JOÃO CARLOS NÉTO

**LOW-POWER MULTIPLICATION METHOD FOR
PUBLIC-KEY CRYPTOSYSTEM**

**MÉTODO DE MULTIPLICAÇÃO DE BAIXA
POTÊNCIA PARA CRIPTOSISTEMA DE
CHAVE-PÚBLICA**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção do
Título de Doutor em Ciências.

São Paulo
2013

JOÃO CARLOS NÉTO

**LOW-POWER MULTIPLICATION METHOD FOR
PUBLIC-KEY CRYPTOSYSTEM**

**MÉTODO DE MULTIPLICAÇÃO DE BAIXA
POTÊNCIA PARA CRIPTOSISTEMA DE
CHAVE-PÚBLICA**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção do
Título de Doutor em Ciências.

Área de Concentração:

Sistemas Digitais

Orientador:

Prof. Dr. Wilson Vicente Ruggiero

Co-orientador:

Prof. Dr. Alexandre Ferreira Tenca

São Paulo
2013

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 10 de junho de 2013.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Néto, João Carlos

Low-power multiplication method for public-key cryptosystem

Método de multiplicação de baixa potência para criptosistema de chave-pública / J.C. Néto. – ed. rev. – São Paulo, 2013.

118 p.

Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Segurança de computadores 2. Criptologia 3. Algoritmos
4. Hardware 5. Arquiteturas paralelas I. Universidade de São Paulo.
Escola Politécnica. Departamento de Engenharia de Computação e
Sistemas Digitais II. t.

AGRADECIMENTOS

Ao professor Alexandre Tenca, meu co-orientador e professor, pela preciosa orientação, dedicação, paciência e confiança depositadas na minha pesquisa e nesta tese, em inúmeras reuniões semanais realizadas.

Ao professor Wilson Ruggiero, meu orientador e professor, pelo inestimável apoio ao meu programa de doutorado, na orientação e certeza do resultado deste trabalho e outros que realizamos.

Aos membros da banca de qualificação, professora Nadia Nedjah e professores Paulo Barreto e Edson Horta, bem como aos membros da comissão julgadora, professora Karin Strauss e professores Paulo Barreto e Routo Terada pelas orientações, críticas e observações relevantes a esta tese.

Aos colegas do LARC - Laboratório de Arquitetura e Redes de Computadores, em especial ao Fernando Redigolo e sua equipe de suporte a infraestrutura e ferramentas providas para este trabalho.

Às empresas Intel e Synopsys pelos seus programas de apoio às universidades.

E, principalmente, à minha esposa Amália e minhas filhas Milene e Karla cujo amor, carinho e compreensão foram importantes na realização desta tese e, também, pelo enorme apoio, auxílio e estímulo que tornaram menos árduos os esforços realizados.

RESUMO

Esta tese estuda a utilização da aritmética computacional para criptografia de chave pública (PKC – Public-Key Cryptography) e investiga alternativas ao nível da arquitetura de sistema criptográfico em hardware que podem conduzir a uma redução no consumo de energia, considerando o baixo consumo de potência e o alto desempenho em dispositivos portáteis com energia limitada. A maioria desses dispositivos é alimentada por bateria. Embora o desempenho e a área de circuitos consistem desafios para o projetista de hardware, baixo consumo de energia se tornou uma preocupação em projetos de sistema críticos.

A criptografia de chave pública é baseada em funções aritméticas como a exponenciação e multiplicação módulo. PKC prove um esquema de troca de chaves autenticada por meio de uma rede insegura entre duas entidades e fornece uma solução de grande segurança para a maioria das aplicações que devem trocar informações sensíveis.

Multiplicação em módulo é largamente utilizada e essa operação aritmética é mais complexa porque os operandos são números extremamente grandes. Assim, métodos computacionais para acelerar as operações, reduzir o consumo de energia e simplificar o uso de tais operações, especialmente em hardware, são sempre de grande valor para os sistemas que requerem segurança de dados. Hoje em dia, um dos mais bem sucedidos métodos de multiplicação em módulo é a multiplicação de Montgomery. Os esforços para melhorar este método são sempre de grande importância para os projetistas de hardware criptográfico e de segurança em sistemas embarcados.

Esta pesquisa trata de algoritmos para criptografia de baixo consumo de energia. Abrange as operações necessárias para implementações em hardware da exponenciação e da multiplicação em módulo. Em particular, esta tese propõe uma nova arquitetura para a multiplicação em módulo chamado "Parallel k -Partition Montgomery Multiplication" e um projeto inovador em hardware para calcular a exponenciação em módulo usando o sistema numérico por resíduos (RNS).

Palavra-chave: Criptografia, Aritmética de alta performance, Exponenciação e multiplicação em módulo, Base numérica alta, Baixa potência, Tolerante a falhas, Sistema numérico por resíduos.

ABSTRACT

This thesis studies the use of computer arithmetic for Public-Key Cryptography (PKC) and investigates alternatives on the level of the hardware cryptosystem architecture that can lead to a reduction in the energy consumption by considering low power and high performance in energy-limited portable devices. Most of these devices are battery powered. Although performance and area are the two main hardware design goals, low power consumption has become a concern in critical system designs.

PKC is based on arithmetic functions such as modular exponentiation and modular multiplication. It produces an authenticated key-exchange scheme over an insecure network between two entities and provides the highest security solution for most applications that must exchange sensitive information.

Modular multiplication is widely used, and this arithmetic operation is more complex because the operands are extremely large numbers. Hence, computational methods to accelerate the operations, reduce the energy consumption, and simplify the use of such operations, especially in hardware, are always of great value for systems that require data security. Currently, one of the most successful modular multiplication methods is Montgomery Multiplication. Efforts to improve this method are always important to designers of dedicated cryptographic hardware and security in embedded systems.

This research deals with algorithms for low-power cryptography. It covers operations required for hardware implementations of modular exponentiation and modular multiplication. In particular, this thesis proposes a new architecture for modular multiplication called Parallel k -Partition Montgomery Multiplication and an innovative hardware design to perform modular exponentiation using Residue Number System (RNS).

Keywords: Cryptography, High-Speed Arithmetic, Modular Exponentiation and Modular Multiplication, High-Radix, Low-Power, Fault-Tolerant, Residue Number System.

CONTENTS

List of Figures

List of Tables

1	Introduction	13
1.1	Motivation	14
1.2	Public-Key Cryptography	16
1.2.1	Diffie-Hellman Key Agreement	16
1.2.2	RSA	17
1.2.3	ECC	18
1.3	Research Objectives	20
1.4	Thesis Outline	20
2	Low-Power Design	22
2.1	Power Consumption	22
2.1.1	Dynamic Power	22
2.1.2	Leakage Power	23
2.2	Power Optimization	24
2.2.1	Voltage Scaling	24
2.2.2	Multiple-Voltage	25

2.2.3	Clock Gating	26
2.2.4	Precomputation Logic	26
2.2.5	Guarded Evaluation	26
2.2.6	Retiming	27
2.2.7	Parallelization	27
2.2.8	IEEE Standard 1801	28
3	Montgomery Multiplication	29
3.1	Notations	29
3.2	Montgomery Reduction	30
3.3	Montgomery Algorithm	31
3.4	Montgomery Exponentiation	34
3.5	Design and Implementation Strategies	35
3.5.1	RNS	36
3.5.2	Recoding of Multiples	36
3.5.3	Partitioning	36
3.5.4	Bipartite Method	37
3.5.5	Tripartite Method	38
4	Residue Number System	40
4.1	Representation	40
4.2	Basic Structure	43
4.3	Moduli Selection	44

4.4	Conversion from Binary to RNS Representation	45
4.4.1	Modulus 2^n	45
4.4.2	Modulus $2^n - 1$	46
4.4.3	Modulus $2^n + 1$	46
4.4.4	Special Moduli-Set $\{2^n - 1, 2^n, 2^n + 1\}$	47
4.5	Conversion from RNS to Binary Representation	49
4.5.1	Chinese Remainder Theorem	50
4.5.2	Mixed Radix Conversion	51
4.6	Montgomery Multiplication in RNS	52
5	Hardware Algorithms for Low-Power Modular Multiplication	56
5.1	k -Partition Method	56
5.1.1	Montgomery Multiplication Partition (MMP)	56
5.1.2	The k -Partition Montgomery Multiplication (k PMM)	59
5.1.2.1	Correctness	60
5.1.2.2	Adding partial results Z_{P_j}	61
5.1.2.3	Asymptotic Analysis of Algorithm 7	61
5.1.2.4	Numerical Example	62
5.2	Montgomery Multiplication in RNS	63
5.2.1	Montgomery Exponentiation in RNS (MEXPRNS)	64
5.2.1.1	Correctness	64
5.2.1.2	Asymptotic Analysis of Algorithm 8	66

5.2.1.3	Numerical Example	68
6	Design of Low-Power Multipliers	70
6.1	Parallel k -Partition Method	70
6.1.1	MM Partition Kernel Architecture	70
6.1.2	MM Partition j Architecture (MMP)	72
6.1.3	Parallel k -Partition MM Architecture (k PMM)	72
6.1.4	Optimizing for Better Power	74
6.1.5	Complexity Evaluation of the Proposed Architecture	76
6.2	Montgomery Exponentiation in RNS	78
6.2.1	Forward Conversion	79
6.2.2	Exponentiation Modulo m_i	81
6.2.3	Reverse Conversion	82
6.2.4	MM Extended Architecture (MME)	83
6.2.5	Dual Mode MM Architecture (DMMM)	85
6.2.6	Dual MM Kernel Architecture (DMMK)	86
7	Experimental Results	91
7.1	Parallel k -Partition Method	91
7.1.1	Benchmark	91
7.1.2	Analysis of the Energy Consumption	96
7.2	Montgomery Exponentiation in RNS	98
7.2.1	Benchmark	98

7.2.2	Analysis of the Energy Consumption	100
8	Future Work	102
8.1	k -Partition Architecture - Further Improvements	102
8.2	k PMM Architecture with Spare Module	103
8.2.1	A Spare MMP	103
8.2.2	Fault Tolerant k PMM Architecture	104
8.2.3	External Fault Detection	105
8.3	Montgomery Exponentiation in RNS	105
8.4	ECC	106
8.5	System Level Energy Characterization	106
8.6	Physical Security	107
9	Conclusions	108
9.1	Research Contributions	108
9.2	Publications	109
	References	111

LIST OF FIGURES

1	Modular multiplication using MM	31
2	Basic Structure of an RNS Processor	43
3	The distribution of bits of X into two decomposed multiplier operands	58
4	The distribution of bits of X into two multiplier operands	63
5	Architecture of the MM Partition Kernel (MMP Kernel)	71
6	MM Partition j Architecture (MMP)	72
7	Fully Parallel k -Partition MM Architecture (k PMM) - Top Level . . .	73
8	Sparse-Carry-Save Adder in Dot Notation	75
9	Optimized Architecture of MM Partition Kernel	75
10	The impact of the block sizes on the power consumption of the MM Partition Kernel	76
11	Architecture of Montgomery Exponentiation in RNS (MEXPRNS) – Top Level	80
12	Architecture of Forward Conversion (FC) Data Router and Controller	81
13	MM Extended Architecture (MME) – Top Level	84
14	Dual Mode MM Architecture (DMMM)	86
15	Architecture of Dual MM Kernel (DMMK)	87
16	Single Multiplication Mode of $CSA\ 2_1$ and $CSA\ 2_0$ Adders, in Dot Notation	88

17	Architecture of Modular Exponentiation (ME) Data Router and Controller	89
18	Architecture of Reverse Conversion (RC) Data Router and Controller	90
19	Comparison in terms of the multiplication time	94
20	Comparison in terms of the total area	94
21	Comparison in terms of the energy consumption	95
22	Dynamic power versus leakage power – k PMM Architecture	95
23	The impact of the number of partitions on the energy consumption	98
24	The average power consumption blocks of the MEXPRNS architecture	101
25	The distribution of bits of X in w -bit digits	102
26	Fault-Tolerant Architecture using a reconfigurable MM Partition	105

LIST OF TABLES

1	Running in parallel 2PMM Algorithm	62
2	Running Montgomery Multiplication in the RNS Algorithm	69
3	Area and time for gate/circuit equivalents	77
4	Area and time per block of the Parallel k -Partition MM	77
5	State Transition of the Control Logic for MEXPRNS - FSM	78
6	State Transition of the Control Logic for Forward Conversion (FC) - FSM	79
7	State Transition of the Control Logic for Modular Exponentiation (ME) - FSM	82
8	State Transition of the Control Logic for Reverse Conversion (RC) - FSM	83
9	The Summary of the Report Timing, the Area, and the Energy Consumption of the MM Architectures	92
10	The Summary of the Report Timing, the Area, and the Energy Consumption of The Montgomery Exponentiation in RNS Architectures	99

1 INTRODUCTION

Research on computer arithmetic and its application to public-key cryptography is emerging as the greatest challenge for mobile devices. Most of these devices are battery powered. Although performance and area are the two main hardware design goals, low power consumption has become a concern in critical system designs.

Public-Key Cryptography (PKC) is based on arithmetic functions such as modular exponentiation and modular multiplication and provides an authenticated key-exchange scheme over an insecure network between two entities (MENEZES; OORSCHOT; VANSTONE, 1996). As a result, each party has the true identity security of the other, and thus they share a session encryption key (symmetric) known only to them to set a secure communication channel with confidentiality and integrity between the entities. PKC is the highest security solution for most applications that must exchange sensitive information.

Modular multiplications are widely used, and this arithmetic operation is more complex because the operands are extremely large numbers. Hence, computational methods to accelerate the operations, reduce the energy consumption, and simplify the use of such operations, especially in hardware, are always of great value for systems that require data security. Currently, one of the most successful modular multiplication methods is Montgomery Multiplication (MONTGOMERY, 1985). Efforts to improve this method are always important to designers of dedicated cryptographic hardware and security in embedded systems (NEDJAH; MOURELLE, 2004).

This thesis addresses the use of computer arithmetic in public-key cryptography, and investigates alternatives on the level of the hardware cryptosystem architecture that can lead to a reduction in the energy consumption by considering low power and high performance in energy-limited portable devices.

1.1 Motivation

Security information in wireless networks is extremely difficult to achieve, notably because of vulnerabilities in communication, the limited physical security of each device, intermittent connectivity, the dynamic adjustment of the topology, the absence of a certification authority, and other limitations and vulnerabilities in this environment. Additionally, the mobile computing devices have a limited capacity for data processing, memory, and power supply. Because of the innovations in wireless technology, the exceptional flexibility provided by mobile devices and their widespread adoption due to their low cost and high convenience, it is necessary to adopt a secure communication between the entities involved.

The use of cryptography to establish a secure communication channel between mobile devices and a server unit is indispensable to ensure secure communication in wireless networks.

However, the fundamental question of modern cryptography is the problem of providing a secure communication channel between two entities in an insecure communication network with the proper balance of power consumption and performance. It is thus necessary to construct a scheme that allows different entities to establish a shared cryptographic key (symmetric) in an open distributed network, where there can be both passive and active attacks. An attacker can intercept, modify, insert, and deny access to messages in communication between entities.

A private/public key pair used in a key-establishment scheme should allow these

entities to authenticate each other to guarantee that the correct encryption key is shared only between the desired entities. The scheme should ensure that each party has the true identity of the other one.

The difficulties that might arise in wireless communication are significant because wireless communication involves distributed computing by mobile devices with low-performance computing (processing and memory), low-power supply (battery), and communication over an insecure and limited communication system. One solution is the use of pre-established symmetric keys for authentication between the entities. However, this connectivity is impossible due to the diversity of mobile communication, the variety of applications, and the economic aspects of using computing resources safely in this environment.

The need for low-power and efficient cryptographic solutions leads to hardware implementations with potential energy savings and high-performance processing. It is well known that, although software platforms provide a flexible implementation solution, there are several restrictions in software platforms, such as slow encryption of a large amount of data processing and the high energy consumption required for repeated modular multiplication, which is the essence of the public-key algorithms. Clearly, this scenario involves many limitations due to computing resources, communication, usability and economic issues. The hardware solutions provide an acceptable level for the implementations of public-key cryptography in most constrained environments.

For these reasons, there is a challenge in providing a secure solution in this circumstance, which justifies studies to investigate the issue of low power consumption and high performance in low-performing portable computers to establish a secure communication channel using public-key cryptography mechanisms.

1.2 Public-Key Cryptography

Diffie and Hellman developed PKC in 1976 (DIFFIE; HELLMAN, 1976). Public-key cryptosystems are based on trap-door one-way functions, where two different keys are used (one for encryption and the other for decryption). Several public-key methods based on different one-way functions have been proposed since the 1970s. These methods, which are used today, base their security on hard mathematical problems, such as the integer factorization problem (Diffie-Hellman key exchange scheme (DIFFIE; HELLMAN, 1976) and RSA (RIVEST; SHAMIR; ADLEMAN, 1978)), the discrete logarithm problem (ElGamal (ELGAMAL, 1985) and DSA (NIST, 2009)), and the elliptic curve discrete logarithm problem (ECC (HANKERSON; MENEZES; VANSTONE, 2003)).

This scheme ensures the security in several Internet systems such as, for example, a personal computer accessing a secure web-server via a browser using the Secure Sockets Layer (SSL) protocol. However, such security depends on the size of the key, which results in an enormous computational effort in the PKC calculation.

1.2.1 Diffie-Hellman Key Agreement

The Diffie-Hellman key agreement is a protocol that gives the first result to the key exchange problem for two parties (MENEZES; OORSCHOT; VANSTONE, 1996). Consider two users called A and B ; each sends the other one message over an insecure network. The basic protocol is to share a secret key K known to both parties A and B , as follows:

1. *One time setup*: An appropriate prime p and a generator α of $\mathbb{Z}_p^*$ ($2 \leq \alpha \leq p - 2$) are selected and published, where $(p - 1)/2 = p'$, and p' is also a prime.

2. *Protocol messages*:

$$A \rightarrow B : \alpha^x \pmod{p} \tag{1.1}$$

$$A \leftarrow B : \alpha^y \pmod{p} \tag{1.2}$$

3. *Protocol actions*: Perform the following steps each time a shared key is required.

- (a) A chooses a random secret x , $1 \leq x \leq p - 2$, and sends B a message (1.1).
- (b) B chooses a random secret y , $1 \leq y \leq p - 2$, and sends A a message (1.2).
- (c) B receives α^x and computes the shared key as $K = (\alpha^x)^y \pmod{p}$.
- (d) A receives α^y and computes the shared key as $K = (\alpha^y)^x \pmod{p}$.

This basic version of the protocol provides protection in the form of the confidentiality of the resulting key from passive attacks, but not from active attacks capable of intercepting, modifying, or injecting messages (MENEZES; OORSCHOT; VANSTONE, 1996). The recommendation NIST SP-800-56A provides the specifications of key agreement schemes, which is based on the Diffie-Hellman and Menezes-Qu-Vanstone algorithms (BARKER; JOHNSON; SMID, 2007).

1.2.2 RSA

The RSA cryptosystem is the most widely used PKC. It may be used to provide both secrecy and digital signatures, and its security is based on the intractability of the integer factorization problem (RIVEST; SHAMIR; ADLEMAN, 1978), (MENEZES; OORSCHOT; VANSTONE, 1996).

All of the cryptographic operations, such as key agreement, encryption, decryption, signature generation, and signature verification are performed using the modular exponentiation of integers.

To encrypt a message m , the sender computes the ciphertext $c \equiv m^e \pmod{M}$, where e is the public or encrypting exponent. To decrypt a ciphertext c and obtain the plaintext m , the receiver computes $m \equiv c^d \pmod{M}$, where d is the private or decrypting exponent.

A two-parameter vector denotes the public key of a user, (M, e) . The first parameter

is the modulus M , which is the product of two large random and distinct secret primes p and q , each roughly of the same size, where $\phi = (p - 1)(q - 1)$. The other parameter, e , is also a random number selected such that $1 < e < \phi$, and $\gcd(e, \phi) = 1$. Using the extended Euclidean algorithm the decryption key d is computed, such that $1 < d < \phi$, and $(ed) \equiv 1 \pmod{\phi}$. In other words, d is the unique integer such that $d \equiv e^{-1} \pmod{\phi}$.

The values (p, q, d) denotes the private key and are kept secret by the user who generates the RSA keys.

The public exponent (e) is used in encryption and signature verification. However, the private exponent (d) is used in decryption and signature generation.

1.2.3 ECC

Elliptic Curve Cryptography (ECC) is a group of cryptographic methods based on elliptic curves, which depend on arithmetic related to the points of the curve. The curve arithmetic is defined in terms of the underlying field operations, the efficiency of which is essential. Efficient curve operations are likewise crucial for performance (HANKERSON; MENEZES; VANSTONE, 2003), (BLAKE et al., 2005).

ECC depends on efficient algorithms for finite field arithmetic operations such as inversion, multiplication and addition. Thus, PKC can be defined over two finite fields, either the prime Galois Field, $GF(p)$ or the binary extension Galois Field, $GF(2^m)$. There are several types of defining equations for elliptic curves, but the most common are the Weierstrass equations, and the following description is based on (IEEE STD 1363, 2000), (HANKERSON; MENEZES; VANSTONE, 2003), (BLAKE et al., 2005), (SECG. SEC 1, 2009).

In $GF(p)$, the Weierstrass equation is given by

$$y^2 \pmod{p} \equiv x^3 + ax + b \pmod{p}, \quad (1.3)$$

where p is a prime number, $4a^3 + 27b^2 \pmod{p} \neq 0$, and $a, b \in GF(p)$. All of the modular arithmetic operations such as addition, subtraction, division, and multiplication involve integers in the range of $[0, p - 1]$. According to the recommendation in (SECG. SEC 1, 2009), the elliptic curve parameter p over $GF(p)$ must have

$$\lceil \log_2 p \rceil \in \{192, 224, 256, 384, 521\}. \quad (1.4)$$

In $GF(2^m)$, the Weierstrass equation is

$$y^2 + xy = x^2 + ax^2 + b \pmod{p}, \quad (1.5)$$

where $b \neq 0$. The elements of $GF(2^m)$ are integers with a length of at most m bits. These numbers can be considered as a binary polynomial of degree $m - 1$. All of the operations, such as addition, subtraction, division, and multiplication, involve polynomials of degree $m - 1$. Following the recommendation in (SECG. SEC 1, 2009), the elliptic curve parameter m over $GF(2^m)$ must have

$$\lceil \log_2 m \rceil \in \{163, 233, 239, 283, 409, 571\}. \quad (1.6)$$

The standard (IEEE STD 1363, 2000) contains nearly all public-key algorithms, and, in particular, it covers Diffie-Hellman Key Exchange with Elliptic Curves (ECDH), the Elliptic Curve Digital Signature Algorithm (ECDSA), the Elliptic Curve Menezes–Qu–Vanstone protocol (ECMQV), and the Elliptic Curve Integrated Encryption Scheme (ECIES), which are different schemes for asymmetric cryptographic key exchange and agreement protocols based on ECC.

ECDH is the elliptic curve variant of the Diffie-Hellman key agreement protocol, which allows two parties, each having an elliptic curve public-private key pair, to establish a shared secret over an insecure network. It provides a variety of security goals that depend on its application, e.g., unilateral implicit key authentication, mutual implicit key authentication, known-key security, and forward secrecy (BLAKE et al., 2005),

(SECG. SEC 1, 2009).

ECMQV is another key agreement scheme based on ECC. It provides some security goals depending on its application, e.g., mutual implicit key authentication, known-key security, and forward secrecy (BLAKE et al., 2005), (SECG. SEC 1, 2009).

1.3 Research Objectives

As mentioned above, modular exponentiation and modular multiplication are the most important arithmetic functions in these public-key cryptosystems because they are the most widely used and require a large amount of data processing.

This research aims to create an efficient hardware architecture to reduce energy consumption without sacrificing performance with the use of arithmetic functions to perform the calculations involved in public-key cryptography.

Our focus will be to analyze, design and implement improvements at the algorithm level and their integrated circuit descriptions to optimize hardware resources. Finally, we will develop a proof of concept by building an architecture in a controlled framework to evaluate and measure the experimental results.

1.4 Thesis Outline

This thesis is organized as follows. Chapter 2 presents a brief overview of the power consumption in digital circuits and low-power design methodologies. Chapter 3 introduces the concepts of Montgomery reduction, general methods for the Montgomery algorithm and some hardware optimization approaches. Chapter 4 covers the relevant knowledge concerning the Residue Number Systems (RNS) and describes the proposed implementation of the Parallel Montgomery Multiplication algorithm in RNS for low-power and high-performance multipliers. Chapter 5 focuses

on different topics to optimize the design of modular multipliers for low power. In Chapter 6, some design approaches for low-power multipliers are proposed to reduce the energy consumption. All experimental results for the optimization and design of the low-power multiplier are summarized in Chapter 7. Directions for future work are discussed in Chapter 8, and Chapter 9 presents the contributions and publication produced as a result of this research.

2 LOW-POWER DESIGN

To provide arithmetic functions for low-power hardware, power consumption becomes the key design focus. This chapter presents a description of the main points involved in the sources for power consumption in digital circuits. Furthermore, some methods for limiting power consumption at different levels of circuit design are shown. The detailed methodologies for low-power design can be found in the following books and standard: (RABAEY; PEDRAM, 1996), (YEAP, 1997), (PEDRAM, 2002), (PIGUET, 2004), (IEEE STD 1801, 2009).

2.1 Power Consumption

Two major sources of power consumption in a given digital circuit block are summarized in the following equation (RABAEY; PEDRAM, 1996):

$$P_{Block} = P_{Dynamic} + P_{Leakage}. \quad (2.1)$$

The dynamic power consumption ($P_{Dynamic}$) depends on the operating frequency and the number of transitions. On the other hand, the static or leakage power ($P_{Leakage}$) depends on the circuit size.

2.1.1 Dynamic Power

The dynamic power consumption is associated with circuit switching activities during logic transitions, with consists of two components, the switching power and

the cell internal power. The dynamic power consumption consists of two power consumption components, as shown in the following equation (RABAEY; PEDRAM, 1996):

$$P_{Dynamic} = \alpha C_L V_{DD}^2 F_{Clk} N + Q_{SC} V_{DD} F_{Clk} N. \quad (2.2)$$

In equation (2.2), the first term represents the switching power consumption required to charge and discharge the internal and net capacitances. Here, we denote the ratio of switching activity over a given time with α , C_L is the load capacitance, V_{DD} is the supply voltage, and F_{Clk} is the operating frequency. The factor N is the switching activity, which is in terms of the number of gate output transitions per clock cycle.

The second term in equation (2.2) represents the power dissipation during output transitions due to current (short-circuit) flowing from the supply to ground. The factor Q_{SC} represents the quantity of charge carried by the short-circuit current per transition.

For several technologies until a 65nm feature size, the dominant source of power consumption in hardware design is dynamic power. The architecture and application are the main issue to reduce the dynamic power (AHUJA; LAKSHMINARAYANA; SHUKLA, 2012). The power optimization techniques discussed in this thesis concentrate on reducing the dynamic power consumption for low-power design.

2.1.2 Leakage Power

The leakage power dissipation is due to the leakage current that flows whenever power is applied in a given digital circuit block. The leakage power is not related to the clock frequency or the switching activity and is represented by the following equation (RABAEY; PEDRAM, 1996):

$$P_{Leakage} = I_{Leakage} V_{DD}, \quad (2.3)$$

where $I_{Leakage}$ denotes the two major currents. The sub-threshold current is caused by a low threshold voltage and the gate current is caused by the reduced thickness of gate oxide produced by the NMOS and PMOS transistors. It is directly determined by the number of gates and the technology process.

For the technology cell library used herein, the values of the cell leakage power are significantly lower than the dynamic power. The integrated circuit industry is projecting that the leakage power will dominate the overall power consumption, given the trend towards high performance and high density which requires smaller geometries (SYLVESTER; KAUL, 2001). However, a great amount of investment is made every year to overcome this problem and again make the leakage power much less significant than this natural tendency is predicting.

2.2 Power Optimization

The power optimization methods presented in this research concentrate on energy savings at the logic and algorithm levels as well as on the circuit and architecture levels.

In this work, some of these methods are available in the tools applied such as Synopsys Design Compiler and Power Compiler. Other methods are introduced by the designer when the hardware description is made.

2.2.1 Voltage Scaling

The basic strategy to reduce the power consumption in a circuit is by reducing the supply voltage because power is the square of the supply voltage, as shown in the following equations:

$$I = \frac{V}{R}, \quad (2.4)$$

$$P = IV = \frac{V^2}{R}, \quad (2.5)$$

where I , V , R , and P denote the current, the supply voltage, the impedances (according to Ohm's law), and the power consumption in the circuit respectively.

A similar amount of power reduction can be achieved in a given digital circuit block for both dynamic and leakage power.

Unfortunately, energy savings with voltage scaling can be limited due to the influence on circuit performance (POUWELSE; LANGENDOEN; SIPS, 2001). In general, a lower voltage tends to reduce the circuit speed.

2.2.2 Multiple-Voltage

Using different voltages in some parts of a given digital circuit may reduce the overall energy consumption of a design (PEDRAM; ABDOLLAHI, 2005).

Some technologies support different threshold voltages, which can provide two or more individual cells to achieve, with a given logic function, each using a different transistor threshold voltage.

For example, the library can provide two cells, one using low threshold voltage transistors and another using high threshold voltage transistors. A low threshold voltage cell has a higher speed and hence a higher subthreshold leakage current. However, a high threshold voltage cell has a low leakage current and less speed.

Therefore, a high threshold voltage cell can use low threshold voltage cells in the timing-critical paths for speed and high threshold voltage cells everywhere else for lower leakage power.

A method to reduce power without changing the circuit function by making use of two supply voltages is described in (USAMI; HOROWITZ, 1995).

2.2.3 Clock Gating

Clock gating is one of the most effective ways to reduce the dynamic power (BENINI; MICHELI, 1998). Clock gating techniques reduce the clock power by stopping the clock signals for selected registers during times when the stored logic values are not changing. This method is useful for registers that must maintain the same values over multiple clock cycles by removing unnecessary switching activity.

2.2.4 Precomputation Logic

This method is a powerful sequential logic optimization that duplicates part of the logic with the purpose of precomputing the output logic values one clock cycle before they are required, and then uses these values to reduce switching activity in the succeeding clock cycle (ALIDINA et al., 1994).

Knowing the output values one clock cycle in advance allows the original logic to be turned off in the next clock cycle and will have significantly reduced switching activity. The size of the logic that precalculates the output values determines the power dissipation reduction, the area growth and the delay increase relative to the original circuit.

In arithmetic circuits, the precomputation logic is an effective way to precompute values that are often employed by the method and store them in registers.

2.2.5 Guarded Evaluation

Guarded evaluation is a method for reducing the power required by a combinational circuit when some of its input values are not needed in the consecutive clock cycle (TIWARI; MALIK; ASHAR, 1995). It works by stopping the clock signal through unused circuits by limiting the dynamic power consumption.

Unlike the clock gating and precomputation logic methods, this method does not

require the synthesis of additional logic to perform the shutdown mechanism. Rather, it works the existing clock signals in the original circuit.

The method is based on placing some guard logic, consisting of transparent latches with an enable signal, at the inputs of each block of the circuit that must be power-managed. When the block executes some useful computation in a clock cycle, the activation signal causes the latches be transparent. Otherwise, the latches retain their past states by blocking any transition within the logic block. Guarded evaluation provides a way to determine where the transparent latches must be placed within the circuit and by which signals they must be controlled (PEDRAM; ABDOLLAHI, 2005).

2.2.6 Retiming

Retiming is the method of changing the position of latches or registers within a circuit to improve its performance, its area, and its power characteristics in such a way that operations are performed in different clock cycles without changing the overall behavior at its outputs (LEISERSON; SAXE, 1991).

A modified cost function that is power aware and tries to place flip-flops under timing constraints in a way that minimizes switching activity was proposed in (MONTEIRO; DEVADAS; GHOSH, 1993).

In synthesis tools such as Synopsys Design Compiler, retiming is used to create pipelined functional units by redistributing a cascade of registers placed at the output of the unit. Thus, the designer should be capable of carefully placing registers in the circuit design.

2.2.7 Parallelization

The use of parallel computing with multiple processing units has become progressively accepted, and it covers a wide range of cost and performance (CORMEN

et al., 2009). The main problems of algorithm parallelization are the partitioning process, resources management and dealing with a trade-off among speed, cost area, and energy consumption.

In parallel architectures, applications may be run on a variable number of the processors, which may operate at different frequencies. The performance and the power consumption of an algorithm running on a parallel architecture demonstrate an opposed trade-off that considers how many processors the algorithm uses, in which frequencies these processors operate, and the structure of the algorithm.

Some investigators have studied the performance scalability of parallel algorithms for some time (PARHAMI, 1999), (GRAMA et al., 2003). Others have studied how to determine the optimal number of processors that minimizes power consumption to execute a given algorithm and maximizes its performance (LI; MARTINEZ, 2006), (KORTHIKANTI; AGHA, 2009), (KORTHIKANTI; AGHA, 2011).

2.2.8 IEEE Standard 1801

The IEEE Standard 1801 for Design and Verification of Low Power Integrated Circuits consists of a set of commands used to specify the design intent for multivoltage electronic systems.

The purpose of this standard is to provide portable low power design specifications that can be used with a variety of commercial products throughout an electronic system design, analysis, verification, and implementation flow (IEEE STD 1801, 2009).

3 MONTGOMERY MULTIPLICATION

This chapter presents an overview of modular multiplication, which is the core operation of the public-key cryptography. The most popular algorithm for modular multiplication is Montgomery Multiplication (MONTGOMERY, 1985). The concepts of Montgomery reduction, the general methods for the Montgomery algorithm and some hardware optimization approaches are introduced.

3.1 Notations

To explain the Montgomery Multiplication (MM) algorithms, we use the following variables and notations.

Let M be an n -bit odd modulus. For generic operands in the ring formed by modulus M , we want to have $n = 1 + \lfloor \log_2 M \rfloor$ to cover all operand bits. When the multiplier operand is shorter, $k < \lfloor \log_2 M \rfloor$, we can use $n = k$, as described in Subsections 3.5.4 and 3.5.5. The Montgomery radix R is typically chosen such that $R = 2^n$. Let R^{-1} be the multiplicative inverse of R , such that $\gcd(M, R) = 1$ and $(RR^{-1}) \equiv 1 \pmod{M}$. In this thesis, M is considered as a generic modulus (a prime number). For specific values of M , the reduction may be much simpler, e.g., pseudo-Mersenne (SOLINAS, 1999).

Let x and y be the multiplication of operands with n bits in the integer domain. Let X and Y be the multiplier and multiplicand operands respectively, with n bits in the Montgomery domain, such that $X \equiv (xR) \pmod{M}$, and $Y \equiv (yR) \pmod{M}$. This

representation is usually referred to as the Montgomery representation. The summation and subtraction of two elements in the Montgomery domain also lead to an element in the Montgomery domain.

3.2 Montgomery Reduction

For a better understanding of the MM algorithms, we first introduce the Montgomery reduction. Let X and Y be two integers in the Montgomery domain and let M , R , and R^{-1} be as above. We denote the product of X and Y in the Montgomery domain as follows:

$$Z = \text{MM}(X, Y, M) \equiv (XYR^{-1}) \pmod{M}. \quad (3.1)$$

The product of X and Y is an integer Z , where $X \equiv (xR) \pmod{M}$, $Y \equiv (yR) \pmod{M}$, and $z \equiv (xy) \pmod{M}$, which satisfies the following equation:

$$\begin{aligned} Z &\equiv (zR) \pmod{M} \equiv [(xy) \pmod{M}]R \pmod{M} \\ &\equiv [(xR) \pmod{M}][(yR) \pmod{M}]R^{-1} \pmod{M} \\ &\equiv (XYR^{-1}) \pmod{M}. \end{aligned} \quad (3.2)$$

The MM method requires conversions of x and y from the integer domain to the Montgomery domain and the conversion of the calculated result back.

The procedure is as follows. To compute $z = (xy) \pmod{M}$, we first have to compute the MM of x and y with $R^2 \pmod{M}$ to find X and Y as follows:

$$X = \text{MM}(x, R^2, M) \equiv (xR) \pmod{M}, \quad (3.3)$$

$$Y = \text{MM}(y, R^2, M) \equiv (yR) \pmod{M}. \quad (3.4)$$

Then, the product of $Z = \text{MM}(X, Y, M) \equiv (xyR) \pmod{M}$ followed by $\text{MM}(Z, 1, M)$

gives the desired result

$$\text{MM}(Z, 1) \equiv (xyRR^{-1}) \bmod M \equiv (xy) \bmod M \equiv z. \quad (3.5)$$

The conversions of elements from the integer domain to the Montgomery domain and back is shown in the following figure:

Figure 1: Modular multiplication using MM

3.3 Montgomery Algorithm

Several Montgomery Multiplication methods were analyzed at the algorithmic level in terms of space and time requirements by Koç et al. (KOC; ACAR; KALISKI JR., 1996). Although those algorithms were originally considered for software implementation, the Coarsely Integrated Operand Scanning (CIOS) method has proved to be the most efficient of all five analyzed algorithms, and it has been extensively used in hardware and software implementations (KOC; ACAR; KALISKI JR., 1996).

The main complexity of modular multiplication methods lies in a series of two lengthy operations. One of them involves the summation of the multiplicand operand multiples, and the other the summation of the modulus multiples to produce the modular reduction.

The MM algorithm is used to speed up the modular multiplication and the squaring

required during the modular exponentiation process in public-key cryptosystems without using division.

This algorithm is based on the residue system suggested by Peter Montgomery in (MONTGOMERY, 1985) to compute $SR^{-1} \pmod{M}$ without division by R , where S is an integer such that $0 \leq S \leq RM$.

The algorithm is based on the property that if $q \equiv SM' \pmod{R}$ and $M' \equiv -M^{-1} \pmod{R}$, then it follows that

$$t = \frac{S + qM}{R} \text{ is exact (} R \text{ divides } S + qM \text{).} \quad (3.6)$$

Before the final reduction $0 \leq (S + qM)/R < (RM + RM)/R$, so $0 \leq t < 2M$.

Equation (3.6) holds, which leads to

$$qM \equiv SMM' \pmod{R} \equiv -S \pmod{R}, \quad (3.7)$$

and hence R divides $S + qM$ because the least significant n bits of $S - (S \bmod R)$ are zeros.

Algorithm 1 shows the pseudo code of the Radix-2 MM for n -bit operands X , Y , and M (KOC; ACAR; KALISKI JR., 1996). It is the most common algorithm to generate a fast and simple hardware implementation.

Algorithm 1 Radix-2 Montgomery Multiplication (MM)

Require: odd M , $n = 1 + \lfloor \log_2 M \rfloor$, $X = \sum_{i=0}^{n-1} x_i 2^i$, $Y = \sum_{i=0}^{n-1} y_i 2^i$, with $0 \leq X, Y < M$

Ensure: $Z \equiv XYR^{-1} \pmod{M}$, with $0 \leq Z < M$

- 1: $S[0] \leftarrow 0$
 - 2: **for** $i \leftarrow 0$ **to** $n - 1$ **step 1 do**
 - 3: $a \leftarrow S[i] + x_i Y$
 - 4: $S[i + 1] \leftarrow (a + a_0 M)/2$
 - 5: **end for**
 - 6: **if** $S[n] \geq M$ **then**
 - 7: $S[n] \leftarrow S[n] - M$
 - 8: **end if**
 - 9: **return** $Z \leftarrow S[n]$
-

The inner loop (lines 2 to 5) of the Radix-2 MM algorithm uses two 2-input adders. The first adder sums Y to the intermediate result $S[i]$, if the current bit of X (or x_i) has a value 1. When the result of the first addition is odd, the second adder sums M to it. The intermediate result $S[i + 1]$ of each iteration is then obtained by dividing the output of the second adder by 2, thus reducing the intermediate result to n bits. The final reduction (lines 6 to 8) can be avoided, as shown by Colin D. Walter (WALTER, 1999).

We quickly prove the correctness of Algorithm 1, based on equation (3.8) which is the property extracted from lines 3 and 4.

$$S[i + 1] \equiv \left(\sum_{j=0}^i x_j 2^j \right) Y 2^{-(i+1)} \pmod{M}, \quad \forall i \geq 0. \quad (3.8)$$

This property is proved by induction.

In the first iteration, $i = 0$ and $S[0] = 0$. Thus, equation (3.8) holds for iteration 1 because

$$\begin{aligned} a &= S[0] + x_0 Y = x_0 Y, \text{ and} \\ S[1] &= \frac{a + a_0 M}{2} \equiv x_0 Y 2^{-1} \pmod{M}. \end{aligned}$$

This congruence is true because $a_0 M \equiv a M M' \pmod{2} \equiv -a \pmod{2}$, and hence 2 divides $a + a_0 M$, which satisfies equation (3.6).

Now, assuming that the property holds for iteration $i - 1$, equation (3.8) can be

shown to hold for iteration i , as follows:

$$\begin{aligned}
 a &= S[i] + x_i Y, \text{ and} \\
 S[i+1] &= \frac{a + a_0 M}{2} \equiv (S[i] + x_i Y) 2^{-1} \pmod{M} \\
 &\equiv (S[i] 2^i + x_i 2^i Y) 2^{-(i+1)} \pmod{M} \\
 &\equiv [(\sum_{j=0}^{i-1} x_j 2^j) Y + x_i 2^i Y] 2^{-(i+1)} \pmod{M} \\
 &\equiv (\sum_{j=0}^i x_j 2^j) Y 2^{-(i+1)} \pmod{M}.
 \end{aligned}$$

In the last iteration, $i = n - 1$, equation (3.8) gives the desired result:

$$S[n] \equiv (\sum_{j=0}^{n-1} x_j 2^j) Y 2^{-n} \pmod{M}.$$

3.4 Montgomery Exponentiation

The PKC schemes introduced in Section 1.2 are based on modular exponentiation in Diffie-Hellman key exchange and RSA or point/divisor multiplication in ECC. These arithmetic functions are performed by Montgomery Multiplication in their most basic forms by implementing a classical square and multiply algorithm that computes an exponentiation.

The Montgomery Exponentiation algorithm computes $z \equiv x^e \pmod{M}$ by using the MM algorithm (MENEZES; OORSCHOT; VANSTONE, 1996). A binary method based on the exponentiation algorithm using parallel processing was proposed in (CHIOU, 1993). By using the MM algorithm, the parallel binary method has been modified to perform the modular squaring and multiplication operations simultaneously.

Algorithm 2 describes the Montgomery Exponentiation algorithm (MEXP), where both MM operations (lines 4 and 6) are executed at the same time. A hardware implementation is shown in Figure 13.

Algorithm 2 Montgomery Exponentiation (MEXP)

Require: M , $n = 1 + \lfloor \log_2 M \rfloor$, $x = \sum_{i=0}^{n-1} x_i 2^i$, $e = \sum_{i=0}^{t-1} e_i 2^i$, where $e_t = 1$, $1 \leq x < M$, $R = 2^n$, with $\gcd(M, R) = 1$, $(RR^{-1}) \equiv 1 \pmod{M}$, and $R^2 \equiv RR \pmod{M}$.

Ensure: $z \equiv x^e \pmod{M}$

```

1:  $u \leftarrow 1$ 
2:  $s \leftarrow \text{MM}(x, R^2, M)$ 
3: for  $i \leftarrow 0$  to  $t - 1$  step 1 do
4:    $s \leftarrow \text{MM}(s, s, M)$ 
5:   if  $e_i = 1$  then
6:      $u \leftarrow \text{MM}(u, s, M)$ 
7:   end if
8: end for
9:  $z \leftarrow \text{MM}(1, u, M)$ 
10: return  $z$ 

```

Several algorithms for increasing the speed of modular exponentiation have been suggested since RSA was proposed. A review and a recommended framework for efficient exponentiation are available in (GORDON, 1998), (MÖLLER, 2003).

3.5 Design and Implementation Strategies

In this thesis, the strategies for low-power design and the implementation (PEDRAM, 2002) of the MM hardware considered different number representations, for example, the residue number system and the recoding of multiples, and some architectures, such as the systolic, scalable, and parallel structures. The scope herein is limited to the application of the proposed technique to the sequential Radix-2 MM algorithm proposed in (MONTGOMERY, 1985), but the same strategy may be used to improve the performance of other MM implementations, such as systolic and scalable architectures (IWAMURA; MATSUMOTO; IMAI, 1993), (WALTER, 2000), (TENCA; KOC, 2003).

3.5.1 RNS

In a Residue Number System (RNS), integers are broken into smaller components such that arithmetic operations can be performed on smaller components independently of each other. This number system is employed because the modular structure results in carry-free operations for high speed processing. The hardware implementation of RNS architectures leads to an enhancement in speed, cost, and power consumption (SZABO; TANAKA, 1967), (SODERSTRAND et al., 1986).

In Chapter 4, the relevant knowledge concerning RNS is presented, and the Parallel Montgomery Multiplication algorithm in RNS for the implementation of low-power and high-performance multipliers is proposed.

3.5.2 Recoding of Multiples

The recoding of the binary multiplier operand into a high-radix digit set (AMBERG; PINCKNEY; HARRIS, 2008) is a well-known technique to implement arithmetic algorithms in hardware. By recoding the multiplier, the number of partial products in modular multiplication is reduced and thus this approach decreases the number of clock cycles required to complete the task. Each clock cycle, however, is expected to take longer to complete. The total computation time will depend on the impact of the additional complexity introduced in the control logic for the selection of multiples of the multiplier operand and the more complex production and addition of partial products for the high-radix digit set (LEU; WU, 2000).

3.5.3 Partitioning

The method proposed by Kaihara et al. (KAIHARA; TAKAGI, 2008) computes modular multiplication using two partitions. It divides the multiplier into two parts and utilizes two different multiplication methods to compute the modular multiplication for

the lower and upper portions in parallel. Both Barrett and Montgomery multiplication algorithms are used. The method cannot be easily extended to address more than two partitions.

Other investigations were conducted to provide a more aggressive partitioning of the multiplication process. By improving the bipartite method, Yoshino et al. (YOSHINO; OKEYA; VUILLAUME, 2009) proposed novel algorithms for computing double-size modular multiplications. Recently, Sakiyama et al. (SAKIYAMA et al., 2011) improved Kaihara's method by using a tripartite method by combining Classic, Montgomery, and Karatsuba multiplication. The main disadvantage of these proposals is that each partition has a unique structure. Therefore, increasing levels of effort to develop and test are required as the number of partitions increase.

The method proposed herein is based on the divide-and-conquer approach to break the computation and assimilation of partial products. An effective result is obtained when the manipulation of multiples of the multiplicand operand is performed by distributing the multiplier operand bits into k partitions that can process them in parallel. The partitions of the original multiplier are used to express k new multiplier operands that are used to perform Montgomery Multiplication in radix 2^k . Multiples of the multiplicand operand can be computed in an easy way, without using Booth encoding (LEU; WU, 2000), because the digit set of each partition is simple. As a result, this approach accelerates modular multiplication by compressing the overall number of iterations of the original Radix-2 MM algorithm from n to n/k plus the combination and reduction of results generated by all partitions.

3.5.4 Bipartite Method

The bipartite method (KAIHARA; TAKAGI, 2008) performs modular multiplication using a representation of residue classes modulo M that permits the splitting of the multiplier X into two parts, $X = X_H R_2 + X_L$, such that R_2 is chosen as $R_2 = 2^k$,

$0 < k < n$, where $0 \leq X_H < 2^{n-k}$ and $0 \leq X_L < 2^k$, and computes $Z \equiv XYR_2^{-1} \pmod{M}$ as follows:

$$\begin{aligned} Z &\equiv [(X_H R_2 + X_L) Y R_2^{-1}] \pmod{M} \\ &\equiv [(X_H Y) \pmod{M} + (X_L Y R_2^{-1}) \pmod{M}] \pmod{M}. \end{aligned} \quad (3.9)$$

The purpose of this method is to improve the speed by using two traditional modular multiplications, processing them in parallel. The classical modular multiplication calculates the left term of equation (3.9), $(X_H R_2) \pmod{M}$, which is based on the Barrett multiplication algorithm (BARRETT, 1984), and the Montgomery method is used to compute the right term, $(X_L Y R_2^{-1}) \pmod{M}$.

Both MM and Barrett multiplication interleave multiplication and modular reduction phases. Barrett requires the precomputation of the reciprocal of the modulus M and single shift and multiplication operations.

The bipartite method has an unbalanced complexity, forcing one algorithm to dominate the longest path. Barret multiplication tends to produce a longer path than MM due to the difficulty to compute the multiples of the modulus M . The different hardware algorithms used in each partition lead to more complexity for testing and fabrication. In practice, the unbalanced complexity problem can be mitigated by using non-uniform partition sizes.

3.5.5 Tripartite Method

The tripartite method (SAKIYAMA et al., 2011) integrates two modular multiplications algorithms (Classic and Montgomery) with the Karatsuba multiplication approach. The basic idea is based on the separation of the multiplier X into two parts, $X = X_H R_3 + X_L$, and the multiplicand Y into other similar parts, $Y = Y_H R_3 + Y_L$, such that R_3 is chosen as $R_3 = 2^k$, $k = \lceil n/2 \rceil$, where $0 \leq X_H < 2^{n-k}$, $0 \leq X_L < 2^k$,

$0 \leq Y_H < 2^{n-k}$, and $0 \leq Y_L < 2^k$. The method calculates $Z \equiv (X.Y.R_3^{-1}) \mod M$ using the following equation:

$$\begin{aligned}
 Z &\equiv [(X_H R_3 + X_L)(Y_H R_3 + Y_L) R_3^{-1}] \mod M \\
 &\equiv [(X_H Y_H R_3) \mod M + (X_H Y_L + X_L Y_H) \mod M + \\
 &\quad (X_L Y_L R_3^{-1}) \mod M] \mod M \\
 &\equiv [(P_1 R_3) \mod M + (P_2 - P_0 - P_1) \mod M + \\
 &\quad (P_0 R_3^{-1}) \mod M] \mod M,
 \end{aligned} \tag{3.10}$$

where $P_0 = X_L Y_L$, $P_1 = X_H Y_H$ and $P_2 = (X_H + X_L)(Y_H + Y_L)$.

By using the Karatsuba method recursively at the algorithmic level, the time complexity is reduced and allows parallel processing.

However, it should be noted that the Bipartite and Tripartite methods are more complex in terms of the control logic among partitions than a uniform method because they use very different algorithms.

4 RESIDUE NUMBER SYSTEM

Residue Number Systems (RNS) are based on the Chinese Remainder Theorem (CRT), which allows for fast parallel arithmetic (GARNER, 1959), (KNUTH, 1981). The properties of RNS have led to its usage in hardware applications, such as cryptography, digital filtering, convolution, correlation, fast Fourier transforms (FFT), image processing, communication, and other applications with high number of arithmetic operations. A fundamental understanding of RNS and their applications are available in the following books and papers: (SZABO; TANAKA, 1967), (SODERSTRAND et al., 1986), (TAYLOR, 1984), (OMONDI; PREMKUMAR, 2007), (BAJARD; IMBERT, 2004).

In this chapter, we review the basic concepts of RNS, and then we conclude by giving a method for computing the Parallel Montgomery Multiplication algorithm in the RNS *Modulo Channels*. We consider an implementation to optimize modular multipliers for low power and high performance.

4.1 Representation

An RNS represents a large integer using a set of smaller integers, with carry-free operations and a lack of ordered features among its residues, such that computation may be performed more efficiently. The carry-free property implies that the operations related to the different residues such as addition, subtraction or multiplication are independent from one residue to another. The lack of ordered features among residue digits implies that some residues can be used in fault tolerance of arithmetic operations

(SODERSTRAND et al., 1986), (PARHAMI, 2001).

In RNS, suppose we have a set of r different moduli, $\{m_1, m_2, \dots, m_r\}$, that are pairwise relatively prime to each other, i.e., $\gcd(m_i, m_j) = 1$, for $i \neq j$.

Let M be the product of the moduli set, which is called the dynamic range of the RNS, because the amount of numbers that can be represented is M . For unsigned numbers, that range is $[0, M-1]$, and, for the cases where we need to represent negative values, the range becomes $[-M/2, M/2 - 1]$ (OMONDI; PREMKUMAR, 2007).

This product is expressed as

$$M = \prod_{i=1}^r m_i. \quad (4.1)$$

An integer x is represented by an ordered set of r residues of positive integers, $\{X_1, X_2, \dots, X_r\}$, defined within the dynamic range.

Consider the following correspondence:

$$x \leftrightarrow \langle X_1, X_2, \dots, X_r \rangle, \quad (4.2)$$

where $x \in \mathbb{Z}_M$ and $X_i \in \mathbb{Z}_{m_i}$. The CRT makes the assertion that the mapping in equation (4.2) is a one-to-one correspondence between $\mathbb{Z}_M$ and the Cartesian product $\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \dots \times \mathbb{Z}_{m_r}$.

The number X_i is said to be the residue of x with respect to m_i , we often use the following notation:

$$X_i \equiv |x|_{m_i} = x \pmod{m_i}, \text{ for } i = 1, 2, \dots, r. \quad (4.3)$$

The advantages of RNS representation are that the standard arithmetic operations addition, subtraction, and multiplication can be performed by modular addition, subtraction, and multiplication for each RNS element in constant time on a parallel architecture (BAJARD; IMBERT, 2004).

If x and y are given in their RNS forms $x \leftrightarrow \langle X_1, X_2, \dots, X_r \rangle$ and $y \leftrightarrow \langle Y_1, Y_2, \dots, Y_r \rangle$, then we may define the operations of addition, subtraction and multiplication with the following equations:

$$\begin{aligned}
 |x + y|_M &\leftrightarrow \langle X_1, X_2, \dots, X_r \rangle + \langle Y_1, Y_2, \dots, Y_r \rangle \\
 &\leftrightarrow \langle Z_1, Z_2, \dots, Z_r \rangle, \text{ where } Z_i \equiv (X_i + Y_i) \bmod m_i \\
 &\leftrightarrow z
 \end{aligned} \tag{4.4}$$

$$\begin{aligned}
 |x - y|_M &\leftrightarrow \langle X_1, X_2, \dots, X_r \rangle - \langle Y_1, Y_2, \dots, Y_r \rangle \\
 &\leftrightarrow \langle Z_1, Z_2, \dots, Z_r \rangle, \text{ where } Z_i \equiv (X_i - Y_i) \bmod m_i \\
 &\leftrightarrow z
 \end{aligned} \tag{4.5}$$

$$\begin{aligned}
 |xy|_M &\leftrightarrow \langle X_1, X_2, \dots, X_r \rangle \cdot \langle Y_1, Y_2, \dots, Y_r \rangle \\
 &\leftrightarrow \langle Z_1, Z_2, \dots, Z_r \rangle, \text{ where } Z_i \equiv (X_i Y_i) \bmod m_i \\
 &\leftrightarrow z
 \end{aligned} \tag{4.6}$$

The operations performed on the elements $\mathbb{Z}_M$ can be equivalently performed on the corresponding r -tuples by performing the operations independently at each coordinate positions in the appropriate system (CORMEN et al., 2009). Hence, to add, subtract or multiply two RNS numbers, we only need to add, subtract or multiply the corresponding value pairs X_i and Y_i , which reduces the length of the carry chain and the latency of the arithmetic operation.

However, the comparison and division operations are very difficult to perform, and the overflows that may occur during the calculations are not easily detected on the RNS representation (HUNG; PARHAMI, 1994), (HITZ; KALTOFEN, 1995), (BAJARD; IMBERT, 2004).

From the viewpoint of this thesis, such difficulties are not considered true draw-

backs. Most of the PKC algorithms perform calculations in a finite field or ring, which eliminates the problem of overflow. In addition, such algorithms do not require divisions and comparisons because they achieve modular reduction based on the arithmetic operations of addition, subtraction and multiplication to perform modular exponentiation, and the reduction can be computed efficiently without division using Montgomery Multiplication.

4.2 Basic Structure

A generic structure of a regular RNS processor is shown in the following figure:

Figure 2: Basic Structure of an RNS Processor

The input operands x and y must first be converted from conventional notation to the RNS representation, with r -residue X_i and Y_i by a process called *Forward Conversion*. Then, the RNS represented input operands are processed in parallel with no dependence among the arithmetic operations by each *Modulo m_i* in *Modulo Channels*. The RNS represented intermediate results Z_i , produced by each *Modulo m_i* , are converted to the output results z in the conventional notation input by a process called *Reverse Conversion*. The *Reverse Conversion* process is based on the Chinese Remainder Theorem (CRT) or Mixed-Radix Conversion (MRC). The utilization of the CRT

allows parallelism while the MRC is an intrinsically sequential approach, in the reverse conversion hardware implementation, which overall is complex and costly (OMONDI; PREMKUMAR, 2007).

4.3 Moduli Selection

The choice of the moduli set is one of the most critical considerations and the greatest challenge for RNS hardware design because the moduli selection affects the RNS representation efficiency, the complexity of the forward and reverse converters and the RNS arithmetic circuits.

Some types of moduli sets have been implemented to simplify the modular reduction by using special modulus forms. A large number of different moduli sets have been proposed, such as the special forms to calculate the modular residue by breaking the input operands into its words and then adding them in various combinations (ABDALLAH; SKAVANTZOS, 1995), (BAJARD; DIDIER; KORNERUP, 2001), (HOSSEINZADEH; NAVI; GORGIN, 2007), (CAO; CHANG; SRIKANTHAN, 2007), (BAJARD; KAIHARA; PLANTARD, 2009), (ASKARZADEH; HOSSEINZADEH; NAVI, 2009). The following examples illustrate some types of moduli sets:

$$\{2^n - 1, 2^n + 1\}$$

$$\{2^n - 1, 2^n, 2^n + 1\}$$

$$\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$$

$$\{2^{n-1}, 2^{n-1} - 1, 2^n - 1, 2^n + 1\}$$

$$\{2^n, 2^n - 1, 2^{n-1} - 1, 2^{n-1} + 1\}$$

$$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$$

The generalized Mersenne form, $2^{k_1} - 2^{k_2} - 1$, offers the possibility of choosing bases in a smaller range and retains the efficiency of the modulo reduction (SOLINAS, 1999), (CIET et al., 2003). The form $\{2^n \pm 1\}$ is usually referred to as low-cost moduli because conversion to and from their residues can be made to be relatively easy to implement and does not require complex operations (ZIMMERMANN, 1999), (OMONDI; PREMKUMAR, 2007).

4.4 Conversion from Binary to RNS Representation

The forward conversion is the process of encoding the input operands from conventional notation, herein binary, into residue notation. The following is a review of the special moduli-set, $\{2^n - 1, 2^n, 2^n + 1\}$, which simplifies the forward conversion algorithm and architecture (SZABO; TANAKA, 1967), (OMONDI; PREMKUMAR, 2007), though these modules are not appropriate for PKC because one cannot choose an arbitrary dynamic range.

Let x be an arbitrary input operand with n bits to convert from binary to RNS representation:

$$x = \sum_{i=0}^{n-1} x_i 2^i \quad (4.7)$$

To compute the residue of x with respect to a modulus M , all that is required is the analysis of the values $|2^i|_M$ as shown in the following equation:

$$|x|_M \equiv \left| \sum_{i=0}^{n-1} x_i 2^i \right|_M = \left| \sum_{i=0}^{n-1} |x_i 2^i|_M \right|_M, \quad (4.8)$$

where x_i is either 0 or 1.

4.4.1 Modulus 2^n

Consider the basic special moduli-set 2^n . The residue of x with respect to this modulus is an easy operation by dividing x by 2^n . Thus, the result of $|x|_{2^n}$ is the n least

significant bits of the binary representation of x .

4.4.2 Modulus $2^n - 1$

The computation of the residues with respect to modulus $2^n - 1$ is quite easy to implement, and the residues are determined based on the following equations:

$$|2^n|_{2^n-1} \equiv |2^n - 1 + 1|_{2^n-1} = 1, \quad (4.9)$$

where $n > 1$. Equation (4.9) can be extended to compute $|2^{q \cdot n}|_{2^n-1}$ as follows:

$$|2^{q \cdot n}|_{2^n-1} \equiv \left| \prod_{i=1}^q |2^n|_{2^n-1} \right|_{2^n-1} = 1, \quad (4.10)$$

where q is an integer.

Therefore, the residue of any number 2^m , for $m \neq n$, with respect to $2^n - 1$, can be determined by using equations (4.9) and (4.10) as follows:

$$|2^m|_{2^n-1} \equiv |2^{qn+r}|_{2^n-1} = ||2^{qn}|_{2^n-1} \times |2^r|_{2^n-1}|_{2^n-1} = |2^r|_{2^n-1}, \quad (4.11)$$

where $q = \left\lfloor \frac{m}{n} \right\rfloor$ and $r \equiv m \pmod{n}$.

4.4.3 Modulus $2^n + 1$

As in the previous case, the residue of any number 2^n , with respect to modulus $2^n + 1$ can be obtained as follows:

$$|2^n|_{2^n+1} \equiv |2^n + 1 - 1|_{2^n+1} = -1 \quad (4.12)$$

Furthermore, equation (4.12) can be extended to an arbitrary power of two, 2^m ,

where $m \neq n$ and $m = qn + r$, to compute the following:

$$\begin{aligned} |2^{qn+r}|_{2^{n+1}} &\equiv |2^{qn}|_{2^{n+1}} \times |2^r|_{2^{n+1}} \\ &= \begin{cases} 2^n, & \text{if } q \text{ is even} \\ 2^n + 1 - 2^r, & \text{otherwise,} \end{cases} \end{aligned} \quad (4.13)$$

where $q = \left\lceil \frac{m}{n} \right\rceil$. Moreover, when q is odd, $|2^{qn}|_{2^{n+1}} = -1$, so it is required to make the residue positive by adding $2^n + 1$.

4.4.4 Special Moduli-Set $\{2^n - 1, 2^n, 2^n + 1\}$

Algorithm 3 describes a general procedure to convert a given binary number x , with $3n$ bits, to RNS representation with respect to the special moduli-set $\{2^n - 1, 2^n, 2^n + 1\}$ (BI; JONES, 1988), (VINNAKOTA; RAO, 1994).

Assume that we have a set of moduli, $m_1 = 2^n + 1$, $m_2 = 2^n$, $m_3 = 2^n - 1$, such that $M = m_1 m_2 m_3$ and x is within the dynamic range, $M = [0, 2^{3n} - 2^n - 1]$, which is uniquely defined by a residue-set $\{X_1, X_2, X_3\}$, where $X_i \equiv |x|_{m_i}$.

Algorithm 3 Forward Conversion for the Special Moduli-Set $\{2^n - 1, 2^n, 2^n + 1\}$

Require: odd M , $3n = 1 + \lfloor \log_2 M \rfloor$, $x = \sum_{i=0}^{3n-1} x_i 2^i$, with $0 \leq x < M$,

$M = m_1 m_2 m_3$, $m_1 = 2^n + 1$, $m_2 = 2^n$, and $m_3 = 2^n - 1$

Ensure: $X_1 \equiv |x|_{2^{n+1}}$, $X_2 \equiv |x|_{2^n}$, and $X_3 \equiv |x|_{2^n-1}$

- 1: $B_1 \leftarrow \sum_{i=2n}^{3n-1} x_i 2^{i-2n}$
 - 2: $B_2 \leftarrow \sum_{i=n}^{2n-1} x_i 2^{i-n}$
 - 3: $B_3 \leftarrow \sum_{i=0}^{n-1} x_i 2^i$
 - 4: $X_1 \leftarrow |B_1 - B_2 + B_3|_{2^{n+1}}$
 - 5: $X_2 \leftarrow B_3$
 - 6: $X_3 \leftarrow |B_1 + B_2 + B_3|_{2^n-1}$
 - 7: **return** X_1 , X_2 , and X_3
-

In lines 1 to 3, an input operand x is split into three n -bit blocks, B_1 , B_2 , and B_3 , as follows:

$$x = B_1 2^{2n} + B_2 2^n + B_3 \quad (4.14)$$

The residue of X_1 (line 4) is obtained as

$$\begin{aligned}
 X_1 &\equiv |x|_{2^{n+1}} \\
 &= |B_1 2^{2n} + B_2 2^n + B_3|_{2^{n+1}} \\
 &= |B_1 2^{2n}|_{2^{n+1}} + |B_2 2^n|_{2^{n+1}} + |B_3|_{2^{n+1}} \quad (4.15)
 \end{aligned}$$

In equation (4.15), the residues of the three sums are simplified to

$$|B_1 2^{2n}|_{2^{n+1}} \equiv |B_1|_{2^{n+1}} \times |2^{2n}|_{2^{n+1}} = B_1 \quad (4.16)$$

$$|B_2 2^n|_{2^{n+1}} \equiv |B_2|_{2^{n+1}} \times |2^n|_{2^{n+1}} = -B_2 \quad (4.17)$$

$$|B_3|_{2^{n+1}} \equiv B_3 \quad (4.18)$$

because B_1 , B_2 , and B_3 are represented in n bits, and thus they are always less than $2^n + 1$. Furthermore, based on equation (4.12), the residues of $|2^{2n}|_{2^{n+1}}$ and $|2^n|_{2^{n+1}}$ are respectively 1 and -1 .

Therefore, we have

$$X_1 \equiv |B_1 - B_2 + B_3|_{2^{n+1}} \quad (4.19)$$

The residue of X_2 (line 5) is the remainder when x is divided by 2^n . Thus, X_2 is the number represented by the least significant n bits of x , i.e., B_3 , as shown in the following equation:

$$\begin{aligned}
 X_2 &\equiv |x|_{2^n} \\
 &= |B_1 2^{2n} + B_2 2^n + B_3|_{2^n} \\
 &= B_3 \quad (4.20)
 \end{aligned}$$

The residue of X_3 (line 6) is then computed as follows:

$$\begin{aligned}
 X_3 &\equiv |x|_{2^n-1} \\
 &= |B_1 2^{2n} + B_2 2^n + B_3|_{2^n-1} \\
 &= |B_1 2^{2n}|_{2^n-1} + |B_2 2^n|_{2^n-1} + |B_3|_{2^n-1}
 \end{aligned} \tag{4.21}$$

As described above, the residues of the three sums in equation (4.21) are obtained in the following equations:

$$|B_1 2^{2n}|_{2^n-1} \equiv |B_1|_{2^n-1} \times |2^{2n}|_{2^n-1} = B_1 \tag{4.22}$$

$$|B_2 2^n|_{2^n-1} \equiv |B_2|_{2^n-1} \times |2^n|_{2^n-1} = B_2 \tag{4.23}$$

$$|B_3|_{2^n-1} \equiv B_3 \tag{4.24}$$

Therefore, we have

$$X_3 \equiv |B_1 + B_2 + B_3|_{2^n-1} \tag{4.25}$$

Forward conversions in RNS have been traditionally implemented and based on special or arbitrary moduli-sets, combinational-logic converts for arbitrary moduli-sets, and so forth. The various techniques can be fruitfully combined to take advantage of their optimized hardware in terms of power, area and speed (OMONDI; PREMKUMAR, 2007).

4.5 Conversion from RNS to Binary Representation

The reverse conversion is the method, usually after some residue-arithmetic operations, of decoding from the RNS represented input operands to the output results of the RNS processor in the conventional notation input. The algorithms for reverse conversion are based on either CRT or MRC. All other methods may be viewed as variants of these methods (SZABO; TANAKA, 1967), (CAO; CHANG; SRIKANTHAN,

2007), (MOHAN; PREMKUMAR, 2007), (OMONDI; PREMKUMAR, 2007).

4.5.1 Chinese Remainder Theorem

The CRT is the basic theorem in RNS, and it ensures the uniqueness of this representation within the range $0 \leq z < M$. The proof of this theorem (CORMEN et al., 2009) can be used to convert z back from its residue. The relationship between z and its residues is shown in the following equation:

$$z \equiv \left(\sum_{i=0}^r Z_i M_i \left| M_i^{-1} \right|_{m_i} \right) \bmod M, \quad (4.26)$$

where $M_i = \frac{M}{m_i}$ and $\left| M_i^{-1} \right|_{m_i}$ are the inverse of M_i modulo m_i , i.e., $(M_i^{-1} M_i) \bmod m_i \equiv 1$.

Algorithm 4 is an efficient method for determining z , given an RNS representation $z \leftrightarrow \langle Z_1, Z_2, \dots, Z_r \rangle$, the residues of z modulo the pairwise co-prime moduli $m_1, m_2, \dots, m_r$ (GARNER, 1959), (MENEZES; OORSCHOT; VANSTONE, 1996).

Algorithm 4 Garner's Algorithm for CRT (GCRT)

Require: a positive integer $M = \prod_{i=1}^r m_i > 1$, with $\gcd(m_i, m_j) = 1$, for all $i \neq j$,
and an RNS representation $z \leftrightarrow \langle Z_1, Z_2, \dots, Z_r \rangle$ of z for m_i

Ensure: the integer z in the conventional notation

```

1: for  $i \leftarrow 2$  to  $r$  step 1 do
2:    $C_i \leftarrow 1$ 
3:   for  $j \leftarrow 1$  to  $(i - 1)$  step 1 do
4:      $u \leftarrow m_j^{-1} \pmod{m_i}$ 
5:      $C_i \leftarrow u C_i \pmod{m_i}$ 
6:   end for
7: end for
8:  $z \leftarrow Z_1$ 
9: for  $i \leftarrow 2$  to  $r$  step 1 do
10:   $u \leftarrow (Z_i - z) C_i \pmod{m_i}$ 
11:   $z \leftarrow z + u \prod_{j=1}^{i-1} m_j$ 
12: end for
13: return  $z$ 
```

4.5.2 Mixed Radix Conversion

The MRC establishes a one-to-one relationship between the RNS representation and a weighted and positional mixed-radix system. In such a system, it is necessary to enforce the restriction that the maximum weight contributed by the lower k digits should never exceed the weight of the first $(k + 1)$ positional digits (OMONDI; PREMKUMAR, 2007).

In MRC, suppose we have the same RNS set of pair-wise relatively prime moduli $\{m_1, m_2, \dots, m_r\}$. If the radices are $r_1, r_2, \dots, r_r$, any number z can be uniquely expressed in MRC representation in the following form:

$$z \cong \langle z_1, z_2, \dots, z_r \rangle \quad (4.27)$$

The interpretation of this representation is shown by the following equation:

$$z \equiv z_r \prod_{i=1}^r r_i + \dots + z_3 r_2 r_1 + z_2 r_1 + z_1, \quad (4.28)$$

where $0 \leq z_i < r_i$, and it ensures a unique representation.

The connection between an MRC representation and an RNS representation with respect to moduli $\{m_1, m_2, \dots, m_r\}$ is found by matching $r_i = m_i$ as follows:

$$z \equiv z_r \prod_{i=1}^r m_i + \dots + z_3 m_2 m_1 + z_2 m_1 + z_1, \quad (4.29)$$

The conversion from the RNS representation to the MRC representation may be viewed as a reverse transformation because the MRC is weighted (OMONDI; PREMKUMAR, 2007).

Given the above reasoning, a method for determining the value of z from RNS representation, $\langle Z_1, Z_2, \dots, Z_r \rangle$, is the following steps for converting the residues Z_i into the MRC representation associated with $z \cong \langle z_1, z_2, \dots, z_r \rangle$, and then the other steps of reverting the digits z_i to the conventional equivalent, z , follow.

The following equations show how to obtain the digits z_i :

$$z_1 \equiv |z|_{m_1} = Z_1 \quad (4.30)$$

$$z_2 \equiv \left| m_1^{-1} \right|_{m_2} (Z_2 - z_1) \Big|_{m_2} \quad (4.31)$$

$$z_3 \equiv \left| m_2^{-1} \right|_{m_3} \left(\left| m_1^{-1} \right|_{m_3} (Z_3 - z_1) - z_2 \right) \Big|_{m_3} \quad (4.32)$$

...

$$z_r \equiv \left| m_{r-1}^{-1} \right|_{m_r} \left(\left| m_{r-2}^{-1} \right|_{m_r} \left(\cdots \left| m_2^{-1} \right|_{m_r} \left(\left| m_{r-1}^{-1} \right|_{m_r} (Z_r - z_1) - z_2 \right) \cdots \right) - z_{r-1} \right) \Big|_{m_r} \quad (4.33)$$

Because we have the equation (4.29), we can apply Horner's algorithm to compute z (HUANG, 1983), (KOC, 1989), (AKKAL; SIY, 2007).

4.6 Montgomery Multiplication in RNS

The use of RNS is well-known in PKC (KAWAMURA et al., 2000), (NOZAKI et al., 2001), (CIET et al., 2003), (BAJARD; DIDIER; KORNERUP, 2001), (BAJARD; IMBERT, 2004), (BAJARD; MELONI; PLANTARD, 2005), (BAJARD et al., 2006), (LIM; PHILLIPS, 2007), (SCHINIANAKIS et al., 2009).

The main advantage of cryptography in RNS is the reason that additions and multiplications for the arithmetic functions, such as modular exponentiation and modular multiplication, are performed independently on the residues. In a parallel architecture with r arithmetic units (*Modulo* m_i), the time needed to perform an addition or a multiplication is bounded by one modular operation on the largest residue (BAJARD et al., 2006).

However, the use of the RNS to RSA cryptosystem handles a limited situation because one cannot choose an arbitrary dynamic range (the product of the moduli set) because one has to choose distinct secret primes p and q to calculate the modulus M as the RNS product of the moduli set (KAWAMURA et al., 2000). Therefore, some surveys consider methods where the RNS dynamic range can be chosen almost independently

of secret primes p and q . The PKCS #1 v2.0 amendment (RSA LABORATORIES, 2000) presents the Multi-Prime RSA scheme where the modulus may have more than two prime factors. Only private-key operations and representations are affected. An efficient RNS modular reduction using base extensions was proposed in (BAJARD; DIDIER; KORNERUP, 2001), where this application is an adaptation in RNS of the Montgomery Multiplication algorithm.

In this thesis, our investigations were conducted to provide an application of the Montgomery Multiplication in RNS for computing $z \equiv x^e \pmod{M}$, using the Montgomery Exponentiation algorithm, which was introduced in Section 3.4, with no change or addition to or other modification for performance in RNS.

The following proof of concept shows how to operate in parallel modular multiplication repeatedly in the Montgomery r -domain through the use of RNS, where each domain is based on $R_i = 2^{n_i}$, $n_i = 1 + \lfloor \log_2 m_i \rfloor$, and R_i^{-1} is the multiplicative inverse of R_i , such that $\gcd(m_i, R_i) = 1$ and $(R_i R_i^{-1}) \equiv 1 \pmod{m_i}$.

Let x be an arbitrary input operand with n bits in their RNS form $x \leftrightarrow \langle X_1, X_2, \dots, X_r \rangle$, and let e be an exponent, which is given here in the most significant bit form, such that $e = \sum_{i=0}^{t-1} e_i 2^i$, with $e_t = 1$.

The proposal is the reconstruction of the r -residue X_i from the RNS representation to the Montgomery r -domain. Then, in each *Modulo* m_i , the Montgomery Exponentiation is performed. The major steps of the scheme are as follows:

1. *Conversion from RNS to Montgomery Domain:* Compute the transformation of each residue, X_i from RNS representation to the Montgomery domain, $\tilde{X}_i$, by computing $\tilde{X}_i \equiv X_i R_i \pmod{m_i}$.
2. *Modulo m_i :* Calculate the intermediate modular exponentiation results, $\tilde{Z}_i$, in each Montgomery r -domain using the MM algorithm repeatedly, where $\tilde{Z}_i \equiv (\tilde{X}_i)^e \pmod{m_i}$.

3. *Conversion from Montgomery Domain to RNS Representation*: Then, convert the results of each modular exponentiation, $\tilde{Z}_i$, back to the RNS representation by performing $Z_i \equiv (\tilde{Z}_i R_i^{-1}) \pmod{m_i}$.

Algorithm 5 shows the pseudo code of the proposed Montgomery Exponentiation in RNS. Following Figure 2, this algorithm involves the *Forward Conversion*, *Module* M_i , and *Reverse Conversion* processes. First, a generic procedure converts the input operand, x , from conventional notation into r -residue notation, X_i . Then, Algorithm 2 (MEXP) performs the modular exponentiation by each *Modulo* M_i in *Modulo Channels*. Finally, based on equation (4.26) a generic CRT procedure converts z back from its residue.

Algorithm 5 The proposed Montgomery Exponentiation in RNS

Require: a positive integer $M = \prod_{i=1}^r m_i > 1$, with $\gcd(m_i, m_j) = 1$, for all $i \neq j$,
 $M_i = \frac{M}{m_i}$ and $|M_i^{-1}|_{m_i}$ is the inverse of M_i modulo m_i , i.e., $(M_i^{-1} M_i) \pmod{m_i} \equiv 1$,
 $n = 1 + \lfloor \log_2 M \rfloor$, $x = \sum_{i=0}^{n-1} x_i 2^i$, $e = \sum_{i=0}^{t-1} e_i 2^i$, with $e_t = 1$, and $1 \leq x < M$,
 $n_i = 1 + \lfloor \log_2 m_i \rfloor$, $R_i = 2^{n_i}$, $\gcd(m_i, R_i) = 1$, $(R_i R_i^{-1}) \equiv 1 \pmod{m_i}$, and
 $R_i^2 \equiv R_i R_i \pmod{m_i}$ as precomputed values.

Ensure: $z \equiv x^e \pmod{M}$

- 1: **for** $i \leftarrow 1$ **to** r **step** 1 **do**
- 2: $X_i \leftarrow x \pmod{m_i}$
- 3: **end for**
- 4: **for each** *Modulo* m_i , **in parallel do**
- 5: $Z_i \leftarrow \text{MEXP}(X_i, e, R_i^2, m_i)$
- 6: **end for**
- 7: $z \leftarrow 0$
- 8: **for** $i \leftarrow 1$ **to** r **step** 1 **do**
- 9: $z \leftarrow z + Z_i M_i |M_i^{-1}|_{m_i} \pmod{M}$
- 10: **end for**
- 11: **return** z

This algorithm involves the following steps:

1. *Forward conversion*: Lines 1 to 3 perform the conversion of the input operand x to the RNS representation of the r -residue X_i , with respect to m_i , as denoted in equation (4.3).

2. *Modulo m_i* : Lines 4 to 6 process the r algorithms MEXP in parallel, which computes their $Z_i \equiv (X_i)^e \bmod m_i$ outputs.
3. *Reverse conversion*: Line 7 converts the intermediate results Z_i , produced by each modular exponentiation, from the RNS representation to the output result $z \equiv x^e \pmod{M}$ in the conventional notation of the input x using the Garner's algorithm for CRT (Algorithm 4).

5 HARDWARE ALGORITHMS FOR LOW-POWER MODULAR MULTIPLICATION

This chapter considers the MM algorithms for low-power hardware implementations. We review the proposed k -Partition Montgomery Multiplication method (NÉTO; TENCA; RUGGIERO, 2011) and the Montgomery Multiplication in RNS method proposed in this thesis.

5.1 k -Partition Method

The proposed partitioning method for Montgomery Multiplication is called the k -Partition MM Method (k PMM). Partitions have basically the same structure but handle different bits of the operands involved in the multiplication. For this reason we give them the generic name of MMP (Montgomery Multiplication Partition). Partitions can run serially or in parallel. In this work, we focus on a fully parallel implementation.

5.1.1 Montgomery Multiplication Partition (MMP)

Algorithm 6 shows a generic pseudo code for the algorithm executed by the j^{th} MMP. This algorithm was inspired by the High-Radix (Radix- 2^k) Montgomery Multiplication ($R2^k$ MM) algorithm, which was described and proved in (TODOROV, 2000). The main difference is the computation of multiples of Y at line 3, where each partition j only checks if bit $x_{j+ik} = 1$ (it does not check all k bits).

In the traditional radix- r MM algorithm, k bits of X are scanned in each clock

Algorithm 6 The proposed Montgomery Multiplication Partition j (MMP)

Require: odd M , $n = 1 + \lfloor \log_2 M \rfloor$, $X = \sum_{i=0}^{n-1} x_i 2^i$, $Y = \sum_{i=0}^{n-1} y_i 2^i$,
 with $0 \leq X, Y < M$, j^{th} -partition, $0 < k, t < n$, $kt = n$, k partitions
Ensure: $Z_{P_j} = \text{MMP}(j, X, Y, M) \equiv (X_{P_j} Y R^{-1}) \bmod M$, with $0 \leq Z_{P_j} < M$,
 where $X_{P_j} = \sum_{i=0}^{n/k-1} x_{j+ik} 2^{j+ik}$

- 1: $S_{P_j}[0] \leftarrow 0$
- 2: **for** $i \leftarrow 0$ **to** $n/k - 1$ **step 1 do**
- 3: $a \leftarrow S_{P_j}[i] + x_{j+ik} 2^j Y$
- 4: $q_{k-1..0} \leftarrow a_{k-1..0} (2^k - M^{-1}_{k-1..0}) \bmod 2^k$
- 5: $S_{P_j}[i+1] \leftarrow (a + q_{k-1..0} M) / 2^k$
- 6: **end for**
- 7: **return** $Z_{P_j} \leftarrow S_{P_j}[n/k]$

cycle, where the radix of the representation is $r = 2^k$, with k an integer such that $0 < k < n$. If we want to scan k bits of X in each iteration, we would need to (1) encode the digits of X , which would require more hardware for recoding, (2) generate more complex partial products than a binary multiplier, and (3) create a more elaborate logic to select the proper adder inputs for the accumulation of partial products and multiples of the modulus M . All of these tasks would increase the complexity of the overall hardware.

This proposal simplifies the computation of multiples of Y by distributing the bits of X among k partitions that can be processed separately using a radix- 2^k multiplication.

The k new decomposed multiplier operands are represented by X_{P_j} and are related to the original X as follows:

$$X = \sum_{i=0}^{n-1} x_i 2^i = \sum_{j=0}^{k-1} X_{P_j}, \quad \text{with} \quad (5.1)$$

$$X_{P_j} = \sum_{i=0}^{n/k-1} x_{j+ik} 2^{j+ik}. \quad (5.2)$$

Given that each radix- 2^k digit of X_{P_j} is in the set $\{0, x_i 2^i\}$ with $0 \leq i < k$, the computation of multiples of Y is significantly reduced.

The final output of the j^{th} MMP is a partial modular multiplication, which is

represented by Z_{P_j} as follows:

$$Z_{P_j} \equiv (X_{P_j} Y R^{-1}) \pmod{M}. \quad (5.3)$$

Based on the above definition of X_{P_j} , Figure 3 shows the distribution of bits of X into $k = 2$ decomposed multiplier operands: X_{P_0} and X_{P_1} .

Figure 3: The distribution of bits of X into two decomposed multiplier operands

We want to show that the following equation holds at each iteration of the proposed algorithm:

$$S_{P_j}[i+1] \equiv \left(\sum_{l=0}^i x_{j+lk} 2^{j+lk} \right) Y 2^{-(i+1)k} \pmod{M}, \quad (5.4)$$

for all i , such that $0 < k, t < n, kt = n, 0 \leq i < n/k$, and $0 \leq j < k$.

The property stated by equation (5.4) is proved by induction.

In the first iteration, $i = 0$ and $S_{P_j}[0] = 0$. Thus, equation (5.4) holds for iteration 1 because

$$\begin{aligned} a &= S_{P_j}[0] + x_j 2^j Y = x_j 2^j Y, \\ q_{k-1..0} &= a_{k-1..0} (2^k - M^{-1}_{k-1..0}) \pmod{2^k}, \text{ and} \\ S_{P_j}[1] &= \frac{a + q_{k-1..0} M}{2^k} \equiv x_j 2^j Y 2^{-k} \pmod{M}. \end{aligned}$$

Using the same arguments presented for equation (3.6), we observe that $q_{k-1..0} M \equiv a M M' \pmod{2^k} \equiv -a \pmod{2^k}$, and thus 2^k divides $q_{k-1..0} M$.

Assuming that the property holds for iteration $i - 1$, it can be shown that equation

(5.4) holds for iteration i as follows:

$$\begin{aligned}
a &= S_{P_j}[i] + x_{j+ik}2^jY, \text{ and} \\
S_{P_j}[i+1] &= \frac{a + q_{k-1..0}M}{2^k} \equiv (S_{P_j}[i] + x_{j+ik}2^jY)2^{-k} \pmod{M} \\
&\equiv (S_{P_j}[i]2^{ik} + x_{j+ik}2^{j+ik}Y)2^{-(i+1)k} \pmod{M} \\
&\equiv \left[\left(\sum_{l=0}^{i-1} x_{j+lk}2^{j+lk} \right) Y + x_{j+ik}2^{j+ik}Y \right] 2^{-(i+1)k} \pmod{M} \\
&\equiv \left(\sum_{l=0}^i x_{j+lk}2^{j+lk} \right) Y 2^{-(i+1)k} \pmod{M}.
\end{aligned}$$

In the last iteration, $i = n/k - 1$, and equation (5.4) leads to the following:

$$S_{P_j}[n/k] \equiv \left(\sum_{l=0}^{n/k-1} x_{j+lk}2^{j+lk} \right) Y 2^{-n} \pmod{M} \equiv X_{P_j} Y R^{-1} \pmod{M} \equiv Z_{P_j}.$$

The basic measure of time complexity is the number of clock cycles required to execute a complete multiplication. The inner loop (lines 2 to 6) runs for n/k clock cycles (one loop for each clock cycle). The running time of Algorithm 6 is therefore $T_{MMP_j}(n) = O(n/k)$.

5.1.2 The k -Partition Montgomery Multiplication (k PMM)

Algorithm 7 shows the pseudo code for the k -Partition MM Method (k PMM), which uses k identical hardware components (MMPs described in Algorithm 6). The number of partitions is not limited to a small number, as previous research indicates.

The partitioning scheme (lines 1 to 3 – Algorithm 7) shows a way to run k MMPs in parallel. This processing can be performed serially, but it requires $k - 1$ times longer. However, the addition of partial results ZP_j (lines 4 to 11) is proposed to run serially, which consumes $k - 1$ clock cycles. This calculation can be performed in parallel at the cost of more circuit area.

Algorithm 7 The proposed k -Partition Montgomery Multiplication ($kPMM$)

Require: odd M , $n = 1 + \lfloor \log_2 M \rfloor$, $X = \sum_{i=0}^{n-1} x_i 2^i$, $Y = \sum_{i=0}^{n-1} y_i 2^i$,
with $0 \leq X, Y < M$, j^{th} -partition, $0 < k < n$, k partitions

Ensure: $Z = kPMM(X, Y, M) \equiv (XYR^{-1}) \pmod{M}$, with $0 \leq Z < M$

```

1: for each partition  $j$ , in parallel do
2:    $Z_{P_j} \leftarrow \text{MMP}(j, X, Y, M)$ 
3: end for
4:  $Z \leftarrow 0$ 
5: for  $j \leftarrow 0$  to  $k - 1$  step 1 do
6:    $Z \leftarrow Z + Z_{P_j}$ 
7:   for  $l \leftarrow 0$  to  $1$  step 1 do
8:     if  $Z \geq M$  then
9:        $Z \leftarrow Z - M$ 
10:    end if
11:  end for
12: end for
13: return  $Z$ 

```

5.1.2.1 Correctness

The correctness of Algorithm 7 can be shown by using a divide-and-conquer approach which involves three steps.

Line 3, in Algorithm 6, divides the multiplier operand X into other multiplier operands X_{P_j} according to equation (5.1). A way to split the bits of X into k n -bit multiplier operands X_{P_j} is represented by equation (5.2).

Line 2, in Algorithm 7, performs a radix- 2^k modular multiplication in each partition. The modular multiplication of X_{P_j} and Y is performed using MMP as stated by equation (5.3).

All k partial multiplications can be performed independently of each other by applying the generic Algorithm 6.

Lines 5 to 11, in Algorithm 7, combine the solutions of the partial modular multi-

plications to generate the solution of the complete modular multiplication as follows:

$$\begin{aligned}
 Z &= \sum_{j=0}^{k-1} MMP(j, X, Y, M) \equiv \sum_{j=0}^{k-1} Z_{P_j} \pmod{M} \\
 &\equiv \left(\sum_{j=0}^{k-1} X_{P_j} \right) YR^{-1} \pmod{M} \\
 &\equiv \left(\sum_{j=0}^{k-1} \sum_{i=0}^{n/k-1} x_{j+ik} 2^{j+i} \right) YR^{-1} \pmod{M} \\
 &\equiv \left(\sum_{j=0}^{n-1} x_j 2^j \right) YR^{-1} \pmod{M} \equiv (XYR^{-1}) \pmod{M}. \tag{5.5}
 \end{aligned}$$

5.1.2.2 Adding partial results Z_{P_j}

Because k is usually a small integer, especially when compared to n (the size of the operands in the modular multiplication), the addition of Z_{P_j} will not require many resources when executed serially and can even be executed in a general-purpose processor that has the proposed k PMM hardware as a co-processor. If the addition and reduction of the values Z_{P_j} are desirable in the proposed hardware, it can be made using the following steps: (1) Check if the Z_{P_j} value in carry-save (CS) form is smaller than M . If so, do not take this result for reduction. This choice is easy because the value Z_{P_j} is always less than $2M$, as shown in (MONTGOMERY, 1985). However, if the value Z_{P_j} is greater than or equal to M , then select this j^{th} partial result for reduction. (2) Sum the k values Z_{P_j} in CS form for each reduction value, if the related Z_{P_j} value has been found to be reduced (i.e., also adds $-M$ to the result) to produce the final result Z_P in binary form. (3) Use classical reduction of Z_P (modulo M) to obtain the final result.

5.1.2.3 Asymptotic Analysis of Algorithm 7

The running time of each MM Partition is $T_{MMP_j}(n) = O(n/k)$. The k partitions running in parallel (lines 1 to 3 – Algorithm 7) compute their CS outputs in $T_{MMP_j}(n)$.

Lines 5 to 10 (Algorithm 7) add those CS outputs in $T_{AdderCS}(k) = O(k)$, for small values of k and such that $k \ll n$. The running time of Algorithm 7 is obtained as $T_{kPMM}(n) = O(n/k)$.

Table 1: Running in parallel 2PMM Algorithm

<i>Partition₁</i>	<i>Partition₀</i>
$Z_{P_1} = 2PMM(X_{P_1}, Y, M)$, with $j = 1$.	$Z_{P_0} = 2PMM(X_{P_0}, Y, M)$, with $j = 0$.
$S[0] = 0$	$S[0] = 0$
$S[2] = S[0] + x_1 2^1 Y = 0$ $q_{1..0} = S[2]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 0$ $S[2] = (S[2] + q_{1..0}M)/2^2$ $= (0 + 0 \cdot 239)/2^2 = 0$	$S[2] = S[0] + x_0 2^0 Y = 189$ $q_{1..0} = S[2]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 1$ $S[2] = (S[2] + q_{1..0}M)/2^2$ $= (189 + 1 \cdot 239)/2^2 = 107$
$S[4] = S[2] + x_3 2^1 Y = 378$ $q_{1..0} = S[4]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 2$ $S[4] = (S[4] + q_{1..0}M)/2^2$ $= (378 + 2 \cdot 239)/2^2 = 214$	$S[4] = S[2] + x_2 2^0 Y = 107$ $q_{1..0} = S[4]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 3$ $S[4] = (S[4] + q_{1..0}M)/2^2$ $= (107 + 3 \cdot 239)/2^2 = 206$
$S[6] = S[4] + x_5 2^1 Y = 214$ $q_{1..0} = S[6]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 0$ $S[6] = (S[6] + q_{1..0}M)/2^2$ $= (214 + 2 \cdot 239)/2^2 = 173$	$S[6] = S[4] + x_4 2^0 Y = 395$ $q_{1..0} = S[6]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 3$ $S[6] = (S[6] + q_{1..0}M)/2^2$ $= (395 + 3 \cdot 239)/2^2 = 278$
$S[8] = S[6] + x_7 2^1 Y = 551$ $q_{1..0} = S[8]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 3$ $S[8] = (S[8] + q_{1..0}M)/2^2$ $= (551 + 3 \cdot 239)/2^2 = 317$	$S[8] = S[6] + x_6 2^0 Y = 467$ $q_{1..0} = S[8]_{1..0}(2^2 - M_{1..0}^{-1}) \bmod 2^2 = 3$ $S[8] = (S[6] + q_{1..0}M)/2^2$ $= (467 + 3 \cdot 239)/2^2 = 296$
$S[8] \geq M$, then $Z_{P_1} = S[8] - M = 78$.	$S[8] \geq M$, then $Z_{P_0} = S[8] - M = 57$.

5.1.2.4 Numerical Example

To illustrate the $kPMM$ method, consider two partitions ($k = 2$) using the following variables: $n = 8$, $M = 239$, $X = 217$, $Y = 189$, and $R = 2^8$.

The MM of X and Y in this case is 135 or simply $MM(217, 189) = 135$. First, the multiplier operand bits are distributed into two partitions, as shown in Figure 4. The

parallel execution of 2PMM uses $n/2$ steps (in this case, 4 steps) to produce the partial products Z_{P_1} and Z_{P_0} .

Figure 4: The distribution of bits of X into two multiplier operands

Table 1 shows the values of internal variables during the algorithm execution. The final result is obtained as $Z \equiv (Z_{P_1} + Z_{P_0}) \bmod M = (78 + 57) \bmod M \equiv 135 = \text{MM}(217, 189)$.

5.2 Montgomery Multiplication in RNS

The proposed method for Montgomery Multiplication in RNS is called the Montgomery Exponentiation in RNS (MEXPRNS). The scope herein is limited to the establishment of a shared cryptographic key (symmetric) an imbalanced network (ZHU et al., 2002), where two sets of entities, namely a set of powerful servers and a set of low-power mobile devices, employ the key-establishment scheme in RSA cryptosystem. We focus on the low-power mobile devices side, where the scheme of key generation consists of a huge amount of modular exponentiation, as described in Subsection 1.2.2. We adopted the modulus $M = pq$ as the RNS dynamic range in this proof of concept for the proposed method, though the range can be enhanced for "multi-prime" as a generalization of the standard RSA scheme (RSA LABORATORIES, 2000).

5.2.1 Montgomery Exponentiation in RNS (MEXPRNS)

Algorithm 8 shows a generic pseudo code for computing $z \equiv x^e \pmod{M}$, which is based on the properties of RNS as explained in Section 4.6 (Algorithm 5), herein using Algorithms 1 and 2.

Algorithm 8 The proposed Montgomery Exponentiation in RNS (MEXPRNS)

Require: odd $M = pq > 1$, where $\gcd(p, q) = 1$, $n = 1 + \lfloor \log_2 M \rfloor$,
 $x = \sum_{k=0}^{n_x-1} x_k 2^k$, $e = \sum_{i=0}^{t-1} e_i 2^i$, with $e_i = 1$, $1 \leq x < M$, $n_x = 1 + \lfloor \log_2 x \rfloor$,
 $R = 2^n$, $\gcd(MR) = 1$, $RR^{-1} \equiv 1 \pmod{M}$, $R_x = 2^{n_x}$,
for $p = m_p$, $q = m_q$, and $i = p, q$, $\gcd(m_i R_x) = 1$, $R_x R_{x_i}^{-1} \equiv 1 \pmod{m_i}$,
 $n_i = 1 + \lfloor \log_2 m_i \rfloor$, $R_i = 2^{n_i}$, $\gcd(p, R_i) = 1$, and the precomputed values:
 $R^2 \equiv RR \pmod{M}$, $R_{x_i}^2 \equiv R_x R_x \pmod{m_i}$, $R_i^2 \equiv R_i R_i \pmod{m_i}$,
 $M_i = M/m_i$, $(M_i^{-1} M_i) \pmod{m_i} \equiv 1$, and $M_{R_i}^{-1} \equiv M_i^{-1} R_i \pmod{m_i}$.

Ensure: $z \equiv x^e \pmod{M}$

- 1: **for each** m_i , $i = p, q$, **in parallel do**
- 2: $W_i \leftarrow \text{MM}(x, 1, m_i)$
- 3: $X_i \leftarrow \text{MM}(W_i, R_{x_i}^2, m_i)$
- 4: **end for**
- 5: **for each Modulo** m_i , $i = p, q$, **in parallel do**
- 6: $Z_i \leftarrow \text{MEXP}(X_i, e, R_i^2, m_i)$
- 7: **end for**
- 8: **for each Modulo** m_i , $i = p, q$, **in parallel do**
- 9: $Zr_i \leftarrow \text{MM}(Z_i, R_i^2, m_i)$
- 10: $Pa_i \leftarrow \text{MM}(Zr_i, M_{R_i}^{-1}, m_i)$
- 11: $Pb_i \leftarrow \text{MM}(Pa_i, 1, m_i)$
- 12: $Zr_i \leftarrow \text{MM}(Pb_i, M_i, M)$
- 13: **end for**
- 14: $Zr \leftarrow Zr_p + Zr_q$
- 15: $z \leftarrow \text{MM}(Zr, R^2, M)$
- 16: **return** z

5.2.1.1 Correctness

The following groups of lines (1 to 4, 5 to 7, 8 to 13, and 14 to 15) for *Forward conversion*, *Modulo m_i* , and *Reverse conversion* show the correctness of Algorithm 8.

Lines 1 to 4 perform the conversion of the input operand x to the RNS representation of 2-residue X_i , with respect to m_i , $i = p, q$, in the following equation:

$$X_i \equiv x \pmod{m_i}, \text{ for } i = p, q. \quad (5.6)$$

These reductions are executed simultaneously on specific parallel hardware using a generic MM algorithm. The correctness of these steps is as follows:

$$\begin{aligned}
X_i &= \mathbf{MM}(\mathbf{MM}(x, 1, m_i), R_{x_i}^2, m_i) \\
&\equiv \left((xR_{x_i}^{-1} \bmod m_i) R_{x_i}^2 R_{x_i}^{-1} \right) \bmod m_i \\
&\equiv x \pmod{m_i}, \text{ for } i = p, q.
\end{aligned} \tag{5.7}$$

Lines 5 to 7 compute the intermediate modular exponentiation results, Z_i , in each Montgomery 2-domain using the MM algorithm repeatedly (Algorithm 2) in parallel, by computing $Z_i \equiv (X_i)^e \pmod{m_i}$, for $i = p, q$. The correctness of this computation is presented in (KNUTH, 1981).

Lines 8 to 13 perform the *Conversion from Montgomery Domain to RNS Representation* by converting the results of each modular exponentiation, Z_i , back to the RNS representation and then computes the output result, z , as the conventional notation of the input x , as shown in the following equation:

$$\begin{aligned}
z &\equiv (Z_p M_p (M_p^{-1} \bmod m_p) + Z_q M_q (M_q^{-1} \bmod m_q)) \bmod M \\
&\equiv (Z_p M_p^{-1} \bmod m_p) M_p \bmod M + (Z_q M_q^{-1} \bmod m_q) M_q \bmod M,
\end{aligned} \tag{5.8}$$

where $M_i = \frac{M}{m_i}$ and $|M_i^{-1}|_{m_i}$ are the inverse of M_i modulo m_i , i.e. $(M_i^{-1} M_i) \bmod m_i \equiv 1$, and $Z_i \equiv Z_i \pmod{m_i}$, for $i = p, q$.

Equation (5.8) is based on equation (4.26). The partial results of the two terms in equation (5.8), $(Z_p M_p^{-1} \bmod m_p) \equiv Zr_p$ and $(Z_q M_q^{-1} \bmod m_q) \equiv Zr_q$, are computed in parallel (lines 8 to 13), using the MM algorithm. The correctness of these steps takes into consideration the precomputed reductions, $M_{R_i}^{-1} \equiv M_i^{-1} R_i \pmod{m_i}$, for $i = p, q$,

as follows:

$$\begin{aligned}
Zr_i &= \mathbf{MM}\left(\mathbf{MM}\left(\mathbf{MM}\left(\mathbf{MM}\left(Z_i, R_i^2, m_i\right), M_{R_i}^{-1}, m_i\right), 1, m_i\right), M_i, M\right) \\
&\equiv \left(\left(\left(Z_i R_i^2 R_i^{-1} \bmod m_i\right) M_i^{-1} R_i^2 R_i^{-1} \bmod m_i\right) R_i^{-1} \bmod m_i\right) M_i R_i^{-1} \bmod M \\
&\equiv \left(\left(Z_i M_i^{-1} \bmod m_i\right) R_i^{-1} \bmod m_i\right) M_i R_i^{-1} \bmod M \\
&\equiv \left(Z_i M_i^{-1} R_i^{-1} \bmod m_i\right) M_i R_i^{-1} \bmod M \\
&\equiv x \pmod{m_i}, \text{ for } i = p, q.
\end{aligned} \tag{5.9}$$

Lines 14 to 15 add the partial results of the two terms, Zr_p and Zr_q , and then reduces the sum back from the Montgomery domain to the expected output result, z , as equation (5.8). These steps hold, which leads to the following equation:

$$\begin{aligned}
z &= \mathbf{MM}\left(Zr_p + Zr_q, R^2, M\right) \\
&\equiv \left(Zr_p + Zr_q\right) R_i^2 R_i^{-1} \bmod M \\
&\equiv \left(\left(Z_p M_p^{-1} R_p^{-1} \bmod m_p\right) M_p R_p^{-1} \bmod M\right) + \\
&\quad \left(\left(Z_q M_q^{-1} R_q^{-1} \bmod m_q\right) M_q R_q^{-1} \bmod M\right) R^2 R^{-1} \bmod M \\
&\equiv \left(Z_p M_p^{-1} \bmod m_p\right) M_p \bmod M + \left(Z_q M_q^{-1} \bmod m_q\right) M_q \bmod M
\end{aligned} \tag{5.10}$$

5.2.1.2 Asymptotic Analysis of Algorithm 8

We consider the $\mathbf{MM}(X, Y, M)$ notation here, according to equation (3.1), as the Radix-2 MM algorithm (Algorithm 1). The running time of the MM algorithm is denoted by the following equation:

$$T_{MM}(n) = O(n), \text{ for } n = 1 + \lfloor \log_2 M \rfloor. \tag{5.11}$$

Equation (5.11) is used to compose the asymptotic analysis of Algorithm 8 as follows:

1. *Forward Conversion* (lines 1 to 4):

The computation time of the input operand conversion x to the 2-residue X_i with respect to m_i , for $i = p, q$, running in parallel requires two Montgomery Multiplications (lines 2 and 9 - Algorithm 8).

The expected running time of these steps is

$$\begin{aligned} T_{FC}(n_x) &= 2T_{MM}(n_x) \\ &= O(n_x), \text{ with } n_x = 1 + \lfloor \log_2 x \rfloor. \end{aligned} \quad (5.12)$$

2. *Modulo m_i* (lines 5 to 7):

The running time of the MEXP algorithm (Algorithm 2), running in parallel for each *Modulo m_i* , requires

(i) Two Montgomery Multiplications (lines 2 and 9 - Algorithm 2).

(ii) The modular exponentiation loop (lines 3 to 8 - Algorithm 2) demands t Montgomery Squares and h Montgomery Multiplications, where t bits is the size of the exponent e and h bits is the Hamming weight of e , with $1 < h \leq t$. In Algorithm 2, both MM operations (lines 4 and 6) can be performed in parallel (CHIOU, 1993). However, by performing the modular squaring and multiplication operations simultaneously, the computation time for the modular exponentiation loop corresponds to the time to perform t Montgomery Multiplications.

Therefore, the running time of these steps is the following:

$$\begin{aligned} T_{MEXP}(n, t) &= 2T_{MM}(n) + tT_{MM}(n) \\ &= O(tn), \text{ with } n = 1 + \lfloor \log_2 m_i \rfloor. \end{aligned} \quad (5.13)$$

3. *Reverse Conversion* (lines 8 to 15):

The running time of the results conversion from each modular exponentiation, Z_i , back to the RNS representation requires

(i) Four Montgomery Multiplications (line 9 to 12 - Algorithm 8).

(ii) The sum of each modular exponentiation results (line 14 - Algorithm 8) demands an adder, e.g., the Carry-Propagate Adder (*CPA*), for which the computation time is $T_{CPA}(n) = O(1)$.

(iii) The last Montgomery Multiplication (line 15 - Algorithm 8) reduces the previous sum back from the Montgomery domain to the expected output result, z , which needs $2n$ bits of modulus M .

Thus, the running time of these steps is

$$\begin{aligned} T_{RC}(n) &= 3T_{MM}(n) + T_{MM}(2n) + T_{CPA}(n) + T_{MM}(2n) \\ &= O(n), \text{ with } n = 1 + \lfloor \log_2 m_i \rfloor. \end{aligned} \quad (5.14)$$

Therefore, the running time of Algorithm 8 for computing $z \equiv x^e \pmod{M}$ is denoted as follows:

$$\begin{aligned} T_{MEXPRNS}(n_x, n, t) &= T_{FC}(n_x) + T_{MEXP}(n, t) + T_{RC}(n) \\ &= O(n_x) + O(tn), \end{aligned} \quad (5.15)$$

where, $n_x = 1 + \lfloor \log_2 x \rfloor$, $n = 1 + \lfloor \log_2 m_i \rfloor$, and t bits is the size of the exponent e .

5.2.1.3 Numerical Example

Table 2 illustrates the running of each computation process of the Montgomery Exponentiation in the RNS method (MEXPRNS) by using the following (i) variables and (ii) precomputed values:

(i) $x = 2456$, $e = 5$, $M = 510971$, $p = m_p = 523$, and $q = m_q = 977$.

(ii) $R = 2^{19}$, $R^2 = 35552$, $R_x = 2^{12}$, $R_{x_p}^2 = 422$, $R_{x_q}^2 = 172$, $R_p = 2^{10}$, $R_p^2 = 484$, $R_q = 2^{10}$, $R_q^2 = 255$, $M_p = 977$, $M_q = 523$, $M_{R_p}^{-1} = 309$, and $M_{R_q}^{-1} = 686$.

The modular exponentiation result of $x^e \pmod{M}$ in this case is 294190.

Table 2: Running Montgomery Multiplication in the RNS Algorithm

<i>Forward Conversion</i>	
$W_p = \text{MM}(1, x, m_p)$ $= \text{MM}(1, 2456, 523) = 400$	$W_q = \text{MM}(1, x, m_q)$ $= \text{MM}(1, 2456, 977) = 117$
$X_p = \text{MM}(W_p, R_{x_p m_p}^2, m_p)$ $= \text{MM}(400, 422, 523) = 364$	$X_q = \text{MM}(W_q, R_{x_q m_q}^2, m_q)$ $= \text{MM}(117, 172, 977) = 502$
<i>Modulo m_i</i>	
$Z_p = \text{MEXP}(X_p, e, R_p^2, U_p, m_p)$ $= \text{MEXP}(364, 5, 484, 523) = 264$	$Z_q = \text{MEXP}(X_q, e, R_q^2, U_q, m_q)$ $= \text{MEXP}(502, 5, 255, 977) = 113$
<i>Reverse Conversion</i>	
$Zr_p = \text{MM}(Z_p, R_p^2, m_p)$ $= \text{MM}(264, 484, 523) = 468$	$Zr_q = \text{MM}(Z_q, R_q^2, m_q)$ $= \text{MM}(113, 255, 977) = 426$
$Pa_p = \text{MM}(Zr_p, M_{R_p}^{-1}, m_p)$ $= \text{MM}(468, 309, 523) = 448$	$Pa_q = \text{MM}(Zr_q, M_{R_q}^{-1}, m_q)$ $= \text{MM}(426, 686, 977) = 692$
$Pb_p = \text{MM}(1, Pa_p, m_p)$ $= \text{MM}(1, 448, 523) = 360$	$Pb_q = \text{MM}(1, Pa_q, m_q)$ $= \text{MM}(1, 692, 977) = 867$
$Zr_p = \text{MM}(Pb_p, M_p, M)$ $= \text{MM}(360, 977, 510971) = 60574$	$Zr_q = \text{MM}(Pb_q, M_q, M)$ $= \text{MM}(867, 523, 510971) = 310139$
$Zr = Zr_p + Zr_q = 60574 + 310139 = 370713$	
$z = \text{MM}(Zr, R^2, M) = \text{MM}(370713, 35552, 510971) = 294190$	

6 DESIGN OF LOW-POWER MULTIPLIERS

Implementing public-key cryptography is a challenging task. In this chapter, we discuss the two architectures of the Parallel k -Partition Montgomery Multiplication ($kPMM$) and the Montgomery Exponentiation in RNS (MEXPRNS) implementations for low power.

6.1 Parallel k -Partition Method

In this section, an overview of the k -Partition Montgomery Multiplication architecture ($kPMM$) is presented, based on the algorithm described in Chapter 5. To reduce the power consumption demanded by the conventional CS representation, we propose fast implementations of multipliers, using a carry-save number representation called sparse CS representation. The section ends with a high level analysis of the delay and the area of the $kPMM$ architecture.

6.1.1 MM Partition Kernel Architecture

Figure 5 shows the MM Partition Kernel block diagram. It represents the main function performed by the MMP module.

The *SelOp1* block (selection of multiples of Y – input operand for the first adder) is generated by a simple logic that sends $x_j 2^j Y$ to carry-save adder 1 (*CSA 1*) (the element to be added to the partial sum in Algorithm 6 – line 3). When the current input bit $x_j = 1$, the n -bit multiplicand operand Y is shifted to the left by 2^j bit positions

Figure 5: Architecture of the MM Partition Kernel (MMP Kernel)

(the multiplication of Y by 2^j); otherwise, the adder receives a zero input. The output operand is called $Op1$ in the figure.

The *ShiftSC* block shifts the intermediate result in CS form ($Sout$ and $Cout$) by 2^k bit positions to the right, generating the input (Sin , Cin) for *CSA 1* (division by 2^k – line 5 of Algorithm 6).

The intermediate results in the carry-save form ($Sout$ and $Cout$) are represented with $n + d$ bits. The maximum value of each multiple of Y and M is represented with $n + k$ bits. The *CSA 2* block adds four input operands: $Sin2$, $Cin2$ and the multiples of M ($Op2$, $Op3$). Therefore, $d = k + 2$ extra bits are needed for the safe carry save representation of the intermediate results.

The bits $Q_i[k - 1 : 0]$ ($q_{k-1..0}$ in line 4 – Algorithm 6) are computed by the *LogicQi* block. Based on the value of the partial sum in CS form ($Sin2$, $Cin2$) and the value of the modulus M , this block defines the value of Q_i to be used. This value is a digit in radix 2^k (TODOROV, 2000).

The function to produce multiples of M (*SelOp2*) is expensive. The outputs of this block ($Op2$, $Op3$) are sent to *CSA 2* (4:2 compressor). As we increase k , there is a wide range of values for Q_i and a more complex logic to generate $Op2 = Q_i[k - 1 : 0]M$. For small values of k , this logic can be implemented by using multiplexers. For larger k

values, the use of a table lookup implementation is recommended, where pre-computed values of multiples of M are stored.

6.1.2 MM Partition j Architecture (MMP)

The block diagram of the j^{th} MM Partition is shown in Figure 6. The two blocks (*SumReg*, *CarryReg*) represent registers that keep the intermediate results in carry save form for each MMP.

Figure 6: MM Partition j Architecture (MMP)

6.1.3 Parallel k -Partition MM Architecture (k PMM)

The top level of the Parallel k -Partition MM architecture is illustrated in Figure 7. It integrates the control and data input signals with *ShiftX*, *LoopCtrl*, and *AdderCS* functions.

The initial value loaded into the *ShiftX* block is X (the multiplicand). This block is a k -bit shifter that has only the k least significant bits (LSBs) of the internal value as output. The output of the *ShiftX* function is distributed among all k partitions. Each output bit becomes the X_j bit to be processed by the respective MMP block. In each clock cycle, this block shifts its output by k bit positions to the right.

Figure 7: Fully Parallel k -Partition MM Architecture (k PMM) - Top Level

The signal *done* is set by the *LoopCtrl* function when the multiplication is completed. This signal indicates when n/k clock cycles have been applied; therefore, all of the partitions have completed their processing.

The *AdderCS* block is responsible for the accumulation of the CS outputs of all partitions. It is activated only after the completion of the MMP computation. Thus, a way to save energy is to leave this hardware block turned off until the signal *done* is activated. This way, this block consumes power only when the accumulation of partition outputs is needed. The accumulation should be conducted in such way that it does not compromise the clock period of each partition. There are several ways to perform this task, but here we are using a sequential circuit that accumulates the results. After $k - 1$ clock cycles, the final multiplication result in CS form is obtained.

6.1.4 Optimizing for Better Power

Figure 5 shows an efficient modification of the carry-save adder (CSA) circuit which is widely known to produce the intermediate sums in CS form. The conventional CSA architecture is composed of full adders that run in parallel and independently. Most authors use the regular CS form (2 bits per column) to represent the intermediate sums, but the use of this CS form requires many flip-flops (FFs), consuming significant area and power.

As shown in Figure 6, the intermediate results of the MM Kernel block are stored in two $(n + d)$ -bit registers (*SumReg*, *CarryReg*), which are responsible for the high energy consumption per MM Partition.

To reduce the number of registers, we apply a transformation to the general CS form (2 bits per column) that generates groups of s columns where only 1 column has 2 bits and the others have 1 bit. To perform this transformation, the groups of s columns from the general CS representation (*Sin* and *Cin*, forming 2 bits per column) are transformed into a binary result of $(s + 1)$ bits, as shown in the following equations:

$$\begin{aligned} W[s : 0] &= Sin[s - 1 : 0] + Cin[s - 1 : 0] \\ Sout[s - 1 : 0] &= W[s - 1 : 0] \end{aligned} \tag{6.1}$$

$$Cout[s] = W[s] \tag{6.2}$$

Because there is an overlap of bit $W[s]$ of one group with bit $W[0]$ of the next higher order group, the sparse CS form (sCS) has a column with 2 bits every s columns. This converter is called a Sparse Carry-Save Adder herein (*sCSA*). In (BEUCHAT; MULLER, 2008), the authors suggested a high-radix carry-save number system, which has a structure similar to *sCSA*. Figure 8 illustrates the *sCSA*, with their inputs and outputs, in dot notation.

Figure 9 shows the new MM Partition Kernel (MMP) diagram, implementing this

Figure 8: Sparse-Carry-Save Adder in Dot Notation

optimization as follows. The outputs of the $CSA2$ block are sent as inputs to the $sCSA$. The $sCSA$ block architecture can be implemented by $b = \lceil (n + d)/s \rceil$ Carry-Propagate Adder (CPA) blocks.

Figure 9: Optimized Architecture of MM Partition Kernel

As described above, it is clear that the energy consumption from the *CarryReg* register is reduced because it requires $(s - 1)(n + d)/s$ fewer FFs to register the carry bit.

The overall reduction in the number of FFs for the CS register ($SumReg, CarryReg$) is calculated as $(s - 1)/(2s)$, which implies that larger values of s will lead to larger reductions in power at the register level. However, the complexity of each CPA used to obtain the sCS format increases with s , which slows the computation and increases the overall area. This effect counter-balances the reduction in the number of FFs. Figure 10 shows the impact on power consumption of a given MM Kernel Partition with s -bit blocks varying from 1 to 16. It is a design

Figure 10: The impact of the block sizes on the power consumption of the MM Partition Kernel

trade-off investigated later in the experimental section.

6.1.5 Complexity Evaluation of the Proposed Architecture

This subsection presents the complexity evaluation of the proposed architecture with the useful expressions to predict the values of the area and the delay for a given configuration without having to perform a synthesis of the circuit. Taking the area and delay values of a GATE, DFF, MUX, FA, and other gates/circuits from a given cell library, one can see a connection with the values obtained by the synthesis tool.

Table 3 shows the gate/circuit equivalents to measure the area and the time per block.

For a configuration of with n -bit input operands Y and M , k partitions and $sCSA$ with blocks with a size of s bits, the area and the critical path delays for each MM are

$$A_{MMP}(n, k, s) = (2^{k+1})A_{GATE} + ((s-1)(n+d)/s)A_{HA} + ((s+1)(n+d)/s+k)A_{FA} +$$

$$((n+d)/s)A_{CPA}(s) + 3(n+d)A_{MUX} + ((s-1)(n+d)/s)A_{DFF}, \text{ and}$$

$$T_{MMP}(n, s) = 4T_{GATE} + 4T_{FA} + T_{CPA}(s) + 2T_{MUX} + T_{DFF}.$$

Table 3: Area and time for gate/circuit equivalents

Area	Time	Gate/circuit equivalents
A_{GATE}	T_{GATE}	2-input AND or OR
A_{HA}	T_{HA}	2-input half adder
A_{FA}	T_{FA}	3-input full adder
$A_{CPA}(s)$	$T_{CPA}(s)$	s -bit CPA
A_{MUX}	T_{MUX}	2-input multiplexer
A_{DFF}	T_{DFF}	1-input D type flipflop

The area and the critical path delays equations for the fully parallel k -Partition MM based on the presented architecture are

$$A_{kPMM}(n, k, s) = k[A_{MMP}(n, k, s)] + 2(n + d)A_{FA} + k(n + d)A_{MUX} + (n + d)A_{DFF}, \text{ and}$$

$$T_{kPMM}(n, k, s) = 4T_{GATE} + 6T_{FA} + T_{CPA}(s) + (2 + k)T_{MUX} + 2T_{DFF}.$$

In Table 4, the equivalent number of gates/circuits (area) and critical path delays (time) from a generic $kPMM$ is decomposed using the upper bound variable values of input bits, e.g., the *ShiftX* block receives an input of $(n + d)$ bits.

Table 4: Area and time per block of the Parallel k -Partition MM

Block(input bits)	Area	Time
<i>ShiftX</i> ($n + d$)	$2(n + d)A_{GATE} + (n + d)A_{DFF}$	$T_{GATE} + T_{DFF}$
<i>SelOp1</i> (n)	$(n + d)A_{MUX}$	T_{MUX}
<i>CSA1</i> ($n + d$)	$((s - 1)(n + d)/s)A_{HA} + ((n + d)/s)A_{FA}$	T_{FA}
<i>LogiQi</i> (k)	kA_{FA}	$2T_{FA}$
<i>SelOp2</i> (n)	$(2^{k+1})A_{GATE} + 2(n + d)A_{MUX}$	$4T_{GATE} + T_{MUX}$
<i>CSA2</i> ($n + d$)	$(n + d)A_{FA}$	T_{FA}
<i>sCSA</i> ($n + d$)	$((n + d)/s)A_{CPA}(s)$	$T_{CPA}(s)$
<i>SumReg</i> ($n + d$)	$(n + d)A_{DFF}$	T_{DFF}
<i>CarryReg</i> (($n + d$)/ s)	$((n + d)/s)A_{DFF}$	T_{DFF}
<i>AdderCS</i> ($n + d$)	$k(n + d)A_{MUX} + (n + d)A_{DFF} + 2(n + d)A_{FA}$	$kT_{MUX} + T_{DFF} + 2T_{FA}$

6.2 Montgomery Exponentiation in RNS

In this section, we present an overview of the Montgomery Exponentiation in the RNS (MEXPRNS) architecture, as described in Algorithm 8. For a low-power design, we propose the *Forward Conversion*, *Modulo m_i* , and *Reverse Conversion* processes, which are based on the Parallel Radix-2 MM architecture.

The top level of MEXPRNS architecture is illustrated in Figure 11, integrating the *Forward Conversion*, *Modulo m_i* , and *Reverse Conversion* blocks, as described in Algorithm 8.

The control logic for MEXPRNS (FSM – Finite State Machine) has seven states, which are represented by the State Transition Table 5. We apply the don't care X symbols in the State Transition Tables whenever the next state does not depend on an input signal. Also, the don't care X symbols can be used in the State Transition Tables on output signals. This means that output signal will not be assigned in such state transition. In Table 5, there are double states to guide the control signals to and from each data router and controller signals (*Forward Conversion*, *Modulo m_i* , and *Reverse Conversion*), by enabling a sequential step per FSM in Figures 12, 17, and 18.

Current State	Input Signals					Next State	Output Signals							
	enable	load	FCdone	MEdone	RCdone		done	FCenable	FCload	MEenable	MEload	FCenable	FCload	BusDirCtrlEnable
any	1	1	X	X	X	0	0	1	1	0	0	0	0	10
0	1	0	0	X	X	1	0	1	0	0	0	0	0	10
1	1	0	1	X	X	2	0	0	0	1	1	0	0	01
2	1	0	X	0	X	3	0	0	0	1	0	0	0	01
3	1	0	X	1	X	4	0	0	0	0	0	1	1	11
4	1	0	X	X	0	5	0	0	0	0	0	1	0	11
5	1	0	X	X	1	6	0	0	0	0	0	0	0	00
6	1	X	X	X	X	6	1	0	0	0	0	0	0	00

Table 5: State Transition of the Control Logic for MEXPRNS - FSM

The MM Extended (MME) architecture is based on dual Radix-2 MM algorithm. Consequently, the MME block receives data and control signals for 2-Partition MM architecture and produces their separate results. The MME blocks, modules p and q ,

represent the modular multiplications that are executed for each of the inner loops, lines 1 to 4, lines 5 to 7, lines 8 to 13 and line 15 in Algorithm 8. Each MM per MME block is enabled or disabled according to the controllers in Figure 11. The MME blocks can be set to perform four, or two MM in parallel, or even only one in a given state. Each MME can be set to join the input operands and results, to produce $2n$ -bit multiplication precision. The settings used per MME are presented in the following subsections.

The signals *BusDtCtrlEnable* control the MUX switches for entering the data and control signals into the MME blocks in a given state. When the signals *RCdone* is set and state 6 is reached (Table 5), the Montgomery Exponentiation in RNS is completed, and the signal *done* is set.

6.2.1 Forward Conversion

Figure 12 shows the *Forward Conversion* (FC) data router and controller block diagram, which guides the MME modules to compute the lines 1 to 4 as illustrated in Algorithm 8. To perform lines 2 and 3, the MME modules are set with n -bit reductions.

Current State	Input Signals						Next State	Output Signals																
	enable	load	MM _{1p} done	MM _{0p} done	MM _{1q} done	MM _{0q} done		FCdone	MME _{1p} enable	MME _{1p} load	MME _{0p} enable	MME _{0p} load	cfg _p	MME _{1q} enable	MME _{1q} load	MME _{0q} enable	MME _{0q} load	cfg _q	Ctrl _{1p} X ₀	Ctrl _{1p} X ₁	Ctrl _{1p} Y	Ctrl _{1q} X ₀	Ctrl _{1q} X ₁	Ctrl _{1q} Y
any	1	1	X	X	X	X	0	0	1	1	0	0	XX	1	1	0	0	XX	X	X	X	X	X	X
0	1	0	0	X	0	X	1	0	1	0	0	0	10	1	0	0	0	10	1	1	1	1	1	1
1	1	0	0	X	0	X	2	0	1	0	0	0	10	1	0	0	0	10	0	1	1	0	1	1
2	1	0	1	X	1	X	3	0	1	0	0	0	00	1	0	0	0	00	X	0	0	X	0	0
3	1	0	X	X	X	X	3	1	0	0	0	0	00	0	0	0	0	00	X	0	0	X	0	0

Table 6: State Transition of the Control Logic for Forward Conversion (FC) - FSM

The State Transition Table 6 shows the four states of the control logic for *Forward Conversion* (FSM). States 0 and 1 represent the line 2 in Algorithm 8, to calculate W_p and W_q values. These states denote the data router of the n LSBs and the n most significant bits (MSBs) of the x operand respectively. State 2 resets the MME modules

Figure 11: Architecture of Montgomery Exponentiation in RNS (MEXPRNS) – Top Level

operation to compute the X_p and X_q values as described in line 3. When state 3 is reached, the last two multiplications are completed and then the signal $FCdone$ is set.

Figure 12: Architecture of Forward Conversion (FC) Data Router and Controller

6.2.2 Exponentiation Modulo m_i

The Modular Exponentiation (ME) data router and controller block diagram is illustrated in Figure 17, which controls the MME modules to perform the MM Exponentiation operations in lines 5 to 7 (Algorithm 8). The MME modules run in parallel to compute the four MMs as described in lines 2, 4, 6 and 9 (Algorithm 2).

The control logic for *Modular Exponentiation* (FSM) has eleven states, which are represented by the State Transition Table 7. States 0 and 1 represent line 2 in Algorithm 2, which computes the s values per MME modules. States 2, 4, 5, and 6 (six lines) represent the calculation of lines 4, 6 or 9 in Algorithm 2, at the first iteration, when $i = 0$ and $t > 0$, or if the exponent, e , is zero. States 3, 7 and 8 (six lines) represent the MM computation in the inner loop line 4 only, or both lines 4 and 6, depending on the e_i value, at the iterations when $i > 0$. When the signal *MEXPdone* is set and states 2, 6, 7, 8, 9 or 10 are reached, the modular exponentiation is completed, and the signal *MEdone* is set.

Current State	Input Signals								Next State	Output Signals																			
	enable	load	MME _{1p} done	MME _{2p} done	MME _{1q} done	MME _{2q} done	e[i]	MEXPdone		MEdone	MEXPdone	MME _{1p} enable	MME _{1p} load	MME _{2p} enable	MME _{2p} load	cfg _p	MME _{1q} enable	MME _{1q} load	MME _{2q} enable	MME _{2q} load	cfg _q	Ctrl _{1p} X	Ctrl _{1p} Y	Ctrl _{2p} X	Ctrl _{2p} Y	Ctrl _{1q} X	Ctrl _{1q} Y	Ctrl _{2q} X	Ctrl _{2q} Y
any	1	1	X	X	X	X	X	X	0	0	0	1	1	0	0	XX	1	1	0	0	XX	X	X	X	XX	X	X	X	X
0	1	0	0	X	0	X	X	0	1	0	0	1	0	0	0	00	1	0	0	0	00	1	1	X	XX	1	1	X	X
1	1	0	1	X	1	X	X	0	2	0	0	1	1	1	1	00	1	1	1	1	00	X	X	X	XX	X	X	X	X
2	1	0	0	0	0	0	0	0	4	0	0	1	0	0	0	00	1	0	0	0	00	0	0	X	XX	0	0	X	X
2	1	0	0	0	0	0	1	0	5	0	0	1	0	1	0	00	1	0	1	0	00	0	0	1	10	0	0	1	1
2	1	0	0	0	0	0	X	1	6	1	1	0	0	1	0	00	0	0	0	0	00	X	X	1	11	X	X	X	X
3	1	0	0	X	0	X	0	0	7	0	0	1	0	0	0	00	1	0	0	0	00	0	0	X	XX	0	0	X	X
3	1	0	0	0	0	0	1	0	8	0	0	1	0	1	0	00	1	0	1	0	00	0	0	0	01	0	0	0	0
4	1	0	1	X	1	X	X	0	2	0	0	1	1	1	1	00	1	1	1	1	00	X	X	X	XX	X	X	X	X
5	1	0	1	1	1	1	X	0	3	0	0	1	1	1	1	00	1	1	1	1	00	X	X	X	XX	X	X	X	X
6	1	0	X	X	X	X	X	1	6	1	1	0	0	0	0	00	0	0	0	0	00	X	X	X	XX	X	X	X	X
7	1	0	1	X	1	X	X	0	3	0	0	1	1	1	1	00	1	1	1	1	00	X	X	X	XX	X	X	X	X
7	1	0	1	X	1	X	X	1	9	0	0	1	0	1	0	00	1	0	1	0	00	X	X	X	XX	X	X	X	X
8	1	0	1	1	1	1	X	0	3	0	0	1	1	1	1	00	1	1	1	1	00	X	X	X	XX	X	X	X	X
8	1	0	1	1	1	1	X	1	10	0	0	1	1	1	1	00	1	1	1	1	00	X	X	X	XX	X	X	X	X
9	1	0	X	X	X	X	X	1	9	1	1	0	0	0	0	00	0	0	0	0	00	X	X	X	XX	X	X	X	X
10	1	0	X	X	X	X	X	1	10	1	1	0	0	0	0	00	0	0	0	0	00	X	X	X	XX	X	X	X	X

Table 7: State Transition of the Control Logic for Modular Exponentiation (ME) - FSM

6.2.3 Reverse Conversion

Figure 18 shows the *Reverse Conversion* (RC) data router and controller block diagram, which guides the MME modules to perform the *Conversion from Montgomery Domain to RNS* representation in parallel, as illustrated in lines 8 to 15 (Algorithm 8). To compute the first three MM (lines 9 to 11), the MME modules are set with n -bit operands, and the last one (line 12) is set with $2n$ bits. The final result, z , is produced with $2n$ bits.

The State Transition Table 8 shows the control logic for *Reverse Conversion* (FSM), with the following relations:

- (a) States 0 and 1 represent line 9.
- (b) States 2 and 3, states 4 and 5, and states 6 and 7 represent lines 10, 11 and 12 respectively.
- (c) States 8 and 9 represent line 15.

When the signals $MME_{1p}done$ and $MME_{0p}done$ are set and the state 10 is reached, the *Reverse Conversion* is completed. In this event, the signal $RCdone$ is set.

Current State	Input Signals						Next State	Output Signals																		
	enable	load	MME _{1p} done	MME _{0p} done	MME _{1q} done	MME _{0q} done		RCdone	MME _{1p} enable	MME _{1p} load	MME _{0p} enable	MME _{0p} load	cfg _p	MME _{1q} enable	MME _{1q} load	MME _{0q} enable	MME _{0q} load	cfg _q	Ctrl _{1p} X	Ctrl _{1p} Y	Ctrl _{1p} M	Ctrl _{0p} X	Ctrl _{0p} Y	Ctrl _{1q} X	Ctrl _{1q} Y	Ctrl _{1q} M
any	1	1	X	X	X	X	0	0	1	1	0	0	XX	1	1	0	0	XX	XXX	XXX	X	X	X	XXX	XXX	X
0	1	0	0	X	0	X	1	0	1	0	0	0	10	1	0	0	0	10	100	100	1	X	X	100	100	1
1	1	0	1	X	1	X	2	0	1	1	0	0	10	1	1	0	0	10	XXX	XXX	X	X	X	XXX	XXX	X
2	1	0	0	X	0	X	3	0	1	0	0	0	10	1	0	0	0	10	010	010	1	X	X	010	010	1
3	1	0	1	X	1	X	4	0	1	1	0	0	00	1	1	0	0	00	XXX	XXX	X	X	X	XXX	XXX	X
4	1	0	0	X	0	X	5	0	1	0	0	0	00	1	0	0	0	00	010	110	1	X	X	010	110	1
5	1	0	1	X	1	X	6	0	1	1	1	1	00	1	1	1	1	00	XXX	XXX	X	X	X	XXX	XXX	X
6	1	0	0	0	0	0	7	0	1	0	1	0	01	1	0	1	0	01	110	101	0	1	1	110	101	0
7	1	0	1	1	1	1	8	0	1	1	1	1	00	0	0	0	0	00	XXX	XXX	X	X	X	XXX	XXX	X
8	1	0	0	0	X	X	8	0	1	0	1	0	01	0	0	0	0	00	101	111	0	0	0	XXX	XXX	X
9	1	0	1	1	X	X	10	1	0	0	0	0	00	0	0	0	0	00	XXX	XXX	X	X	X	XXX	XXX	X
10	1	0	X	X	X	X	10	1	0	0	0	0	00	0	0	0	0	00	XXX	XXX	X	X	X	XXX	XXX	X

Table 8: State Transition of the Control Logic for Reverse Conversion (RC) - FSM

6.2.4 MM Extended Architecture (MME)

The top level of the MM Extended architecture is shown in Figure 13. It is based on the Parallel 2-Partition MM architecture as described in Figure 7.

The MME architecture provides a dual operating mode, in order to multiply operands of different binary lengths (n bits or $2n$ bits) as applied in Algorithm 8.

An "single multiplication mode" is called when the MME calculation is set to perform 1 step of the Radix-2 MM Algorithm 1 (lines 3 to 4) for $2n$ -bit operand,

Figure 13: MM Extended Architecture (MME) – Top Level

producing the intermediate results as the following arrangement of operands:

$$(S_1[i+1]|S_0[i+1]) = \frac{(S_1[i]|S_0[i]) + (Y_1|Y_0)(X_1|X_0)[i] + (M_1|M_0)q_0}{2}, \quad (6.3)$$

where $(A|B)$ is the concatenation of A and B , and q_0 is the 1 LSB of the $(S_1[i]|S_0[i]) + (Y_1|Y_0)(X_1|X_0)[i]$ calculation. Equation 6.3 uses $2n$ -bit MM step.

A "double multiplication mode" is called when the MME operation is set to execute 1 step of MM radix-2 algorithm over two n -bit operands, generating two separate intermediate results, as follows:

$$S_1[i+1] = \frac{S_1[i] + Y_1X_1[i] + M_1q_1}{2}, \text{ and} \quad (6.4)$$

$$S_0[i+1] = \frac{S_0[i] + Y_0X_0[i] + M_0q_0}{2}, \quad (6.5)$$

where q_0 and q_1 are the 1 LSB of the $S_0[i] + Y_0X_0[i]$ and $S_1[i] + Y_1X_1[i]$ computation, respectively. For each case, equations 6.4 and 6.5 use n -bit MM step.

The MME dual operating mode is established by setting the signal $cfg[0]$. When the $cfg[0]$ bit value is zero, MME performs in single multiplication mode. Otherwise, MME executes in double multiplication mode.

In the particular case of Algorithm 8, line 2, the MME operation must be set with the $2n$ -bit multiplier operand X , and the n -bit operands multiplicand Y and modulus M . For this reduction, the signal $cfg[1]$ is set.

The multiplicand values X_1 and X_0 are loaded into the module *ShiftX* when the signals $load_1$ and $load_0$ are active, respectively. This block is a 1-bit shifter that has only 1 LSB of the internal value as output. The output of the *ShiftX* function are the $Xj[1]$ and $Xj[0]$ bits, as the j^{th} -bit of X_1 and X_0 respectively. In each clock cycle, this block shifts 1 bit position to the right.

The signals $done_1$ and $done_0$ are set by the *LoopCtrl* function, when the multiplication is completed. These signals indicate when n or $2n$ clock cycles were applied, depending on the signal $cfg[1]$.

The *Dual Adder* block is responsible to compute the CS outputs of DMMM. It is activated only after the completion of the DMMK calculation. Thus, a way to save energy is to leave this hardware block turned off until the signals $done_1$ and $done_0$ are activated.

6.2.5 Dual Mode MM Architecture (DMMM)

The block diagram of the Dual Mode MM is shown in Figure 14. The four blocks (*Sum₁Reg*, *Sum₀Reg*, *Carry₁Reg*, and *Carry₀Reg*) represent registers that keep the intermediate results in carry save form for each DMMK. As described in Subsection 6.2.4, when the single multiplication mode is set the value represented by two combined registers, (*Sum₁Reg*, *Sum₀Reg*) and (*Carry₁Reg*, *Carry₀Reg*) are $(2n + 2)$ bits of the intermediate results. Otherwise, each register represents $(n + 1)$ of bits.

Figure 14: Dual Mode MM Architecture (DMMK)

6.2.6 Dual MM Kernel Architecture (DMMK)

Figure 15 shows the Dual MM Kernel block diagram. It represents the basic function performed by the DMMK module, and it is based on two parallel Radix-2 MM Partition Kernel modules (Figure 5). For the MME double multiplication mode operation, DMMK performs as described in Section 6.1, using 2-Partition MM architecture.

In the following we describe the control and data input signals for the MME single multiplication mode operation.

In this case, the $ShiftSC_0$ and $ShiftSC_1$ blocks work extended, and the signal

Figure 15: Architecture of Dual MM Kernel (DMMK)

$cfg[0]$ controls the propagation of $S_0[0]$ and $C_0[0]$ bits into the bit positions $S_1[n]$ and $C_1[n]$, respectively, making the intermediate result with $2n$ bits ($(S_0|S_1)$ and $(C_0|C_1)$). Thus, these blocks shift the intermediate result by 1-bit position to the right in the CS form generating the input for the $CSA\ 1_1$ and $CSA\ 1_0$ blocks.

The $SelOp1_1$ block receives the signal $Xj[1]$ with the same value of the signal $Xj[0]$. As a result, $SelOp1_1$ selects the value Y_1 or zeros, depending on the $Xj[1]$ value, as the $SelOp1_0$ block does for the value Y_0 or zeros. These blocks work extended to join $2n$ bits of the multiples of the Y operand, $(Y_0|Y_1)$, generating the input operand $(Op1_0|Op1_1)$ for $CSA\ 1_1$ and $CSA\ 1_0$. In this way, $CSA\ 1_1$ and $CSA\ 1_0$ turn into a $2n$ -bit adder.

The $LogicQi_1$ block is disabled for single multiplication mode. In this case, the bit value $Qi_0[0]$ is transmitted from the $LogicQi_0$ block as input to $SelOp2_1$. According

to the $Q_{i_0}[0]$ value, $SelOp2_1$ selects the value M_1 or zeros, as $SelOp2_0$ does for the value M_0 or zeros. In this way, the $SelOp2_1$ and $SelOp2_0$ blocks also work extended to combine the $2n$ bits of the multiples of the M operand, $(M_0|M_1)$, by compounding the $(Op2_0|Op2_1)$ operands for $CSA\ 2_0$ and $CSA\ 2_1$.

The $Cin2_1[0]$ bit value is zero for an operand size of n bits. In this case, the $Cin2_0[n]$ bit value is transferred into this bit position, $Cin2_1[0]$, to produce the $2n$ bits of the intermediate result in CS form, $(Sin2_1|Sin2_0)$ and $(Cin2_1|Cin2_0)$, as input for $CSA\ 2_1$ and $CSA\ 2_0$ blocks.

Figure 16 illustrates the $CSA\ 2_1$ and $CSA\ 2_0$ adders, with their inputs, and outputs, in dot notation.

Figure 16: Single Multiplication Mode of $CSA\ 2_1$ and $CSA\ 2_0$ Adders, in Dot Notation

Figure 17: Architecture of Modular Exponentiation (ME) Data Router and Controller

Figure 18: Architecture of Reverse Conversion (RC) Data Router and Controller

7 EXPERIMENTAL RESULTS

In this chapter, we summarize the experimental results for the optimization and design of the low-power multiplier.

The functionality of the Parallel k -Partition MM and the Montgomery Exponentiation in RNS methods were verified using simulation. The blocks presented in Sections 6.1 and 6.2 were described in VHDL and simulated using VCS (Synopsys simulation tool). The designs were developed using the same design facilities and tools. The hardware description was synthesized with Synopsys Design Compiler using a 90nm CMOS library “saed90nm_typ.db”.

7.1 Parallel k -Partition Method

In this section, we describe the results of the fully parallel k PMM architecture implementation, the benchmark condition, and the analysis of energy consumption. In addition, we present some enhancements for future work.

7.1.1 Benchmark

A baseline architecture for comparison was established using the Radix-2 MM. In this work, we conducted experiments focused on k PMM architectures with k varying from 1 to 6. The proposed method herein uses only the MM algorithm while the others in (KAIHARA; TAKAGI, 2008) and (SAKIYAMA et al., 2011) use mixed algorithms. The new algorithm enables a simpler design process, and all of the partitions are the same

(uniform design) while others have different hardware in each partition (which should reduce the design and testing time).

Table 9: The Summary of the Report Timing, the Area, and the Energy Consumption of the MM Architectures

n bits	Architecture	Multiplication Time (μ s)	Total Area (mm^2)	Total Area Increase (times)	Total Power per bit Multiplication (mW)	Energy Consumption for one Montgomery Multiplication ($\text{mW}\cdot\mu\text{s}$)	Reduction Energy Consumption (%)
256	Radix-2 MM	5.12	0.07		1.75	0.45	
	1-Partition MM	5.12	0.09	0.39	1.24	0.32	29%
	2-Partition MM	2.56	0.14	1.09	2.16	0.28	38%
	3-Partition MM	1.71	0.24	2.68	3.47	0.30	34%
	4-Partition MM	1.28	0.32	3.86	4.66	0.30	33%
	5-Partition MM	1.02	0.42	5.46	5.89	0.30	33%
	6-Partition MM	0.85	0.56	7.57	7.70	0.33	27%
512	Radix-2 MM	17.92	0.13		3.26	1.67	
	1-Partition MM	17.92	0.18	0.39	2.42	1.24	26%
	2-Partition MM	8.96	0.28	1.08	4.47	1.14	31%
	3-Partition MM	5.97	0.49	2.71	7.20	1.23	26%
	4-Partition MM	4.48	0.64	3.87	9.65	1.23	26%
	5-Partition MM	3.58	0.86	5.52	11.96	1.22	27%
	6-Partition MM	2.99	1.15	7.71	16.25	1.39	17%
1024	Radix-2 MM	46.08	0.27		6.45	6.61	
	1-Partition MM	46.08	0.37	0.38	5.01	5.13	22%
	2-Partition MM	23.04	0.56	1.07	8.86	4.54	31%
	3-Partition MM	15.36	1.00	2.71	13.81	4.71	29%
	4-Partition MM	11.52	1.31	3.88	18.73	4.79	27%
	5-Partition MM	9.22	1.79	5.64	23.59	4.83	27%
	6-Partition MM	7.68	2.38	7.86	31.70	5.41	18%
2048	Radix-2 MM	112.64	0.52		12.87	26.36	
	1-Partition MM	112.64	0.73	0.40	10.33	21.15	20%
	2-Partition MM	56.32	1.10	1.12	17.70	18.12	31%
	3-Partition MM	37.55	1.98	2.82	27.77	18.96	28%
	4-Partition MM	28.16	2.62	4.06	36.68	18.78	29%
	5-Partition MM	22.53	3.65	6.04	49.16	20.14	24%
	6-Partition MM	18.77	4.82	8.29	60.97	20.81	21%

Table 9 shows the summary of multiple experiments used for the comparison of the seven architectures. Each architecture was implemented for four different values of n (256, 512, 1024, and 2048 bits). Using Synopsys Design Compiler to perform hardware synthesis, a clock period value is adopted for each value of n (20, 35, 45, and 55 ns, respectively), which is not less than the largest critical path of any architecture to calculate the total area, the dynamic power, and the leakage power. For a given operand size (n) it is required to set the equivalent clock period value to determine the dynamic power. Furthermore, it is required to normalize the dynamic power obtained

for each architecture by multiplying this value by a factor (the clock period used for synthesis divided by an arbitrarily value of 55 ns), which allows the calculation of the energy consumption of all architectures. The leakage power values obtained do not depend on the clock period; thus, we used the synthesis values. For each case, we compare the following parameters to perform a complete modular multiplication:

- Number of MM clock cycles = $\frac{n \text{ bits}}{\text{number of MM partitions}}$;
- Multiplication Time = clock period $\times$ number of MM clock cycles;
- Total Area = Combinational + Non-combinational + Net Interconnect Area;
- Dynamic Power = Cell Internal + Net Switching Power;
- Total Power per bit Multiplication = Dynamic Power + Leakage Power;
- Energy Consumption for one Montgomery Multiplication = Multiplication Time $\times$ Total Power per n bits Multiplication.

In these experiments, we adopted the same block size for the sparse CS representation, which produces the minimum average power consumption per MMP block. The size for the sCS format, $s = 8$, with low power consumption was chosen, after the implementation of 64 different 1PMM modules, combining a value of s from 1 to 16 bits with each of the four values of n .

The multiplication time for the experiments 2PMM, 3PMM, 4PMM, 5PMM, and 6PMM had a proportional reduction in the overall computational time, which is close to the expected theoretical value obtained by dividing the multiplication time in the radix-2 MM algorithm by 2, 3, 4, 5, and 6 respectively.

The increase in the critical path length, as n increases is caused by the function that produces multiples of M . The hardware complexity for each partition increases as more partitions are used, and the critical path delay is augmented as a result. The blocks *LogicQi* and *SelOp2* are the ones that most affect the critical path delay.

Figure 19 shows the multiplication time given the same clock period, for each value of n . Figure 20 shows the impact of the number of partitions and the word size in the multiplier area. The method has an $O(n)$ area growth with the number of operands bits and the number of partitions.

Figure 19: Comparison in terms of the multiplication time

Figure 20: Comparison in terms of the total area

Figure 21 shows the most significant result. When comparing the established base-line architecture (Radix-2 MM) against the experiments on k PMM architectures, we observed a 27% average reduction in the energy consumption for the k PMM circuits described in Table 9, which cover moduli with 256, 512, 1024 and 2048 bits.

For the technology cell library used, the values of the cell leakage power are significantly less than the dynamic power. Figure 22 shows the average distribution

Figure 21: Comparison in terms of the energy consumption

of power between leakage and dynamic for each moduli and illustrates that the contribution of the leakage power increases due to the increase in the circuit area.

It should be noted that the integrated circuit industry claims that the leakage power is projected to dominate the overall power consumption, given the trend towards high performance and high density which requires smaller geometries (SYLVESTER; KAUL, 2001). Therefore, given this trend of increased participation of the leakage power in the future technology, the proposed $k\text{PMM}$ hardware may have reduced energy benefits, but the computation speed up will continue to be a strong advantage of this approach. For the technology cell library used herein, the $k\text{PMM}$ works well. However, for the latest technologies, the energy gain will most likely be smaller. Thus, the application of this solution should consider the target technology.

Figure 22: Dynamic power versus leakage power – $k\text{PMM}$ Architecture

7.1.2 Analysis of the Energy Consumption

The analysis of the energy consumption was achievable by using Power Compiler, which is part of the Synopsys Design Compiler synthesis tools. It performs both RTL and gate-level power optimization and gate-level power analysis. By applying Power Compiler various power reduction techniques, including clock-gating, operand isolation, multi-voltage leakage power optimization, and gate-level power optimization, can increase the power savings and the area and timing optimization in the front-end synthesis domain. The Power Compiler methodology (NEDELICHEV, 1997), the library models and the analysis technology are described in (SYNOPSYS, 2012).

Based on the experiments, it is possible to obtain an analytical model of the energy consumption for the Parallel k -Partition MM architecture. The model contemplates only the essential blocks of the k PMM. The energy consumption values depend on the cell library used for synthesis.

First, we calculate the energy consumption of a partition of the MMP architecture (E_{MMP}), such that 1 milliwatt (mW) is consumed in 1 microsecond (μs) when processing an n -bit input.

The average value of the total power for a given MMP block is denoted by the average power consumption of its modules, as shown in the following equation:

$$P_{MMP} = P_{SumReg} + P_{CarryReg} + P_{OpSel1} + P_{CSA1} + P_{OpSel2} + P_{CSA2} + P_{sCSA} \quad (mW) \quad (7.1)$$

Additionally, the following equations are the average values observed of the average power consumption for the common blocks ($ShiftX$, $AdderCS$) from k PMM.

$$P_{ShiftX} = P_{ShiftXReg} \quad (mW) \quad (7.2)$$

$$P_{AdderCS} = P_{SelP} + P_{CSReg} + P_{CSA3} \quad (mW) \quad (7.3)$$

By fitting a linear regression model (FREEDMAN, 2005), it is possible to predict the values of P_{MMP} , P_{ShiftX} , and $P_{AdderCS}$. These values depend on the choice of k partitions and n input bits of the k PMM and the clock period adopted (55 ns). With statistical significance defined as $p < 0.05$, the following equation adequately fits the experimental results.

$$P_{MMP}(b, k) = b(\alpha_0 + \alpha_1 k) \quad (mW) \quad (7.4)$$

$$P_{ShiftX}(b, k) = b(\beta_0 + \beta_1 k) \quad (mW) \quad (7.5)$$

$$P_{AdderCS}(b, k) = b(\gamma_0 + \gamma_1 k) \quad (mW) \quad (7.6)$$

where $b = 1, 2, 4$, or 8 , if $n = 256, 512, 1024$, or 2048 bits respectively, $\alpha_0 = 0.7210$, $\alpha_1 = 0.08241$, $\beta_0 = 0.4349$, $\beta_1 = 0.00001$, $\gamma_0 = 0.5861$, and $\gamma_1 = 0.01195$.

Consequently, given the multiplication time ($T_{MMP} = n/k$ clock cycles) to process n input bits, the computational time to accumulate the CS outputs of all the partitions ($T_{AdderCS} = k - 1$ clock cycles), and the nanoseconds into microseconds conversion (dividing by 1000), the energy consumption of a given MMP block among k partitions and the common blocks ($ShiftX$, $AdderCS$) of the k PMM can be represented by the following equations:

$$E_{MMP} = P_{MMP}(T_{MMP})/1000 \quad (mW-\mu s) \quad (7.7)$$

$$E_{ShiftX} = P_{ShiftX}(T_{MMP})/1000 \quad (mW-\mu s) \quad (7.8)$$

$$E_{AdderCS} = P_{AdderCS}(T_{AdderCS})/1000 \quad (mW-\mu s) \quad (7.9)$$

Finally, the energy consumption of the fully parallel k PMM architecture can be represented as follows:

$$E_{kPMM} = E_{ShiftX} + k(E_{MMP}) + E_{AdderCS} \quad (mW-\mu s) \quad (7.10)$$

Figure 23 is based on equation (7.10). Four trend curves for different n input bits

are displayed as the energy consumption of the k PMM related to the 16 different k partitions. This chart presents the expected energy growth, as the parallelism MMP is done, which are average values of the energy consumption per k PMM that were observed in several experiments, as shown in the summary in Table 9.

Figure 23: The impact of the number of partitions on the energy consumption

7.2 Montgomery Exponentiation in RNS

This section presents the results of the Montgomery Exponentiation in the RNS architecture implementation, the benchmark condition, and the analysis of energy consumption.

7.2.1 Benchmark

We based our baseline architecture on the Exponentiation Radix-2 MM (MExpRadix-2 – Algorithm 2) and showed that the Montgomery Exponentiation in the RNS architecture (MEXPRNS – Algorithm 8) outperforms it for different sizes of problem on the architectures which were tested.

Table 10 shows the summary of the experiments used for the comparison of the two architectures, which were implemented for four different values of n (256, 512,

Table 10: The Summary of the Report Timing, the Area, and the Energy Consumption of The Montgomery Exponentiation in RNS Architectures

n bits	Architecture	Exponentiation Time (ms)	Total Area (mm ²)	Total Area Increase (%)	Total Power per bit Exponent (mW)	Energy Consumption for one Montgomery Exponentiation (mW-ms)	Reduction Energy Consumption (%)
256	MExpRadix-2	1.31	0.23		2.87	0.19	
	MEXPRNS	0.66	0.37	58%	3.04	0.10	47%
512	MExpRadix-2	9.18	0.47		5.70	1.50	
	MEXPRNS	4.59	0.75	60%	6.17	0.81	46%
1024	MExpRadix-2	47.19	0.96		11.25	11.80	
	MEXPRNS	23.59	1.55	61%	13.10	6.87	42%
2048	MExpRadix-2	230.69	1.94		22.31	93.58	
	MEXPRNS	115.34	3.19	65%	25.81	54.14	42%

1024, and 2048 bits). The hardware synthesis were performed using the Synopsys Design Compiler, with a clock period value adopted for each value of n (20, 35, 45, and 55 ns, respectively). The clock period value adopted is not less than the largest critical path of any architecture to calculate the total area, the dynamic power, and the leakage power. For a given operand size (n) it is required to set the equivalent clock period value to determine the dynamic power, which allows the calculation of the energy consumption of all architectures. We compare the following parameters to perform a complete Montgomery Exponentiation:

- Number of MEXP clock cycles = $\frac{n \text{ bits}}{\text{number of Modulo Channels}}$;
- Exponentiation Time = clock period $\times$ number of MEXP clock cycles $\times n$ bits of exponent;
- Total Area = Combinational + Non-combinational + Net Interconnect Area;
- Dynamic Power = Cell Internal + Net Switching Power;
- Total Power per Exponent bit = Dynamic Power + Leakage Power;
- Energy Consumption for one Montgomery Exponentiation = Exponentiation Time $\times$ Total Power per Exponent bit.

The total area and the total area increase columns show the impact of the number of the Modulo Channels in the MM Extended architecture area. Although the experiments were performed with only two Modulo Channels (p and q), the MEXPRNS method has an expected $O(n)$ area growth with the number of operands bits and the number of partitions.

The energy consumption for one Montgomery Exponentiation and reduction energy consumption columns show the most significant result. When comparing the established baseline architecture (MExpRadix-2) against the experiments on the MEXPRNS architectures, we observed a 44% average reduction in the energy consumption for the MEXPRNS circuits described in Table 10.

7.2.2 Analysis of the Energy Consumption

The analysis of the energy consumption was achievable as described in Section 7.1.2 by using the Power Compiler and its methodology (NEDELCEV, 1997), the library models and the analysis technology (SYNOPSYS, 2012).

An analytical model of the energy consumption for the Montgomery Exponentiation in RNS is based on the experiments. The model considers only the relevant power consumption blocks of the MEXPRNS architecture.

We calculate the energy consumption of the MEXPRNS architecture with two Modulo Channels ($E_{MEXPRNS}$), such that 1 milliwatt (mW) is consumed in 1 micro-second (μs) when processing $z \equiv x^e \pmod{M}$, with n bits in each input operand.

The total power for a given MME block is denoted by the average power consumption of its modules, as shown in the following equation:

$$P_{MME} = 2P_{SumReg} + 2P_{CarryReg} + 2P_{CSA1} + 2P_{CSA2} + P_{ShiftX} + P_{DualAdder} \quad (mW) \quad (7.11)$$

Figure 24: The average power consumption blocks of the MEXPRNS architecture

Additionally, the following equations represent the average values observed of the average power consumption for the common blocks (FC , ME , RC) from MEXPRNS.

$$P_{FC} = P_{FCMux} \quad (mW) \quad (7.12)$$

$$P_{ME} = P_{MEMux} + P_{UReg} \quad (mW) \quad (7.13)$$

$$P_{RC} = P_{FCMux} + P_{Adder} \quad (mW) \quad (7.14)$$

Therefore, given the exponentiation time ($T_{MEXPRNS}$) to process n input bits, the energy consumption for one Montgomery Exponentiation with two Modulo Channels and the common blocks (FC , ME , RC) can be represented by the following equation:

$$\begin{aligned}
 E_{MEXPRNS} &= 2P_{MEXPRNS}(T_{MEXPRNS})/1000 + \\
 &= P_{FC}(T_{MEXPRNS})/1000 + \\
 &= P_{ME}(T_{MEXPRNS})/1000 + \\
 &= P_{RC}(T_{MEXPRNS})/1000 \quad (mW-\mu s)
 \end{aligned} \quad (7.15)$$

Based on the experiments, Figure 24 shows each major block of the MEXPRNS architecture.

8 FUTURE WORK

This chapter presents some enhancements for future work. The opportunities to continue this research are various. The following sections describe the relevant topics that should be considered.

8.1 k -Partition Architecture - Further Improvements

The proposed k -partition method allows changes in its architecture to consider two or more bits of X rather than one bit per partition. Thus, the number of partitions is reduced, and the addition of partial results is simplified with respect to the proposed Algorithm 6. In this sense, we should generalize the proposal, by showing how the multiplier operands can be decomposed into w -bit digits, as shown in the following figure:

Figure 25: The distribution of bits of X in w -bit digits

Some adjustments are necessary to compute multiples of Y , to calculate multiples of M , and to accumulate those multiples. The new way to split the bits of X into other

X_{P_j} multiplier operands is represented by the following equation:

$$X_{P_j} = \sum_{i=0}^{n/(kw)-1} \sum_{l=0}^{w-1} x_{jw+ikw+l} 2^{jw+ikw+l}, \quad (8.1)$$

with $0 < kw, t < n$, and $kwt = n$.

A further reduction in power may be obtained with other implementation improvements such as more aggressive clock-gating. The implementation may be made more flexible to handle a variety of operand precisions with the use of scalable architectures. There are thus several possible alternatives that can be pursued to accomplish design goals other than the basic architecture described in this work.

8.2 k PMM Architecture with Spare Module

Support to fault tolerance is achieved with the k PMM architecture with a spare module when we give it the capability to swap a faulty MMP with a spare reconfigurable MMP (called the Spare MMP). The partitioning process of the uniform k -Partition method leads to an easier implementation of a reconfigurable system, which enables the realization of a fault-tolerant hardware. This research will be briefly described in the following subsections.

8.2.1 A Spare MMP

Each MM Partition was wired in Figure 7 to work as a specific partition number (handling a particular bit inside a k -bit group of X), and its architecture can be modified to perform the computation of any partition. Once such a design is available, one or more Spare MMPs can be added to the multiplier and reconfigured to perform the function of any MMP that fails.

Hence, when a fault in one MMP is detected a Spare MMP can be brought up with an appropriate reconfiguration in the multiplier to provide inputs and read the output

of the new module.

8.2.2 Fault Tolerant k PMM Architecture

The generalization of the MMP architecture requires some adjustments in the *SelOp1* function to handle different multiples of Y , depending on the given partition p that it is targeted for replacement. A multiplexer can be used to shift the Y value left by p bit positions with a p value in the range $[0, k - 1]$.

One or more Spare MMPs can remain idle during normal operation until the occurrence of a fault. However, a Spare MMP can be used as a checker for the correct operation of other MMP modules. This checker module may be used in a round-robin scheduling mechanism, and the outputs of a different module in each cycle can be compared to increase the fault detection coverage.

The proposed fault tolerant fully parallel k PMM architecture with a Spare MMP is shown in Figure 26. The Spare MMP replaces a given faulty MMP and produces its partial modular multiplication in CS form (S_{outbk}, C_{outbk}).

The activation of the Spare MMP and the deactivation of a faulty MMP is performed by the $(k + 1)$ -bit input of the configuration spare in the signal *FaultyMMP*. If the bit *FaultyMMP*[k] = 1 a fault occurred in MMP_k , and it must be replaced by the Spare MMP. When a given faulty MMP block has its *FaultyMMP* signal set to 1, it is turned off to save energy. Likewise, when all k MMPs are working without errors, the Spare MMP is disabled and does not consume power.

Finally, when the multiplication is completed, the *AdderCS* performs the addition of partial results from MMPs operating correctly and discards the results from those turned off (either the Spare or the faulty MMP).

Figure 26: Fault-Tolerant Architecture using a reconfigurable MM Partition

8.2.3 External Fault Detection

Fault detection can be performed by the observation of incorrect results, which is recognized by the system using the fault tolerant parallel k PMM architecture.

One can determine the faulty MMP by using test vectors that have the bits of only one X_j set, which is manipulated by a particular MMP. The multiplication using these test vectors allows the determination of the faulty MMP.

8.3 Montgomery Exponentiation in RNS

The proposed MEXPRNS method allows to perform the Montgomery Exponentiation for an unlimited dynamic range (the product of the moduli set), using

only two different secret primes p and q to perform operations modulo M .

Future work should be dedicated on implementations and experiments using Montgomery Exponentiation in the RNS architecture for a moduli set that would be mapped to different Channels in the RNS processor. The study should determine if it will be advantageous to use a larger number of small channels, or a small number of large channels.

8.4 ECC

The proposed methods herein focus on modular multiplication, which can be used in point operations, in ECC.

As future work, to be more useful for ECC, these methods could be extended to work over two types of finite fields, either the prime Galois Field, $GF(p)$, or the binary extension Galois Field, $GF(2^m)$, or even support both fields (unified multiplier architecture (SAVAS; TENCA; KOC, 2000)).

In addition, the comparative analysis between ECC and RSA to identify the option and conditions for which one of those provide the best energy savings would be an entirely new work. The application of the multipliers proposed in the thesis should be considered in this type of work.

It should be noted that the legacy systems built on RSA will continue to exist for years, and security in embedded systems must be designed to deal with both ECC and RSA in multiple environments.

8.5 System Level Energy Characterization

Typically, mobile devices require high performance for short periods followed by relatively long idle periods, for example, to establish a secure communication channel

using PKC.

This thesis presents research on computer arithmetic and its application to public-key cryptography for low power consumption and high performance.

Future work should be extended to include an energy consumption characterization across many layers in mobile devices, for example, at the circuit, architecture, and algorithm levels, for potential energy savings.

8.6 Physical Security

The physical implementation of cryptographic algorithms can leak information about secret data to an attacker through side-channel attacks, e.g., fluctuations in power consumption or electromagnetic radiation (KOCHER, 1996), (KOCHER; JAFFE; JUN, 1999), (AGRAWAL et al., 2003). Techniques to prevent these attacks are currently being developed (MESSERGES, 2000), (MAY; MULLER; SMART, 2001), (STANDAERT et al., 2006). One area of research could be to study how these techniques can be applied for the low-power hardware implementations presented herein, without exceeding the power and area limitations.

Moreover, the future work related to prevent attacks should also address fault attacks. These attacks and the countermeasures are still not very well understood.

9 CONCLUSIONS

In this thesis we consider algorithms for low-power hardware implementations. We investigated the operations required for hardware implementations of the modular exponentiation and modular multiplication and created an efficient hardware architecture to reduce the energy consumption without sacrificing performance with the use of arithmetic functions to perform the calculations involved in public-key cryptography.

9.1 Research Contributions

The major contributions of this thesis are as follows:

- (a) In Chapter 5 the k -Partition Montgomery Multiplication method is proposed for low-power hardware implementations. In addition, our investigations were conducted to provide an application of the Montgomery Multiplication in RNS to compute $z \equiv x^e \pmod{M}$. Detailed analysis on the correctness and asymptotic analysis of the proposed methods are proved.
- (b) A proof of concept for the RSA cryptosystem implementation using the two architectures of the k -Parallel Montgomery Multiplication Partition and the Montgomery Exponentiation in RNS are provided in Chapters 6 and 7.

In Chapters 2, 3 and 4 are also shown a set of research challenges related to the themes that were conducted by this thesis as follows:

1. In Chapter 2, a survey of the literature on low-power hardware, power consumption, the sources for power consumption in digital circuits, some methods for limiting power consumption, and the detailed methodologies for low-power design is provided.
2. The concepts of Montgomery reduction, the general methods for the Montgomery algorithm, the parallel Montgomery Exponentiation, and some strategies for low-power design and implementation are presented in Chapter 3.
3. A survey on RNS and its usage in hardware applications is described in Chapter 4. Furthermore, we review the basics concepts of RNS, and then we derive a method to compute the Parallel Montgomery Multiplication algorithm in RNS. We propose an implementation to optimize modular multipliers for low power and high performance.

9.2 Publications

The following publications and papers in review were produced as results of the research effort during this thesis.

1. Conference Papers: in (NÉTO; TENCA; RUGGIERO, 2010), we describe a method to generate efficient implementations of sequential Montgomery multiplication. An efficient solution is obtained when inactive adders in a cycle are re-assigned to perform useful computation. The resulting hardware algorithm and architecture accelerate the modular multiplication by looking ahead of the input data of two iterations and, in some cases, compressing two iterations into one, without increasing the iteration time too much. Experiments show a 33.6% average reduction in clock cycles when the proposed multiplier is applied to implement modular exponentiation in the 2048-bit RSA cryptosystem.

2. Conference Papers: in (NÉTO; TENCA; RUGGIERO, 2011), we present a short proposal of a new approach to speed up the Montgomery multiplication by distributing the multiplier operand bits into k partitions that can process in parallel. In addition to the gain in speed, the approach provides a 20% average reduction in energy consumption for multiplication operands with 256, 512, 1024, and 2048 bits.
3. Journal Articles: (NÉTO; TENCA; RUGGIERO, 2012, Accepted for publication) is under review at the IEEE Transactions on Computers. We proposed an extension of a previous study (NÉTO; TENCA; RUGGIERO, 2011), where a detailed analysis on the correctness of the partitioning method is presented. The power consumption demanded by conventional carry-save (CS) representation is reduced by using a flexible sparse CS representation. The complexity and the energy consumption evaluation of the proposed architecture are shown. In addition, extended experiments on the fully parallel architecture implementation were performed. Furthermore, a fault-tolerant hardware extending the proposed method is presented and discussed.

REFERENCES

ABDALLAH, M.; SKAVANTZOS, A. A systematic approach for selecting practical moduli sets for residue number systems. In: *Proceedings of the 27th Southeastern Symposium on System Theory (SSST'95)*. Washington, DC, USA: IEEE Computer Society, 1995. p. 445–449. ISBN 0-8186-6985-3.

AGRAWAL, D. et al. The EM Side-Channel(s). In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Redwood Shores, CA, USA: Springer, 2003. v. 2523, p. 29–45. ISBN 978-3-540-00409-7.

AHUJA, S.; LAKSHMINARAYANA, A.; SHUKLA, S. *Low Power Design with High-level Power Estimation and Power-aware Synthesis*. Springer New York, 2012. ISBN 9781461408727.

AKKAL, M.; SIY, P. A new mixed radix conversion algorithm MRC-II. *J. Syst. Archit.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 53, n. 9, p. 577–586, 2007. ISSN 1383-7621.

ALIDINA, M. et al. Precomputation-based sequential logic optimization for low power. In: *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994. (ICCAD '94), p. 74–81. ISBN 0-89791-690-5.

AMBERG, P.; PINCKNEY, N.; HARRIS, D. M. Parallel high-radix Montgomery multipliers. In: *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*. Monterey, CA, USA: IEEE, 2008. p. 772–776. ISSN 1058-6393.

ASKARZADEH, M.; HOSSEINZADEH, M.; NAVI, K. A new approach to overflow detection in moduli set $\{2^n - 3, 2^n - 1, 2^n + 1, 2^n + 3\}$. In: *Proceedings of the 2009 Second International Conference on Computer and Electrical Engineering - Volume 01*. Washington, DC, USA: IEEE Computer Society, 2009. (ICCEE '09), p. 439–442. ISBN 978-0-7695-3925-6.

BAJARD, J.-C.; DIDIER, L.-S.; KORNERUP, P. Modular multiplication and base extensions in residue number systems. In: *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*. Washington, DC, USA: IEEE Computer Society, 2001. (ARITH '01), p. 59.

BAJARD, J.-C. et al. Residue systems efficiency for modular products summation: Application to elliptic curves cryptography. In: *Proc. Advanced Signal Processing Algorithms, Architectures, and Implementations XVI*. San Diego, California, USA: SPIE, 2006. v. 6313. ISBN 9780819463920.

- BAJARD, J.-C.; IMBERT, L. A full RNS implementation of RSA. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 53, n. 6, p. 769–774, jun. 2004. ISSN 0018-9340.
- BAJARD, J.-C.; KAIHARA, M.; PLANTARD, T. Selected RNS bases for modular multiplication. *Computer Arithmetic, IEEE Symposium on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 25–32, 2009. ISSN 1063-6889.
- BAJARD, J.-C.; MELONI, N.; PLANTARD, T. Efficient RNS bases for cryptography. In: *Proceedings of IMACS 2005 World Congress*. Paris, France, 2005.
- BARKER, E. B.; JOHNSON, D.; SMID, M. E. *SP 800-56A. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (Revised)*. Gaithersburg, MD, United States, 2007.
- BARRETT, P. *Communications authentication and security using public key encryption - A design for implementation*. Master's thesis, Oxford University, September 1984.
- BENINI, L.; MICHELI, G. D. *Dynamic power management - Design techniques and CAD tools*. Kluwer Academic Publishers, 1998. ISBN 978-0-7923-8086-3.
- BEUCHAT, J.-L.; MULLER, J.-M. Automatic generation of modular multipliers for FPGA applications. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 57, n. 12, p. 1600–1613, dez. 2008. ISSN 0018-9340.
- BI, G.; JONES, E. Fast conversion between binary and residue numbers. *Electronics Letters*, v. 24, n. 19, p. 1195–1197, sep 1988. ISSN 0013-5194.
- BLAKE, I. et al. *Advances in elliptic curve cryptography*. New York, NY, USA: Cambridge University Press, 2005. ISBN 052160415X.
- CAO, B.; CHANG, C.-H.; SRIKANTHAN, T. A residue-to-binary converter for a new five-moduli set. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, v. 54, n. 5, p. 1041–1049, 2007. ISSN 1549-8328.
- CHIOU, C. W. Parallel implementation of the RSA public-key cryptosystem. *International Journal of Computer Mathematics*, v. 48, n. 3-4, p. 153–155, 1993.
- CIET, M. et al. Parallel FPGA implementation of RSA with residue number systems - Can side-channel threats be avoided? In: . Cairo, Egypt: In 46th IEEE Intl Midwest Symposium on Circuits and Systems, 2003. p. 806–810.
- CORMEN, T. H. et al. *Introduction to algorithms (3rd ed.)*. The MIT Press, 2009. I-XIX, 1-1292 p. ISBN 978-0-262-03384-8.
- DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976.
- ELGAMAL, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, v. 31, n. 4, p. 469–472, 1985.

FREEDMAN, D. *Statistical models: theory and practice*. Cambridge University Press, 2005. Hardcover. ISBN 0521854830.

GARNER, H. L. The residue number system. *IEEE Trans. Electronic Computers*, v. 8, p. 140–147, 1959.

GORDON, D. M. A survey of fast exponentiation methods. *Journal of Algorithms*, v. 27, n. 1, p. 129 – 146, 1998. ISSN 0196-6774.

GRAMA, A. et al. *Introduction to parallel computing: design and analysis of algorithms*. Addison-Wesley, 2003. ISBN 0201648652.

HANKERSON, D.; MENEZES, A. J.; VANSTONE, S. *Guide to elliptic curve cryptography*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003. ISBN 038795273X.

HITZ, M. A.; KALTOFEN, E. Integer division in residue number systems. *IEEE Trans. Computers*, v. 44, n. 8, p. 983–989, 1995.

HOSSEINZADEH, M.; NAVI, K.; GORGIN, S. A new moduli set for residue number system: $\{r^n - 2, r^n - 1, r^n\}$. In: *Electrical Engineering, 2007. ICEE '07. International Conference on*. Hong Kong: IAENG, 2007. p. 1 –6.

HUANG, C. A fully parallel mixed-radix conversion algorithm for residue number applications. *Computers, IEEE Transactions on*, C-32, n. 4, p. 398–402, 1983. ISSN 0018-9340.

HUNG, C. Y.; PARHAMI, B. Fast RNS division algorithms for fixed divisors with application to RSA encryption. *Inf. Process. Lett.*, v. 51, n. 4, p. 163–169, 1994.

IEEE STD 1363. *Standard specifications for public key cryptography*. 2000. 1-227 p.

IEEE STD 1801. *Standard for design and verification of low power integrated circuits*. 2009. 1-218 p.

IWAMURA, K.; MATSUMOTO, T.; IMAI, H. Systolic-arrays for modular exponentiation using Montgomery method. In: RUEPPEL, R. (Ed.). *Advances in Cryptology – EUROCRYPT'92*. Balatonfüred, Hungary: Springer Berlin Heidelberg, 1993. v. 658, p. 477–481. ISBN 978-3-540-56413-3.

KAIHARA, M.; TAKAGI, N. Bipartite modular multiplication method. *Computers, IEEE Transactions on*, v. 57, n. 2, p. 157 –164, feb. 2008. ISSN 0018-9340.

KAWAMURA, S. et al. Cox-rower architecture for fast parallel Montgomery multiplication. In: *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*. Berlin, Heidelberg: Springer-Verlag, 2000. (EUROCRYPT'00), p. 523–538. ISBN 3-540-67517-5.

KNUTH, D. E. *The art of computer programming, V. II: Seminumerical Algorithms*, 2nd Ed., Addison-Wesley, 1981. ISBN 0-201-03822-6.

KOC, C. K. A fast algorithm for mixed-radix conversion in residue arithmetic. In: *Computer Design: VLSI in Computers and Processors, 1989. ICCD '89. Proceedings., 1989 IEEE International Conference on*. Cambridge, MA, USA: IEEE, 1989. p. 18–21.

KOC, C. K.; ACAR, T.; KALISKI JR., B. S. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, v. 16, n. 3, p. 26–33, 1996.

KOCHER, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1996. p. 104–113. ISBN 3-540-61512-1.

KOCHER, P. C.; JAFFE, J.; JUN, B. Differential power analysis. In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1999. p. 388–397. ISBN 3-540-66347-9.

KORTHIKANTI, V. A.; AGHA, G. Analysis of parallel algorithms for energy conservation in scalable multicore architectures. In: *Proceedings of the 2009 International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2009. p. 212–219. ISBN 978-0-7695-3802-0.

_____. Energy-performance trade-off analysis of parallel algorithms for shared memory architectures. *Sustainable computing: informatics and systems*, v. 1, n. 3, p. 167 – 176, 2011. ISSN 2210-5379.

LEISERSON, C. E.; SAXE, J. B. Retiming synchronous circuitry. *Algorithmica*, v. 6, n. 1, p. 5–35, 1991.

LEU, J.-J.; WU, A.-Y. Design methodology for Booth-encoded Montgomery module design for RSA cryptosystem. In: *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*. Geneva, Switzerland: IEEE, 2000. v. 5, p. 357–360.

LI, J.; MARTINEZ, J. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In: *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, p. 77–87, 2006. ISSN 1530-0897.

LIM, Z.; PHILLIPS, B. An RNS-enhanced microprocessor implementation of public key cryptography. In: *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*. Monterey, CA, USA: IEEE, 2007. p. 1430 –1434. ISSN 1058-6393.

MAY, D.; MULLER, H. L.; SMART, N. P. Random register renaming to foil DPA. In: *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2001. p. 28–38. ISBN 3-540-42521-7.

MENEZES, A.; OORSCHOT, P. C. v.; VANSTONE, S. A. *Handbook of applied cryptography*. CRC Press, 1996. ISBN 0-8493-8523-7.

MESSERGES, T. S. *Power analysis attacks and countermeasures for cryptographic algorithms*. Phd thesis, Chicago, IL, USA, 2000.

MOHAN, P.; PREMKUMAR, A. RNS-to-binary converters for two four-moduli sets $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ and $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, v. 54, n. 6, p. 1245–1254, 2007. ISSN 1549-8328.

MÖLLER, B. Improved techniques for fast exponentiation. In: *Proceedings of the 5th international conference on Information security and cryptology*. Berlin, Heidelberg: Springer-Verlag, 2003. (ICISC'02), p. 298–312. ISBN 3-540-00716-4.

MONTEIRO, J.; DEVADAS, S.; GHOSH, A. Retiming sequential circuits for low power. In: *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993. (ICCAD '93), p. 398–402. ISBN 0-8186-4490-7.

MONTGOMERY, P. L. Modular multiplication without trial division. *Mathematics of Computation*, v. 44, n. 170, p. 519–521, abr. 1985.

NEDELICHEV, I. Power compiler: a gate-level power optimization and synthesis system. In: *Proceedings of the 1997 International Conference on Computer Design (ICCD '97)*. Washington, DC, USA: IEEE Computer Society, 1997. p. 74–79. ISBN 0-8186-8206-X.

NEDJAH, N.; MOURELLE, L. M. *Embedded cryptographic hardware: methodologies and architectures*. Nova Science Publishers, 2004. ISBN 1594540128.

NÉTO, J. C.; TENCA, A. F.; RUGGIERO, W. V. Towards an efficient implementation of sequential Montgomery multiplication. In: *Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on*. Monterey, CA, USA: IEEE, 2010. p. 1680–1684. ISSN 1058-6393.

_____. A parallel k -partition method to perform Montgomery multiplication. In: *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*. Santa Monica, CA, USA: IEEE, 2011. p. 251–254. ISSN 2160-0511.

_____. A parallel and uniform k -partition method for Montgomery multiplication. *IEEE Transactions on Computers*, IEEE Computer Society, 2012, Accepted for publication.

NIST. *Federal information processing standard (FIPS PUB 186-3) - Digital Signature Algorithm (DSA)*. 2009.

NOZAKI, H. et al. Implementation of RSA algorithm based on RNS Montgomery multiplication. In: *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2001. (CHES '01), p. 364–376. ISBN 3-540-42521-7.

OMONDI, A.; PREMKUMAR, B. *Residue number systems: theory and implementation*. London, UK, UK: Imperial College Press, 2007. ISBN 1860948669.

PARHAMI, B. *Introduction to parallel processing: algorithms and architectures*. Norwell, MA, USA: Kluwer Academic Publishers, 1999. ISBN 0306459701.

_____. RNS representation with redundant residues. In: *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*. Monterey, CA, USA: IEEE, 2001. v. 2, p. 1651–1655. ISSN 1058-6393.

PEDRAM, M. *Power aware design methodologies*. Norwell, MA, USA: Kluwer Academic Publishers, 2002. ISBN 1402071523.

PEDRAM, M.; ABDOLLAHI, A. Low-power RT-level synthesis techniques: a tutorial. *IEE Proc. on Computers and Digital Techniques*, v. 152, n. 3, p. 333 – 343, may 2005. ISSN 1350-2387.

PIGUET, C. *Low-power electronics design*. CRC Press, 2004. (Computer Engineering). ISBN 9780849319419.

POUWELSE, J.; LANGENDOEN, K.; SIPS, H. Dynamic voltage scaling on a low-power microprocessor. In: *Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2001. p. 251–259. ISBN 1-58113-422-3.

RABAEY, J. M.; PEDRAM, M. *Low power design methodologies*. Kluwer Academic, 1996. ISBN 9780792396307.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. M. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, v. 21, n. 2, p. 120–126, 1978.

RSA LABORATORIES. *PKCS #1 v2.0 Amendment 1: Multi-Prime RSA*. July 2000. <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-0a1.pdf>.

SAKIYAMA, K. et al. Tripartite modular multiplication. *Integration*, v. 44, n. 4, p. 259–269, 2011.

SAVAS, E.; TENCA, A. F.; KOC, C. K. A scalable and unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$. In: *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK, UK: Springer-Verlag, 2000. (CHES '00), p. 277–292. ISBN 3-540-41455-X.

SCHINIYAKIS, D. M. et al. An RNS implementation of an f_p elliptic curve point multiplier. *Trans. Cir. Sys. Part I*, IEEE Press, Piscataway, NJ, USA, v. 56, n. 6, p. 1202–1213, 2009. ISSN 1549-8328.

SECG. SEC 1. *Elliptic curve cryptography, Version 2.0*. Standards for Efficient Cryptography Group, 2009.

- SODERSTRAND, M. A. et al. (Ed.). *Residue number system arithmetic: modern applications in digital signal processing*. Piscataway, NJ, USA: IEEE Press, 1986. ISBN 0-87942-205-X.
- SOLINAS, J. A. *Generalized Mersenne numbers*. Technical Report CORR 99-39, Centre for Applied Cryptographic Research, The University of Waterloo, Ontario, Canada, 1999.
- STANDAERT, F.-X. et al. Towards security limits in side-channel attacks. In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Yokohama, Japan: Springer Berlin Heidelberg, 2006. v. 4249, p. 30–45. ISBN 978-3-540-46559-1.
- SYLVESTER, D.; KAUL, H. Future performance challenges in nanometer design. In: *Proceedings of the 38th annual Design Automation Conference*. New York, NY, USA: ACM, 2001. p. 3–8. ISBN 1-58113-297-2.
- SYNOPSYS. Power compiler user guide. *Synopsys Inc.*, June 2012.
- SZABO, N.; TANAKA, R. *Residue arithmetic and its applications to computer technology*. New York, USA: McGraw-Hill, 1967.
- TAYLOR, F. J. Residue arithmetic a tutorial with examples. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 17, n. 5, p. 50–62, 1984. ISSN 0018-9162.
- TENCA, A.; KOC, C. A scalable architecture for modular multiplication based on Montgomery's algorithm. *Computers, IEEE Transactions on*, v. 52, n. 9, p. 1215 – 1221, sept. 2003. ISSN 0018-9340.
- TIWARI, V.; MALIK, S.; ASHAR, P. Guarded evaluation: pushing power management to logic synthesis/design. In: *Proceedings of the 1995 international symposium on Low power design*. New York, NY, USA: ACM, 1995. (ISLPED '95), p. 221–226. ISBN 0-89791-744-8.
- TODOROV, G. *ASIC design, implementation and analysis of a scalable high-radix Montgomery multiplier*. Master's thesis, Oregon State University, USA, December 2000.
- USAMI, K.; HOROWITZ, M. Clustered voltage scaling technique for low-power design. In: *Proceedings of the 1995 international symposium on Low power design*. New York, NY, USA: ACM, 1995. (ISLPED '95), p. 3–8. ISBN 0-89791-744-8.
- VINNAKOTA, B.; RAO, V. B. Fast conversion techniques for binary-residue number systems. *Circuits and Systems I: fundamental theory and applications, IEEE Transactions on*, v. 41, n. 12, p. 927–929, dec 1994. ISSN 1057-7122.
- WALTER, C. D. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, v. 35, n. 21, p. 1831 – 1832, oct 1999. ISSN 0013-5194.
- _____. Improved linear systolic array for fast modular exponentiation. *IEE Proceedings: Computers and Digital Techniques*, v. 147, n. 5, p. 323–328, 2000.

YEAP, G. K. *Practical low power digital VLSI design*. Springer, 1997. ISBN 0792380096.

YOSHINO, M.; OKEYA, K.; VUILLAUME, C. Faster double-size bipartite multiplication out of Montgomery multipliers. *IEICE Transactions*, v. 92-A, n. 8, p. 1851–1858, 2009.

ZHU, F. et al. Password authenticated key exchange based on RSA for imbalanced wireless networks. In: CHAN, A.; GLIGOR, V. (Ed.). *Information Security*. Springer Berlin Heidelberg, 2002. v. 2433, p. 150–161. ISBN 978-3-540-44270-7.

ZIMMERMANN, R. Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. In: *14th IEEE Symposium on Computer Arithmetic (Arith-14 99)*, Adelaide, Australia. IEEE Computer Society, 1999. p. 158–167. ISBN 0-7695-0116-8.