SERIAL-SERIAL FINITE FIELD MULTIPLICATION

by

Abdulaziz Mohammad Alkhoraidly

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

Dhahran, Saudi Arabia

In Partial Fulfillment of the Requirements

for the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

June 2005

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by Abdulaziz M. Alkhoraidly under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN COMPUTER SCIENCE.

Thesis Committee

Dr. Mohammed Alsuwaiyel
(Chairman)

Dr. Mohammad K. Ibrahim
(Co-chairman)

Dr. Khaled Salah
(Member)

Dr. Kanaan Faisal
(Department Chairman)

Dr. Jaralla Al-Ghamdi
(Member)

Dr. Mohammad Al-Ohali
(Dean of Graduate Studies)

Dr. Moataz A. Ahmed
(Member)

Date

Dedicated to my honorable father,

my warmhearted mother,

my ambitious brothers,

and to my lovely fiancée.

I owe you more than I can possibly express.

Acknowledgment

Indeed, all praise is due to Allah for His countless blessings. We praise Him, seek His aid and ask His forgiveness. We seek shelter in Allah from the evil of our own souls and the evil of our own actions.

I learnt a lot during the years I spent in King Fahd University of Petroleum and Minerals. All the graduates of this university carry deep feelings for it, and I am not an exception. I would like to take this chance to extend my appreciation and best wishes to the university, its administration, faculty, staff and students.

Moreover, I feel deeply indebted to my thesis advisor, Dr. Mohammed Alsuwaiyel, and to my thesis co-advisor, Dr. Mohammad K. Ibrahim, for the unconstrained support and patience they continuously exhibited since we first met. My gratitude is also due to the thesis committee members, Dr. Khaled Salah, Dr. Jaralla Al-Ghamdi and Dr. Moataz A. Ahmed for their help and enlightening comments.

Last but not least, sincere appreciation is due to my father, mother, brothers and fiancée, the people who keep supporting me unconditionally. I owe it all to them.

Contents

List of Tables

List of Figures

List of Algorithms

Abstract

NAME:	Abdulaziz Mohammad Alkhoraidly
TITLE:	Serial-serial Finite Field Multiplication
MAJOR FIELD:	Computer Science
DATE OF DEGREE:	June 2005

Finite field multiplication is an important primitive in many applications, among which cryptography the most notable. In order to provide a higher resistance against cryptanalysis, cryptographic parameters are continuously growing in size, which constitutes a serious problem for cryptosystems designers. Firstly, the growth of parameters increases the required time, area and the power consumption of cryptographic operations. Secondly, it renders useless any unscalable hardware components. Accordingly, it is essential to implement all primitive operations in a way that limits these effects.

Following these lines, this thesis introduces a novel formulation of finite field multiplication. In the new formulation, all inputs and outputs of the finite field multiplier are communicated in a serial manner, thus the name serial-serial. It is possible to modify any finite field multiplier to allow for serial-serial behavior (e.g. by adding serial-to-parallel and parallel-to-serial registers). However, our formulation exhibits the serial-serial behavior with no modification whatsoever to the multiplier interface and without affecting its performance. Moreover, we present some implementation enhancements that can be applied to the resulting structures.

ملخص الرسالة

الاسم : عبدالعزيز بن محمد بن عبدالعزيز الخريدلي

عنوان الرسالة : عملية الضرب التسلسلية-التسلسلية في الحقول المنتهية

التخصص : علوم الحاسب الآلي

تاريخ التخرج : يونيو ٢٠٠٥

تعتبر عملية الضرب في الحقول المنتهية من أهم العمليات الأساسية في الكثير من التطبيقات ومن أهمها تلك المتعلقة بعلم التعمية وأمن المعلومات، ومن الملاحظ أن الأرقام المستعملة في حسابات التعمية تزيد في الحجم باستمرار لمقاومة محاولات كسر التشفير، تعتبر هذه الزيادة في الحجم مشكلة لمصممي أجهزة التشفير لما يتبعها من زيادة في الوقت الذي تستغرقه العمليات الحسابية وكذلك في الطاقة المستهلكة، بالإضافة إلى ذلك فإن الأجهزة التي لم يراع في تصميمها قابليتها لمعالجة أرقام أكبر لا يمكن استعمالها عند تغير المواصفات القياسية للتعمية، تبعا لذلك فإنه من الضروري تصميم كل العمليات الأساسية بشكل يقلل قدر الإمكان من آثار تلك الزيادة.

وكأحد العلاجات لهذه المشكلة نقدم في هذه الرسالة صياغة جديدة لعملية الضرب في الحقول المنتهية، تتصف الصياغة الجديدة بأن كلا من المدخلات وناتج الضرب تنتقل بشكل تسلسلي (رقما رقما)، ولذا سميت بالصياغة التسلسلية-التسلسلية لتفريقها عن الصياغات التسلسلية الأخرى، ومن المعلوم أن كل الصياغات الأخرى لعملية الضرب في الحقول المنتهية يمكن تحويرها لتكون تسلسلية-تسلسلية، ولكن الصياغة التي نقدمها تتصف بهذه الصفة دون أي تحويرات أو إضافات على التصميم، ودون التأثير على أدائه، كذلك نقدم في هذه الرسالة بعض التحسينات التي يمكن تطبيقها على التصاميم الناتجة لتحسين خصائصها.

Chapter 1

Introduction

1.1 An Overview

Finite field multiplication is an important primitive that is employed in many applications, among which cryptography the most notable. Cryptographic parameters are continuously growing in size to provide a higher resistance against cryptanalysis, which constitutes a serious problem for cryptosystems designers. Firstly, the growth of parameters increases the execution time and the power consumption of cryptographic operations. Secondly, it renders useless any non-scalable hardware components. Accordingly, it is essential to implement all primitive operations in a way that limits these effects.

Following these lines, this thesis introduces a novel formulation that can solve one of the problems that are associated with the cryptographic applications, namely,

the width of the busses used to transfer the inputs and output of the multiplier. The growth of cryptographic parameters size influences directly the size of busses required to transfer them. The wider the busses, the more difficult the placement and routing on the chip becomes.

To counter this problem, our novel formulation exhibits a behavior that is purely serial-serial, i.e. all inputs and the output of the finite field multiplier are communicated serially, one digit at a time.

Obviously, it is possible to modify any available finite field multiplier to allow for serial-serial behavior (e.g. by adding serial-to-parallel and parallel-to-serial registers or by adding pipelining delays), resulting in increased time and/or area requirements. However, our formulation exhibits an inherent serial-serial behavior with no modification whatsoever to the multiplier interface and without sacrificing its performance (time and/or area) compared to other popular formulations. This has been achieved by manipulating the mathematical description of the finite field multiplication operation to get an inherently serial-serial formulation.

Then, we present some implementation enhancements that can be applied to the resulting structures to enhance various aspects of their performance. These enhancements are well-known in the field and many more implementation enhancements can be borrowed from available structures.

1.2 Scope of the Thesis

Finite field multiplication encapsulates two slightly different operations, namely, multiplication in prime fields and multiplication in extension fields. However, the deep structural similarity between these two operations allows for a unifying notation to be adopted and for a unified formulations to be worked out. Moreover, multiplication in a finite ring of integers (a.k.a. modular or modulo multiplication), which is the generic version of prime field multiplication, can be also included in the unifying notation. Accordingly, all the results of our work are applicable to any of these three operations unless otherwise stated.

These operations can be described in various levels of abstraction, like:

1. The mathematical description.
2. The functional description (a.k.a. dataflow).
3. The Time-space mapping of the dataflow to one or more functional units.
4. The implementation of functional units.

In this work, the time-space mapping of finite field multiplication dataflows to functional units will be addressed. The aim is to come up with a new time-space mapping that has the property of being inherently serial-serial without compromising its performance.

1.3 Main Contributions

The main contributions of this work are the following:

1. Proposing a unified formulation for finite field and modulo multiplication that is inherently serial-serial.
2. Designing a generic serial-serial finite field and modulo multiplier as a realization of the aforementioned formulation.
3. Proposing some implementation enhancements for the resulting structures.

1.4 Organization of the Thesis

This document proceeds as follows. Chapter 2 presents a brief but sufficient background on some related abstract algebraic concepts. Chapter 3 reviews some of the existing finite field multiplication formulations and implementations. Chapter 4 introduces and examines the novel serial-serial formulation. It also provides a generic design for a serial-serial finite field multiplier. Chapter 5 lists and discuses some implementation enhancements that can be applied to enhance various features of the resulting structures. Chapter 6 concludes this document and discusses some areas for future work.

Chapter 2

Algebraic Preliminaries

2.1 Introduction

This chapter aims to provide a brief, but sufficient, coverage of the mathematical concepts often referred to throughout this work. Abstract algebra textbooks, like [8] and [46], and some cryptography reference books, like [39], can be consulted for a more extensive treatment of these concepts. Moreover, a generalized polynomial representation for finite field elements is introduced as it is used throughout this thesis.

This chapter is organized as follows. Firstly, the key concepts of algebraic groups and rings are discussed with examples. These are used then to introduce fields with a discussion of their main features and types. After that, polynomial rings are discussed and used to explain extension finite fields. The chapter is closed by introducing a

generalized polynomial representation for finite field elements of different type.

2.2 Groups and Rings

In order to define finite fields, it is essential to introduce the notions of a group and a ring. A group is constructed on a set of elements using a binary operation that has specific properties. The following definition outlines these properties.

Definition 2.1 (Groups) *A group $(G, \times)$ is a set G associated with a binary operation $\times$ defined on G such that the following requirements are satisfied:*

1. Closure: *$a \times b \in G$ for all $a, b \in G$.*
2. Associativity: *$(a \times b) \times c = a \times (b \times c)$ for all $a, b, c \in G$.*
3. Identity: *there exists $e \in G$ such that $a \times e = e \times a = a$ for all $a \in G$.*
4. Inverse: *there exists $a^{-1} \in G$ such that $a \times a^{-1} = a^{-1} \times a = e$ for all $a \in G$.*

Furthermore, a group is called abelian (or commutative) if it satisfies the Commutivity *axiom, i.e. $a \times b = b \times a$ for all $a, b \in G$.*

This definition uses the *multiplicative* notation, in which the binary operation is denoted as $\times$ and the inverse of an element a is denoted as a^{-1}. This notation is commonly used when the binary operation resembles multiplication, e.g. multiplication modulo a positive integer. In such notation, the identity element is labeled "1".

In case the binary operation resembles addition, the *additive* notation is used instead where $+$ denotes the operation, $-a$ denotes the inverse of a, and the identity element is labeled "0".

An example of a group is the set of integers $\mathbb{Z}$ under addition, where the identity element is 0 and the inverse of an integer a is $-a$. A group can be finite or infinite based on the number of its elements. For a finite groups, the number of elements is referred to as the *group order*. On the other hand, the *order of an element* a in a finite group is the smallest integer c such that $a^c = e$, where e is the identity element. The order of a group is divisible by the order of any of its elements. A finite group of order n can be represented as an $n \times n$ table. If the group is abelian, the table would be diagonally symmetric.

Example 2.1 (A finite group) *The set* $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ *under the operation of addition modulo* 5 *is a finite group represented by table 2.1. It can be shown that* 0 *is the additive identity and that, for example, the inverse of the element* 4 *is* 1 *since* $4 + 1 = 0$.

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Table 2.1: $\mathbb{Z}_5$ under Addition Modulo 5

A ring can be created by augmenting a second binary operation to a group.

Definition 2.2 (Rings) *A ring $(R, +, \times)$ is a set R associated with two binary operations, $+$ and $\times$, defined on R such that the following requirements are satisfied:*

1. *$(R, +)$ is an abelian group.*
2. Closure under $\times$: *$a \times b \in R$ for all $a, b \in R$.*
3. Associativity under $\times$: *$(a \times b) \times c = a \times (b \times c)$ for all $a, b, c \in R$.*
4. Distributivity of $\times$ over $+$: *$a \times (b + c) = (a \times b) + (b \times c)$ for all $a, b, c \in R$.*

Furthermore, a ring is commutative if it satisfies the Commutivity *axiom, i.e. $a \times b = b \times a$ for all $a, b \in R$.*

In a ring R, if the operation $\times$ has an identity element, the ring is called a *ring with identity*. An example of a commutative ring is the set of integers $\mathbb{Z}$ under addition and multiplication. Another example is the set $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$, which is a commutative ring under the addition and multiplication modulo n.

2.3 Fields

Definition 2.3 (Fields) *A field F is a commutative ring with identity in which all the nonzero elements of F form an abelian group under multiplication.*

In other words, the field is a combination of two abelian groups. The first group is additive, containing all elements of the field, and has the identity of "0". The other

one is multiplicative, containing all elements but the "0", and has an identity of "1". While the set of rational numbers $\mathbb{Q}$ is an example of a field, $\mathbb{Z}$ is an example of a ring that is not a field, since the only elements that have multiplicative inverses are 1 and -1 .

A field that has a finite number of elements is called a *finite field*, or a *Galois field*, after the French 19th century mathematician Evariste Galois. A finite field is either a prime field or an extension field. As the name indicates, *prime fields* have orders of prime numbers. Accordingly, $\mathbb{Z}_n$, which has been already mentioned as a ring under addition and multiplication modulo n, is a field if and only if n is a prime number. A prime field can be denoted as $\mathbb{Z}_p$, $\mathbb{F}_p$ or $GF(p)$, where p is the order of the field.

An *extension field*, on the other hand, has an order of p^m, where p is a prime and $m > 1$. As its name indicates, such field is created by extending the prime field $\mathbb{F}_p$ where m denotes the *degree* of extension. Extension fields are denoted as $\mathbb{F}_{p^m}$ or $GF(p^m)$. A more elaborate discussion of extension fields will follow in section 2.5.

In general, a finite field can be denoted as $\mathbb{F}_q$ where $q = p^m$ and $m \geq 1$. The multiplicative group, i.e. nonzero elements, of $\mathbb{F}_q$ is denoted $\mathbb{F}_q^*$. An interesting fact is that all finite fields of the same order are *isomorphic*, i.e. have similar structures, even if they are represented differently. In other words, these fields can be made identical by only renaming their elements.

The simplest example of a finite field is $\mathbb{F}_2$ or $GF(2)$, since any field must have the elements "0" and "1" by definition. Table 2.2 shows the definition of addition and multiplication operation in $\mathbb{F}_2$. Addition in $\mathbb{F}_2$ is equivalent to logical XOR while

multiplication is equivalent to logical AND.

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Table 2.2: Addition and Multiplication in $\mathbb{F}_2$

2.4 Polynomials Rings

Before discussing the representation and operations of extension fields, it is essential to study polynomials defined over finite fields. Such polynomial can be written as

$$f(x) = \sum_{i=0}^{n} f_i \mu^i = f_0 + f_1 x + f_2 x^2 + \ldots + f_n x^n \tag{2.1}$$

where the coefficients f_i are elements of a finite field $\mathbb{F}_p$, x is an indeterminate, and deg $f(x) = n$ is a positive integer. Moreover, f_n is called the leading coefficient while f_0 is the constant term. The set of such polynomials is labeled $\mathbb{F}_p[x]$. Addition can be defined on the elements of this set as

$$f(x) + g(x) = \sum_{i=0}^{\max(\deg f, \deg g)} (f_i + g_i) x^i \tag{2.2}$$

It is important to emphasize that, since f_i and g_i are both elements of $\mathbb{F}_p$, their addition is performed mod p. In other words, there is no carry between terms of different degrees. Multiplication is defined in a similar way, namely,

$$f(x)g(x) = \sum_{i=0}^{\deg f} \left(\sum_{j=0}^{\deg g} (f_i g_j) x^{i+j} \right) \tag{2.3}$$

Accordingly, the degree of the product is deg f+deg g. It has been shown that $\mathbb{F}_p[x]$ defines an *infinite commutative ring* under polynomial addition and multiplication.

Polynomial operations can be performed modulo a given polynomial, e.g. $f(x)$. In this case, the result of conventional polynomial addition or multiplication has to be reduced by division over $f(x)$ and the remainder of division is reported as the result of the operation. The resulting set, along with the modular polynomial addition and multiplication, defines a finite commutative ring that is denoted $\mathbb{F}_p[x]/f(x)$.

Example 2.2 (Modular Polynomial Multiplication) *Let* $a(x) = x^3 + x + 1$ *and* $b(x) = x^2 + 1$ *in* $\mathbb{F}_2[x]/f(x)$ *where* $f(x) = x^4 + 1$. *It is required to calculate* $a(x)b(x)$. *Looking at* $f(x)$, *note that* $x^4 = -1 = 1$. *With all operations on coefficients performed modulo 2:*

$$\begin{aligned}(x^3 + x + 1)(x^2 + 1) &= x^5 + x^2 + x + 1 \\ &= x(x^4 + 1) + x^2 + 1 \\ &= x^2 + 1\end{aligned}$$

A polynomial in $\mathbb{F}_p[x]$ that is not divisible by any other polynomial in that ring is said to be *prime* or *irreducible* polynomial. Table 2.3 shows the factorization of all the polynomials over $\mathbb{F}_2$ of degree $\leq$2.

Polynomial	Factorization
1	Irreducible
x	Irreducible
$x+1$	Irreducible
x^2	xx
x^2+1	$(x+1)(x+1)$
x^2+x	$x(x+1)$
x^2+x+1	Irreducible

Table 2.3: Factorization of Polynomials over $\mathbb{F}_2$ with degree ≤ 2

2.5 Extension Finite Fields

If $f(x)$, used to define the ring $\mathbb{F}_q[x]/f(x)$, is a prime polynomial over the field $\mathbb{F}_q$, then $\mathbb{F}_q[x]/f(x)$ is a finite field, namely the extension finite field denoted $\mathbb{F}_{q^m}$, where $m = \deg f$.

A root α of the irreducible polynomial with degree m over $\mathbb{F}_q$ can be used to construct a field if the order of α is $q^m - 1$. Irreducible polynomials with such roots are called primitive polynomials. The powers $\{\alpha^i\}_{i=0}^{q^m-2}$ form an abelian group under multiplication. By adding the element "0", the resulting set becomes an abelian group under addition. Accordingly, a finite field is constructed. Elements of the field can be represented as powers of α, polynomials of degree $m-1$, or vectors of length m with coefficients from $\mathbb{F}_q$.

The previous construction can be performed with any primitive polynomial of a given degree, and hence these polynomials are called *generator polynomials*. The degree m of the generator polynomial is the extension degree of the field, while q is the characteristic of the field. The field from which the extension was made, i.e. $\mathbb{F}_q$,

is always contained within the extension field as a *subfield*.

It would be helpful to consider the analogy to the set of complex numbers, $\mathbb{C}$, which is actually an extension of the of the field of real numbers, $\mathbb{R}$. The extension is defined by the element i, a root of an irreducible polynomial $x^2 - 1$. In this case, the extension degree is 2 and the reduction of higher powers of i is performed according to the rule $i^2 = -1$. However, in the case of extension finite fields, the reduction rule is governed by the generator polynomial used to construct the field.

Example 2.3 (Extending $\mathbb{F}_2$ to $\mathbb{F}_{2^3}$) *The extension of $\mathbb{F}_2$ to $\mathbb{F}_{2^3}$ can be made by taking the first $q^m - 1 = 2^3 - 1 = 7$ powers of a root α of an irreducible polynomial, e.g. $g(x) = x^3 + x + 1$ and adding "0". The elements of the extension field would be $0, 1, \alpha, \alpha^2, \ldots, \alpha^6$, and the operations of addition and multiplication are carried modulo $g(x)$.*

Table 2.4 shows the elements of the field in exponential, polynomial and vector form. This means that $\alpha^3 + \alpha + 1 = 0$, i.e. $\alpha^3 = \alpha + 1$ over $\mathbb{F}_2$ and, for example, $\alpha^4 = \alpha^3\alpha = (\alpha + 1)\alpha = \alpha^2 + \alpha$.

2.6 Generalized Polynomial Representation of Finite Field Elements

The representation of finite field elements is a key subject when it comes to the implementation of finite field operations. Although the two classes of finite fields, i.e.

Exponential form	Polynomial form	Vector form
$\alpha^{-\infty}$	0	000
α^0	1	001
α^1	α	010
α^2	α^2	100
α^3	$\alpha + 1$	011
α^4	$\alpha^2 + \alpha$	110
α^5	$\alpha^2 + \alpha + 1$	111
α^6	$\alpha^2 + 1$	101

Table 2.4: Elements of $\mathbb{F}_{2^3}$ with $g(x) = x^3 + x + 1$

prime and extension fields, have essentially different structures, their representations can be unified. Such unification allows one to abstract over the type of the field, enabling the development of general formulations for field operations regardless of the field type. For that reason, a generalized polynomial representation is adopted throughout this work.

A generalized polynomial representation of finite field elements can be written as

$$f(\mu) = \sum_{i=0}^{m-1} f_i \mu^i = f_0 + f_1\mu + f_2\mu^2 + \ldots + f_{m-1}\mu^{m-1} \tag{2.4}$$

where symbols are interpreted based on the type of the field as follows:

- In case of $\mathbb{F}_p$, μ denotes the radix of the representation, f_i is a radix-μ digit, and m is the number of radix-μ digits required to represent the prime p.
- In case of $\mathbb{F}_{p^m}$, μ denotes the root of the irreducible polynomial, i.e. α, while f_i is a digit polynomial whose coefficients are in $\mathbb{F}_p$ and m is the extension degree of the field.

Chapter 3

Finite Field Multiplication

3.1 Introduction

This chapter reviews the definition of finite field multiplication and the notation used to describe it. Moreover, it discusses some representative examples of its implementations. Lastly, the application of finite field multiplication in cryptography is discussed.

3.2 Overview of Finite Field Multiplication

3.2.1 Definition and Notation

Finite field multiplication operation is very similar to conventional multiplication operation (either integer multiplication in case of $\mathbb{F}_p$ or polynomial multiplication in case of $\mathbb{F}_{p^m}$) with a single key difference, namely, that the product has to be a member of the field. This requires the reduction of the result in case it falls out of the field.

Example 3.1 (Finite field multiplication) *In* $\mathbb{F}_{13}$*, the multiplication of* 5 *by* 2 *results in* 10*, while the multiplication of* 5 *by* 3 *results in* 2 *(*$5 \times 3 = 15$*, which is divided by* 13 *to generate* 2 *as a remainder).*

Finite field multiplication can be described as follows. Given two elements, X and Y, of a field $\mathbb{F}_q$, then X and Y can be represented in a polynomial form as follows.

$$X = \sum_{i=0}^{d-1} x_i \mu^i \tag{3.1}$$

$$Y = \sum_{i=0}^{d-1} y_i \mu^i \tag{3.2}$$

where d is the maximum number of digits required to represent the field elements, and

- in case of a prime fields, i.e. q is prime, $\mu = 2^r$ is the radix and x_i and y_i are radix-2^r digits that can be represented in a redundant form.

- in the case of a extension field, i.e. $q = p^m$ is a power of a prime number, $\mu = \alpha^r$, α is the root of the generator polynomial, and x_i and y_i are digit polynomials of X and Y, respectively, whose coefficients are elements of $\mathbb{F}_p$.

The result of the finite field multiplication of X and Y, denoted by R, is given by

$$R = XY \bmod G(\mu) \tag{3.3}$$

where

$$G(\mu) = \sum_{i=0}^{d} g_i \mu^i \tag{3.4}$$

and where

- in case of a prime field, $G(\mu)$ is the field order, p, and g_i is a radix-2^r digit that could be represented in a redundant form.
- in the case of a extension field, $G(\mu)$ is the generator polynomial and g_i is a digit polynomial of $G(\mu)$ with coefficients that are elements of $\mathbb{F}_p$.

3.2.2 Implementing Finite Field Multiplication

In general, finite field multiplication can be performed in two major modes:

Reduction-after-multiplication. In this mode, conventional multiplication is performed first. Then, the result is reduced to make sure that it falls in the field.

A major disadvantage of this mode is its hardware requirements such as the need for a double-word bus to carry the product to the reduction unit.

Reduction-during-multiplication. In this mode, the finite field result of field multiplication is obtained by applying reduction to the partial products of the conventional multiplication operation as they are generated and accumulated. This mode is much more adopted due to its inherent hardware efficiency.

Finite field reduction can be performed iteratively starting by either the least-significant or the most-significant digit of the quantity to be reduced. Montgomery multiplication [41] is a famous reduction-during-multiplication algorithm that starts from the least-significant position. The original Montgomery algorithm was proposed for modular multiplication and is based on binary representation of elements. However, it has been generalized since then for finite fields as well as for higher radix representation of the elements of the field [49].

In Montgomery multiplication, each digit of X, starting with the least significant digit, is multiplied by Y. The reduction is carried out from the least significant digit, since the least significant digits are generated first. The algorithm can be described using the following iteration

$$R_i = x_i Y + (\mu^{-1} R_{i-1} \bmod G(\mu)) \tag{3.5}$$

Accordingly, the final result will have the form

$$R = XY\mu^{-d} \bmod G(\mu) \tag{3.6}$$

The factor μ^{-d} is inherent in all least-significant-digit-first finite field multiplication algorithms as a result of the division by μ in each iteration as shown in equation 3.5.

Least-significant-digit-first algorithms are more natural for $\mathbb{F}_p$ due to the carry propagation from the least to most significant digits. However, these algorithms are not usually used for $\mathbb{F}_{p^m}$ due to the absence of carry propagation. For $\mathbb{F}_{p^m}$, most-significant-digit-first algorithms are more efficient since they do not result in any additional scaling factor such as that in equation 3.5 and produce an un-scaled result at the end of the multiplication algorithm.

Least-significant-digit-first and most-significant-digit-first algorithms can be implemented using a variety of structures. Most of the existing structures are effectively based on different time-space mapping of the dataflow diagrams of the finite field multiplication operation.

On one end of the spectrum, an array of processing elements is used, where each processing element performs the computation of a single node in the two-dimensional dataflow. Such realizations are referred to as parallel implementations. Parallel implementations are not efficient for large word-lengths such as those encountered in cryptography due to their large area and power requirements.

The other end of the spectrum is to use one processing element to implement the operation of all the nodes in the two-dimensional dataflow. Such realizations are usually referred to as serial implementations. Serial implementations are not efficient for large word lengths such as those encountered in cryptography due to their long

execution time.

Other variations of the basic serial finite field multiplication algorithms have also been proposed where a digit of one operand is multiplied by a block of another operand. A single digit-block multiplier is used to carry out the complete multiplication in a serial fashion. Usually, the block size is larger than the digit size to speed up the computations of the basic serial multiplier. As a consequence, such multiplication algorithms require different bus-widths between the memory module and the digit-block multiplier module.

Another implementation style is mapping dataflow graphs into a linear array of processing elements. The mapping can be done in one of two ways: (i) by projecting along the vertical axis to obtain the serial-parallel implementation, or (ii) by projecting along the horizontal-axis to obtain the serial-serial implementation.

In serial-parallel structures, the digits of one of the input operands are fed serially while the digits of the other input operand must be fed in parallel since they are all required at every cycle. The advantage of the serial-parallel realization is that the first digit of the product needed to perform field reduction is obtained after an initial delay of one cycle. Defining the initial delay as the number of cycles required before the first digit of the result needed for field reduction is obtained, it is clear that the serial-parallel realization inherently has an initial delay which is independent of the word length. The drawbacks of the serial-parallel realizations are as follows:

- it requires parallel loading of the digits of one of the operands, since the digits of one of the operands are all required at every cycle. It is significant to note

that the use of parallel transfer of data back and forth from memory and the multiplication structures is a major drawback for large word lengths such as those encountered in cryptography. The reason is that parallel loading for large wordlength requires large bus-widths which are costly in area and have a significant impact on the execution time.

- it also has to support serial loading of the other operand and hence the hardware must support buses with different bus-widths between the memory module and the multiplier module; one being for parallel communication and one for serial communication.

The major advantage of serial-serial implementations is that all the operand communications are serial in nature and hence no parallel loading is needed. It should be noted that both serial multiplication structures and serial-serial multiplication realizations require only serial communication for all operands and hence require buses with the same bus width. Serial-serial multipliers have the advantage of requiring less number of memory accesses than their serial counterparts. Also, to achieve the same execution time, the bus width of the serial multiplier needs to be higher than that needed in the serial-serial multiplier. Both of these features make serial-serial multiplier ideal for low power devices such as smart cards.

It should be noted that serial-parallel multipliers can be modified by loading the operand whose digits are needed at every cycle serially one digit at a time into a serial-in-parallel-out register. The drawback is that this will incur an initial delay prior to the commencement of the multiplication operation, which is dependent on the word length.

3.2.3 Some Existing Serial-serial Finite Field Multipliers

Serial-serial structures that do not require all the digits of all the operands in every cycle have been proposed. One such structure was reported in [61]. However, such structures have a major drawback in that the initial delay and clock cycle are dependent on the word length of the operands. For large word lengths such as those in cryptography, this long initial delay and clock cycle represent a significant drawback.

The authors of the structure in [61] use extra pipelining registers to make the initial delay and the clock cycle independent of the word length. It is significant to note that the use of extra pipelining registers will significantly increase the hardware and will require additional clock cycles, which are proportional to the word length.

A serial-serial finite field reduction structure is disclosed in the U.S. Patent Number 5,101,431 of Even for "Systolic Array for Modular Multiplication". However, as disclosed therein, the initial delay and clock cycle are dependent on the word length of the operands. For large word lengths such as those in cryptography, the long initial delay and clock cycles are a draw back. The patent also discloses the use of extra pipelining registers to make the initial delay and the clock-cycle independent of the word length. It is significant to note that the use of extra pipelining registers will significantly increase the hardware and will require additional clock-cycles, which are proportional to the word length.

Another serial-serial structure is disclosed in a published patent application of Mellott et al. Number 2002/0161810 entitled "Method and Apparatus for Multiplication and/or Modular Reduction Processing." In this structure the initial delay and

clock-cycle are also dependent on the word-length. However, to reduce the effect of word length dependency, the inventors use a multi-port adder, which may be implemented for example as a tree adder. This will reduce the effect of word length dependency, but it will not eliminate it completely.

It is also significant to note that the use of a multi-port adder will lead to an irregular structure, and hence the structure is not systolic, i.e. it is not modular, and requires irregular communication.

Therefore there is a need for a realization which (i) allows serial loading of all needed operands, and (ii) has an initial delay and clock cycle which are inherently independent of the word length.

It should also be noted that any new realization must allow for scalability, which is an additional requirement for the application of finite field multiplication in cryptography. Scalable multiplication structures are those that can be reused or replicated in order to generate long-precision results independently of the data path precision for which the unit was originally designed. Scalability is needed because the key lengths could be increased for higher security. Any cryptographic hardware such as smart cards should not become obsolete with a new key length.

3.3 Applications of Finite Field Multiplication in Cryptography

Due to the security requirements of data transfer and digitization, the current need for cryptography for data security mechanism is ever growing. The basic principle of cryptography is that a plaintext is converted into a ciphertext through encryption using a particular encryption key. When a receiver receives the ciphertext, decryption with a key that is related to the encryption key can recover the plaintext. Because the data is transferred or stored as ciphertext, data security is achieved since an adversary cannot interpret the ciphertext.

In particular, public-key cryptography enables secure communication between users that have not previously agreed upon any shared keys. This is most often done using a combination of symmetric and asymmetric cryptography: public-key techniques are used to establish user identity and a common symmetric key, and a symmetric encryption algorithm are used for the encryption and decryption of the actual messages. The former operation is called key agreement. Prior establishment is necessary in symmetric cryptography, which uses algorithms for which the same key is used to encrypt and decrypt a message. Public-key cryptography, in contrast, is based on key pairs. A key pair consists of a private key and a public key. As the names imply, the private key is kept private by its owner, while the public key is made public (and typically associated to its owner in an authentic manner). In asymmetric encryption, the encryption step is performed using the public key, and decryption using the private key. Thus the encrypted message can be sent along an

insecure channel with the assurance that only the intended recipient can decrypt it.

The use of cryptographic key pairs is disclosed in the U.S. Patent of Hellman Number 4,200,770. The Hellman patent also disclosed the application of key pairs to the problem of key agreement over an insecure communication channel. The algorithms specified in the aforementioned patent relies for their security on the difficulty of the mathematical problem of finding a discrete logarithm.

In order to undermine the security of a discrete-logarithm based crypto-algorithm, an adversary must be able to perform the inverse of finite field exponentiation (i.e., solve a discrete logarithm problem). There are mathematical methods for finding a discrete logarithm (e.g., the Number Field Sieve), but these algorithms would take an unreasonably long time using sophisticated computers when certain conditions are met in the specification of the crypto algorithm.

In particular, it is necessary for the used numbers involved to be large enough. The larger the numbers used, the more time and computing power is required to find the discrete logarithm and break the cryptosystem. On the other hand, very large numbers lead to very long public keys and transmissions of cryptographic data. The use of very large numbers also requires large amounts of time and computational power in order to perform the crypto algorithm. Thus, cryptographers are always looking for ways to minimize the size of the numbers involved, and the time and power required, in performing the encryption and/or authentication algorithms. The payoff for finding such a method is that cryptography can be done faster, cheaper, and in devices that do not have large amounts of computational power (e.g., hand-held smart-cards).

A discrete-logarithm based crypto-algorithm can be performed in any mathematical setting in which certain algebraic rules hold true. In mathematical language, the setting must be a finite cyclic group. The choice of the group is critical in a cryptographic system. The discrete logarithm problem may be more difficult in one group than in another for which the numbers are of comparable size. The more difficult the discrete logarithm problem, the smaller the numbers that are required to implement the crypto-algorithm while maintaining the same level of security. Working with smaller numbers is easier, more efficient and requires less storage compared to working with larger numbers. So, by choosing the right group, a user may be able to work with smaller numbers, make a faster cryptographic system, and get the same, or better, cryptographic strength than from another cryptographic system that uses larger numbers.

3.3.1 Elliptic Curve Cryptography

A method of adapting discrete-logarithm based algorithms to the setting of elliptic curves was disclosed independently by V. Miller in [40] and N. Koblitz in [31]. It appears that finding discrete logarithms in this kind of group is particularly difficult. Thus elliptic curve-based crypto algorithms can be implemented using much smaller numbers than in a finite-field setting of comparable cryptographic strength. Hence, the use of elliptic curve cryptography is an improvement over finite-field based public-key cryptography. The Elliptic Curve Cryptosystem relies upon the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP) to provide its effectiveness as a cryptosystem. Using multiplicative notation, the problem can be described as: given

points B and Q in the group, find a number k such that $B^k = Q$; where k is called the discrete logarithm of Q to the base B. Using additive notation, the problem becomes: given two points B and Q in the group, find a number k such that $kB = Q$.

In an Elliptic Curve Cryptosystem, the integer k is kept private and is often referred to as the secret key. The point Q together with the base point B are made public and are referred to as the public key. The security of the system, thus, relies upon the difficulty of deriving the secret k, knowing the public points B and Q. The main factor that determines the security strength of such a system is the size of its underlying finite field. In a real cryptographic application, the underlying field is made so large that it is computationally infeasible to determine k in a straightforward way by computing all the multiples of B until Q is found.

The core of the elliptic curve geometric arithmetic is an operation called scalar multiplication which computes kB by adding together k copies of the point B. Scalar multiplication over elliptic curve is also referred to as exponentiation over elliptic curve.

3.3.2 RSA Cryptography

Another well-known public key cryptosystem is Rivest, Shamir, Adleman (RSA) which is also based on the discrete logarithm problem of over the field $\mathbb{F}_p$. RSA requires field exponentiation over $\mathbb{F}_p$.

In what follows, field exponentiation is used to refer to scalar multiplication in

elliptic curve cryptography, as well as finite field exponentiation in RSA cryptography.

A drawback of such cryptographic systems is that calculation of field exponentiation remains a daunting mathematical task even to an authorized receiver using a high speed computer. With the prevalence of public computer networks used to transmit confidential data for personal, business and governmental purposes, it is anticipated that most computer users will want cryptographic systems to control access to their data. Despite the increased security, the difficulty of finite field exponentiation calculations will substantially drain computer resources and degrade data throughput rates, and thus represents a major impediment to the widespread adoption of commercial cryptographic systems.

Accordingly, a critical need exists for an efficient finite field exponentiation method and apparatus to provide a sufficient level of communication security while minimizing the impact to computer system performance and data throughput rates.

Field exponentiation over an elliptic curve is heavily dependent on field multiplications. Point addition and doubling, which are the basic operations for exponentiation over an elliptic curve, require many field multiplication operations.

Field multiplication is also the basic operation used to compute field exponentiation in RSA cryptography, which is used in smart cards or other secure devices. RSA requires the rapid finite field multiplication of large integers, for example 512- or 1024-bits in length, in order to carry out finite field exponentiations.

Therefore, field multiplication is widely used in encryption/decryption, authentication, key distribution and many other applications such as those described above.

Chapter 4

Serial-serial Finite Field Multiplication

4.1 Introduction

In this chapter, the novel serial-serial time-space mapping of finite field multiplication will be introduced. Firstly, the chapter discusses the discard-based least-to-most finite field multiplication algorithm. Then, the serial-serial time-space mapping of this algorithm will be explained. Afterwards, a serial-serial multiplier for prime fields will be designed as an example of the resulting class of multipliers. Lastly, a most-to-least version of the mapping will be introduced.

4.2 A Serial-serial Time-space Mapping of Finite Field Multiplication

While developing our serial-serial multiplier, we will adopt a least-significant-digit-first algorithm since it is more suitable for cases where there is carry propagation, i.e. $\mathbb{F}_p$. Moreover, we will perform field reduction by discarding rather than correction since discarding shortens the critical path of each iteration in the loop.

To perform field reduction through correction, a quantity should be added to the partial sum to make its least-significant digit become zero. Moreover, this quantity has to be a multiple of the modulus M not to disturb the calculations. One way to achieve this is to find a multiple of the modulus that has a least-significant digit of $\mu - 1$. Then, the digit that is to be turned to zero would be multiplied by this number to give a quantity that would turn the least-significant digit into zero. This is the essence of Algorithm 3 in [49].

In our case, however, we don't want to add this correction quantity beforehand to make sure that the least-significant digit will be zero. Rather, we would like to discard that digit and then correct for it in the next iteration. Correcting for this digit is equivalent to multiplying it by μ^{-1}, i.e. the inverse of the radix modulo M, and adding it again to the partial sum. This inverse, denoted Θ hereafter, is always guaranteed to exist when $\gcd(\mu, M) = 1$. This algorithm, which is the one that we will use to develop serial-serial field multiplication, is listed as Algorithm 1.

4.2.1 The Adopted Finite Field Multiplication Algorithm

A least-to-most finite field multiplication of two n-digit numbers, X and Y, can be represented by the following equation.

$$R = XY\mu^{-n}(\mathrm{mod}M) \tag{4.1}$$

The factor μ^{-n} is inherent in all least-to-most multiplication algorithms due to the repeated division by μ in the n cycles of the algorithm.

Rewriting X in terms of digits, the following is obtained.

$$R = \mu^{-n}\sum_{i=0}^{n-1} x_i Y\mu^i = \mu^{-1}\left(\mu^{-(n-1)}\sum_{i=0}^{n-1} x_i Y\mu^i\right)(\mathrm{mod}M) \tag{4.2}$$

Let's denote R by R_n. Then,

$$R_n = \mu^{-1}R_{n-1}(\mathrm{mod}M) \tag{4.3}$$

where

$$\begin{aligned} R_{n-1} &= \mu^{-(n-1)}\sum_{i=0}^{n-1} x_i Y\mu^i = x_{n-1}Y + \left(\mu^{-(n-2)}\sum_{i=0}^{n-2} x_i Y\mu^i\right) \\ &= x_{n-1}Y + R_{n-2}(\mathrm{mod}M) \end{aligned} \tag{4.4}$$

In general,

$$R_i = x_i Y + \mu^{-1}R_{i-1}(\mathrm{mod}M), 0 \le i \le n \tag{4.5}$$

At this point, there are two options of performing the division $\mu^{-1}R_{i-1}$. The first option is to find a multiple of M that can be added to R_{i-1} to make its least-significant digit equal to 0, thus enabling the division by μ without loss. Another option is to perform the division despite the loss, i.e. *discard* the least-significant digit, and correct for it in a later cycle. Although the first option can be used for our purpose, we are adopting the second one since it has a higher potential of resulting in an efficient implementation. Denoting the least significant digit of R_{i-1} as q_i, the last equation can be then rewritten as

$$R_i = x_i Y + \mu^{-1} R_{i-1} + \mu^{-1} q_i (\mathrm{mod} M), 0 \leq i \leq n \tag{4.6}$$

To perform the correction, we let $\Theta = \mu^{-1} \bmod M$, i.e. denoting the inverse of the radix modulo M. Then, we can write

$$R_i = \mu^{-1} R_{i-1} + x_i Y + q_i \Theta, 0 \leq i \leq n \tag{4.7}$$

wherein no loss is incurred. The resulting algorithm, listed as Algorithm 1, is the one that we will use to develop our serial-serial multiplication algorithm. A visual representation of Algorithm 1 is shown in Figure 4.1.

Algorithm 1 Discard-based Least-to-most Finite Field Multiplication

Require: $2X, 2Y < \mu^n$, $4M < \mu^{n-1}$
Ensure: $R = XY\mu^{-n} (\mathrm{mod} M)$, $2R < \mu^n$
1: $R_{-1} = 0$
2: **for** $i = 0 \rightarrow n$ **do**
3: $\quad q_i = R_{i-1} \bmod \mu$ ▷ q_i is the least-significant digit of R_{i-1}
4: $\quad R_i = R_{i-1} \operatorname{div} \mu + x_i Y + q_i \Theta$
5: **end for**
6: $R = R_n$

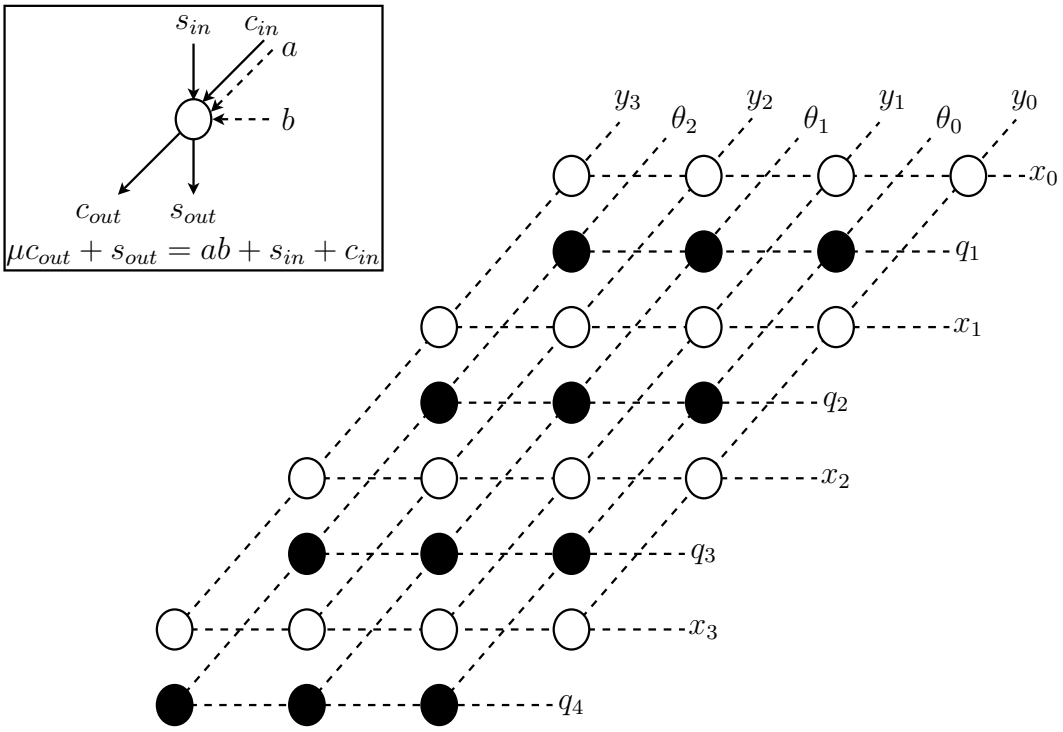

Figure 4.1: The dataflow representing Algorithm 1: Discard-based Least-to-most Finite Field Multiplication

Example 4.1 (Discard-based Least-to-most Finite Field Multiplication) *Let* $X = 1234$, $Y = 5678$, $M = 301$ *and* $\mu = 10$. *In this case,* $\Theta = \mu^{-1} \bmod M = 271$ *since* $271 \times 10 \bmod 301 = 1$. *Figure 4.2 shows how would this multiplication progress according to Algorithm 1. The result can be confirmed by checking that* $1234 \times 5678 = 3077 \times 10000 (\mathrm{mod} M)$.

To justify the limits imposed on the wordlength of the operands in Algorithm 1, the following analysis is carried out. The wordlength is n digits. Multiplication will take $n + 1$ cycles and include n divisions by the radix μ.

$$R = \mu^{-n} \left(XY + \sum_{k=1}^{n} q_k \Theta \mu^k \right) \tag{4.8}$$

In case of $\mathbb{Z}_M$, the radix will have r bits, i.e. 2^r. The wordlength would be nr.

$$R = 2^{-nr} \left(XY + \sum_{k=1}^{n} q_k \Theta 2^{kr} \right) \tag{4.9}$$

Let α denote the maximum number of bits in X and Y.

$$\max(X) = \max(Y) = 2^{\alpha} - 1 \tag{4.10}$$

The modulus, M, has β bits at most. Θ is always less than the modulus.

$$\max(\Theta) = 2^{\beta} - 2 \tag{4.11}$$

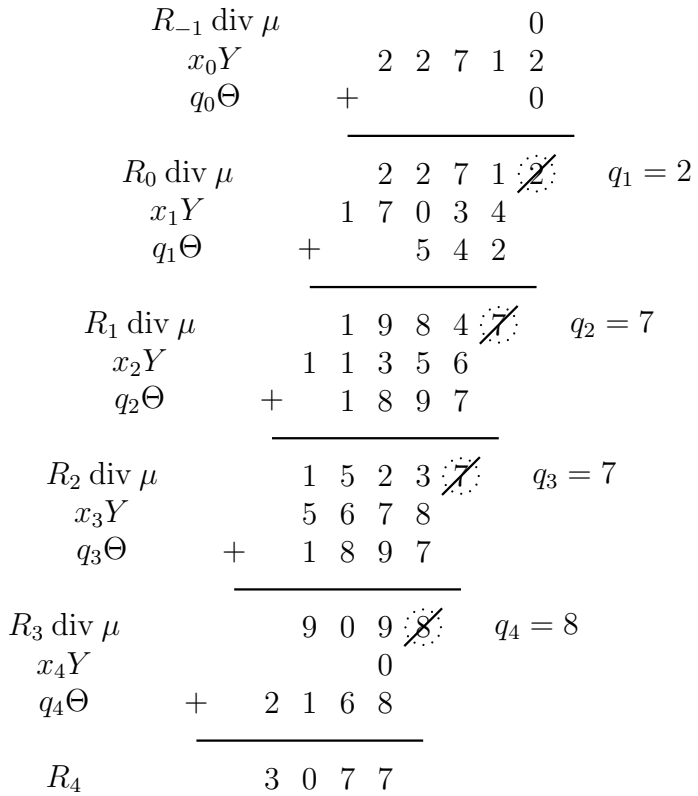

Figure 4.2: An example on discard-based modular multiplication with $X = 1234$, $Y = 5678$, $\Theta = 271$, $\mu = 10$ and $n = 4$.

Moreover, the maximum value of $\sum_{k=1}^{n} q_k 2^{kr}$ is

$$\max\left(\sum_{k=1}^{n} q_k 2^{kr}\right) = 2^r(2^{nr}-1) = 2^{(n+1)r} - 2^r \tag{4.12}$$

Given the above, the maximum value of R is

$$\max(R) = 2^{-nr}\left[(2^\alpha - 1)^2 + \left(2^{(n+1)r} - 2^r\right)\left(2^\beta - 2\right)\right] \leq 2^\alpha - 1 \tag{4.13}$$

$$\max(R) = 2^{-nr}\left[2^{2\alpha} - 2^{\alpha+1} + 1 + 2^{(n+1)r+\beta} - 2^{(n+1)r+1} - 2^{\beta+r} + 2^{r+1}\right] \leq 2^\alpha - 1 \tag{4.14}$$

The division by 2^{nr} will eliminate some terms as follows.

$$\max(R) = 2^{2\alpha-nr} - 2^{\alpha-nr+1} + 2^{r+\beta} - 2^{r+1} - 2^{\beta-nr+r} \leq 2^\alpha - 1 \tag{4.15}$$

Setting $\alpha = nr - 1$ and $\beta = nr - r - 2$ satisfies the inequaltiy and determines the maximum number of bits for X, Y, M and R.

4.2.2 A Novel Serial-serial Time-space Mapping

A structure can be described to be serial-serial if all of its inputs are supplied serially, one digit at a time, and all of its outputs are also generated serially.

Accordingly, in the first cycle, the only information available to us is the first digit of each operand, namely, x_0 and y_0. Looking at figure 4.3, it can be realized that the digit multiplication denoted by 0 can be performed since all of its inputs are available. This digit multiplication will generate q_1, namely the least-significant digit of the current partial sum, which will be needed in the next cycle to perform the correction. In the next cycle, x_1 and y_1 should be supplied in addition to the least significant digit of Θ, namely θ_0. Given that q_1 has already been produced in the last cycle, all operations marked with 1 can now be performed. In the next cycle, x_2, y_2, q_2 and θ_1 will be available enabling operations labeled with 2.

After all inputs are supplied, i.e. in cycle 4, few cycles are required for the carries to propagate through the structure generating the final result, one digit at a time.

To be able to formulate finite field multiplication in a serial manner, the recurrence in Algorithm 1 has to be unfolded as follows.

$$R = \mu^{-n} \left(\sum_{i=0}^{n} x_i Y \mu^i + \sum_{k=0}^{n} q_k \Theta \mu^k \right) \tag{4.16}$$

This can be rewritten as

$$R = \mu^{-n} \left(\sum_{i=0}^{n} x_i \sum_{j=0}^{n} y_j \mu^{i+j} + \sum_{k=0}^{n} q_k \sum_{h=0}^{n} \theta_h \mu^{k+l} \right) \tag{4.17}$$

where $x_n = y_n = q_0 = \theta_n = 0$. This is equivalent to,

$$R = \mu^{-n}\left(\sum_{i=0}^{n} x_i \sum_{j=0}^{n} y_j \mu^{i+j} + \sum_{k=0}^{n} q_k \sum_{h=0}^{n} \theta_{h-1}\mu^{k+h-1}\right) \tag{4.18}$$

Since this is serial-serial, one new digit of each number will be processed each iteration. Since it is least-to-most, the last digit to be processed will the highest-significant digit. Denoting R by S_n, we can write

$$\begin{aligned}
S_n &= \mu^{-n}(x_n y_n \mu^{2n} + x_n \mu^n \sum_{j=0}^{n-1} y_j \mu^j + y_n \mu^n \sum_{i=0}^{n-1} x_i \mu^i + \\
&\quad q_n \theta_{n-1} \mu^{2n-1} + q_n \mu^n \sum_{h=0}^{n-1} \theta_{h-1}\mu^{h-1} + \theta_{n-1}\mu^{n-1} \sum_{k=0}^{n-1} q_k \mu^k + \\
&\quad \sum_{i=0}^{n-1} x_i \sum_{j=0}^{n-1} y_j \mu^{i+j} + \sum_{k=0}^{n-1} q_k \sum_{h=0}^{n-1} \theta_{h-1}\mu^{k+h-1}) \\
&= x_n y_n \mu^n + x_n \sum_{j=0}^{n-1} y_j \mu^j + y_n \sum_{i=0}^{n-1} x_i \mu^i + \\
&\quad q_n \theta_{n-1}\mu^{n-1} + q_n \sum_{h=0}^{n-1} \theta_{h-1}\mu^{h-1} + \theta_{n-1} \sum_{k=0}^{n-1} q_k \mu^{k-1} + \\
&\quad \mu^{-1}\left[\mu^{-(n-1)}\left(\sum_{i=0}^{n-1} x_i \sum_{j=0}^{n-1} y_j \mu^{i+j} + \sum_{k=0}^{n-1} q_k \sum_{h=0}^{n-1} \theta_{h-1}\mu^{k+h-1}\right)\right]
\end{aligned} \tag{4.19}$$

This can be re-written as

$$\begin{aligned} S_n &= x_n y_n \mu^n + x_n \sum_{j=0}^{n-1} y_j \mu^j + y_n \sum_{i=0}^{n-1} x_i \mu^i + \\ & q_n \theta_{n-1} \mu^{n-1} + q_n \sum_{h=0}^{n-1} \theta_{h-1} \mu^{h-1} + \theta_{n-1} \sum_{k=0}^{n-1} q_k \mu^{k-1} + \\ & \mu^{-1} S_{n-1} \end{aligned} \tag{4.20}$$

where

$$S_{n-1} = \mu^{-(n-1)} \left(\sum_{i=0}^{n-1} x_i \sum_{j=0}^{n-1} y_j \mu^{i+j} + \sum_{k=0}^{n-1} q_k \sum_{h=0}^{n-1} \theta_{h-1} \mu^{k+h-1} \right) \tag{4.21}$$

In general,

$$\begin{aligned} S_g &= x_g y_g \mu^g + x_g \sum_{j=0}^{g-1} y_j \mu^j + y_g \sum_{i=0}^{g-1} x_i \mu^i + \\ & q_g \theta_{g-1} \mu^{g-1} + q_g \sum_{h=0}^{g-1} \theta_{h-1} \mu^{h-1} + \theta_{g-1} \sum_{k=0}^{g-1} q_k \mu^{k-1} + \\ & \mu^{-1} S_{g-1} \end{aligned} \tag{4.22}$$

This equation is a general description of the serial-serial least-to-most finite field multiplication.

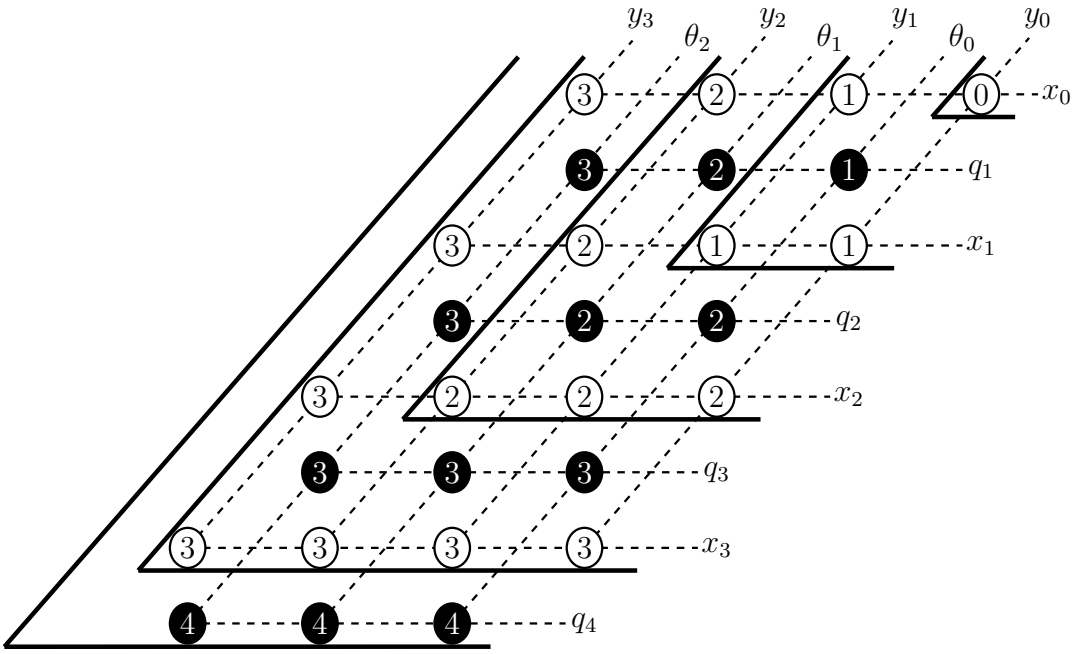

Figure 4.3: A serial-serial time-space mapping of the finite field multiplication dataflow

4.3 An Implementation of a Serial-serial Finite Field Multiplier

In the previous section, a serial-serial formulation for the least-to-most finite field multiplication operation was obtained. In this section, we will proceed by designing a serial-serial $\mathbb{Z}_M$ multiplier, which is a general case of the $\mathbb{F}_p$ multiplier. In this case, μ will be equal to 2^r, where r is the number of bits used to represent a single digit.

Noting that the product of two digits consists of two digits, i.e. $ab = [ab]_H 2^r + [ab]_L$, where $[ab]_H$ is the high-significance digit and $[ab]_L$ is the low-significance digit, equation 4.22 can be rewritten as

$$\begin{aligned} S_g &= \left([x_g y_g]_H 2^r + [x_g y_g]_L\right) 2^{rg} + \left([q_g \theta_{g-1}]_H 2^r + [q_g \theta_{g-1}]_L\right) 2^{r(g-1)} \\ &+ \sum_{j=0}^{g-1} \left([x_g y_j]_H 2^r + [x_g y_j]_L\right) 2^{rj} + \sum_{i=0}^{g-1} \left([x_i y_g]_H 2^r + [x_i y_g]_L\right) 2^{ri} \\ &+ \sum_{h=0}^{g-1} \left([q_g \theta_{h-1}]_H 2^r + [q_g \theta_{h-1}]_L\right) 2^{r(h-1)} \\ &+ \sum_{k=0}^{g-1} \left([\theta_{g-1} q_k]_H 2^r + [\theta_{g-1} q_k]_L\right) 2^{r(k-1)} + 2^{-r} S_{g-1} \end{aligned} \tag{4.23}$$

S_g can have n digits at most. Accordingly, equation 4.23 can be rewritten as

$$\begin{aligned}\sum_{l=0}^{n-1} S_g^{[l]} &= \left([x_g y_g]_H 2^r + [x_g y_g]_L\right) 2^{rg} + \left([q_g \theta_{g-1}]_H 2^r + [q_g \theta_{g-1}]_L\right) 2^{r(g-1)} \\ &+ \sum_{j=0}^{g-1} \left([x_g y_j]_H 2^r + [x_g y_j]_L\right) 2^{rj} + \sum_{i=0}^{g-1} \left([x_i y_g]_H 2^r + [x_i y_g]_L\right) 2^{ri} \\ &+ \sum_{h=0}^{g-1} \left([q_g \theta_{h-1}]_H 2^r + [q_g \theta_{h-1}]_L\right) 2^{r(h-1)} \\ &+ \sum_{k=0}^{g-1} \left([\theta_{g-1} q_k]_H 2^r + [\theta_{g-1} q_k]_L\right) 2^{r(k-1)} + 2^{-r} S_{g-1} \end{aligned} \tag{4.24}$$

Using equation 4.24, a serial-serial multiplier finite field multiplier can be built. The block diagram of that multiplier, in which each cell works with a single significance at any cycle, is shown in Figure 4.4.

Each cell has five inputs and one output. Four of the five inputs are the broadcast lines while the fifth is the sum digit coming from the next higher cell. Meanwhile, each cell stores its carry digit and the higher digits of the products to be used in the next iteration. The function of each cell can be directly obtained by manipulating equation 4.24. Based on the fact that cell d will store the values of x_d, y_d, θ_d and q_{d+1}, it is clear that at the g^{th} cycle, only the first g cells will be operational. Based on equation 4.24, cell d has one of five operation modes based on the iteration index g:

1. If $g < d$, then cell d would be idle.

2. If $g = d$, then cell d would accumulate the term of significance 2^{gr} in equation

4.24, which is $[x_g y_g]_L$, meaning that the function of the cell would be defined as:

$$\left[S_g^{[d]}\right]_c 2^r + \left[S_g^{[d]}\right]_s = [x_g y_g]_L \tag{4.25}$$

In addition, it will store x_g and y_g as its x_d and y_d, respectively.

3. If $g = d + 1$, then cell d would accumulate the terms of significance $2^{(g-1)r}$ in equation 4.24.

$$\left[S_g^{[d]}\right]_c 2^r + \left[S_g^{[d]}\right]_s = [x_d y_g]_L + [x_g y_d]_L + [q_g \theta_{g-1}]_L + [x_{g-1} y_{g-1}]_H \tag{4.26}$$

In addition, it will store q_g and θ_{g-1} as its q_d and θ_d, respectively.

4. If $g = d + 2$, then the cell d would accumulate terms of significance $2^{(g-2)r}$ in equation 4.24.

$$\begin{aligned} \left[S_g^{[d]}\right]_c 2^r + \left[S_g^{[d]}\right]_s &= [x_d y_g]_L + [x_g y_d]_L + [q_d \theta_{g-1}]_L + [q_g \theta_d]_L \\ &\quad + [x_d y_{g-1}]_H + [x_{g-1} y_d]_H + [q_{g-1} \theta_{g-2}]_H \\ &\quad + \left[S_{g-1}^{[d]}\right]_c + \left[S_{g-1}^{[d+1]}\right]_s \end{aligned} \tag{4.27}$$

5. If $g > d + 2$ (the general case), then cell d would calculate the terms of significance 2^d.

Cycle	**Result**	**Cell 0**	**Cell 1**	**Cell 2**	**Cell 3**
0	q_1	x_0y_0			
1	q_2	$x_1y_0 + x_0y_1 + q_1\theta_0$	x_1y_1		
2	q_3	$x_2y_0 + x_0y_2 + q_1\theta_1 + q_2\theta_0$	$x_2y_1 + x_1y_2 + q_2\theta_1$	x_2y_2	
3	q_4	$x_3y_0 + x_0y_3 + q_1\theta_2 + q_3\theta_0$	$x_3y_1 + x_1y_3 + q_2\theta_2 + q_3\theta_1$	$x_2y_3 + x_3y_2 + q_3\theta_2$	x_3y_3
4	r_0	$q_4\theta_0$	$q_4\theta_1$	$q_4\theta_2$	↙
5	r_1	↙	↙	↙	
6	r_2	↙	↙		
7	r_3	↙			

Table 4.1: Time-Space mapping of serial-serial $\mathbb{F}_p$ multiplication

$$\begin{aligned}\left[S_g^{[d]}\right]_c 2^r + \left[S_g^{[d]}\right]_s &= [x_d y_g]_L + [x_g y_d]_L + [q_d\theta_{g-1}]_L + [q_g\theta_d]_L \\ &\quad + [x_d y_{g-1}]_H + [x_{g-1}y_d]_H + [q_d\theta_{g-2}]_H + [q_{g-1}\theta_d]_H \\ &\quad + \left[S_{g-1}^{[d]}\right]_c + \left[S_{g-1}^{[d+1]}\right]_s \end{aligned} \tag{4.28}$$

Figure 4.5 shows the general structure of this multiplier cell. As it can be clearly seen, only serial communication of the input and output operands is used. Table 4.1 summarizes this procedure for $n = 4$.

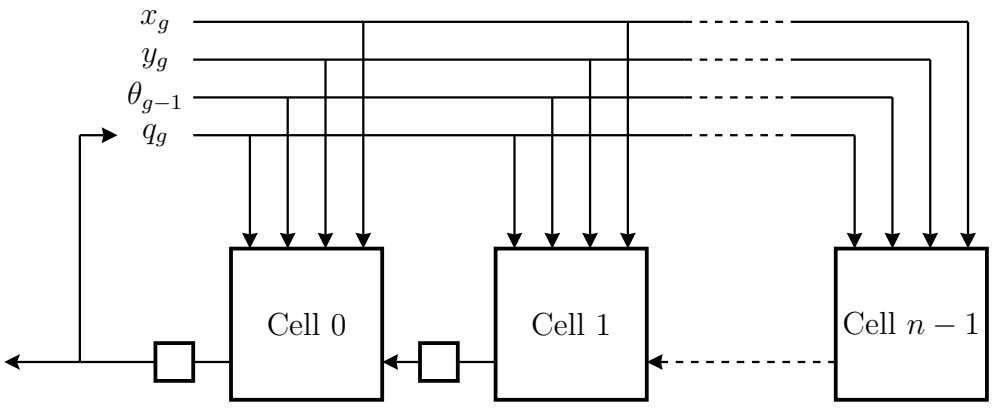

Figure 4.4: A serial-serial finite field multiplier

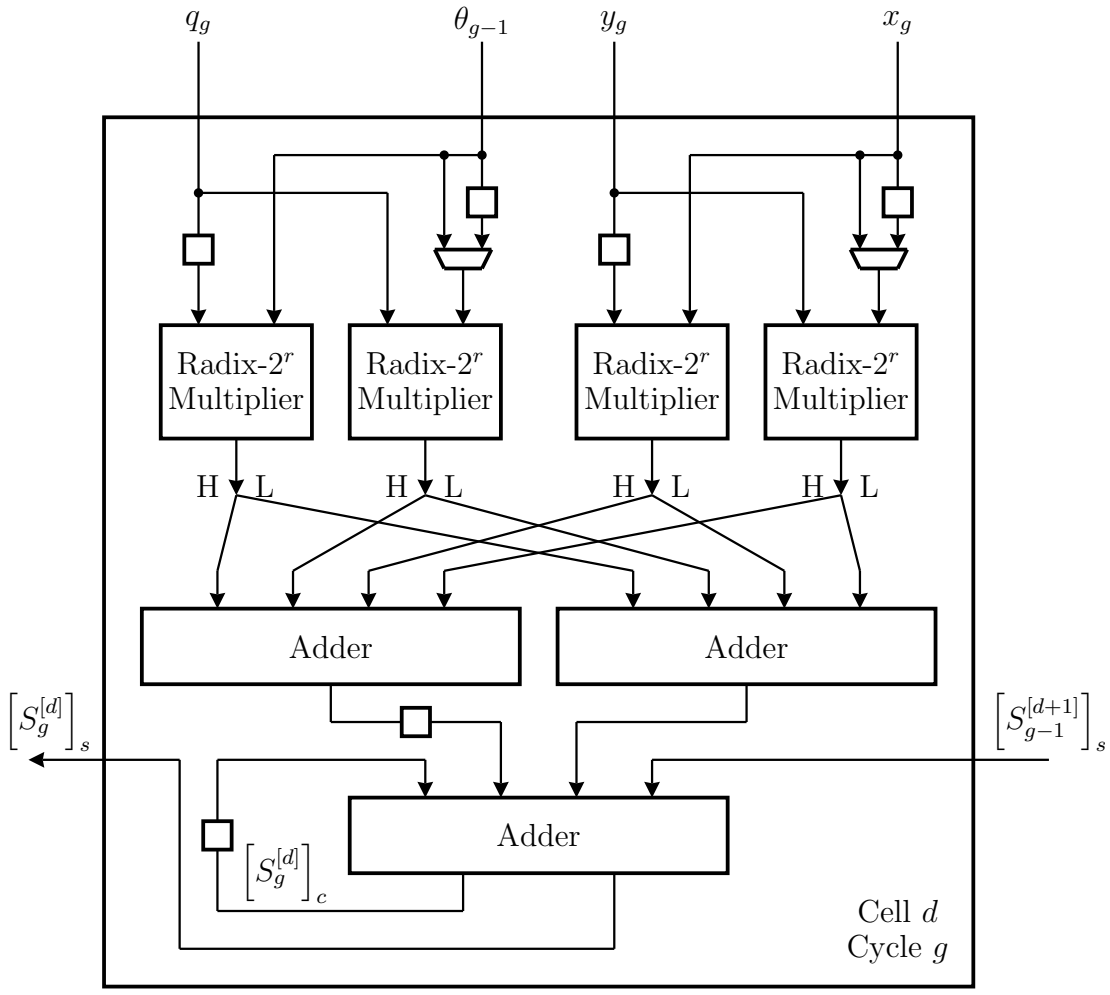

Figure 4.5: A cell of the serial-serial finite field multiplier

4.4 Most-to-least Serial-serial Finite Field Multiplication

Since there is no carry propagation in $\mathbb{F}_{p^m}$ arithmetic, multiplication can be performed starting from high- to low-significant digits. The most important advantage over the least-to-most algorithm is that the operands and the outputs will be represented in the usual domain rather than in the Montgomery domain. In other words, no conversion is required before or after the multiplication. Most-to-least finite field multiplication can be achieved by correcting for the most-significant digit of the partial sum. In this case, the correction factor Θ would be equal to $\mu^n \bmod M$. Algorithm 2, which is the most-to-least counterpart of Algorithm 1, represents this operation. A visual representation of this algorithm is shown in figure 4.6.

Algorithm 2 Discard-based Most-to-least Finite Field Multiplication

Require: $X, Y \in \mathbb{F}_{p^m}$
Ensure: $R = XY (\mathrm{mod} M)$, $R \in \mathbb{F}_{p^m}$
1: $R_{-1} = 0$
2: **for** $i = 0 \rightarrow n-1$ **do**
3: $\quad q_{n-i} = R_{i-1} \text{ div } \mu^{n-1}$ ▷ q_{n-i} is the most-significant digit of R_{i-1}
4: $\quad R_i = \mu R_{i-1} + x_{n-i-1} Y + q_{n-i} \Theta$
5: **end for**
6: $R = R_{n-1}$

The most-to-least finite field multiplication grid can be mapped in a serial-serial manner in the same way that has been applied earlier, as shown in Figure 4.7.

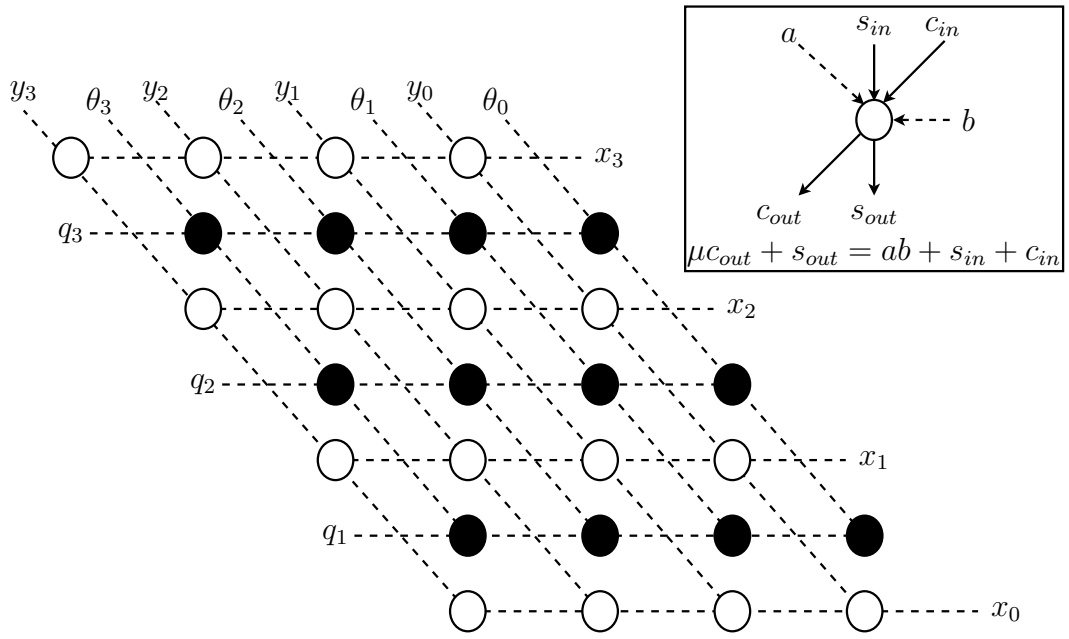

Figure 4.6: Most-to-least finite field multiplication dataflow

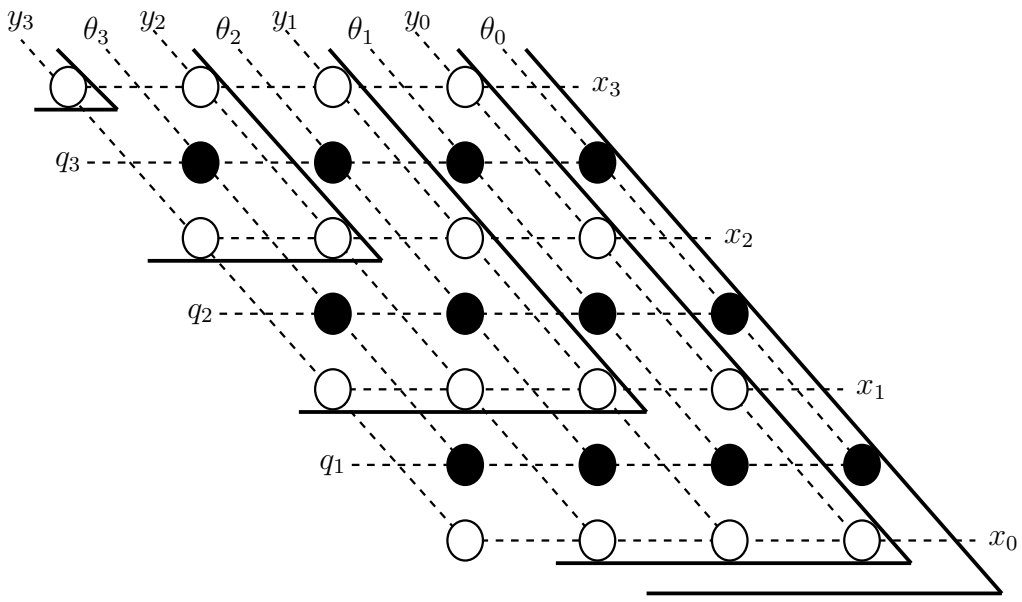

Figure 4.7: A serial-serial time-space mapping of the most-to-least finite field multiplication dataflow

Chapter 5

Enhanced Implementations

5.1 Introduction

This chapter discusses some enhancements that can be applied while implementing a serial-serial finite field multiplier in order to reduce the size of the structure and/or increase its throughput.

Although these changes will be materialized here using the $\mathbb{F}_p$ multiplication structure that has been designed in Chapter 4, their application is by no means limited to that structure. Any serial-serial structure that follows from our results can be enhanced accordingly.

Cycle	Result	Cell 0	Cell 1
0	q_1	x_0y_0	
1	q_2	$x_1y_0 + x_0y_1 + q_1\theta_0$	x_1y_1
2	q_3	$x_2y_0 + x_0y_2 + q_1\theta_1 + q_2\theta_0$	$x_2y_1 + x_1y_2 + q_2\theta_1$
3	q_4	$x_3y_0 + x_0y_3 + q_1\theta_2 + q_3\theta_0$	$x_3y_1 + x_1y_3 + q_2\theta_2 + q_3\theta_1$
4	r_0	$\{x_2y_2\} + q_4\theta_0$	$q_4\theta_1$
5	r_1	$\{x_2y_3 + x_3y_2 + q_3\theta_2\}$	$\{x_3y_3\}$
6	r_2	$\{q_4\theta_2\}$	$\swarrow$
7	r_3	$\swarrow$	

Table 5.1: New time-space mapping of serial-serial $\mathbb{F}_p$ multiplication

5.2 An Area-efficient Implementation

Upon studying table 4.1, it can be noticed that during the second half of the cycles, digit multipliers in the cells are all idle. Moreover, cells at the upper half are idle half of the cycles.

It is possible, through remapping the operations to cells, to reduce the number of required cells by 50% without modifying the internal design of the cell, resulting in a reduction of the required area by the same percentage. Table 5.1 shows the new mapping in which only half of the cells are used. Shifted calculations are surrounded by curly braces.

The only change that is required to take advantage of the new mapping is to supply the upper half of the inputs twice. This is easily achievable through buffering them for $n/2$ cycles. The new structure is shown in Figure 5.1.

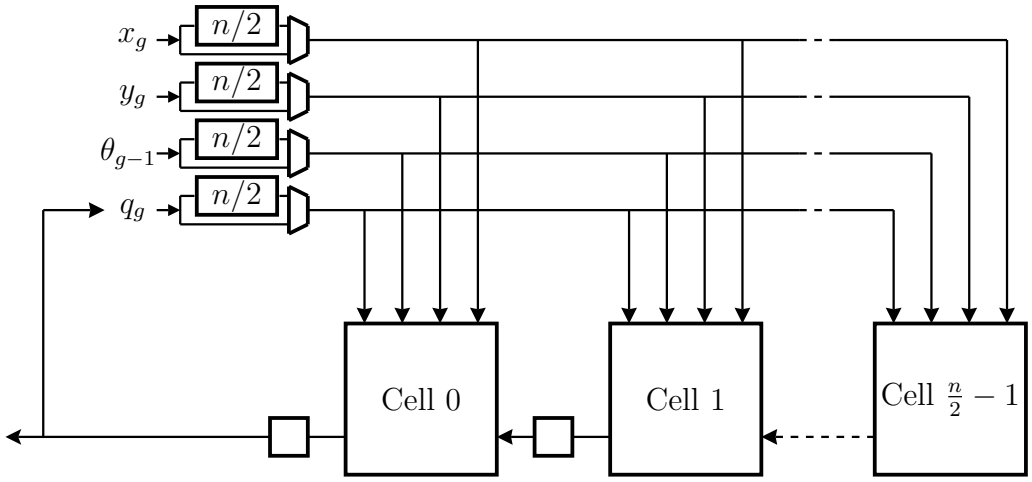

Figure 5.1: An area-efficient implementation of the serial-serial finite field multiplier

5.3 A Systolic Array Implementation

A systolic array is defined as a regular array of similar processing cells with localized, regular interconnection. Systolic arrays are very attractive for an implementation due to the following:

- Regular cell design enables high-density VLSI implementations, which leads directly to higher performance and lower overhead.
- Strict local interconnection eliminates routing problems associated with broadcast lines.
- Power consumption is reduced due to the reduced logic depth.

To implement the proposed serial-serial finite field multiplier as a systolic array, it is required to insert delays into the four broadcast lines that carry the inputs. This can be achieved by repositioning each other delay from between the cells to the broadcast lines, as shown in Figure 5.2. Such modification preserves the original functionality of the structure. To achieve regularity, each pair of consecutive cells are grouped into a single cell of the systolic array.

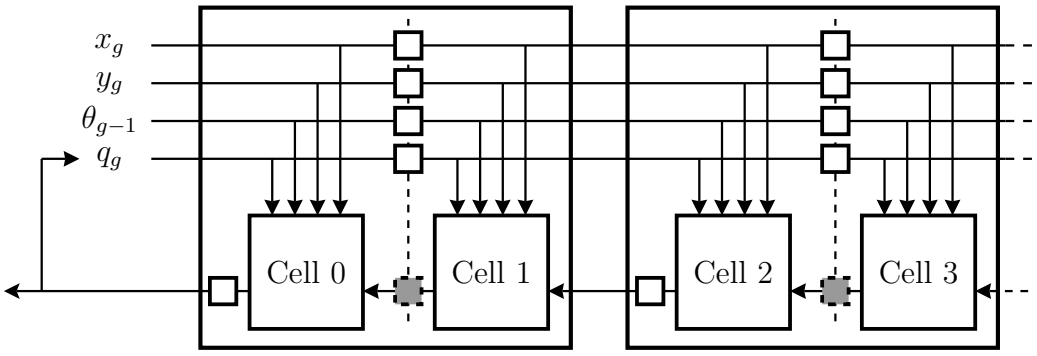

Figure 5.2: A systolic implementation of the serial-serial finite field multiplier

5.4 A Pipelined Implementation

Pipelining is a technique in which a functional unit is segmented such that it can accept new operands every cycle while the generation of the output takes more than one cycle. Although pipelining is achieved by inserting delays into the datapath, increasing the required area, the potential for multiplying the throughput of the functional unit makes the net effect worthwhile.

Pipelining a structure in which the dataflow is uni-directional is a trivial task. However, when there exist feedback links, delays cannot be inserted unless some conditions (the cut-set rules) are satisfied.

As it is, Algorithm 1 does not allow pipelining because of the dependency of each iteration on the one directly preceding it. In order to be able to pipeline our structure, Algorithm 1 has to be modified to allow pipelining by breaking the dependency between consecutive iterations. This can be achieved through correcting for a discarded digit in an later iteration rather than in the directly following iteration.

Let δ denote the number of iterations for which the correction is delayed, i.e. the digit generated in iteration i will be corrected for in iteration $i + \delta$, and let $\Theta = \mu^{-\delta} \bmod M$. Then, Algorithm 1 can be rewritten to enable pipelining as Algorithm 3.

Note that Algorithm 3 reduces to Algorithm 1 when $\delta = 1$. Using this algorithm, it is possible to pipeline the cells of serial-serial finite field multiplication structure to any number of stages by setting δ to that number. For example, Figure 5.3 shows

Algorithm 3 Discard-based Least-to-most Finite Field Multiplication with Delayed Correction for δ Cycles

Require: $2X, 2Y < \mu^n$, $4M < \mu^{n-1}$
Ensure: $R = XY\mu^{-n-\delta+1} (\mathrm{mod}\, M)$, $2R < \mu^n$
1: $R_{-1} = 0$, $R_{-2} = 0$, ..., $R_{-\delta} = 0$
2: **for** $i = 0 \ldots n + \delta - 1$ **do**
3: $q_i = R_{i-\delta} \bmod \mu$ ▷ q_i is the least-significant digit of $R_{i-\delta}$
4: $R_i = R_{i-1} \operatorname{div} \mu + x_i Y + q_i \Theta$
5: **end for**
6: $R = R_{n+\delta-1}$

the resulting structure with $\delta = 4$, i.e. with 4 pipeline stages.

5.5 A Scalable Implementation

Scalability is defined as the ability to perform high-precision operations using low-precision units. This is especially important in cryptographic applications due to the continuous growth of cryptographic parameters to satisfy security requirements. For an unscalable operation, it would be required to replace the hardware whenever the operand wordlength is changed.

Multiplication is an inherently scalable operation since sub-words can be multiplied independently and the final result would be the sum of the results of all sub-word products aligned to the correct significance. The same applies to finite field multiplication since it can be partitioned into two overlapping integer multiplications.

Equation 4.18 can be rewritten in a scalable way as follows. Let ϵ denote the size of the sub word in digits. Then, equation can be rewritten as

$$
\begin{aligned}
R &= \mu^{-n}\left(\sum_{i=0}^{n} x_i \sum_{j=0}^{n} y_j \mu^{i+j} + \sum_{k=0}^{n} q_k \sum_{h=0}^{n} \theta_{h-1}\mu^{k+h-1}\right) \\
&= \mu^{-n} \sum_{\phi=0}^{n/\epsilon-1} \sum_{\tau=0}^{n/\epsilon-1} \mu^{(\phi+\tau)\epsilon} \left(\sum_{i=0}^{\epsilon-1} x_{i+\phi\epsilon} \sum_{j=0}^{\epsilon-1} y_{j+\tau\epsilon}\mu^{i+j}\right. \\
&\quad \left. + \sum_{k=0}^{\epsilon-1} q_{k+\phi\epsilon} \sum_{h=0}^{\epsilon-1} \theta_{h-1+\tau\epsilon}\mu^{k+h-1}\right) \qquad (5.1)
\end{aligned}
$$

where τ and ϕ denote the block indecies. This can be rewritten as

$$
R = \mu^{-n} \sum_{\phi=0}^{n/\epsilon-1} \sum_{\tau=0}^{n/\epsilon-1} \mu^{(\phi+\tau)\epsilon} \left(S_{\epsilon-1}^{(\phi,\tau)}\right) \qquad (5.2)
$$

where $S_{\epsilon-1}^{(\phi,\tau)}$ is defined as

$$
S_{\epsilon-1}^{(\phi,\tau)} = \sum_{i=0}^{\epsilon-1} x_{i+\phi\epsilon} \sum_{j=0}^{\epsilon-1} y_{j+\tau\epsilon}\mu^{i+j} + \sum_{k=0}^{\epsilon-1} q_{k+\phi\epsilon} \sum_{h=0}^{\epsilon-1} \theta_{h-1+\tau\epsilon}\mu^{k+h-1} \qquad (5.3)
$$

The expression for $S_{\epsilon-1}^{(\phi,\tau)}$, which describes the functionality of a single block, is in fact the same as the expression in equation 4.18, which describes the whole multiplication operation. The summations signified by the outer summations can be implemented in any way as long as the relative significance of blocks is taken into consideration. Similarly, the block multiplication can be implemented in any suitable way. In other words, the previous equation defines two two-dimensional summations, each of which can be implemented in any suitable way.

Implementing the block multiplication in a serial-serial manner is achieved by

defining $S_{\epsilon-1}^{(\phi,\tau)}$ as a serial-serial recurrence as follows.

$$\begin{aligned} S_{\epsilon-1}^{(\phi,\tau)} &= x_{\epsilon-1+\phi\epsilon} y_{\epsilon-1+\tau\epsilon} \mu^{2(\epsilon-1)} \\ &\quad + x_{\epsilon-1+\phi\epsilon} \sum_{j=0}^{\epsilon-2} y_{j+\tau\epsilon} \mu^{\epsilon-1+j} + y_{\epsilon-1+\tau\epsilon} \sum_{i=0}^{\epsilon-2} x_{i+\phi\epsilon} \mu^{\epsilon-1+i} \\ &\quad + q_{\epsilon-1+\phi\epsilon} \theta_{\epsilon-2+\tau\epsilon} \mu^{2(\epsilon-1)-1} \\ &\quad + q_{\epsilon-1+\phi\epsilon} \sum_{h=0}^{\epsilon-2} \theta_{h-1+\tau\epsilon} \mu^{\epsilon-2+h} + \theta_{\epsilon-2+\tau\epsilon} \sum_{k=0}^{\epsilon-2} q_{k+\phi\epsilon} \mu^{\epsilon-2+k} \\ &\quad + \mu^{-1} S_{\epsilon-2}^{(\phi,\tau)} \end{aligned} \tag{5.4}$$

Independently, the outer summations can also be implemented in a serial-serial manner. It should also be noted that while implementing a scalable finite field multiplier, it is possible to map the blocks to one or more block multipliers. The blocks can be evaluated and accumulated in any order provided that the accumulation mechanism takes care of the respective significance of the blocks. Also, the internal structure of each block multiplier is flexible itself, i.e. any multiplication algorithm can be used to implement the block multiplier, including serial-serial multiplication as already illustrated.

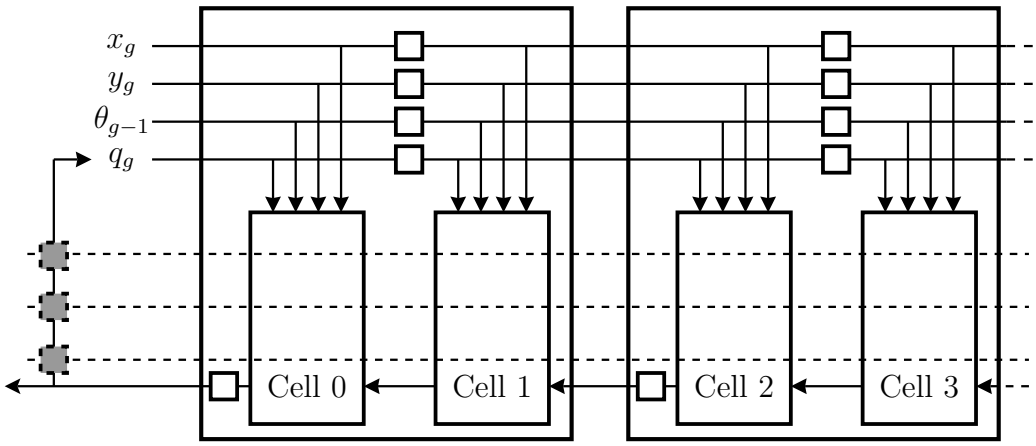

Figure 5.3: A pipelined implementation of the serial-serial finite field multiplier

Chapter 6

Conclusion

6.1 Introduction

This chapter concludes the thesis by outlining the contributions achieved. Moreover, it discusses some of the open problems for future work.

6.2 Summary of the Contributions

In this thesis, the following has been achieved:

1. An inherently serial-serial formulation of finite field multiplication operation has been introduced.
2. A serial-serial prime field multiplier has been designed as an example of the

resulting new class of serial-serial finite field multipliers.

3. Modifications have been applied to the example multiplier to reduce its area by 50%, increase its throughput, impelement it as a systolic array, and exploit its inherent scalability.

6.3 Future Work

The proposal of this new time-space mapping opens the way for future work in various dimensions. For example,

- **Alternative algorithms**: The algorithm adopted for creating a serial-serial finite field multiplier is one of many other algorithms for finite field multiplication. It may be possible to get better formulations using other algorithms.
- **Performance issues**: The multiplier that was designed as an example is by no means optimized for maximum performance. Better performance can be certainly achieved through low-level optimizations.
- **Serialization methodology**: The methodology applied in this work to map a dataflow into a serial-serial structure can be used to serialize other arithmetic operations.

Nomenoclature

$\mathbb{F}_q$: The finite field of order q.

$\mathbb{Z}_M$: The set of integers modulo M.

μ: The representation radix.

M: p in $\mathbb{F}_p$, $G(x)$ (the generator polynomial) in $\mathbb{F}_{p^m}$, and M in $\mathbb{Z}_M$.

X: The multiplier.

Y: The multiplicand.

R: The finite field multiplication result.

Θ: A constant derived from M used to correct for discarded digits.

S_i: A serial-serial partial-sum.

q_i: A discarded digit.

δ: The number of delay cycles in a pipelined finite field multiplication structure.

Bibliography

[1] A. Aggoun, A. Ashur, and M. K. Ibrahim. Area-time efficient serial-serial multipliers. In *ISCAS 2000: The 2000 IEEE International Symposium on Circuits and Systems, 2000*, pages 585–588. IEEE, May 2000.

[2] Giuseppe Alia and Enrico Martinelli. A vlsi modulo m multiplier. *IEEE transactions on computers*, 40:873–878, July 1991.

[3] David Narh Amanor. Efficient hardware architectures for modular multiplication. Master's thesis, The University of Applied Sciences Offenburg, Germany, February 2005.

[4] A. S. Ashur, M. K. Ibrahim, and A. Aggoun. Systolic digit-serial multiplier. In *IEE Proceedings on Circuits, Devices and Systems*, pages 14–20. IEE, February 1996.

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

[6] Gary Baldwin, Bernard Morris, David Fraser, and Angelo Tretola. A modular, high-speed serial pipeline multiplier for digital signal processing. *IEEE Journal of solid-state circuits*, 13:400–408, June 1978.

[7] Jean-Luc Beuchat. *A VHDL Library for Integer and Modular Arithmetic*, 0.1 edition, September 2004.

[8] P. B. Bhattacharya, S. K. Jain, and S. R. Nagpaul. *Basic Abstract Algebra*. Cambridge University Press, 2 edition, 1995.

[9] V. Bunimov and M. Schimmler. Area and time efficient modular multiplication of large integers. In *ASAP '03: IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, June 2003.

[10] V. Bunimov and M. Schimmler. A simple algorithm for time optimal modular multiplication and exponentiation. In *IASTED '03: Proceedings of the 15th International Conference Parallel and Distributed Computing and Systems*, November 2003.

[11] V. Bunimov and M. Schimmler. Area-time optimal modular multiplication. In *Embedded Cryptographic Hardware: Methodologie & Architectures*, chapter 10. State Univ. of Rio de Janeiro, 2004.

[12] V. Bunimov and M. Schimmler. High radix modular multiplication of large integers optimised with respect to area and time. In *The 2004 International Conference on VLSI*, June 2004.

[13] V. Bunimov, M. Schimmler, and B. Tolg. A complexity-effective version of montgomery's algorithm. In *WCED '02: Workshop on Complexity Effective Designs*, May 2002.

[14] Edmund K. Cheng and Carver A. Mead. A twoś complement pipeline multiplier. In *ICASSP '76: IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 647 – 650. IEEE, April 1976.

[15] C. W. Chiou and T. C. Yang. Iterative modular multiplication algorithm without magnitude comparison. *Electronics Letters*, 30:2017–2018, November 1994.

[16] Hans Eberle, Nils Gura, Sheueling Chang Shantz, and Vipul Gupta. A cryptographic processor for arbitrary elliptic curves over GF(2^m). Technical report, Sun Microsystems, May 2003.

[17] Stephen E. Eldrige and Colin D. Walter. Hardware implementation of montgomery's modular multiplication algorithm. *IEEE transactions on computers*, 42:693–699, June 1993.

[18] Shimon Even. Systolic modular multiplication. In A. J. Menezes and S. A. Vanstone, editors, *Lecture notes in computer sciences; 537 on CRYPTO '90: Advances in cryptology*, pages 619–624. Springer-Verlag, 1991.

[19] William L. Freking and Keshab K. Parhi. Montgomery modular multiplication and exponentiation in the residue number system. In *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers*, pages 1312–1316. IEEE, October 1999.

[20] William L. Freking and Keshab K. Parhi. Parallel modular multiplication with application to vlsi rsa implementation. In *ISCAS '99. Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, pages 490–495. IEEE, May 1999.

[21] William L. Freking and Keshab K. Parhi. A unified method for iterative computation of modular multiplication and reduction operations. In *ICCD '99: International Conference on Computer Design*, pages 80–87. IEEE, October 1999.

[22] Weixin Gai and Hongyi Chen. A systolic linear array for modular multiplication. In *2nd International Conference on ASIC*, pages 171–174. IEEE, October 1996.

[23] Gunnar Gaubatz. Versatile montgomery multiplier architectures. Master's thesis, Worcester Polytechnic Institute, April 2002.

[24] Jyh-Huei Guo and Chin-Liang Wang. A novel digit-serial systolic array for modular multiplication. In *ISCAS '98: IEEE International Symposium on Circuits and Systems*, pages 177–180. IEEE, June 1998.

[25] Jin-Hua Hong and Cheng-Wen Wu. Radix-4 modular multiplication and exponentiation algorithms for the rsa public-key cryptosystem. In *Proceedings of the ASP-DAC 2000. Asia and South Pacific Design Automation Conference*, pages 565–570. IEEE, January 2000.

[26] Yeong-Jiunn Juang, Erl-Huei Lu, Jau-Yien Lee, and Chin-Hsing Chen. A new architecture for fast modular multiplication. In *International Symposium on VLSI Technology, Systems and Applications*, pages 357–360. IEEE, May 1989.

[27] Min-Sup Kang and Dong-Wook Kim. Systolic array based on fast modular multiplication algorithm for rsa cryptosystem. In *Proceedings of the IEEE Region 10 Conference*, pages 305–308. IEEE, September 1999.

[28] Min-Sup Kang and F. J. Kurdahi. A novel systolic vlsi architecture for fast rsa modular multiplication. In *IEEE Asia-Pacific Conference on ASIC*, pages 81–84. IEEE, August 2002.

[29] Chinuk Kim. VHDL implementation of systolic modular multplications on RSA cryptosystem. Master's thesis, The City college of the City University of New York, January 2001.

[30] Sungwook Kim and Gerald E. Sobelman. Digit-serial modular multiplication using skew-tolerant domino CMOS. In *ICASSP '01: IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1173–1176. IEEE, May 2001.

[31] N. Koblitz. *A course in number theory and cryptography.* Springer, 1987.

[32] Çetin Kaya Koç. Montgomery reduction with even modulus. *IEE Proceedings on Computers and Digital Techniques*, 141(5):314–316, September 1994.

[33] Çetin Kaya Koç. High-speed RSA implementation. Technical Report TR 201, RSA Security, November 1996.

[34] Çetin Kaya Koç, Tolga Acar, and Burton S. Kaliski, Jr. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[35] Peter Kornerup. High-radix modular multiplication for cryptosystems. In *ARITH '93: Proceedings of the 11th Symposium on Computer Arithmetic*, pages 277–283. IEEE Computer Society, July 1993.

[36] Peter Kornerup. A systolic, linear-array multiplier for a class of right-shift algorithms. *IEEE Transactions on Computers*, 43:892–898, August 1994.

[37] Taek-Won Kwon, Chang-Seok You, Won-Seok Heo, Yong-Kyu Kang, and Jun-Rim Choi. Two implementation methods of a 1024-bit rsa cryptoprocessor based on modified montgomery algorithm. In *The 2001 IEEE International Symposium on Circuits and Systems*, pages 650–653. IEEE, May 2001.

[38] M. Mekhallalati, M. K. Ibrahim, and A. Ashur. Radix modular multiplication algorithm. *Journal of Circuits, Systems, and Computers*, 6:547–567, 1996.

[39] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, 5th edition, August 2001.

[40] Victor S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Lecture notes in computer sciences; 218 on CRYPTO '85: Advances in cryptology*, pages 417–426. Springer-Verlag, 1986.

[41] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.

[42] Nadia Nedjah and Luiza de Macedo Mourelle. Reconfigurable hardware implementation of montgomery modular multiplication and parallel binary exponentiation. In *Proceedings. Euromicro Symposium on Digital System Design*, pages 226–233. IEEE, September 2002.

[43] Nadia Nedjah and Luiza de Macedo Mourelle. Two hardware implementations for the montgomery modular multiplication: sequential versus parallel. In *Proceedings. 15th Symposium on Integrated Circuits and Systems Design*, pages 3–8. IEEE, September 2002.

[44] O. Nibouche, A. Bouridane, and M. Nibouche. Bit-level architectures for montgomery's multiplication. In *ICECS 2001: The 8th IEEE International Conference on Electronics, Circuits and Systems*, pages 273–276. IEEE, September 2001.

[45] O. Nibouche, A. Bouridane, and M. Nibouche. New iterative algorithms and architectures of modular multiplication for cryptography. In *ICECS 2001: The 8th IEEE International Conference on Electronics, Circuits and Systems*, pages 879–882. IEEE, September 2001.

[46] W. Keith Nicholson. *Introductio to Abstract Algebra*. PWS Publishing Company, 1993.

[47] J. H. Oh and S. J. Moon. Modular multiplication method. *IEE Proceedings on Computers and Digital Techniques*, 145(4):317–318, July 1998.

[48] Siddika Berna Örs, Lejla Batina, Bart Preneel, and Joos Vandewalle. Hardware implementation of a montgomery modular multiplier in a systolic array. In *IPDPS '03: Proceedings of the international parallel and distributed processing symposium*. IEEE, April 2003.

[49] Holger Orup. Simplifying quotient determination in high-radix modular multiplication. In *ARITH '95: Proceedings of the 12th Symposium on Computer Arithmetic*, pages 193–199. IEEE Computer Society, July 1995.

[50] Christof Paar. Implementation options for finite field arithmetic for elliptic curve cryptosystems. In *ECC '99: The 3rd workshop on Elliptic Curve Cryptography*, November 1999.

[51] H. Ploog and D. Timmermann. On multiple precision based montgomery multiplication without precomputation of $n'_0 = -n_0^{-1} \bmod w$. In *Proceedings of the International Conference on Computer Design*, pages 589–590. IEEE, September 2000.

[52] Janardhan H. Satyanarayana and Keshab K. Parhi. A theoretical approach to estimation of bounds on power consumption in digital multipliers. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44:473–481, June 1997.

[53] Johann Groß schädl. A low-power bit-serial multiplier for finite fields $gf(2^m)$. In *ISCAS '01: IEEE International Symposium on Circuits and Systems*, pages 37–40. IEEE, May 2001.

[54] M. Schimmler and V. Bunimov. Fast modular multiplication by operand changing. In *ITCC '04: International Conference on Information Technology*, April 2004.

[55] Sheueling Chang Shantz. From euclid's gcd to montgomery multiplication to the great divide. Technical report, Sun Microsystems, June 2001.

[56] Jia-Lin Sheu, Ming-Der Shieh, Chien-Hsing Wu, and Ming-Hwa Sheu. A pipelined architecture of fast modular multiplication for rsa cryptography. In *ISCAS '98: Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, pages 121–124. IEEE, May 1998.

[57] Jun-Bum Shin, Joungkyou Kim, and Hyung Lee-Kwang. Optimisation of montgomery modular multiplication algorithm for systolic arrays. *Electronics Letters*, 34(19):1830–1831, September 1998.

[58] Li Shuguo, Zhou Runde, and Ge Yuanqing. A 1024-bit RSA crypto-coprocessor for smart cards. In *4th International Conference on ASIC*, pages 352–355. IEEE, October 2001.

[59] Leilei Song and Keshab K. Parhi. Efficient finite field serial/parallel multiplication. In *ASAP '96: Proceedings of International Conference on Application Specific Systems, Architectures and Processors*, pages 72–82. IEEE, August 1996.

[60] Naofumi Takagi. A modular multiplication algorithm with triangle additions. In *11th Symposium on Computer Arithmetic*, pages 272–276. IEEE, July 1993.

[61] Alexandre F. Tenca and Çetin Kaya Koç. A scalable architecture for montgomery multiplication. In Çetin Kaya Koç and C. Paar, editors, *First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 94–108. Springer Verlag, August 1999.

[62] Alexandre F. Tenca, Georgi Todorov, and Çetin Kaya Koç. High-radix design of a scalable modular multiplier. In Çetin Kaya Koç, D. Naccache, and C. Paar,

editors, *CHES '01: Cryptographic Hardware and Embedded Systems*, pages 189–205. Springer Verlag, May 2001.

[63] Georgi Todorov. ASIC design, implementation and analysis of a scalable high-radix montgomery multiplier. Master's thesis, Oregon State University, 2000.

[64] Ivar Tångring. A design study of an arithmetic unit for finite fields. Master's thesis, Linköping University, June 2003.

[65] Wei-Chang Tsai, C.B. Shung, and Sheng-Jyh Wang. Two systolic architectures for modular multiplication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(1):103–107, February 2000.

[66] Colin D. Walter. Systolic modular multiplication. *IEEE Transactions on Computers*, 42(3):376–378, March 1993.

[67] Colin D. Walter. Still faster modular multiplication. *Electronics Letters*, 31(4):263–264, February 1995.

[68] Colin D. Walter. Space/time trade-offs for higher radix modular multiplication using repeated addition. *IEEE transactions on Computers*, 46(2):139–141, February 1997.

[69] Colin D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35(21):1831–1832, October 1999.

[70] Jhing-Fa Wang, Po-Chuan Lin, and Ping-Kun Chiu. A staged carry-save-adder array for montgomery modular multiplication. In *IEEE Asia-Pacific Conference on ASIC*, pages 97–100. IEEE, Aug 2002.

[71] Albert Tung-Hoe Wong. A new scalable systolic array processor architecture for discrete convolution. *University of Kentucky*, 2003.

[72] Ching-Chao Yang, Tian-Sheuan Chang, and Chien-Wei Jen. A new rsa cryptosystem hardware design based on montgomery's algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(7):908–913, July 1998.

[73] T. Yanik, E. Savas, and C. K. Koc. Incomplete reduction in modular arithmetic. *IEE Proceedings on Computers and Digital Techniques*, 149:46–52, March 2002.

[74] C. N. Zhang, Y. Xu, and C. C. Wub. A bit-serial systolic algorithm and vlsi implementation for rsa. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 523–526. IEEE, August 1997.

Vita

Abdulaziz Mohammad Alkhoraidly was born on January 10th, 1980 in Onaizah, Saudi Arabia. He obtained a B.S. in Computer Science (2002) and a B.S. in Computer Engineering (2002), both with first honors, from King Fahd University of Petroleum and Minerals (KFUPM). Since then, he has worked as a graduate assistant in the department of Information and Computer Science in KFUPM while pursuing his M.S. degree in Computer Science. His research interests include cryptography and computer arithmetic. Abdulaziz can be reached on `azizkh@kfupm.edu.sa` or `azizkh@gmail.com`.