-1} \rangle_{m_3}.$$

Continuing this process, the mixed radix digits, g_i, can be retrieved from RNS residues as

$$\begin{aligned}
g_1 &= x_1 \\
g_2 &= \langle (x_2 - g_1)m_1^{-1} \rangle_{m_2} \\
g_3 &= \langle ((x_3 - g_1)m_1^{-1} - g_2)m_2^{-1} \rangle_{m_3} \\
&\ldots \\
g_N &= \langle ((\ldots((x_N - g_1)m_1^{-1} - g_2)m_2^{-1} \ldots)m_{N-2}^{-1} - g_{N-1})m_{N-1}^{-1} \rangle_{m_N}
\end{aligned}$$

All the $\langle m_i^{-1} \rangle_{m_j}$ are constants that can be pre-computed. Figure 2.1 illustrates this process implemented using $\frac{N(N-1)}{2}$ modular blocks. Each block performs an operation like $\langle (x_j - g_i)m_i^{-1} \rangle_{m_j}$. As can be seen from Figure 2.1, the disadvantage of this algorithm is the sequentiality of the structure. The computation of g_i has to wait for the result of g_{i-1}.

Take $X = \{2, 3, 2\}$ in RNS $\{3, 5, 7\}$ as an example. Pre-computed constants include

$$\begin{aligned}
\langle m_1^{-1} \rangle_{m_2} &= 2, \\
\langle m_1^{-1} \rangle_{m_3} &= 5, \\
\text{and } \langle m_2^{-1} \rangle_{m_3} &= 3.
\end{aligned}$$

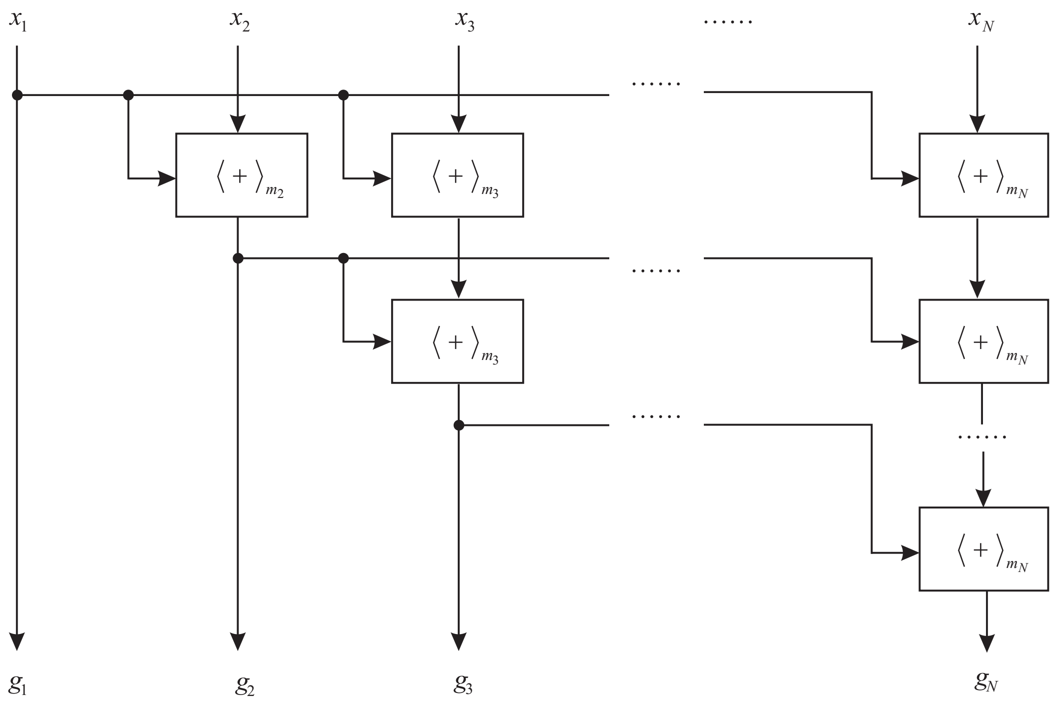

x_1
x_2
x_3
……
x_N
$\langle + \rangle_{m_2}$
$\langle + \rangle_{m_3}$
$\langle + \rangle_{m_N}$
$\langle + \rangle_{m_3}$
$\langle + \rangle_{m_N}$
$\langle + \rangle_{m_N}$
g_1
g_2
g_3
g_N

Figure 2.1: Conversion from the RNS to the Mixed Radix Number System (MRS)

Then the corresponding MRS digits are

$$\begin{aligned} g_1 &= x_1 = 2 \\ g_2 &= \langle (x_2 - g_1) m_1^{-1} \rangle_{m_2} = \langle (3-2) \times 2 \rangle_5 = 2 \\ g_3 &= \langle ((x_3 - g_1) m_1^{-1} - g_2) m_2^{-1} \rangle_{m_3} = \langle ((2-2) \times 5 - 2) \times 3 \rangle_7 = 1 \end{aligned}$$

Therefore {2,2,1} in MRS is equal to {2,3,2} in RNS, and the decimal form can be obtained by computing an inner product of the MRS digit set and weight set as

$$X = 2 \times 1 + 2 \times 3 + 1 \times 15 = 23.$$

Conversion from RNS to Binary using Chinese Remainder Theorem (CRT)

Instead of deriving the mixed radix representation of a RNS and then using the weights in the MRS to complete the conversion, the position weights for a RNS can be directly derived using the *Chinese Remainder Theorem* (CRT). The CRT is important in residue arithmetic as it is the basis of almost all of the operations of a RNS, including conversion to binary, scaling and comparison [Szabo67, Elleithy91]. It is also referred to in later sections of this thesis such as in Section 4.1.2, 4.2.1 and 4.4.1. Using the CRT, an integer X can be expressed as

$$X = \left\langle \sum_{i=1}^{N} D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_D , \tag{2.1}$$

where D, D_i and $\langle D_i^{-1} \rangle_{m_i}$ are pre-computed constants introduced in Section 2.1.1.

For $X = \{2,3,2\}$ in RNS $\{3,5,7\}$, the pre-computed constants are

$$\begin{aligned} D &= 3 \times 5 \times 7 = 105, \\ D_i &= \{35, 21, 15\}, \\ \text{and } \langle D_i^{-1} \rangle_{m_i} &= \{2, 1, 1\}. \end{aligned}$$

Then following Equation (2.1),

$$\begin{aligned} X &= \left\langle \sum_{i=1}^{3} D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_D \\ &= \langle 35 \times \langle 2 \times 2 \rangle_3 + 21 \times \langle 1 \times 3 \rangle_5 + 15 \times \langle 1 \times 2 \rangle_7 \rangle_{105} = 23. \end{aligned}$$

The disadvantage of the CRT is its dependence on a modulo D operation [Omondi07]. Given that D is a large integer, this reduction incurs a significant hardware overhead. This will be discussed in Chapter 4 in more detail.

2.1.3 RNS Arithmetic

If A, B and C have RNS representations given by $A = \{a_1, a_2, \ldots, a_N\}$, $B = \{b_1, b_2, \ldots, b_N\}$ and $C = \{c_1, c_2, \ldots, c_N\}$, then denoting $*$ to represent the operations +, -, or $\times$, the RNS version of $C = \langle A * B \rangle_D$ satisfies

$$C = \{\langle a_1 * b_1 \rangle_{m_1}, \langle a_2 * b_2 \rangle_{m_2}, \ldots, \langle a_N * b_N \rangle_{m_N}\}.$$

Again take the simple RNS $\{3,5,7\}$ as an example. If $A = 23 = \{2,3,2\}$ and $B = 40 = \{1,0,5\}$, then

$$\begin{aligned} C_{sum} &= \langle A + B \rangle_D = \{\langle 2+1 \rangle_3, \langle 3+0 \rangle_5, \ldots, \langle 2+5 \rangle_7\} = \{0,3,0\} \\ &= \langle 23 + 40 \rangle_{105} = \langle 63 \rangle_{105} = 63, \qquad (2.2) \\ C_{diff} &= \langle A - B \rangle_D = \{\langle 2-1 \rangle_3, \langle 3-0 \rangle_5, \ldots, \langle 2-5 \rangle_7\} = \{1,3,4\} \\ &= \langle 23 - 40 \rangle_{105} = \langle -17 \rangle_{105} = 88, \qquad (2.3) \\ \text{and } C_{prod} &= \langle A \times B \rangle_D = \{\langle 2 \times 1 \rangle_3, \langle 3 \times 0 \rangle_5, \ldots, \langle 2 \times 5 \rangle_7\} = \{2,0,3\} \\ &= \langle 23 \times 40 \rangle_{105} = \langle 920 \rangle_{105} = 80. \qquad (2.4) \end{aligned}$$

The correct values of $A - B$ and $A \times B$ in (2.3) and (2.4) are -17 and 920 respectively; however these overflow the dynamic range [0, 104], and so they have to be reduced modulo $D = 105$ to form the results representable by RNS. Therefore, the equality $C = A * B$ holds if it can be assured that $A * B$ falls in the dynamic range D as in (2.2).

Thus addition, subtraction and multiplication can be concurrently performed on the N residues within N parallel channels, and it is this high speed parallel operation that makes the RNS attractive. There is, however, no such parallel form of modular reduction within RNS and this problem has long prevented wider adoption of RNS. Hence, this becomes the main issue of this thesis. As an aside, note that a Maple package to perform basic RNS arithmetic operations was developed to support the research described in this thesis. This tool, RNSpack, can be found at:
`http://gforge.eleceng.adelaide.edu.au/gf/project/rnspack/`

2.1.4 Moduli Selection

RNS moduli need to be co-prime. Hence one common practice is to select prime numbers for the RNS moduli. Sometimes, however, a set of non-prime numbers can also be co-prime, and therefore, RNS moduli selection becomes a case-specific problem.

In our application, RNS is used to accelerate a 1024-bit modular multiplication. This means that binary inputs to the RNS are all 1024 bits. Therefore, the dynamic range D of the RNS should be no smaller than 2048 bits so that the product of two 1024-bit numbers does not overflow.

The other rule to be considered is the even distribution of this 2048-bit dynamic range into the N moduli. The smaller the RNS channel width w, the faster the computation within RNS and the more remarkable the advantage of RNS. Therefore, we want w as small as possible. On the other hand, suppose the N RNS moduli are $m_1, m_2, \ldots, m_N$. If m_1 is 16 bits and m_2 is 64 bits long, the computation in the m_2 channel can be much slower than in the m_1 channel. Thus, in this thesis, the N moduli are selected to be the same word length. This means that the dynamic range of the RNS system is evenly distributed into the N moduli.

The remaining work is to make sure N co-prime w-bit moduli exist. For example, suppose $w = 9$, $N = \lceil \frac{2048}{9} \rceil = 228$, i.e. 228 co-prime moduli must be found within the range from $2^8 = 256$ to $2^9 - 1 = 511$. Because there are only 128 odd integers from 256 to 511, it is impossible to find 228 co-prime numbers. Now, $w = 10$. $N = \lceil \frac{2048}{10} \rceil = 205$ channels and these 205 co-prime moduli should be within the range from $2^9 = 512$ to $2^{10} - 1 = 1023$, included. However, according to our search algorithm, there are only 83 co-prime numbers found among the 256 odd numbers within [512, 1023], so $w = 10$ is invalid. Similarly when $w = 11$. For the case of $w = 12$, a set of 268 co-prime numbers have been found within $[2^{11}, 2^{12} - 1] = [2048, 4095]$, in which there are 255 primes. Thus even the number of primes is enough for the requirement of the number of moduli, $N = \lceil \frac{2048}{12} \rceil = 171$.

In addition, redundancy in dynamic range, i.e. more redundant channels, is always required in building a RNS system. Consequently, to construct a RNS system with 2048-bit dynamic range and equal word length moduli, the minimal channel width w is very likely to be 12 bits.

2.1.5 Base Extension

So far it has been assumed that once the modulus set has been determined, all operations are carried out with respect to that set only. This is not always so. A frequently occurring computation is that of *base extension*(BE), which is defined as follows [Omondi07]. Given a residue representation $\{x_1, x_2, \ldots, x_N\}$ in RNS $\{m_1, m_2, \ldots, m_N\}$ and an additional set of moduli, $\{m_{N+1}, m_{N+2}, \ldots, m_{N+s}\}$, such that $m_1, m_2, \ldots, m_N, m_{N+1}, \ldots, m_{N+s}$ are all co-prime, the residue representation $\{x_1, x_2, \ldots, x_N, x_{N+1}, \ldots, x_{N+s}\}$ is to be computed. This is also described as "base extend X to the new modulus set $m_1, m_2, \ldots, m_N, m_{N+1}, \ldots, m_{N+s}$". Base extension is very useful in dealing with difficult operations of conversion to positional systems, scaling, division and dynamic range extension.

Efficient algorithms for base extension are presented in [Szabo67, Barsi95, Shenoy89b] and [Posch95]. The scheme in [Szabo67] uses the MRS conversion introduced in Section 2.1.2 which is relatively slow and costly; [Shenoy89b] employs two extra RNS channels with moduli both greater than N; [Posch95] performs an approximate extension; and [Barsi95] achieves exact scaling without any extra RNS channel but can be as slow as the MRC in some rare cases. These algorithms are discussed in further detail in Section 4.1, 4.4 and 4.5.

2.2 RSA Public-Key Cryptography

2.2.1 Public-Key Cryptography

Public-key cryptography is a mechanism for secret communication between parties who have never before exchanged a secret message [Schneier96]. Its counterpart is *private-key cryptography*, in which both sender and receiver must have knowledge of a shared private key and must have firstly secretly exchanged the private key before exchanging secret messages. Public-key cryptography eliminates this initial secret exchange step.

A very important application of public-key cryptography is *key exchange* for a private-key system. The first example of public-key cryptography, published by Diffe and Hellman in 1976 [Diffie76], was in fact a key exchange scheme. Their public-key distribution system allows two parties to exchange a private key over a public channel. In this way a public-key system is used to initiate a private-key system, which is subsequently used to encrypt data. This is a good combination of private and public-key systems in that private-key systems are usually simpler and faster than public-key systems at a given level of security.

The generation of digital signatures is another significant application of public-key cryptography. Like a conventional signature, a digital signature is affixed to a message to prove the identity of its sender. A public-key digital signature, however, has many more merits apart from this [Boyd93]. These include: integrity (which makes a signature unique for a particular document), concurrence (which provides proof that two parties were in agreement or disagreement), nonrepudiation (which prevents denial that a message was sent or that it was received) and timeliness (which provides proof of the time of a message's transmission). All of these properties are of great importance in applications such as electronic commerce transactions.

2.2.2 The RSA Cryptosystem

The *RSA cryptosystem* [Rivest78] is a simple and widely used public-key cryptosystem. It is based on one of the mathematical trapdoor problems: it is easy to multiply two large primes but hard to factorize the product.

The RSA cryptosystem has been described many times in literature with a tradition that a couple, Alice and Bob, act as the secretive pair exchanging messages. Now let us also begin the description of RSA with these conspirators.

Before Alice starts receiving any messages, she has to establish her private and public keys. This is done by selecting two large prime numbers, U and V, of approximately equal length such that their product $M = U \times V$ has a required n-bit word length. Alice must also choose an integer e which is less than M and relatively prime to $(U-1)(V-1)$. Alice's public key consists of two numbers, M and e. Finally Alice must compute her private key, the modular inverse $k = e^{-1} \mod (U-1)(V-1)$ and either dispose of the prime factors U and V or keep them secret.

To send a secret message to Alice, Bob performs the encryption

$$G = F^e \mod M,$$

where F is the n-bit plaintext message and G is the cyphertext. To decrypt this message, Alice computes

$$F = G^k \mod M.$$

Note that the encryption uses the public key pair only. This means anyone can send Alice a message. Decryption requires the knowledge of the private key k, which means only Alice can see the plaintext of messages sent to her.

The generation of a cryptographic signature is a reverse of the process above. This is because the nature of a signature requires that anyone can verify that the signature is correct, yet only the holder of the private key can generate a signature. Since Alice is the "private key holder" in the case above, she can sign a message by performing an operation equivalent to decryption,

$$S = F^k \mod M,$$

where F is the n-bit message to be signed and S is her signature. Bob can verify the signature S by encrypting it with the public key and comparing the result with the original message F using

$$F = S^e \mod M.$$

As can be seen from these equations, only one kind of operation is involved in RSA encryption and decryption, namely modular exponentiation. Modular exponentiation is equivalent to a series of modular multiplications [Knuth69]. Therefore, the performance of RSA depends largely on the efficiency of the modular multiplication operation. Most practical RSA systems today use keys which are 1024 bits long [Schneier96]. Hence the focus of this thesis, is to improve 1024-bit modular multiplication.

2.2.3 Exponentiation

Modular exponentiation is classified as *single exponentiation* and *multi-exponentiation*. RSA cryptography uses single exponentiation

$$C = A^B \mod M.$$

Multi-exponentiation takes the form

$$C = \prod_{i=1}^{l} A_i^{B_i} \mod M.$$

This is also widely used in public-key cryptosystems. For example, the product of two modular exponentiations are needed in the digital signature proposed by Brickell and McCerley in [Brickell91], the DSS standard [Standards91], and Schnorr's identity verification of smart card [Schnorr89]. Three modular exponentiations are used in ElGamal's scheme [ElGamal85].

Modular exponentiation can be performed using repeated modular multiplications in the same way as exponentiation can be performed with repeated multiplications. Algorithms for multi-exponentiation appear in [ElGamal85, Chang94, Yen93] and [Yen94]. Fewer new algorithms for single exponentiation have appeared in recent years. This is probably because single modular exponentiation seems to be 'sequential' in nature, and using extra hardware does not seem to help much [Rivest85]. It is possible to prove that $\log_2 B$ is a lower bound on the number of multiplications required to evaluate A^B [Knuth97]. [Knuth97, Menezes97] and [Dhem98] provide good surveys for modular exponentiations. The following subsections introduce the two well-known methods for single exponentiation in both hardware and software: the left-to-right and right-to-left methods. The focus is on single modular exponentiation as this is required for RSA. Note, however, that modular multiplication is the basis of both single and multi-exponentiation.

Left-to-Right Exponentiation

The left-to-right algorithm [Knuth97] is described in Algorithm 2.1.

Evaluation requires $n-1$ modular squares (the very first modular square is trivial) and $nz(B)-1$ modular multiplications, where $nz(B)$ is the number of non-zero bits in B. Take $B = 23 = (10111)_2$ as an example and $nz(B) = 4$. The process is illustrated in Equation (2.5).

$$1 \rightarrow A \rightarrow A^2 \rightarrow (A^4 \times A)^2 \rightarrow (A^{10} \times A)^2 \rightarrow A^{22} \times A = A^{23}, \tag{2.5}$$

Algorithm 2.1 Left-to-Right Exponentiation

```
Ensure: $C \equiv A^B \mod M$, $B$ is $n$-bit long
  $C = 1$
  for $i = n - 1$ downto 0 do
    $C = C \times C \mod M$
    if $b_i = 1$ then
      $C = C \times A \mod M$
    end if
  end for
```

where 4 modular squares and 3 modular multiplications are used.

Right-to-Left Exponentiation

The right-to-left algorithm [Knuth97] is described in Algorithm 2.2.

Algorithm 2.2 Right-to-Left Exponentiation

```
Ensure: $C \equiv A^B \mod M$, $B$ is $n$-bit long
  $C = 1$
  $D = A$
  for $i = n - 1$ downto 0 do
    if $b_i = 1$ then
      $C = C \times D \mod M$
    end if
    $D = D \times D \mod M$
  end for
```

Again $n - 1$ modular squares and $nz(B) - 1$ modular multiplications are required and this is the same as the left-to-right algorithm. However, this algorithm has an advantage when parallel hardware is available: the square and multiplication can be performed in parallel [Rivest85, Orup91, Chiou93].

Table 2.2: An Example of Right-to-Left Modular Exponentiation

i	b_i	C	D
		1	A
0	1	$1 \times A$	A^2
1	1	$A \times A^2$	A^4
2	1	$A^3 \times A^4$	A^8
3	0	A^7	A^{16}
4	1	$A^7 \times A^{16} = A^{23}$	

In Algorithm 2.1, when $b_i = 1$, the statement $C = C \times A \mod M$ needs the result of C from the previous operation $C = C \times C \mod M$ to continue, and so these two statements have to be performed in series. Thus, the right-to-left algorithm has a shorter delay within each loop than the left-to-right algorithm though they both have the same number of loops. [Chiou93] gave an implementation using this to obtain a speed benefit of 30%.

The example of $B = 23 = (10111)_2$ is illustrated in Table 2.2.

Now that we have algorithms for modular exponentiation, our attention turns to modular multiplication, the major topic of this thesis.

Chapter 3

Four Ways to Do Modular Multiplication in a Positional Number System

The purpose of this chapter is to survey existing modular multiplication algorithms in a positional number system. They are into 4 new classes: Classical, Sum of Residues, Barrett and Montgomery. Each of the class of algorithm is analyzed in detail and further developments are made to the Montgomery, Sum of Residues and Barrett algorithm. Implementations are also prepared for the channel modular multipliers within the long wordlength RNS modular multiplication algorithm implemented in Chapter 5.

3.1 Introducing Four Ways to Do Modular Multiplication

To develop a first algorithm for modular multiplication, let us begin with a simple schoolbook multiplication. Consider two integers A and B with digits a_i and b_i in radix r:

$$\begin{aligned} A &= \sum_{i=0}^{l} a_i r^i \\ B &= \sum_{i=0}^{l} b_i r^i \end{aligned}$$

Then a simple multiplication is computed in Equation (3.1).

$$\begin{aligned} C &= A \times B \\ &= b_l A r^l + b_{l-1} A r^{l-1} + \cdots + b_1 A r + b_0 A \\ &= (\ldots((b_l A r + b_{l-1} A) r b_{l-2} A) r + \cdots + b_1 A) r + b_0 A. \end{aligned} \tag{3.1}$$

A and B are both l-digit integers and the product C is $2l$ digits. Equation (3.1) forms an algorithm suitable for an iterative implementation as illustrated in Algorithm 3.1. Note that as the loop is from $l-1$ down to 0, C_{i+1} denotes the value C_i in the previous iteration.

Algorithm 3.1 A Basic Multiplication

```
Ensure: $C_0 = A \times B$
  $C_l = 0$
  for $i = l - 1$ to 0 do
    $C_i = b_i A + r C_{i+1}$ {Partial product accumulation}
  end for
```

Now the modular reduction step has to be included so that we perform a modular multiplication as in Equation (3.2)

$$C = A \times B \mod M \tag{3.2}$$

One way is to complete the multiplication and then modulo reduce the result in a separate step as in Equation (3.3) and Algorithm 3.2.

$$\begin{aligned} C &= A \times B \mod M \\ &= (\ldots((b_l A r + b_{l-1} A) r + b_{l-2} A) r \\ &\quad + \cdots + b_1 A) r + b_0 A \mod M \end{aligned} \tag{3.3}$$

Algorithm 3.2 Separated Modular Multiplication

```
Ensure: $C_0 = A \times B \mod M$
  $C_l = 0$
  for $i = l - 1$ to 0 do
    $C_i = b_i A + r C_{i+1}$ {Partial product accumulation}
  end for
  $C_0 = C_0 \mod M$ {Modular reduction step}
```

The other way is to place the modular reduction step inside every pair of brackets as in Equation (3.4). This corresponds to the insertion of the statement $C_i = C_i \mod M$ into the end of each iteration in Algorithm 3.1 and results in Algorithm 3.3, an *interleaved modular multiplication* algorithm.

$$\begin{aligned} C &= A \times B \mod M \\ &= (\ldots((b_l A r + b_{l-1} A \mod M) r + b_{l-2} A \mod M) r \\ &\quad + \cdots + b_1 A \mod M) r + b_0 A \mod M \end{aligned} \tag{3.4}$$

3.1.1 Classical Modular Multiplication

The way in which the modular reduction step in Algorithm 3.3, $C_i = C_i \mod M$, is implemented defines the class of modular multiplication algorithm. For a *Classical* algorithm, this step is performed by subtracting a

Algorithm 3.3 Interleaved Modular Multiplication

```
Ensure: $C_0 = A \times B \mod M$
  $C_l = 0$
  for $i = l - 1$ to 0 do
    $C_i = b_i A + rC_{i+1}$ {Partial product accumulation}
    $C_i = C_i \mod M$ {Modular reduction step}
  end for
```

multiple of the modulus at each iteration as shown in Algorithm 3.4, where QDS stands for *quotient digit selection*.

Algorithm 3.4 Classical Modular Multiplication

```
Ensure: $C_0 = A \times B$
  $C_l = 0$
  for $i = l - 1$ to 0 do
    $C_i = b_i A + rC_{i+1}$ {Partial product accumulation}
    $q = \mathrm{QDS}(C_i, M)$ {Quotient digit selection}
    $C_i = C_i - qM$ {Reduction step}
  end for
```

Papers that take this approach include [Blakley83, Brickell83, Orup91] and [Walter92]. Reduction in this way can be understood as a division in which the quotient is discarded and the remainder retained. Development of modular multipliers along this line has closely followed the development of division, especially SRT division (as originally in [Robertson58]).

The quotient digit selection function (QDS) has received a great deal of attention to: permit quotient digits (q) to be trivially estimated from only the most significant bits of the partial result C; allow the partial result to be stored in a redundant form; and move the QDS function from the critical path (e.g. [Orup91, Walter92]).

The two methods from [Orup91, Walter92] were combined and implemented in Section 3.5 for comparison.

3.1.2 Sum of Residues Modular Multiplication

Some early modular multipliers [Kawamura88, Findlay90, Tomlinson89, Su96, Chen99] perform the modular reduction step in Algorithm 3.3 by accumulating residues modulo M instead of by adding or subtracting multiples of the modulus as in the case for Classical or Montgomery modular multiplication. This results in the *Sum of Residues* (SOR) algorithm shown in Algorithm 3.5, in which the residues $q \times r^{l+1} \mod M$ are accumulated modulo M. They may be pre-computed and retrieved from a table ([Kawamura88, Tomlinson89]) or evaluated recursively during the modular multiplication ([Findlay90]).

Algorithm 3.5 Sum of Residues Modular Multiplication

```
Ensure: $C_0 = A \times B \mod M$
  $C_l = 0$
  $q = 0$
  for $i = l - 1$ to 0 do
    $C_i = b_i A + rC_{i+1}$ {Partial product accumulation}
    $C_i = C_i + (q \times r^{l+1} \mod M)$ {Residues accumulation}
    $q = \lfloor \frac{C_i}{r^l} \rfloor$
    $C_i = C_i - q \times r^l$ {Quotient digit selection and reduction}
  end for
  $C_0 = (C_0 + (q \times r^{l+1} \mod M)) \mod M$
```

Section 3.2 examines this algorithm in further detail and presents some improvements to it.

3.1.3 Barrett Modular Multiplication

The relationship between division and modular multiplication is made explicit in Equation (3.5).

$$C = A \times B \mod M = A \times B - \left\lfloor \frac{A \times B}{M} \right\rfloor \times M. \tag{3.5}$$

A and B are two n-bit multiplicands and M denotes the n-bit modulus. This equation suggests an alternative mechanism: one may perform the division $\lfloor \frac{A \times B}{M} \rfloor$ by multiplying by M^{-1}. Papers that follow this line include [Barrett87, Walter94] and [Dhem98]. A typical example is the improved Barrett algorithm based on [Barrett84], developed in [Barrett87] and improved in [Dhem98] by introducing more variable parameters.

The *Improved Barrett* modular multiplication described in [Dhem98] is a separated modular multiplication scheme in which the product $A \times B$ is modular reduced using multiplication by a pre-computed inverse of the modulus. If we let $C_0 = A \times B$ and $Y = \lfloor \frac{A \times B}{M} \rfloor = \lfloor \frac{C_0}{M} \rfloor$, Equation (3.5) becomes

$$C = A \times B \mod M = C_0 - Y \times M. \tag{3.6}$$

The advantage of Improved Barrett modular multiplication lies in the fast computation of Y as

$$Y = \left\lfloor \frac{C_0}{M} \right\rfloor = \left\lfloor \frac{\frac{C_0}{2^{n+v}} \frac{2^{n+u}}{M}}{2^{u-v}} \right\rfloor,$$

where u and v are two parameters. Furthermore, the quotient Y can be estimated with an error of at most 1 from

$$\hat{Y} = \left\lfloor \frac{\left\lfloor \frac{C_0}{2^{n+v}} \right\rfloor \left\lfloor \frac{2^{n+u}}{M} \right\rfloor}{2^{u-v}} \right\rfloor.$$

The value $K = \left\lfloor \frac{2^{n+u}}{M} \right\rfloor$ is a constant and can be pre-computed. The algorithm is shown in Algorithm 3.6.

Algorithm 3.6 Improved Barrett Modular Multiplication

```
Require: $u$, $v$ {Pre-defined parameters}
Require: $K = \left\lfloor \frac{2^{n+u}}{M} \right\rfloor$ {A pre-computed constant}
Ensure: $C \equiv A \times B \mod M$
  $C_0 = A \times B$
  $C_1 = \left\lfloor \frac{C_0}{2^{n+v}} \right\rfloor$ {Right shift by $n + v$}
  $C_2 = C_1 \times K$
  $Y = \left\lfloor \frac{C_2}{2^{u-v}} \right\rfloor$ {Right shift by $u - v$}
  $C = C_0 - Y \times M$
```

3.1.4 Montgomery Modular Multiplication

In recent years the *Montgomery* modular multiplication algorithm [Montgomery85] has been the most popular [Orup95, Walter99, Batina02] of the 4 classes of modular multiplication algorithms. It computes $C = A \times B \times R^{-1} \mod M$ rather than a fully reduced residue $C = A \times B \mod M$. Here A, B, C and M are all n-bit integers and $R = 2^n$. An extra modular multiplication can be used to convert the result to a fully reduced residue:

$$C = A \times B \mod M = (A \times B \times R^{-1}) \times (R^2) \times R^{-1} \mod M.$$

Another way is to use the *Montgomery residues* $AR \mod M$ and $BR \mod M$ instead of A and B as the multiplicands fed into the algorithm. Computation can then proceed with Montgomery residues as an internal representation. The result of a Montgomery modular multiplication is itself a Montgomery residue $ABR \mod M = (AR) \times (BR) \times R^{-1} \mod M$, which can be fed directly into a subsequent Montgomery modular multiplication. Inputting $ABR \mod M$ and 1 into a Montgomery modular multiplier produces a fully reduced residue as $AB \mod M = (ABR) \times 1 \times R^{-1} \mod M$.

This algorithm has evolved a great deal since its introduction in 1985 [Montgomery85]. Improvements include the use of a higher radix and interleaved structures. The algorithm can be rewritten to permit trivial QDS or to move the QDS from the critical path [Walter99, Batina02].

A binary Montgomery modular multiplication calculates

$$\begin{aligned}
& A \times B \times R^{-1} \mod M \\
= \; & A \times B \times 2^{-n} \mod M \\
= \; & (a_0B + a_1B2 + a_2B2^2 + \cdots + a_{n-1}B2^{n-1}) \times 2^{-n} \mod M \\
= \; & a_0B2^{-n} + a_1B2^{-(n-1)} + a_2B2^{-(n-2)} + \cdots + a_{n-1}B2^{-1} \mod M \\
= \; & (\ldots((a_0B2^{-1} + a_1B)2^{-1} + a_2B)2^{-1} + \cdots + a_{n-1}B)2^{-1} \mod M
\end{aligned}$$

This equation describes an interleaved modular multiplier with the partial result C_i in the ith iteration satisfying $C_{i+1} = (C_i + a_iB)2^{-1} \mod M$. This can be computed in another way [Montgomery85]. If M is odd, then

$$\begin{aligned}
q_i &= C_i \times (-M(0)^{-1}) \mod 2 \\
\text{and } C_i2^{-1} \mod M &= (C_i + q_iM)/2.
\end{aligned}$$

Here $-M(0)^{-1}$ is the least significant bit of $\langle -M^{-1} \rangle_2$. Thus,

$$C_{i+1} = (C_i + a_iB + q_iM)/2 \mod M$$

holds. This means by choosing appropriate quotient digit q_i, the least significant bit of C can be made 0 by adding q_iM. The the division by 2 is simply done by right shifting C by one bit. This simple algorithm is shown in Algorithm 3.7.

A comparison of Algorithm 3.3 and Algorithm 3.5 with Algorithm 3.7 gives the main difference from the Classical and the Sum of Residues algorithms to the Montgomery algorithm. The former two reduces partial products from the most significant digits to the least while the latter works the reverse way. Further development and implementation of the Montgomery modular multiplication algorithm is made in Section 3.4.

Algorithm 3.7 Interleaved Montgomery Modular Multiplication at Radix 2

```
Require: $R = 2^n$
Ensure: $C_n \equiv A \times B \times R^{-1} \mod M$
  $C_0 = 0$
  for $i = 0$ to $n - 1$ do
    $C = C_i + a_i B$ {Partial product accumulation}
    $q_i = C_i \times (-M(0)^{-1}) \mod 2$ {Quotient digit selection}
    $C_{i+1} = (C + q_i M)/2$ {Reduction step}
  end for
```

3.2 Reinvigorating Sum of Residues

3.2.1 Tomlinson's Algorithm

In recent years, Sum of Residues algorithms have either been overlooked [Walter99] or incorporated within one of the other class of reduction algorithms [Chen99]. In this section, we revisit this distinct class of algorithms and make some development on it.

A typical SOR algorithm was proposed by Tomlinson in [Tomlinson89] and is shown in Algorithm 3.8. Tomlinson's algorithm performs the reduction by setting the most significant bits to zero and accounting for this change by adding the pre-computed residue ($q \times 2^{n+1} \mod M$). We take a slightly improved architecture, illustrated in Figure 3.1, of this algorithm as a starting point for further development. A Carry Save Adder (CSA) [Burks46] is used to perform the three-term addition $C_i = 2C_{i+1} + a_i B + (q \times 2^{n+1} \mod M)$. To make sure $2C_{i+1}$ is n bits long, the same as the other two addends, q is set to be the upper 3 bits of the current partial result instead of 2.

[Chen99] gives a similar algorithm but sets q only two bits long. This means that the partial result C_i may be $n + 1$ bits long. To bound it within

Algorithm 3.8 Tomlinson's Sum of Residues Modular Multiplication

```
Ensure: $C_0 \equiv A \times B \mod M$, $C < 2^{n+1}$
  $C_n = 0$
  $q = 0$
  for $i = n - 1$ downto 0 do
    $C_i = 2C_{i+1} + a_i B$
    $C_i = C_i + (q \times 2^{n+1} \mod M)$ {The residue $(q \times 2^{n+1} \mod M)$ is pre-computed.}
    $q = \lfloor \frac{C_i}{2^n} \rfloor$ {$q$ is the upper 2 bits of $C$}
    $C_i = C_i - q \times 2^n$ {Set the upper 2 bits of $C$ to zero}
  end for
  $C_0 = (C_0 + (q \times 2^{n+1} \mod M)) \mod M$
```

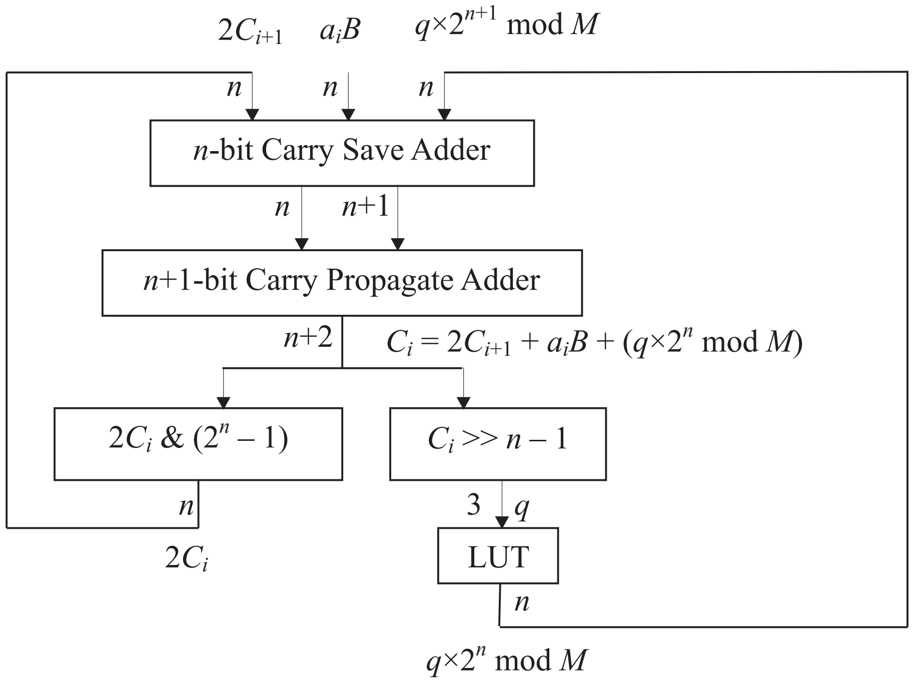


Figure 3.1: An Architecture for Tomlinson's Modular Multiplication Algorithm [Tomlinson89]

n bits, a subtracter is used to constantly subtract M until C_i has only n bits. This redundant step greatly increases the latency of the algorithm.

3.2.2 Eliminating the Carry Propagate Adder

There are two obvious demerits of the architecture in Figure 3.1. Firstly, a Carry Propagate Adder (CPA) is used to transform the redundant representation of C_i to its non-redundant form. This is required because the upper 3 bits of C_i have to be known to look up $q \times 2^n \mod M$ before the next iteration. The CPA delay contributes significantly to the latency of the implementation. The second problem is that the look-up of $q \times 2^n \mod M$ is on the critical path.

Both of these problems can be solved by keeping the intermediate result in a redundant carry save form. The CPA of Figure 3.1 is eliminated so that the calculation of the partial result becomes $C1_i + C2_i = C1_{i+1} + C2_{i+1} + a_i B + ((q1+q2) \times 2^n \mod M)$, where $C1_i$ and $C2_i$ are the redundant representation of C_i as sum and carry terms respectively. A modified architecture is shown in Figure 3.2. The CPA is replaced by a second CSA.

The pre-computed residue $(q1+q2) \times 2^n \mod M$, which must be retrieved from a look-up table (LUT), can be sent to the second CSA rather than the first. All three addends to the first CSA are available at the beginning of each iteration and the table look-up step can be performed in parallel with the first CSA.

In Figure 3.1 it can be seen that the carry output of the first CSA is $n + 1$ bits wide. This can not be input directly to the second CSA which is only n-bits wide. Thus, in Figure 3.2 the MSB of the $(n + 1)$-bit carry is sent to the LUT circuit instead. The LUT retrieves two possible values of

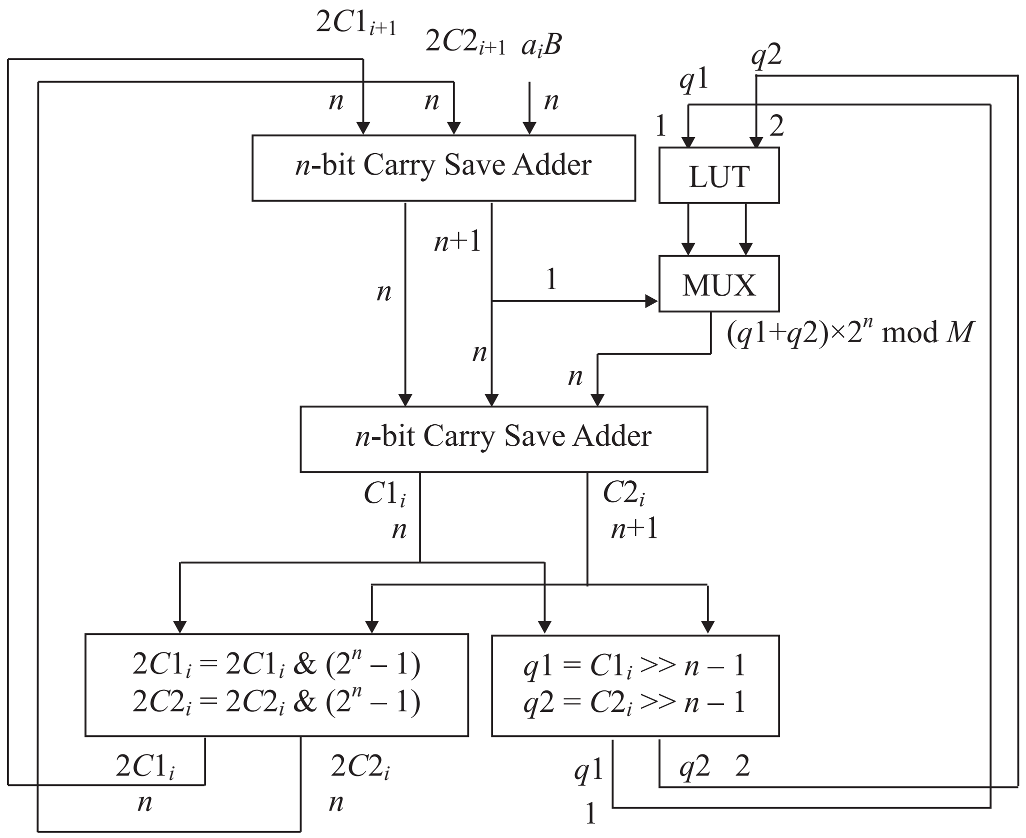


Figure 3.2: Modified Sum of Residues Modular Multiplier Architecture

$(q1 + q2) \times 2^n \bmod M$ corresponding to the case of either a 0 or 1 in the MSB of the carry output from the first CSA. A MUX selects the appropriate value of $(q1 + q2) \times 2^n \bmod M$ once the MSB is available. Thus, although the LUT executes in parallel with the first CSA, an additional MUX appears on the critical path.

3.2.3 Further Enhancements

If the second CSA in Figure 3.2 can be modified to accept an $(n+1)$-bit input, the MUX can be eliminated. The left of Figure 3.3 shows a conventional n-bit CSA. Note that the output sum is only n bits wide. To accept an $(n + 1)$-bit input, we can just copy the MSB of the $(n + 1)$-bit input to the MSB of output sum. This is illustrated in the right of Figure 3.3. This modified CSA accepts 1 $(n+1)$-bit input and 2 n-bit inputs and produces 2 $(n+1)$-bit outputs.

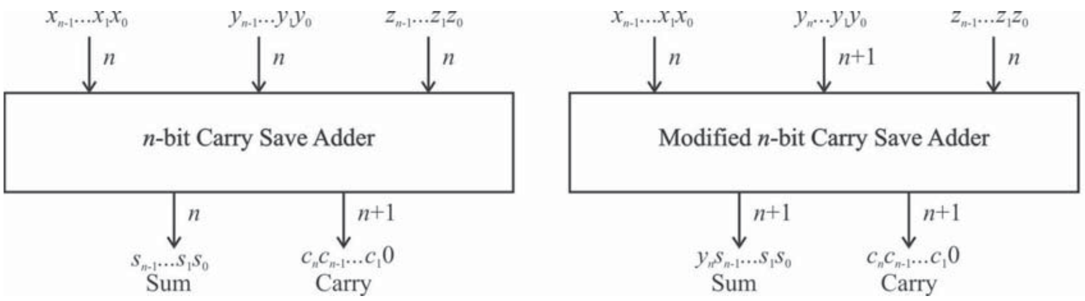


Figure 3.3: n-bit Carry Save Adders

Figure 3.4 shows the resulting modular multiplication architecture. The algorithm corresponding to this new architecture is given as Algorithm 3.9. The CPA has been eliminated from the iteration and the residue lookup has been shifted from the critical path. Also, no subtraction is needed at the end of the algorithm to bound the output within $n + 1$ bits. If $C1_0$ and $C2_0$ are simply summed using a CPA, the resulting output C_0 could be $n + 2$ bits, which needs some further subtraction to be reduced. Therefore the same technique as in the loop is applied. Both $C1_0$ and $C2_0$ are set to $n - 1$ bits and the n-bit residue corresponding to the 2 upper reset bits is retrieved from another LUT. The final sum yields an $(n + 1)$-bit output C_0.

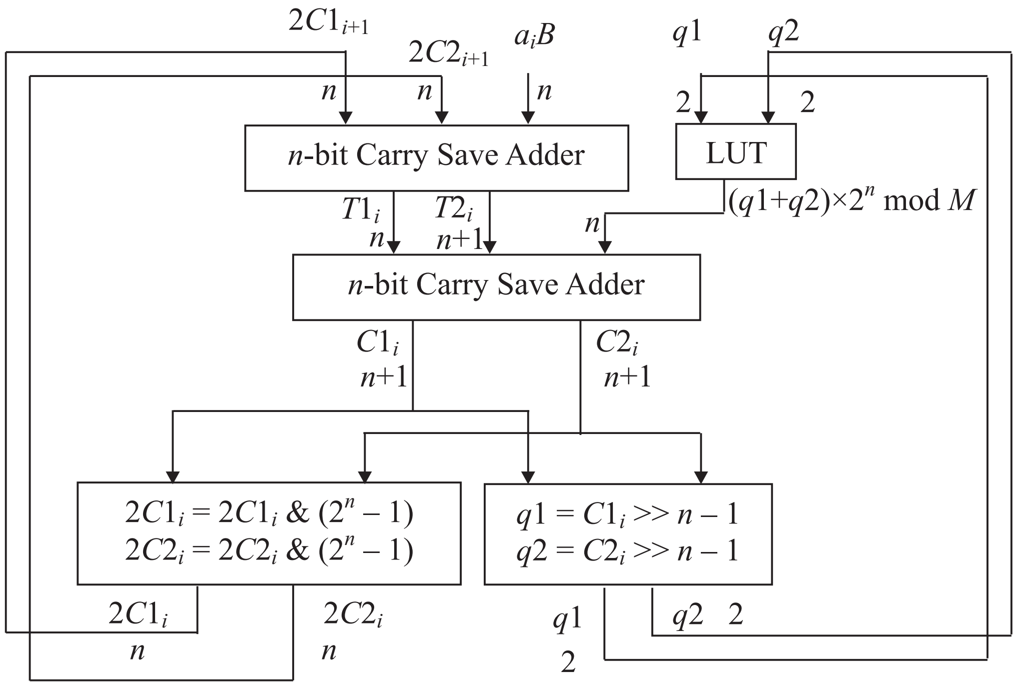


Figure 3.4: New Sum of Residues Modular Multiplier Architecture

The LUTs have a 4-bit input and a n-bit output so that a $(2^4 \times n)$-bit ROM can be used. For example, a 64-bit modular multiplier only needs a 1K-bit ROM, which is reasonable for a RNS channel modular multiplier .

Figure 3.5 shows an example of the new algorithm for the case $r = 2$, $n = 4$, $A = 15 = (1111)_2$, $B = 11 = (1011)_2$ and $M = 9 = (1001)_2$. It is also noted that at the last step a second LUT of the same size is needed.

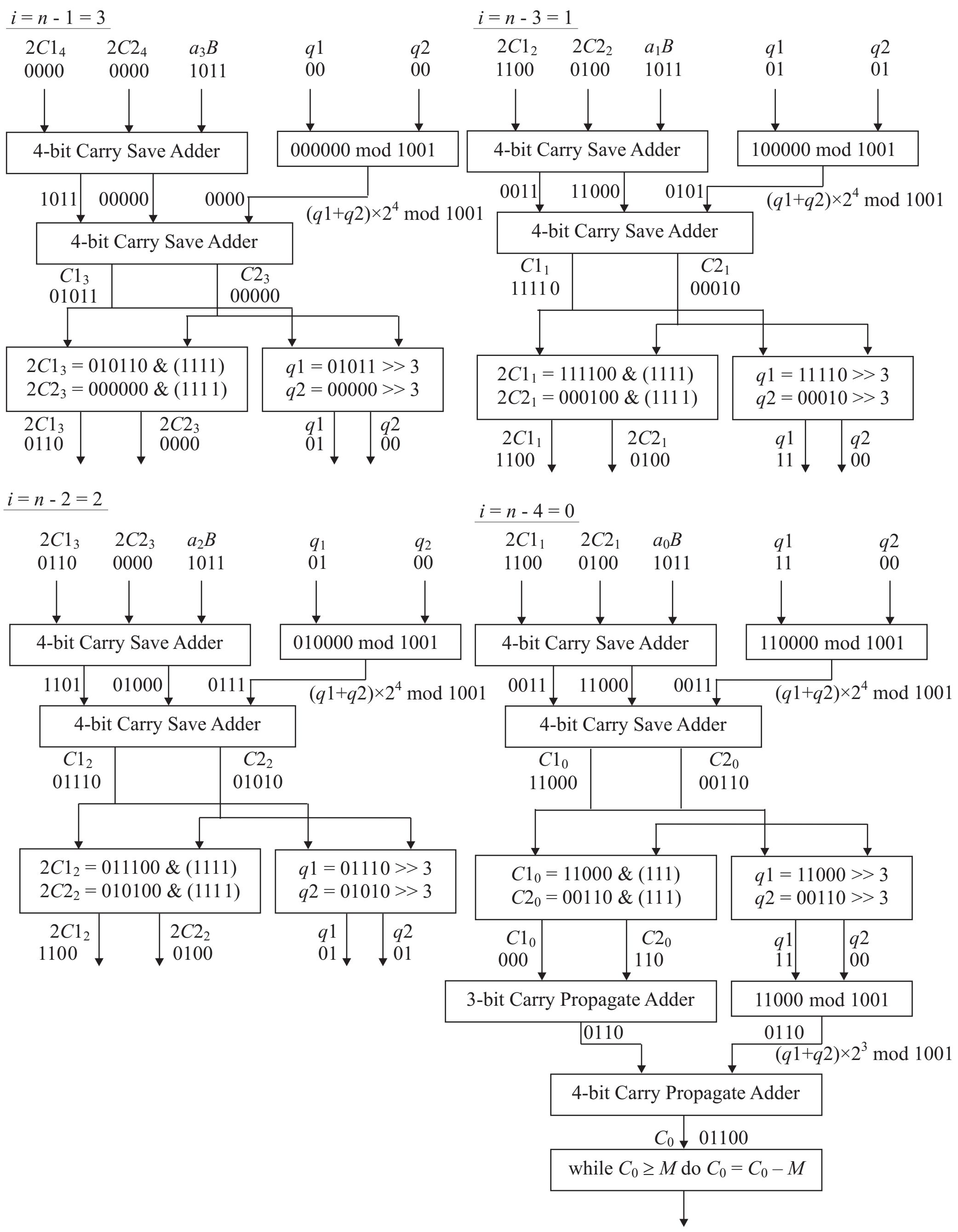


Figure 3.5: An Example for New Sum of Residues Modular Multiplication $n = 4, A = (1111)_2, B = (1011)_2$ and $M = (1001)_2$

Algorithm 3.9 New Sum of Residues Modular Multiplication

```
Ensure: $C_0 \equiv A \times B \mod M$, $C < 2^{n+1}$
  $C1_n = C2_n = q1 = q2 = 0$
  for $i = n-1$ downto 0 do
    $\{T1_i, T2_i\} = 2C1_{i+1} + 2C2_{i+1} + a_i B$ {Carry save addition}
    $\{C1_i, C2_i\} = T1_i + T2_i + ((q1+q2) \times 2^n \mod M)$ {Carry save addition}
    {The residue $((q1 + q2) \times 2^n \mod M)$ is precomputed}
    $\{q1, q2\} = \{C1_i >> (n-1), C2_i >> (n-1)\}$ {$q1$ and $q2$ are the upper 2 bits of $C1_i$ and $C2_i$ respectively.}
    $\{C1_i, C2_i\} = \{2C1_i \ \& \ (2^n - 1), 2C2_i \ \& \ (2^n - 1)\}$ {Set the upper 2 bits of $C1_i$ and $C2_i$ to zero}
  end for
  $\{C1_0, C2_0\} = \{2C1_0, 2C2_0\}$ {Right shift so they are both $n-1$ bits}
  $C_0 = C1_0 + C2_0 + ((q1 + q2) \times 2^{n-1} \mod M)$
```

3.2.4 High Radix

A radix-r version of the algorithm can be produced as in Figure 3.6. If $r = 2^k$ this version executes in n/k iterations; however a larger LUT and $(n+k)$-bit CSAs are required.

3.2.5 Summary of the Sum of Residues Modular Multiplication

In this section, we set out to invigorate the Sum of Residues modular multiplication algorithms. We developed a new structure (shown in Figure 3.4) of this distinct class of modular multiplication algorithm based on Tomlinson's Algorithm [Tomlinson89]. This new algorithm will be implemented in Section 3.5 for a comparison with other classes of algorithms.

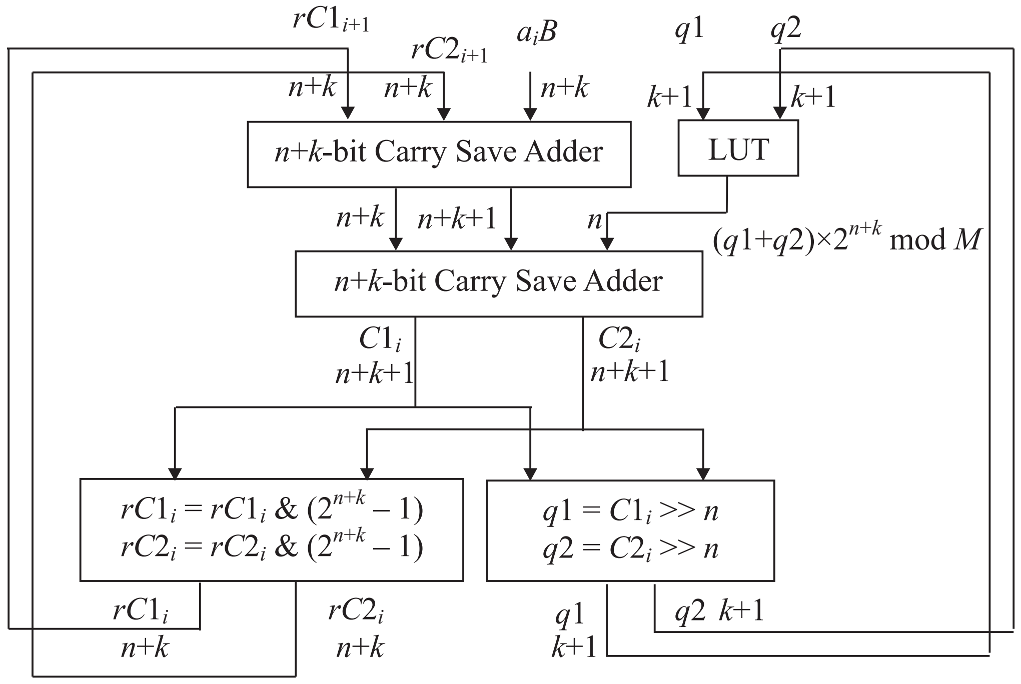


Figure 3.6: New High-Radix Sum of Residues Modular Multiplier Architecture

3.3 Bounds of Barrett Modular Multiplication

In Section 3.1.3, the Improved Barrett modular multiplication algorithm is introduced. In this section, the bounds of the output and input of the algorithm are derived, and hence the range of the two parameters u and v is given. Implementations are also made on Xilinx Virtex2 FPGA platform and the delay and area of this algorithm are simulated.

3.3.1 Bound Deduction

Bounds on the Estimated Quotient

The estimated quotient $\hat{Y}$ is at most 1 less than the actual quotient Y if u and v are chosen appropriately, as shown below.

Recall $Y = \left\lfloor \frac{C_0}{M} \right\rfloor = \left\lfloor \frac{\frac{C_0}{2^{n+v}} \frac{2^{n+u}}{M}}{2^{u-v}} \right\rfloor$ and $\hat{Y} = \left\lfloor \frac{\left\lfloor \frac{C_0}{2^{n+v}} \right\rfloor \left\lfloor \frac{2^{n+u}}{M} \right\rfloor}{2^{u-v}} \right\rfloor$, then

$$\begin{aligned}
Y \geq \hat{Y} &> \frac{\left\lfloor \frac{C_0}{2^{n+v}} \right\rfloor \left\lfloor \frac{2^{n+u}}{M} \right\rfloor}{2^{u-v}} - 1 \\
&> \frac{(\frac{C_0}{2^{n+v}} - 1)(\frac{2^{n+u}}{M} - 1)}{2^{u-v}} - 1 \\
&= \frac{C_0}{M} - \frac{C_0}{2^{n+u}} - \frac{2^{n+v}}{M} + \frac{1}{2^{u-v}} - 1 \\
&\geq \left\lfloor \frac{C_0}{M} \right\rfloor - \frac{C_0}{2^{n+u}} - \frac{2^{n+v}}{M} + \frac{1}{2^{u-v}} - 1 \\
\Leftrightarrow Y \geq \hat{Y} &> Y - \frac{C_0}{2^{n+u}} - \frac{2^{n+v}}{M} + \frac{1}{2^{u-v}} - 1
\end{aligned} \tag{3.7}$$

because $x \geq \lfloor x \rfloor > x - 1$ always holds for any natural x.

Because M is the n-bit modulus, A and B are two n-bit multiplicands and $C_0 = A \times B$, C_0 is $2n$ bits long. Thus,

$$2^{n-1} \leq M \leq 2^n - 1 < 2^n \text{ and } 2^{2n-1} \leq C_0 \leq 2^{2n} - 1 < 2^{2n}.$$

Then (3.7) becomes

$$\begin{aligned} & Y \geq \hat{Y} > Y - \frac{2^{2n}}{2^{n+u}} - \frac{2^{n+v}}{2^{n-1}} + \frac{1}{2^{u-v}} - 1 \\ \Leftrightarrow \quad & Y \geq \hat{Y} > Y - (2^{n-u} + 2^{v+1} + 1 - 2^{v-u}) \end{aligned} \tag{3.8}$$

If we choose $u \geq n+1$ and $v \leq -2$, then $0 < 2^{n-u} \leq \frac{1}{2}$, $0 < 2^{v+1} \leq \frac{1}{2}$ and $0 < 2^{n-u} - 2^{v-u} \leq \frac{1}{2}$. Thus, $1 < 2^{n-u} + 2^{v+1} + 1 - 2^{v-u} < 2$. Therefore, (3.8) becomes

$$Y \geq \hat{Y} > Y - 1.xx.$$

Because $\hat{Y}$ is an integer, $\hat{Y} = Y$ or $\hat{Y} = Y - 1$. Namely, the maximal error on the estimated quotient is limited to 1 by choosing $u \geq n+1$ and $v \leq -2$.

Bounds on the Output

The worst-case word length of the estimated output $\hat{C}$ will be checked below. Recall Equation (3.6) $C = A \times B \mod M = C_0 - Y \times M$ and the remainder C is certainly no more than n bits long. Now Y is replaced by $\hat{Y}$ and (3.6) becomes

$$\hat{C} = C_0 - \hat{Y} \times M \tag{3.9}$$

If $\hat{Y} = Y$, (3.9) will be the same as (3.6) and the result $\hat{C}$ is at most n bits long. If $\hat{Y} = Y - 1$, (3.9) will be $\hat{C} = A \times B - (Y-1) \times M = A \times B - Y \times M + M = C + M$. Because both C and M are n bits long at most, the output is $n+1$ bits long at most. Consequently, the output of the Improved Barrett Modular Multiplication Algorithm is $n+1$ bits.

Bounds on the Input

Because the output is likely to be the input of another modular multiplier, which will itself use Improved Barrett Modular Multiplication, we should ensure output $\hat{C}$ is $n+1$ bits when there are two $n+1$ bit inputs, A and B. We will now show that this consistency exists if u and v are appropriately selected.

Recall Equation (3.7)

$$Y \geq \hat{Y} > Y - \frac{C_0}{2^{n+u}} - \frac{2^{n+v}}{M} + \frac{1}{2^{u-v}} - 1.$$

Now M is n bits and A and B become $n+1$ bits, and so $C_0 = A \times B$ is $2n+2$ bits long. Thus,

$$2^{n-1} \leq M \leq 2^n - 1 < 2^n \text{ and } 2^{2n+1} \leq C_0 \leq 2^{2n+2} - 1 < 2^{2n+2}.$$

Then Equation (3.7) becomes

$$\begin{aligned} & Y \geq \hat{Y} > Y - \frac{2^{2n+2}}{2^{n+u}} - \frac{2^{n+v}}{2^{n-1}} + \frac{1}{2^{u-v}} - 1 \\ \Leftrightarrow \quad & Y \geq \hat{Y} > Y - (2^{n-u+2} + 2^{v+1} + 1 - 2^{v-u}) \end{aligned} \tag{3.10}$$

If we choose $u - 2 \geq n + 1$ i.e. $u \geq n + 3$ and also $v \leq -2$, then (3.10) becomes the same as (3.8):

$$Y \geq \hat{Y} > Y - 1.xx.$$

Therefore, $\hat{Y} = Y$ or $\hat{Y} = Y - 1$ and the output $\hat{C}$ is still $n+1$ bits long in the case of $n+1$-bit inputs by choosing $u \geq n+3$ and $v \leq -2$.

Now, $\hat{Y} \leq Y = \left\lfloor \frac{C_0}{M} \right\rfloor \leq \left\lfloor \frac{2^{2n+2}}{2^{n-1}} \right\rfloor = 2^{n+3}$. While M cannot be 2^{n+1} because it is usually odd, $\hat{Y} < 2^{n+3}$. Therefore, the bound on the estimated quotient $\hat{Y}$ is $n+3$ bits. In conclusion, the bounds on the quotient inputs and output are $n+3$, $n+1$ and $n+1$ respectively. To save hardware, the parameters are suggested to be $u = n+3$ and $v = -2$.

3.3.2 Performance at Different Word Lengths

As stated in Section 2.1.4, the RNS channel width n should be at least 12 bits in a RNS system with 2048-bit dynamic range. The delay of a modular multiplier is expected to increase as the word length n increases. This is apparent in the results from the algorithm implemented and evaluated in terms of speed and area on an FPGA platform using Xilinx tools.

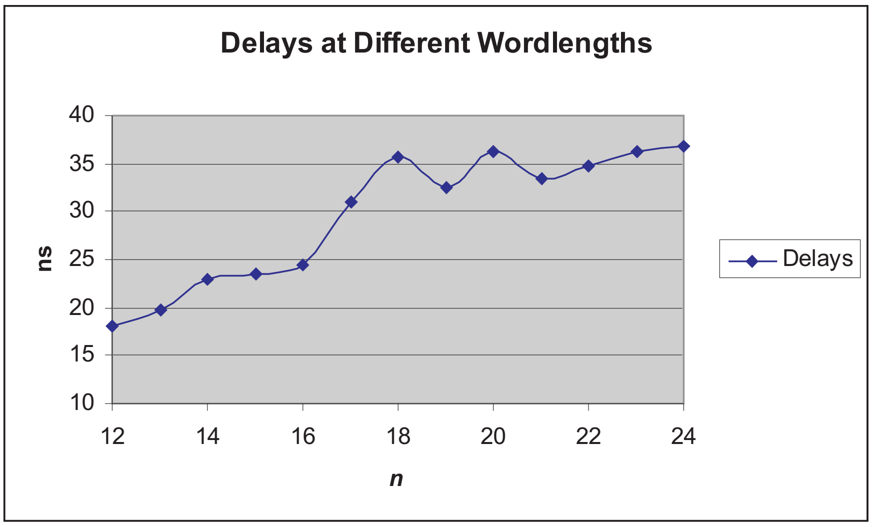
A Xilinx Virtex2 FPGA was used as the implementation target. All of the implementations were performed using the Xilinx ISE environment using XST for synthesis and ISE standard tools for place and route. Speed optimization with standard effort was used for all of the implementations:

Target FPGA:	Virtex2 XC2V1000 with a -6 speed grade, 1M gates, 5120 slices and 40 embedded 18×18 multipliers
Xilinx 6.1i:	XST - Synthesis
	ISE - Place and Route
Optimization Goal:	Speed
Language:	VHDL

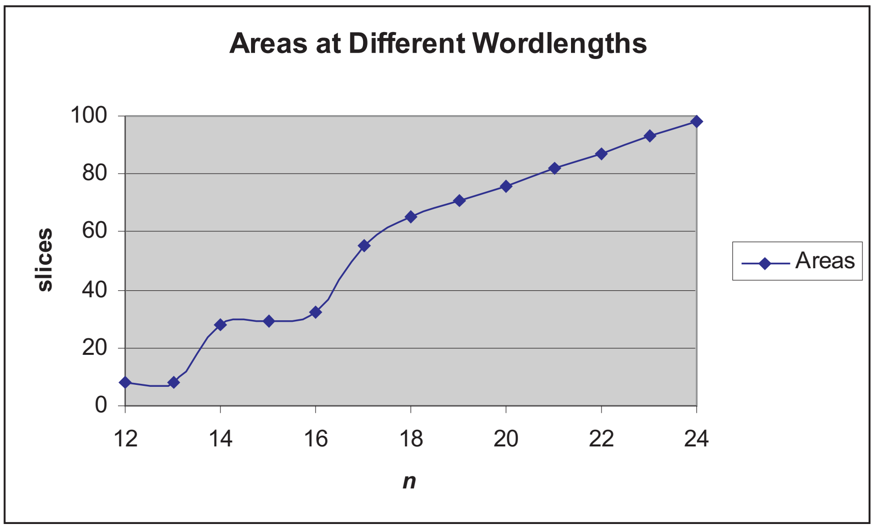
Pure delays of the combinatorial circuit were measured excluding those between pads and pins. They were generated from the Post-Place and Route Static Timing Analyzer with a standard place and route effort level.

The results of the Improved Barrett Modular Multiplier from 12 bit to 24 bits are shown in Figure 3.7(a). As can be seen from Figure 3.7, the performance in terms of both speed and space complexity is best at $n = 12$.

This multiplier is used for the RNS channel modular multiplication in the final implementation of the RNS in Chapter 5.



(a) Delays



(b) Areas

Figure 3.7: Delays and Areas of Improved Barrett Modular Multiplication Algorithm

3.4 Montgomery Modular Multiplication on FPGA

In this section, published techniques of the Montgomery modular multiplication algorithm are analyzed, including interleaved/separated structure, high radix, trivial quotient digit selection and quotient pipelining. They are also implemented on the same FPGA platform described in Section 3.3.2.

3.4.1 Separated Montgomery Modular Multiplication Algorithm

In contrast to interleaved Montgomery modular multiplication in Algorithm 3.7, a separated Montgomery modular multiplier performs the multiplication and then reduces the product using the Montgomery algorithm. Recall n is the word length of modulus M and suppose C_0 is the $2n$-bit product of $A \times B$. The Montgomery modular reduction algorithm calculates

$$\begin{aligned} & C_0 \times R^{-1} \mod M \\ = \; & C_0 \times 2^{-n} \mod M \\ = \; & (\dots((C_0 2^{-1})2^{-1})2^{-1} \dots)2^{-1} \mod M \\ = \; & (\dots(((C_0 + q_0 M)/2 + q_1 M)/2 + q_2 M)/2 + \cdots + q_{n-1} M)/2 \mod M \end{aligned}$$

The algorithm is shown in Algorithm 3.10

3.4.2 High Radix Montgomery Algorithm

Algorithm 3.11 shows an interleaved modular multiplication at a radix r and Algorithm 3.12 shows a separated version of this algorithm. The radix is

Algorithm 3.10 Separated Montgomery Modular Multiplication at Radix 2

```
Require: $R = 2^n$
Ensure: $C_n \equiv A \times B \times R^{-1} \mod M$
  $C_0 = A \times B$
  for $i = 0$ to $n - 1$ do
    $q_i = C_i \times (-M(0)^{-1}) \mod 2$ {Quotient digit selection}
    $C_{i+1} = (C_i + q_i M)/2$ {Reduction step}
  end for
```

chosen to be a power of 2, i.e. $r = 2^k, k = 0, 1, 2, \ldots$. Note $-M(0)^{-1}$ is the least significant digit of $\langle -M^{-1} \rangle_r$ in base r.

Radix-2

Consider the case $r = 2$ in Algorithm 3.12. The product C_0 yet to be reduced modulo M is $2n$ bits and M is n bits long. There are n iterations and exactly one bit is reduced in each iteration since the least significant bit of the current C_i is made 0 by adding $q_i \times M$. After n iterations, the result will be presented as C_n. Table 3.1 illustrates the process for the case of $n = 8$. To show the reduction process more clearly, assume there is no word length growth on the partially reduced results C_i. However, this is impossible and the bound of the growth will be calculated in Section 3.4.3.

Radix-r

Now consider the case when $k > 1$, $r > 2$, and assume that n bits are equivalent to l digits. Then there is 1 digit, i.e. k bits, to be reduced in each iteration. For example, if $k = 2$, we have radix 4 and 2 bits will be reduced in one iteration; there will be $l = 8/2 = 4$ iterations in the case of

Table 3.1: Montgomery Modular Reduction Process at Radix 2

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C_0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
q_0M									*	*	*	*	*	*	*	*
$C_0 + q_0M$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	0
C_1		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
q_1M									*	*	*	*	*	*	*	*
$C_1 + q_1M$		*	*	*	*	*	*	*	*	*	*	*	*	*	*	0
C_2			*	*	*	*	*	*	*	*	*	*	*	*	*	*
q_2M									*	*	*	*	*	*	*	*
$C_2 + q_2M$			*	*	*	*	*	*	*	*	*	*	*	*	*	0
C_3				*	*	*	*	*	*	*	*	*	*	*	*	*
q_3M									*	*	*	*	*	*	*	*
$C_3 + q_3M$				*	*	*	*	*	*	*	*	*	*	*	*	0
C_4					*	*	*	*	*	*	*	*	*	*	*	*
q_4M									*	*	*	*	*	*	*	*
$C_4 + q_4M$					*	*	*	*	*	*	*	*	*	*	*	0
C_5						*	*	*	*	*	*	*	*	*	*	*
q_5M									*	*	*	*	*	*	*	*
$C_5 + q_5M$						*	*	*	*	*	*	*	*	*	*	0
C_6							*	*	*	*	*	*	*	*	*	*
q_6M									*	*	*	*	*	*	*	*
$C_6 + q_6M$							*	*	*	*	*	*	*	*	*	0
C_7								*	*	*	*	*	*	*	*	*
q_7M									*	*	*	*	*	*	*	*
$C_7 + q_7M$								*	*	*	*	*	*	*	*	0
C_8 $=< C_0 \times 2^{-8} >_M$									*	*	*	*	*	*	*	*

Table 3.2: Montgomery Modular Reduction Process at Radix 4

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C_0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
q_0M									*	*	*	*	*	*	*	*
$C_0 + q_0M$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	0	0
C_1			*	*	*	*	*	*	*	*	*	*	*	*	*	*
q_1M									*	*	*	*	*	*	*	*
$C_1 + q_1M$			*	*	*	*	*	*	*	*	*	*	*	*	0	0
C_2					*	*	*	*	*	*	*	*	*	*	*	*
q_2M									*	*	*	*	*	*	*	*
$C_2 + q_2M$					*	*	*	*	*	*	*	*	*	*	0	0
C_3							*	*	*	*	*	*	*	*	*	*
q_3M									*	*	*	*	*	*	*	*
$C_3 + q_3M$							*	*	*	*	*	*	*	*	0	0
C_4									*	*	*	*	*	*	*	*
$=< C_0 \times r^{-4} >_M$																
$=< C_0 \times 2^{-8} >_M$																

$n = 8$. This reduction process is illustrated in Table 3.2 with the assumption that no word length grows during the process. The corresponding interleaved and separated algorithms are shown in Algorithm 3.11 and Algorithm 3.12 respectively.

A complication arises if n is not divisible by k. In these cases, the number of iterations l can be $l = \left\lceil \frac{n}{k} \right\rceil$ or $l = \left\lfloor \frac{n}{k} \right\rfloor$ with some additional correction steps after l iterations. This is just what the function $C_n = \text{Correct}(C_l)$ does in Algorithm 3.11 and Algorithm 3.12. Take $n = 8$ and $k = 3$ as an example for the version of separated modular multiplication in Algorithm 3.12, where one digit equals 3 bits and hence 3 bits are reduced in each iteration.

Let us consider how many iterations are required. If $l = \left\lceil \frac{n}{k} \right\rceil = \left\lceil \frac{8}{3} \right\rceil = 3$,

Algorithm 3.11 Interleaved Montgomery Modular Multiplication at Radix r

```
Require: $R = 2^n$, $r = 2^k$ and $l = \lfloor \frac{n}{k} \rfloor$
Ensure: $C_n \equiv A \times B \times R^{-1} \mod M$
  $C_0 = 0$
  for $i = 0$ to $l - 1$ do
    $C = C_i + a_i B$ {Partial product accumulation}
    $q_i = C_i \times (-M(0)^{-1}) \mod r$ {Quotient digit selection}
    $C_{i+1} = (C + q_i M)/r$ {Reduction step}
  end for
  $C_n = \text{Correct}(C_l)$ {Correction step}
```

Algorithm 3.12 Separated Montgomery Modular Multiplication at Radix r

```
Require: $R = 2^n$, $r = 2^k$ and $l = \lfloor \frac{n}{k} \rfloor$
Ensure: $C_n \equiv A \times B \times R^{-1} \mod M$
  $C_0 = A \times B$
  for $i = 0$ to $l - 1$ do
    $q_i = C_i \times (-M(0)^{-1}) \mod r$ {Quotient digit selection}
    $C_{i+1} = (C_i + q_i M)/r$ {Reduction step}
  end for
  $C_n = \text{Correct}(C_l)$ {Correction step}
```

Table 3.3: Montgomery Modular Reduction Process at Radix 8 ($l = \lceil \frac{n}{k} \rceil$)

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C_0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
q_0M									*	*	*	*	*	*	*	*
$C_0 + q_0M$	*	*	*	*	*	*	*	*	*	*	*	*	*	0	0	0
C_1				*	*	*	*	*	*	*	*	*	*	*	*	*
q_1M									*	*	*	*	*	*	*	*
$C_1 + q_1M$				*	*	*	*	*	*	*	*	*	*	0	0	0
C_2							*	*	*	*	*	*	*	*	*	*
q_2M									*	*	*	*	*	*	*	*
$C_2 + q_2M$							*	*	*	*	*	*	*	0	0	0
C_3										*	*	*	*	*	*	*
$=< C_0 \times r^{-3} >_M$																
$=< C_0 \times 2^{-9} >_M$																
Corrected C_3									*	*	*	*	*	*	*	0
$=< 2 \times C_0 \times 2^{-9} >_M$																
$=< C_0 \times 2^{-8} >_M$																

9 bits will be reduced and the result is $C_0 \times 2^{-9} \mod M$ instead of $C_0 \times 2^{-8} \mod M$. Then $C_0 \times 2^{-9} \mod M$ is multiplied by 2 in the correction step to obtain the correct answer. This is shown in Table 3.3.

Rather than having $l = \lceil \frac{n}{k} \rceil$, let us now consider $l = \lfloor \frac{n}{k} \rfloor = \lfloor \frac{8}{3} \rfloor = 2$. In this case, 6 bits will be reduced and the result is $C_0 \times 2^{-6} \mod M$. Then $C_0 \times 2^{-6} \mod M$ is further reduced to $C_0 \times 2^{-8} \mod M$ either by two additional radix-2 iterations or one radix-4 iteration. This is shown in Table 3.4 with one radix-4 iteration added at the end as a correction step.

Of the two cases considered, $l = \lfloor \frac{n}{k} \rfloor$ is better because the result from $l = \lceil \frac{n}{k} \rceil$ is likely not to be fully reduced. This is proved in Section 3.4.3. Therefore, the version of $l = \lfloor \frac{n}{k} \rfloor$ iterations is adopted in Algorithm 3.11 and

Table 3.4: Montgomery Modular Reduction Process at Radix 8 ($l = \lfloor \frac{n}{k} \rfloor$)

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C_0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
q_0M									*	*	*	*	*	*	*	*
$C_0 + q_0M$	*	*	*	*	*	*	*	*	*	*	*	*	*	0	0	0
C_1				*	*	*	*	*	*	*	*	*	*	*	*	*
q_1M									*	*	*	*	*	*	*	*
$C_1 + q_1M$				*	*	*	*	*	*	*	*	*	*	0	0	0
C_2							*	*	*	*	*	*	*	*	*	*
q_2M									*	*	*	*	*	*	*	*
$C_2 + q_2M$							*	*	*	*	*	*	*	*	0	0
Corrected C_2 $< C_0 \times 2^{-8} >_M$									*	*	*	*	*	*	*	*

Algorithm 3.12 with one $n \mod k$ correction step added at the end when n is not divisible by k.

The performance of the multipliers using different radices are compared in Figure 3.8 using the Xilinx FPGA simulation described on page 45. Following the word length in the tables above, $n = 16$ bits is chosen here, which is greater than 12 bits, the lower bound of n derived in Section 2.1.4. As can be seen from Figure 3.8, the higher the radix $r = 2^k$, the faster the modular multiplier runs, especially when k becomes close to the word length n. However, very high radices can be impossible to implement, as proved in Section 3.4.5 below, and, therefore, the radix is often selected to be as high as possible under some further conditions.

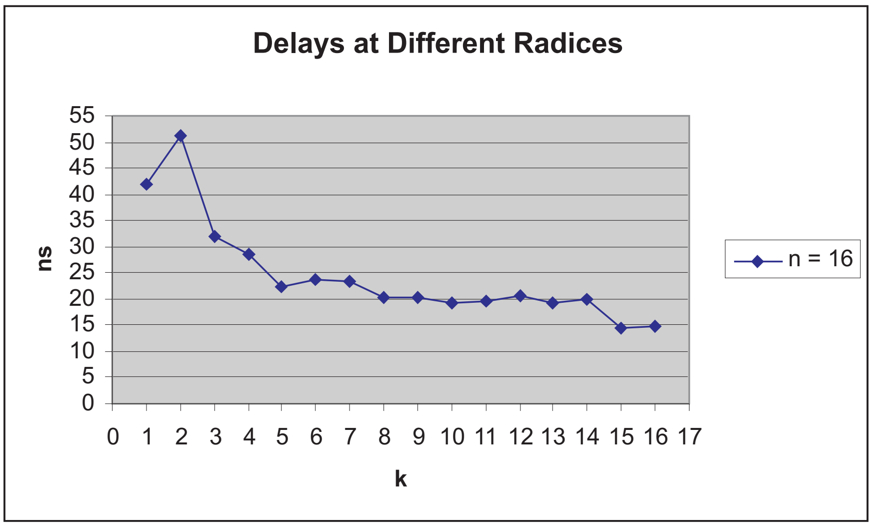


Figure 3.8: Delays of Montgomery Modular Multiplication at Different Radices.

3.4.3 Bound Deduction

In this section we consider the magnitudes of the intermediate results of Montgomery modular multiplication algorithm and reduce their bounds.

Bounds of Interleaved Montgomery Modular Multiplication Algorithm

Recall the Interleaved Montgomery Modular Multiplication Algorithm in Algorithm 3.11. The partial result C_i in the current iteration has the following relationship with C_{i+1} in next iteration:

$$rC_{i+1} = C_i + a_iB + q_iM, \tag{3.11}$$

where B and M are n bits long, a_i and q_i are to be no more than k bits long and $r = 2^k$. We want the word length of C_i and C_{i+1} to be consistent and certainly as short as possible to save hardware.

If C_i is x bits long, the worst case of the right side of (3.11), $C_i+a_iB+q_iM$ is $2^x-1+(2^k-1)(2^n-1)+(2^k-1)(2^n-1) = 2^x-1+2(2^k-1)(2^n-1) = 2^kC_{i+1}$. If C_i and C_{i+1} are the same word length, then $C_{i+1} \leq 2^x - 1$ should hold. Thus,

$$2^x - 1 + 2(2^k - 1)(2^n - 1) \leq 2^k(2^x - 1),$$

which gives

$$2^x \geq 2^{n+1} - 1.$$

Because $n > 0$, $1 < 2^n = 2^{n+1} - 2^n$, hence $2^n < 2^{n+1} - 1$ and $2^x > 2^{n+1} - 1, x \geq n+1$. To minimize hardware choose $x = n+1$. In other words $n+1$ bits is sufficient for the partial result C_i of the interleaved Montgomery modular multiplier. It can also be seen that the final result C_n is at most $n+1$ bits long regardless of n and k.

Bounds of Separated Montgomery Modular Multiplication Algorithm

Recall the Separated Montgomery Modular Multiplication Algorithm in Algorithm 3.12 on page 51. The partial result C_i in the current iteration has the following relationship with C_{i+1} in its next iteration.

$$rC_{i+1} = C_i + q_iM, \tag{3.12}$$

where M is n bits long, q_i is k bits long at most and $r = 2^k$. Again assuming that C_i is x bits long, the right side of (3.12), $C_i + q_iM$ can be $2^x - 1 + (2^k - 1)(2^n - 1) = 2^kC_{i+1}$ in the worst case. If C_i and C_{i+1} are consistent, $C_{i+1} \leq 2^x - 1$ should hold. Thus,

$$2^x - 1 + (2^k - 1)(2^n - 1) \leq 2^k(2^x - 1),$$

which gives

$$x \geq n.$$

This means for all the word lengths greater than or equal to n bits C_i and C_{i+1} are consistent. However, the word length of C_i cannot be shorter than $2n$, because $A \times B$ is assigned to C_0 at the very first step. Therefore, the partial result C_i of the separated modular multiplier is $2n$ bits long.

Now the range of the final result C_n is to be determined. There are two alternations for the number of iterations l, i.e. $l = \left\lceil \frac{n}{k} \right\rceil$ and $l = \left\lfloor \frac{n}{k} \right\rfloor$, as discussed in Section 3.4.2 above.

If $l = \left\lceil \frac{n}{k} \right\rceil$, then let $kl = n + p$, where $0 \leq p \leq k - 1$. All of the l iterations in Algorithm 3.12 up to the line $C_n = \text{Correct}(C_l)$ are connected and expressed in the equation below:

$$r^l C_l = C_0 + q_0 M + r q_1 M + r^2 q_2 M + \cdots + r^{l-1} q_{l-1} M,$$

where M is n bits long, q_i is k bits long at most, C_0 is $2n$ bits long and $r = 2^k$. Thus,

$$\begin{aligned}
& 2^{kl} C_l \leq (2^{2n} - 1) + (r-1)(2^n - 1) + r(r-1)(2^n - 1) + r^2(r-1)(2^n - 1) \\
& + \cdots + r^{l-1}(r-1)(2^n - 1) \\
= \ & (2^{2n} - 1) + (r-1)(2^n - 1)(1 + r + r^2 + \cdots + r^{l-1}) \\
= \ & (2^{2n} - 1) + (2^n - 1)(2^{kl} - 1) \\
\Rightarrow \ & 2^{kl} C_l \leq (2^{2n} - 1) + (2^n - 1)(2^{kl} - 1) \qquad (3.13)
\end{aligned}$$

Because $kl = n + p$, (3.13) becomes

$$\begin{aligned}
& 2^{n+p} C_l \leq (2^{2n} - 1) + (2^n - 1)(2^{n+p} - 1) \\
\Rightarrow \ & 2^p C_l \leq (2^n - 1)(2^p + 1) < 2^p(2^{n+1} - 1) \\
\Rightarrow \ & C_l \leq 2^{n+1} - 1.
\end{aligned}$$

Therefore, $C_l = C_0 2^{-kl} \mod M$ is $n + 1$ bits long at most. However, we need $C_n = C_0 2^{-n} \mod M$, so $C_0 2^{-kl} \mod M$ should be left-shifted by p bits to produce $C_n = C_0 2^{-n} \mod M$. In other words, the process Correct(C_l) adds p "0"s to the end of $C_0 2^{-kl} \mod M$. Consequently, the final result C_n is $n+p+1$ bits long, where $p = k - (n \mod k)$. Hence the result is not fully Montgomery modular reduced. For example, if $n = 16$, and $k = 7$, then $p = k - (n \mod k) = 5$ and $n + p + 1 = 16 + 5 + 1 = 22$ bits. This is why this approach is not used in Algorithm 3.11 and Algorithm 3.12.

If $l = \lfloor \frac{n}{k} \rfloor$, then let $kl + p = n$, where $0 \leq p \leq k - 1$. All of the l iterations in Algorithm 3.12, along with the Correct(C_l) line, are connected and expressed in the equation below:

$$2^n C_n = C_0 + q_0 M + r q_1 M + r^2 q_2 M + \cdots + r^{l-1} q_{l-1} M + r^l q_l' M,$$

where q_l' is p bits long at most, M is n bits long, q_i is k bits long at most, C_0 is $2n$ bits long and $r = 2^k$. Thus,

$$\begin{aligned} & 2^n C_n \leq (2^{2n} - 1) + (r-1)(2^n - 1) + r(r-1)(2^n - 1) + r^2(r-1)(2^n-1) \\ & + \cdots + r^{l-1}(r-1)(2^n-1) + r^l(2^p - 1)(2^n - 1) \\ = \;& (2^{2n} - 1) + (2^{kl} - 1)(2^n - 1) + 2^{kl}(2^p - 1)(2^n - 1) \\ = \;& 2^{n+1}(2^n - 1) \\ \Rightarrow \;& C_n \leq 2^{n+1} - 2 < 2^{n+1} - 1. \end{aligned}$$

Therefore, $C_n = C_0 2^{-n} \mod M$ is at most $n+1$ bits, which means it is fully reduced.

3.4.4 Interleaved vs. Separated Structure

Figure 3.9 shows the comparison between interleaved and separated multipliers in terms of delays and areas respectively. Various radices are included and n is chosen to be 16. In Figure 3.9(a), the separated modular multiplier is obviously faster than the interleaved one at $k < 8$, i.e. when k is less than

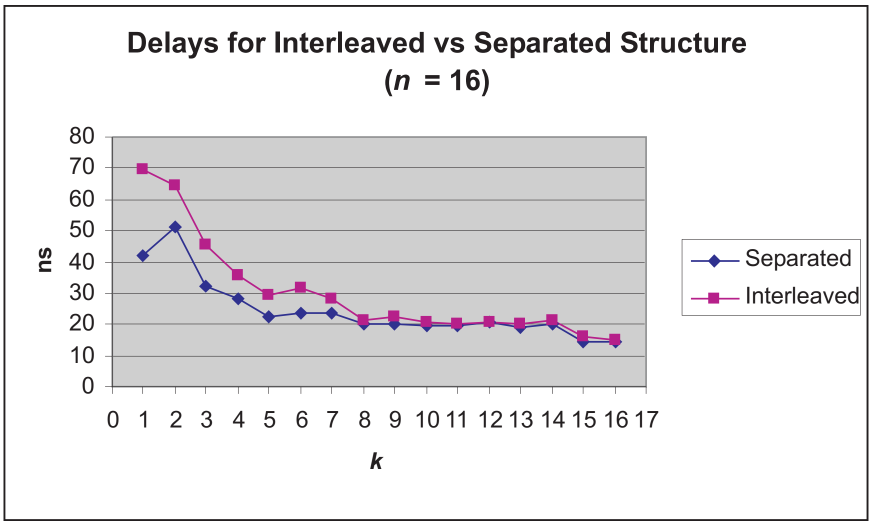
$n/2$, while when k becomes larger than $n/2$, this advantage becomes vague. However, as proved in Section 3.4.5 below, k cannot be too large. (Typically k must not be greater than $n/2$.) Therefore, the separated structure is recommended to implement the Montgomery Modular Multiplier because it is faster than the interleaved structure at most of the possible values of k. Although our main optimization goal is speed, Figure 3.9(b) shows that the separated structure also gains some advantage over the interleaved one in hardware area.

3.4.5 Trivial Quotient Digit Selection

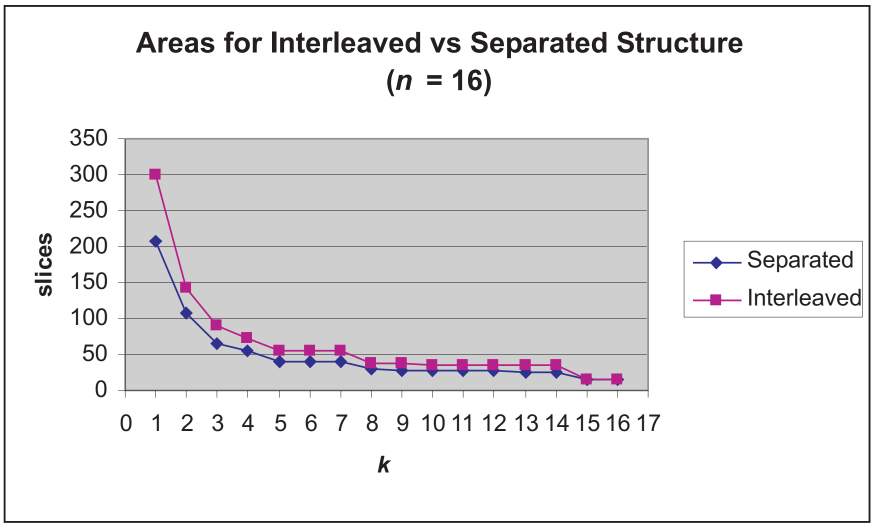
Techniques have been published to improve performance by making the quotient digit selection step trivial and moving it from the critical path [Shand93, Walter95]. Recall the Separated Montgomery Modular Multiplication Algorithm shown in Algorithm 3.12 on page 51. In each iteration, q_i is computed using $q_i = C_i \times (-M(0)^{-1}) \mod r$, where $-M(0)^{-1}$ is the least significant bit of $\langle -M^{-1} \rangle_2$. This step is called quotient digit selection (QDS). If M is always selected to guarantee that $M \mod r = r - 1$, then $-M(0)^{-1} \mod r = M(0)M(0)^{-1} \mod r = 1$ can also be guaranteed, and thus $q_i = C_i(0)$ holds and the QDS step examines only the least significant digit of the partial result C_i. This is known as trivial QDS and is included in Algorithm 3.13.

However, this brings a restriction on the radix $r = 2^k$. Because M should always satisfy $M \mod r = r - 1$, which means that the least significant digit of M must satisfy $M(0) = r - 1 = 2^k - 1$, i.e. there should be k "1"s at the end of M. Thus for $n = 8$, M should be in the form *******1 if $r = 2$, ******11 if $r = 4$ and *****111 if $r = 8$.

Recall the conclusion from Section 2.1.4: to construct a RNS system with



(a) Delays



(b) Areas

Figure 3.9: Delays and Areas of Interleaved and Separated Montgomery Modular Multiplication at Different Radices

Algorithm 3.13 Separated Montgomery Modular Multiplication Algorithm with Trivial QDS

```
Require: $R = 2^n$, $r = 2^k$ and $l = \left\lfloor \frac{n}{k} \right\rfloor$
Ensure: $C_n \equiv A \times B \times R^{-1} \mod M$
  $C_0 = A \times B$
  for $i = 0$ to $l - 1$ do
    $C_{i+1} = (C_i + C_i(0)M)/r$ {Reduction step with trivial QDS}
  end for
  $C_n$ = Correct($C_l$) {Correction step}
```

2048-bit dynamic range using equal word length moduli, the channel width n should be at least 12 bits.

However, if trivial QDS is used, the range of the available integers may not be enough for the co-prime moduli to be found. For example, if $n = 12$ and $r = 4$, i.e. $k = 2$, 171 co-prime moduli must be found from the set, $\{y|y = 4x+3, 2^9 \leq x \leq 2^{10}-1\}$, which has $2^{10}-2^9 = 512$ integers. Only 132 co-prime numbers can be found in these 512 integers*, and this is not enough. Therefore, $n = 12$ can only be feasible in the case of radix-2, i.e. $k = 1$. We name this kind of set a "range set", and obviously there is a unique range set for each pair of n and k. Similarly, if $n = 13$, $N = \left\lceil \frac{2048}{12} \right\rceil = 158$ co-prime numbers are needed. If $k = 2$, 242 co-prime numbers have been found in its range set $\{y|y = 4x + 3, 2^{10} \leq x \leq 2^{11} - 1\}$, which means $k = 2$ is feasible. However if $k = 3$, only 126 co-prime numbers can be found in its range set $\{y|y = 8x + 7, 2^9 \leq x \leq 2^{10} - 1\}$, which means $k = 3$ is not feasible and $k = 2$ is the largest k that suits $n = 13$. Thus, for each value of word length n, there will be a largest feasible k. We name this the *k-boundary*. Table 3.5 lists the k-boundary for n from 12 to 32.

Recall the result from Figure 3.8 in Section 3.4.2. The higher the radix

*This result was obtained using a bruteforce search in Maple.

Table 3.5: The k-boundary of Montgomery Reduction for n from 12 to 32

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

Table 3.6: Shortest Delays of Separated Montgomery Multiplication Algorithm using Trivial QDS at n from 12 to 32

n	12	13	14	15	16	17	18
k	1	1	3	4	5	6	7
delays	32.087	34.282	34.100	27.646	22.297	25.085	24.869
n	19	20	21	22	23	24	25
k	9	10	10	11	13	14	15
delays	22.006	23.684	24.403	25.133	25.583	25.858	25.720
n	26	27	28	29	30	31	32
k	16	17	18	19	20	21	23
delays	25.479	28.483	28.896	27.924	28.217	29.916	31.953

$r = 2^k$, the faster the modular multiplier runs, especially when k becomes roughly the same as the word length n. Nonetheless, here we can see that to use trivial QDS, k cannot be that large, because k cannot be greater than the k-boundary in each word length. Thus, k should always be selected to be one of those values close to the k-boundary or just the k-boundary itself. This was also proved in our implementation, where trivial QDS gave a speed-up of 2 over the non-trivial QDS version. All the possible values of k for n from 12 to 32 were simulated and the shortest delays at each word length n are listed in Table 3.6 and shown in Figure 3.10.

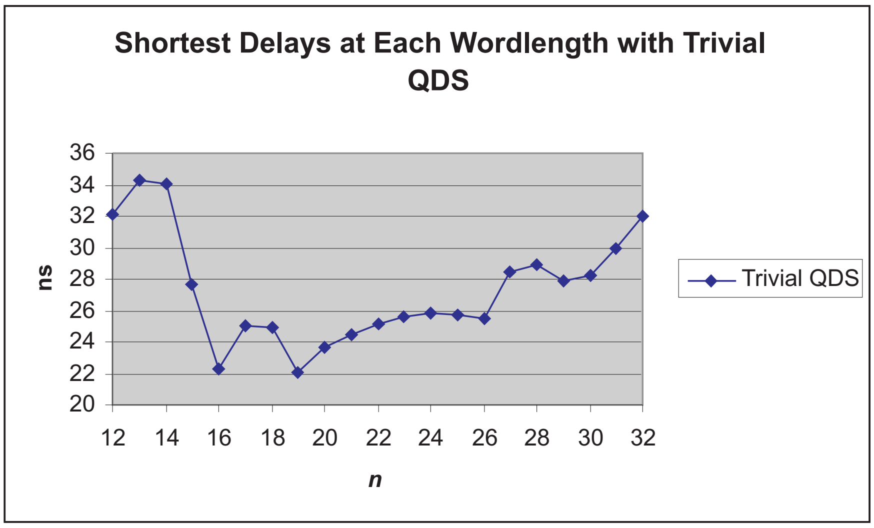


Figure 3.10: Shortest Delays of Separated Montgomery Multiplication Algorithm using Trivial QDS at n from 12 to 32

3.4.6 Quotient Digit Pipelining

For the Separated Montgomery Algorithm, although we have trivial QDS as $q_i = C_i(0)$, as indicated in $C_{i+1} = (C_i + C_i(0)M)/r$ from Algorithm 3.13, the generation of C_{i+1} must wait for the generation of the addend $C_i(0)M$ in ith iteration. A technique called *quotient digit pipelining* was published in [Shand93] and improved in [Walter95, Orup95] to speed the Montgomery Algorithm. The $C_i(0)M$ generated is used several iterations later so that C_i does not have to wait for it to generate C_{i+1}.

Recall the equation of Montgomery Modular Reduction:

$$\begin{aligned}
& C \times R^{-1} \mod M \\
= \; & (\dots(((C + C_0(0)M)/r + C_1(0)M)/r + C_2(0)M)/r \\
& + \cdots + C_{l-1}(0)M)/r \mod M \\
= \; & (\dots(((C - C_0(0) + C_0(0)(M+1))/r - C_1(0) + C_1(0)(M+1))/r - C_2(0) \\
& + C_2(0)(M+1))/r - \cdots - C_{l-1}(0) + C_{l-1}(0)(M+1))/r \mod M \\
= \; & ((\dots((((((C - C_0(0))/r - C_1(0))/r - C_2(0) + C_0(0)(M+1))/r^2)/r \\
& - C_3(0) + C_1(0)(M+1))/r^2)/r - C_4(0) + C_2(0)(M+1)/r^2)/r \\
& - \cdots - C_{l-2}(0) + C_{l-4}(0)(M+1)/r^2)/r - C_{l-2}(0) \\
& + C_{l-4}(0)(M+1)/r^2)/r - C_{l-1}(0) + C_{l-3}(0)(M+1)/r^2)/r \\
& + C_{l-2}(0)(M+1)/r^2 + C_{l-1}(0)(M+1)/r \mod M
\end{aligned} \tag{3.14}$$

As can be seen from (3.14), because $C_i + C_i(0)M$ will set the least significant digit of the result to 0, $C_i + C_i(0)M$ can be split into two parts, $C_i - C_i(0)$ and $C_i(0) + C_i(0)M = C_i(0)(M+1)$, i.e. $C_i + C_i(0)M = [C_i - C_i(0)] + [C_i(0)(M+1)]$. These two parts are computed in parallel in the ith iteration. So $C_i(0)(M+1)/r^2$ is planned to be added to C_{i+2} two iterations later, and $C_i - C_i(0)$ is added by $C_{i-2}(0)(M+1)/r^2$ already computed two iterations ago without waiting for the computation of $C_i(0)(M+1)$. The level of quotient digit pipelining d indicates how many iterations $C_i(0)M$ is added after it was calculated. In the previous example $d = 2$. The cost of quotient digit pipelining is that $C_{l-2}(0)(M+1)/r^2 + C_{l-1}(0)(M+1)/r \mod M$ has to be computed after l iterations. This is reasonable compared with the advantage in speed of the algorithm. The algorithm is shown in Algorithm 3.14.

The selection of d depends on the speed to compute $C_i(0)(M+1)/r^d$. $d = 2$ or 3 was enough in the FPGA implementation studied here. When d was set greater, no remarkable change was observed in speed.

The biggest disadvantage is the limitation quotient digit pipelining im-

Algorithm 3.14 Separated Montgomery Modular Multiplication Algorithm with Trivial QDS and Quotient Digit Pipelining

```
Require: $R = 2^n$, $r = 2^k$ and $l = \left\lfloor \frac{n}{k} \right\rfloor$
Ensure: $C_n \equiv A \times B \times R^{-1} \mod M$
  $C_0 = A \times B$
  for $i = -d$ to $-1$ do
    $C_i(0) = 0$ {Initialisation of quotient digit}
  end for
  for $i = 0$ to $l - 1$ do
    $C_{i+1} = (C_i - C_i(0) + C_{i-d}(0)(M+1)/r^d)/r$ {Reduction step with quotient digit generated previously}
    $C_i(0)(M + 1)/r^d$ {Generation of quotient digit to be used $d$ iterations later}
  end for
  $C_{l-d} = (C_l)$ {Initialisation of $C_{l-d}$ for the final loop}
  for $i = l - d$ to $l - 1$ do
    $C_{i+1} = C_i + C_i(0)(M + 1)/r^{l-i}$ {Reduction step with quotient digit generated previously}
  end for
  $C_n = \text{Correct}(C_l)$ {Correction step}
```

poses on the radix $r = 2^k$. The largest possible k will be smaller than the k-boundary of the version without quotient digit pipelining. The reason is that in the ith iteration,

$$C_{i+1} = (C_i - C_i(0) + C_{i-d}(0)(M+1)/r^d)/r = (C_i - C_i(0))/r + C_{i-d}(0)(M+1)/r^{d+1}$$

should be computed. This means the last $d + 1$ least significant digits of $C_i(0)(M + 1)$ should be 0. Therefore,

$$C_i(0)(M + 1) = 0 \mod r^{d+1} \Leftrightarrow M = -1 \mod r^{d+1}$$

Thus, there should be $k(d + 1)$ "1"s at the end of M instead of k "1"s in the version without quotient digit pipelining above. Similarly, $k(d + 1) \leq k$-boundary instead of $k \leq k$-boundary. If $d = 2$, k should be no more than one third of the k-boundary. Table 3.7 lists the possible values of k and d for n from 12 to 25.

This is obviously a bad effect because the decrease in k slows the multiplier as discussed in Section 3.4.2 and 3.4.5. However, the use of quotient digit pipelining speeds up the algorithm, and therefore, we expect a tradeoff point in the implementation where the speed is highest.

All the possible values of k and d for n from 12 to 32 were simulated and the shortest delays at each word length n are listed in Table 3.8 and shown in Figure 3.11 below.

$n = 24$ has the shortest delay across all word lengths. To judge if the quotient digit pipelining technique really accelerates the modular multiplier, consider Figure 3.12. The version without quotient digit pipelining is faster than the quotient digit pipelining version within the interval [16, 26]. This means the advantage the greater k brings about in the version without quotient digit pipelining overwhelms the advantage the quotient digit pipelining technique brings. Overall, the shortest delay is 22.006(ns) and this occurs for

Table 3.7: k and d in Quotient Digit Pipelining of Montgomery Algorithm for n from 12 to 25

n	12	13	14	15		16		17			18
k-boundary	1	2	3	4		5		6			7
k	1	1	1	1	2	1	2	1	2	3	1
d	NA	1	1,2	1–3	1	1–4	1	1–5	1,2	1	1–6

n	18		19				20				
k-boundary	7		9				10				
k	2	3	1	2	3	4	1	2	3	4	5
d	1,2	1	1–8	1–3	1,2	1	1–9	1–4	1,2	1	1

n	23						24				
k-boundary	13						14				
k	1	2	3	4	5	6	1	2	3	4	5
d	1–12	1–5	1–3	1,2	1	1	1–13	1–6	1–3	1,2	1

n	24		25								
k-boundary	14		15								
k	6	7	1	2	3	4	5	6	7		
d	1	1	1–14	1–6	1–4	1,2	1,2	1	1		

Table 3.8: Shortest Delays of Separated Montgomery Multiplication Algorithm using Quotient Digit Pipelining at n from 13 to 32

n	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
k	1	1	2	2	3	3	3	4	5	6	4	6	6	5	5	9	9	10	10	8
d	1	2	1	1	1	1	2	1	2	1	2	1	1	2	2	1	1	2	1	1
delays	34.318	34.205	29.820	28.735	28.483	27.182	28.912	26.875	27.478	30.054	31.999	26.764	29.158	29.686	31.056	28.836	30.415	28.787	29.033	28.009

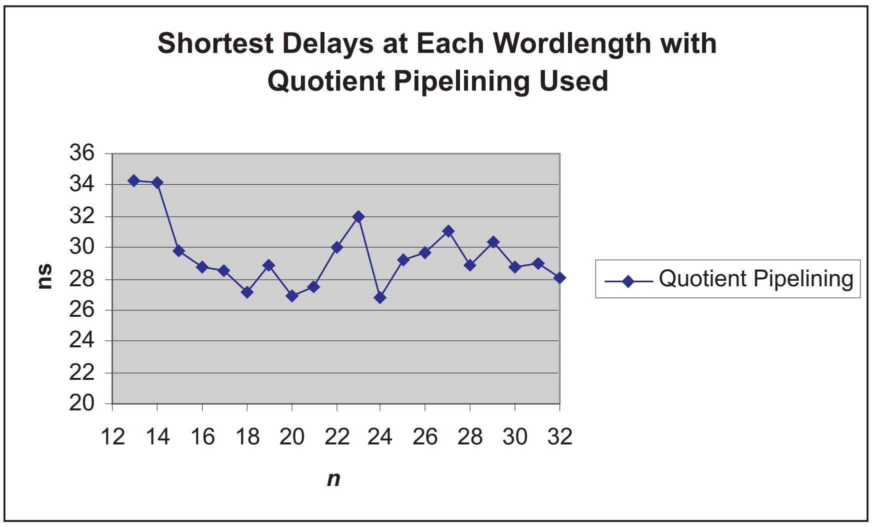


Figure 3.11: Shortest Delays of Separated Montgomery Multiplication Algorithm using Quotient Digit Pipelining at n from 13 to 32

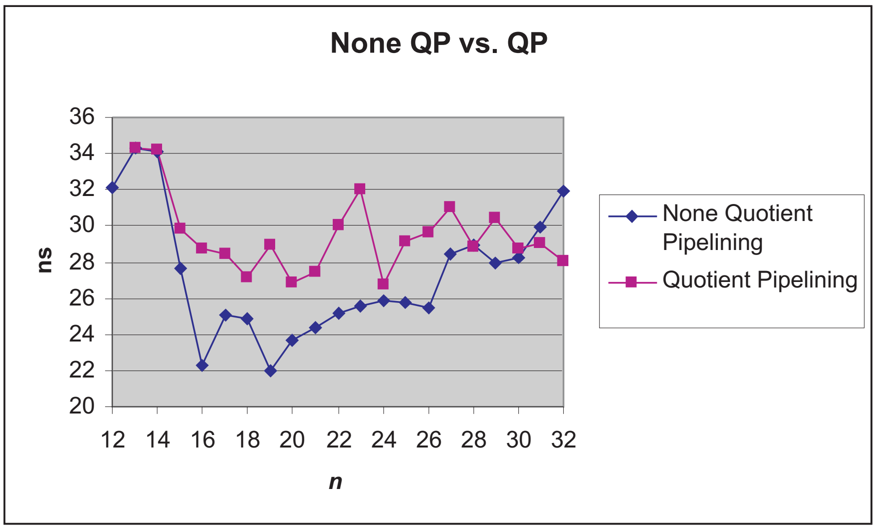


Figure 3.12: Shortest Delays of Separated Montgomery Multiplication Algorithm with & without Quotient Digit Pipelining at n from 13 to 32

a separated Montgomery modular multiplier with $n = 19$ and $k = 9$ without quotient digit pipelining.

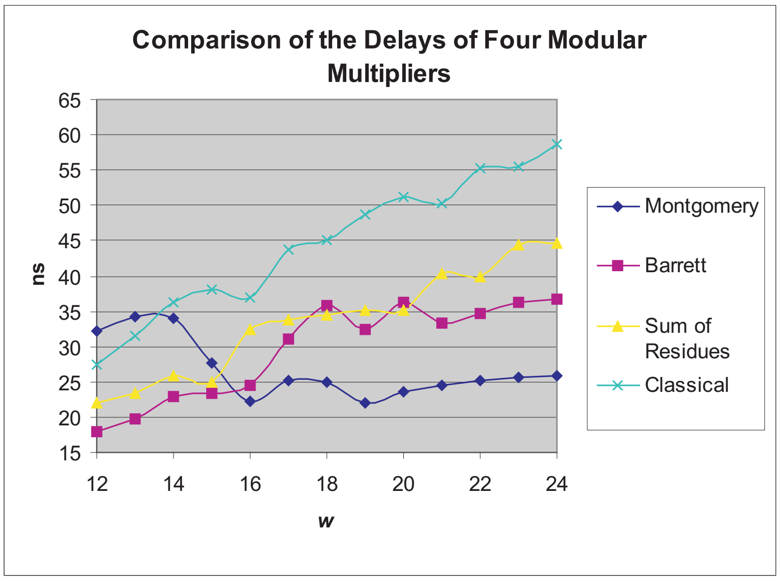
Finally, a conclusion of Montgomery modular multiplication algorithm is obtained: high radix and trivial QDS are both beneficial while the quotient digit pipelining technique is not suitable for FPGA implementations.

3.5 Modular Multiplications within RNS Channels

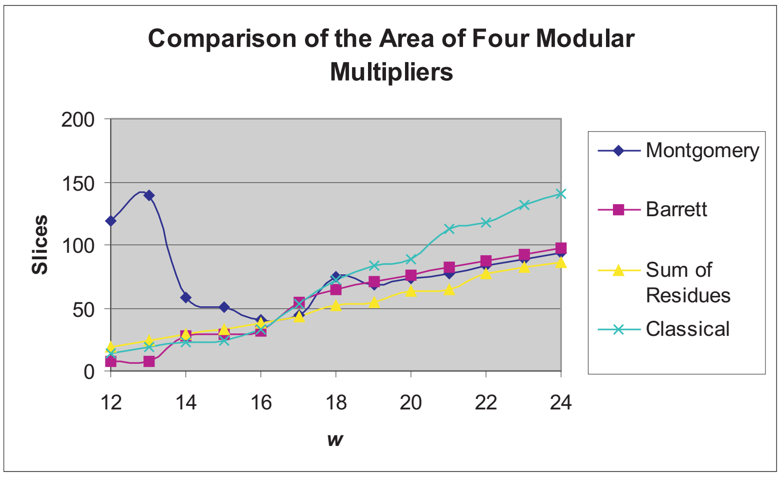
A common characteristic for all modular multiplication algorithms is to produce a result $C > M$ such that a few subtractions of M are required to fully reduce the result. The usual approach is to design the algorithm so that C can be fed back to the input without overflow, even if C is not fully reduced. In terms of the resulting residue C, Montgomery algorithm computes $C = A \times B \times R^{-1} \mod M$ and hence needs conversion to and from the residue $C = A \times B \mod M$, while the other three compute the residue $C = A \times B \mod M$ directly. Nonetheless, the four classes of algorithms discussed in this chapter are all suitable for channel modular multiplication within a RNS.

Figure 3.13 compares these four classes of modular multiplications on the Xilinx Virtex2 FPGA [Kong07]. Barrett and Montgomery take the result from Section 3.3 and 3.4 respectively; the two methods from [Orup91, Walter92] introduced in Section 3.1.1 are combined for the Classical algorithm; and the new algorithm derived in Section 3.2 is used for the simulation of Sum of Residues algorithm in binary.

As can be seen from Figure 3.13, Barrett is fastest for channel width $w \leq 16$. For longer wordlengths, Montgomery is better than others and Classical is slowest. All of the delays show monotonically increase along with the wordlength except Montgomery. This is because the higher the radix, the more strict the choice of modulus in the Montgomery algorithm. This means it limits the selection of RNS modulus set if Montgomery algorithm is selected for the channel modular multiplication implementation. Hence, Barrett was selected instead of Montgomery for the implementation of the RNS where each channel is 18 bits wide in Chapter 5.



(a) Comparison of Delays of Four Modular Multipliers in Binary



(b) Comparison of Areas of Four Modular Multipliers

Figure 3.13: Delays and Areas of Four Classes of Modular Multipliers in Binary

Chapter 4

Four Ways to Do Modular Multiplication in the Residue Number System

The purpose of this chapter is to explore the application of the four classes of modular multiplication algorithms from positional number systems to long word length modular multiplication in a RNS. Section 4.1 surveys modular multiplication algorithms currently available in the RNS. Each of the subsequent 4 sections discusses one of the four classes of algorithms. New Classical, Sum of Residues and Barrett algorithms are developed for modular multiplication in the RNS. Existing Montgomery RNS algorithms are also discussed.

Chapter 2 states some facts: the long word length modular multiplication $Z = A \times B \mod M$, where A, B and the modulus M are all n-bit positive integers, is the most frequent operation in Elliptic Curve Cryptosystems (ECC) [Doche06]; in RSA, it is the only operation required to implement the modular exponentiations which constitute encryption and decryption [Rivest78]; and the Residue Number System (RNS) [Soderstrand86] offers advantages for long word length arithmetic of this kind by representing integers in independent short word length channels. Therefore, implementing public-key cryptosystems using RNS becomes an interesting avenue of research [Kawamura88, Bajard04b]. The drawback to this approach is RNS modular reduction which is a computationally complex operation. Thus, the modular reduction $Z = X \mod M$, also denoted as $\langle X \rangle_M$, is the focus of this chapter.

Early publications [Szabo67] avoided RNS modular reduction altogether by converting from RNS representation to a positional system, performing modular reduction there, and converting the result back into RNS. This approach was discussed in Section 2.1.2. Later, algorithms using look-up tables were proposed to perform short word length modular reduction. Most of these avoided converting numbers from RNS to positional systems, but were limited to 32-bit inputs, as surveyed in Section 4.1. More recently, variations of Montgomery's reduction algorithm [Montgomery85] have been developed for long word length modular reduction, which work entirely within a RNS [Kawamura00, Freking00, Bajard04a].

Montgomery's reduction algorithm is only one of the alternatives available in positional number systems [Kong05]. This raises a question: can any of the other reduction algorithms from positional number systems be applied to the RNS? This chapter provides an answer in the affirmative by presenting RNS reduction algorithms using the other three classes of algorithms: Classical, Sum of Residues and Barrett.

4.1 Short Word Length RNS Modular Multiplications using Look-Up Tables

This section will examine existing modular reduction algorithms in RNS, and then give a new algorithm for doing short word length modular multiplication in RNS. All of these algorithms use look-up tables (LUTs) and only perform short word length modular multiplication in RNS, typically a modulus $M \leq 32$ bits. They cannot be used to do encryptions and decryptions of public-key cryptosystems because sufficiently large LUTs would not only be slow, but infeasibly large in existing technology. Nonetheless, there are other applications for short word length RNS such as Digital Signal Processing (DSP) [Alia84, Barraclough89].

The fast RNS operations, additions and multiplications, have been used for DSP and it has proved to be most profitable to speed up computations in DSP. Some architectures were designed with a very high degree of processing parallelism and communication parallelism tailored to the response time of adders and multipliers. [Parhami96] uses longer intermediate pseudo-residues for look-up table implementations. [Alia98] proposes a high speed pipelined Fast Fourier Transform (FFT) algorithm with relatively optimal VLSI complexity. Several adder and multiplier units were conceived with different characteristics, such as multiply-accumulate units (MAC) [Preethy99] and power-of-2 scaling units [Mahesh00]. Furthermore, general purpose DSP chips were shown to achieve great higher data processing bandwidth when incorporated with RNS processors [Bhardwaj98]. For these applications, a new LUT-based RNS scaling algorithm is described in Section 4.1.3.

4.1.1 RNS Modular Reduction, Scaling and Look-Up Tables

To perform the modular reduction $Z = X \mod M$, one may use

$$X \mod M = X - \left\lfloor \frac{X}{M} \right\rfloor \times M.$$

This modular reduction will be easily done in the RNS if the scaling step

$$Y = \left\lfloor \frac{X}{M} \right\rfloor$$

can be performed effectively within the RNS as subtraction and multiplication can both be accomplished by efficient RNS channel operations. Note that in the scaling step, the *scaling factor* M is a known constant.

RNS Scaling using Look-Up Tables

Most of the RNS scaling schemes from the literature operate using look-up tables. Scaling results are pre-computed and stored in a network of ROM tables as shown in Figure 4.1. The various schemes lead to different structures in the ROM network and, in general, trade reduced latency (achieved through exploiting parallelism within the network) against hardware cost. All LUTs in an implementation are assumed to have the same size. The number of look-up cycles (LUCs) is typically used as a measure of the latency of the scaling algorithm and the number of look-up tables as the hardware cost.

Sizing Look-Up Tables

Both the time and space complexity are heavily dependent on the size of the ROMs selected as well as the width of each RNS modulus. Therefore, to make a uniform base for comparison of them, r is used to denote the

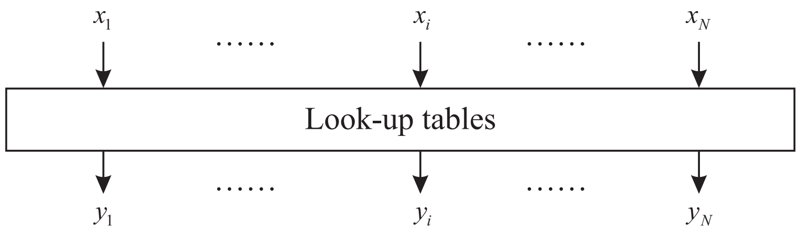


Figure 4.1: RNS Scaling using Look-Up Tables.

number of residue inputs addressing each LUT and it is assumed that r remains the same for all of the LUTs within an implementation. Take the 5-bit moduli $\{19, 23, 29, 31\}$ as an example. If ROMs with an address space of 32K $= 4\text{K} \times 8$ bits are used, then $r = \left\lfloor \frac{\log_2 4\text{K}}{5} \right\rfloor = 2$ because each memory can accommodate two residue inputs at most.

The size of LUTs increases sharply as the modulus word length goes up. Large LUTs are not only slow but are not commercially available. This is exactly why the algorithms based on LUTs are not applicable to long word length modular reduction. In Section 2.1.4, it was shown that the channel width w of a RNS needs to be at least 12 bits for a 1024-bit RSA system. Given $r = 2$, the size of one LUT will reach $2^{r \times w} \times w = 16\text{M} \times 12 \approx 200\text{M}$ bits, and this is not acceptable for any hardware system to be constructed with thousands of such LUTs.

4.1.2 Existing RNS Scaling Schemes using Look-Up Tables

Scaling by Conversion to and from a Positional Representation

A straightforward approach to scaling a RNS number X is to first convert it to a positional number system where it is scaled by M. The result is converted back to RNS representation. The last two steps are fairly convenient and the most time-consuming step is to convert a RNS number into a positional system. As stated in Section 2.1.2, there are two common conversion techniques [Szabo67]: conversion using the Chinese Remainder Theorem (CRT) and using the Mixed Radix System (MRS).

As illustrated in Figure 2.1, $\frac{N(N-1)}{2}$ LUTs and $N-1$ LUCs are used to complete a MRS conversion. To accomplish the whole scaling by conversion to and from a positional representation, the latency and complexity of performing $Y = \lfloor \frac{X}{M} \rfloor$ in that positional system and the conversion back to RNS system should be added. Thus, such a scaling method based on MRS can be both complex and slow.

Scaling in the 3 Modulus RNS

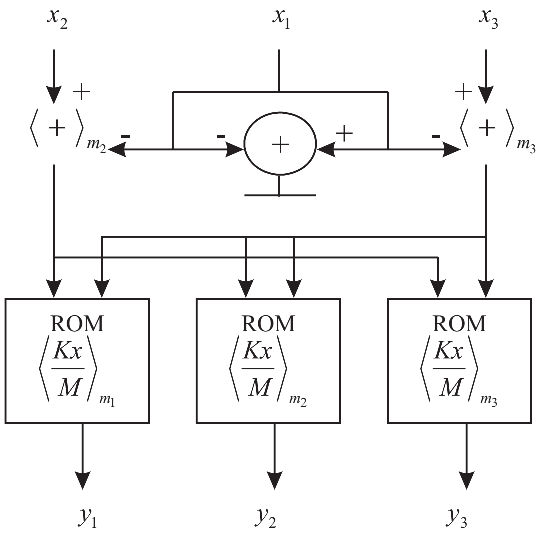
The RNS with modulus set $\{2^{k-1}, 2^k, 2^{k+1}\}$ has been the topic of considerable interest as efficient hardware mechanisms exist for reduction with respect to these moduli. An autoscale multiplier is described in [Taylor82] for this modulus set. The autoscale multiplier performs the operation $\lfloor \frac{KX}{M} \rfloor$ where K is a constant, which may lead to overflow of the product KX.

One of the motives of [Taylor82] was to incorporate the whole scaling process for each output digit, into one look-up table. However, at the time of publication, commercially available memory of an appropriate speed category was limited to a 12-bit address space. If this memory was used to implement scaling by conversion to a positional representation, the system would achieve a dynamic range that was considered too small for most applications.

To extend the dynamic range, [Taylor82] presented a scaling structure that is based on a partial conversion from the RNS to a positional binary representation using MRS conversion. The conversion is truncated at the point at which the result is within the addressing capacity of a high-speed memory. This truncation introduces some errors to the scaling. For the MRS conversion, $\bar{x}$ is defined as

$$\bar{x} = \langle X - x_1 \rangle_D = \{0, \langle x_2 - x_1 \rangle_{m_2}, \langle x_3 - x_1 \rangle_{m_3}\} = \{0, \bar{x_2}, \bar{x_3}\}.$$

Thus, the two residues $\bar{x_2}$ and $\bar{x_3}$ can represent an approximation to x. Figure 4.2 shows the autoscale architecture.



x_2
x_1
x_3
+
-
$\langle + \rangle_{m_2}$
+
+
-
$\langle + \rangle_{m_3}$
ROM $\left\langle \frac{Kx}{M} \right\rangle_{m_1}$
ROM $\left\langle \frac{Kx}{M} \right\rangle_{m_2}$
ROM $\left\langle \frac{Kx}{M} \right\rangle_{m_3}$
y_1
y_2
y_3

Figure 4.2: The 3 Modulus RNS Scaler

Using 12-bit ROMs an autoscale multiplier can be built with dynamic range $\frac{12N}{N-1}$ bits. Note however, that this scaling technique is most attractive for systems with 3 moduli as the effective dynamic range decreases with increased number of moduli. For example when $N = 3$, the system using 12-bit ROMs has an effective dynamic range of 18 bits; however the effective dynamic range is only 14 bits when $N = 7$.

This scaling algorithm works with an arbitrary value of M and requires only $O(N)$ LUTs and 1 LUC. This remarkably low latency is achieved at the cost of limited dynamic range. Also note that the scaling is not completed entirely within the RNS: the difference of residues from adjacent channels is required to address the LUTs.

Scaling using Estimation

Recall the Chinese Remainder Theorem (CRT) introduced in Section 2.1.2 on page 13. The Chinese Remainder Equation (2.1) is re-written here:

$$X = \left\langle \sum_i D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_D . \tag{2.1}$$

Also recall the definitions for RNS dynamic range $D = m_1 m_2 \dots m_N$, $D_i = \frac{D}{m_i}$ and $\langle D_i^{-1} \rangle_{m_i}$ in Section 2.1.1. For the RNS with moduli $\{m_1, \dots, m_s\}$, the CRT in (2.1) becomes

$$\langle X \rangle_M = \left\langle \sum_{i=1}^{S} M_i \langle M_i^{-1} x_i \rangle_{m_i} \right\rangle_M , \tag{4.1}$$

where $M_i = \frac{M}{m_i}$ and $\langle M_i^{-1} \rangle_{m_i}$ is its multiplicative inverse with respect to m_i. Many of the schemes surveyed [Jullien78, Griffin89, Shenoy89a, Barsi95] take M to be a product of several of the moduli because this permits the rapid evaluation of $\langle X \rangle_M$ from (4.1).

From $Y = \left\lfloor \frac{X}{M} \right\rfloor = \frac{X - \langle X \rangle_M}{M}$,

$$y_i = \langle Y \rangle_{m_i} = \left\langle \frac{X - \langle X \rangle_M}{M} \right\rangle_{m_i} = \left\langle \langle X - \langle X \rangle_M \rangle_{m_i} \times \langle M^{-1} \rangle_{m_i} \right\rangle_{m_i} , \tag{4.2}$$

where $\langle M^{-1} \rangle_{m_i}$ is the multiplicative inverse of M with respect to m_i. A sufficient condition for the existence of $\langle M^{-1} \rangle_{m_i}$ is that M and m_i are co-prime and $M \neq 0$ [Richman71]. Since $M = \prod_{i=1}^{S} m_i$, then $\langle M^{-1} \rangle_{m_i}$ does not exist for $1 \leq i \leq S$. Hence the choice of M as a product of a subset of the moduli has come at a cost: while Equation (4.2) above can be used to deduce y_i for $S + 1 \leq i \leq N$, a different method must be applied for $1 \leq i \leq S$.

In [Jullien78], an estimation algorithm with an absolute error of $\frac{S+1}{2}$ is used to derive y_i for $S+1 \leq i \leq N$. The estimated result is

$$y_i = \left\langle \sum_{j=1}^{S} A_i(x_j) + A_i(x_i) \right\rangle_{m_i}$$

where $A_i(x_j) = \left\langle \frac{D\langle D_j^{-1} x_j \rangle}{M m_j} \right\rangle_{m_i}$. This can be pre-computed and entered into the LUTs as illustrated in Figure 4.3. For this figure it is assumed that $r = 2$, $N = 6$ and $S = 3$ such that $M = \prod_{i=1}^{3} m_i$. It is possible to see that it takes $\lceil \log_r(S+1) \rceil$ LUCs and $(N-S)\left\lceil \frac{S}{r-1} \right\rceil$ LUTs to estimate y_i for $S+1 \leq i \leq N$.

Finding y_i for $1 \leq i \leq S$ is a typical process of base extension (BE), as introduced in Section 2.1.5. Note that the scaled result $Y < \prod_{i=1}^{S} m_i$ and can therefore be uniquely represented by $\{y_{S+1}, \ldots, y_N\}$ in the RNS with moduli $\{m_{S+1}, \ldots, m_N\}$. Base extending this $(N-S)$-channel representation of Y back into the original N-channel RNS gives y_i for the whole RNS modulus set $1 \leq i \leq N$. In the scaling scheme in [Jullien78], the BE is accomplished using a partial MRS conversion in a recursive form, as discussed in Section 2.1.2. This requires $N - S + \lceil \log_r(N-S) \rceil$ LUCs and $\left\lceil \frac{(N+S)(N-S-1)}{2} \right\rceil$ LUTs.

Because the base extension technique to generate y_i for $1 \leq i \leq S$ depends on the values of y_i for $S+1 \leq i \leq N$, it must occur subsequent to the estimation of these results, rather than in parallel. Therefore, $\lceil \log_r(S+1) \rceil + N - S \lceil \log_r(N-S) \rceil = N - S + \lceil \log_r(N+1) \rceil + 1$ LUCs are required for the scaling process, i.e. a time complexity of $O(N)$. A total of $(N-S)\left\lceil \frac{S}{r-1} \right\rceil + \left\lceil \frac{(N+S)(N-S-1)}{2} \right\rceil$ LUTs are required, i.e. space complexity of $O(N^2)$. With $O(N)$ LUCs and $O(N^2)$ LUTs, this scaling scheme is an improvement over the one using MRS; however the error of up to $\pm\frac{S+1}{2}$ is larger than most other scaling schemes. Moreover, the base extension method processes numbers of long word lengths outside of the RNS and this decreases the parallelism available in the second half of the scaling process.

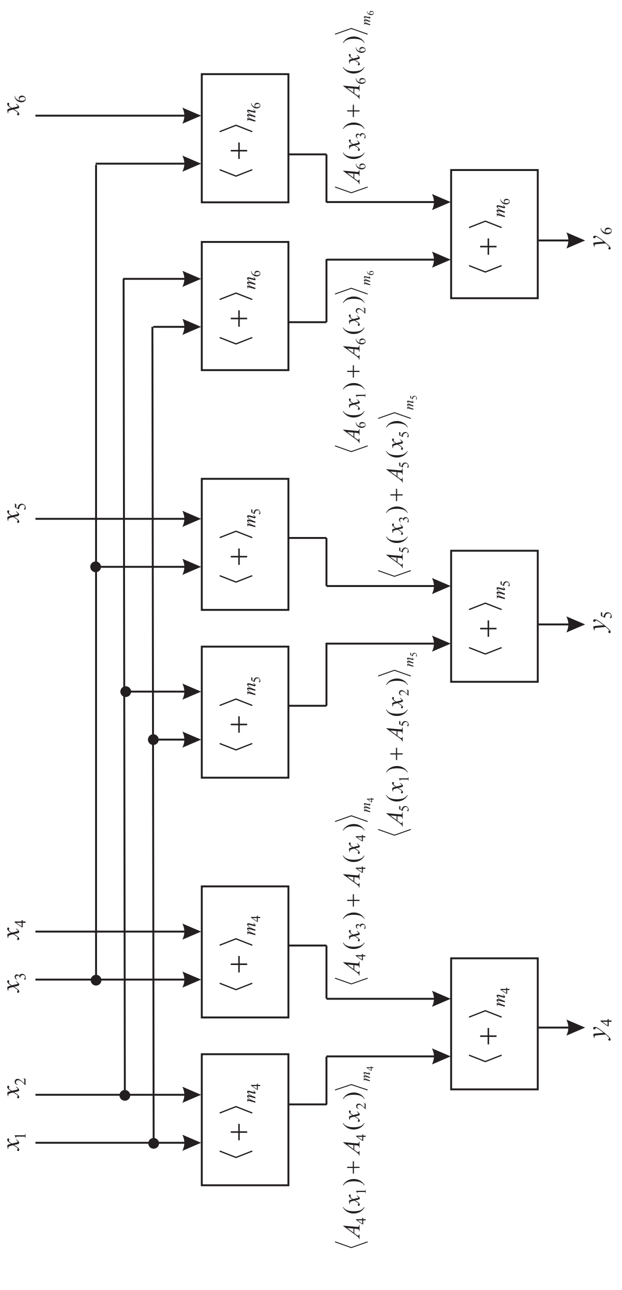

x_1
x_2
x_3
x_4
x_5
x_6
$\langle + \rangle_{m_4}$
$\langle + \rangle_{m_5}$
$\langle + \rangle_{m_6}$
$\langle A_4(x_1)+A_4(x_2)\rangle_{m_4}$
$\langle A_4(x_3)+A_4(x_4)\rangle_{m_4}$
$\langle A_5(x_1)+A_5(x_2)\rangle_{m_5}$
$\langle A_5(x_3)+A_5(x_5)\rangle_{m_5}$
$\langle A_6(x_1)+A_6(x_2)\rangle_{m_6}$
$\langle A_6(x_3)+A_6(x_6)\rangle_{m_6}$
y_4
y_5
y_6

Figure 4.3: RNS Scaling using Estimation for Channels $S+1 \leq i \leq N$

Scaling using a Decomposition of the CRT

The scaling schemes in [Shenoy89a, Griffin89] and [Aichholzer93] are based on a decomposition of the CRT. The scheme from [Shenoy89a] achieves the more remarkable effect in reducing the latency within the scaling process. It also represents a significant advance towards the goal of performing as much of the scaling as possible within the RNS. It decreases the required number of LUTs to $O(\log_r N)$ by expressing the scaled integer Y as a summation of terms that can be evaluated in parallel:

$$Y = \left\lfloor \frac{X}{M} \right\rfloor \approx \sum_{i=1}^{N} f(x_i). \tag{4.3}$$

This equation is typical of residue arithmetic processes that achieve $O(\log_r N)$ time complexity. This will be deduced shortly in the last subsection.

Once again the scaling in [Shenoy89a] is performed by a product of some of the moduli $M = \prod_{i=1}^{S} m_i$. Substituting the two CRT equations (2.1) and (4.1) into $Y = \lfloor \frac{X}{M} \rfloor = \frac{X - \langle X \rangle_M}{M}$ yields

$$Y = \frac{\left\langle \sum_{i=1}^{N} D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_D - \left\langle \sum_{i=1}^{S} M_i \langle M_i^{-1} x_i \rangle_{m_i} \right\rangle_M}{M}. \tag{4.4}$$

Two parameters, r_x and r'_x, are then introduced [Shenoy89b] to reduce the two large modulo operations, $\langle \sum_i \cdot \rangle_D$ and $\langle \sum_i \cdot \rangle_M$, in (4.4). It can be derived that

$$Y = r'_x - r_x \frac{D}{M} + \sum_{i=S+1}^{N} \frac{D_i}{M} \langle D_i^{-1} x_i \rangle_{m_i} + \sum_{i=1}^{S} \frac{1}{m_i} \left(\frac{D}{M} \langle D_i^{-1} x_i \rangle_{m_i} - \langle M_i^{-1} x_i \rangle_{m_i} \right). \tag{4.5}$$

By a novel decomposition of the CRT [Shenoy89b], r_x and r'_x can be calculated by constructing 2 redundant channels in the RNS scaler as

$$r_x = \left\langle D^{-1} \left(\sum_{i=1}^{N} \left\langle D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_{m_r} - x_r \right) \right\rangle_{m_r}$$

and

$$\acute{r_x} = \left\langle M^{-1} \left\langle \sum_{i=1}^{S} M_i \langle M_i^{-1} x_i \rangle_{m_i} - \langle \acute{x} \rangle_{\acute{m_r}} \right\rangle_{\acute{m_r}} \right\rangle_{\acute{m_r}},$$

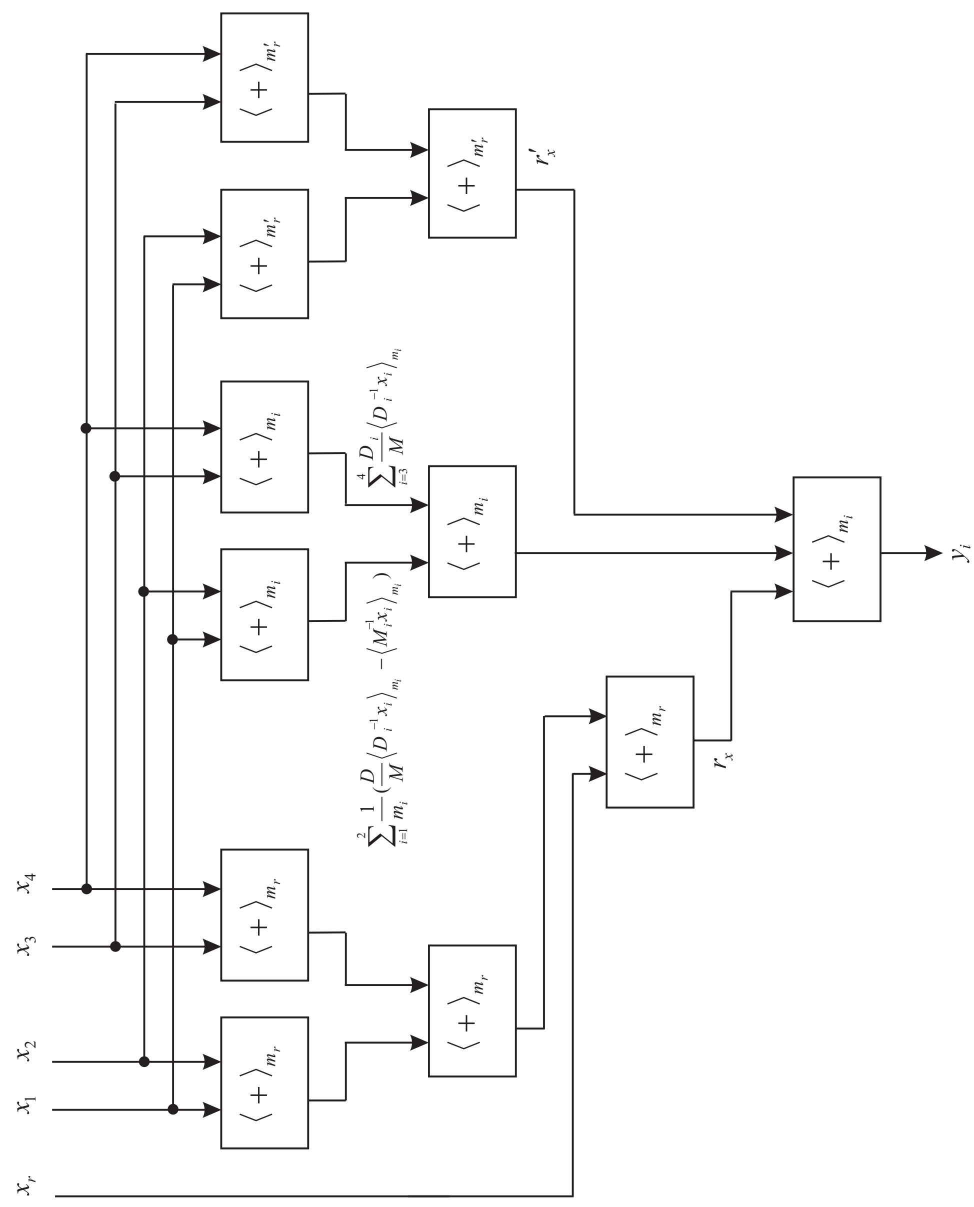
where m_r and $\acute{m_r}$ are additional RNS moduli. The channel m_r with its residue x_r needs to be maintained during other operations. This is because the $\acute{x}$ required to deduce $\acute{r_x}$ can be obtained through the original residues, x_i, by

$$\acute{x} = M_l \left\langle \sum_{i=1}^{S} \frac{\langle M_i^{-1} x_i \rangle_{m_i}}{M_l} \right\rangle_{m_l} = \left\langle \sum_{i=1}^{S} M_i \langle M_i^{-1} x_i \rangle_{m_i} \right\rangle_{M_l}$$

where m_l is chosen to be the modulus among the S moduli such that $m_l \geq S$. This leads to a maximum error of ± 1 in the scaled result [Shenoy89a]. The factors r_x and $\acute{r_x}$ need only be computed once within the scaling and can then be broadcast to all of the channels. The output residues y_i, can then be obtained by performing a modulo m_i operation on both sides of Equation (4.5). By implementing the channels in parallel it is possible to compute y_i in $O(\log_r N)$ LUCs. Figure 4.4 shows the block diagram of one modulus channel and the two redundant channels of this scaler for the case $r = 2$, $N = 4$ and $S = 2$.

It is also proved [Shenoy89b] that the sum of x_r and $\acute{x_r}$ is at most $\log_2(NS)$ bits long. Hence for the example shown in Figure 4.4, the sum of x_r and $\acute{x_r}$ is at most 3 bits. Thus, even though $r = 2$, it is possible to input x_r, $\acute{x_r}$ and the result of the sum terms into a single ROM in the last look-up cycle. Note that the size of the two redundant channels increases with the number of moduli in the RNS system. For example, in the RNS with modulus set $\{7, 11, 13, 17, 19, 23\}$, if $S = 3$ then the redundant channel width is $\log_2(NS) \approx 4.17$ bits. Given $D = 7436429 \approx 22.83$ bits the redundant channels represent 18.27% percent of dynamic range.

The total number of look-up tables is $\left\lceil \frac{N^2+(r+2)N+r-2}{r-1} \right\rceil$ which is $O(N^2)$. The major advantage of this scheme is its speed. Scaling by a product



x_4
x_3
x_2
x_1
x_r
$\langle + \rangle_{m_r'}$
$\langle + \rangle_{m_i}$
$\langle + \rangle_{m_r}$
r_x'
r_x
y_i
$\sum_{i=3}^{4} \frac{D_i}{M} \langle D_i^{-1} x_i \rangle_{m_i}$
$\sum_{i=1}^{2} \frac{1}{m_i} (\frac{D}{M} \langle D_i^{-1} x_i \rangle_{m_i} - \langle M_i^{-1} x_i \rangle_{m_i})$

Figure 4.4: RNS Scaling using a Decomposition of the CRT (two redundant channels with channel m_i)

of several moduli is achieved within $\lceil \log_r(N+1) \rceil + \lceil \log_r 3 \rceil$ LUCs which is $O(\log_r N)$. However, two extra moduli are employed in two redundant channels that require extra hardware. This represents a hardware overhead of $\log_D(NS)$ percent (typically $> 10\%$). Moreover, the channel m_r should be maintained in other RNS operations such that the modulus set become $\{m_1, \ldots, m_N, m_r\}$.

Scaling using Parallel Base Extension

The scaling scheme shown in Figure 4.3 takes a product of some moduli as the scaling factor M and uses MRS to base extend $\{y_{S+1}, \ldots, y_N\}$ to $\{m_1, \ldots, m_S\}$. The scaling scheme in [Barsi95] uses the same form of M but this time a base extension is used to generate all of the y_i. The simplicity of this scaling scheme is due to the innovation of a parallel base extension technique. One of the significant goals of the authors was to perform scaling without the two redundant channels required by the scheme shown in Figure 4.4 and without increasing the latency and hardware overhead.

Scaling by $M = \prod_{i=1}^{S} m_i$ can be achieved with two base extension steps. Firstly the representation of $\langle X \rangle_M$ in $\{x_1, x_2, \ldots, x_S\}$ is base extended to the moduli $\{m_{S+1}, m_{S+2}, \ldots, m_N\}$ to obtain a representation of $\langle X \rangle_M$ in the whole dynamic range [0, M). In the base extension scheme of [Barsi95] this costs $O(\log_r S)$ LUCs. One can then use:

$$\begin{aligned} Y &= \left\lfloor \frac{X}{M} \right\rfloor = \frac{X - \langle X \rangle_M}{M} = \left\langle \frac{X - \langle X \rangle_M}{M} \right\rangle_{\frac{D}{M}} \\ &= \left\langle \left\langle M^{-1} \right\rangle_{\frac{D}{M}} \left(\langle X \rangle_{\frac{D}{M}} - \langle \langle X \rangle_M \rangle_{\frac{D}{M}} \right) \right\rangle_{\frac{D}{M}} \end{aligned}$$

to deduce $\{y_{S+1}, y_{S+2}, \ldots, y_N\}$. This requires one LUC and $N - S$ LUTs. Finally one can base extend this result back to $\{y_1, y_2, \ldots, y_S\}$ to obtain the final representation of y. This base extension takes $O(\log_r(N - S))$ LUCs.

The entire scaling process requires $O(\log_r S)+O(\log_r(N-S))+1 \approx O(\log_r N)$ LUCs and is shown in Figure 4.5 where the BE blocks perform base extensions using the hardware structure described in [Barsi95]. A total of $O(N^2)$ LUTs are required.

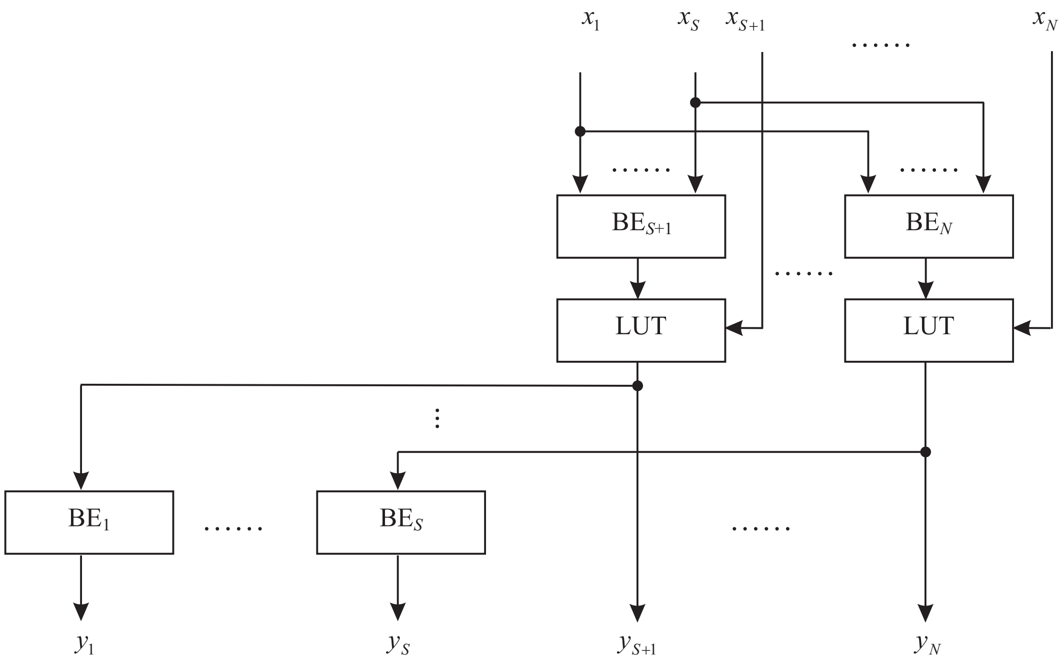


Figure 4.5: RNS Scaling using Parallel Base Extension (BE) Blocks

Note that another two publications [Ulman98] and [Garcia99] take alternative approaches to this same problem but both of them involve larger LUTs. The latter replaces the base extension blocks with large LUTs with up to $S + 1$ inputs. For a RNS with more than 3 5-bit channels, such large LUTs are not available and hence the scheme is only viable in some specific cases as stated in [Garcia99].

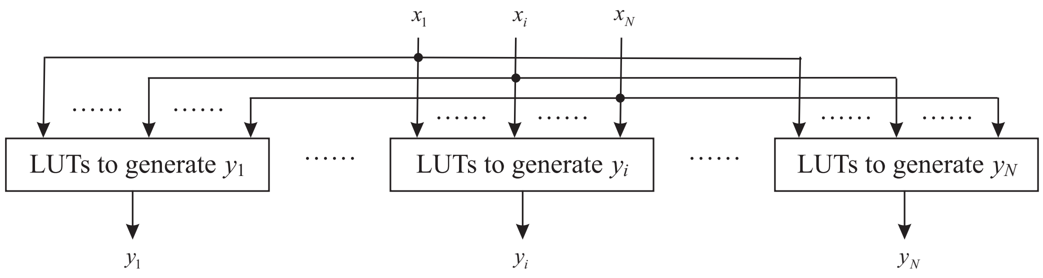


Figure 4.6: Parallel Architecture to Perform $Y = \left\lfloor \frac{X}{M} \right\rfloor \approx \sum_{i=1}^{N} f(x_i)$.

The Time Complexity of $O(\log_r N)$

A ROM network to perform (4.3)

$$Y = \left\lfloor \frac{X}{M} \right\rfloor \approx \sum_{i=1}^{N} f(x_i) \tag{4.3}$$

is shown in Figure 4.6. A more detailed diagram appears in Figure 4.7. This is a typical architecture of residue arithmetic processes that achieve $O(\log_r N)$ time complexity. Many scaling schemes [Shenoy89a, Barsi95, Garcia99, Posch95, Ulman98, Aichholzer93] can be implemented with this structure. In this subsection, the time complexity of residue arithmetic structures following Equation (4.3) is derived.

Suppose each available LUT can accept only r inputs at most while generating only one output as discussed in Section 4.1.1 above. Then, each channel of the parallel scaling structure in Figure 4.6 can be drawn as a tree as in Figure 4.7, where it is assumed that there are N input residues and l look-up cycles are consumed to accomplish the scaling in channel i.

In the first cycle, the number of LUTs is $\lceil N/r \rceil$. Thus, there are $\lceil N/r \rceil$ input residues to the LUTs in the second cycle, where the number of LUTs will be $\lceil \lceil N/r \rceil /r \rceil$. This proceeds recursively until only one LUT is needed, i.e. $\lceil \ldots \lceil \lceil N/r \rceil /r \rceil \ldots /r \rceil = 1$ as illustrated in Figure 4.7. Using the result from Number Theory [Richman71], $\lceil N/r^2 \rceil = \lceil \lceil N/r \rceil /r \rceil$, gives the number

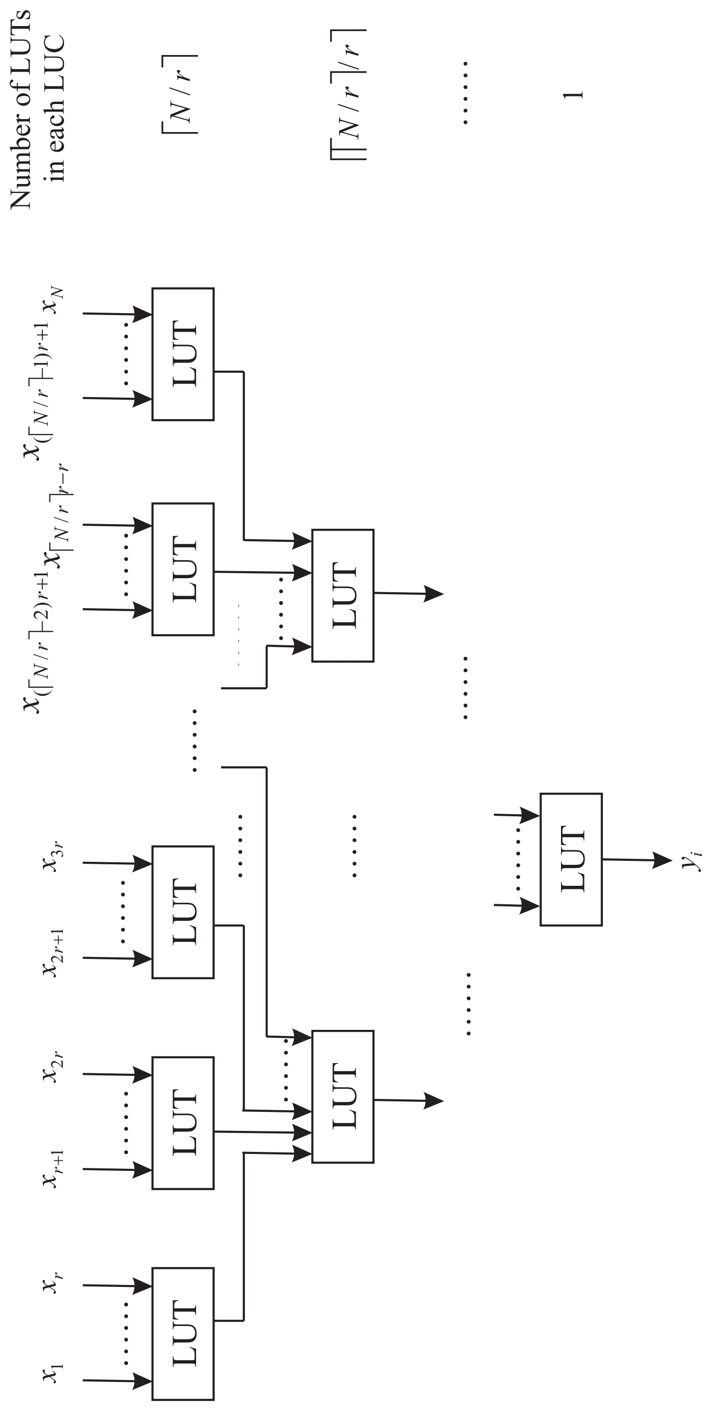

Number of LUTs in each LUC
$\lceil N/r \rceil$
$\lceil \lceil N/r \rceil / r \rceil$
1
x_1
x_r
x_{r+1}
x_{2r}
x_{2r+1}
x_{3r}
$x_{(\lceil N/r \rceil - 2)r+1}$
$x_{\lceil N/r \rceil r - r}$
$x_{(\lceil N/r \rceil - 1)r+1}$
x_N
LUT
y_i

Figure 4.7: One Channel of a Parallel Residue Arithmetic Process using Memories with Addressing Capacity $= r$.

of LUTs as $\lceil N/r \rceil$ in the first cycle, $\lceil N/r^2 \rceil$ in the second and so on, until the last cycle, where $\lceil \ldots \lceil \lceil N/r \rceil /r \rceil \ldots /r \rceil = \lceil N/r^l \rceil = 1$. Then from $N/r^l \leq \lceil N/r^l \rceil = 1$, we have, $N/r^l \leq 1 \Rightarrow l \geq \log_r N$.

If $0 < N/r^{l-1} \leq 1$, then $\lceil N/r^{l-1} \rceil = 1$. This means only $l-1$ cycles are needed and this contradicts our original assumption that l cycles are required. Therefore, $N/r^{l-1} > 1 \Rightarrow l < \log_r N + 1$ and $\log_r N \leq l < \log_r N + 1$, so that

$$l = \lceil \log_r N \rceil .$$

This represents the exact time complexity of the i-th channel of the residue arithmetic process shown in Figure 4.7. Because all N channels run in parallel, $\lceil \log_r N \rceil$ is also the exact time complexity of the scaling scheme constructed using r-input LUTs.

It can also be proved that the exact space complexity of each channel is $\left\lceil \frac{N-1}{r-1} \right\rceil$ and so the exact space complexity of the whole arithmetic process is $N \left\lceil \frac{N-1}{r-1} \right\rceil$, which is at the level of $O(N^2)$.

4.1.3 A New RNS Scaling Scheme using Look-Up Tables

This section presents a new RNS short word length scaling scheme using LUTs. For those schemes surveyed in Section 4.1.2, $O(\log_r N)$ and $O(N^2)$ are the lowest time and hardware complexity respectively. Furthermore, the scaling factor M is usually chosen to be a product of some of the moduli $M = \prod_{i=1}^{S} m_i$. This scheme decreases hardware complexity to $O(N)$ without affecting time complexity. Also, it allows M to be any number co-prime to the RNS moduli.

The New Scaling Scheme

Consider Equation (4.2)

$$y_i = \left\langle \langle X - \langle X \rangle_M \rangle_{m_i} \times \langle M^{-1} \rangle_{m_i} \right\rangle_{m_i}.$$

This is equivalent to

$$y_i = \left\langle \langle x_i - \langle X \rangle_M \rangle_{m_i} \times \langle M^{-1} \rangle_{m_i} \right\rangle_{m_i}. \tag{4.6}$$

Because M is co-prime with all m_i, $\langle M^{-1} \rangle_{m_i}$ always exists and (4.6) can be used to evaluate y_i in every channel. For a constant M, $\langle M^{-1} \rangle_{m_i}$ can be pre-computed and stored in a LUT. Note that [Meyer-Bäse03] presents a similar scheme by fixing $M = 2$.

The first step in this scheme is to evaluate $\langle X \rangle_M$ from the RNS representation of X, i.e. $\{x_1, x_2, \ldots, x_N\}$. This again is a typical base extension problem as long as M is not too large. The base extension approach in [Barsi95] can then be used.

As there are only two inputs to (4.6), x_i and $\langle X \rangle_M$, (4.6) can be implemented using a single LUT for each output residue y_i provided the word length of M is at most $(r - 1)$ times the RNS channel width w. In this case the scaling step only uses 1 LUC and N LUTs.

For example, if $r = 3$ and the channel width is $w = 5$ bits, the addressing capacity of each LUT is $2^{3\times 5} \times 5 = 32\text{K} \times 5$ bits. In this case M can be as large as $(r - 1) \times 5 = 10$ bits. M can be made larger if a larger LUT is used or several LUTs are concatenated to allow more addressing capacity. In the example above, if the largest available LUT is $512\text{K} = 64\text{K} \times 8$, i.e. $2^{M+5} \leq 64\text{K}$, then the scaling factor M can be as large as $\log_2 64 + 10 - 5 = 11$ bits.

The scaling scheme is illustrated in Figure 4.8. The base extension block costs $O(\log_r N)$ LUCs and $O(N)$ LUTs, and the scaling step consumes 1 LUC and N LUTs. Thus the time complexity of this new scaling process is $O(\log_r N) + 1 = O(\log_r N)$ and the space complexity is $O(N) + N = O(N)$. The latter is an improvement for the scaling problem in RNS using LUTs since all other methods incur a hardware cost of $O(N^2)$. The main reason is they need $O(N)$ LUTs to scale in one channel. When scaling over N channels, their space complexities become $O(N^2)$.

More specifically, suppose the base extension block in Figure 4.5 is used, which has an exact time complexity of $\log_r N + 2$ and an exact space complexity of $2\left\lceil \frac{N-1}{r-1} \right\rceil + 4$. The exact time complexity of this new scaling process is $\log_r N + 3$ and the exact space complexity is $2\left\lceil \frac{N-1}{r-1} \right\rceil + N + 4$.

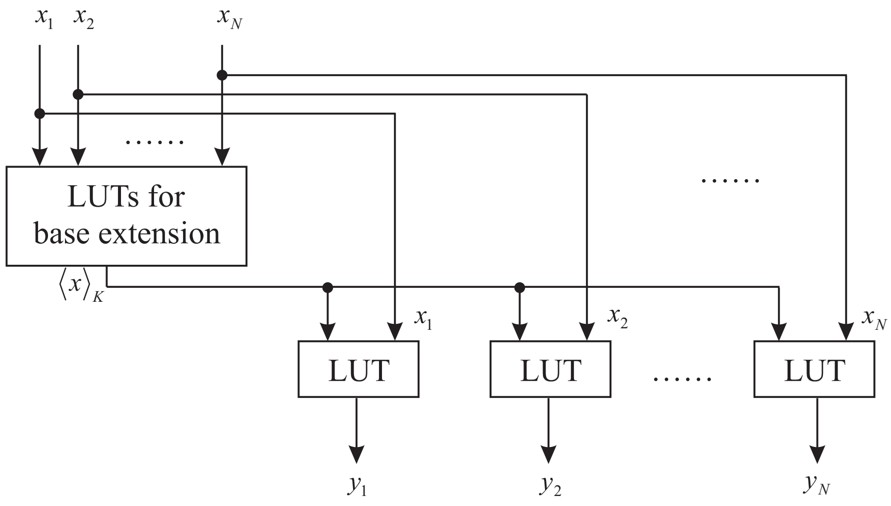


Figure 4.8: A New Architecture to Perform Short Word length Scaling in RNS

An Example

As an example, consider the RNS moduli $m_1 = 23$, $m_2 = 25$, $m_3 = 27$, $m_4 = 29$ and $m_5 = 31$, and suppose the integer $X = 578321 = \{9, 21, 8, 3, 16\}$ is to be scaled by $M = 1039$. $\langle M^{-1} \rangle_{m_i}$ has been pre-computed as $\{6, 9, 25, 23, 2\}$

for $1 \leq i \leq 5$. Base extend X to M to compute $\langle X \rangle_M = 637$. Then, according to (4.6), the scaled residues are computed as $y_1 = \langle \langle 9 - 637 \rangle_{23} \times 6 \rangle_{23} = 4$, $y_2 = \langle \langle 21 - 637 \rangle_{25} \times 9 \rangle_{25} = 6$, $y_3 = \langle \langle 8 - 637 \rangle_{27} \times 25 \rangle_{27} = 16$, $y_4 = \langle \langle 3 - 637 \rangle_{29} \times 23 \rangle_{29} = 5$, and $y_5 = \langle \langle 16 - 637 \rangle_{31} \times 2 \rangle_{31} = 29$. Thus, $Y = \{4, 6, 16, 5, 29\} = 556 = \left\lfloor \frac{578321}{1039} \right\rfloor$. Note that in this example all operations can be performed using 64K $\times$ 8 bit LUTs.

Evaluation

Table 4.1 shows a comparison between the 5 surveyed scaling schemes with the new one.

Note that none of these schemes can be used for modular multiplication of over 1024 bits because their LUT implementations are not suitable for the RNS with long dynamic ranges and wide channel widths. Long word length modular multiplication is discussed in more detail in the following 4 sections.

Table 4.1: A Comparison of 6 Short Word length RNS Scaling Schemes using LUTs

Scaling scheme	**Scaling factor** M	**LUCs**	**LUTs**	**Limitation**
Scaling using MRS conversion [Szabo67]	Any integer within $[0, D)$	$O(N)$	$O(N^2)$	High overhead
3 modulus scaling [Taylor82]	Any integer within $[0, D)$	1	$O(N)$	Narrow D; only suitable for 3 moduli
Scaling using estimation [Jullien78]	A product of some moduli	$O(N)$	$O(N^2)$	Large error of $\frac{S+1}{2}$; high complexity generating y_i for $1 \leq i \leq S$
Scaling using a decomposition of the CRT [Shenoy89a]	A product of some moduli	$O(\log_r N)$	$O(N^2)$	Two redundant channels; one to be maintained
Scaling using parallel base extension [Barsi95]	A product of some moduli	$O(\log_r N)$	$O(N^2)$	High overhead in some rare cases
The new scaling	Any integer $< (r-1)w$ and co-prime to m_i	$O(\log_r N)$	$O(N)$	Low hardware complexity; dependent on base extension

4.2 RNS Classical Modular Multiplication

This section describes a Classical algorithm for modular multiplication in the Residue Number System. Recall the Classical modular multiplication algorithm for modular multiplication in a positional system introduced in Section 3.1.1. In the Classical algorithm, the modular reduction is performed by subtracting a multiple of the modulus from the current partial product, as in Algorithm 3.4. Typically $X \mod M = X - Y \times M$ is evaluated following a step called *quotient digit selection* to decide the multiple $Y = \left\lfloor \frac{X}{M} \right\rfloor$. Quotient digit selection involves a comparison between the magnitudes of X and M. Thus a way is sought to quickly determine the relative magnitudes of RNS numbers. This is not easy in a RNS. The solution developed here uses the Core Function [Akushskii77, Miller83] for this purpose.

Existing RNS algorithms using the Core Function for scaling and conversion between RNS and positional systems can be found in [Burgess97] and [Burgess03]. Their drawback is that they do not support sufficiently long word lengths for public-key cryptography.

4.2.1 The Core Function

The Core $C(X)$

The core function [Gonnella91] for a RNS $\{m_1, m_2, \ldots, m_N\}$ is defined for an integer X as

$$C(X) = \sum_i w_i \left\lfloor \frac{X}{m_i} \right\rfloor = \sum_i (X - x_i)\frac{w_i}{m_i} = X \sum_i \frac{w_i}{m_i} - \sum_i \frac{x_i w_i}{m_i} \tag{4.7}$$

where x_i is the channel residue of $X \mod m_i$. The w_is are arbitrary constants known as the *channel weights*. Setting $X = D$, where D is the dynamic range of the RNS, in (4.7) gives

$$C(D) = \sum_i w_i \left\lfloor \frac{D}{m_i} \right\rfloor = \sum_i w_i \frac{D}{m_i} = D \sum_i \frac{w_i}{m_i}. \tag{4.8}$$

Hence,

$$\frac{C(D)}{D} = \sum_i \frac{w_i}{m_i}. \tag{4.9}$$

Substituting (4.9) into (4.7) gives

$$C(X) = \frac{XC(D)}{D} - \sum_i \frac{x_i w_i}{m_i}. \tag{4.10}$$

This is the commonly used form of the core function. It shows a nearly linear plot of $C(X)$ against X with a slope $\frac{C(D)}{D}$ and some noise due to the summation term $\sum_i \frac{x_i w_i}{m_i}$ [Soderstrand86], as illustrated in Figure 4.9 from [Burgess03]. Therefore, $C(X)$ could be used as a rough representation of the magnitude of X, as required for a classical modular reduction algorithm. The magnitude of the noise depends on the magnitudes of the weights w_i and the specific value of $C(D)$ for a given RNS [Burgess97].

Let $\mathrm{Mag}(X) = \frac{XC(D)}{D}$ and $\mathrm{Noise}(X) = \sum_i \frac{x_i w_i}{m_i}$, then (4.10) becomes

$$\mathrm{Mag}(X) = C(X) + \mathrm{Noise}(X). \tag{4.11}$$

Therefore,

$$\frac{C(X)}{C(M)} \approx \frac{\mathrm{Mag}(X)}{\mathrm{Mag}(M)} = \frac{X}{M}. \tag{4.12}$$

The Chinese Remainder Theorem for a Core Function

In this subsection, let us derive the Chinese Remainder Theorem for a Core Function, as in [Soderstrand86]. From (4.8), we have

$$C(D) = \sum_i w_i \frac{D}{m_i} = \sum_i D_i w_i. \tag{4.13}$$

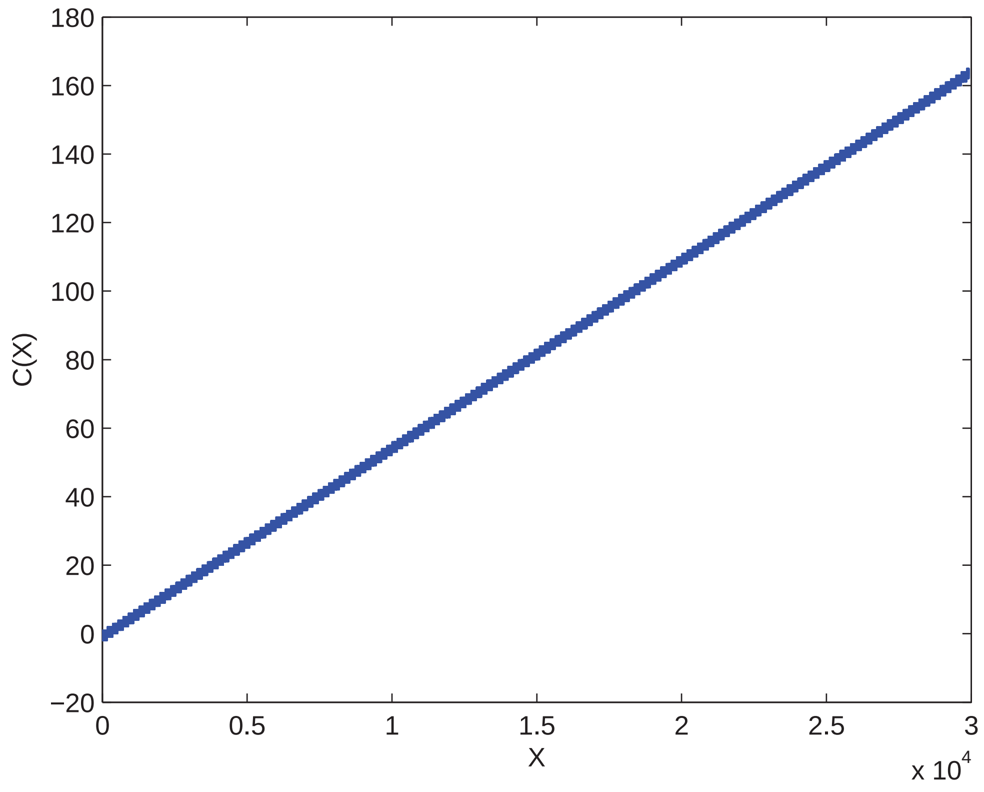


Figure 4.9: A Typical Core Function C(X): $D = 30030$; $C(D) = 165$

A modular reduction, with respect to m_j, of both sides of this equation yields

$$\langle C(D) \rangle_{m_j} = \left\langle \sum_i D_i w_i \right\rangle_{m_j} = \langle D_j w_j \rangle_{m_j} .$$

Then,

$$w_j = \left\langle C(D) D_j^{-1} \right\rangle_{m_j} . \tag{4.14}$$

Hence, the weights can be decided once $C(D)$ has been chosen so long as Equation (4.13) is satisfied. This implies that some of the weights must be negative to ensure $C(D) \ll D$.

Again recall the Chinese Remainder Theorem (CRT) introduced in Section 2.1.2 on page 13. The Chinese Remainder Equation (2.1) gives

$$X = \left\langle \sum_i D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_D , \tag{2.1}$$

where D, D_i and $\langle D_i^{-1} \rangle_{m_i}$ are pre-computed constants. Let $\sigma_i = D_i \langle D_i^{-1} \rangle_{m_i}$, which can also be pre-computed, so that the CRT equation becomes

$$X = \left\langle \sum_i \sigma_i x_i \right\rangle_D = \sum_i \sigma_i x_i - \alpha D, \tag{4.15}$$

where α is an unknown integer. Substituting (4.15) into the basic core function (4.10) yields

$$\begin{aligned} C(X) &= \frac{C(D)}{D}(\sum_i \sigma_i x_i - \alpha D) - \sum_i \frac{x_i w_i}{m_i} \\ &= \sum_i x_i (\frac{\sigma_i C(D)}{D} - \frac{w_i}{m_i}) - \alpha C(D). \end{aligned}$$

Setting $X = \sigma_i = D_i \langle D_i^{-1} \rangle_{m_i}$ in (4.10),

$$\begin{aligned} C(\sigma_i) &= \frac{\sigma_i C(D)}{D} - \sum_j \frac{\langle D_i D_i^{-1} \rangle_{m_j} w_j}{m_j} \\ &= \frac{\sigma_i C(D)}{D} - \frac{w_i}{m_i} \\ &= \frac{D_i^{-1} C(D) - w_i}{m_i}. \end{aligned} \tag{4.16}$$

Therefore,

$$C(X) = \sum_i x_i C(\sigma_i) - \alpha C(D).$$

Then we have the Chinese Remainder Theorem for a core function as

$$\langle C(X) \rangle_{C(D)} = \left\langle \sum_i x_i C(\sigma_i) \right\rangle_{C(D)} , \tag{4.17}$$

where $C(\sigma_i)$ can be pre-computed using (4.16).

An example should make things clearer. Consider a RNS {17, 19, 23, 25, 27, 29, 31}, giving $D = 4508102925$ and $\langle D_i^{-1} \rangle_{m_i} = \{1, 15, 13, 3, 8, 4, 1\}$. Choose $C(D) = 2^8 = 256$. Then, from (4.14), the weights are found as

$$\begin{aligned}
w_1 &= \langle 256 \times 1 \rangle_{17} = 1 \text{ or } -16 \\
w_2 &= \langle 256 \times 15 \rangle_{19} = 2 \text{ or } -17 \\
w_3 &= \langle 256 \times 13 \rangle_{23} = 16 \text{ or } -7 \\
w_4 &= \langle 256 \times 3 \rangle_{25} = 18 \text{ or } -7 \\
w_5 &= \langle 256 \times 8 \rangle_{27} = 3 \text{ or } -4 \\
w_6 &= \langle 256 \times 4 \rangle_{29} = 9 \text{ or } -20 \\
w_7 &= \langle 256 \times 1 \rangle_{31} = 8 \text{ or } -23
\end{aligned}$$

In order to minimize the noise in the core function, weights with small magnitudes should be chosen. In this example, the weight set is selected as $w_i = \{1, 2, -7, -7, -4, 9, 8\}$. For w_5, -4 is selected rather than 3 because w_5 needs to be negative to offset the positive magnitude of the sum of the other 6 weights. The legitimacy of these weights can be checked against Equation (4.13):

$$\begin{aligned}
C(D) &= 1 \times 265182525 + 2 \times 237268575 - 7 \times 196004475 - 7 \times 180324117 \\
&\quad -4 \times 166966775 + 9 \times 155451825 + 8 \times 145422675 \\
&= 256,
\end{aligned}$$

as required.

4.2.2 A Classical Modular Multiplication Algorithm in RNS using the Core Function

This section presents a new algorithm for modular multiplication in RNS using the Core Function. Following an efficient RNS multiplication $X =$

$A \times B$, the RNS modular reduction is to compute $X \mod M = X - Y \times M$, where

$$Y = \left\lfloor \frac{X}{M} \right\rfloor = \left\lfloor \frac{\mathrm{Mag}(X)}{\mathrm{Mag}(M)} \right\rfloor \tag{4.18}$$

as $\mathrm{Mag}(X) = \frac{XC(D)}{D}$ and $\mathrm{Mag}(M) = \frac{MC(D)}{D}$. An algorithm implementing this depends on the efficiency of computing $\mathrm{Mag}(X)$ using (4.11) as $\mathrm{Mag}(M)$ can be pre-computed. Note that for a modular reduction the algorithm does not have to produce the exact Y. The result will satisfy $X \mod M \equiv X - Y \times M$ and for many applications it is sufficient that the word length of the result is consistent from each round of modular multiplication.

The advantage of the core function lies in the fact that a number $C(X)$, computed through Equation (4.17), is used to approximate the scaled magnitude of X, $\mathrm{Mag}(X)$, as shown in (4.12). Therefore, $Y = \left\lfloor \frac{\mathrm{Mag}(X)}{\mathrm{Mag}(M)} \right\rfloor \approx \left\lfloor \frac{C(X)}{C(M)} \right\rfloor$ is always used in previously published RNS scaling techniques using core functions. However, there are two unsolved problems as stated below.

Two Problems with the Core Function used in RNS Modular Reduction

First, because $C(X)$ has to be large enough to minimize the error incurred by the noise term in (4.11), it has to be very large for a long word length X as used in public-key cryptosystems. Second, the modular reduction $\langle C(X) \rangle_{C(D)}$ has to be performed in (4.17). This is hard to do in a RNS. These are the reasons why most of the previously published techniques for RNS scaling using the core function [Burgess97, Burgess03, Miller83, Soderstrand86] are difficult to apply to long word length operations. The following work gives a Classical Modular Reduction Algorithm for RNS using the Core Function which addresses these two problems.

First, both terms, $C(X)$ and $\mathrm{Noise}(X)$, are computed in the algorithm instead of the former only, namely, $Y = \left\lfloor \frac{\text{Mag}(X)}{\text{Mag}(M)} \right\rfloor \approx \left\lfloor \frac{C(X)+\text{Noise}(X)}{\text{Mag}(M)} \right\rfloor$. This requires an efficient algorithm to find Noise(X). A suitable algorithm for this is described later. The advantage of this approach is that $C(X)$ does not need to be very large as Mag(X) can be accurately derived. Also, $C(D)$ can be selected to be a power of 2 so that $\langle C(X)\rangle_{C(D)}$ can be computed efficiently outside the RNS.

Pre-Computations

Let $C(D) = 2^h < M$, then Equation (4.16) becomes $C(\sigma_i) = \frac{2^h D_i^{-1} - w_i}{m_i}$. Both $C(\sigma_i)$ and $\text{Mag}(M) = \frac{2^h M}{D}$ can be pre-computed. As in [Miller83], the dynamic range D of the RNS is selected to satisfy $\sqrt{D} > M$ and then $\text{Mag}(M) = \frac{2^h M}{D} < 1$. Therefore, $\left\lfloor \frac{1}{\text{Mag}(M)} \right\rfloor$ has to be pre-computed instead of Mag(M). Then Equation (4.18) becomes

$$Y = \left\lfloor \frac{X}{M} \right\rfloor = \left\lfloor \frac{\text{Mag}(X)}{\text{Mag}(M)} \right\rfloor \approx \lfloor \text{Mag}(X) \rfloor \times \left\lfloor \frac{1}{\text{Mag}(M)} \right\rfloor. \tag{4.19}$$

This will give Y a small error of $\left\lceil \text{Mag}(X) + \frac{1}{\text{Mag}(M)} \right\rceil$ at most. This means that the final result from (4.19) may, on occasion, be too great by several factors of M. In an implementation of RSA cryptography, for example, this error can be carried through a long series of modular multiplications and only corrected at the end. In this case the user must ensure that the dynamic range of the RNS, D, is sufficiently large to hold intermediate values without overflow. Typically this incurs a cost of a few extra RNS channels.

Although $\left\lfloor \frac{1}{\text{Mag}(M)} \right\rfloor$ is pre-computed, unfortunately sometimes $\text{Mag}(X) = \frac{2^h X}{D} < 1$, which makes $\lfloor \text{Mag}(X) \rfloor = 0$. A scaling factor L is needed to increase Mag(X) so that Equation (4.19) becomes

$$Y = \left\lfloor \frac{X}{M} \right\rfloor = \left\lfloor \frac{L \times \text{Mag}(X)}{L \times \text{Mag}(M)} \right\rfloor \approx \lfloor L \times \text{Mag}(X) \rfloor \times \left\lfloor \frac{1}{L \times \text{Mag}(M)} \right\rfloor, \tag{4.20}$$

where L is an integer such that $L \times \mathrm{Mag}(X) > 1$ and $L \times \mathrm{Mag}(M) < 1$. This makes both $\lfloor L \times \mathrm{Mag}(X) \rfloor > 0$ and $\left\lfloor \frac{1}{L \times \mathrm{Mag}(M)} \right\rfloor > 0$. Here L is the least possible integer such that $L \times \mathrm{Mag}(X) > 1$ and for the convenience of computation, $L = 2^l$ is chosen, where l is the number of "0"s in front of the first non-zero fractional bit of $\mathrm{Mag}(X)$. Using the example on page 97, suppose the modulus $M = 32767$ and the pre-computed $\mathrm{Mag}(M) = \frac{2^h M}{D} = 0.0021367 = (0.00000000100011)_2$. $\mathrm{Mag}(X) = 0.0326 = (0.00001000010110)_2$ gives $l = 5$ and $L = 2^5$ as $2^5 \times (0.00001000010110)_2 = (1.000010110)_2 > 1$ and $2^5 \times (0.00000000100011)_2 = (0.000100011)_2 < 1$. 5 is selected instead of 7 or 8 is because the smaller the $L \times \mathrm{Mag}(M)$, the larger the $\frac{1}{L \times \mathrm{Mag}(M)}$ and the less error $\left\lfloor \frac{1}{L \times \mathrm{Mag}(M)} \right\rfloor$ incurs. For example, $\left\lfloor \frac{\mathrm{Mag}(X)}{\mathrm{Mag}(M)} \right\rfloor = \left\lfloor \frac{0.0326}{0.0021367} \right\rfloor = \left\lfloor \frac{(0.00001000010110)_2}{(0.00000000100011)_2} \right\rfloor = 15$. $\lfloor 2^5 \times (0.00001000010110)_2 \rfloor \times \left\lfloor \frac{1}{2^5 \times (0.00000000100011)_2} \right\rfloor = 1 \times 14 = 14$ and $\lfloor 2^8 \times (0.00001000010110)_2 \rfloor \times \left\lfloor \frac{1}{2^8 \times (0.00000000100011)_2} \right\rfloor = 8 \times 1 = 8$.

During the pre-computation, the possible values of L are not known when $\mathrm{Mag}(M) = (0.00000000100011)_2$ is calculated. Therefore, $\left\lfloor \frac{1}{2^l \times \mathrm{Mag}(M)} \right\rfloor$ is pre-computed for $l = 1, 2, \ldots, 8$ and the correct value will be selected once l is found during the actual computation process.

The required pre-computations are:

- $\langle D_i^{-1} \rangle_{m_i}$ for each RNS channel.
- w_i for each RNS channel.
- $C(D) = 2^h$.
- $C(\sigma_i) = \frac{2^h D_i^{-1} - w_i}{m_i}$ in binary.
- $-\left\lfloor \frac{1}{\mathrm{Mag}(M)} \right\rfloor \times M$ in the RNS.
- $-\left\lfloor \frac{1}{2^l \times \mathrm{Mag}(M)} \right\rfloor \times M$ for $l = 1 \ldots h$ in the RNS.

Core Computation

Equation (4.17) becomes $C(X) = \langle \sum_i x_i C(\sigma_i) \rangle_{2^h}$ after setting $C(D) = 2^h$. Then $C(X) = \langle \sum_i \langle x_i C(\sigma_i) \rangle_{2^h} \rangle_{2^h}$ is used to compute $C(X)$. The inner multiplication $\langle x_i C(\sigma_i) \rangle_{2^h}$ is actually a sum of w h-bit numbers as x_i is bounded by the channel width w bits. This is because only the least significant h bits of each partial product generated during this multiplication need to be kept, as shown within the dotted rectangle in Figure 4.10. These trimmed partial products can be easily obtained by discarding the top j bits of $C(\sigma_i)$ and adding j "0"s to its end if the j-th bit of x_i is 1. For example, if $h = 8$, $C(\sigma_i) = 202 = (11001010)_2$ and $x_i = 18 = (10010)_2$, then $\langle x_i C(\sigma_i) \rangle_{2^h} = (10010100)_2 + (10100000)_2$ because x_i has "1"s at the 1st and 4th bits respectively.

These w h-bit partial products can be summed in $O(\log w)$ complexity using an h-bit tree [Wallace64, Dadda65]. Such a basic tree [Weinberger81, Santoro89, Ohkubo95, Itoh01] is normally composed of a few h-bit Carry Save Adders (as described in Section 3.2 on page 37) followed by a h-bit Carry Propagate Adder [Weinberger58]. Note that it is only necessary to keep the h least significant bits while dropping any carry-outs above the $h-1$-th bit because of the reduction modulo 2^h to follow. Compared with array adders [Baugh73, Hatamian86, Sutherland99, Weste04], these trees achieve higher speed at the price of irregular layout and some long interconnect wires during implementation. Various hybrid structures have also been proposed that offer tradeoffs between these two extremes [Zuras86, Hennessy90, Mou90, Dhanesha95, Weste04]. They can achieve nearly as few levels of logic as the Wallace tree while offering more regular wiring.

The rest of the work is to perform the consecutive modular addition $\left\langle \sum_{i=1}^{N} \bullet \right\rangle_{2^h}$. Once again the addition tree method can be used to arrive at

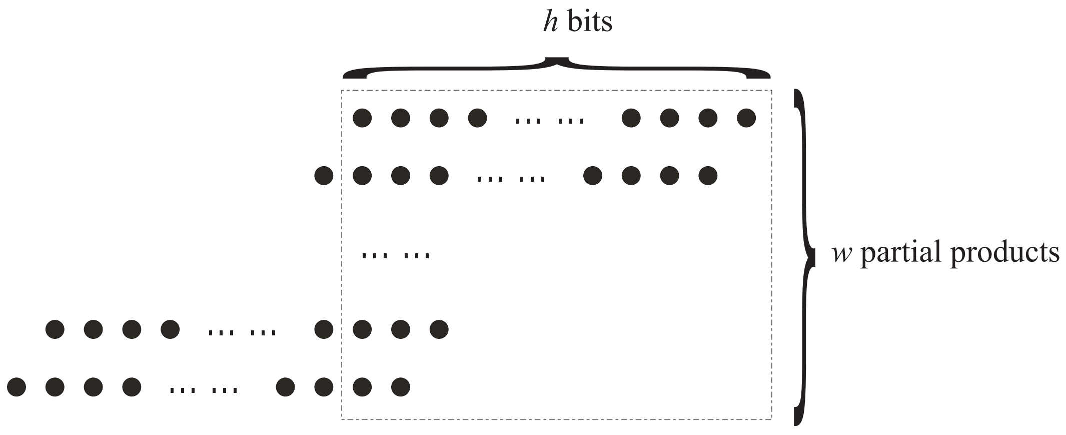


Figure 4.10: The h Least Significant Bits to be Maintained during the Modular Multiplication $\langle x_i C(\sigma_i)\rangle_{2^h}$

a low complexity of $O(\log N)$, where N is the number of RNS channels.

Noise Computation

$\text{Noise}(X) = \sum_i \frac{x_i w_i}{m_i}$ is to be computed at the same time as the core computation. This seems to be hard as it involves division, which is much harder than the multiplication within a RNS channel. However, note that $\frac{w_i}{m_i}$ can be pre-computed, and so this is actually a RNS channel multiplication with errors incurred by the inaccuracy of $\frac{w_i}{m_i}$. The following paragraphs describe a simple way to do this multiplication by examining the precision of the noise.

Let $f_i = \frac{x_i w_i}{m_i}$, then $\text{Noise}(X) = \sum_{i=1}^{N} f_i$. Each f_i is truncated or rounded up to maintain the precision needed, as it is not an integer. Here rounding f_i towards $-\infty$ is chosen, for example, $2.568 \rightarrow 2.5$, $27.66 \rightarrow 27.6$, $-1.210 \rightarrow -1.3$, etc. Define $f_i = \text{main}(f_i) + \text{trivial}(f_i)$, where $\text{main}(f_i)$ is the rounded result, i.e. 2.5, 27.6 and -1.3 respectively in the 3 cases above; $\text{trivial}(f_i)$ is the part of value discarded from f_i, i.e. 0.068, 0.06 and 0.090 in the cases. Note that $\text{trivial}(f_i)$ is always positive. Then, $\text{Noise}(X) = \sum_i f_i = \sum_i(\text{main}(f_i) +$

trivial(f_i)) $= \sum_i$ main(f_i) $+ \sum_i$ trivial(f_i). Thus, Noise(X) $- \sum_i$ main(f_i) $=$ $\sum_i$ trivial(f_i) > 0, is the error of main(f_i) as an estimate to the noise.

If $\sum_i$ trivial(f_i) < 1, then $0 <$ Noise(X) $- \sum_i$ main(f_i) < 1. Hence, $\lfloor$Noise(X)$\rfloor = \lfloor\sum_i$ main(f_i)$\rfloor$ or $\lfloor\sum_i$ main(f_i)$\rfloor$+1. This implies the $\lfloor$Mag(X)$\rfloor$ needed in (4.19) has an error of at most 1 because from (4.11), $\lfloor$Mag(X)$\rfloor =$ $\lfloor C(X)$+Noise(X)$\rfloor$ and $C(X)$ is an integer. One way to achieve $\sum_{i=1}^{N}$ trivial(f_i) < 1 is to ensure each trivial(f_i) is less than $1/N$, where N is the number of RNS modulus channels. In the example on page 97, $N = 7$ and $1/N = 0.142857$. Thus keeping $\lceil \log_{10} N\rceil = 1$ fractional digit in main(f_i) will make trivial(f_i) $< 0.1 < 1/7$, and hence $\lfloor$Noise(X)$\rfloor = \lfloor\sum_i$ main(f_i)$\rfloor$. Similarly in binary, $\lceil \log_2 N\rceil = 3$ fractional bits in main(f_i) will make trivial(f_i) $<$ $(0.001)_2 < (0.001001\ldots)_2 = 1/7$.

In fact, to keep $x_i \times \frac{w_i}{m_i}$ a fast short word length multiplication, only $w + \lceil \log_2 N\rceil + 1$ bits of the pre-computed $\frac{w_i}{m_i}$ are stored. This is because x_i is w bits long and can increase the inaccuracy by at most w times. Now the estimate to $\frac{w_i}{m_i}$ is only $\lceil \log_2 N\rceil + 1$ bits more than the channel width w and $x_i \times \frac{w_i}{m_i}$ is only a short multiplication. Therefore, the noise computation, Noise(X) $= \sum_i \frac{x_i w_i}{m_i}$, is expected to have only a similar delay to a RNS channel modular multiplication.

Again take the RNS with 7 channels of 5-bit width as in the example on page 97. $5 + \lceil \log_2 7\rceil + 1 = 9$ fractional bits of $\frac{w_i}{m_i}$ are to be kept. Hence 14-bit $\frac{x_i w_i}{m_i}$ and Noise(X) $= \sum_{i=1}^{7} \frac{x_i w_i}{m_i}$ add up to $14 + \lceil \log_2 7\rceil = 17$ bits with 8 integer bits all exact and 3 out of 9 fractional bits exact. This is why the $\frac{x_i w_i}{m_i}$ values are kept to 4 decimal fractional digits in the examples in Table 4.3 and Table 4.4 shown on page 107 and 108, respectively.

The Algorithm

The Classical Modular Reduction Algorithm in RNS using the Core Function is shown in Algorithm 4.1.

Algorithm 4.1 The Classical Modular Reduction Algorithm in RNS using the Core Function

```
Require: $M, N, h, \{m_1, \ldots, m_N\}$ as described above.
Require: pre-computed values $C(D) = 2^h$
Require: pre-computed tables $\langle D_i^{-1} \rangle_{m_i}$, $\frac{w_i}{m_i}$, $C(\sigma_i) = \frac{2^h D_i^{-1} - w_i}{m_i}$, $\left\langle - \left\lfloor \frac{1}{\mathrm{Mag}(M)} \right\rfloor \times M \right\rangle_{m_i}$ for $i = 1 \ldots N$ and $\left\langle - \left\lfloor \frac{1}{2^l \times \mathrm{Mag}(M)} \right\rfloor \times M \right\rangle_{m_i}$ for $i = 1 \ldots N$ and $l = 1 \ldots h$
Ensure: $Z \equiv A \times B \bmod M$
 1: $X = A \times B$ {1 RNS multiplication in $N$ channels}
 2: $C(X) = \left\langle \sum_{i=1}^{N} \langle x_i C(\sigma_i) \rangle_{2^h} \right\rangle_{2^h}$ | $\mathrm{Noise}(X) = \sum_{i=1}^{N} \frac{x_i w_i}{m_i}$ {Two parallel computations with Core using fast tree structure and Noise equivalent to 1 RNS channel multiplication}
 3: $\lfloor \mathrm{Mag}(X) \rfloor = \lfloor C(X) + \mathrm{Noise}(X) \rfloor$
 4: if $\lfloor \mathrm{Mag}(X) \rfloor \geq 1$ then
 5:    $\langle \lfloor \mathrm{Mag}(X) \rfloor \rangle_{m_i}$ for $i = 1 \ldots N$ {Convert $\lfloor \mathrm{Mag}(X) \rfloor$ to its residue representation}
 6:    $Z = X + \lfloor \mathrm{Mag}(X) \rfloor \times (- \left\lfloor \frac{1}{\mathrm{Mag}(M)} \right\rfloor \times M)$ {1 RNS multiplication and 1 RNS addition in $N$ channels}
 7: else
 8:    Choose $l$ so that $\lfloor 2^l \times \mathrm{Mag}(X) \rfloor = 1$
 9:    $Z = X + (- \left\lfloor \frac{1}{2^l \times \mathrm{Mag}(M)} \right\rfloor \times M)$ {1 RNS multiplication and 1 RNS addition in $N$ channels}
10: end if
```

4.2.3 Examples

To provide some examples of the new algorithm, consider once again the RNS modulus set $\{17, 19, 23, 25, 27, 29, 31\}$ with dynamic range $D = 4508102925$, $M = 37627$, $\langle D_i^{-1} \rangle_{m_i} = \{1, 15, 13, 3, 8, 4, 1\}$, $C(D) = 2^8 = 256$, and the weight set $w_i = \{1, 2, -7, -7, -4, 9, 8\}$. Table 4.2 lists the pre-computed parameters. To demonstrate the process for $\lfloor \text{Mag}(X) \rfloor \geq 1$, take $A = 65537$ and $B = 65535$. The calculation of $Z = A \times B \mod M = 65537 \times 65535 \mod 37627 = 33380$ using Algorithm 4.1 is listed in Table 4.3. This gives $Z = \{2, 18, 0, 22, 21, 8, 22\} = 15874347 \equiv 33380 \mod 37627$. For the case of $\lfloor \text{Mag}(X) \rfloor \leq 1$, suppose $A = 167$ and $B = 3463$, then $Z = 167 \times 3463 \mod 37627 = 13916$ is listed in Table 4.4. This gives $Z = \{16, 15, 0, 18, 0, 10, 21\} = 51543 \equiv 13916 \mod 37627$.

Table 4.2: An Example of Pre-Computed Parameters for the Classical Modular Reduction Algorithm in RNS using the Core Function

N		7						
D		4508102925 (33 bits)						
h		8						
$C(D)$		$2^8 = 256$						
M		37627 (16 bits)						
$\text{Mag}(M)$		$0.002136710754 = (0.00000000100011)_2$						
i		1	2	3	4	5	6	7
m_i		17	19	23	25	27	29	31
$\langle D_i^{-1} \rangle_{m_i}$		1	15	13	3	8	4	1
w_i		1	2	-7	-7	-4	9	8
$C(\sigma_i) = \frac{2^h D_i^{-1} - w_i}{m_i}$		15	202	145	31	76	35	8
$-\left\lfloor \frac{1}{\text{Mag}(M)} \right\rfloor \times M$		14	11	8	14	18	2	21
$-\left\lfloor \frac{1}{2^l \times \text{Mag}(M)} \right\rfloor \times M$	$l = 1$	7	15	4	7	9	1	26
	$l = 2$	12	17	2	16	18	15	13
	$l = 3$	9	12	12	9	17	0	3
	$l = 4$	13	6	6	17	22	0	17
	$l = 5$	1	16	14	22	19	7	5
	$l = 6$	9	8	7	11	23	18	18
	$l = 7$	16	17	3	19	6	16	21
	$l = 8$	11	12	1	23	11	15	7

Table 4.3: An Example of the Classical Modular Reduction Algorithm in RNS using the Core Function for a case in which $\lfloor \mathrm{Mag}(X) \rfloor \geq 1$

i	1	2	3	4	5	6	7
A	2	6	10	12	8	26	3
B	0	4	8	10	6	24	1
$X = A \times B$	0	5	11	20	21	15	3
$\langle x_i C(\sigma_i) \rangle_{2^h}$	0	242	59	108	60	13	24
$C(X) = \langle \sum_i x_i C(\sigma_i) \rangle_{2^h}$	250						
$\frac{x_i w_i}{m_i}$	0.0000	0.5263	-3.3479	-5.6000	-3.1112	4.6551	0.7741
$\mathrm{Noise}(X) = \sum_i \frac{x_i w_i}{m_i}$	-6.1036						
$\mathrm{Mag}(X) = C(X) + \mathrm{Noise}(X)$	$243.8964 > 1$						
$\lfloor \mathrm{Mag}(X) \rfloor$	243						
$Z = X + \lfloor \mathrm{Mag}(X) \rfloor \times (- \left\lfloor \frac{1}{\mathrm{Mag}(M)} \right\rfloor \times M)$	2	18	0	22	21	8	22

Table 4.4: An Example of the Classical Modular Reduction Algorithm in RNS using the Core Function for a case in which $\lfloor \mathrm{Mag}(X) \rfloor < 1$

i	1	2	3	4	5	6	7
A	14	15	6	17	5	22	12
B	12	5	13	13	7	12	22
$X = A \times B$	15	18	9	21	8	3	16
$\langle x_i C(\sigma_i) \rangle_{2^h}$	225	52	25	139	96	105	128
$C(X) = \langle \sum_i x_i C(\sigma_i) \rangle_{2^h}$	2						
$\frac{x_i w_i}{m_i}$	0.8823	1.8947	-2.7392	-5.8800	-1.1852	0.9310	4.1290
$\mathrm{Noise}(X) = \sum_i \frac{x_i w_i}{m_i}$	-1.9674						
$\mathrm{Mag}(X) = C(X) + \mathrm{Noise}(X)$	$0.0326 = (0.00001000010110)_2 < 1$						
Choose l	5						
$L = 2^l$	2^5						
Choose $- \left\lfloor \frac{1}{2^l \times \mathrm{Mag}(M)} \right\rfloor \times M$	1	16	14	22	19	7	5
$Z = X + (- \left\lfloor \frac{1}{2^l \times \mathrm{Mag}(M)} \right\rfloor \times M)$	16	15	0	18	0	10	21

4.3 RNS Sum of Residues Modular Multiplication

Recall the Sum of Residues modular reduction algorithm in Section 3.2. X is reduced modulo M by finding a sum of residues modulo M [Findlay90, Tomlinson89]. If $X = \sum_i x_i$ then $\sum_i \langle x_i \rangle_M \equiv X \mod M$ is obtained. Although this does not produce a fully reduced result, it is possible to determine bounds for intermediate values such that the output from one modular multiplication can be used as the input to subsequent modular multiplications without overflow. This section uses this sum of residues method for modular multiplication in the RNS, with the advantage that all of the residues $\langle x_i \rangle_M$ can be evaluated in parallel.

4.3.1 Sum of Residues Reduction in the RNS

To derive a RNS algorithm for sum of residues reduction, we start again with the Chinese Remainder Theorem (CRT). Using the CRT, an integer X can be expressed as

$$X = \left\langle \sum_i D_i \langle D_i^{-1} x_i \rangle_{m_i} \right\rangle_D , \tag{2.1}$$

where D, D_i and $\langle D_i^{-1} \rangle_{m_i}$ are pre-computed constants. Defining $\gamma_i = \langle D_i^{-1} x_i \rangle_{m_i}$ in (2.1) yields

$$\begin{aligned} X &= \left\langle \sum_{i=1}^{N} \gamma_i D_i \right\rangle_D \\ &= \sum_{i=1}^{N} \gamma_i D_i - \alpha D. \end{aligned} \tag{4.21}$$

Actually Equation (4.21) has already appeared in a different style as Equation (4.15) in Section 4.2.1 on page 96. The difference is it was used then for core function to form a classical RNS algorithm while here it gives rise to a RNS modular reduction algorithm using a sum of residues. Reducing (4.21) modulo M yields

$$\begin{aligned} Z &= \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M \qquad (4.22) \\ &= \sum_{i=1}^{N} Z_i - \langle \alpha D \rangle_M \\ &\equiv X \mod M \end{aligned}$$

for $Z_i = \gamma_i \langle D_i \rangle_M$. Thus X can be reduced modulo M using a sum of the residues Z_i and a correction factor $\langle \alpha D \rangle_M$.

Note that $\gamma_i = \langle D_i^{-1} x_i \rangle_{m_i}$ can be found using a single RNS multiplication in channel i as $\langle D_i^{-1} \rangle_{m_i}$ is just a pre-computed constant. All N γ_is can be produced simultaneously in their respective channels in the time of a single RNS multiplication. Similarly, only one RNS multiplication is needed for $Z_i = \gamma_i \langle D_i \rangle_M$ as $\langle \langle D_i \rangle_M \rangle_{m_i}$ can be pre-computed.

4.3.2 Approximation of α

Now α becomes the only value yet to be found. In this section, let us deduce the solution provided by [Kawamura00] in a different perspective and try to improve it by decomposing its approximations. This is the first improvement over [Kawamura00] and gives more accuracy by permitting exact γ_i.

Dividing both sides of (4.21) by D yields

$$\alpha + \frac{X}{D} = \frac{\sum_{i=1}^{N} \gamma_i D_i}{D} = \sum_{i=1}^{N} \frac{\gamma_i}{m_i}. \qquad (4.23)$$

Since $0 \leq X/D < 1$, $\alpha \leq \sum_{i=1}^{N} \frac{\gamma_i}{m_i} < \alpha + 1$ holds. Therefore,

$$\alpha = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{m_i} \right\rfloor. \qquad (4.24)$$

In subsequent discussions, $\hat{\alpha}$ is used to approximate α. Firstly, an approximation of $\hat{\alpha} = \alpha$ or $\alpha - 1$ will be given. Secondly, some extra work will exactly assure $\hat{\alpha} = \alpha$ under certain prerequisites.

Deduction of $\hat{\alpha} = \alpha$ or $\alpha - 1$

The first approximation is introduced here: a denominator m_i in (4.24) is replaced by 2^w, where w is the RNS channel width and $2^{w-1} < m_i \leq 2^w$. Then the estimate of (4.24) becomes

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} \right\rfloor . \tag{4.25}$$

The error incurred by this denominator's approximation is denoted as

$$\epsilon_i = \frac{(2^w - m_i)}{2^w}.$$

Then,

$$2^w = \frac{m_i}{1 - \epsilon_i}.$$

Recall that according to the definition of a RNS in Section 2.1.1, the RNS moduli are ordered such that $m_i < m_j$ for all $i < j$. Therefore, the largest error

$$\epsilon = \max(\epsilon_i) = \frac{(2^w - m_1)}{2^w}.$$

Let us now investigate the accuracy of $\hat{\alpha}$:

$$\begin{aligned} 0 &\leq \gamma_i \leq m_i - 1 \\ \Rightarrow 0 &\leq \frac{\gamma_i}{m_i} \leq \frac{m_i - 1}{m_i} < 1 \\ \Rightarrow 0 &\leq \sum_{i=1}^{N} \frac{\gamma_i}{m_i} < N. \end{aligned} \tag{4.26}$$

Therefore,

$$\begin{aligned}\sum_{i=1}^{N}\frac{\gamma_i}{2^w} &= \sum_{i=1}^{N}\frac{\gamma_i(1-\epsilon_i)}{m_i} && (4.27)\\ &\geq \sum_{i=1}^{N}\frac{\gamma_i(1-\epsilon)}{m_i}\\ &= (1-\epsilon)\sum_{i=1}^{N}\frac{\gamma_i}{m_i}\\ &= \sum_{i=1}^{N}\frac{\gamma_i}{m_i}-\epsilon\sum_{i=1}^{N}\frac{\gamma_i}{m_i}\\ \Rightarrow \sum_{i=1}^{N}\frac{\gamma_i}{2^w} &> \sum_{i=1}^{N}\frac{\gamma_i}{m_i}-N\epsilon. && (4.28)\end{aligned}$$

The last inequality holds due to Equation (4.26). If $0 \leq N\epsilon \leq 1$, then $\sum_{i=1}^{N}\frac{\gamma_i}{m_i} - N\epsilon > \sum_{i=1}^{N}\frac{\gamma_i}{m_i} - 1$. Thus, $\sum_{i=1}^{N}\frac{\gamma_i}{2^w} > \sum_{i=1}^{N}\frac{\gamma_i}{m_i} - 1$. In addition, obviously $\sum_{i=1}^{N}\frac{\gamma_i}{2^w} < \sum_{i=1}^{N}\frac{\gamma_i}{m_i}$. Therefore,

$$\sum_{i=1}^{N}\frac{\gamma_i}{m_i}-1<\sum_{i=1}^{N}\frac{\gamma_i}{2^w}<\sum_{i=1}^{N}\frac{\gamma_i}{m_i}. \tag{4.29}$$

Then,

$$\hat{\alpha}=\left\lfloor\sum_{i=1}^{N}\frac{\gamma_i}{2^w}\right\rfloor=\left\lfloor\sum_{i=1}^{N}\frac{\gamma_i}{m_i}\right\rfloor=\alpha,$$

or,

$$\hat{\alpha}=\left\lfloor\sum_{i=1}^{N}\frac{\gamma_i}{m_i}\right\rfloor-1=\alpha-1.$$

when $0 \leq N\epsilon \leq 1$.

This raises the question: is it easy to satisfy the condition $0 \leq N\epsilon \leq 1$ in a RNS? The answer is: the larger the dynamic range of the RNS, the easier. This is contrary to most published techniques that are only applicable to RNS with small dynamic range as surveyed in Section 4.1.

Given $0 \leq N\epsilon \leq 1$ and $\epsilon = \frac{(2^w-m_1)}{2^w}$,

$$\frac{N-1}{N}\leq\frac{m_1}{2^w}\leq 1,$$

Table 4.5: The Maximum Possible N against w in RNS Sum of Residues Reduction

w (bits)	2–4	5	6	7	8	9	10
Maximum N	2	3	3	5	7	9	12
D (bits)	4–8	15	18	35	56	81	120
w (bits)	11	12	13	14	15	16	17
Maximum N	17	26	52	80	123	211	376
D (bits)	187	312	676	1120	1845	3376	6392
w (bits)	18	19	20	21	22	23	24
Maximum N	524	732	1044	1526	2307	3574	5256
D (bits)	9432	13908	20880	32046	50754	82202	126144

which means there must be at least N co-prime numbers existing within the interval $I = [\frac{N-1}{N}2^w, 2^w]$ for the use of RNS moduli. Take $w = 32$ and $N = 69$ as an example. Then the dynamic range $D > 31 \times 69 = 2139$ bits > 2048 bits, which gives sufficient word length for a 1024-bit modulus M. $I = [\frac{68}{69}2^{32}, 2^{32}]$ has a rough span of 10^7, and there are 2807426 primes found within it, which is much larger than the required number, $N = 69$. Another example could be a RNS with 10 8-bit moduli. $I = [\frac{9}{10}2^8, 2^8] = [231, 256]$, which spans 26 integers with only 4 primes among them, much less than $N = 10$.

Table 4.5 lists the maximum N against different w from 2 to 24, which witnesses the fact that the number of channels N available increases drastically along with the linear increase of the channel width w. This is because the span of interval I is $2^w - \frac{N-1}{N}2^w = \frac{2^w}{N}$. 2^w increases much faster than N, which gives a sharp increase of the span of I with more primes existing within it as the dynamic range D of the RNS increases.

Apart from this, it is also easy to satisfy the harsher condition $0 \leq N\epsilon \leq$

$\frac{1}{2}$. This requires

$$\frac{2N-1}{2N} \leq \frac{m_1}{2^w} \leq 1,$$

which can be derived using the process above. This will be used for further developments in next subsection.

The actual problem now is $\hat{\alpha}$ could be α or $\alpha - 1$. From Equation (4.21), $\hat{X}$ could be X or $X + D$. Then two values of $X \mod M$ will result and it is difficult to tell the correct one. Thus, $\hat{\alpha}$ needs to be the exact α.

Ensuring $\hat{\alpha} = \alpha$

To make sure $\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} \right\rfloor$ in (4.25) is equal to α instead of $\alpha - 1$, a correction factor Δ can be added to the floor function. Equation (4.25) becomes

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta \right\rfloor. \tag{4.30}$$

Substituting Equation (4.23) into Equation (4.28) and Equation (4.29) yields

$$\alpha + \frac{X}{D} - N\epsilon < \sum_{i=1}^{N} \frac{\gamma_i}{2^w} < \alpha + \frac{X}{D}.$$

Adding Δ on both sides yields

$$\alpha + \frac{X}{D} - N\epsilon + \Delta < \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta < \alpha + \frac{X}{D} + \Delta. \tag{4.31}$$

If $\Delta \geq N\epsilon$, then $\Delta - N\epsilon \geq 0$ and $\alpha + \frac{X}{D} - N\epsilon + \Delta \geq \alpha$. If $0 \leq X < (1-\Delta)D$, then $\frac{X}{D} + \Delta < 1$ and $\alpha + \frac{X}{D} + \Delta < \alpha + 1$. Hence,

$$\alpha < \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta < \alpha + 1. \tag{4.32}$$

Therefore,

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{2^w} + \Delta \right\rfloor = \alpha$$

holds. The two prerequisites from the deduction above are

$$\begin{cases} N\epsilon \leq \Delta < 1 \\ 0 \leq X < (1-\Delta)D \end{cases} \tag{4.33}$$

It has already been shown in the previous section that the first condition $N\epsilon < \Delta < 1$ is easily satisfied as long as Δ is not too small. For example, Δ could be $\frac{1}{2}$. The second one is not that feasible at first sight as it requires X be less than half the dynamic range D in the case of $\Delta = \frac{1}{2}$. However, $\frac{1}{2}D$ is just one bit shorter than D, which is a number over two thousand bits. Therefore, this can be easily achieved by extending D by several bits to cover the upper bound of X. This is deduced in the following section. Hence, we have obtained an $\hat{\alpha} = \alpha$.

4.3.3 Bound Deduction

The RNS dynamic range to do a 1024-bit multiplication should at least be 2048 bits. However as stated in Section 2.1.4, algorithms always require some redundant RNS channels. This section is dedicated to confirming how many channels are actually needed for the new RNS Sum of Residues algorithm. Equation (4.22), the basis of the RNS Sum of Residues algorithm, is rewritten here:

$$Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M. \tag{4.22}$$

Note that the result Z may be greater than the modulus M and would require subtraction of a multiple of M to be fully reduced. Instead, the dynamic range D of the RNS can be made large enough that the results of modular multiplications can be used as operands for subsequent modular multiplications without overflow.

Given that $\gamma_i < m_i < 2^w$, $\langle D_i \rangle_M < M$ and $\langle \alpha D \rangle_M \geq 0$,

$$Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M < N 2^w M. \tag{4.34}$$

Thus, take operands $A < N2^wM$ and $B < N2^wM$ such that $X = A \times B < N^2 2^{2w} M^2$.

According to Equation (4.33), we must ensure that X does not overflow $(1-\Delta)D$. If it is assumed M can be represented in h channels so that $M < 2^{wh}$, then

$$X < N^2 2^{2wh+2w}.$$

$X < (1-\Delta)D$ is required for

$$D > 2^{N(w-1)},$$

which will be satisfied if

$$N^2 2^{2wh+2w} < (1-\Delta) 2^{N(w-1)}.$$

This is equivalent to

$$N > 2h + 2 + \frac{2\log_2 \frac{N}{1-\Delta} + N}{w}.$$

For example, for $w \geq 32$, $N < 128$ and $\Delta = \frac{1}{2}$, it will be sufficient to choose $N \geq 2h + 7$. Note that this bound is conservative and fewer channels may be sufficient for a particular RNS, as in the example on page 119. This is because the bound of Z can be directly computed as

$$Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M \leq \sum_{i=1}^{N} (m_i - 1) \langle D_i \rangle_M.$$

using the pre-computed RNS constants, m_i and $\langle D_i \rangle_M$, instead of worst case bounds N and M as in (4.34).

4.3.4 The RNS Sum of Residues Modular Multiplication Algorithm

Another Approximation

Equation (4.24) giving the exact α is rewritten here:

$$\alpha = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{m_i} \right\rfloor . \tag{4.24}$$

2^w has been used to approximate the denominator m_i to form Equation (4.25) and Equation (4.30). Note that a numerator γ_i can also be simplified by being represented using its most significant q bits, where $q < w$. Hence,

$$\hat{\gamma_i} = 2^{w-q} \left\lfloor \frac{\gamma_i}{2^{w-q}} \right\rfloor . \tag{4.35}$$

The error incurred by this numerator's approximation is denoted as

$$\delta_i = \frac{\gamma_i - \hat{\gamma_i}}{m_i}.$$

Then

$$\hat{\gamma_i} = \gamma_i - \delta_i m_i.$$

The largest possible error will be

$$\delta = \frac{2^{w-q} - 1}{m_1}.$$

Note that this approximation, treated as a necessary part of the computation of α in [Kawamura00], is actually not imperative. It has been shown the algorithm works fine without this approximation in previous discussions although it does simplify the computations in hardware.

Replacing the γ_i in Equation (4.30) by $\hat{\gamma_i}$ yields

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\hat{\gamma_i}}{2^w} + \Delta \right\rfloor . \tag{4.36}$$

Then, Equation (4.27) becomes

$$\begin{aligned}
\sum_{i=1}^{N} \frac{\hat{\gamma}_i}{2^w} &= \sum_{i=1}^{N} \frac{(\gamma_i - \delta_i m_i)(1-\epsilon_i)}{m_i} \\
&= \sum_{i=1}^{N} \frac{\gamma_i(1-\epsilon_i)}{m_i} - \sum_{i=1}^{N}(1-\epsilon_i)\delta_i \\
&\geq (1-\epsilon)\sum_{i=1}^{N} \frac{\gamma_i}{m_i} - N\delta \\
\sum_{i=1}^{N} \frac{\hat{\gamma}_i}{2^w} &> \sum_{i=1}^{N} \frac{\gamma_i}{m_i} - N(\epsilon+\delta). \qquad (4.37)
\end{aligned}$$

This is because

$$\begin{aligned}
0 &< 1-\epsilon_i = \frac{m_i}{2^w} < 1 \\
\Rightarrow 0 &< \sum_{i=1}^{N}(1-\epsilon_i) < N,
\end{aligned}$$

and Equation (4.26)

$$0 \leq \sum_{i=1}^{N} \frac{\gamma_i}{m_i} < N.$$

Note that the only difference between Equation (4.28) and (4.37) is that the ϵ in the former is replaced by the $\epsilon + \delta$ in the latter. Following a similar development to Section 4.3.2, Equation (4.31) becomes

$$\alpha + \frac{X}{D} - N(\epsilon+\delta) + \Delta < \sum_{i=1}^{N} \frac{\hat{\gamma}_i}{2^w} + \Delta < \alpha + \frac{X}{D} + \Delta. \qquad (4.38)$$

The two prerequisites in (4.33) are now

$$\begin{cases} N(\epsilon+\delta) \leq \Delta < 1 \\ 0 \leq X < (1-\Delta)D \end{cases} \qquad (4.39)$$

This will again guarantee

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\hat{\gamma}_i}{2^w} + \Delta \right\rfloor = \alpha.$$

Substituting (4.35) into Equation (4.36) yields

$$\hat{\alpha} = \left\lfloor \sum_{i=1}^{N} \frac{\left\lfloor \frac{\gamma_i}{2^{w-q}} \right\rfloor}{2^q} + \Delta \right\rfloor. \tag{4.40}$$

This is the final equation used in the new algorithm to estimate α.

An example

To demonstrate the applicability of this scheme to, for example, 1024-bit RSA cryptography, consider the set of consecutive 32-bit prime moduli $[m_1, m_2, \ldots, m_{69}] = [4294965131, 4294965137, \ldots, 4294966427]$. For $h = 33$, $\left\lfloor \log_2 \left(\prod_{i=1}^{h} m_i \right) \right\rfloor = 1055$ and the RNS has over 1024 bits of dynamic range for the modulus M. Choosing $N = 69$ gives $\lfloor \log_2 D \rfloor = 2207$ bits of dynamic range. The maximum value of the intermediate product X is $\left(\sum_{i=1}^{N} (m_i - 1) \langle D_i \rangle_M \right)^2$ such that $\lfloor \log_2 X \rfloor \leq 2188$ and $X < D$ as required.

Selecting $q = 7$ and $\Delta = 0.75$ ensures $0 \leq N(\epsilon + \delta) \leq \Delta < 1$ and $0 \leq X < (1 - \Delta)D$ as required for exact determination of α.

The Algorithm

The Sum of Residues modular multiplication algorithm in RNS is shown in Algorithm 4.2. It computes $Z \equiv A \times B \mod M$ using Equation (4.22).

$$Z = \sum_{i=1}^{N} \gamma_i \langle D_i \rangle_M - \langle \alpha D \rangle_M.$$

Note that from Equation (4.24) and (4.26), $\alpha < N$. Thus, $\langle \alpha D \rangle_M$ can be pre-computed in RNS for $\alpha = 0 \ldots N - 1$.

An implementation of this algorithm is described in Chapter 5.

Algorithm 4.2 The Sum of Residues Modular Multiplication Algorithm in RNS

```
Require: $M, N, w, \Delta, q, \{m_1, \dots, m_N\}, (N2^wM)^2 < (1-\Delta)D, N(\frac{(2^w-m_1)}{2^w} + \frac{2^{w-q}-1}{m_1}) \leq \Delta < 1$.
Require: pre-computed table $\langle D_i^{-1} \rangle_{m_i}$ for $i = 1, \dots, N$
Require: pre-computed table $\langle \langle D_i \rangle_M \rangle_{m_j}$ for $i, j = 1, \dots, N$
Require: pre-computed table $\langle \langle \alpha D \rangle_N \rangle_{m_i}$ for $\alpha = 1, \dots, N-1$ and $i = 1, \dots, N-1$
Require: $A < N2^wM, B < N2^wM$
Ensure: $Z \equiv A \times B \bmod M$
 1: $X = A \times B$
 2: $\gamma_i = \langle x_i D_i^{-1} \rangle_{m_i}$ for $i = 1, \dots, N$
 3: $Z_i = \gamma_i \times \langle D_i \rangle_M$ for $i = 1, \dots, N$
 4: $Z = \sum_{i=1}^{N} Z_i$
 5: $\alpha = \left\lfloor \sum_{i=1}^{N} \left\lfloor \frac{\gamma_i}{2^{w-q}} \right\rfloor / 2^q + \Delta \right\rfloor$
 6: $Z = Z - \langle \alpha D \rangle_M$
```

4.4 RNS Barrett Modular Multiplication

This section presents a RNS modular multiplication algorithm in which the reduction modulo a long word length M is performed by multiplying by the inverse of the modulus.

Section 3.1.3 proposes Equation (3.5)

$$X \mod M = X - \left\lfloor \frac{X}{M} \right\rfloor \times M \tag{3.5}$$

for modular multiplication in binary. The scaling step $Y = \lfloor \frac{X}{M} \rfloor$ is achieved by multiplying X by M^{-1} using Barrett algorithm. The following sections use this to develop a RNS reduction algorithm in which all of intermediate operations occur within the RNS.

4.4.1 RNS Modular Reduction using Barrett Algorithm

Since multiplication and subtraction are both trivial in RNS, the scaling $\lfloor \frac{X}{M} \rfloor$ in (3.5) again becomes the key problem. An instance of the Improved Barrett algorithm in [Dhem98] is used, as described in Section 3.1.3:

$$Y = \left\lfloor \frac{X}{M} \right\rfloor = \left\lfloor \frac{\lfloor \frac{X}{2^{n+v}} \rfloor \lfloor \frac{2^{n+u}}{M} \rfloor}{2^{u-v}} \right\rfloor + \epsilon \tag{4.41}$$

where n is the word length of modulus M. Setting $u = n + 3$ and $v = -2$ bounds the error term ϵ within 1, as shown in Section 3.3.1. Then, (4.41) becomes

$$Y = \left\lfloor \frac{X}{M} \right\rfloor = \left\lfloor \frac{\lfloor \frac{X}{2^{n-2}} \rfloor \times K}{2^{n+5}} \right\rfloor. \tag{4.42}$$

where $K = \lfloor 2^{2n+3}/M \rfloor$ is a constant and can be computed and stored in advance.

Subsections below examine the following steps involved in RNS scaling using Barrett's algorithm in the following order:

- Scaling by a power of 2: $\left\lfloor \frac{X}{2^u} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{X}{2^{u-1}} \right\rfloor}{2} \right\rfloor$.
- Scaling by 2: $\left\lfloor \frac{X}{2} \right\rfloor = (X - \langle X \rangle_2) \times 2^{-1}$.
- Reduction modulo 2: $\langle X \rangle_2 = \langle \gamma_1 \rangle_2 \oplus \langle \gamma_2 \rangle_2 \oplus \ldots \langle \gamma_N \rangle_2 \oplus \langle \alpha \rangle_2$, where $\gamma_i = \langle D_i^{-1} \times x_i \rangle_{m_i}$.
- Calculation of $\langle \alpha \rangle_2$: $\langle \alpha \rangle_2 = \langle \langle \sum_{i=1}^{N} \theta_i - \phi \rangle_{m_r} \rangle_2$, where $\theta_i = \langle D^{-1} D_i \gamma_i \rangle_{m_r}$, $\phi = \langle D^{-1} x_r \rangle_{m_r}$ and m_r is a redundant RNS channel.

Scaling by a power of 2

To apply Barrett's equation (4.42), scaling by the binary powers 2^{n-2} and 2^{n+5} must be performed in RNS. Since $\left\lfloor \frac{\left\lfloor \frac{X}{2^{a-1}} \right\rfloor}{2} \right\rfloor = \left\lfloor \frac{X}{2^a} \right\rfloor$ [Richman71], X can be scaled by 2^a by iteratively being scaled by 2. Higher radix versions are also possible in which X is iteratively scaled by a power 2^l. Subsequent subsections assume scaling by 2 but the issue of high-radix versions will be revisited on page 127.

Scaling by 2

Scaling by 2 is achieved by using $\left\lfloor \frac{X}{2} \right\rfloor = (X - \langle X \rangle_2) \times 2^{-1}$. In the RNS channels this is performed as

$$y_i = \langle \left\lfloor \frac{X}{2} \right\rfloor \rangle_{m_i} = \langle (x_i - \langle X \rangle_2) \times \langle 2^{-1} \rangle_{m_i} \rangle_{m_i} \tag{4.43}$$

where $\langle 2^{-1} \rangle_{m_i}$ has been pre-computed.

Reduction modulo 2

The next problem is to find $\langle X\rangle_2$. To do this a redundant modulus m_r is introduced into the RNS. $m_r \geq N$ is chosen and m_r is co-prime with all of the members of the original modulus set $\{m_1, ..., m_N\}$. The original input X is therefore represented as $X = \{x_1, x_2, ..., x_N, x_r\}$. Note that a similar solution from [Shenoy89a] is discussed in Section 4.1.2; however in that case two redundant moduli are used instead of one.

From the Chinese Remainder Theorem,

$$\begin{aligned} X &= \langle \sum_{i=1}^{N} D_i\gamma_i\rangle_D \\ &= D_1\gamma_1 + D_2\gamma_2 + ... + D_N\gamma_N - \alpha D \end{aligned} \tag{4.44}$$

where $\gamma_i = \langle D_i^{-1}x_i\rangle_{m_i}$ and α is an unknown integer. Note that this equation is identical to Equation (4.21) which is then used to perform the reduction modulo M. However, in this section it will be used to perform a reduction modulo the redundant channel modulus m_r to obtain α. Reducing both sides of (4.44) modulo m_r yields

$$\begin{aligned} x_r &= \left\langle \sum_{i=1}^{N} \langle\langle D_i\rangle_{m_r}\langle\gamma_i\rangle_{m_r}\rangle_{m_r} - \langle\alpha D\rangle_{m_r} \right\rangle_{m_r} \\ \Rightarrow \langle\alpha\rangle_{m_r} &= \left\langle \left\langle \sum_{i=1}^{N} \langle\langle D_i\rangle_{m_r}\langle\gamma_i\rangle_{m_r}\rangle_{m_r} - x_r \right\rangle_{m_r} \times \langle D^{-1}\rangle_{m_r} \right\rangle_{m_r}. \end{aligned}$$

Since Equation (4.26) in Section 4.3.2 gives

$$0 \leq \sum_{i=1}^{N} \frac{\gamma_i}{m_i} < N, \tag{4.26}$$

it can then be shown that $0 \leq \alpha < N$ because

$$\alpha = \left\lfloor \sum_{i=1}^{N} \frac{\gamma_i}{m_i} \right\rfloor. \tag{4.24}$$

Now that $m_r \geq N$ and $\alpha < N$, $\alpha = \langle \alpha \rangle_{m_r}$. So

$$\begin{aligned}
\alpha &= \left\langle \sum_{i=1}^{N} \left\langle \langle D^{-1} \rangle_{m_r} \langle D_i \rangle_{m_r} \langle \gamma_i \rangle_{m_r} \right\rangle_{m_r} - \langle \langle D^{-1} \rangle_{m_r} x_r \rangle_{m_r} \right\rangle_{m_r} && (4.45) \\
\Rightarrow \langle \alpha \rangle_2 &= \langle \langle \sum_{i=1}^{N} \langle \langle D^{-1} \rangle_{m_r} \langle D_i \rangle_{m_r} \langle \gamma_i \rangle_{m_r} \rangle_{m_r} \\
&\quad - \langle \langle D^{-1} \rangle_{m_r} x_r \rangle_{m_r} \rangle_{m_r} \rangle_2. && (4.46)
\end{aligned}$$

where $\langle D_i \rangle_{m_r}$ and $\langle D^{-1} \rangle_{m_r}$ can both be pre-computed.

From (4.44):

$$\langle X \rangle_2 = \langle \langle D_1 \gamma_1 \rangle_2 + \langle D_2 \gamma_2 \rangle_2 + ... + \langle D_N \gamma_N \rangle_2 - \langle \alpha D \rangle_2 \rangle_2. \tag{4.47}$$

As $\langle a \rangle_2 \in \{0, 1\}$, $\langle ab \rangle_2 = \langle a \rangle_2 \langle b \rangle_2$ is valid for any integers a and b. Also, working modulo 2, both addition $'+'$ and subtraction $'-'$ operations can be performed using Boolean addition $'\oplus'$ (which is equivalent to a logic XOR). Hence Equation (4.47) can be re-written as

$$\langle X \rangle_2 = \langle D_1 \rangle_2 \langle \gamma_1 \rangle_2 \oplus \langle D_2 \rangle_2 \langle \gamma_2 \rangle_2 \oplus ... \oplus \langle D_N \rangle_2 \langle \gamma_N \rangle_2 \oplus \langle \alpha \rangle_2 \langle D \rangle_2.$$

If all of the RNS moduli m_i are primes other than 2, then D_i and D are all odd and hence

$$\langle X \rangle_2 = \langle \gamma_1 \rangle_2 \oplus \langle \gamma_2 \rangle_2 \oplus ... \oplus \langle \gamma_N \rangle_2 \oplus \langle \alpha \rangle_2. \tag{4.48}$$

As $\gamma_i = \langle D_i^{-1} x_i \rangle_{m_i}$ is only an efficient RNS channel modular multiplication, $\langle X \rangle_2$ can be computed from (4.48) given $\langle \alpha \rangle_2$ from (4.46). The computation of $\langle \alpha \rangle_2$ is obviously slower than other steps which just involve short modular multiplications in RNS channels and boolean XOR operations. The next subsection will, therefore, explore ways to accelerate the evaluation of $\langle \alpha \rangle_2$.

Computation of $\langle\alpha\rangle_2$

The computation of $\langle\alpha\rangle_2$ is slower than other intermediate computations but two strategies can be used to break this bottleneck.

The first strategy is to ensure that (4.48) does not wait for evaluation of (4.46). To do this $\langle X\rangle_2$ is found using (4.48) for both possible cases: $\langle\alpha\rangle_2 = 0$ and $\langle\alpha\rangle_2 = 1$. Then two results for $\lfloor \frac{X}{2} \rfloor$ can be found from (4.43). Finally, when the result from (4.46) is available, it is used to select the correct value of $\lfloor \frac{X}{2} \rfloor$.

A second strategy is to speed up the evaluation of (4.46) using careful selection of the redundant modulus m_r. If we set $\theta_i = \langle D^{-1}D_i\gamma_i\rangle_{m_r}$ and $\phi = \langle D^{-1}x_r\rangle_{m_r}$, (4.46) becomes a series of N additions modulo m_r:

$$\langle\alpha\rangle_2 = \langle\langle\sum_{i=1}^{N}\theta_i - \phi\rangle_{m_r}\rangle_2. \tag{4.49}$$

One might consider choosing $m_r = 2^k$ to accelerate this. In that case, however, $\langle 2^{-1}\rangle_{m_r}$ does not exist and the scaled residue $y_r = \langle\lfloor \frac{X}{2} \rfloor\rangle_{m_r}$ cannot be computed for the next round of scaling using $\langle(x_r - \langle X\rangle_2) \times \langle 2^{-1}\rangle_{m_r}\rangle_{m_r}$. Therefore $m_r = 2^k - 1$ is chosen.

Then, one addition modulo m_r in (4.49) becomes $\langle\theta_i + \theta_{i+1}\rangle_{2^k-1} = \langle s\rangle_{2^k-1}$, where $0 \le s \le 2m_r - 2$ as $0 \le \theta_i \le m_r - 1$ and $0 \le \theta_{i+1} \le m_r - 1$. Suppose $t = \langle s\rangle_{m_r}$, then

$$s = l \times (2^k - 1) + t = l \times 2^k + (t - l). \tag{4.50}$$

It can be seen that l is the most significant bit of the $k+1$-bit number s. Reducing both sides of (4.50) modulo 2^k gives

$$t = \langle s\rangle_{2^k} + l, \tag{4.51}$$

where $\langle s\rangle_{2^k}$ is actually the least significant k bits of s. Thus, an addition modulo $2^k - 1$ can be conveniently performed using an end-around adder

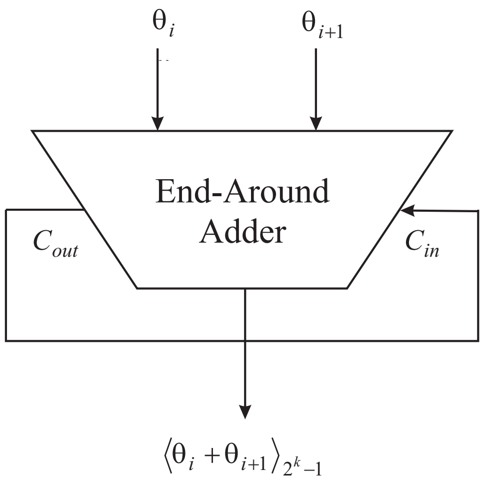


Figure 4.11: An End-Around Adder to Compute $\langle\theta_i + \theta_{i+1}\rangle_{2^k-1}$.

[Parhami00] in which the carry output l from a conventional adder is fed back into the carry input at the least significant end of the adder. The latency is only marginally longer than ordinary addition.

Figure 4.11 shows the scheme. Note that the end-around adder can be used in the RNS channels directly, because the two inputs θ_i and θ_{i+1} are both less than $m_r = 2^k - 1$, which makes the output bounded by the modulus $m_r = 2^k - 1$ too.

If a binary tree of adders is used, only $\lceil \log_2(N) \rceil$ levels of addition are required to evaluate (4.49).

Moreover, the modular multiplication units computing $\theta_i = \langle D^{-1} D_i \gamma_i \rangle_{m_r}$ and $\phi = \langle D^{-1} x_r \rangle_{m_r}$ will also be much faster than units for general modular multiplication because of the special modulus m_r [Beuchat03]. In the case of θ_i, Equation (4.51) still holds as

$$\theta_i = t = \langle s \rangle_{2^k} + \left\lfloor \frac{s}{2^k} \right\rfloor, \tag{4.52}$$

where $s = \langle D^{-1} D_i \rangle_{m_r} \times \langle \gamma_i \rangle_{m_r}$ [Zimmermann99]. (4.52) can be easily implemented using a multiplier followed by an adder.

Figure 4.12 gives the architecture for the whole process of scaling by 2.

High-Radix Architecture

In the high-radix version of the scaling algorithm, Equation (4.43) becomes

$$y_i = \langle \left\lfloor \frac{X}{2^l} \right\rfloor \rangle_{m_i} = \langle (x_i - \langle X \rangle_{2^l}) \times \langle 2^{-l} \rangle_{m_i} \rangle_{m_i} \tag{4.53}$$

where $2^l = r$ is the radix and $\langle 2^{-l} \rangle_{m_i}$ has been pre-computed. Equation (4.47) becomes

$$\langle X \rangle_{2^l} = \langle \langle D_1 \gamma_1 \rangle_{2^l} + \langle D_2 \gamma_2 \rangle_{2^l} + ... + \langle D_N \gamma_N \rangle_{2^l} - \langle \alpha D \rangle_{2^l} \rangle_{2^l}, \tag{4.54}$$

and Equation (4.49) becomes

$$\langle \alpha \rangle_{2^l} = \langle \langle \sum_{i=1}^{N} \theta_i - \phi \rangle_{m_r} \rangle_{2^l}. \tag{4.55}$$

Only the final $\langle \bullet \rangle_2$ operation in Equation (4.49) is replaced by $\langle \bullet \rangle_{2^l}$ in (4.55). This means the time to compute Equation (4.55) is almost independent of the radix. Hence this algorithm is suitable for high-radix implementations where the radix can be selected higher to reduce the number of iterations to compute $\left\lfloor \frac{X}{M} \right\rfloor$. Figure 4.13 gives the high-radix architecture for the scaling by 2^l process.

4.4.2 The Algorithm

The RNS Barrett modular multiplication algorithm at radix-2 is shown in Algorithm 4.3.

All the operations in this algorithm are short word length and within the RNS. When the channel moduli are not too large, the evaluation time is

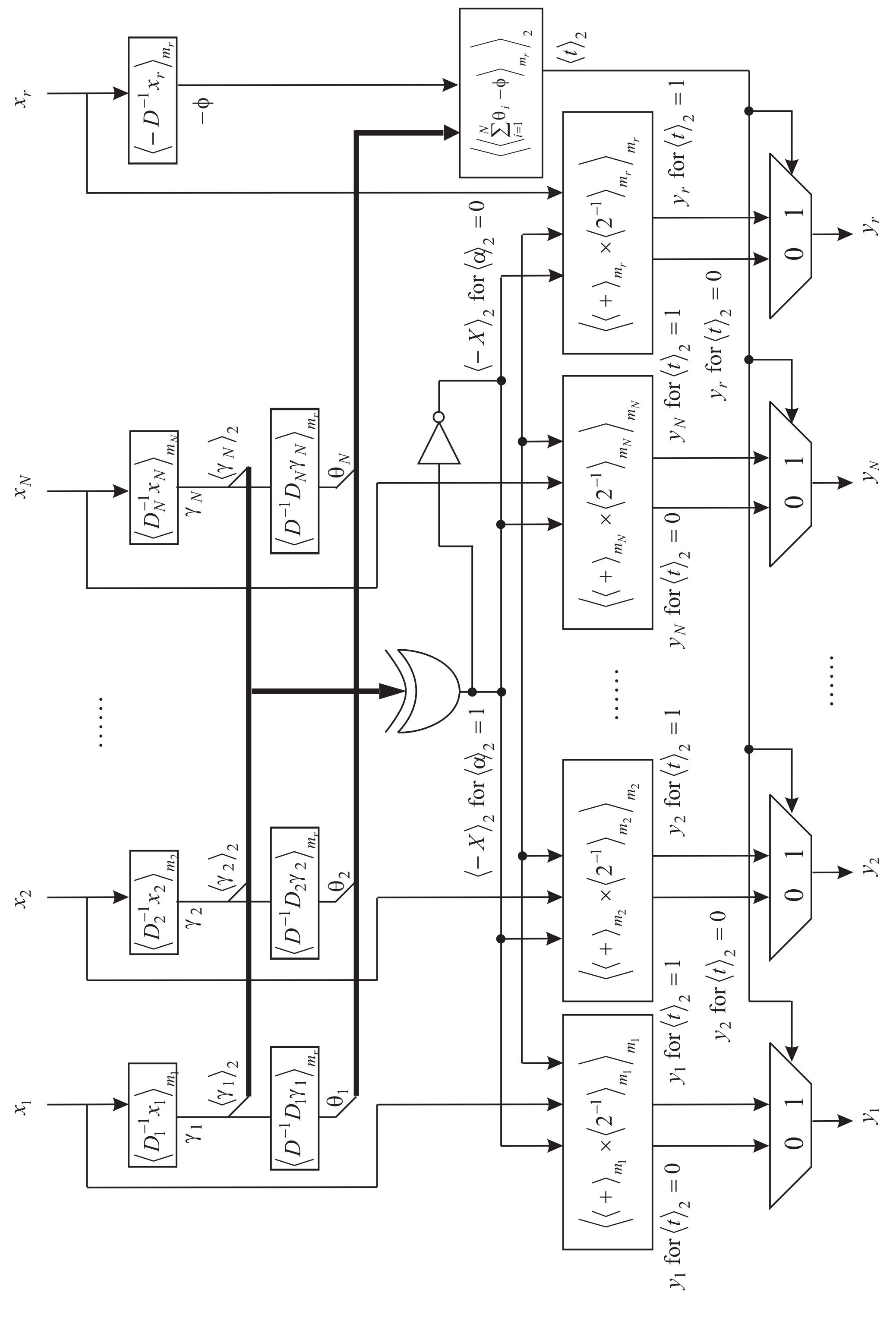


Figure 4.12: Architecture for Scaling by 2 in RNS

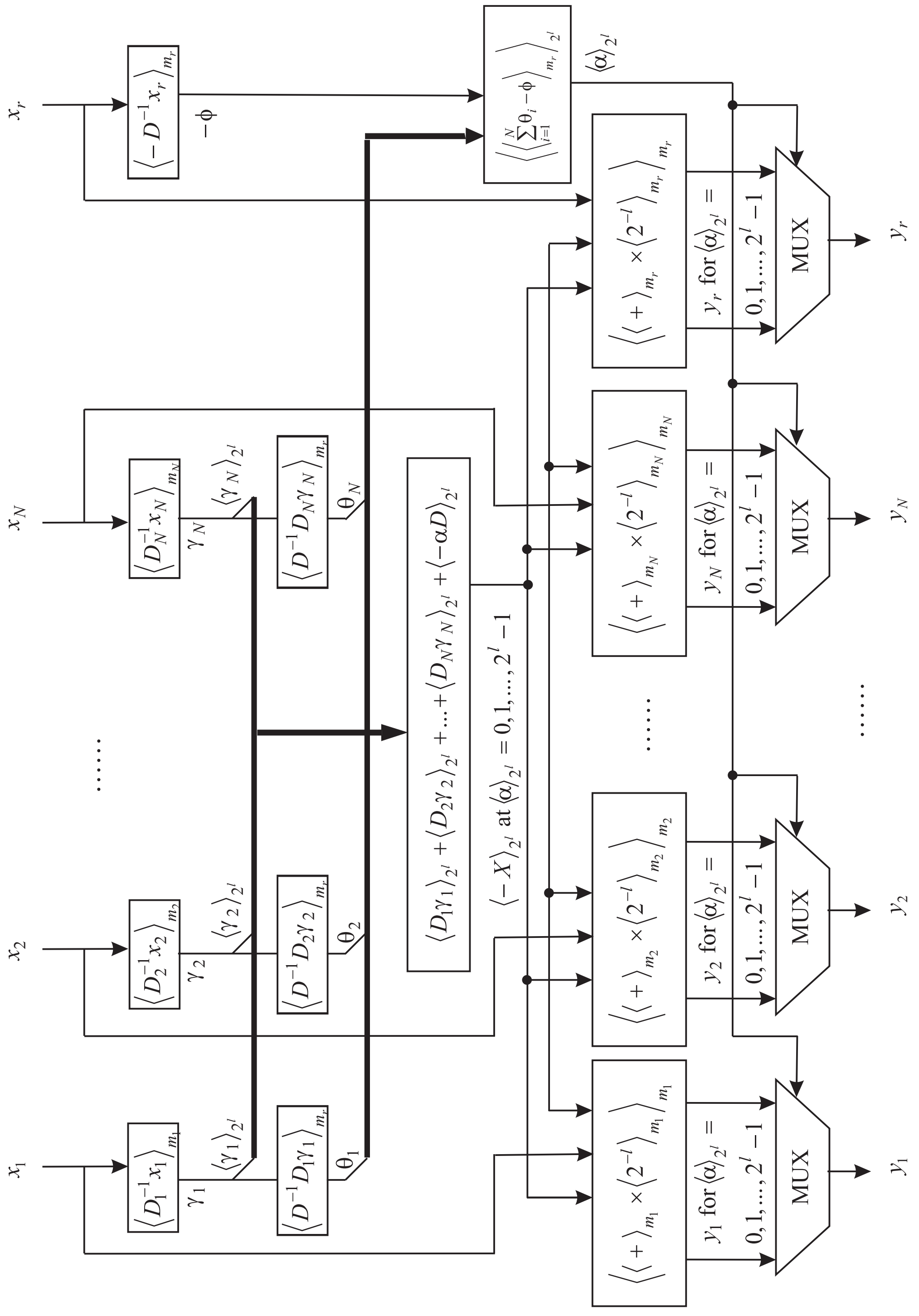

x_r
$\left\langle -D^{-1} x_r \right\rangle_{m_r}$
$-\phi$
$\left\langle \left\langle \sum_{i=1}^{N} \theta_i - \phi \right\rangle_{m_r} \right\rangle_{2^l}$
$\langle \alpha \rangle_{2^l}$
x_N
$\left\langle D_N^{-1} x_N \right\rangle_{m_N}$
γ_N
$\langle \gamma_N \rangle_{2^l}$
$\left\langle D^{-1} D_N \gamma_N \right\rangle_{m_r}$
θ_N
x_2
$\left\langle D_2^{-1} x_2 \right\rangle_{m_2}$
γ_2
$\langle \gamma_2 \rangle_{2^l}$
$\left\langle D^{-1} D_2 \gamma_2 \right\rangle_{m_r}$
θ_2
x_1
$\left\langle D_1^{-1} x_1 \right\rangle_{m_1}$
γ_1
$\langle \gamma_1 \rangle_{2^l}$
$\left\langle D^{-1} D_1 \gamma_1 \right\rangle_{m_r}$
θ_1
$\langle D_1\gamma_1 \rangle_{2^l} + \langle D_2\gamma_2 \rangle_{2^l} + \ldots + \langle D_N\gamma_N \rangle_{2^l} + \langle -\alpha D \rangle_{2^l}$
$\langle -X \rangle_{2^l}$ at $\langle \alpha \rangle_{2^l} = 0, 1, \ldots, 2^l - 1$
$\left\langle \langle + \rangle_{m_r} \times \langle 2^{-l} \rangle_{m_r} \right\rangle_{m_r}$
y_r for $\langle \alpha \rangle_{2^l} = 0, 1, \ldots, 2^l - 1$
MUX
y_r
$\left\langle \langle + \rangle_{m_N} \times \langle 2^{-l} \rangle_{m_N} \right\rangle_{m_N}$
y_N for $\langle \alpha \rangle_{2^l} = 0, 1, \ldots, 2^l - 1$
MUX
y_N
$\left\langle \langle + \rangle_{m_2} \times \langle 2^{-l} \rangle_{m_2} \right\rangle_{m_2}$
y_2 for $\langle \alpha \rangle_{2^l} = 0, 1, \ldots, 2^l - 1$
MUX
y_2
$\left\langle \langle + \rangle_{m_1} \times \langle 2^{-l} \rangle_{m_1} \right\rangle_{m_1}$
y_1 for $\langle \alpha \rangle_{2^l} = 0, 1, \ldots, 2^l - 1$
MUX
y_1

Figure 4.13: High-Radix Architecture for Scaling by 2^l in RNS

Algorithm 4.3 The Barrett Modular Multiplication Algorithm in RNS

```
Require: $M, N, w, \{m_1, \ldots, m_N, m_r\}$ as described above.
Require: pre-computed values $\langle D^{-1}\rangle_{m_r}$ and $\langle 2^{-1}\rangle_{m_r}$
Require: pre-computed table $\langle D_i\rangle_{m_i}$, $\langle D^{-1}D_i\rangle_{m_r}$, $\langle M\rangle_{m_i}$ and $\langle 2^{-1}\rangle_{m_i}$ for
    $i = 1, \ldots, N$
Ensure: $Z \equiv A \times B \bmod M$
 1: $x_i = \langle a_i \times b_i\rangle_{m_i}$ for $i = 1, \ldots, N$
 2: $\phi = \langle D^{-1}x_r\rangle_{m_r}$
 3: $\gamma_i = \langle D_i^{-1}x_i\rangle_{m_i}$ for $i = 1, \ldots, N$
 4: $\theta_i = \langle D^{-1}D_i\gamma_i\rangle_{m_r}$ for $i = 1, \ldots, N$
 5: $\langle X\rangle_2 = \langle\gamma_1\rangle_2 \oplus \langle\gamma_2\rangle_2 \oplus ... \oplus \langle\gamma_N\rangle_2$
 6: $y_i' = \langle(x_i - \langle X\rangle_2) \times \langle 2^{-1}\rangle_{m_i}\rangle_{m_i}$ for $i = 1, \ldots, N$
 7: $y_i'' = \langle(x_i - \text{NOT}(\langle X\rangle_2)) \times \langle 2^{-1}\rangle_{m_i}\rangle_{m_i}$ for $i = 1, \ldots, N$
 8: $\langle\alpha\rangle_2 = \langle\langle\sum_{i=1}^{N}\theta_i - \phi\rangle_{m_r}\rangle_2$
 9: if $\langle\alpha\rangle_2 == 0$ then
10:     $y_i = y_i'$ for $i = 1, \ldots, N$
11: else
12:     $y_i = y_i''$ for $i = 1, \ldots, N$
13: end if
14: $z_i = \langle x_i - y_i \times \langle M\rangle_{m_i}\rangle_{m_i}$ for $i = 1 \ldots N$
```

dominated by the modular accumulation $\langle\langle\sum_{i=1}^{N}\theta_i - \phi\rangle_{m_r}\rangle_{2^l}$ in the redundant channel with its special modulus of the form $2^k - 1$. A benefit of the algorithm is that modular addition and multiplication operations within this channel are much faster than for a general modulus. If these are referred to as *fast* modular operation steps, then algorithm requires $(2n+3)/l$ fast modular multiplications, $(2n+3)/l$ multiplications modulo general m_i and $\lceil\log_2(N)\rceil(2n+3)/l$ fast modular additions.

Figure 4.14 compares the latency for the 2 fast channel modular operations with 2 operations modulo general m_i. The RNS channel word length w starts from 12 bits because this is the minimum w required to construct a RNS system with 2048-bit dynamic range and equal word length moduli as stated in Section 2.1.4. The Xilinx Virtex2 FPGA used in Section 3.3.2 on page 45, XC2V1000 with a -6 speed grade, is used again. The Barrett algorithm in binary discussed in Section 3.1.3 and 3.3 is used for general modular multiplication and addition within the channels. The result from Section 3.3.2 is taken for general modular multiplication.

It can be seen that on the FPGA, a fast modular multiplication is even faster than a general modular addition. Also, a fast modular addition has roughly the same latency as an addition. At the end of next section, these results will be used to compare the RNS Barrett and RNS Montgomery algorithm as the latter uses general modular multiplications and additions, but fewer of them.

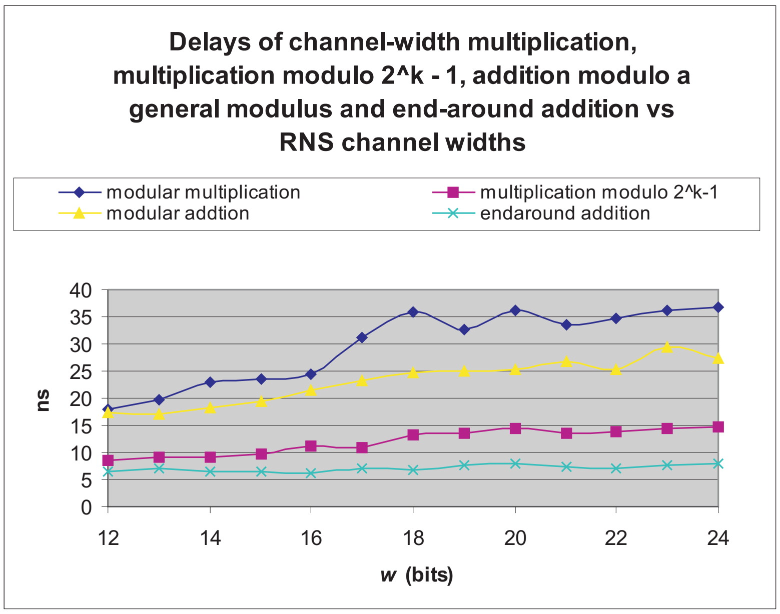


Figure 4.14: Delay of Channel-Width Multiplication and Addition modulo a General Modulus and Multiplication and Addition modulo $2^k - 1$ used for RNS Barrett Algorithm against RNS Channel Width w

4.5 RNS Montgomery Modular Multiplication

Existing schemes for multiplication modulo a long word length modulus in RNS [Posch95, Freking00, Bajard98, Bajard01, Phillips01a, Phillips03] are based on Montgomery reduction [Montgomery85]. Most of them are quite similar and this section will examine these techniques.

Recall the Montgomery algorithm in Section 3.1.4. Montgomery reduction of $X \times R^{-1}$ modulo M proceeds by adding a multiple of M according to $\langle X \rangle_M$. This approach works well in the RNS as $\langle X \rangle_M$ can be found easily for some values of R.

4.5.1 Montgomery Modular Reduction in RNS

In the RNS, the value R is chosen to be the dynamic range $D = \prod_{i=1}^{N} m_i$. This is consistent in all of the existing RNS Montgomery algorithms. Hence the RNS Montgomery reduction computes

$$Z \equiv X \times D^{-1} \mod M, \tag{4.56}$$

where Z, X and M are all represented in the RNS with modulus set $\Omega = \{m_1, m_2, \ldots, m_N\}$. As in the Montgomery algorithm in binary in Section 3.1.4, an integer q is sought such that $q < D$ and $X + qM$ is a multiple of D. Hence, the resulting division $\frac{X+qM}{D}$ is exact and easily performed in RNS by multiplying $X + qM$ by D^{-1} since $Z \equiv \langle X \times D^{-1} \rangle_M = \frac{X+qM}{D}$ [Montgomery85]. Therefore, the RNS Montgomery modular reduction algorithm is composed of 4 steps:

- Find q in Ω such that $\langle X + qM \rangle_D = 0$.

- Base extend q from Ω to $\bar{q}$ in $\bar{\Omega}$.
- Find $Z = (X + \bar{q}M)D^{-1}$ in $\bar{\Omega}$.
- Base extend Z from $\bar{\Omega}$ to Ω noting that $Z \equiv X \times D^{-1} \mod M$.

Details of these steps will be examined below.

Because $X+qM$ is a multiple of D, the representation of $X+qM$ in RNS Ω is $\{0, 0, \ldots, 0\}$ [Bajard00]. This means

$$\langle x_i + q_i \langle M \rangle_{m_i} \rangle_{m_i} = 0 \text{ for } i = 1, \ldots, N.$$

Then

$$q_i = \langle x_i M^{-1} \rangle_{m_i} \text{ for } i = 1, \ldots, N$$

where $\langle M^{-1} \rangle_{m_i}$ is pre-computed. Thus, q is obtained such that $X + qM$ is divisible by D. However, this means $(X + qM) \times D^{-1}$ cannot be computed because D^{-1} does not exist in Ω as $D = \prod_{i=1}^{N} m_i$. Therefore, it is necessary to perform this computation in another RNS with modulus set $\bar{\Omega} = \{m_{N+1}, m_{N+2}, \ldots, m_{2N}\}$, where $\bar{D} = \prod_{j=N+1}^{2N} m_j$ and $\bar{D}$ has to be co-prime to D. Again, this becomes a typical issue of base extension: q needs to be base extended from $\{q_1, q_2, \ldots, q_N\}$ to $\{q_{N+1}, q_{N+2}, \ldots, q_{2N}\}$.

Base Extension I

The technique in [Bajard01] improves this base extension based on previous publications [Posch95, Kawamura00]. From CRT,

$$q = \left\langle \sum_{i=1}^{N} D_i \langle D_i^{-1} q_i \rangle_{m_i} \right\rangle_D = \sum_{i=1}^{N} D_i \langle D_i^{-1} q_i \rangle_{m_i} - \alpha D,$$

where $\alpha < N$ from Equation (4.24) and (4.26). Set

$$\bar{q} = q + \alpha D = \sum_{i=1}^{N} D_i \langle D_i^{-1} q_i \rangle_{m_i}.$$

Reducing $\bar{q}$ modulo m_j for $j = N+1, \ldots, 2N$ yields

$$\bar{q}_j = \left\langle \sum_{i=1}^{N} \langle D_i \rangle_{m_j} \langle D_i^{-1} q_i \rangle_{m_i} \right\rangle_{m_j}, \tag{4.57}$$

where $\langle D_i^{-1} \rangle_{m_i}$ and $\langle D_i \rangle_{m_j}$ can both be pre-computed. Therefore,

$$Z = (X + \bar{q}M)D^{-1} = (X + qM)D^{-1} + \alpha M, \tag{4.58}$$

and then

$$Z \equiv X \times D^{-1} \mod M.$$

[Bajard04a] gives conditions for this result by assuming $X < DM$. From $\alpha < N$, $q < D$ and $\bar{q} = q + \alpha D$, $\bar{q} < (N+1)D$. Then, $Z < (DM + (N+1)DM)D^{-1} = (N+2)M$. Thus, both $(N+2)M < D$ and $(N+2)M < \bar{D}$ have to be satisfied to assure $Z < D$ and $Z < \bar{D}$ such that Z has a valid RNS representation in Ω and $\bar{\Omega}$. Moreover, to be able to reuse the result $Z < (N+2)M$ as inputs for the next round of RNS Montgomery modular multiplication, $((N+2)M)^2 < DM$ has to be satisfied. This gives $(N+2)^2 M < D$. Note that this has already implied $(N+2)M < D$. Thus, the prerequisites for this algorithm are

$$\begin{cases} (N+2)^2 M < D \\ (N+2)M < \bar{D}. \end{cases}$$

These are easy to satisfy. Note that $\bar{D}$ can be greater or less than D.

Base Extension II

The last step is to base extend $Z \equiv X \times D^{-1} \mod M$ back to RNS Ω. For this base extension, all RNS Montgomery algorithms [Posch95, Bajard98, Kawamura00, Bajard01, Phillips01b, Phillips03] take a method similar to [Shenoy89a], which has been discussed in Section 4.4.1. A redundant modulus channel $m_r > N$ is introduced with m_r co-prime to m_i for $i = 1, \ldots, 2N$. $\langle M \rangle_{m_r}$ and $\langle D^{-1} \rangle_{m_r}$ are pre-computed. $x_r = \langle X \rangle_{m_r}$ must be input from the beginning and $\bar{q}_r$ has to be computed using Equation (4.57). Following CRT,

$$Z = \sum_{j=N+1}^{2N} \bar{D}_j \left\langle \langle \bar{D}_j^{-1} \rangle_{m_j} Z_j \right\rangle_{m_j} - \beta \bar{D}. \tag{4.59}$$

β can be obtained similarly to Equation (4.45) as

$$\beta = \left\langle \sum_{j=N+1}^{2N} \left\langle \langle \bar{D}^{-1} \bar{D}_j \rangle_{m_r} \left\langle \left\langle \langle \bar{D}_j^{-1} \rangle_{m_j} Z_j \right\rangle_{m_j} \right\rangle_{m_r} \right\rangle_{m_r} - \left\langle \langle \bar{D}^{-1} \rangle_{m_r} Z_r \right\rangle_{m_r} \right\rangle_{m_r}$$

where Z_j for $j = N+1, \ldots, 2N$ and r is computed from Equation (4.58) as

$$Z_j = \left\langle \left\langle x_j + \bar{q}_j \langle M \rangle_{m_j} \right\rangle_{m_j} \langle D^{-1} \rangle_{m_j} \right\rangle_{m_j}.$$

Given β, Z_i is evaluated by reducing both sides of (4.59) modulo m_i for all $i = 1, \ldots, N$,

$$Z_i = \left\langle \sum_{j=N+1}^{2N} \langle \bar{D}_j \rangle_{m_i} \left\langle \langle \bar{D}_j^{-1} \rangle_{m_j} Z_j \right\rangle_{m_j} - \langle \beta \bar{D} \rangle_{m_i} \right\rangle_{m_i}.$$

4.5.2 The Algorithm

The RNS Montgomery modular multiplication algorithm is shown in Algorithm 4.4. There are $N + 5$ RNS channel multiplications and $N - 2$ channel additions needed to perform a RNS Montgomery modular multiplication [Bajard01, Bajard04a].

4.5.3 A Comparison between RNS Barrett and Montgomery Modular Multiplication

Recall that the RNS Barrett algorithm shown in Algorithm 4.3 has the advantage that most of its modular multiplications and modular additions on the

Algorithm 4.4 The Montgomery Modular Multiplication Algorithm in RNS

```
Require: $M, N, \Omega = \{m_1, \ldots, m_N\}, \bar{\Omega} = \{m_{N+1}, \ldots, m_{2N}\}, m_r$ as described above.
Require: $(N+2)^2 M < D = \prod_{i=1}^{N} m_i$ and $(N+2)M < \bar{D} = \prod_{j=N+1}^{2N} m_j$
Require: pre-computed values $\langle D^{-1} \rangle_{m_r}$ and $\langle 2^{-1} \rangle_{m_r}$
Require: pre-computed table $\langle M^{-1} \rangle_{m_i}$, $\langle M \rangle_{m_j}$, $\langle D_i^{-1} \rangle_{m_i}$, $\langle D_i \rangle_{m_j}$, $\langle \bar{D}_j \rangle_{m_i}$, $\langle \bar{D}_j^{-1} \rangle_{m_j}$, $\langle \bar{D}_j \rangle_{m_r}$, $\langle \bar{D} \rangle_{m_i}$, $\langle D^{-1} \rangle_{m_j}$ and $\langle \bar{D}^{-1} \rangle_{m_r}$ for $i = 1, \ldots, N$ and $j = N+1, \ldots, 2N$ and $r$
Ensure: $Z \equiv A \times B \bmod M$
1: $x_i = \langle a_i \times b_i \rangle_{m_i}$ for $i = 1, \ldots, 2N$ and $r$
2: $q_i = \langle x_i M^{-1} \rangle_{m_i}$ for $i = 1, \ldots, N$
3: $\bar{q}_j = \left\langle \sum_{i=1}^{N} \langle D_i \rangle_{m_j} \langle D_i^{-1} q_i \rangle_{m_i} \right\rangle_{m_j}$ for $j = N+1, \ldots, 2N$ and $r$
4: $Z_j = \left\langle \left\langle x_j + \bar{q}_j \langle M \rangle_{m_j} \right\rangle_{m_j} \langle D^{-1} \rangle_{m_j} \right\rangle_{m_j}$ for $j = N+1, \ldots, 2N$ and $r$
5: $\beta = \left\langle \sum_{j=N+1}^{2N} \left\langle \langle \bar{D}^{-1} \bar{D}_j \rangle_{m_r} \left\langle \left\langle \langle \bar{D}_j^{-1} \rangle_{m_j} Z_j \right\rangle_{m_j} \right\rangle_{m_r} \right\rangle_{m_r} - \left\langle \langle \bar{D}^{-1} \rangle_{m_r} Z_r \right\rangle_{m_r} \right\rangle_{m_r}$
6: $Z_i = \left\langle \sum_{j=N+1}^{2N} \langle \bar{D}_j \rangle_{m_i} \left\langle \langle \bar{D}_j^{-1} \rangle_{m_j} Z_j \right\rangle_{m_j} - \langle \beta \bar{D} \rangle_{m_i} \right\rangle_{m_i}$ for $i = 1, \ldots, N$
```

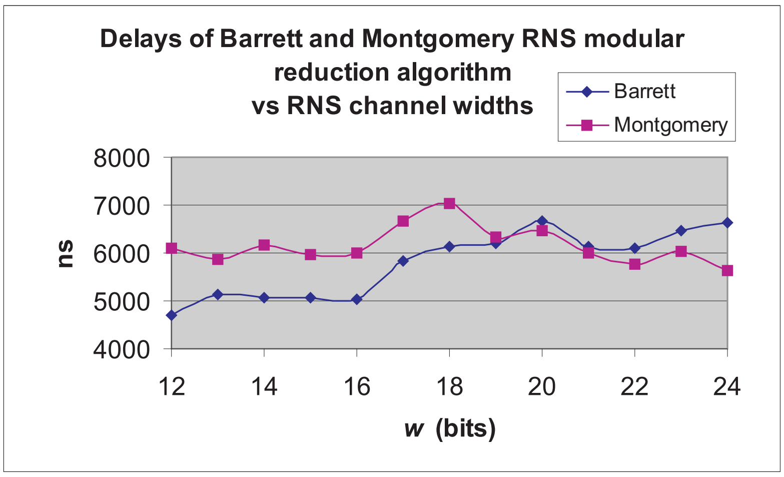


Figure 4.15: Delay of Barrett and Montgomery RNS Modular Multiplication against RNS Channel Width w

critical path are modulo its special modulus of the form $2^k - 1$ within the redundant channel. Figure 4.14 shows these *fast* modular operations are indeed faster than multiplications and additions modulo general moduli. However, the RNS Barrett algorithm requires $(2n+3)/l$ fast modular multiplications, $(2n+3)/l$ multiplications modulo a general m_i and $\lceil \log_2(N) \rceil (2n+3)/l$ fast modular additions. These are much more than the RNS Montgomery algorithm, which needs $N+5$ and $N-2$ for general modular multiplication and addition respectively.

Figure 4.15 shows the delay of the Barrett RNS algorithm and the Montgomery RNS algorithm for reduction of a 2048-bit product by a 1024-bit modulus at radix-16 on the same FPGA platform as in Section 3.3.2. These results are produced by adding the delays of channel operations in Figure 4.14 for the two algorithms.

It can be seen from Figure 4.15 that the RNS Barrett algorithm is faster when the RNS channels are less than 18 bits. As the width of the channels increases, the cost of channel operations increases while the number of channels N decreases, and hence the Montgomery approach becomes faster.

Note that in next chapter, the implementation of RNS Sum of Residues algorithm shows a better performance than both of these two implementations.

Chapter 5

Implementation of RNS Sum of Residues Modular Multiplication

The purpose of this chapter is to give an implementation for a long word length RNS modular multiplier on an FPGA platform. The Sum of Residues algorithm is chosen to achieve this. It forms a highly parallel and scalable architecture which can perform 1024-bit RSA decryption with the fastest published rate of 0.4 ms per decryption on a Xilinx Virtex5 device.

An implementation of a RNS modular multiplication algorithm depends heavily on the size of the hardware used. A hardware platform able to hold a 1024-bit RNS modular multiplication may become obsolete as demand grows for 2048-bit encryption. Thus, the RNS Sum of Residues algorithm is selected to be implemented because it is suitable for a scalable architecture and also because of its parallelism.

5.1 A Scalable Structure for Sum of Residues Modular Multiplication in RNS

Recall Algorithm 4.2 described in Section 4.3.4 as below:

Algorithm 5.1 The Sum of Residues Modular Multiplication Algorithm in RNS

```
Require: $M, N, w, \Delta, q, \{m_1, \dots, m_N\}, (N2^wM)^2 < (1-\Delta)D, N(\frac{(2^w-m_1)}{2^w} + \frac{2^{w-q}-1}{m_1}) \leq \Delta < 1$.
Require: pre-computed table $\langle D_i^{-1} \rangle_{m_i}$ for $i = 1, \dots, N$
Require: pre-computed table $\langle \langle D_i \rangle_M \rangle_{m_j}$ for $i, j = 1, \dots, N$
Require: pre-computed table $\langle \langle \alpha D \rangle_N \rangle_{m_i}$ for $\alpha = 1, \dots, N-1$ and $i = 1, \dots, N-1$
Require: $A < N2^wM, B < N2^wM$
Ensure: $Z \equiv A \times B \bmod M$
1: $X = A \times B$
2: $\gamma_i = \langle x_i D_i^{-1} \rangle_{m_i}$ for $i = 1, \dots, N$
3: $Z_i = \gamma_i \times \langle D_i \rangle_M$ for $i = 1, \dots, N$
4: $Z = \sum_{i=1}^{N} Z_i$
5: $\alpha = \left\lfloor \sum_{i=1}^{N} \left\lfloor \frac{\gamma_i}{2^{w-q}} \right\rfloor / 2^q + \Delta \right\rfloor$
6: $Z = Z - \langle \alpha D \rangle_M$
```

5.1.1 A 4-Channel Architecture

A structure for Algorithm 4.2 is illustrated in Figure 5.1 using a 4-channel RNS. Table 5.1 lists the pre-computed values required. The structure performs the following steps:

- First the product $X = A \times B$ is computed within the RNS.
- Next a RNS multiplication is performed to find the γ_is.
- Then RNS multiplications are used to compute the Z_is while the γ_is are used to generate α.
- The sum $\sum Z_i$ is performed while $\langle -\alpha D \rangle_M$ is retrieved from a table.
- Finally Z is produced by adding $\langle -\alpha D \rangle_M$ from the $\sum Z_i$.

Hence this is a highly parallel structure with only 3 RNS multiplication and $N + 1$ RNS addition steps. All of the computations are done in short word length (at most w bits) within the RNS.

To extend this architecture to longer word lengths that fit in a single integrated package, one might consider arranging a row of processing elements. The drawback of this approach is the communication required between the elements of the row. As the length of the row increases, so does the number of values to be communicated between the elements. The delay of the intra-row communication soon outweighs any benefit of the parallel structure.

Instead, the basic block shown in Figure 5.1 can be modified to permit a scalable structure using a square array of identical blocks. This eliminates all communications between the columns of the array. There must be enough columns in the array to fit the number of channels required for the large operands. The number of rows in the array is increased to account for the

Table 5.1: The Pre-Computed Constants for the RNS Sum of Residues Modular Multiplication Algorithm in Algorithm 4.2

i	1	2	3	4
$\langle D_i^{-1} \rangle_{m_i}$	$\langle D_1^{-1} \rangle_{m_1}$	$\langle D_2^{-1} \rangle_{m_2}$	$\langle D_3^{-1} \rangle_{m_3}$	$\langle D_4^{-1} \rangle_{m_4}$
$\langle \langle D_j \rangle_M \rangle_{m_i}$	$\langle \langle D_1 \rangle_M \rangle_{m_1}$	$\langle \langle D_1 \rangle_M \rangle_{m_2}$	$\langle \langle D_1 \rangle_M \rangle_{m_3}$	$\langle \langle D_1 \rangle_M \rangle_{m_4}$
	$\langle \langle D_2 \rangle_M \rangle_{m_1}$	$\langle \langle D_2 \rangle_M \rangle_{m_2}$	$\langle \langle D_2 \rangle_M \rangle_{m_3}$	$\langle \langle D_2 \rangle_M \rangle_{m_4}$
	$\langle \langle D_3 \rangle_M \rangle_{m_1}$	$\langle \langle D_3 \rangle_M \rangle_{m_2}$	$\langle \langle D_3 \rangle_M \rangle_{m_3}$	$\langle \langle D_3 \rangle_M \rangle_{m_4}$
	$\langle \langle D_4 \rangle_M \rangle_{m_1}$	$\langle \langle D_4 \rangle_M \rangle_{m_2}$	$\langle \langle D_4 \rangle_M \rangle_{m_3}$	$\langle \langle D_4 \rangle_M \rangle_{m_4}$
$\langle \langle -\alpha D \rangle_M \rangle_{m_i}$	$\langle \langle -D \rangle_M \rangle_{m_1}$	$\langle \langle -D \rangle_M \rangle_{m_2}$	$\langle \langle -D \rangle_M \rangle_{m_3}$	$\langle \langle -D \rangle_M \rangle_{m_4}$
	$\langle \langle -2D \rangle_M \rangle_{m_1}$	$\langle \langle -2D \rangle_M \rangle_{m_2}$	$\langle \langle -2D \rangle_M \rangle_{m_3}$	$\langle \langle -2D \rangle_M \rangle_{m_4}$
	$\langle \langle -3D \rangle_M \rangle_{m_1}$	$\langle \langle -3D \rangle_M \rangle_{m_2}$	$\langle \langle -3D \rangle_M \rangle_{m_3}$	$\langle \langle -3D \rangle_M \rangle_{m_4}$

residues Z_i which must be computed (in parallel) and accumulated (in series). For example, Figure 5.2 shows a 4-channel block arranged for the scalable array. This is a modified version of Figure 5.1 with extra input and output ports to account for communication down a column.

5.1.2 The Scaled Architecture for Modular Multiplication

A 12-channel modular multiplication can be built with a 3×3 array of the block in Figure 5.2. The resulting structure is shown in Figure 5.3. The leftmost column performs calculations in channels m_1 to m_4; the middle column is for m_5 to m_8; and the rightmost for m_9 to m_{12}. The columns execute independently. This is achieved by computing γ_1 to γ_N and α in each of the columns.

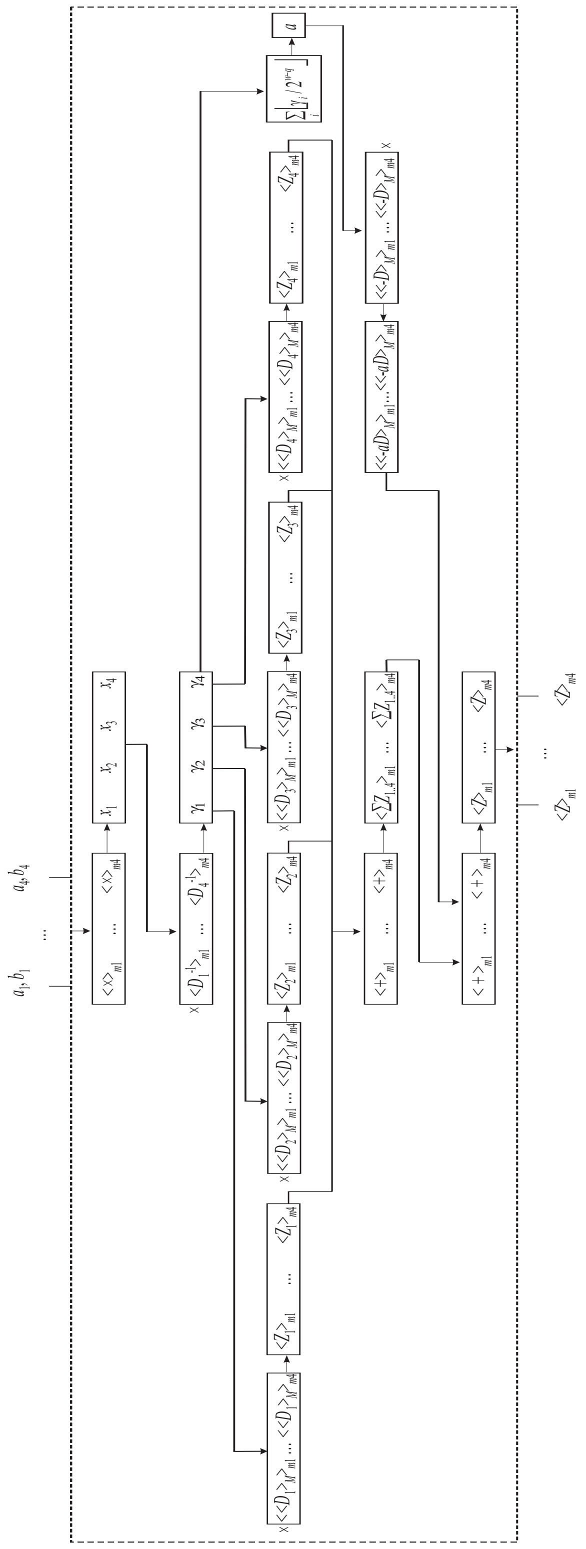


Figure 5.1: A 4-Channel RNS Modular Multiplier

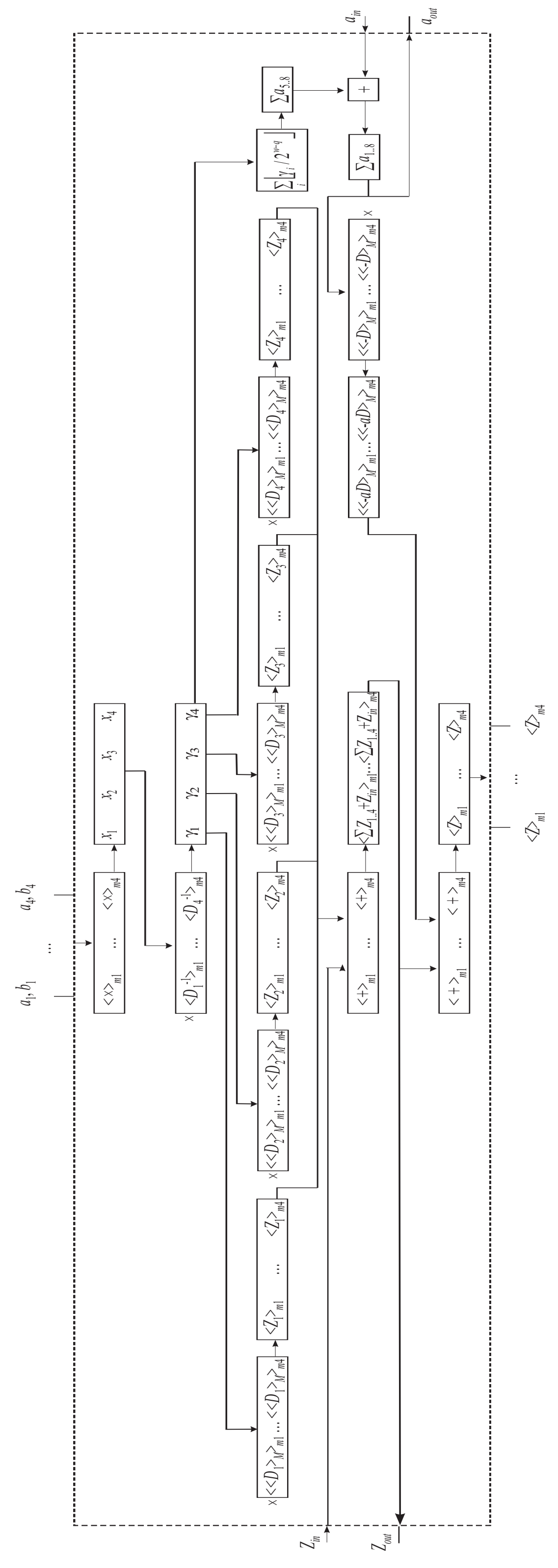


Figure 5.2: A 4-Channel Modular Multiplier Block Suitable for a Scalable Array of Blocks

The rows of the array in Figure 5.3 evaluate the residues. The first row finds Z_1 to Z_4 and $\sum_{i=1}^{4} Z_i$; the second row Z_5 to Z_8, the sum $\sum_{i=5}^{8} Z_i$ and $\sum_{i=1}^{8} Z_i = \sum_{i=1}^{4} Z_i + \sum_{i=5}^{8} Z_i$; and the last row finds Z_9 to Z_{12}, the sum $\sum_{i=9}^{12} Z_i$ and $\sum_{i=1}^{8} Z_i + \sum_{i=9}^{12} Z_i$. The sum of the residues is found by accumulating the results down a column. This comes at a cost of latency but bounds the number of I/O pins for the blocks and ensures the array is scalable. A similar scheme is used to accumulate the terms required to find α. This is off the critical path of the modular multiplication. Note that each row produces outputs for the result Z but these will be meaningless for all but the last row.

The indices in Figure 5.2 are arranged to illustrate the first block in the second row of Figure 5.3. First $\{\gamma_5, \ldots, \gamma_8\}$ are computed from the inputs $\{a_5 b_5, \ldots, a_8 b_8\}$. Then RNS multiplications with $\{\langle D_5\rangle_M, \ldots, \langle D_8\rangle_M\}$ modulo $\{m_1, \ldots, m_4\}$ are used to compute $\{\langle Z_5\rangle_{m_1}, \ldots, \langle Z_5\rangle_{m_4}\}$ $\ldots$ $\{\langle Z_8\rangle_{m_1}, \ldots, \langle Z_8\rangle_{m_4}\}$. These results are added to $\left\{\left\langle \sum Z_{1\ldots4}\right\rangle_{m_1}, \ldots, \left\langle \sum Z_{1\ldots4}\right\rangle_{m_4}\right\}$ (available at the Z_{in} input) to generate $\left\{\left\langle \sum Z_{1\ldots8}\right\rangle_{m_1}, \ldots, \left\langle \sum Z_{1\ldots8}\right\rangle_{m_4}\right\}$. These values are sent to the next element down the column via the Z_{out} output.

The critical path is shown on the middle column in Figure 5.3. The latency of the modular multiplier is given by

$$d_1 + (k-2)d_z + d_n, \tag{5.1}$$

where k is the number of rows in the array, d_1 is the delay from the a_i and b_i inputs to Z_{out} of the first block, d_z is the delay from Z_{in} to Z_{out} of any block between the first and the last block, and d_k is the delay from Z_{in} of the last block to the final output Z.

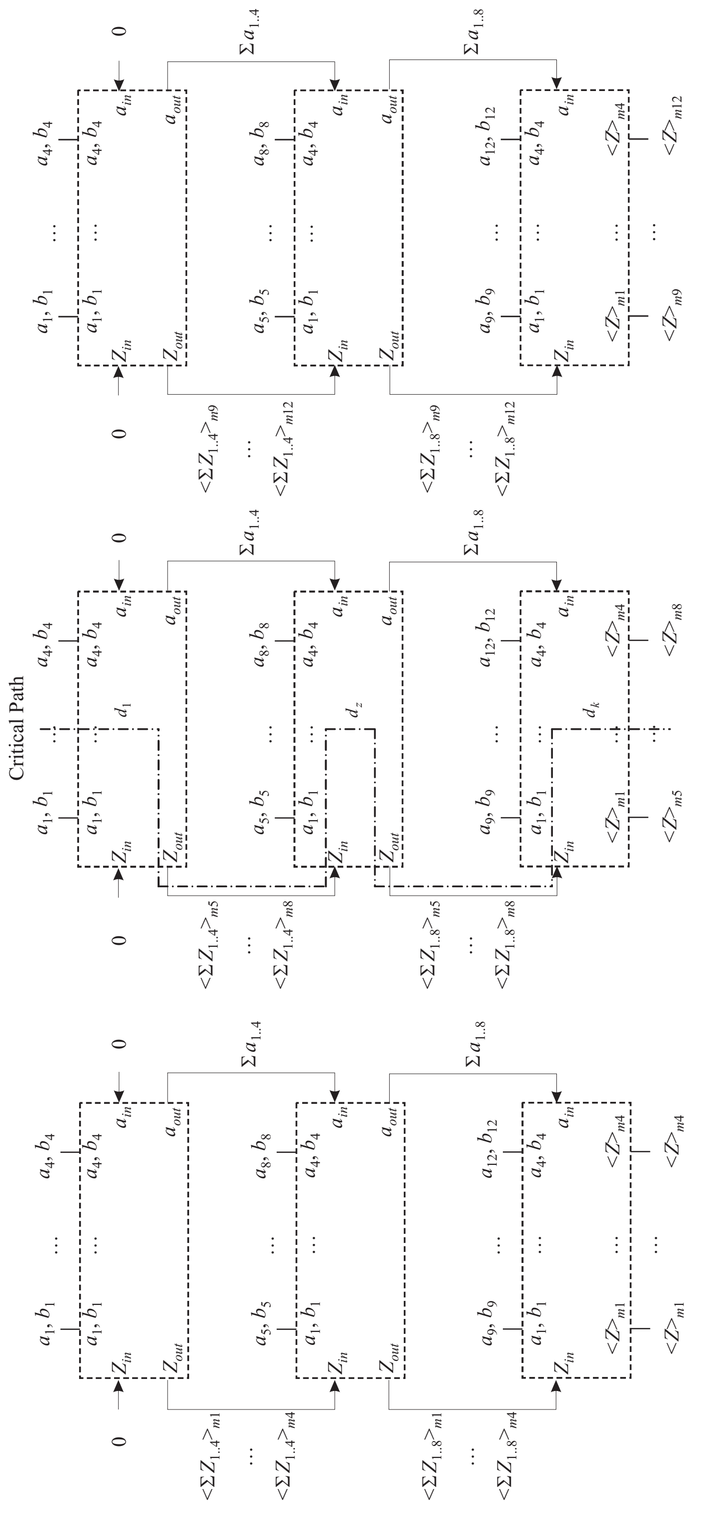


Figure 5.3: A 12-Channel Modular Multiplier Built from a 3 × 3 Array of 4-Channel Blocks

5.2 Implementation Results

To utilize the hardware efficiently, each block shown in Figure 5.2 has to hold as many channels as possible. This depends largely on the size of the hardware used. Thus, the latest version of the Xilinx Virtex5 FPGA is used as the implementation target to evaluate the performance of the RNS modular multiplier. The implementation was performed using the Xilinx ISE environment using XST for synthesis and ISE standard tools for place and route. Speed optimization with standard effort has been used.

Target FPGA:	Virtex5 XC5VSX95T with a -3 speed grade, 14720 CLB (Configurable Logic Blocks) slices and 640 DSP slices
Xilinx 9.1i:	XST - Synthesis ISE - Place and Route
Optimization Goal:	Speed
Language:	VHDL

The largest RNS modular multiplier this FPGA can accommodate was found to be 64 bits. Repeated modular multiplication of 64-bit inputs can be performed without overflow in a 10-channel RNS where each channel is 18 bits wide. The pin-to-pin delay was measured from the Post-Place and Route Static Timing Analyzer with a standard place and route effort level.

The modular multiplications within the 18-bit channels is performed using the Barrett algorithm discussed Sections 3.1.3 and 3.3 and optimized for the Xilinx Virtex2 FPGA platform used in Sections 3.3 and 3.4. The performance of this modular multiplier has already been shown in Figure 3.7 on page 46.

The latency of a single block of this 64-bit modular multiplier is 81.7 ns and the component delays for Equation (5.1) were $d_1 = 77.18$ ns, $d_z =$

20.87 ns and $d_k = 25.45$ ns. This was done by using the consecutive prime moduli $[m_1, m_2, \ldots, m_{10}] = [262051, 262069, \ldots, 262139]$, $q = 5$ and $\Delta = 0.5$. The final design consumed 78% of the DSP slices on Virtex5 but only 10% of the CLB slices. This is because the channel modular multipliers make extensive use of the hardware multipliers available in the DSP slices of the FPGA. With more DSP slices it would have been possible to include more channels within each FPGA.

Modular multiplication of 1024-bit inputs can be performed with a 16×16 array of 64-bit blocks using the consecutive prime moduli $[m_1, m_2, \ldots, m_{160}] = [260017, 260023, \ldots, 262139]$, $q = 10$ and $\Delta = 0.6$. From Equation (5.1) the delay for this array can be estimated to be $77.18 + 20.87 \times 14 + 25.45 = 394.8$ ns ≈ 0.4 μs. Similarly a 2048-bit modular multiplication can be performed with an 32×32 array using the consecutive prime moduli $[m_1, m_2, \ldots, m_{320}] = [258101, 258107, \ldots, 262139]$, $q = 10$ and $\Delta = 0.8$. The delay in this case is $77.18 + 20.87 \times 30 + 25.45 = 728.7$ ns.

According to Section 2.2.3, the number of multiplications required to evaluate a RSA decryption $C = A^B \mod M$ is $\log_2 B$ on parallel hardware. Then a 1024-bit RSA decryption could be achieved in 394.8 ns $\times$ 1024 $\approx$ 0.4 ms and a 2048-bit in 728.7 ns $\times$ 2048 $\approx$ 1.5 ms.

This can be compared with some previously published low latency results.

- The design in [Blum99] can complete a 1024-bit modular multiplication in 8.2 μs on a Xilinx XC4000 FPGA. The Montgomery modular multiplication algorithm [Montgomery85] discussed in Section 3.4 is implemented on systolic arrays with all the operations performed in binary.
- The design in [McIvor04] can perform the same operation in 2.6 μs on the same FPGA. This design again uses Montgomery algorithm in a

positional system and improves the speed by using redundant representations for intermediate results.

- The design in [Nozaki01] is the only published implementation based on the Montgomery modular multiplication in RNS [Posch95]. It uses the RNS Montgomery algorithm discussed in Section 4.5 with a Cox-Rower architecture from [Kawamura00]. This is implemented on a 0.25 μm LSI prototype and achieves 2.34 μs for a 1024-bit modular multiplication at its fastest speed.

- Finally, Figure 4.15 shows the speed of a 1024-bit modular multiplication in RNS using RNS Barrett and RNS Montgomery algorithms discussed in Section 4.4 and 4.5. They are both over 4 μs at $w = 18$. Moreover, these results are relatively theoretical because they are produced from a theoretical sum of the results of implemented channel modular operators only. The actual speed could be lower.

Chapter 6

Conclusion and Future Perspectives

The purpose of this chapter is to conclude this thesis by reflecting on its original objectives and contributions. Some suggestions for further study are made.

6.1 Conclusion

The four classes of modular multiplication algorithms possible in positional number systems can also be used for long word length modular multiplication in the RNS; moreover, using the RNS in this way will lead to faster implementations than those which restrict themselves to positional number systems. This outcome was the foremost objective of this thesis and was demonstrated through developing new Classical, Barrett and Sum of Residues algorithms for modular multiplication in the RNS and implementing the RNS Sum of Residues algorithm on an FPGA.

In Chapter 3, four classes of positional modular multiplication algorithms are analyzed, improved and implemented. Chapter 4 uses RNS to speed up long word length modular multiplication. This is achieved by devising new algorithms of Classical, Barrett and Montgomery classes that restrict all intermediate operations within short word length RNS channels. This is further demonstrated by the implementation in Chapter 5.

6.2 Future Perspectives

The RNS modular multiplication algorithms show good performance in speeding up long word length modular multiplication for the RSA Cryptosystem on an FPGA. As the word length of public-key systems increases over time, Elliptic Curve Cryptography, which uses shorter keys and moduli, will probably become another interesting application of the RNS in the future. Apart from the multiplication modulo a long word length modulus, the Elliptic Curve system involves modular additions and inversions, which also challenge the capacity of the RNS.

The benefits of the RNS to public-key systems may not be limited to speed. Investigations can proceed in other aspects of a public-key system based on the RNS, such as security. The parallel architectures possible with RNS are different to the relatively serial procedures possible in a positional system. To what extent can a RNS help a public-key system circumvent power and timing attacks by disguising internal operations through its parallel nature? This may not only be of theoretical interest.

Bibliography

[Aichholzer93] O. Aichholzer and H. Hassler. A fast method for modulus reduction in residue number system. In *Proc. Economical Parallel Processing*, pages 41–54, Vienna, Austria, 1993.

[Akushskii77] I. J. Akushskii, V. M. Burcev, and I. T. Pak. A new positional characteristic of non-positional codes and its application. In *Coding Theory and the Optimization of Complex Systems*. 1977.

[Alia84] G. Alia, F. Barsi, and E. Martinelli. A fast near optimum vlsi implementation of fft using residue number system. *The VLSI Journal*, 2:133–147, 1984.

[Alia98] G. Alia, F. Barsi, and E. Martinelli. Optimal vlsi complexity design for high speed pipeline fft using rns. *Computers & Electrical Engineering*, 24:167–182, 1998.

[Bajard98] J. C. Bajard, L. S. Didier, and P. Kornerup. An rns montgomery modular multiplication algorithm. *IEEE Trans. Comput.*, 47(7):766–776, July 1998.

[Bajard00] J.C. Bajard, L.S. Didier, P. Kornerup, and F. Rico. Some improvements on rns montgomery modular multiplication.

In *Proc. SPIE, International Symposium on Optical Science and Technology*, August 2000.

[Bajard01] J. C. Bajard, L. S. Didier, and P. Kornerup. Modular multiplication and base extensions in residue number systems. In *Proc. 15th IEEE Symposium on Computer Arithmetic*, volume 2, pages 59–65, 2001.

[Bajard04a] J. C. Bajard and L. Imbert. A full RNS implementation of RSA. *IEEE Trans. Comput.*, 53(6), June 2004.

[Bajard04b] J. C. Bajard, L. Imbert, P. Y. Liardet, and Y. Teglia. Leak resistant arithmetic. *Cryptographic Hardware and Embedded System (CHES)*, pages 62–75, 2004.

[Barraclough89] S. R. Barraclough. The design and implementation of the ims a110 image and signal processor. In *Proc. IEEE CICC*, pages 24.5/1–4, San Diego, May 1989.

[Barrett84] Paul Barrett. Communications authentication and security using public key encryption - a design for implementation. Master's thesis, Oxford University, September 1984.

[Barrett87] Paul Barrett. Implementing the Rivest, Shamir and Adleman public-key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology - Crypto 86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, Berlin/Heidelberg, Germany, 1987.

[Barsi95] F. Barsi and M. C. Pinotti. Fast base extension and precise scaling in RNS for look-up table implementations. *IEEE Trans. Signal Processing*, 43(10):2427–2430, October 1995.

[Batina02] L. Batina and G. Muurling. Montgomery in practice: How to do it more efficiently in hardware. In B. Preneel, editor, *Proceedings of RSA 2002 Cryptographers' Track*, number 2271 in Lecture Notes in Computer Science, pages 40–52. Springer, San Jose, USA, Feb 2002.

[Baugh73] C. Baugh and B. Wooley. A two's complement parallel array multiplication algorithm. *IEEE Trans. Comput.*, 22(12):1045–1047, December 1973.

[Beuchat03] Jean-Luc Beuchat, Laurent Imbert, and Arnaud Tisserand. Comparison of modular multipliers on fpgas. In *Proceedings of SPIE*, volume 5205, pages 490–498, 2003.

[Bhardwaj98] M. Bhardwaj and B. Ljusanin. The renaissance – a residue number system based vector co-processor. In *Proc. 32nd Asilomar Conference on Signals, Systems and Computers*, pages 202–207, 1998.

[Blakley83] G. R. Blakley. A computer algorithm for calculation the product ab modulo m. *IEEE Trans. Comput.*, C-32:497–500, June 1983.

[Blum99] T. Blum and C. Paar. Montgomery modular exponentiation on reconfigurable hardware. In *Proc. 14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia, April 1999.

[Boyd93] C. Boyd. Modern data encryption. *Electronics and Communication Engineering*, pages 271–278, 1993.

[Brickell83] Ernest F. Brickell. A fast modular multiplication algorithm with application to two key cryptography. In *Advances in Cryptology - Crypto 82*, Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, Germany, 1983.

[Brickell91] E. F. Brickell and K. S. McCurley. Interactive identification and digital signature. *AT & T Technical Journal*, pages 78–86, 1991.

[Burgess97] Neil Burgess. Scaled and unscaled residue number system to binary conversion techniques using the core function. In *Proc. 13th IEEE Symposium on Computer Arithmetic*, pages 250–257, June 1997.

[Burgess03] N. Burgess. Scaling an RNS number using the core function. In *Proc. 16th IEEE Symposium on Computer Arithmetic*, 2003.

[Burks46] A. Burks, H. Golodstine, and J. von Neumann. *Preliminary discussion of the logical desing of an electronic computing instrument*, volume 1. Princeton, advanced study edition, 1946.

[Chang94] C. C. Chang, W. J. Horng, and D. J. Buehrer. A cascade exponentiation evaluation scheme based on the lempel-ziv-welch compression algorithm. *Journal of Information Science and Engineering*, 1994.

[Chen99] Chien-Yuan Chen and Chin-Chen Chang. A fast modular multiplication algorithm for calculating the product ab modulo n. *Information Processing Letters*, 72:77–81, 1999.

[Chiou93] C. W. Chiou. Parallel implementation of the RSA public - key cryptosystem. *International Journal of Computer Mathematics*, 48(153-155), 1993.

[Crandall01] R. Crandall and C. Pomerance. *Prime Numbers, A Computational Perspective.* Springer-Verlag, 2001.

[Dadda65] L. Dadda. Some schemes for parallel multipliers. In *Alta Frequenza*, volume 34, pages 349–356, May 1965.

[Dhanesha95] H. Dhanesha, K. Falakshahi, and M. Horowitz. Array-of-arrays architecture for parallel floating point multiplication. In *Proc. Conf. Advanced Research in VLSI*, pages 150–157, 1995.

[Dhem98] J.-F. Dhem. *Design of an efficient public-key cryptographic library for RISC based smart cards.* PhD thesis, Université Catholique de Louvain, May 1998.

[Diffie76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, November 1976.

[Doche06] C. Doche and L. Imbert. Extended double-base number system with applications to elliptic curve cryptography. In *Progress in Cryptology - INDOCRYPT 2006*, volume 4329. Springer, November 2006.

[ElGamal85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31(4):469–472, 1985.

[Elleithy91] K.M. Elleithy and M.A. Bayoumi. A massively parallel RNS architecture. In *25th Asilomar Conference, 1991*, volume 2 of *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, pages 408–412 vol. 1, Pacific Grove, CA, Nov 1991. Institute of Electrical and Electronics Engineers Computer Society.

[Findlay90] P. A. Findlay and B. A. Johnson. Modular exponentiation using recursive sums of residues. In *Advances in Cryptology*

- *Crypto 89*, volume 435 of *Lecture Notes in Computer Science*, pages 371–386. Springer, Berlin/Heidelberg, Germany, 1990.

[Freking00] W. L. Freking and K. K. Parhi. Modular multiplication in the residue number system with application to massively-parallel public-key cryptography systems. In *Proc. 34th Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1339–1343, 2000.

[Garcia99] A. Garcia and A. Lloris. A look-up scheme for scaling in the RNS. *IEEE Trans. Comput.*, 48(7):748–751, July 1999.

[Gonnella91] J. Gonnella. The application of core functions to residue number systems. *IEEE Trans. Signal Processing*, 39:284–288, 1991.

[Griffin89] M. Griffin, M. Sousa, and F. Taylor. Efficient scaling in the residue number system. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 89, 1989.

[Hankerson04] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, 2004.

[Hatamian86] M. Hatamian and G. Cash. A 70-mhz 8-bit $\times$ 8-bit parallel pipelined multiplier in 2.5-μm cmos. *JSSC*, 21(4):505–513, August 1986.

[Hennessy90] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, San Mateo, 1990.

[Itoh01] N. Itoh and et al. A 600-mhz 54 $\times$ 54-bit multiplier with rectangular-styled wallace tree. *IEEE Journal of Solid-State Circuits*, 36(2):249–257, February 2001.

[Jenkins93] W. K. Jenkins. Finite arithmetic concepts. In S. K. Mitra and J. F. Kaiser, editors, *Handbook for Digital Signal Processing*, pages 611–675. Wiley, 1993.

[Jullien78] G. A. Jullien. Residue number scaling and other operations using rom arrays. *IEEE Trans. Comput.*, 27(4):325–336, April 1978.

[Kawamura88] Shin-Ichi Kawamura and Kyoko Hirano. A fast modular arithmetic algorithm using a residue table. In *Advances in Cryptology - Eurocrypt 88*, volume 330 of *Lecture Notes in Computer Science*, pages 245–250. Springer, Berlin/Heidelberg, Germany, 1988.

[Kawamura00] S. Kawamura, M. Koike, F. Sano, and A. Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In *Advances in Cryptology - Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 523–538. Springer, 2000.

[Knuth69] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison - Wesley, 1969.

[Knuth97] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison - Wesley, third edition, 1997.

[Koblitz87] N. Koblitz. *A Course in Number Theory and Cryptography.* Graduate Texts in Mathematics 114. Springer-Verlag, 1987.

[Kong05] Y. Kong and B. Phillips. Residue number system scaling schemes. In S. F. Al-Sarawi, editor, *Proc. SPIE, Smart Structures, Devices, and Systems II*, volume 5649, pages 525–536, February 2005.

[Kong07] Y. Kong and B. Phillips. Simulations of modular multipliers on fpgas. In *Proceedings of the IASTED Asian Conference on Modelling and Simulation*, page 11281131, Beijing, China, October 2007.

[Mahesh00] M. N. Mahesh and M. Mehendale. Low power realization of residue number system based fir filters. In *13th Int. Conf. On VLSI Design*, pages 30–33, Bangalore, India, January 2000.

[McIvor04] C. McIvor, M. McLoone, and J. V. McCanny. Modified montgomery modular multiplication and RSA exponentiation techniques. *IEE Proceedings Computers and Digital Techniques*, 151(6), November 2004.

[Menezes97] A. J. Menezes, P. V. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[Meyer-Bäse03] U. Meyer-Bäse and T. Stouraitis. New power-of-2 RNS scaling scheme for cell-based ic design. *IEEE Trans. VLSI Syst.*, 11(2):280–283, April 2003.

[Miller83] D. D. Miller, J. N. Polky, and J. R. King. A survey of soviet developments in residue number theory applied to digital filtering. In *Proceedings of the 26th Midwest Symposium on Circuits and Systems*, 1983.

[Montgomery85] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[Mou90] Z. Mou and F. Jutand. A class of close-to-optimum adder trees allowing regular and compact layout. In *Proc. IEEE Intl. Conf. Computer Design*, pages 251–254, 1990.

[Nozaki01] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura. Implementation of RSA algorithm based on RNS montgomery multiplication. In *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2001)*, pages 364–376, September 2001.

[Ohkubo95] N. Ohkubo and et al. A 4.4 ns cmos 54 $\times$ 54-bit multiplier using pass-transistor multiplexer. *IEEE Journal of Solid-State Circuits*, 30(3):251–257, 1995.

[Omondi07] A. Omondi and B. Premkumar. *Residue Number Systems – Theory and Implementation*, volume 2 of *Advances in Computer Science and Engineering: Texts.* Imperial College Press, UK, 2007.

[Orup91] Holger Orup and Peter Kornerup. A high-radix hardware algorithm for calculating the exponential m^e modulo n. In *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, volume 576, pages 51–57, 1991.

[Orup95] Holger Orup. Simplifying quotient determination in high-radix modular multiplication. In *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pages 193–199, 1995.

[Parhami96] B. Parhami. A note on digital filter implementation using hybrid rns-binary arithmetic. *Signal Processing*, 51:65–67, 1996.

[Parhami00] B. Parhami. *Computer Arithmetic – Algorithms and Hardware Designs*. Oxford University Press, 2000.

[Phillips01a] B. Phillips. Modular multiplication in the montgomery residue number system. In *Proc. 35th Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1637–1640, 2001.

[Phillips01b] B. Phillips. Montgomery residue number systems. *Electronics Letters*, 37(21):1286–1287, 2001.

[Phillips03] B. Phillips. Scaling and reduction in the residue number system with pairs of conjugate moduli. In *Proc. 37th Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 2247– 2251, 2003.

[Posch95] K. C. Posch and R. Posch. Modulo reduction in residue number systems. *IEEE Trans. Parallel Distrib. Syst.*, 6(5):449–454, May 1995.

[Preethy99] A. P. Preethy and D. Radhakrishnan. A 36-bit balanced moduli mac architecture. In *Proc. IEEE Midwest Symposium On Circuits and Systems*, pages 380–383, 1999.

[Richman71] F. Richman. *Number Theory : An Introduction to Algebra.* Contemporary undergraduate mathematics series. 1971.

[Rivest78] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Rivest85] R. L. Rivest. RSA chips (past/present/future). In *Advances in Cryptology - Eurocrypt 84*, volume 209 of *Lecture Notes in Computer Science*, pages 159–165. Springer, New York, 1985.

[Robertson58] J. E. Robertson. A new class of digital division methods. *IRE Transactions on Electronic Computers*, 7:218–222, September 1958.

[Santoro89] M. Santoro. *Design and Clocking of VLSI Multipliers*. Ph.d. thesis, Stanford University, CSL-TR-89-397, 1989.

[Schneier96] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 2nd edition, 1996.

[Schnorr89] C. P. Schnorr. Efficient identification and signature for smart cards. In *Advances in Cryptology - Crypto 89*, Lecture Notes in Computer Science. Springer, New York, 1989.

[Shand93] M. Shand and J. Vuillemin. Fast implementations of RSA cryptography. In *Proc. 11th IEEE Symposium on Computer Arithmetic*, pages 252–259, 1993.

[Shenoy89a] A. Shenoy and R. Kumaseran. A fast and accurate RNS scaling technique for high speed signal processing. *IEEE Trans. Acoust., Speech, Signal Processing*, 37(6):929–937, June 1989.

[Shenoy89b] A. Shenoy and R. Kumaseran. Fast base extension using a redundant modulus in RNS. *IEEE Trans. Comput.*, 38(2):292–297, February 1989.

[Soderstrand86] M. A. Soderstrand, W. Jenkins, and G. Jullien. Residue number system arithmetic: Modern applications. *Digital Signal Processing*, 1986.

[Standards91] National Institute Standards and Technology. A proposed federal information processing standard for digital signature standard (dss). In *Federal Register*, volume 56, pages 42980–42982. 1991.

[Su96] F. F. Su and T. Hwang. Comments on iterative modular multiplication without magnitude comparison. In *Proc. 6th National Conference on Information Security*, pages 21–22, Taichung Taiwan, 1996.

[Sutherland99] I. S. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann, San Francisco, CA, 1999.

[Szabo67] N. S. Szabo and R. H. Tanaka. *Residue Arithmetic and its Applications to Computer Technology*. McGraw Hill, New York, 1967.

[Taylor82] F. J. Taylor and C. H. Huang. An autoscale residue multiplier. *IEEE Trans. Comput.*, 31(4):321–325, April 1982.

[Tomlinson89] A. Tomlinson. Bit-serial modular multiplier. *Electronics Letters*, 25(24):1664, November 1989.

[Ulman98] Z. D. Ulman and M. Czyzak. Highly parallel, fast scaling of numbers in nonredundant residue arithmetic. *IEEE Trans. Signal Processing*, 46:487–496, February 1998.

[Wallace64] C. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, pages 14–17, February 1964.

[Walter92] C. D. Walter. Faster multiplication by operand scaling. In *Advances in Cryptology - Crypto 91*, volume 576 of *Lecture Notes in Computer Science*, pages 313–323. Springer, Berlin/Heidelberg, Germany, 1992.

[Walter94] C. D. Walter. Logarithmic speed modular multiplication. *Electronics Letters*, 30(17):1397–1398, August 1994.

[Walter95] C. D. Walter. Still faster modular multiplication. *Electronics Letters*, 31(4):263–264, February 1995.

[Walter99] C. D. Walter. Montgomery's multiplication technique: how to make it smaller and faster. In C.-K. Koc and C. Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99 Worcester, MA, USA, August 12-13, 1999 Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 80–93. Springer-Verlag, Berlin, Germany, 1999.

[Weinberger58] A. Weinberger and J. Smith. A logic for high-speed addition. In *System Design of Digital Computer at the National Bureau of Standards: Methods for High-Speed Addition and Multiplication*, volume 23 of *Circular 591*, chapter 1, pages 3–12. National Bureau of Standards, February 1958.

[Weinberger81] A. Weinberger. 4-2 carry-save adder module. In *IBM Technical Disclosure Bulletin*, volume 23, pages 3811–3814. January 1981.

[Weste04] N. Weste and D. Harris. *CMOS VLSI Design – A Circuit and Systems Perspecitive.* Addison Wesley, third edition, 2004.

[Yen93] S. M. Yen and C. S. Laih. The fast cascade exponentiation algorithm and its application on cryptography. In *Advances in Cryptology - Auscrypt 92*, Lecture Notes in Computer Science. Springer, New York, 1993.

[Yen94] S. M. Yen, C. S. Laih, and A. K. Lenstra. Multi-exponentiation. In *IEE Proc. Computers and Digital Techniques*, volume 141, pages 325–326, 1994.

[Zimmermann99] R. Zimmermann. Efficient vlsi implementation of modulo (2^n1) addition and multiplication. In Koren and Kornerup, editors, *Proceedings 14th IEEE Symposium on Computer Arithmetic*, pages 158–167, Adelaide, Australia, 1999. IEEE Computer Society Press.

[Zuras86] D. Zuras and W. McAllister. Blalanced delay trees and combinatorial division in vlsi. *IEEE Journal of Solid-State Circuits*, 21(5):814–819, October 1986.