Hardware Architectures of Elliptic Curve Based Cryptosystems over Binary Fields

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Chang Shu
B.S. Electrical Engineering
Tsinghua University, P.R. China, 1999
M.S. Electrical Engineering
Graduate School
Chinese Academy of Sciences, P.R. China, 2002

Director: Dr. Kris Gaj
Associate Professor
Electrical and Computer Engineering

Spring Semester 2007
George Mason University
Fairfax, VA

HARDWARE ARCHITECTURES OF ELLIPTIC CURVE BASED CRYPTOSYSTEMS OVER BINARY FIELDS

by

Chang Shu

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Electrical and Computer Engineering

Committee:

KGaj
Dr. Kris Gaj, Dissertation Director

Shih Chun Chang
Dr. Shih-Chun Chang, Committee Member

Brian L. Mark
Dr. Brian L. Mark, Committee Member

Dr. Ravi Sandhu, Committee Member

Dr. Andre Manitius, Chairman
Department of Electrical and Computer Engineering

Dr. Lloyd Griffiths, Dean
The Volgenau School of Information Technology and Engineering

Date: 02/08/2007

Spring Semester 2007
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my parents and my brother for their endless love. Thank you!

Acknowledgments

This research was partially supported by the research project, Library Development and Experiments using Prototype Reconfigurable Parallel Computers (LUCITE), sponsored by Department of Defense.

First, I want to thank my advisor, Prof. Kris Gaj (ECE of GMU). I owe a lot to him for the tremendous source of knowledge and inspiration that he has been to me. He led me to the area, efficient hardware design for elliptic curve based cryptosystems, and guided me throughout the dissertation work. I could not forget many weekends that we worked together to resolve the hurdles that I encountered throughout the research, and many nights that he spent on going over my draft papers. I have benefited from his constructive criticism, invaluable advice and many discussions. I can hardly ask more from an advisor. I am grateful to Prof. Soonhak Kwon (Mathematics Dept. at Sungkyunkwan University, Korea). He introduced me to the subject of Tate pairing cryptosystems, and helped me in the formulation and execution of the part of Tate pairing of my research work.

I owe special thanks to other dissertation committee members, Prof. Shih-Chun Chang (ECE of GMU), Prof. Brian L. Mark (ECE of GMU), and Prof. Ravi Sandu (ISE of GMU). They provided sustained guidance and advice. It is hard for me to complete my Ph.D. without their support.

I would like to thank Prof. Andre Manitius (ECE Chair of GMU) and Prof. Yariv Ephraim (the Ph.D. coordinator in the ECE Department at GMU) for their kind suggestions of my Ph.D. studies.

I am also grateful to Prof. Tarek El-Ghazawi (ECE of the George Washington University), Dr. Mohamed Taher (ECE of GWU), Proshanta Saha, Esam Al-Araby

and Miaoqing Huang, doctoral candidates (ECE of GWU), for their valuable help when I participated in the LUCITE project.

I owe a debt of thanks to numerous people who helped me make the thesis work a reality. These people include Brian Larson, Dick Riegner (Silicon Graphics, Inc.), and Frederic Raynal (ST Microelectronics, Inc.). Even though I cannot list here all of the people who helped me accomplish this work, nevertheless I am indebted to all of them.

Finally, I must thank my parents in China who have been behind me all the time. It is difficult to imagine reaching this point without their support.

Shu, Chang

Fairfax, VA, USA

January, 2007

Table of Contents

List of Tables

List of Figures

Abstract

HARDWARE ARCHITECTURES OF ELLIPTIC CURVE BASED CRYPTOSYSTEMS OVER BINARY FIELDS

Chang Shu, Ph.D.

George Mason University, 2007

Dissertation Director: Dr. Kris Gaj

Public key cryptosystems were invented in 1976 by Whitfield Diffie and Martin Hellman to solve the security problems such as confidentiality, authenticity, integrity and non-repudiation during communications in public networks. The first practical realization, namely RSA, was proposed by Ron Rivest, Adi Shamir and Len Adleman in 1977. The security of RSA is based on the hardness of factoring large integers. In 1985, Neal Koblitz and Victor Miller independently discovered elliptic curve cryptography (ECC), which can provide the same functionality as RSA. Its security is based on the elliptic curve discrete logarithm problem (ECDLP). Compared with RSA, ECC can achieve the same security strength with smaller key sizes. Weil and Tate pairings were originally used to break elliptic curve protocols. However since Joux proposed the idea of applying pairings in tripartite Diffe-Hellman protocol many scholars have been interested in constructing new cryptographic protocols based on pairing schemes.

This thesis concentrates on the efficient hardware architectures of elliptic curve based cryptosystems over binary fields. These architectures require superior design for finite field arithmetic. We derive the digit-serial multiplier for these underlying

fields constructed via low Hamming weight irreducible polynomials such as trinomials or pentanomials. For normal basis (NB) multipliers, the methods of shortening the critical path and simplifying the complexity are also covered. We present a new hybrid architecture whereby not only the circuit complexity can be decreased but also the basis conversion can be simplified.

Aside from the field arithmetic unit, other issues such as top architecture, parallel computations, sharing resource and efficient control should be necessarily considered to improve the design efficiency. Different target platforms for ECC are investigated. We have designed a low latency ECC accelerator via a single FPGA device based on López-Dahab algorithm, which can run 1.5-to-3 times faster than the one designed by the research group of Sun Microsystems, with approximately the same cost. Furthermore the design methodology for ECC library aimed for a reconfigurable computer, SRC-6, is addressed.

Tate pairing based cryptosystems have recently emerged as an alternative to traditional public key cryptosystems, because of their ability to be used in multi-party identity-based key management schemes. Due to the inherent parallelism of the existing pairing algorithms, high performance can be achieved via hardware realizations. In this work, a new FPGA-based architecture of the Tate pairing-based computation over binary fields was proposed. The computational latency for one pairing has been reduced, and our implementation runs 10-to-20 times faster than the equivalent implementations of other pairing-based schemes at the same level of security strength. Additionally, an improvement in the product of latency by area by a factor between 12 and 46 for an equivalent type of implementation has been achieved. Furthermore, we have also ported our pairing designs for 8 field sizes ranging from 239 to 557 to the reconfigurable computer, SGI Altix-4700 supported by Silicon Graphics, Inc., and the performance and cost have been characterized.

Our research demonstrates that both software and hardware implementations of pairing-based cryptosytems can operate at a similar speed and security level as traditional elliptic curve cryptosystems.

Chapter 1: Introduction

1.1 Objective

The concept of public-key cryptography was first proposed in [12] by Whitfield Diffie and Martin Hellman in 1976 to provide an elegant way to distribute keys over an insecure communication channel. Even though symmetric key algorithm is more efficient than public-key algorithm in encryption or authentication, public-key cryptography can provide several functionalities such as key exchange, key establishment and digital signature which can not be realized via symmetric key cryptography.

The first practical realization of public-key cryptography, well-known RSA, was conceived by Ron Rivest, Adi Shamir and Len Adleman in 1977. Its security is based on the hardness of integer factorization. Alternatively, the difficulty of solving discrete logarithm problems in finite fields or in the group of algebraic curves over a finite field can be also exploited to construct other public-key cryptosystems, such as elliptic curve cryptography [4, 5] and Tate pairing based cryptography [16].

Compared with RSA, ECC can be built over binary fields to allow more efficient implementations. Second it can achieve the same security strength as RSA with smaller key sizes. These advantages attract the research community as well as the industry to study the efficient realization of ECC. In particular, there are quite a few standards involving EC, such as ANSI X9.62 [1] and IEEE P1363 [7]. Several basic issues need to be considered for the hardware implementations of ECC: 1) application driven requirements, i.e., different target platforms; 2) efficient top algorithm for

scalar multiplication; 3) top architecture; and 4) efficient underlying field arithmetic. As different target platforms have different requirements of performance and cost, the design methodology may vary. The platforms we investigate for ECC range from a single FPGA device to a reconfigurable computer, SRC-6. López-Dahab algorithm is chosen for the top architecture. The second important issue we address is the efficient design for the underlying field arithmetic, which absolutely determines the performance of ECC accelerator.

Weil and Tate pairings were originally used to attack the ECC protocol. However since the possiblity of its application in tripartite Diffie-Hellman protocol was recognized, more and more scholars have been interested in constructing protocols which can not be constructed in another way. In addition, pairings based on elliptic curves can be also applied in identity-based cryptosystems where any string, such as email address, can be a valid public key. By now several research groups [33,35] have published their hardware implementation results for cubic fields by which smaller underlying field size can achieve the same security strength as 1024-bit RSA cryptosystem. However these architectures sacrifice the efficiency of field arithmetic. In case that binary fields are used to underly the elliptic curves, larger field sizes or embedded degrees should be chosen to realize the same security strength. However for larger embedded degrees, it is unavoidable to have a more complicated top algorithm. For instance, the embedded degree is chosen as 12 in [65], which requires more multiplexers for steering data and more multipliers working in parallel to achieve high operation speed. Kwon [32] proposed an algorithm of Tate pairing with elliptic curves over binary fields with embedded degree 4. This algorithm is much simpler than the one in [65], i.e., the complexity of datapath can be decreased considerably, whereas larger underlying field size will be used. Therefore, it is necessary to perform some

experiments to decide which scheme is better in terms of latency and area.

1.2 Summary of Research Contributions

In this thesis, we focus on the hardware architectures of EC based cryptosystems. Our contributions to this research area can be summarized as follows:

1. Deriving the highly efficient digit-serial multiplier via low Hamming weight irreducible polynomials based on left-to-right algorithm and a new hybrid multiplier for which the basis conversion technique is also covered.

2. Implementations of ECC over binary fields recommended by NIST [17] via a single FPGA device which can run 1.5-to-3 times as fast as the one developed by Sun Microsystems [63].

3. The ECC library development for a reconfigurable computing system, SRC-6 [76], in which different partitioning schemes and design methodologies were considered.

4. The first realizations of Tate pairing cryptosystems on binary elliptic curves via FPGAs which can run 10-to-20 times faster than the other two schemes, cubic elliptic and binary hyperelliptic, at the same level of security strength. Moreover, an improvement in the product of latency by area by a factor between 12 and 46 for an equivalent type of implementation has been achieved.

5. Reconfigurable computing approach for Tate pairing cryptosystems over binary fields. The reconfigurable computer, SGI Altix-4700 supported by Silicon Graphics, Inc., is chosen for our pairing designs for 8 field sizes ranging from

239 to 557. This is a first complete investigation of porting pairing on binary elliptic curves to an end-to-end system.

1.2.1 New Architectures of Binary Field Multipliers

Since the computations of EC-based cryptography contain intensive field multiplications, the efficiency of multipliers directly determines the performance of the whole cryptographic processor. Compared with bit-serial or bit-parallel multipliers, digit-serial multiplier is frequently used in EC based cryptosystems as it allows tradeoff between timing and area. If a polynomial basis is generated by the root of a trinomial or a pentanomial, then the circuit complexity can be simplified. Most significant digit (MSD) serial multipliers have lower circuit complexity and lower power consumption than the least significant digit (LSD) serial multiplier. We derive MSD serial multipliers for the fields constructed via trinomials or pentanomials. Composite field arithmetic should be necessarily considered for Tate pairing and ECC over composite fields in case that the loss of security strength is not significant. We propose a new composite field multiplier where a normal basis is chosen for the top level (bit-serial structure) and a trinomial basis is chosen for the ground level (bit-parallel structure). This hybrid structure is superior not only because of its lower circuit complexity but also because of its simplicity for basis conversion. Additionally, we are the first to perform comparisons of the efficiency between the conventional digit-serial multipliers and the hybrid multipliers in terms of FPGA implementations. For the special case that the extension field can be constructed only via a pentanomial and in turn both the top and the ground fields can be constructed via trinomials, this kind of hybrid multiplier can achieve lower product of latency by area.

1.2.2 Optimizations of ECC Processor for a Single FPGA Device

Several researchers have published their implementations of ECC via FPGA [56–59, 63] or ASIC technology [2]. All these elliptic curve cryptographic accelerators can be categorized as two functional groups, see [59]. They are

1. Accelerators which use general purpose processors to implement curve operations but implement the finite field operations using hardware.

2. Accelerators which perform both the curve and field operations in hardware.

The disadvantage of the first approach is that operation speed is deterred by the overhead of the communication between processor and memory. In addition it takes more time to decode the instruction synchronously even for a simple field addition. We adopt the second architecture containing a main controller, arithmetic and logic unit (ALU) and register files. Top algorithm for ECC is always first considered to decide the top architecture. In our implementations, we choose López-Dahab algorithm [41] using projective coordinates. The computations based on this algorithm can be divided mainly into two stages, group doubling-addition and coordinate conversion. Compared with other algorithms, this algorithm requires fewest number of multiplications involved in each doubling-addition round and only one multiplicative inversion is executed at the end. Most of computations concentrate on the first stage and can be completed using an iterative structure. Those 6 multiplications involved in each iteration can be performed in parallel and completed in 2 multiplication rounds in case of 3 multipliers adopted. To achieve high performance, the digit sizes of these multipliers inside the iterative structure should be large, e.g., 16 or 32. And they should not be shared by other computations involved in the second stage to avoid

complicated datapath. The coordinate conversion involves ten multiplications, one inversion and several additions. However since the inputs of these 10 multiplications come from different places, the complexity of datapath will be increased significantly if all multiplications are completed by one multiplier. Therefore we use one additional multiplier together with the one inside the inverter to finish these 10 multiplications. Since the coordinate conversion is performed only once, the digit sizes of multipliers in the converter should be small to save area without loss of performance. By now, the ALUs designed by other researchers contain only one multiplier so that multiplications are performed sequentially. Our accelerators support parallel computations and have a simpler datapath than others. We compare the performance and cost of our ECC accelerators with the best accelerators published in CHES'02 [63] considering that the same FPGA device and the same top algorithm are selected for both designs. The underlying fields are $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$ recommended by NIST [17] in which pentamomial basis and trinomial basis are chosen to represent the field elements. As a result, our accelerators can run 1.5-to-3 times faster than those reported by Sun Microsystems of CHES'02 with almost the same resource utilizations.

1.2.3 Building ECC Macro Library for a Reconfigurable Computer

Reconfigurable Computers, SRC-6, are high-end computers based on the close system-level integration of traditional microprocessors and FPGAs which offer the promise of performing computations in hardware to increase performance and efficiency while retaining much of the flexibility of a software solution. Computations on such a reconfigurable system can be described as 1) High Level Language (HLL) function running on a traditional microprocessor; 2) HLL function running on an FPGA, or

3) Hardware Description Language (HDL) macro running on an FPGA. According to the hierarchy of operations involved in ECC, we investigated different partitioning schemes and performed analysis in terms of the trade-off between SRC-6 programming verus the traditional hardware design model. Furthermore we develop hardware library in VHDL for both the binary field operations and the group functions of ECC with two different versions based on the polynomial basis and the normal basis representation of the field elements. Performance comparisons with the open-source software implementations are demonstrated.

1.2.4 FPGA Accelerated Tate Pairing Based Cryptosystems

We propose a low latency hardware accelerator for the Tate pairing-based cryptosystems on supersingular elliptic curves over binary fields. We choose FPGAs as our target devices not only because they can serve as fast prototyping platforms, but also because of their reconfigurability. The reconfigurability is crucial in this application because of an early stage of development of the field, the lack of standards, and the constant progress in the cryptanalysis of pairing based cryptosystems, which may affect key sizes, and thus the sizes of all operands, already in the near future.

Our implementation is based on the algorithms presented in [10] and [32]. Even though the binary elliptic curves require relatively long operands (compared to the cubic elliptic and binary hyperelliptic cases), the arithmetic operations in the algorithms we apply are simple and easy to parallelize. We introduce a compact design for the extension field multiplier by sharing an XOR array in case that one of the two operands is the same for the underlying field multiplications. Our controller is realized using hardwired logic. We derive the method to simplify the datapath for the final exponentiation. We also consider the optimal choices of parameters such as

digit sizes of multipliers to further optimize the design. Consequently, our pairing accelerator can run 10-to-20 times faster than the ones published in [33,35,65], at the same level of security strength with the lower product of latency by area.

1.2.5 Reconfigurable Computing Approach for Tate Pairing Cryptosystems

We choose a Reconfigurable computer, SGI RASC RC-100 provided by Silicon Graphics, Inc., as the end-to-end system for our pairing computations over 8 selected binary fields ranging from $\mathbb{F}_{2^{239}}$ to $\mathbb{F}_{2^{557}}$. Performance and cost for our reconfigurable approach of pairing are shown. This is the first published complete investigation of pairing on binary elliptic curves in a real-time system.

1.3 Outline

In Chapter 2 we introduce the mathematical foundations of finite fields, elliptic curves and pairings, restricting ourselves to the material necessary to understand this work. We start with the finite field theory, covering definitions, constructing method, representations via different bases and composite field structure. With these definitions and properties, we are capable of deriving new architectures for field arithmetic as well as designing the circuits efficiently. Next we briefly introduce the basic concepts from the theory of elliptic curves. Furthermore we explain some of the mathematics behind pairings including the concepts of bilinear pairings, divisor and Tate pairing over finite fields.

Chapter 3 focuses on the efficient design for field arithmetic including adder, squarer, multiplier and inverter. We introduce the standard reduction method exploiting the property of trinomials or pentanomials by which the circuit complexity

of squarers and multipliers can be simplified. We show the derivation of the MSD serial architecture for PB multipliers. We introduce the technique for simplifying the circuit complexity as well as shortening the critical path for NB multipliers. We also propose a new hybrid multiplier for composite field and address the issue of basis conversion for this multiplier. We perform comparisons in terms of FPGA implementations between the conventional digit-serial PB multiplier and the hybrid multiplier, both of which are constructed via a trinomial or a pentanomial.

In Chapter 4, we first compare different top algorithms for computing the scalar multiplication in terms of performance, cost and ease of hardware implementation. We choose López-Dahab algorithm for our implementations via a single FPGA device. The design issues such as top architecture, parallel computation, sharing resource, and optimal choice of multipliers are covered. Finally the comparisons with other works are performed.

In Chapter 5, we investigate implementations of ECC on a reconfigurable computer, SRC-6 provided by SRC Computer, Inc. The architecture and the programming mode of this reconfigurable system are introduced. Different partitioning schemes between software and hardware according to the hierarchy of operations involved in ECC are presented and realized. Analysis of the trade-off between SRC-6 programming versus the traditional hardware design model is performed. Furthermore, we discuss the issues of developing ECC hardware library for a reconfigurable computer.

In Chapter 6, we first present two algorithms for Tate pairing cryptosytems on binary elliptic curves. The issues of choosing curves and fields are addressed. Our FPGA-based implementations of pairing are described in details. Comparisons with other researchers for various pairing scheme, in terms of timing and area, are performed.

Chapter 7 focuses on the issues of porting pairing to a reconfigurable computer, SGI Altix-4700 supported by Silicon Graphics, Inc. The experimental results of our reconfigurable computing approach of pairing over the 8 selected binary fields are demonstrated. We also compare pairing and scalar multiplication in terms of performance and cost for both software and hardware. Finally, we end this dissertation with the conclusions of our work in Chapter 8.

Chapter 2: Mathematical Background

This introductory chapter contains a survey of some basic mathematical concepts related to finite fields, elliptic curve cryptography, and pairing based cryptography that will be employed throughout this dissertation. We restrict our attention to material that is relevant for this work. More details can be found in [4, 19, 23, 24]

2.1 Finite Fields

In this section, we briefly introduce several definitions and properties of finite fields that are fundamental for efficient hardware implementations of finite field arithmetic.

2.1.1 Introduction

We first provide the definitions of group, ring and field.

Definition 2.1.1. [24] A group is a set $\mathbb{G}$ together with a binary operation $*$ on $\mathbb{G}$, such that the following three properties hold:

1. $*$ is associative; that is, for any $a, b, c \in \mathbb{G}$, $a * (b * c) = (a * b) * c$.
2. There is an identity (or unity) element e in $\mathbb{G}$ such that for all $a \in G$, $a * e = e * a = a$.
3. For each $a \in \mathbb{G}$, there exists an inverse element $a^{-1} \in \mathbb{G}$ such that $a * a^{-1} = a^{-1} * a = e$. If the group also satisfies
4. For all $a, b \in \mathbb{G}$, $a * b = b * a$

then the group is called abelian (or commmunicative)

Definition 2.1.2. [24] A subset $\mathbb{H}$ of the group $\mathbb{G}$ is a subgroup of $\mathbb{G}$ if $\mathbb{H}$ is itself a group with respect to the operation of $\mathbb{G}$.

Theorem 2.1.1. [23] If $\mathbb{G}$ is a finite group and $\mathbb{H}$ is a subgroup of $\mathbb{G}$, then the number of elements in $\mathbb{H}$ divides the number of elements in $\mathbb{G}$.

Definition 2.1.3. [24] A ring $(\mathbb{R}, +, \cdot)$ is a set $\mathbb{R}$, together with two binary operations, denoted by $+$ and $\cdot$, such that:

1. $\mathbb{R}$ is an abelian group with respect to $+$.
2. $\cdot$ is associative, that is , $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in \mathbb{R}$
3. The distributive laws hold; that is, for all $a, b, c \in \mathbb{R}$ we have $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$.

A subset $\mathbb{S}$ of a ring $\mathbb{R}$ is called a subring of $\mathbb{R}$ provided $\mathbb{S}$ is closed under $+$ and $\cdot$ and forms a ring under these operations

Definition 2.1.4. [24] A field $(\mathbb{F}, +, \cdot)$ is a ring in which the nonzero elements form an abelian group under $\cdot$. If the set $\mathbb{F}$ is finite, then the field is said to be finite.

Definition 2.1.5. [24] A subset $\mathbb{S}$ of a field $\mathbb{F}$ is called a subfield of $\mathbb{F}$ provided $\mathbb{S}$ is closed under $+$ and $\cdot$ and forms a field under these operations

Definition 2.1.6. [24] If $\mathbb{R}$ is an arbitrary ring and there exists a positive integer n such that $nr = r + r + \ldots + r$ *(n times)* $= 0$, where $r \in \mathbb{R}$ and '0' denotes the identity of $\mathbb{R}$ under addition, the least such positive integer n is called the characteristic of $\mathbb{R}$ and $\mathbb{R}$ is said to have (positive) characteristic n. If no such positive integer n exists, R is said to have characteristic 0.

Theorem 2.1.2. [24] Let $\mathbb{F}$ be a finite field. Then $\mathbb{F}$ has p^m elements, where the prime p is the characteristic of $\mathbb{F}$ and m is the degree of $\mathbb{F}$ over its prime subfield.

For any prime power p^m, there is essentially only one finite field of order p^m. In Theorem 2.1.2, if $m = 1$, then $\mathbb{F}$ is called a prime field. If $p = 2$, then $\mathbb{F}$ is called a binary field or a characteristic-two finite field. In this field substraction is equivalent to addition. For this work, we focus on binary fields.

Definition 2.1.7. Let $\mathbb{F}$ be a field. $\overline{\mathbb{F}}$ is the algebraic closure of $\mathbb{F}$ if every polynomial with coefficient in $\mathbb{F}$ factors completely into linear factors.

Two important tools regarding finite field theory are the trace function and the norm function. The trace function of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ is

$$Tr_{\mathbb{F}_{q^m}|\mathbb{F}_q}(\alpha) = \sum_{i=0}^{m-1} \alpha^{q^i} \tag{2.1}$$

and the norm function is

$$N_{\mathbb{F}_{q^m}|\mathbb{F}_q}(\alpha) = \prod_{i=0}^{m-1} \alpha^{q^i} \tag{2.2}$$

2.1.2 Representations of Elements of Binary Fields

The most attractive advantages of using binary fields (or finite fields of characteristic two) are their carry-free arithmetic and the availability of different equivalent representations of the field, which make them superior from the point of view of hardware implementations. The binary field $\mathbb{F}_{2^m}$ can be viewed as a vector space over $\mathbb{F}_2$ with dimension m. All field elements can be represented uniquely as binary vectors of dimension m, relative to a given basis $(\alpha_1, \alpha_2, ..., \alpha_m)$, i.e., in the form $a = \sum_{i=1}^{m} a_i \alpha_i$,

where $a_i \in \{0,1\}$. The following theorem is often referred to judge whether a set, namely $\{\beta_1, \beta_2, ..., \beta_m\}$, can be chosen as a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ (for binary fields, $q = 2^n$).

Theorem 2.1.3. [44] The set of elements $B = \{\beta_1, \beta_2, ..., \beta_m\}$ is a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ if and only if the matrix A is nonsingular, i.e., $\det A \neq 0$, where

$$A = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 & \dots & \beta_m \\ \beta_1^q & \beta_2^q & \beta_2^q & \dots & \beta_m^q \\ \beta_1^{q^2} & \beta_2^{q^2} & \beta_3^{q^2} & \dots & \beta_m^{q^2} \\ \dots & \dots & \dots & \dots & \dots \\ \beta_1^{q^{m-1}} & \beta_2^{q^{m-1}} & \beta_3^{q^m} & \dots & \beta_m^{q^{m-1}} \end{bmatrix} \tag{2.3}$$

There is a variety of ways to represent elements in a binary finite field, depending on the choice of a basis for representation. Polynomial basis (PB) and normal basis (NB) are commonly used and supported by the NIST [17] and other standards. The definitions of these two bases are given as below:

Definition 2.1.8. [24] Let $\alpha \in \mathbb{F}_{q^m}$ be the root of an irreducible polynomial of degree m over $\mathbb{F}_q$. The polynomial basis of $\mathbb{F}_{q^m}$ is then $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$.

For efficient realizations of binary field arithmetic, low Hamming weight irreducible polynomials are generally chosen to construct the fields, since there always exist irreducible trinomials or pentanomials of the forms: $f_t(x) = x^m + x^k + 1$ or $f_p(x) = x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ for every $m \leq 10,000$ [25]. Alternatively, other irreducible polynomials, such as all one polynomials (AOP) [14,15], can be exploited when using redundant basis representation.

Definition 2.1.9. [24] Let $\mathbb{F}_{q^m}$ be an extension of $\mathbb{F}_q$ and let $\beta \in \mathbb{F}_{q^m}$. Then the elements $\beta, \beta^q, \dots, \beta^{q^{m-1}}$ are called the conjugates of β with respect to $\mathbb{F}_q$.

Definition 2.1.10. [24] A basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ of the form $\{\beta, \beta^q, ..., \beta^{q^{m-1}}\}$, consisting of a suitable element $\beta \in \mathbb{F}_{q^m}$ and its conjugates with respect to $\mathbb{F}_q$, is called a normal basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$.

For any finite field $\mathbb{F}_q$ and any extension $\mathbb{F}_{q^m}$ of $\mathbb{F}_q$, there exists a normal basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. The proof can be found in [24]. In favor of hardware design of binary field arithmetic using normal basis representation, square is simple and equivalent to a cyclic shift. However the circuit complexity of a multiplier is heavily dependent on the choice of the normal basis used. For a few fields, there exist optimal normal bases, by which the multiplication has the lowest computational complexity.

2.1.3 Binary Composite Field Structure

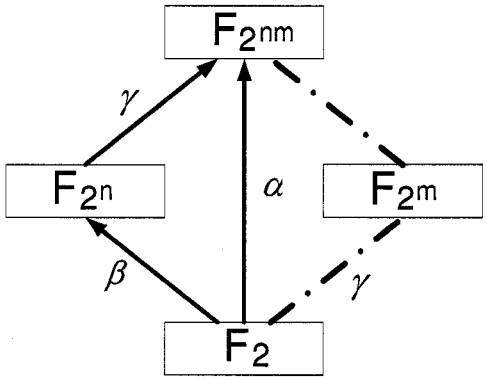


Figure 2.1: The binary composite field structure

We first define some notation used in Fig. 2.1. In our work, we restrict the choice of n and m to values for which $gcd(n, m) = 1$. Let α denote the generator of the basis $B_s = \{\alpha_0, \alpha_1, ..., \alpha_{nm-1}\}$ of $\mathbb{F}_{2^{nm}}$ over $\mathbb{F}_2$. Let β denote the generator of the basis $B_g = \{\beta_0, \beta_1, ..., \beta_{n-1}\}$ of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. And let γ denote the generator of the basis $B_t = \{\gamma_0, \gamma_1, ..., \gamma_{m-1}\}$ of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Then, we have the following theorem.

Theorem 2.1.4. If $gcd(n, m) = 1$, then B_t is also a basis of $\mathbb{F}_{2^{nm}}$ with respect to $\mathbb{F}_{2^n}$.

Proof. To prove that B_t is a basis of $\mathbb{F}_{2^{nm}}$ over $\mathbb{F}_{2^n}$, it suffices to check that the following matrix is non-singular (See Theorem 2.1.3).

$$A_1 = \begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_{m-1} \\ \gamma_0^{2^n} & \gamma_1^{2^n} & \gamma_2^{2^n} & \cdots & \gamma_{m-1}^{2^n} \\ \gamma_0^{2^{2n}} & \gamma_1^{2^{2n}} & \gamma_2^{2^{2n}} & \cdots & \gamma_{m-1}^{2^{2n}} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \gamma_0^{2^{(m-1)n}} & \gamma_1^{2^{(m-1)n}} & \gamma_2^{2^{(m-1)n}} & \cdots & \gamma_{m-1}^{2^{(m-1)n}} \end{bmatrix} \tag{2.4}$$

Since $gcd(n, m) = 1$, it is not difficult to prove that the matrix A_1 can be obtained by permutation of the rows of the following matrix A_0.

$$A_0 = \begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \cdots & \gamma_{m-1} \\ \gamma_0^{2} & \gamma_1^{2} & \gamma_2^{2} & \cdots & \gamma_{m-1}^{2} \\ \gamma_0^{2^2} & \gamma_1^{2^2} & \gamma_2^{2^2} & \cdots & \gamma_{m-1}^{2^2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \gamma_0^{2^{m-1}} & \gamma_1^{2^{m-1}} & \gamma_2^{2^{m-1}} & \cdots & \gamma_{m-1}^{2^{m-1}} \end{bmatrix} \tag{2.5}$$

As B_t is a basis of $\mathbb{F}_{2^m}$ with respect to $\mathbb{F}_2$, A_0 must be non-singular. So is A_1. □

Theorem 2.1.4 can be exploited to derive several types of hybrid multipliers considering that B_s, B_t and B_g can be any bases such as polynomial bases and normal bases. We will discuss the constructing method in the next chapter.

2.2 Elliptic Curve Cryptosystems

In this section, we briefly introduce some basic concepts and collect various results for elliptic curve cryptosystmes. This survey is far from a comprehensive treatment but is sufficient for this work.

2.2.1 Introduction

Definition 2.2.1. [26] An elliptic curve $\mathbf{E}$ over a field K is defined by a Weierstrass equation

$$\mathbf{E}: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.6}$$

where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$, where Δ is the discriminant of $\mathbf{E}$ and is defined as follows:

$$\Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \tag{2.7a}$$

$$d_2 = a_1^2 + 4a_2 \tag{2.7b}$$

$$d_4 = 2a_4 + a_1a_3 \tag{2.7c}$$

$$d_6 = a_3^2 + 4a_6 \tag{2.7d}$$

$$d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \tag{2.7e}$$

If L is any extension of K, then the set of *L-rational points* on $\mathbf{E}$ is

$\mathbf{E}(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \infty$ where ∞ is the point at infinity.

For this work, we choose binary fields as the underlying fields of elliptic curves, i.e., $K = \mathbb{F}_{2^m}$, so that the original Weierstrass equations can be simplified and there are two cases to consider [26]. If $a_1 \neq 0$, then the change of variables transforms $\mathbf{E}$ to the curve

$$y^2 + xy = x^3 + ax^2 + b \tag{2.8}$$

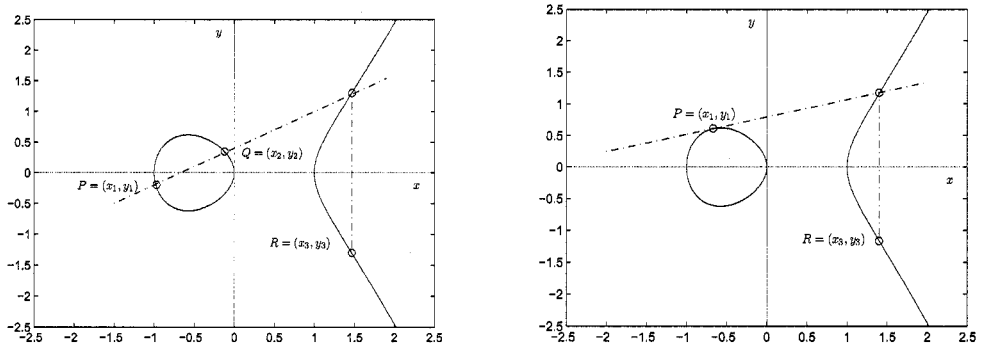
where $a, b \in K$. Such a curve is said to be *non-supersingular*. If $a_1 = 0$, then by the

change of variables we can transforms $\mathbf{E}$ to the curve

$$y^2 + cy = x^3 + ax + b \tag{2.9}$$

where $a, b \in K$. Such a curve is said to be *supersingluar*. The non-supersingular curves are usually chosen to underly elliptic curve cryptosystems. On the other hand, supersingular curves can be applied in Tate-pairing cryptosystems.

2.2.2 Group Law



(a) Addition: $P + Q = R$.

(b) Doubling: $P + P = R$.

Figure 2.2: Geometric addition and doubling of elliptic curve points

The points on an elliptic curves with ∞ form an abelian group under a certain addition following the chord-and-tangent rule, i.e., adding two points in $\mathbf{E}(K)$ to give a third point in $\mathbf{E}(K)$. This group is used to construct the elliptic curve cryptosytems.

The addition is explained geometrically. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve $\mathbf{E}$. The straight line joining P and Q must intersect the curve at one further point (see Fig. 2.2(a)) as the line intersects a cubic curve [18]. If we reflect the point with respect to the x-axis we obtain the sum R of P and Q.

The double R of P can be defined as follows. Take the tangent to the curve at P. Such a line must intersect $\mathbf{E}(K)$ in exact one other point. Then R is the reflection of this point with respect to the x-axis (see Fig. 2.2(b)). Next, we describe the group law for non-supersingular curve $\mathbf{E}(\mathbb{F}_{2^m})$: $y^2 + xy = x^3 + ax^2 + b$ [26].

1. *Identity.* $P + \infty = \infty + P = P$ for all $P \in \mathbf{E}(\mathbb{F}_{2^m})$.

2. *Negatives.* If $P = (x, y) \in \mathbf{E}(\mathbb{F}_{2^m})$, then $(x, y) + (x, x + y) = \infty$. The point $(x, x + y)$ is denoted by $-P$ and is called the negative of P. Also, $-\infty = \infty$.

3. *Point addition.* Let $P = (x_1, y_1) \in \mathbf{E}(\mathbb{F}_{2^m})$ and $Q = (x_2, y_2) \in \mathbf{E}(\mathbb{F}_{2^m})$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \tag{2.10a}$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \tag{2.10b}$$

with $\lambda = (y_1 + y_2)/(x_1 + x_2)$.

4. *Point doubling.* Let $P = (x_1, y_1) \in \mathbf{E}(\mathbb{F}_{2^m})$, where $P \neq -P$. Then $2P = (x_3, y_3)$, where

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \tag{2.11a}$$

$$y_3 = x_1^2 + \lambda x_3 + x_3 \tag{2.11b}$$

with $\lambda = x_1 + y_1/x_1$.

5. *Scalar multiplication.* Let $P = (x_1, y_1) \in \mathbf{E}(\mathbb{F}_{2^m})$ and $k \in \mathbb{Z}$. Then

$$[k]P = \underbrace{P + P + \ldots + P}_{k \text{ times}} \tag{2.12}$$

The number of points in $\mathbf{E}(\mathbb{F}_{2^m})$, denoted $\#\mathbf{E}(\mathbb{F}_{2^m})$, is called the order of $\mathbf{E}$ over $\mathbb{F}_{2^m}$. Theorem 2.2.1 provides the bounds for $\#\mathbf{E}(\mathbb{F}_{2^m})$ [26].

Theorem 2.2.1. *(Hasse)* Let $\mathbf{E}$ be an elliptic curve defined over $\mathbb{F}_{2^m}$. Then

$$2^m + 1 - 2^{\frac{m+2}{2}} \leq \#\mathbf{E}(\mathbb{F}_{2^m}) \leq 2^m + 1 + 2^{\frac{m+2}{2}}$$

Those curves whose order has a large prime factor are suitable to be chosen to construct elliptic curve cryptosystems. The security is based on the intractability of discrete logarithm problems of elliptic curves, i.e., it is difficult to get the knowledge of k even though P and kP are given.

2.2.3 Koblitz Curves

Definition 2.2.2. [26] Koblitz curves, also known as anomalous binary curves, are elliptic curves defined as follows:

$$\mathbf{E}_0 : y^2 + xy = x^3 + 1$$

$$\mathbf{E}_1 : y^2 + xy = x^3 + x^2 + 1$$

Koblitz curves are recommended by the NIST standards. The benefit of using them comes from the fact that a scalar multiplication can be performed efficiently using the Frobenius map $\tau : \mathbf{E}_a(\mathbb{F}_{2^m}) \rightarrow \mathbf{E}_a(\mathbb{F}_{2^m})$ $(a \in \{0, 1\})$ which is defined as follows [21]

$$\tau(\infty) = \infty,\ \tau(x, y) = (x^2, y^2),$$

Compared with point addition or doubling, the evaluation of Frobenius map is much simpler, in particular, when normal basis is chosen to represent the field element. For each point P on a Koblitz curve $\mathbf{E}_a(\mathbb{F}_{2^m})$, we have $(\tau^2 + 2)P = \mu\tau(P)$ where

$\mu = (-1)^{1-a}$ [60]. Hence the Frobenius map can be regarded as a complex number τ satisfying

$$\tau^2 + 2 = \mu\tau \tag{2.13}$$

Obtaining an efficient realization of Koblitz curve cryptosystems is equivalent to finding a good expression for an integer k of the form $k = \sum_{i=0}^{l} \mu_i \tau^i$.

2.3 Pairing Based Cryptosystems

The purpose of this section is to introduce some definitions and properties behind pairings, including bilinearity, divisors, and Tate pairing, most of which come from [19, 45, 48].

2.3.1 Introduction

The first use of Weil and Tate pairings was confined to attack elliptic curve cryptosystems. However, pairing can be also exploited to construct several protocols, such as one-round tripartite key agreement, identity based encryptions, and a short signature, which can not be constructed in any other known way. Pairing in elliptic curve cryptography is a function mapping a pair of elliptic curve points to an element of the multiplicative group of a finite field.

2.3.2 Bilinear Pairings

Let n be a prime number. Let G_1 and G_2 be abelian groups of order n written in additive notation with identity ∞, and let G_3 be a multiplicatively-written group of order n with identity 1.

Definition 2.3.1. [48] A bilinear pairing is a function

$$e : G_1 \times G_2 \rightarrow G_3 \tag{2.14}$$

that satisfies the following conditions:

1. *Bilinearity:* For all $P, P' \in G_1$ and all $Q, Q' \in G_2$ we have $e(P + P', Q) = e(P, Q)e(P', Q)$ and $e(P, Q + Q') = e(P, Q)e(P, Q')$.

2. *Non-degeneracy:*

   - For all $P \in G_1$, with $P \neq 0$, there is some $Q \in G_2$ such that $e(P, Q) \neq 1$
   - For all $Q \in G_2$, with $Q \neq 0$, there is some $P \in G_1$ such that $e(P, Q) \neq 1$

3. *computability:* e can be efficiently computed.

The Weiling and Tate pairings on elliptic curves over finite fields are two such examples. We focus on the Tate pairing in this work. Some consequences of bilinearity are given below.

Lemma 2.3.1. [19] Let e be a bilinear pairing as above. Let $P \in G_1$ and $Q \in G_2$. Then

1. $e(P, 0) = e(0, Q) = 1$.

2. $e(-P, Q) = e(P, Q)^{-1} = e(P, -Q)$.

3. $e([j]P, Q) = e(P, Q)^j = e(P, [j]Q)$ for all $j \in \mathbb{Z}$.

2.3.3 Divisors

Divisors, the concept of which comes from algebraic geometry, are useful devices for keeping track of the zeros and poles of a rational function. Most of the following consequences related to divisor theories are from in [19, 45].

Definition 2.3.2. [19] Let $\mathbf{C}$ be a curve over K, usually an elliptic curve. A *divisor* D is a formal sum of $\overline{K}$ points, where $\overline{K}$ denotes the algebraic closure of K

$$D = \sum_{P \in \mathbf{C}} n_P(P),$$

where $n_P \in \mathbb{Z}$ and all but finitely many n_P are zero.

The divisor with all $n_P = 0$ is denoted 0. The set of divisors on $\mathbf{C}$ is denoted $Div_K(\mathbf{C})$ and has a natural group structure of addition. The support of a divisor of D is the set of all points such that $n_P \neq 0$. The degree of a divisor D is $deg(D) = \sum_P n_P$. Let $\mathbb{D}^0$ be the set of all divisors of degree 0. Then $\mathbb{D}^0$ is a subgroup of $Div_K(\mathbf{C})$ [19].

If f is a non-zero function on $\mathbf{C}$, then $ord_P(f)$ counts the multiplicity of f at P. $ord_P(f)$ is positive when $f(P) = 0$ and is negative if f has a pole at P. The divisor of a non-zero function f, written (f), is the divisor $\sum_{P \in \mathbf{C}(\overline{K})} ord_P(f)(P)$. It follows that $(fg) = (f) + (g)$ and $(f/g) = (f) - (g)$ [19].

Two divisors D and D' are said to be equivalent (written $D \sim D'$) if $D' = D + (f)$ for some rational function f.

Definition 2.3.3. [80] A divisor $D \in \mathbb{D}^0$ is called principal divisor if $D = div(f)$ for some rational function $f \in \mathbf{C}(\overline{K})$. The set of all principle divisors is denoted $\mathbb{P}$. The quotient group $\mathbb{J} = \mathbb{D}^0/\mathbb{P}$ is called the Jacobian of the curve $\mathbf{C}$.

We can use the terminology of divisors to describe the group law on an elliptic curve (this new expression is from [19]). In Fig 2.2, the chord line between P and Q or the tangent line at P, $P = Q$, has the equation $l(x, y) = 0$. Let $S = (x_S, y_S)$ denotes the hit point on the elliptic curve. Then, the function $l(x, y)$ has divisor $(l) = (P) + (Q) + (S) - 3(\infty)$. The vertical line $v(x) = x - x_s$ passes through the points S and $R = P + Q$ by definition. The divisor on E of $v(x)$ can be written

$(v) = (S) + (R) - 2(\infty)$. Therefore the equation $R = P + Q$ is the same as the divisor equality $(R) - (\infty) = (P) - (\infty) + (Q) - (\infty) - (l/v)$ and mapping of P to the divisor class of $(P) - (\infty)$ is a group homomorphism.

Theorem 2.3.1. [19] Let $\mathbf{E}$ be an elliptic curve over a field K. Let

$$D = \sum_P n_P(P)$$

be a degree zero divisor on E. Then $D \sim 0$ (i.e., there is a function f such that $D = (f)$) if and only if $\sum_P [n_p]P = \infty$ on $\mathbf{E}$.

Let f be a function and let $D = \sum_P n_P(P)$ be a divisor of degree zero such that the support of D is disjoint to the support of (f). Define

$$f(D) = \prod_P f(P)^{n_P}$$

2.3.4 Tate Pairing

Definitions

Let $\mathbf{E}$ be an elliptic curve over a field K_0. Let n be a positive integer which is coprime to the characteristic of the field K_0. The set of nth roots of unity is defined to be $\mu_n = \{u \in \overline{K_0^*} : u^n = 1\}$. Let the field $K = K_0(\mu_n)$ to be the extension of K_0 generated by the nth roots of unity. Define

$$\mathbf{E}(K)[n] = \{P \in \mathbf{E}(K) : [n]P = \infty\}$$

$$n\mathbf{E}(K) = \{[n]P : P \in \mathbf{E}(K)\}$$

Then $\mathbf{E}(K)[n]$ is a group of exponent n and $n\mathbf{E}(K)$ is a subgroup of $\mathbf{E}(K)$ and the quotient group $\mathbf{E}(K)/n\mathbf{E}(K)$ is a group of exponent n. Define

$$(K^*)^n = \{u^n : u \in K^*\}$$

Then the groups $K^*/(K^*)^n$ and μ_n are isomorphic.

Definition 2.3.4. [19] Let $P \in \mathbf{E}(K)[n]$ and let $Q \in \mathbf{E}(K)$ representing an equivalent class in $\mathbf{E}(K)/n\mathbf{E}(K)$. Since $[n]P = \infty$, it follows that there exists a function f such that $(f) = n(P) - n(\infty)$. Let D be any degree zero divisor equivalent to $(Q) - (\infty)$ or $(Q+S) - (S)$ such that D is defined over K and the support of D is disjoint from the support of (f), i.e., $f(D) \neq 0$ and so $f(D) \in K^*$. The Tate pairing of P and Q is defined to be

$$\langle P, Q \rangle_n = f(D)$$

interpreted as an element of $K^*/(K^*)^n$.

Properties

Theorem 2.3.2. [19] Let $\mathbf{E}$ be an elliptic curve over K_0 and let n be coprime to the characteristic of K_0. Let $K = K_0(\mu_n)$. The Tate pairing satisfies:

1. (Bilinearity) For all $P, P_1, P_2 \in \mathbf{E}(K)[n]$ and $Q, Q_1, Q_2 \in \mathbf{E}(K)/n\mathbf{E}(K)$,

$$\langle P_1 + P_2, Q \rangle_n = \langle P_1, Q \rangle_n \langle P_2, Q \rangle_n$$

and

$$\langle P, Q_1 + Q_2 \rangle_n = \langle P, Q_1 \rangle_n \langle P, Q_2 \rangle_n$$

2. (Non-degeneracy) Suppose K is a finite field. For all $P \in \mathbf{E}(K)[n]$, $P \neq \infty$, there is some $Q \in \mathbf{E}(K)/n\mathbf{E}(K)$ such that $\langle P, Q \rangle_n \neq 1$. Similarly, for all $Q \in$

$\mathbf{E}(K)/n\mathbf{E}(K)$ with $Q \notin n\mathbf{E}(K)$ there is some $P \in \mathbf{E}(K)$ such that $\langle P, Q\rangle \neq 1$.

3. (Galois invariance) If $\sigma \in Gal(\overline{K}/K_0)$, then $\langle \sigma(P), \sigma(Q)\rangle_n = \sigma(\langle P, Q\rangle_n)$ where $Gal(\overline{K}/K_0)$ denotes the set of automorphism σ of K such that $\sigma(x) = x$ for every $x \in K_0$.

Tate Pairing over Finite Fields

Let $K_0 = \mathbb{F}_q$ be a finite field. Let $\mathbf{E}$ be an elliptic curve defined over K_0 and let n be an integer coprime to q which divides $\#\mathbf{E}(K_0)$. The field $K = K_0(\mu_0)$ is some finite extension $\mathbb{F}_{q^k}$ where k is called the embedded degree or security multiplier and for simplicity it is the smallest integer such that n divides $(q^k - 1)$ [19].

The value of Tate pairing is an equivalence class in $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^n$ and for practical reasons this value can be raised to the power $(q^k-1)/n$ so that a unique representation of this class can be obtained. Then the bilinear pairing can be defined as:

$$e(P, Q) = \langle P, Q\rangle_n^{(q^k-1)/n}$$

which maps into the group $\mu_n \subset \mathbb{F}^*_{q^k}$ rather than the group $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^n$ [19].

Miller's Algorithm

The crucial ingredient of the algorithm is to find a function such that $(f) = n(P) - n(\infty)$. Miller proposed an construction of such a function using the double-and-add method. Let f_i denote the function such that $(f_i) = i(P) - ([i]P) - (i-1)(\infty)$. Such a function is uniquely determined up to a constant multiple. We need to compute f_n.

Lemma 2.3.2. [19] The functions f_i can be chosen to satisfy the following conditions.

1. $f_1 = 1$.

2. Let l and v be the straight lines used in the computation of $[i]P+[j]P = [i+j]P$. Then $f_{i+j} = f_i f_j \frac{l}{v}$.

To evaluate $f_n(D)$, one can choose $D = (Q + S) - (S)$. Miller's algorithm for computing Tate pairing is provided as follows.

Algorithm 1 Miller's Algorithm

Require: $P, Q \in \mathbf{E}(K)$, where P has order n.

Ensure: $\langle P, Q \rangle_n$.

1: Choose a suitable point $S \in \mathbf{E}(K)$.

2: $Q' \leftarrow Q + S$, $T \leftarrow P$, $m \leftarrow \lfloor \log_2(n) \rfloor - 1$, $f \leftarrow 1$.

3: **while** $m \geq 0$ **do**

4: Calculate lines l and v for doubling T.

5: $T \leftarrow [2]T$, $f \leftarrow f^2 \frac{l(Q')v(S)}{v(Q')l(S)}$.

6: **if** the mth bit of n is one **then**

7: Calculate lines l and v for addition of T and P.

8: $T \leftarrow T + P$, $f \leftarrow f \frac{l(Q')v(S)}{v(Q')l(S)}$

9: **end if**

10: $m \leftarrow m - 1$

11: **end while**

12: Return f.

Chapter 3: Hardware Design for Finite Field Arithmetic

Finite field arithmetic has been attracting an increased attention of researchers due to its extensive applications in cryptographic algorithms and error correction codes adopted in the internet and wireless communication systems. It is well known that the arithmetic in $\mathbb{F}_{2^m}$ is easier to implement in hardware than the arithmetic in finite fields of characteristic greater than 2, since addition in $\mathbb{F}_{2^m}$ is equivalent to bit-wise XOR. In this work, we concentrate on the design of binary field arithmetic, particularly for the multiplier, considering that the computations involved in elliptic curve based cryptosystems contain intensive multiplications. Our contributions to this topic can be summarized as follows:

1. Derived the standard polynomial basis (PB) digit-serial multiplier using left-to-right algorithm and compare it with the one based on right-to-left algorithm in terms of timing and area.

2. Compared the standard (PB) digit-serial multipliers with the hybrid multipliers for several special composite fields constructed via low Hamming weight irreducible polynomials in terms of FPGAs implementations.

3. Derived a new hybrid multiplier in which a normal basis bit-serial architecture is used at the top level and a special trinomial basis bit-parallel architecture is used at the ground level. Using our design, not only circuit complexity can be decreased but also basis conversion can be performed simply.

4. Basis conversion for the new hybrid multiplier is explained in detail.

3.1 Squarer

Squaring can certainly be performed using a multiplier. However when it is necessary for general exponentiation as well as inversion of a field element, squaring should be completed in a shortest possible time. In case normal basis is chosen to represent the field element, squarer is free in terms of both timing and area as it is equivalent to cyclic shift [3] (see Fig. 3.1). Proof is given as follows. Suppose that $\{\beta, \beta^2, \beta^{2^2}, ..., \beta^{2^{m-1}}\}$ is a normal basis of $\mathbb{F}_{2^m}$ and $a = \sum_{i=0}^{m-1} a_i \beta^{2^i} \in \mathbb{F}_{2^m}$ where $a_i \in \mathbb{F}_2$. Then by $\beta^{2^m} = \beta$, we have

$$a^2 = a_0\beta^2 + a_1\beta^{2^2} + \ldots + a_{m-1}\beta^{2^m} = a_{m-1}\beta + a_0\beta^2 + a_1\beta^{2^2} + \ldots + a_{m-2}\beta^{2^{m-1}} \quad (3.1)$$

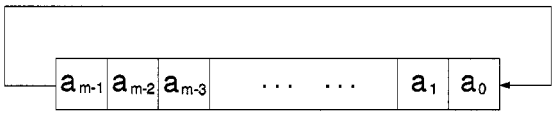


Figure 3.1: Normal basis squarer in $\mathbb{F}_{2^m}$

Squaring a field element in $\mathbb{F}_{2^m}$ represented via a polynomial basis, $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$, is ruled by the following equation.

$$a^2 = (\sum_{i=0}^{m-1} a_i \alpha^i)^2 = \sum_{i=0}^{m-1} a_i \alpha^{2i} \quad (3.2)$$

If the generator of the polynomial basis is the root of a low Hamming weight irreducible polynomial such as a trinomial or a pentanomial, the reduction becomes simple, so the circuit has a low complexity. For instance, suppose that $\mathbb{F}_{2^4}$ is constructed via the trinomial $f_4(x) = x^4 + x + 1$ and let α be the root of $f_4(x) = 0$. Let

$a = \sum_{i=0}^{m-1} a_i \alpha^i$ where $a_i \in \mathbb{F}_2$. By replacing $\alpha^4 = \alpha + 1$, we have

$$a^2 = a_3\alpha^6 + a_2\alpha^4 + a_1\alpha^2 + a_0 = a_3\alpha^2(\alpha + 1) + a_2(\alpha + 1) + a_1\alpha^2 + a_0$$

$$= a_3\alpha^3 + (a_3 + a_1)\alpha^2 + a_2\alpha + (a_0 + a_2) \tag{3.3}$$

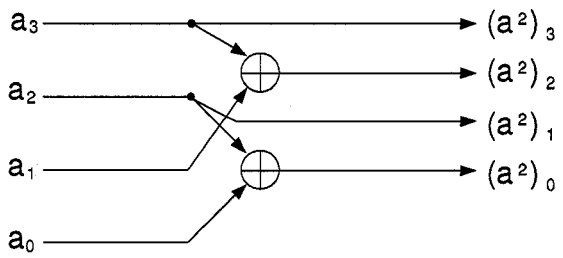


Figure 3.2: Polynomial basis squarer in $\mathbb{F}_{2^4}$ constructed via $f_4(x) = x^4 + x + 1$

Hence, the squarer in $\mathbb{F}_{2^4}$ can be realized via several XOR gates shown in Fig. 3.2.

3.2 Multiplier

Finite field multiplier always plays a central role determining the performance of hardware accelerators of cryptographic applications because these computations usually involve intensive field multiplications. As a result, it is imperative to design the multipliers with high efficiency. Bit parallel multipliers can complete one multiplication in one clock cycle to achieve high operation speed. However they also have maximum circuit complexity, which make them unsuitable for those multiplications with large operand sizes. Bit serial multipliers, generating one bit of the product each clock cycle, are efficient in terms of area, however the operation speed is sacrificed. Digit serial multipliers, allowing the tradeoff between latency and cost, are the most suitable for the cryptographic applications with large operand sizes.

Multiplier is essentially a device to compute the coefficients of the product of two elements in the field. The structures may be significantly different in case that different bases are selected to represent the field elements, e.g., polynomial bases (PB) and normal bases (NB). However, the shift-and-add structure is commonly shared among these different types of multipliers. In the following sections we start with the efficient designs for PB and NB multipliers. Then, we discuss the construction methods of hybrid multipliers for the composite fields $\mathbb{F}_{2^{nm}}$. We derive a novel hybrid multiplier which is not only efficient in terms of low latency and low circuit complexity, but also is convenient to perform basis conversion.

3.2.1 Polynomial Basis Multiplier

Efficient reduction method is crucial to improve the design of a PB multiplier. Hence irreducible trinomials or pentanomials are frequently used to construct the binary fields $\mathbb{F}_{2^m}$. The reduction technique, introduced in previous section for PB squarer, can be applied to derive PB multipliers as well. In the following subsections, we first introduce two algorithms, left-to-right and right-to-left, to derive the bit-serial and digit-serial architectures. Then, we compare these two algorithms in terms of latency and area. Finally Mastrovito method [42] is introduced for the bit parallel architecture.

Bit-Serial Structure

Two algorithms, right-to-left and left-to-right, are generally used to derive the least significant or most significant bit serial multipliers (LSB or MSB).

In Algorithms 2 and 3, $f(x)$ is generally chosen as a trinomial or a pentanomial. For better understanding, we use one example for both multipliers in $\mathbb{F}_{2^9}$ constructed

Algorithm 2 Bit-serial right-to-left multiplication

Require: $a = (a_{m-1}, \ldots, a_1, a_0)$, $b = (b_{m-1}, \ldots, b_1, b_0) \in \mathbb{F}_{2^m}$ and generating polynomial $f(x)$.

Ensure: $c = a \cdot b$.

1: Set $c \leftarrow 0$.
2: **for** $i = 0$ to $m - 1$ **do**
3: $\quad c \leftarrow c + a_0 b$
4: $\quad b \leftarrow b \cdot x \bmod f(x)$
5: $\quad a \leftarrow a \gg 1$
6: **end for**

via $f_9(x) = x^9 + x + 1$, see Fig. 3.3 and 3.4, where $\oplus$ denotes an XOR gate and $\odot$ denotes an AND gate.

Algorithm 3 Bit-serial left-to-right multiplication

Require: $a = (a_{m-1}, \ldots, a_1, a_0)$, $b = (b_{m-1}, \ldots, b_1, b_0) \in \mathbb{F}_{2^m}$ and generating polynomial $f(x)$.

Ensure: $c = a \cdot b$.

1: Set $c \leftarrow 0$.
2: **for** $i = m - 1$ downto 0 **do**
3: $\quad c \leftarrow c \cdot x + a_{m-1} b \bmod f(x)$
4: $\quad a \leftarrow a \ll 1$
5: **end for**

Linear feed back shift registers are used to perform reductions $x \cdot b(x) \bmod f(x)$ or $x \cdot c(x) \bmod f(x)$ correspondingly in either architecture. However m registers for the operand b can be saved in the second structure compared with the first one in which both the contents of b and c need to be updated each iteration.

The lengths of the critical paths in both architectures are the same and equal to

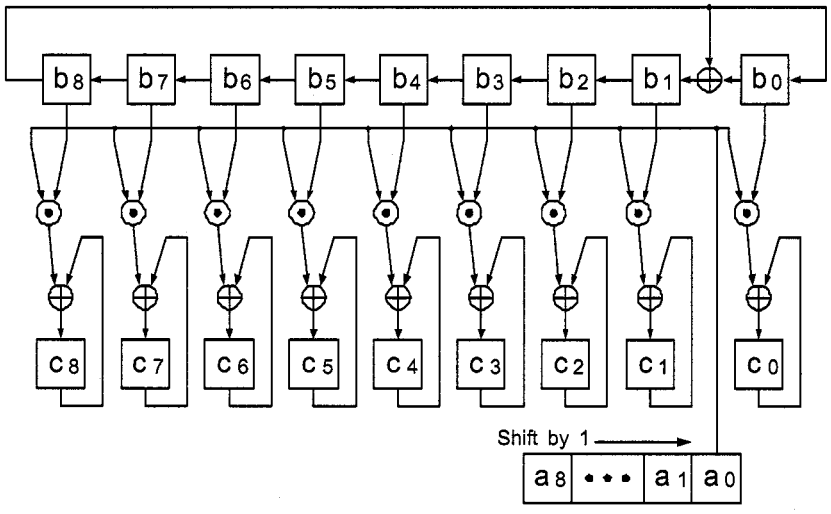


Figure 3.3: Least significant bit-serial structure over $\mathbb{F}_{2^9}$ constructed via $f_9(x) = x^9 + x + 1$

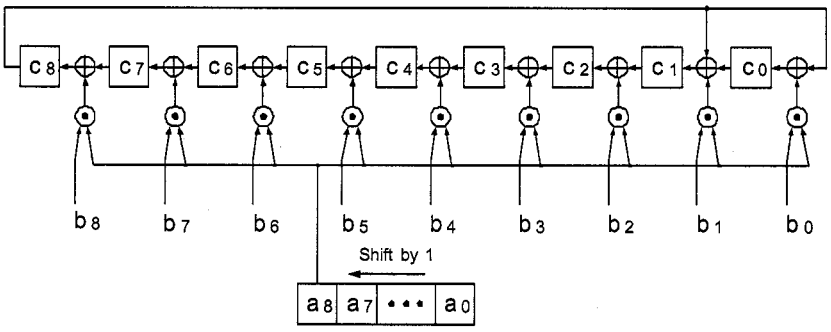


Figure 3.4: Most significant bit-serial structure over $\mathbb{F}_{2^9}$ constructed via $f_9(x) = x^9 + x + 1$

$T_A + T_X$, where T_A denotes the delay of a 2-in-1-out AND gate and T_X denotes the delay of a 2-in-1-out XOR gate.

Digit-Serial Structure

The digit-serial multiplier is a parallel version of the bit-serial one. Instead of computing one bit of the product, the digit-serial multiplier can compute multiple bits of the product in each clock cycle. Let D denote the digit size, then it takes $\lceil \frac{m}{D} \rceil$ clock cycles to complete one multiplication in $\mathbb{F}_{2^m}$. The MSD serial multiplier is superior in terms of low power [74] as well as less area, which makes it suitable for cryptographic applications. Let $n = min\{l \mid l \in \mathbb{Z},\ l \geq m,\ \text{and}\ D \mid l\}$. Let $a' = \sum_{i=0}^{n-1} a'_i x^i$, where $a'_i = a_i$ if $0 \leq i \leq m-1$, otherwise $a'_i = 0$.

Algorithm 4 Digit-serial left-to-right multiplication

Require: $a = (a_{m-1}, \ldots, a_1, a_0)$, $b = (b_{m-1}, \ldots, b_1, b_0) \in \mathbb{F}_{2^m}$ and generating polynomial $f(x)$.

Ensure: $c = a \cdot b \bmod f(x)$.

1: $c \leftarrow 0$, $a' \leftarrow a$.
2: **for** $i = \lceil \frac{m}{D} \rceil - 1$ downto 0 **do**
3: $\quad d_{D-1} \leftarrow a'_{n-1} x^{D-1} \cdot b \bmod f(x)$.
4: $\quad d_{D-2} \leftarrow a'_{n-2} x^{D-2} \cdot b \bmod f(x)$.
5: $\quad \cdots$ {Computations of d_j can be done in parallel.}
6: $\quad d_1 \leftarrow a'_{n-D+1} x \cdot b \bmod f(x)$.
7: $\quad d_0 \leftarrow a'_{n-D} \cdot b \bmod f(x)$.
8: $\quad c \leftarrow (x^D \cdot c \ mod\ f(x)) + \sum_{j=0}^{D-1} d_j$.
9: $\quad a' \leftarrow a' \ll D$.
10: **end for**

In accordance with Algorithm 4, we can derive the digit serial multiplier in $\mathbb{F}_{2^m}$. We provide an example in which $m = 239$ and $D = 4$ and the generating polynomial is $f_{239}(x) = x^{239} + x^{36} + 1$, see Fig. 3.5.

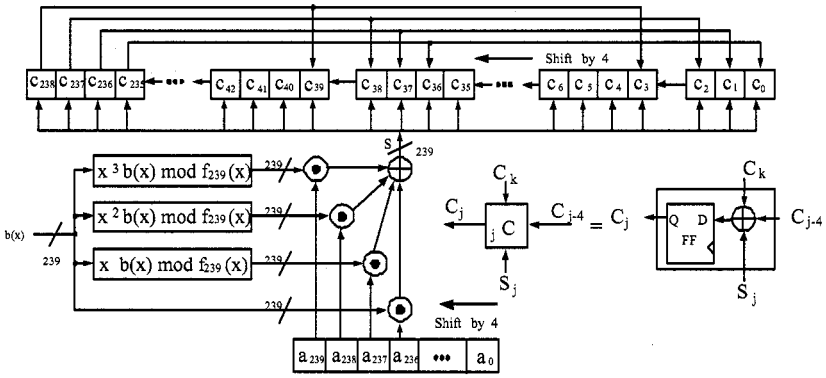


Figure 3.5: Most Significant Digit-serial multiplier over $\mathbb{F}_{2^{239}}$ with $D = 4$

The MSD serial multiplier, constructed in a similar way as the MSB serial multiplier, contains mainly two parts, LFSRs for computing $c(x) \leftarrow x^D \cdot c(x) + s(x)$ and XOR-AND arrays for computing $s(x) = \sum_{i=0}^{D-1} a_{n-D+i} x^i b(x) \bmod f_m(x)$.

Bit-Parallel Structure

Bit-parallel multipliers can complete a multiplication in one clock cycle. However high circuit complexity counteracts the benefit of fast operation speed. As a result these multipliers can not be adopted directly in the cryptographic applications with large operand sizes. Nevertheless, the bit-parallel structure can be still applied to the ground level of the hybrid multiplier of the composite field $\mathbb{F}_{2^{nm}}$. The Karatsuba-Ofman algorithm [62], repeatedly breaking the multiplication into pieces, is often used in software. Nevertheless the overhead in breaking down and recombining the parts, involved in the Karatsuba-Ofman algorithm, makes this algorithm prohibitive for hardware. The conventional method to derive such a bit-parallel multiplier contains

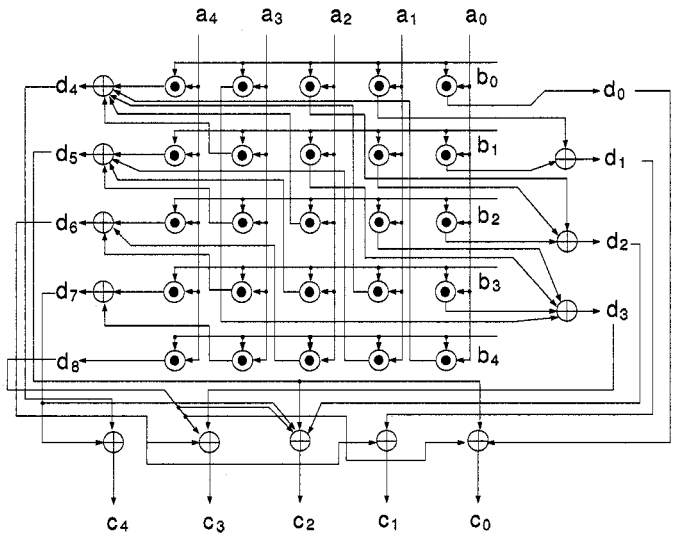


Figure 3.6: Bit-parallel multiplier over $\mathbb{F}_{2^5}$ constructed via $f_5(x)$.

mainly two steps. The first is to use Mastrovito's method [42] to compute the partial product with $2m-1$ bits before reduction. The second is to perform the reduction exploiting the standard technique for low Hamming weight irreducible polynomial [29, 81].

We provide an example of the bit-parallel multiplier in $\mathbb{F}_{2^5}$ constructed via $f_5(x) = x^5 + x^2 + 1$. Let a, b and $c = a \cdot b$ denote the elements in $\mathbb{F}_{2^5}$. Let a_s, b_t and c_j denotes their coefficients in $\mathbb{F}_2$ accordingly where $0 \leq s, t \leq 4$ and $0 \leq j \leq 8$.

By Mastrovito's method [42, 77],

$$c = \sum_{j=0}^{8} d_j \beta^j \qquad\qquad d_j = \sum_{0 \leq s,t \leq 4}^{s+t=j} a_s b_t \tag{3.4}$$

where $d_j \in \mathbb{F}_2$. With the following equations,

$$x^5 = x^2 + 1 \qquad x^6 = x^3 + x \qquad x^7 = x^4 + x^2 \qquad x^8 = x^3 + x^2 + 1 \tag{3.5}$$

then we can get the coefficients c_j as follows.

$$c_4 = d_7 + d_4 \qquad c_3 = d_8 + d_6 + d_3 \qquad c_2 = d_8 + d_7 + d_5 + d_2$$

$$c_1 = d_6 + d_1 \qquad c_0 = d_8 + d_5 + d_0 \tag{3.6}$$

The bit-parallel multiplier over $\mathbb{F}_{2^5}$ is shown in Fig. 3.6, where $\odot$ denotes an AND gate and $\bigoplus$ denotes a XOR gate. A trinomial bit-parallel multiplier in $\mathbb{F}_{2^m}$ can be built with at most m^2 AND gates and $m^2 - 1$ XOR gates and the critical path is less than $T_A + (2 + \lceil log_2 m \rceil)T_X$. A pentanomial one can be built with at most m^2 AND gates and $m^2 + 2m - 3$ XOR gates [29] and the critical path is less than $T_A + (3 + \lceil log_2 m \rceil)T_X$. More details of timing and complexity analysis for bit-parallel architecture can be found in [29,81].

3.2.2 Multipliers via Maximum Weight Irreducible Polynomials

Irreducible All-One-Polynomial (AOP) over $\mathbb{F}_{2^m}$, of the form $f_m(x) = x^m + x^{m-1}+$, $\ldots, +x+1$, can be utilized to construct efficient bit-parallel multiplier [39]. The basic idea is to exploit the following equation so that x^i, where $i > m$, can be replaced with x^{i-m-1}.

$$(x+1)f_m(x) = (x+1)(x^m + x^{m-1}+, \ldots, +x+1)$$

$$x^{m+1} = 1 \tag{3.7}$$

In [14,15], the bit-serial multipliers using irreducible all-one polynomials are proposed. In this architecture, the partial product is stored in redundant registers of size $m+1$ instead of m to decrease the wire density of feedback network. m XOR gates are used to

perform the final reduction. In fact, the feedback network is equivalent to cyclic shift. And this method can be applied to derive the MSD/LSD serial multipliers since the feedback network will be certainly simple according to Equation 3.7. To be specific, m can be equal to 2,4,10,12,18,28,36,52,58,60,66,82,100,106,130,138,148,162,172,178, 180,196,210,226,268 and 292 such that there exists AOP in $\mathbb{F}_{2^m}$ ($m \leq 300$). For larger values of m, one can refer to [44] (it is not difficult to prove that m must be an even number). In Table 3.1, we list the alternative field trinomials or pentanomials for such special binary fields. Additionally, we can derive a hybrid-digit serial multiplier whose ground field is constructed via irreducible AOP.

Table 3.1: Alternative field trinomials or pentanomials.

Field size m	$f_m(x)$	Field size m	$f_m(x)$	Field size m	$f_m(x)$
100	$x^{100}+x^{15}+1$	162	$x^{162}+x^{27}+1$	210	$x^{210}+x^{7}+1$
106	$x^{106}+x^{15}+1$	172	$x^{171}+x^{+}1$	226	$x^{226}+x^{10}+x^{7}+x^{3}+1$
130	$x^{130}+x^{3}+1$	178	$x^{178}+x^{31}+1$	268	$x^{268}+x^{25}+1$
138	$x^{138}+x^{8}+x^{7}+x+1$	180	$x^{180}+x^{3}+1$	292	$x^{292}+x^{37}+1$
148	$x^{148}+x^{27}+1$	196	$x^{196}+x^{3}+1$		

Table 3.2: Irreducible $F_{m,n}(x) = \frac{x^{m+1}+1}{x+1} + x^n$ with $n \leq m/2$ for $100 \leq m \leq 300$, and irreducible pentanomials $f_m(x)$ with degree m.

m	n	$f_m(x)$	m	n	$f_m(x)$
101	6, 18	$x^{101}+x^{7}+x^{6}+x+1$	197	11, 27	$x^{197}+x^{9}+x^{4}+x^{2}+1$
117	14, 19	$x^{117}+x^{5}+x^{2}+x+1$	213	26, 67	$x^{213}+x^{6}+x^{5}+x^{2}+1$
125	6, 31, 38, 46	$x^{125}+x^{7}+x^{6}+x^{5}+1$	221	35, 74	$x^{221}+x^{8}+x^{6}+x^{2}+1$
133	22, 31, 46	$x^{133}+x^{9}+x^{8}+x^{2}+1$	229	39, 63	$x^{229}+x^{10}+x^{4}+x+1$
141	67	$x^{141}+x^{10}+x^{4}+x+1$	237	59, 86, 94	$x^{237}+x^{7}+x^{4}+x+1$
143	40, 41, 68	$x^{143}+x^{5}+x^{3}+x^{2}+1$	245	3, 87, 102	$x^{245}+x^{6}+x^{4}+x+1$
149	6, 43, 55, 70	$x^{149}+x^{10}+x^{9}+x^{7}+1$	261	34	$x^{261}+x^{7}+x^{6}+x^{4}+1$
157	3, 46	$x^{157}+x^{6}+x^{5}+x^{2}+1$	269	7, 95, 123	$x^{269}+x^{7}+x^{6}+x+1$
173	43	$x^{173}+x^{8}+x^{5}+x^{2}+1$	277	90, 130, 135	$x^{277}+x^{12}+x^{6}+x^{3}+1$
181	67, 75, 78	$x^{181}+x^{7}+x^{6}+x+1$	285	127	$x^{285}+x^{10}+x^{7}+x^{5}+1$
189	34, 62, 71	$x^{189}+x^{6}+x^{5}+x^{2}+1$	293	47, 131	$x^{293}+x^{11}+x^{6}+x+1$

Another strategy for fast reduction is to select $f(x)$ so that it has a low Hamming

weight multiple $g(x)$ of degree slightly greater than n. Multiplication is then performed modulo $g(x)$, followed by a reduction by $f(x)$ whenever a representation in canonical form is desired. In [52], Ahmadi and Menezes proposed the case of weight-m polynomials, namely $f(x) = F_{m,n}(x)$ where

$$F_{m,n}(x) = x^m + x^{m-1} + \ldots + x^{n+1} + x^{n-1} + \ldots + x + 1 = \frac{x^{m+1}+1}{x+1} + x^n \quad (3.8)$$

and we can take $g(x) = (x+1)f(x) = x^{m+1} + x^{n+1} + x^n + 1$. The Hamming weight of $g(x)$ is 4, and its middle terms are consecutive. Even though it is not as efficient as the field multiplications using irreducible trinomials but it can be more efficient than those multiplications using irreducible pentanomials. In Table 3.2, we provide $F_{m,n}(x)$ from [52] and irreducible pentanomials for $\mathbb{F}_{2^m}$, where $100 \leq m \leq 300$.

3.2.3 Normal Basis Multiplier

In this section, we first collect the results related to constructions for normal basis of low complexity. The Massey-Omura architecture [24] is introduced. Then we address the issues of efficient design for normal basis multiplier including both the technique of shortening the critical path and lowering the circuit complexity.

Lowest circuit complexity can be obtained for several special fields by wisely constructing the normal bases. These bases are called optimal normal bases (ONB). We present two constructions for optimal normal basis due to [43].

Theorem 3.2.1. Suppose $m+1$ is a prime and q is primitive in $\mathbb{Z}_{m+1}$, where q is a prime or prime power. Then the m nonunit $(m+1)$th roots of unity are linearly independent and they form an optimal normal basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$

Theorem 3.2.2. Let $2m+1$ be a prime and assume that either

1. 2 is primitive in $\mathbb{Z}_{2m+1}$, or

2. $2m + 1 \equiv 3 \mod 4$ and 2 generate the quadratic residues in $\mathbb{Z}_{2m+1}$. Then $\gamma = \theta + \theta^{-1}$ generates an optimal normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$, where θ is a primitive $2m + 1$ st root of unity.

Proofs of these two theorems can be found in [44]. The two bases constructed according to Theorems 3.2.1 and 3.2.2 are called a type I and type II optimal normal bases, respectively.

Massey-Omura's Architecture

The idea of Massey-Omura's architecture for normal basis multiplier is to use the same combinational circuits, together with rotate registers, to compute the coefficients of the product sequentially. This architecture has a parallel-in, serial-out structure.

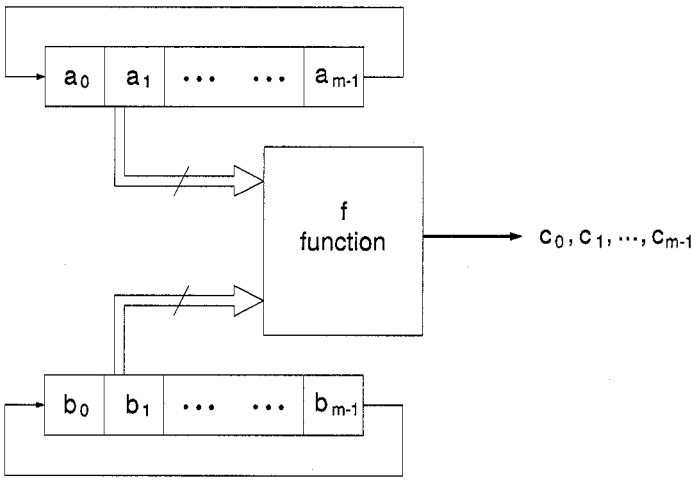


Figure 3.7: Massey-Omura parallel-in, serial-out architecture for $\mathbb{F}_{2^m}$.

Suppose that $B = \{\gamma, \gamma^2, \gamma^{2^2}, \ldots, \gamma^{2^{m-1}}\}$ is a normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Let $a = (a_0, a_1, \ldots, a_{m-1})$ and $B = (b_0, b_1, \ldots, b_{m-1})$ be two elements of $\mathbb{F}_{2^m}$ written as vectors, where $a_i, b_i \in \mathbb{F}_2$. Let $c = (c_0, c_1, \ldots, c_{m-1})$ be their product: $c = ab$. Then

any coordinates, say c_0, of c is a function of a and b, i.e., $c_0 = f(a,b)$. With the function f, we can also compute $(c^2)_0 = f(a^2,b^2)$, i.e., $c_1 = f(a^2,b^2)$. Since squaring an element in $\mathbb{F}_{2^m}$ with normal basis representation is equivalent to rotation, the Massey-Omura's architecture, shown in Fig. 3.7, can be obtained [49]. One example for $\mathbb{F}_{2^5}$ is used to illustrate the derivation of the f function. The field and the normal basis presented in this example will be used in the remaining part of this section.

Let $\gamma = \theta + \theta^{-1}$, where $\theta \in \mathbb{F}_{2^{10}}$ and $\theta^{11} = 1$ (Since $2^{10} \equiv 1 \bmod 11$, θ can be written in the form $\theta = \alpha^{\frac{2^{10}-1}{11}}$ where α is the primitive element of $\mathbb{F}_{2^{10}}$). The f function can be obtained by computing $\gamma\gamma^{2^i}$ where $0 \leq i \leq 4$.

$$
\begin{aligned}
\gamma\gamma &= \gamma^2 \\
\gamma\gamma^2 &= (\theta+\theta^{-1})(\theta^2+\theta^{-2}) = \theta+\theta^{-1}+\theta^3+\theta^{-3} \\
&= \theta+\theta^{-1}+\theta^8+\theta^{-8} = \gamma+\gamma^{2^3} \\
\gamma\gamma^{2^2} &= (\theta+\theta^{-1})(\theta^4+\theta^{-4}) = \theta^5+\theta^{-5}+\theta^3+\theta^{-3} \\
&= \theta^{16}+\theta^{-16}+\theta^8+\theta^{-8} = \gamma^{2^4}+\gamma^{2^3} \\
\gamma\gamma^{2^3} &= (\theta+\theta^{-1})(\theta^8+\theta^{-8}) = \theta^7+\theta^{-7}+\theta^9+\theta^{-9} \\
&= \theta^4+\theta^{-4}+\theta^2+\theta^{-2} = \gamma^{2^2}+\gamma^2 \\
\gamma\gamma^{2^4} &= (\theta+\theta^{-1})(\theta^{16}+\theta^{-16}) = \theta^{15}+\theta^{-15}+\theta^{17}+\theta^{-17} \\
&= \theta^4+\theta^{-4}+\theta^{16}+\theta^{-16} = \gamma^{2^2}+\gamma^{2^4}
\end{aligned}
\tag{3.9}
$$

Then we have

$$c_0 = f(a,b) = a_1b_0 + (a_0 + a_3)b_1 + (a_3 + a_4)b_2 + (a_1 + a_2)b_3 + (a_2 + a_4)b_4. \quad (3.10)$$

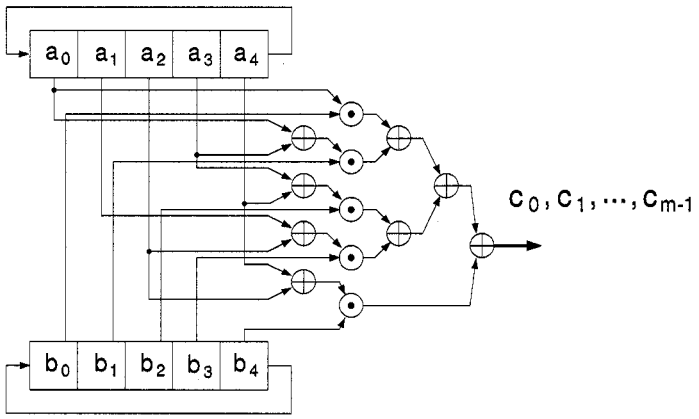


Figure 3.8: The Massey-Omura type II ONB multiplier in $\mathbb{F}_{2^5}$.

In accordance with Equation 3.10, we can get the Massey-Omura multiplier in $\mathbb{F}_{2^5}$, see Fig. 3.8.

Technique of Shortening the Critical Path

The quite long critical path of the Massey-Omura's normal basis multiplier, proportional to $log_2 m$, has a negative effect on timing which make it unsuitable to be directly adopted in real applications. Agnew et al. [3] proposed a sequential multiplier which has a parallel-in, parallel-out structure based on the same algorithm of multiplication used in the Massey-Omura's architecture. Instead of computing one coefficient each clock cycle, the Agnew's normal basis multiplier computes the partial sum for all coefficients of the product (see Equation 3.11), and the results come out in parallel after m clock cycles. The most attractive advantage of this architecture is that the

length of the critical path can be shortened significantly.

$$c_0 = \underline{a_1 b_0} + (a_0 + a_3)b_1 + (a_3 + a_4)b_2 + (a_1 + a_2)b_3 + (a_2 + a_4)b_4$$

$$c_1 = a_2 b_1 + \underline{(a_1 + a_4)b_2} + (a_0 + a_4)b_3 + (a_2 + a_3)b_4 + (a_0 + a_3)b_0$$

$$c_2 = a_3 b_2 + (a_0 + a_2)b_3 + \underline{(a_0 + a_1)b_4} + (a_3 + a_4)b_0 + (a_1 + a_4)b_1$$

$$c_3 = a_4 b_3 + (a_1 + a_3)b_4 + (a_1 + a_2)b_0 + \underline{(a_0 + a_4)b_1} + (a_0 + a_2)b_2$$

$$c_4 = a_0 b_4 + (a_2 + a_4)b_0 + (a_2 + a_3)b_1 + (a_0 + a_1)b_2 + \underline{(a_1 + a_3)b_3} \tag{3.11}$$

Fig. 3.9 shows the circuit of Agnew et al. in $\mathbb{F}_{2^5}$ where a type II ONB is used.

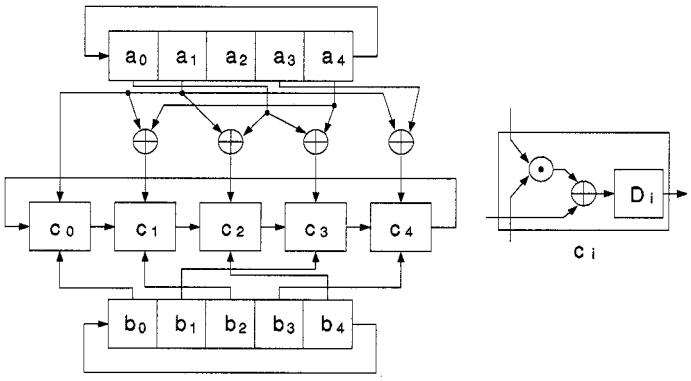


Figure 3.9: The Massey-Omura type II ONB multiplier in $\mathbb{F}_{2^5}$ with shorter critical path proposed by Agnew et al.

Technique of Simplifying the Circuit Complexity

Kwon et al. [31] improved the multiplier of Agnew et al. so that a more compact multiplier in terms of area could be obtained. The critical path delay of this new architecture is the same as that of Agnew el al. The main idea is to re-order the columns of Equation 3.11 so that less combinational cells can be used, see Equation

3.12.

$$
\begin{aligned}
c_0 &= \underline{(a_3+a_4)b_2} + a_1b_0 + (a_1+a_2)b_3 + (a_0+a_3)b_1 + (a_2+a_4)b_4 \\
c_1 &= (a_4+a_0)b_3 + \underline{a_2b_1} + (a_2+a_3)b_4 + (a_1+a_4)b_2 + (a_0+a_3)b_0 \\
c_2 &= (a_0+a_1)b_4 + a_3b_2 + \underline{(a_3+a_4)b_0} + (a_0+a_2)b_3 + (a_1+a_4)b_1 \\
c_3 &= (a_1+a_2)b_0 + a_4b_3 + (a_0+a_4)b_1 + \underline{(a_1+a_3)b_4} + (a_0+a_2)b_2 \\
c_4 &= (a_2+a_3)b_1 + a_0b_4 + (a_0+a_1)b_2 + (a_2+a_4)b_0 + \underline{(a_1+a_3)b_3}
\end{aligned} \tag{3.12}
$$

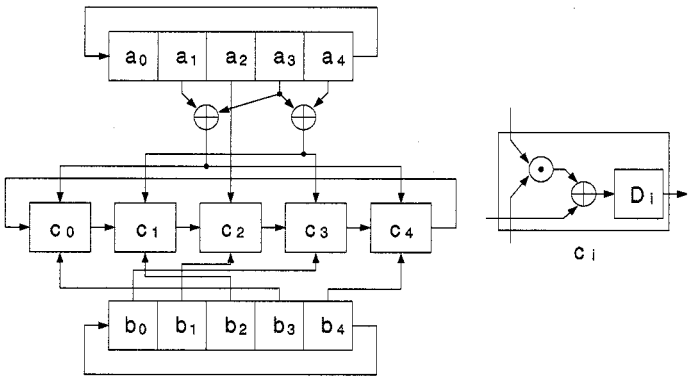


Figure 3.10: The Massey-Omura type II multiplier in $\mathbb{F}_{2^5}$ with shorter critical path and lower circuit complexity proposed by Kwon et al.

The complexity analysis for timing and area is shown as follows. Compared with Agnew et al.'s architecture, this novel multiplier needs less than $m+(m-1)(3m-1)/2$ XOR gates instead of $(2m-1)^2$ in case of a type II ONB exists for $\mathbb{F}_{2^m}$. The number of AND gates are the same for both architectures and equal to m. $3m$ flip-flops are needed for both. The critical path delay is less than $T_A + 2T_X$ for both multipliers using a type II ONB.

3.3 Inverter

The most difficult finite field operation to implement in hardware is inversion. The extended Euclidean algorithm (EEA) is a natural choice for computing inverses in polynomial representations.

Algorithm 5 Polynomial Basis Inversion Algorithm using EEA

Require: $a \in \mathbb{F}_{2^m}$, $a \neq 0$.

Ensure: $a^{-1} \bmod f$.

1: $u \leftarrow a, v \leftarrow f, g_1 \leftarrow 1, g_2 \leftarrow 0$.
2: **while** $u \neq 1$ **do**
3: $\quad j \leftarrow deg(u) - deg(v)$.
4: $\quad$ **if** $j < 0$ **then**
5: $\quad\quad u \leftrightarrow v, g_1 \leftrightarrow g_2, j \leftarrow -j$.
6: $\quad$ **end if**
7: $\quad u \leftarrow u + z^j v, g_1 \leftarrow g_1 + z^j g_2$.
8: **end while**
9: Return(g_1).

Algorithm 5 maintains the invariants

$$ag_1 + fh_1 = u \qquad\qquad ag_2 + fh_2 = v$$

for some h_1 and h_2 not explicitly calculated. The algorithm terminates when $u = 1$, in which case $g_1 = a^{-1}$ [28]. The alternative approach is to exploit Fermat's little theorem by computing the multiplicative inversion using multiple multiplications, see Equation 3.13.

$$a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2 \tag{3.13}$$

Hence a^{-1} can be computed once $a^{2^{m-1}-1}$ is known. A technique for minimizing the number of general multiplications in this computation is proposed by Itoh and Tsujii in [27]. The method is based on the identities.

$$a^{2^{m-1}-1} = \begin{cases} a^{(2^{\frac{m-1}{2}}-1)(2^{\frac{m-1}{2}}+1)} = (a(a^{2^{\frac{m-1}{2}}}))^{2^{\frac{m-1}{2}}-1}, & m \text{ odd} \\ aa^{2^{m-1}-2} = a(a^{2^{m-2}-1})^2, & m \text{ even} \end{cases} \tag{3.14}$$

By recursively applying Equation 3.14, the inversion can be computed via $\lfloor log_2(m-1) + H_w(m-1) - 1 \rfloor$ multiplications and several squarings, where $H_w()$ denotes the Hamming weight of the binary representation of the operand (more details can be found in [27, 53]). This algorithm can be applied to the inversion in both normal basis representation and polynomial basis representation in case squaring is easy to be computed. Inversion by multiplication does not add significantly to the complexity of a hardware design, but can severely impact performance if it is needed frequently. This is the reason why most hardware designers try to avoid inversions during intermediate computations as possible as they could. Additional functionality must be incorporated into the controller but extensive modifications to the core circuite are not required. If affine coordinates are preferred, then inversion will undoubtedly be computed using extended Euclidean algorithm, such that more circuit complexity will be added to achieve high performance [26].

3.4 Composite Field Arithmetic

It is well known that a composite field $\mathbb{F}_{2^{nm}}$ yields efficient implementations if the basis is chosen wisely. Let $gcd(n, m) = 1$. Based upon Theorem 2.1.4, it is possible to devise a hybrid multiplier by applying bit-parallel architectures to arithmetic in the ground field $\mathbb{F}_{2^n}$ and bit-serial structure to arithmetic in the tower field $\mathbb{F}_{(2^n)^m}$ [55].

Some hybrid architectures of multipliers for composite fields have been published. For instance, in [51], normal basis is chosen for both ground and tower fields. In [56], polynomial basis is also chosen for both and constructed via low Hamming weight irreducible polynomials such as trinomials and pentanomials. Even though this hybrid multiplier is believed to be more efficient than the conventional digit-serial multiplier introduced in Section 3.2.1, there has been very little empirical evidence, in particular for FPGA implementations to support this view. In this work, we implement both multipliers for several special composite fields in VHDL and port them to the same FPGA device. Finally comparisons in terms of timing and area are demonstrated. More details will be provided in the first part of this section.

The efficiency of the bit-parallel & serial structure significantly determine the performance of the hybrid multiplier. However to use such a composite field multiplier in real applications, the matrix for basis conversion should be obtained within computational time. We present a novel hybrid normal basis multiplier based on the bit-serial architecture devised by Kwon et al. [31], considering that this bit-serial multiplier is efficient in terms of timing and area. The ground field is constructed via special trinomial so that high performance can be accomplished at the bit-parallel level due to the simplicity of trinomials. Secondly, the conversion matrix can be obtained easily. More details related to the efficient design of this composite field arithmetic including squaring, multiplication and inversion, and the technique of basis conversion are covered in the second part of this section.

3.4.1 Hybrid Multiplier Constructed via Low Hamming Weight Irreducible Polynomials

We concentrate on these composite fields which can be constructed via an irreducible pentanomial of degree nm but not an irreducible trinomial of degree nm, where $gcd(n, m) = 1$. In Fig. 2.1, let α denote the generator of the polynomial basis of $\mathbb{F}_{2^{nm}}$ with respect to $\mathbb{F}_2$. Let β denote the generator of the polynomial basis of $\mathbb{F}_{2^n}$ with respect to $\mathbb{F}_2$. And let γ denote the generator of the polynomial basis of $\mathbb{F}_{2^m}$ with respect to $\mathbb{F}_2$. There always exist irreducible trinomials or pentanomials for the field size $n \leq 10,000$ which can be exploited to construct the binary fields with respect to $\mathbb{F}_2$ [25]. In our approach, we assume that α is the root of an irreducible pentanomial of degree nm. The conventional representation is to use the standard basis constructed by α, namely $B_s = \{1, \alpha, \alpha^2, \cdots, \alpha^{nm-1}\}$, such that the simplicity of pentanomials can be exploited to obtain the digit-serial multiplier shown in Fig. 3.5 (for pentanomial cases, the feedback network of LFSRs is more complicated).

Table 3.3: Selected pentanomials for large fields and trinomials for both tower fields and ground fields

$f_s(x)$ of degree nm	$f_t(x)$ of degree m	$f_g(x)$ of degree n
82, 8, 3, 1	2, 1	41, 3
164, 10, 8, 7	4, 1	
205, 9, 5, 2	5, 2	
246, 11, 2, 1	6, 1	

For the composite field $\mathbb{F}_{2^{nm}}$ where $gcd(n, m) = 1$, the field elements can be represented in a different way. Let β and γ are the roots of irreducible trinomials or pentanomials of degree of n and m separately, then γ can be raised up to generate the polynomial basis, namely $B_t = \{1, \gamma, \gamma^2, \cdots, \gamma^{m-1}\}$, for the tower field $\mathbb{F}_{(2^n)^m}$ with respect to $\mathbb{F}_{2^n}$. β can be used to construct the polynomial basis, namely $B_g =$

Table 3.4: Selected pentanomials for large fields, trinomials for tower fields and pentanomials for ground fields

$f_s(x)$ of degree nm	$f_t(x)$ of degree m	$f_g(x)$ of degree n
96, 10, 9, 6	3, 1	32, 7, 3, 2
160, 5, 3, 2	5, 2	
224, 9, 8, 3	7, 1	
288, 11, 10, 1	9, 1	

$\{1, \beta, \beta^2, \cdots, \beta^{n-1}\}$ for the ground field $\mathbb{F}_{2^n}$ with respect to $\mathbb{F}_2$, see Theorem 2.1.4.

For some degrees nm, pentanomials are the lowest Hamming weight irreducible polynomials to construct $\mathbb{F}_{2^{nm}}$. However it is still possible to get a more efficient digit-serial multiplier by adopting the second representation because there may exist at least one irreducible trinomial of degree n or m. In Table 3.3, we list generating pentanomials $f_s(x)$ for large fields and generating trinomials $f_t(x)$ and $f_g(x)$ for both tower fields and ground fields. In Table 3.4, the generating polynomial for $\mathbb{F}_{2^n}$ is a pentanomial. The composite exponents are bounded by $80 \leq nm \leq 300$ because of the requirements of operand sizes for cryptographic applications. These tables are organized as follows: A trinomial $x^n + x^k + 1$, with $n > k > 0$ is represented by the pair n, k. A pentanomial $x^n + x^{k_3} + x^{k_2} + x^{k_1} + 1$, with $n > k_3 > k_2 > k_1 > 0$, is represented by the quadruple n, k_3, k_2, k_1 [25]. For example, the field $\mathbb{F}_{2^{205}}$ has no trinomial basis and a pentanomial basis can be used in this case. The simplest pentanomial basis is generated by the roots of $x^{205} + x^9 + x^5 + x^2 + 1$. However since $205 = 5 \cdot 41$ and trinomial basis exists for both fields, $\mathbb{F}_{2^5}$ and $\mathbb{F}_{2^{41}}$, we may realize efficient field arithmetic using the trinomial bases. In case $80 \leq nm \leq 300$, there exists approximately 30 composite fields which can be constructed in the first way and approximately 34 ones which can be constructed in the second way.

For convenience of understanding, we provide a concrete example in which $n = 5$,

$m = 9$ and the generating polynomials of $\mathbb{F}_{2^{45}}$, $\mathbb{F}_{2^9}$ and $\mathbb{F}_{2^5}$ are $f_s(x) = x^{45} + x^4 + x^3 + x + 1$, $f_t(x) = x^9 + x + 1$ and $f_g(x) = x^5 + x^2 + 1$ accordingly. We are applying the bit-serial structure to the tower field (see Fig. 3.4 where $\odot$ denotes the bit-parallel multiplier and $\bigoplus$ denotes 5 bitwise XOR arrays in $\mathbb{F}_{2^5}$) and applying the bit-parallel structure to the ground field (see Fig. 3.6) to get the hybrid architecture.

The complexity comparisons in terms of ASIC estimation are summarized in Table 3.5, where Hybrid I denotes the hybrid multiplier in which the ground field is constructed via a trinomial and Hybrid II denotes the one in which the ground field is constructed via a pentanomial.

Table 3.5: Complexity comparisons between conventional and hybrid digit-serial multipliers constructed via low Hamming weight irreducible polynomials

Architectures	# AND	# XOR	Critical path length
Conventional	mn^2	$mn^2 + 2n^2 + 2n$	$T_A + (1 + \lceil \log_2(2n+1) \rceil) T_X$
Hybrid I	mn^2	$mn^2 + mn + n - m$	$T_A + (\lceil \log_2(n+2) \rceil) T_X$
Hybrid II	mn^2	$mn^2 + 3mn + n - 3m$	$T_A + (4 + \lceil \log_2 n \rceil) T_X$

For the conventional architecture, the total number of 2-input AND gates is mn^2 (n is the digit size) and the number of 2-input XOR gates is determined by three parts, feedback network, modular components for $\alpha^i b$ and the partial sum in Step 8 of Algorithm 4. For Hybrid I, a bit-parallel multiplier in $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ can be built with at most n^2 AND gates and $n^2 - 1$ XOR gates [81]. For Hybrid II, a bit-parallel multiplier can be built with at most n^2 AND gates and $n^2 + 2n - 3$ XOR gates [29]. The hybrid multiplier contains m bit-parallel multipliers and together with $(m+1)n$ XOR gates considering that the tower field is constructed via an irreducible trinomial in our approach.

Table 3.6: Performance & cost comparisons between conventional and hybrid I multipliers

Architectures	nm	# FF	# LUT	# CLB slices	Clock period	Latency (ns)
Conv	82	831	3311	1992	7.162	14.32
	164	2065	6307	3630	7.666	30.66
	205	2605	7987	4630	8.275	41.38
	246	3227	9594	5162	9.357	56.14
Hyb I	82	594	2963	1885	6.837	13.67
	164	1438	5681	3339	6.998	27.99
	205	1686	7101	3916	7.329	36.65
	246	2424	8669	4434	8.695	52.17

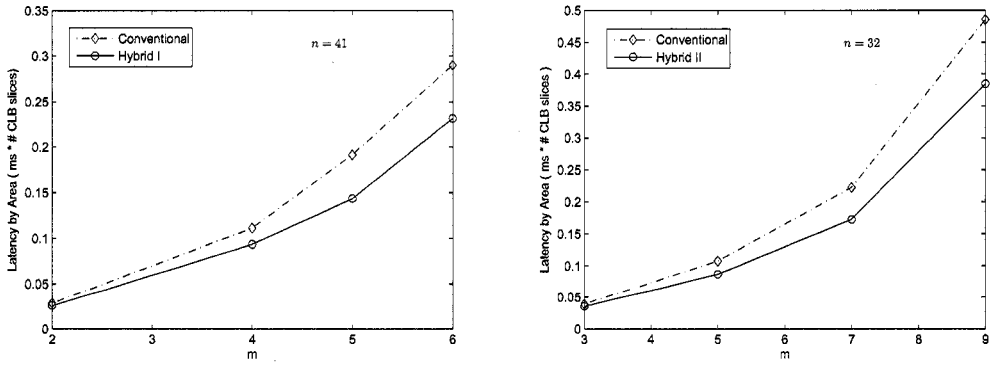
The difference in terms of latency between the conventional and hybrid architectures is not big. If the $m \ll n$, the hybrid architecture is more efficient in terms of area. The wire density due to the feedback network in the hybrid multipliers is decreased. Additionally more regularity can be gained in the hybrid multipliers since the bit-parallel component has the same structure.

Table 3.7: Performance & cost comparisons between conventional and hybrid II multipliers

Architectures	nm	# FF	# LUT	# CLB slices	Clock period	Latency (ns)
Conv	96	1026	2994	1776	7.354	22.06
	160	1701	4833	2895	7.425	37.13
	224	2382	1755	4155	7.631	53.42
	288	3058	8686	5001	10.787	97.08
Hyb II	96	880	2757	1649	7.218	21.65
	160	1289	4541	2470	6.997	34.99
	224	1879	6335	3430	7.182	50.27
	288	2384	8016	4276	9.992	89.93

Three kinds of multipliers for those composite fields listed in Tables 3.3 and 3.4 (conventional, Hybrid I and Hybrid II) are implemented using the same FPGA device, XC2V1000-5-FF896 which contains 15,360 slice flip flops, 15,360 4-input look-up

tables (LUTs) and 7,680 configurable logic blocks (CLBs). Both synthesis and place & route are completed via Xilinx-ISE 7.1.



(a) Conventional vs. Hybrid I

(b) Conventional vs. Hybrid II

Figure 3.11: Conventional vs. Hybrid multipliers constructed via low Hamming weight irreducible polynomials, latency by area product

The performance comparisons based on the FPGA implementation results after placing & routing are summarized in Table 3.6 and 3.7. The comparisons of latency by area product are shown in Fig. 3.11. The optimization goal for synthesis is speed instead of area so that registers are replicated by 3 times on average to shorten the critical path for both architectures. However we find that less registers are used in the hybrid multipliers than in the conventional multipliers. This is because the hybrid multiplier is more regular and the wire density in feedback networks is decreased. In Fig. 3.11(a) and 3.11(b), we can see that the hybrid multipliers are 10%-to-33% more efficient than the conventional ones in terms of the product of latency by area. In particular its regularity and wire density make it more suitable for FPGA realizations than the conventional architecture.

3.4.2 A Novel Hybrid Multiplier and its Basis Conversion

Sunar et al. [78] recently presented two methods to construct composite field representations for efficient conversion. Let α be the primitive elements of $\mathbb{F}_{2^{nm}}$ defined by an irreducible polynomial. For the former, they demonstrated that the generator of the ground field can be constructed wisely as some power of α, and the generator of the tower field can be chosen as either α or some power of α. In this way, computations of the conversion matrix become very simple. But there is no guarantee that the generating polynomials of both ground field and tower field have the simple form such as trinomial or pentanomial. For the latter, they choose a low Hamming weight degree-m irreducible polynomial, namely $f_t^*(x)$ to build the tower field and construct the change of basis matrix from the field $\mathbb{F}_{2^{nm}}$ generated by the minimal polynomial of α, namely $f_t(x) = m_\alpha(x)$, to the field $\mathbb{F}_{2^{nm}}$ generated by $f_t^*(x)$, so that the bit-serial structure can be efficient. However, computing such matrix is equivalent to solving a set of equations which may be non-linear, i.e., the deduction process can be very complicated. In Reference [79], Sunar presented an algorithm based on the existing factorization algorithm to solve the basis conversion problem where the two representations are initially given, so it's possible to obtain efficient arithmetics for both tower field and ground field in case that two low Hamming weight irreducible polynomials are chosen to construct the composite field. It takes randomized polynomial time to find such conversion matrix when using this algorithm.

In our approach, we present a hybrid normal basis multiplier based on the bit-serial architecture devised by Kwon et al. [31], considering that this bit-serial multiplier is efficient in terms of timing and area. The ground field is constructed via a special trinomial, with the form $f_g(x) = x^{2^g} + x + 1$ or $f_g(x) = x^{2^g-1} + x^{2^t-1} + 1$, so that high performance can be accomplished at the bit-parallel level due to the simplicity

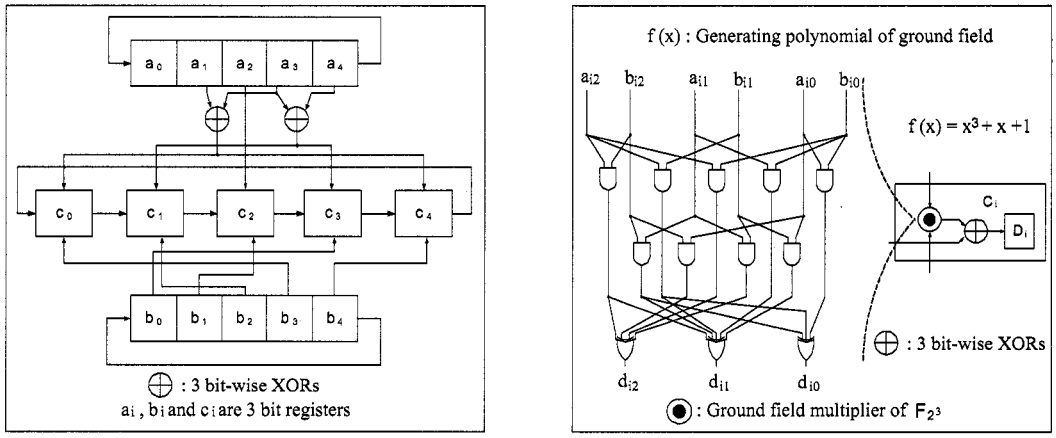
of trinomials. Secondly, the conversion matrix can be obtained easily, as computing such a matrix is equivalent to solving a set of linear equations. And these equations can be obtained by multiple squaring of the ground field generator with the standard basis representation. This process can be completed simply by programming. Even though there isn't such a trinomial for each field, we found that the existence satisfies most requirements of digit size, e.g., $2, 3, 4, 7, 15, 31, 63$ and 127. To find the relation between the normal element of $\mathbb{F}_{2^m}$ and the generator of $\mathbb{F}_{2^{nm}}$ is relatively easy. If the exhaustive search method is used, the computational time is $\mathcal{O}(\frac{\Phi(2^m-1)}{m})$. Since m is much smaller than nm, e.g., $m < 20$, this task can be done by machine very shortly.

In the following, we first discuss the field arithmetic for the composite field in the new hybrid basis representation. Then, we explain the technique of its basis conversion.

Multiplier

According to Theorem 2.1.4, we can construct the hybrid basis of the composite field $\mathbb{F}_{2^{nm}}$, where $gcd(n, m) = 1$, as follows. In Fig. 2.1, let α denote the generator of a standard trinomial/pentanomial basis $B_s = (1, \alpha, \ldots, \alpha^{nm-1})$ for $\mathbb{F}_{2^{nm}}$ with respect to $\mathbb{F}_2$; let γ denote the generator of the normal basis $B_t = (\gamma, \gamma^2, \ldots, \gamma^{2^{m-1}})$ for $\mathbb{F}_{2^{nm}}$ with respect to $\mathbb{F}_{2^n}$; and let β denote the root of $f_g(x) = x^{2^g} + x + 1 = 0$ or $f_g(x) = x^{2^{g-1}} + x^{2^t-1} + 1 = 0$, where $n = 2^g$ or $n = 2^g - 1$ such that $B_g = (1, \beta, \beta^2, \ldots, \beta^{n-1})$ can be used to represent the ground field elements with respect to $\mathbb{F}_2$. One example of $m = 5$ and $n = 3$ is used to illustrate the following derivations in which γ satisfies Equation 3.9 and $f_s(x) = x^{15} + x + 1$ and $f_g(x) = x^3 + x + 1$. The bit-serial structure can be obtained in accordance with Equation 3.12 and the bit-parallel structure can be derived using Mastrovito's method. The new hybrid multiplier in $\mathbb{F}_{(2^3)^5}$ is shown

in Fig. 3.12.



(a) Bit-serial structure by Kwon et al. of $\mathbb{F}_{2^3}$

(b) Bit-parallel structure of $\mathbb{F}_{2^3}$

Figure 3.12: The architecture of a hybrid multiplier over $\mathbb{F}_{(2^3)^5}$

For the subfield architecture, it requires n^2 AND gates and n^2-1 XOR gates, and the latency of the subfield multiplier $T_{\odot}$ is upper bounded by $T_A + 2T_X\lceil log_2 n\rceil$ [54], where T_A denotes the latency of 2-input AND gate and T_X denotes the latency of 2-input XOR gate. In accordance with [31], Kwon's type II ONB bit serial architecture requires m subfield multipliers and no more than $m + \frac{m-1}{2}(\frac{3m-1}{2})$ subfield adders and the critical path latency is upper bounded by $T_{\odot} + (1+\lceil log_2 k\rceil)T_{\oplus}$, where $T_{\oplus}$ denotes the latency of the subfield adder (actually, $T_{\oplus} = T_X$). By now, we can summarize the area and timing of this hybrid multipliers as follows,

$$\#AND = mn^2$$

$$\#XOR \leq m(n^2-1) + (m + \frac{m-1}{2}(k-1)(\frac{3m-1}{2}))n$$

$$T \leq T_A + (1 + 2\lceil log_2 n\rceil + \lceil log_2 k\rceil)T_X \tag{3.15}$$

The advantages of such hybrid multipliers are:

1. Circuit complexity can be decreased considerably due to the efficient bit-serial architecture at the top level.

2. Compared with the straitforward method of parallelizing normal basis multiplier, this hybrid multiplier is more regular since the bit-parallel component has the same structure. Therefore it is easy for EDA tools to place and route.

3. The bit-parallel multiplier of ground field is very efficient in terms of timing and area due to the chosen trinomial.

Squarer

By the identity $\gamma^{2^m} = \gamma$,

$$a^2 = (\sum_{i=0}^{m-1} a_i \gamma^{2^i})^2 = \sum_{i=0}^{m-1} a_i^2 \gamma^{2^{i+1}} = (a_{m-2}^2, a_{m-3}^2, \ldots, a_0^2, a_{m-1}^2), \text{ where } a_i \in \mathbb{F}_{2^n}$$

Squaring at the top level is free and equivalent to a cyclic shift. Squaring at ground

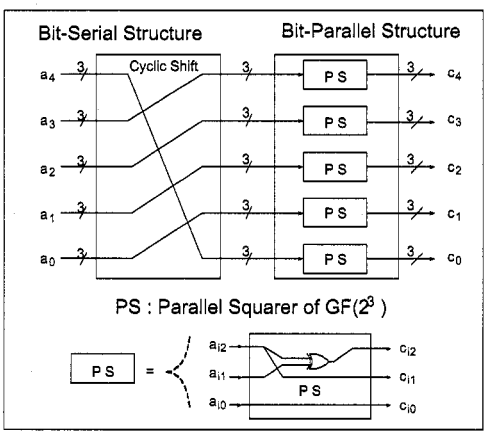


Figure 3.13: The architecture of a hybrid squarer over $\mathbb{F}_{(2^3)^5}$

level can be computed by Equation 3.2. Since the ground field is constructed via the

simple irreducible polynomial, the reduction is relatively cheap. The architecture of this hybrid squarer is illustrated with the same example of $\mathbb{F}_{(2^3)^5}$(See Fig 3.13.)

Inverter

Inversion methods based on repeated multiplication can be made more efficient for composite field $\mathbb{F}_{(2^n)^m}$ [18]. For non-zero $a \in \mathbb{F}_{(2^n)^m}$, we can write

$$a^{-1} = \frac{1}{a^r} \cdot a^{r-1} \tag{3.16}$$

where $r = \frac{2^{nm}-1}{2^n-1}$. Obviously, $A = a^r$ is in an element belonging to $\mathbb{F}_{2^n}$. Since r can be expressed as a sum of powers as follows,

$$r - 1 = 2^n + 2^{2n} + 2^{3n} + \ldots + 2^{(m-1)n} \tag{3.17}$$

a^{r-1} can be computed using the addition chain. This method requires

$$\lfloor log_2(m-1) \rfloor + H_w(m-1) - 1 \tag{3.18}$$

general multiplications and at most $m-1$ Frobenius operations to the power of 2^n, with both types of operations performed in $\mathbb{F}_{(2^n)^m}$.

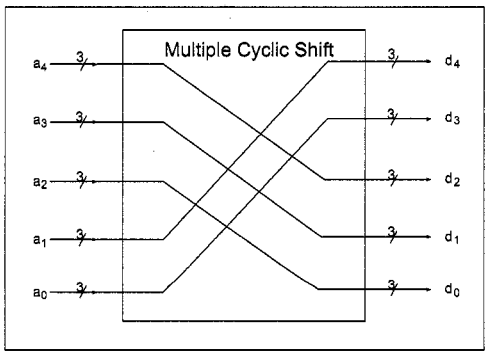


Figure 3.14: The architecture of computing a^{2^3} over $\mathbb{F}_{(2^3)^5}$

In our hybrid architecture, computation of $d = a^{2^n}$ is free and equivalent to multiple cyclic shifts as the normal basis is chosen for the tower field. Secondly, the coefficient a_i is in the subfield $\mathbb{F}_{2^n}$, i.e., $a_i^{2^n} = a_i$ (See Fig 3.14). a^r can be obtained from a^{r-1} with one more multiplication in $\mathbb{F}_{(2^n)^m}$

$$a^r = a \cdot a^{r-1} \tag{3.19}$$

Since the identity of the composite field $\mathbb{F}_{(2^n)^m}$ can be represented as,

$$1 = \sum_{i=0}^{m-1} \gamma_i \tag{3.20}$$

each ground field element, namely $d_0 \in \mathbb{F}_{2^n}$, can be represented in B_t as follows, i.e., all the coefficients are the same as each other.

$$d_0 = \sum_{i=0}^{m-1} d_0 \gamma_i \tag{3.21}$$

We need to develop the inverter for the ground field. If n is small, it is possible to use a parallel architecture. For the same example where $n = 3$, let $d_0 = \sum_{j=0}^{2} d_{0j}\beta^j$ ($d_0 \neq 0$) and $d_0' = d_0^{-1} = \sum_{j=0}^{2} d_{0j}'\beta^j$, where $d_{0j}, d_{0j}' \in \mathbb{F}_2$. By the identity $d_0 \cdot d_0' = 1$,

$$\begin{bmatrix} d_{02}+d_{00} & d_{01} & d_{02} \\ d_{02}+d_{01} & d_{02}+d_{00} & d_{01} \\ d_{01} & d_{02} & d_{00} \end{bmatrix} \cdot \begin{bmatrix} d_{02}' \\ d_{01}' \\ d_{00}' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.22}$$

After deductions, we obtain the coefficients of d'_0 as follows,

$$d'_{02} = d_{01} + d_{02} + d_{00}d_{02}$$

$$d'_{01} = d_{02} + d_{00}d_{01}$$

$$d'_{00} = d_{00} + d_{01} + d_{02} + d_{01}d_{02} \tag{3.23}$$

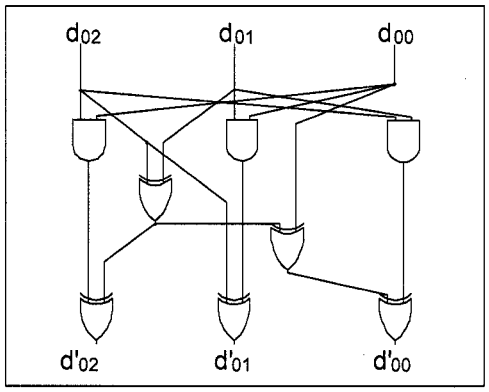


Figure 3.15: Bit-parallel inverter over $\mathbb{F}_{2^3}$

For large value of n, Itoh-Tsujj's architecture can be adopted, which requires

$$\lfloor log_2(n-1) \rfloor + H_w(n-1) - 1$$

subfield multiplications and at most $n-1$ Frobenius operations.

Now the question arises: Is it easy to perform the conversion between the hybrid basis, namely $B_h = (\gamma, \beta\gamma, \ldots, \beta^{n-1}\gamma; \gamma^2, \beta\gamma^2, \ldots, \beta^{n-1}\gamma^2; \ldots; \gamma^{2^{m-1}}, \beta\gamma^{2^{m-1}}, \ldots, \beta^{n-1}\gamma^{2^{m-1}})^T$, and the standard basis, namely $B_s = \{1, \alpha, \alpha^2, \ldots, \alpha^{nm-1}\}$. In the following section, we address the basis conversion between B_h and B_s.

Basis Conversion

We construct the ground field $\mathbb{F}_{2^n}$ using special irreducible trinomials, namely $f_g(x) = x^{2^g} + x + 1$ or $f_g(x) = x^{2^g-1} + x^{2^t-1} + 1$, where $n = 2^g$ or $n = 2^g - 1$. The trinomials, listed in Table 3.8., can satisfy most requirements of digit-size for parallel computation.

Table 3.8: Special irreducible trinomials for constructing the ground field

Field Size n	trinomials	Field Size n	trinomials
2	$x^2 + x + 1$	15	$x^{15} + x + 1$
3	$x^3 + x + 1$	31	$x^{31} + x^3 + 1$
4	$x^4 + x + 1$	63	$x^{63} + x + 1$
7	$x^7 + x + 1$	127	$x^{127} + x + 1$

Let B_s denote a polynomial basis of $\mathbb{F}_{2^{nm}}$ with respect to $\mathbb{F}_2$, generated by α, which is the root of the generating polynomial $f_s(x)$, and B_g denote the polynomial basis of the ground field $\mathbb{F}_{2^n}$ with respect of $\mathbb{F}_2$, generated by β which is the root of $f_g(x)$. Our task is to find the relation between β (the generator of B_g) and α (the generator of B_s). Since $\mathbb{F}_{2^n}$ is a subfield of $\mathbb{F}_{2^{nm}}$, β can be represented via B_s.

$$\beta = \sum_{i=0}^{nm-1} a_i \alpha^i, \ \textit{where} \ a_i \in \mathbb{F}_2 \tag{3.24}$$

By $f_g(\beta) = 0$, we have

$$\beta^{2^g} + \beta + 1 = 0 \ \textit{or} \ \beta^{2^g-1} + \beta^{2^t-1} + 1 = 0 \tag{3.25}$$

For the latter, multiplying β on both sides, we can get $\beta^{2^g} + \beta^{2^t} + \beta = 0$. Since the formulae of square is easily obtained by the generating polynomial of $\mathbb{F}_{2^{nm}}$ and the property of the binary field, we can perform multiple squares of β to obtain β^{2^g}

and β^{2^t} whose coefficients with respect to α^i are actually the linear combination of $\{a_0, a_1, \ldots, a_{nm-1}\}$, i.e., nm linear equations of a_i can be derived. Actually there are n possible solutions corresponding to the conjugate class of β or $n+1$ solutions including the trivial one, $\beta = 0$.

Algorithm 6 Computing the coefficients of β^{2^g}

Require: The linear functions, $coef_i(\overrightarrow{a})$

Ensure: The coefficients of β^{2^g} stored in $\overrightarrow{a}$.

1: **for** $i = 0$ to $nm - 1$ **do**
2: $\quad \widehat{a_i} \leftarrow 00, \ldots, 01 \ll i$;
3: **end for**
4: **for** $j = 1$ to g **do**
5: $\quad$ **for** $i = 0$ to $nm - 1$ **do**
6: $\quad\quad \widehat{b_i} \leftarrow coef_i(\overrightarrow{a})$;
7: $\quad$ **end for**
8: $\quad \overrightarrow{a} \leftarrow \overrightarrow{b}$
9: **end for**
10: return $\overrightarrow{a}$;

Algorithm 6. is used to derive the linear equations. First we clarify some notations. Let $\overrightarrow{a} = \{\widehat{a_0}, \widehat{a_1}, \ldots, \widehat{a_{nm-1}}\}$, where $\widehat{a_i}$, a nm-bit string, is used to store the ith-coefficient of multiple square of β, e.g., $nm = 6$, $\widehat{a_2} = 010010$ means that the coefficient with respect to α^2 is $a_4 + a_1$. $coef_i(\overrightarrow{a})$ denotes the function to compute the ith-coefficient of the square of the element represented by $\overrightarrow{a}$ and B_s. If $(\beta^2)_3 = a_5 + a_3$, where $(\beta^2)_3$ denotes the coefficient of β^2 with respect to α^3, then $coef_3(\overrightarrow{a}) = \widehat{a_5} + \widehat{a_3}$.

The first loop of Algorithm 6. is to initialize $\overrightarrow{a}$ as β by Equation 3.24. The inner loop is to compute the coefficients for each square recursively. This algorithm can be easily programmed via bit-orientation language. Therefore we can get the linear equations by Equation 3.25. The coefficients of β with respect to B_s can be computed. We will illustrate this process in detail via the following example.

Let $f_g(x) = x^3 + x + 1$ and $f_s(x) = x^{15} + x + 1$. Let $\beta = \sum_{i=0}^{14} a_i \alpha^i$, where $a_i \in \{0, 1\}$. We need to compute $\{a_i\}$.

Step 1: Derive the formulae computing the coefficients of β^2.

Since $\beta^2 = \sum_{i=0}^{14} a_i \alpha^{2i}$, by replacing $\alpha^{15} = \alpha + 1$, we can obtain

$$
\begin{array}{lll}
(\beta^2)_{14} = a_{14} + a_7 & (\beta^2)_{13} = a_{14} & (\beta^2)_{12} = a_{13} + a_6 \\
(\beta^2)_{11} = a_{13} & (\beta^2)_{10} = a_{12} + a_5 & (\beta^2)_9 = a_{12} \\
(\beta^2)_8 = a_{11} + a_4 & (\beta^2)_7 = a_{11} & (\beta^2)_6 = a_{10} + a_3 \\
(\beta^2)_5 = a_{10} & (\beta^2)_4 = a_9 + a_2 & (\beta^2)_3 = a_9 \\
(\beta^2)_2 = a_8 + a_1 & (\beta^2)_1 = a_8 & (\beta^2)_0 = a_0
\end{array} \tag{3.26}
$$

Correspondingly, we can get $coef_i(\overrightarrow{a})$.

Step 2: Compute β^{2^2} According to Algorithm 6., we can compute β^{2^2} recursively

by Equation (3.26).

$$
\begin{array}{lll}
(\beta^4)_{14} = a_{14} + a_{11} + a_7 & (\beta^4)_{13} = a_{14} + a_7 & (\beta^4)_{12} = a_{14} + a_{10} + a_3 \\
(\beta^4)_{11} = a_{14} & (\beta^4)_{10} = a_{13} + a_{10} + a_6 & (\beta^4)_9 = a_{13} + a_6 \\
(\beta^4)_8 = a_{13} + a_9 + a_2 & (\beta^4)_7 = a_{13} & (\beta^4)_6 = a_{12} + a_9 + a_5 \\
(\beta^4)_5 = a_{12} + a_5 & (\beta^4)_4 = a_{12} + a_8 + a_1 & (\beta^4)_3 = a_{12} \\
(\beta^4)_2 = a_{11} + a_8 + a_4 & (\beta^4)_1 = a_{11} + a_4 & (\beta^2)_0 = a_0
\end{array} \tag{3.27}
$$

Step 3: Solve the linear equations. By $\beta \cdot f_g(\beta) = \beta^4 + \beta^2 + \beta$, we can get the linear equations of $\{a_i\}$ as follows.

$$
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\cdot
\begin{bmatrix} a_{14} \\ a_{13} \\ a_{12} \\ a_{11} \\ a_{10} \\ a_9 \\ a_8 \\ a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\tag{3.28}
$$

After deductions, we can obtain 4 solutions in which $\beta = 0$ should be neglected. And the remaining 3 solutions actually represent the three elements of the conjugate class of β.

Without loss of generality,

$$\begin{bmatrix} \beta \\ \beta^2 \\ \beta^{2^2} \end{bmatrix} = \begin{bmatrix} \alpha^9+\alpha^8+\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2 \\ \alpha^{12}+\alpha^{10}+\alpha^8+\alpha^6+\alpha^3+\alpha^2+\alpha \\ \alpha^{12}+\alpha^{10}+\alpha^9+\alpha^5+\alpha^4+\alpha \end{bmatrix} \tag{3.29}$$

And,

$$B_g = \begin{bmatrix} 1 \\ \beta \\ \beta^2 \end{bmatrix} = \begin{bmatrix} 1 \\ \alpha^9+\alpha^8+\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2 \\ \alpha^{12}+\alpha^{10}+\alpha^8+\alpha^6+\alpha^3+\alpha^2+\alpha \end{bmatrix} \tag{3.30}$$

Therefore, we get the conversion matrix Λ_g such that $B_g = \Lambda_g \cdot B_s$,

$$\Lambda_g = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3.31}$$

We introduced a method to construct the ground field for both efficient arithmetic and conversion via a special kind of trinomial. The process is equivalent to solving a set of linear equations. Especially, we presented an algorithm which can be easily programmed to obtain such linear equations, and this algorithm will be also adopted in the following context for conversions between B_t and B_s.

We need to find the representation of the normal element γ of $\mathbb{F}_{2^m}$ with B_s, i.e., to obtain the coefficients $b_i \in \mathbb{F}_2$ such that $\gamma = \sum_{i=0}^{m-1} b_i \alpha^i$, where $b_i \in \mathbb{F}_2$. There are two methods to solve this problem. Both approaches will be illustrated with the same example used previously in which the composite field is $\mathbb{F}_{(2^3)^5}$. Although α and γ are not necessarily primitives elements of their corresponding fields, we suppose that both are primitive in our approach.

The first approach is to get the knowledge of $\gamma^t = \alpha^s$, where $1 \leq t \leq 2^m - 2$

and $s = \frac{2^{mn}-1}{2^m-1}$, so that the representation of γ with B_s can be easily computed by some field operations such as multiplications and inversions within the composite field $\mathbb{F}_{2^{nm}}$.

We can compute the minimal polynomial of γ with respect to $\mathbb{F}_2$.

$$m_\gamma(x) = (x+\gamma)(x+\gamma^2)(x+\gamma^{2^2})(x+\gamma^{2^3})(x+\gamma^{2^4}) \tag{3.32}$$

By Equation 3.9,

$$m_\gamma(x) = x^5 + x^4 + x^2 + x + 1 \tag{3.33}$$

If we use this equation, i.e., $m_\gamma(\gamma) = 0$ and replacing γ with $\sum_{i=0}^{14} b_i\alpha^i$ directly, we will get some non-linear equations of $\{b_i\}$. Solving these equations will be complicated. Alternatively, we find that $m_\gamma(x)$ is bijectively mapped to the coset $\{1, 2, 4, 8, 16\}$ of $\mathbb{Z}/31\mathbb{Z}^*$. Similarly we can get the other minimal polynomials of the other powers of γ, see Table 3.9.

Table 3.9: Minimal polynomials of the powers of γ over $\mathbb{F}_{2^5}$

Powers of γ	Minimal polynomials	Cosets
γ	$m_\gamma(x) = x^5 + x^4 + x^2 + x + 1$	$\{1, 2, 4, 8, 16\}$
γ^3	$m_{\gamma^3}(x) = x^5 + x^3 + 1$	$\{3, 6, 12, 24, 17\}$
γ^5	$m_{\gamma^5}(x) = x^5 + x^3 + x^2 + x + 1$	$\{5, 10, 20, 9, 18\}$
γ^7	$m_{\gamma^7}(x) = x^5 + x^2 + 1$	$\{7, 14, 28, 25, 19\}$
γ^{11}	$m_{\gamma^{11}}(x) = x^5 + x^4 + x^3 + x^2 + 1$	$\{11, 22, 13, 26, 21\}$
γ^{15}	$m_{\gamma^{15}}(x) = x^5 + x^4 + x^3 + x + 1$	$\{15, 30, 29, 27, 31\}$

The second step is to compute α^s, where $s = 1 + 2^5 + 2^{10}$ and the minimal polynomial of α is $m_\alpha(x) = x^{15} + x + 1$ ($\alpha^{15} + \alpha + 1 = 0$). It requires 10 squarings and 2 multiplications.

$$\gamma^t = \alpha^9 + \alpha^6 + \alpha^5 + \alpha^4 \tag{3.34}$$

By this equation, we can compute the minimal polynomial of γ_t as follows,

$$\begin{aligned} m_{\gamma^t(x)} &= (x+\gamma^t)(x+\gamma^{2t})(x+\gamma^{4t})(x+\gamma^{8t})(x+\gamma^{16t}) \\ &= x^5+x^3+x^2+x+1 \end{aligned} \tag{3.35}$$

In accordance with Tab 3.9., t locates in the coset, $\{5, 10, 20, 9, 18\}$. Without loss of generality, let $t = 9$ considering that both primitive element α and Gauss period γ are determined up to conjugate classes. Then we can get

$$\begin{bmatrix} \gamma^5 \\ \gamma^{10} \\ \gamma^{20} \\ \gamma^9 \\ \gamma^{18} \end{bmatrix} = \begin{bmatrix} \alpha^{10}+\alpha^9+\alpha^8+\alpha^5+\alpha^2+\alpha \\ \alpha^{10}+\alpha^6+\alpha^5+\alpha^3+\alpha \\ \alpha^{12}+\alpha^{10}+\alpha^5+\alpha^2 \\ \alpha^9+\alpha^6+\alpha^5+\alpha^4 \\ \alpha^{12}+\alpha^{10}+\alpha^8+\alpha^4+\alpha^3 \end{bmatrix} \tag{3.36}$$

By identity,

$$\gamma = \frac{\gamma^{10}}{\gamma^9} = \alpha^{12}+\alpha^8+\alpha^6+\alpha^5+\alpha^4+\alpha+1 \tag{3.37}$$

$$B_t = \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} = \begin{bmatrix} \alpha^{12}+\alpha^8+\alpha^6+\alpha^5+\alpha^4+\alpha+1 \\ \alpha^{12}+\alpha^9+\alpha^8+\alpha+1 \\ \alpha^{10}+\alpha^9+\alpha^4+\alpha^3+\alpha+1 \\ \alpha^8+\alpha^5+\alpha^4+\alpha^3+\alpha^2+1 \\ \alpha^{10}+\alpha^8+\alpha^6+\alpha^4+\alpha^2+\alpha+1 \end{bmatrix} \tag{3.38}$$

Therefore, we get the conversion matrix Λ_t such that $B_t = \Lambda_t \cdot B_s$,

$$\Lambda_t = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{3.39}$$

The second approach is from Fermat's theorem,

$$\gamma^{2^m} = \gamma \tag{3.40}$$

Applying Algorithm 6, we can obtain a set of linear equations where there exist 2^m possible solutions. Since m is relative small compared with nm, it is possible to use exhaustive search method to find the representations of the conjugate class of γ according to Equation 3.9. The computational complexity is $\mathcal{O}(\frac{\Phi(2^m-1)}{m})$, see [55]. More details of this method can be found in the following example.

By Equation 3.40. and Algorithm 6., we have

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} b_{14} \\ b_{13} \\ b_{12} \\ b_{11} \\ b_{10} \\ b_9 \\ b_8 \\ b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.41}$$

After deductions,

$$\begin{aligned} b_{14} &= b_{13} = b_{11} = b_7 = 0 & b_{12} &= b_3 + b_2 + b_1 \\ b_{10} &= b_3 + b_2 & b_9 &= b_4 + b_3 + b_1 \\ b_8 &= b_3 + b_1 & b_6 &= b_4 + b_3 \\ b_5 &= b_4 + b_3 + b_2 \end{aligned} \tag{3.42}$$

If b_0, b_1, b_2, b_3 and b_4 are given, the other coefficients of γ can be decided, i.e., totally 32 solutions satisfying Equation 3.40. Only 5 of them are the expected ones. By Equation 3.9. and exhaustive search we can get the same results as Equation 3.39. Combining with previous results, we can get the conversion matrix Λ between B_h and B_s as follows.

$$\begin{bmatrix} \gamma_0 \\ \gamma_0\beta \\ \gamma_0\beta^2 \\ \gamma_1 \\ \gamma_1\beta \\ \gamma_1\beta^2 \\ \gamma_2 \\ \gamma_2\beta \\ \gamma_2\beta^2 \\ \gamma_3 \\ \gamma_3\beta \\ \gamma_3\beta^2 \\ \gamma_4 \\ \gamma_4\beta \\ \gamma_4\beta^2 \end{bmatrix} = \begin{bmatrix} 0&0&1&0&0&0&1&0&1&1&1&0&0&1&1 \\ 0&1&1&1&1&1&0&0&1&1&1&1&1&0&0 \\ 0&1&1&1&0&0&0&1&1&1&0&0&0&0&0 \\ 0&0&1&0&0&1&1&0&0&0&0&0&0&1&1 \\ 1&0&0&0&0&0&1&0&0&1&0&1&0&0&0 \\ 0&0&0&1&0&1&0&1&0&1&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&1&1&0&1&1 \\ 0&1&0&0&0&0&1&1&0&0&1&1&1&1&0 \\ 1&1&0&0&1&0&0&0&1&0&0&0&1&1&0 \\ 0&0&0&0&0&0&1&0&0&1&1&1&1&0&1 \\ 0&1&1&0&0&0&1&0&0&1&0&0&1&1&0 \\ 1&0&1&1&0&0&1&0&1&0&1&0&0&1&0 \\ 0&0&0&0&1&0&1&0&1&0&1&0&1&1&1 \\ 1&1&0&1&1&0&0&1&0&0&1&0&0&0&0 \\ 0&0&1&1&0&1&0&0&0&0&1&0&0&1&0 \end{bmatrix} \cdot \begin{bmatrix} \alpha^{14} \\ \alpha^{13} \\ \alpha^{12} \\ \alpha^{11} \\ \alpha^{10} \\ \alpha^{9} \\ \alpha^{8} \\ \alpha^{7} \\ \alpha^{6} \\ \alpha^{5} \\ \alpha^{4} \\ \alpha^{3} \\ \alpha^{2} \\ \alpha^{1} \\ 1 \end{bmatrix} \tag{3.43}$$

By now, we have shown how to perform basis conversion between the hybrid basis and the standard polynomial basis representation by the property of several special trinomials.

3.5 Summary

This chapter addresses the issues of efficient hardware design of binary field arithmetic including squaring, multiplication and multiplicative inversion in different basis representations. For polynomial basis, we compare two traditional algorithms for bit/digit-serial multipliers, left-to-right and right-to-left, in terms of performance and cost. The techniques of shortening the critical path and decreasing the circuit complexity for bit-serial normal basis multipliers are also covered.

We compare the hybrid digit-serial multipliers with the conventional digit-serial multipliers for several special composite fields constructed via low Hamming weight irreducible polynomials in terms of gate count and latency. We port both designs to the same FPGA device. According to the experimental results, we demonstrate that the hybrid architecture is more efficient in terms of timing and area. In particular, its regularity and wire density make it more suitable for FPGA realizations than the conventional architecture.

Finally, a novel hybrid multiplier, which is not only efficient in arithmetic but also convenient for basis conversion, is proposed. The bit-serial normal basis structure proposed by Kwon et al. is applied to the tower field arithmetic. The irreducible trinomials with special forms are chosen to construct the ground field such that bit-parallel multipliers can be realized efficiently and at the same time the process of performing basis conversion is equivalent to solving a set of linear equations.

Chapter 4: Optimizations of ECC Processor for a Single FPGA device

4.1 Introduction

In this chapter, we focus on the efficient FPGA realization of the dominant computations in elliptic curve cryptography, namely scalar multiplication or point multiplication $[k]P$. We are interested in non-supersingluar elliptic curves over binary fields defined in Equation 2.8. The basic method for computing $[k]P$ is the well known "add-and-double" (or binary) method, and the add-and-subtract method is the improved version, which requires m point doublings and $\frac{m}{3}$ point additions on average [26]. If some extra memory is available, this algorithm can be speed up by using a window method, which process w digits of k at a time. In case that P is a fixed point, for example in ECDSA signature generation, scalar multiplication algorithms can exploit precomputed data that depends only on P. For Koblitz curves, the Frobenious map can be exploited to simplify the group operations by writing the integer k in the form $k = \sum_{i=0}^{l-1} k_i \tau^i$ where each $k_i \in \mathbb{F}_2$ and τ is defined in Equation 2.13. Even though the point doubler can be replaced with a simpler circuit, the complicated conversion method for k make it difficult for realization in hardware. Additionally, the above algorithms can not be directly applied to the public key cryptosystems due to their weakness against side-channel attack. López and Dahab [41] proposed a fast algorithm of point multiplication over $\mathbb{F}_{2^m}$ without precomputation based on Montgomery's ladder method. One advantage of using this algorithm is

that fewer field multiplications will be involved on average than in the traditional addition-substraction method. Secondly, since projective instead of affine coordinates are adopted, inversion is performed only once at the coordinate transformation step. This algorithm is secure against side-channel attacks. We adopt it for our ECC architecture.

There have been several ECC processors published, for instances [56–59, 63]. One approach among of them is that the group operations are scheduled by a general purpose processor (or microprocessor) and the field arithmetic is realized in hardware. Alternatively, both the group arithmetic and the field arithmetic are implemented in hardware. The second architecture is suitable for those applications in which the optimization goal is low latency since the overhead of communications between the microprocessor and arithmetic logic unit can be eliminated. Therefore we adopt the second approach in our ECC processor. The arithmetic logic units in those published processors contain only one field multiplier so that the intensive multiplications are executed sequentially. In our work, we adopt multiple multipliers for both the add-and-double stage and coordinate transformation stage to achieve high operation speed. Additionally, the optimal choices of the digit-sizes of multipliers are considered for further optimizing the processor in terms of area and timing.

We port our design to the same FPGA device as the one used by Gura et al. [63] considering that they also adopt López-Dahab algorithm for their ECC processors. Two underlying fields, $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$ recommended by NIST standard [17] are selected for our experiments. Comparisons between our ECC processors and Gura et al.'s in terms of performance and cost are demonstrated. Additionally we choose a larger target device for our ECC processor and compare the latency for one scalar multiplications with the one from the software LiDIA [71].

4.2 López-Dahab Algorithm

López-Dahab algorithm is an efficient implementation of Montgomery's ladder method for computing $[k]P$ on non-supersingular elliptic curves over $\mathbb{F}_{2^m}$ which is defined to be the set of solutions $(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$ to the equation, $y^2 + xy = x^3 + ax^2 + b$, where a and $b \in \mathbb{F}_{2^m}, b \neq 0$, together with the point at infinity denoted by ∞. Let P_1 and P_2 be two points on the curve. The original binary method is shown as follows.

Algorithm 7 Binary method

Require: An integer $k > 0$ and a point $P = (x, y)$.

Ensure: $Q = kP$.

1: Set $k \leftarrow (k_{l-1}, ..., k_1, k_0)_2$.
2: $P_1 \leftarrow P$, $P_2 \leftarrow 2P$.
3: **for** $i = l - 2$ downto 0 **do**
4: **if** $k_i = 1$ **then**
5: Set $P_1 \leftarrow P_1 + P_2$, $P_2 \leftarrow 2P_2$.
6: **else**
7: Set $P_2 \leftarrow P_2 + P_1$, $P_1 \leftarrow 2P_1$.
8: **end if**
9: **end for**

The x-coordinate of $P_1 + P_2$ can be computed as follows.

$$x_3 = \frac{x_1 y_2 + x_2 y_1 + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \tag{4.1}$$

It follows that $y_1^2 + y_2^2 + x_1 y_1 + x_2 y_2 + x_1^3 + x_2^3 = 0$ as P_1 and P_2 are elliptic points. Algorithm 7 maintains the invariant relationship $P = (x, y) = P_2 + (-P_1)$. By

$-P_1 = (x_1, x_1 + y_1)$ and Equation 4.1,

$$x = \frac{x_1 y_2 + x_2(x_1 + y_1) + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \tag{4.2}$$

Adding Equations 4.1 and 4.2, together with the doubling formula in Equation 2.11, we can get the following results:

$$x_3 = \begin{cases} x + (\frac{x_1}{x_1+x_2})^2 + \frac{x_1}{x_1+x_2}, & P_1 \neq P_2 \\ x_1^2 + \frac{b}{x_1^2}, & P_1 = P_2. \end{cases} \tag{4.3}$$

The y-coordinate of P_1 can be computed when P and the x-coordinates of P_1 and $P_1 + P$ are known. More details of the proof can be found in [41]. Therefore, the computations of y-coordinates of P_1 and P_2 in the main loop of Algorithm 7 can be skipped. By representing the x-coordinates of P_i by X_i/Z_i for $i \in \{1, 2\}$, we can get the projective version of Equation 4.3.

$$\begin{cases} x(2P_i) = X_i^4 + b \cdot Z_i^4, \\ z(2P_i) = Z_i^2 \cdot X_i^2. \end{cases} \tag{4.4}$$

$$\begin{cases} Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2, \\ X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1). \end{cases} \tag{4.5}$$

A method based on these formulas is described in the next algorithm. $Madd(X_1, Z_1, X_2, Z_2)$ is the procedure for point addition and the results will be stored in the variables X_1 and Z_1. $Mdouble(X_1, Z_1)$ is the procedure for point doubling and the results will be stored in the variables X_1 and Z_1. Mxy is the procedure for coordinate conversion. More details of these three procedures can be found in [41]. Algorithm 8 performs exactly the following number of field operations in $\mathbb{F}_{2^m}$.

$$\#INV = 1, \qquad \#MULT = 6\lfloor log_2 k \rfloor + 10,$$

$$\#ADD = 3\lfloor log_2 k \rfloor + 7, \qquad \#SQR = 5\lfloor log_2 k \rfloor + 3.$$

Algorithm 8 Projective version of Montgomery scalar multiplication

Require: An integer $k > 0$ and a point $P = (x, y)$.

Ensure: $Q = kP$.

1: **if** $k = 0$ or $x = 0$ **then**
2: output $(0, 0)$ and stop.
3: **end if**
4: Set $k \leftarrow (k_{l-1}, ..., k_1, k_0)$.
5: Set $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$.
6: **for** $i = l - 2$ downto 0 **do**
7: **if** $k_i = 1$ **then**
8: $Madd(X_1, Z_1, X_2, Z_2)$, $Mdouble(X_2, Z_2)$.
9: **else**
10: $Madd(X_2, Z_2, X_1, Z_1)$, $Mdouble(X_1, Z_1)$.
11: **end if**
12: **end for**
13: return $(Q = Mxy(X_1, Z_1, X_2, Z_2))$.

4.3 Hardware Implementations

In this section, we focus on optimizations of ECC processor on a single FPGA device. To obtain a high efficient design, one must consider the issues including top architecture, parallelism & sharing resources and efficient realization of underlying

field arithmetic. Two underlying fields, $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$ constructed via $f_{163}(x) = x^{163} + x^7 + x^6 + x^3 + 1$ and $f_{233}(x) = x^{233} + x^{74} + 1$, are chosen for our experiments so that the standard reduction technique can be exploited to design a high efficient digit-serial multiplier shown in Fig. 3.5. The most attractive advantage of using Algorithm 8 is that fewer multiplications in $\mathbb{F}_{2^m}$ are involved. Moreover, in each add-and-double iteration, both point addition and point doubling can be computed in parallel.

4.3.1 Top Architecture

There are basically two kinds of structures. The first one is the traditional stored-program machine (SPM) which contains three functional units: a processor, a controller and memory [64]. The processor includes registers, datapaths, control lines and ALU. The controller should be capable of steering data to the proper destination according to the instruction. The memory is used to store instructions and data. To adopt such an architecture, the designer needs to develop ALU according to the operations necessary for ECC, and accordingly build the instruction set. Since the intermediate operands of ECC are data-dependent most of the field operations can not be completed in a single or small number of clock cycles. Therefore these computations are not suitable to be pipelined. Moreover in programmed-directed operations, instructions are synchronously fetched, decoded and executed, which will deter the operation speed of the ECC accelerator because of the overhead of communication between the memory and the processor. Alternatively, the processor can be constructed via a main controller, interconnection networks, register files and ALU. The controller may be designed as a finite state machine (FSM) scheduling operations for ALU. The intermediate results will be stored in the register files in order to eliminate the overhead of communication between memory and ALU. We adopt the second

architecture for our ECC processor.

4.3.2 Parallel Computations

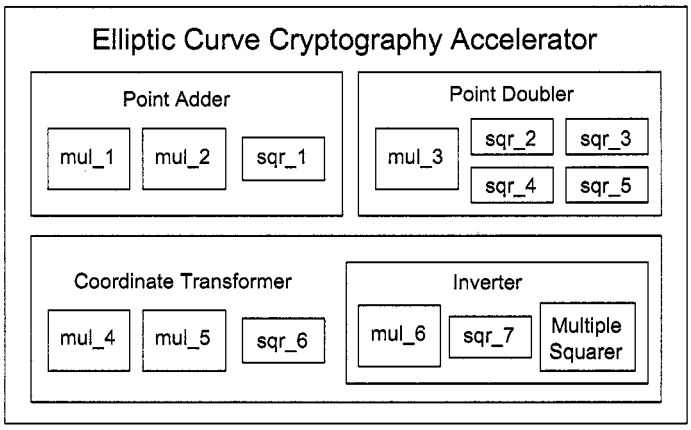


Figure 4.1: The diagram of the ECC accelerator.

One advantage of hardware implementation is that it supports parallel computations to provide high operation speed as long as multiple operations can be performed at the same time. Field multiplication is the most intensive arithmetic operation necessary for scalar multiplication. Since our optimization goal is low latency, field multipliers are not shared among point adder, point doubler, and coordinate transformer to avoid complicated data-path that may have negative impact on timing and routing. According to Equations 4.4 and 4.5, the computations for each add-and-double iteration can be completed in two multiplication rounds in case that two multipliers are used in the point adder and one multiplier is used in the point doubler. Two additional multipliers and one inverter are used in the coordinate transformer, and the multiplier inside the inverter is not shared outside. If only one multiplier is adopted in the converter, then multiple levels of multiplexers can not be avoided, see Fig. 4.1.

4.3.3 Optimal Choice of Digit Size of Multipliers

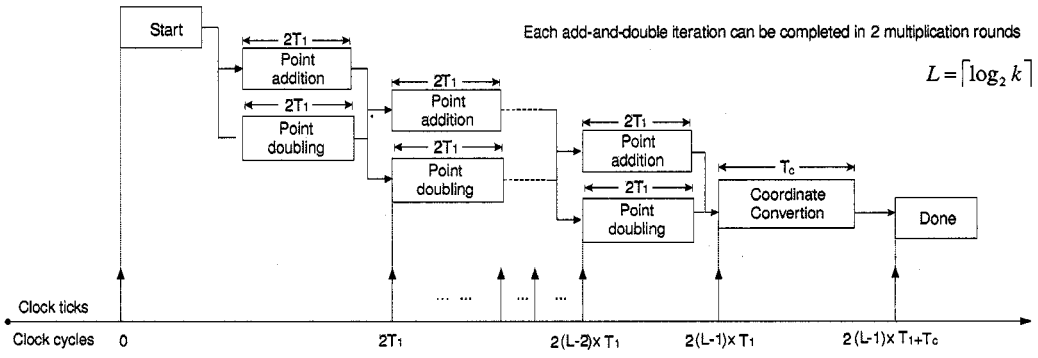


Figure 4.2: The timing diagram of scalar multiplication.

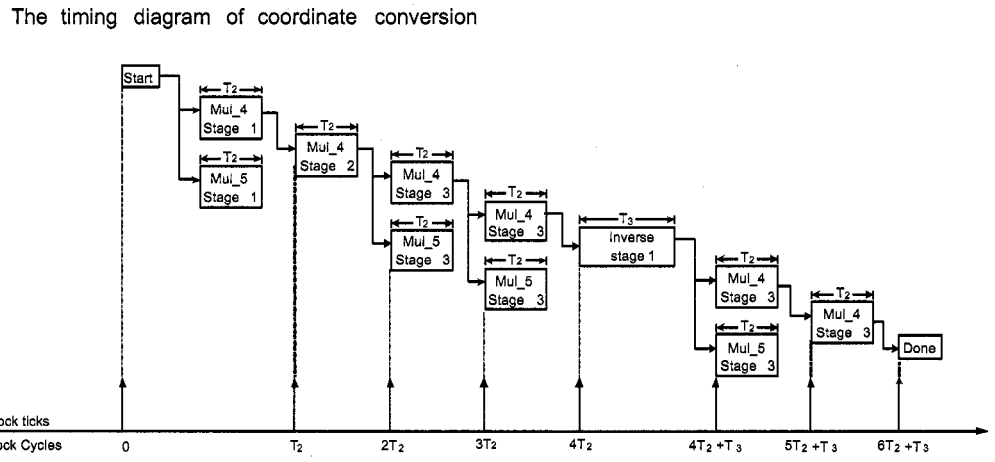


Figure 4.3: The timing diagram of coordinate conversion.

Since most computations are performed in the first stage (see Fig. 4.2), add-and-double stage, the multipliers, *mul*_1, *mul*_2 and *mul*_3, should have large digit size D_1 to achieve high operation speed. On the other hand, in order to decrease the resource utilization without loss of performance, the multipliers, working only at the final exponentiation stage (see Fig. 4.3), can be slow with small digit size D_2. For the exponentiation performed as a part of inversion, we adopt the component which can compute a^{2^4} each clock cycle so that it takes around $\frac{m}{4}$ cycles for powering. Let T_1

Table 4.1: Digit sizes of multipliers (for the target device, Xilinx XC2V6000)

	Digit size D					
	mul_1	mul_2	mul_3	mul_4	mul_5	mul_6
$\mathbb{F}_{2^{163}}$	32	32	32	8	8	8
$\mathbb{F}_{2^{233}}$	32	32	32	8	8	8
$\mathbb{F}_{2^{283}}$	16	16	16	4	4	4

denote the latency of one multiplication completed by *mul*_1, *mul*_2 or *mul*_3. Let T_2 denote the latency of one multiplication completed by *mul*_4, *mul*_5 and *mul*_6. Let T_3 denote the latency of one inversion completed by the inverter. Let T_c denote the latency of the coordinate conversion. Then we have the following equations estimating the total latency of one scalar multiplication denoted by Δ.

$$T_1 = \lceil \frac{m}{D_1} \rceil$$

$$T_2 = \lceil \frac{m}{D_2} \rceil$$

$$T_3 = T_2(\lfloor log_2(m-1) \rfloor + HW(m-1) - 1) + \frac{m}{4}$$

$$T_c = 6T_2 + T_3$$

$$\Delta = 2(\lceil log_2 k \rceil - 1) \times T_1 + T_c \tag{4.6}$$

For convenience of understanding, let $m = 233$. We assume that the clock period is fixed. If we choose $D_1 = D_2$ then the time spent on the multiplications of the coordinate conversion stage is 1.3% of the time spent on the multiplications of the add-and-double stage. If we choose $D_2 = \frac{1}{4}D_1$, then the percentage is 5.2%. If we choose $D_2 = \frac{1}{16}D_1$, then the percentage is 21%. By trading off the latency and area, we choose $D_2 = \frac{1}{4}D_1$. The value of D_1 primarily depends on the available

resources and the capability of the EDA tool for placing and routing. The digit sizes of multipliers of ECC accelerator over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$ are listed in Table 4.1.

4.3.4 Results

Table 4.2: Timing, resource utilization and performance comparisons with LiDIA

Hardware, Xilinx XC2V6000					Software, LiDIA	
Fields	FFs	LUTs	Clock Period (ns)	FPGA Latency per kP (μs)	LiDIA Latency per kP (μs)	Speedup vs. LiDIA
$\mathbb{F}_{2^{163}}$	10,918	26,999	9.971	33	5290	159
$\mathbb{F}_{2^{233}}$	14,215	36,582	9.949	57	9680	171
$\mathbb{F}_{2^{283}}$	24,664	35,196	11.096	146	14060	96

The results of timing, resource utilization and speed comparisons with the software implementation based on LiDIA [71] are summarized in Table 4.2. The target FPGA device was Xilinx XC2V6000, software implementation was run under Linux on a Dell PowerEdge-2800 workstation containing four 2.8GHz Intel Xeon CPUs and 4Gbyte RAM. Performance comparisons with the accelerators developed by Gura et al. [63] are provided in Table 4.3, for which both designs are ported into the same FPGA device, Xilinx XCV2000E.

Table 4.3: Performance comparisons with Gura et al's results

	Gura et al.		Our design ($D_1 = 16$)		Our design ($D_1 = 32$)	
	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$
FFs	6,442	NA	7,425	10,474	7,467	10,637
LUTs	19,508	NA	18,749	25,838	25,768	35,800
Frequency (MHz)	66.5	66.5	69.6	65.2	62.7	60.3
Latency (μs)	143	225	75	144	53	100

According to Table 4.2, less LUTs are used in the ECC accelerator over $\mathbb{F}_{2^{283}}$ than over $\mathbb{F}_{2^{233}}$ because of the smaller digit size of multipliers. The reason that we adopt smaller digit size of multipliers over $\mathbb{F}_{2^{283}}$ is to avoid timing and routing problems due to larger field degree. The work frequency of ECC accelerators is around 100 MHz, and it takes only 33 μs to complete one scalar multiplication over $\mathbb{F}_{2^{163}}$, 57 μs for $\mathbb{F}_{2^{233}}$ and 146 μs for $\mathbb{F}_{2^{283}}$. Moreover 100-to-170 speedups can be gained in our ECC accelerators via FPGAs over software implementation, LiDIA. In Table 4.3, our accelerator ($D_1 = 16$) for $\mathbb{F}_{2^{163}}$ can run twice as fast as the accelerator designed by Gura et al., whereas the resource utilizations are almost the same for both designs. In general, 1.5-to-3 speedups can be gained in our accelerator over Gura et al.

4.4 Summary

Our accelerator can run 1.5-to-3 times as fast as the accelerator designed by Gura et al. with the same resource utilization. Fast speed can be achieved due to efficient field arithmetic, top-level algorithm and rational partition of the design. Finally, the best choice of the digit sizes of multipliers can further optimize our design.

Chapter 5: Reconfigurable Computing Approach for Elliptic Curve Cryptosystems over Binary Fields

5.1 Introduction

Reconfigurable computers, based on a hybrid architecture combining a reconfigurable hardware processing unit (such as FPGAs) with a software-programmable processor, are devised to fill the gap between the hardwired technology and the general-purpose processor. These allow computer users to customize the reconfigurable processing unit to complete computation-intensive tasks without loss of flexibility of a software solution. FPGAs are commonly chosen to be integrated into the system as reconfigurable components. To make reconfigurable computers available to computer programmers and scientists, it is expected that programming should not require any knowledge of hardware design, assuming that sufficiently large library of elementary operations has been earlier developed. As a result, it is desirable that hardware functions executed in FPGAs could be described using both high-level programming languages (HLLs), such as C and Fortran, as well as hardware description languages (HDLs), such as VHDL and Verilog. While HDLs provide enough precision and power to describe library components optimized for maximum speed and/or minimum area, HLLs offer ease of programming and a short time to the solution. Although reconfigurable computers are intended to be used in the future as general-purpose machines, their current implementations are particularly suitable for computationally intensive

problems with modest input/output requirements. For such problems, the significant performance advantage over traditional von Neumann machines is accomplished using

- Parallelism that matches each algorithm's parallelism.
- Computational resources that match the algorithm's requirements.
- Elimination of instruction processing overhead.
- Elimination of the load/store overhead.

Elliptic curve cryptography is particularly suitable for implementation on reconfigurable computers because of the need for computationally intensive operations with unconventionally long operands. As a platform for our experiments, we have chosen one of the first general-purpose, stand-alone reconfigurable computers available on the market, SRC-6 [76]. As shown in Fig. 1, each function executed on the SRC-6 reconfigurable computer can be implemented using three different approaches:

1. As an HLL function running on a traditional microprocessor.
2. As an HLL function running on an FPGA.
3. As an HDL macro running on an FPGA.

Consequently, any program developed for execution on SRC-6 needs to be partitioned taking into account two independent boundaries, the first between the execution on a microprocessor vs. execution on an FPGA system; and the second between the program entry in HLL vs. program entry in HDL.

The scalar multiplication in elliptic curve cryptosystems is particularly suitable for our experiments, as it involves a four-level hierarchy of functions of various complexity. In this work, we first investigate several different schemes for partitioning

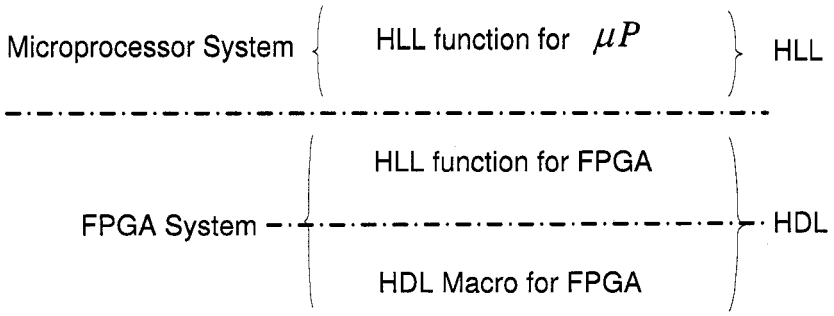


Figure 5.1: Three ways of implementing a function on a reconfigurable computer

our ECC benchmark across the boundaries shown in Fig. 5.1. Our goal is to find out, which level functions need to be implemented by a hardware designer as VHDL library macros, and at what level computer programmers and scientists can take over. Furthermore, the issues related to building the user macro library for ECC are also covered.

5.2 SRC Reconfigurable Computer

Before describing the architecture, we first introduce some notations of SRC-6 shown in Table 5.1. SRC-6 is a combination of two dual-microprocessor boards and one MAP board. A block diagram depicting a half of the SRC-6 machine is shown in Fig. 5.2. Each microprocessor board is connected to the MAP board through the SNAP interconnect, which can support a peak bandwidth of 800 MB/s.

SRC-6 has a similar compilation process to a conventional microprocessor-based computing system, but needs to support additional tasks in order to produce logic for the MAP reconfigurable processor, as shown in Fig. 5.3. There are two types of the application source files to be compiled. Source files of the first type are compiled targeting execution on the Intel microprocessors. Source files of the second type are compiled targeting execution on the MAP processor. These MAP source files contain

Table 5.1: Notations of SRC-6

Acronyms	Explaination
P4	Intel Pentium 4 microprocessor
L2	Level 2 Cache
MIOC	Memory and I/O bridge controller
MAP	Reconfigurable processor
DDR Interface	Double Data Rate Memory Interface
SNAP	SRC-developed Memory Interconnect
MAP	Reconfigurable Processor
HLL	High Level Language
HDL	Hardware Description Language

functions composed of HLL instructions and HDL macro calls. Such functions will be referred to in this thesis as MAP functions. Here, macro is defined as a piece of hardware logic designed to implement a certain function. Since users often wish to extend the built-in set of operators, the compiler allow users to integrate their own macros, encoded in VHDL or Verilog, into the compilation process. All macros must be optimized to operate at the clock frequency of 100MHz. A macro is invoked from within a C or Fortran function by means of a function call.

In Fig. 5.4, we demonstrate the mapping between macro calls and the corresponding contents of a MAP FPGA. Macro 2, called twice in Function 1, results in two instantiations of the logic block representing Macro 2. Values of arguments in the macro calls determine interconnects between macro instantiations in hardware. The contents of each MAP function in software determines the configuration of the entire FPGA device in hardware. Each time a new MAP function is called, the contents of the entire FPGA changes. If the same function is called several times in sequence, the reconfiguration is performed only once. During the subsequent MAP function calls, only data and control transfers take place. This way, SRC-6 implements run-time

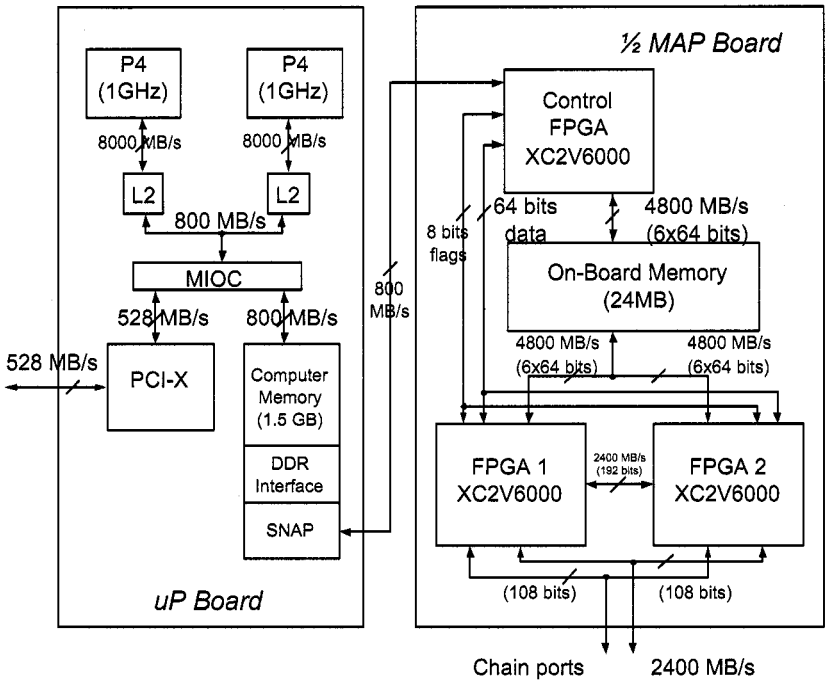


Figure 5.2: Hardware architecture of SRC-6

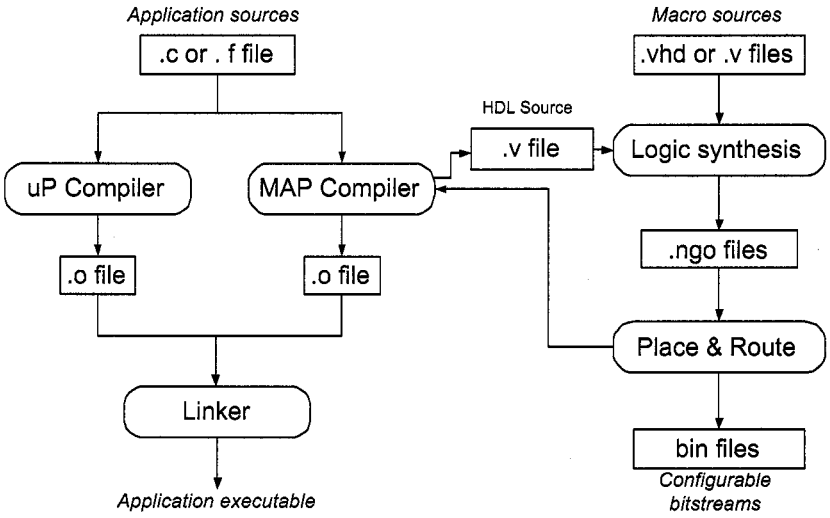


Figure 5.3: Compilation process of SRC-6

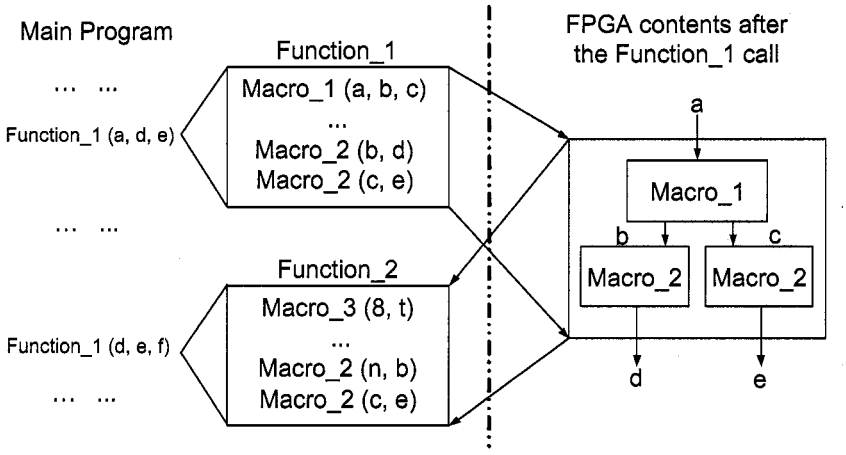


Figure 5.4: Programming model of SRC-6

reconfiguration [50].

5.3 Partitioning Schemes of Elliptic Curve Cryptosystems in SRC-6

López-Dahab algorithm is selected to compute kP of elliptic curves over $\mathbb{F}_{2^m}$ in polynomial basis representations in this work. A hierarchy of operations involved in an elliptic curve scalar multiplication is given in Fig. 5.5. Four levels of operations are involved in this hierarchy: scalar multiplication kP at the high level (H), point addition $P + Q$, point doubling $2P$, and projective-to-affine conversion $P2A$ at the medium level (M), inversion (INV) at the low level 2 (L2), and the $\mathbb{F}_{2^m}$ multiplication (MUL), squaring (SQR), and addition (XOR) at the lowest level (L1). Functions belonging to each of these four hierarchy levels (high, medium, low 2 and low 1) can be implemented using three different implementation approaches:

1. As a C function compiled for a general-purpose microprocessor;

2. As a C function compiled by the SRC MAP compiler to the hardware description

code running on the User FPGA;

3. As a VHDL hardware macro running on the User FPGA.

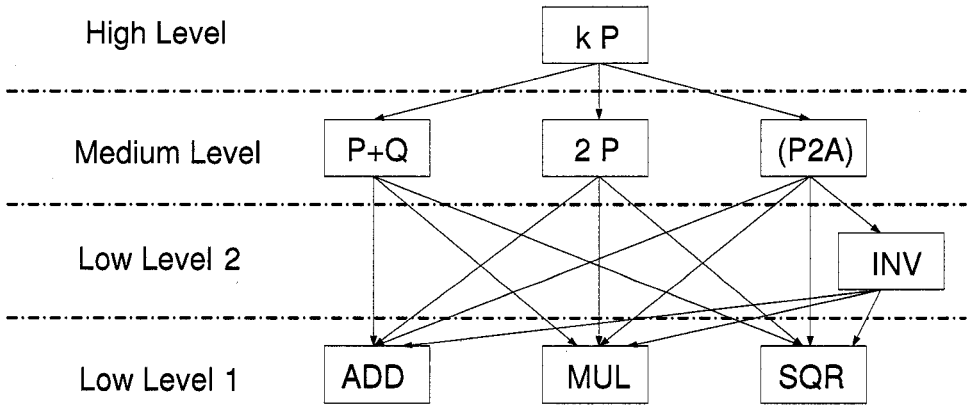
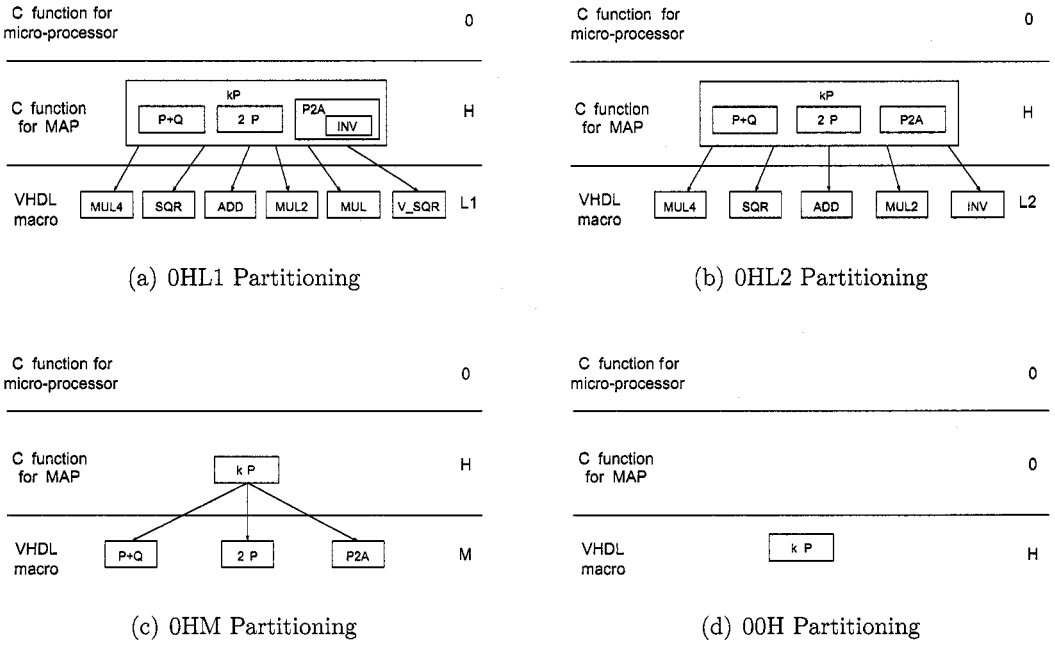


Figure 5.5: Hierarchy of the ECC operations

Two possible extreme cases are to implement scalar multiplication kP entirely in software as a C microprocessor function, or entirely in hardware, using traditional hardware design methodology (i.e., as a VHDL hardware macro, as shown in Fig. 5.6(d)). Several intermediate partitioning schemes are possible, and are presented in Figs. 5.6(a),5.6(b) and 5.6(c). Each of these approaches is characterized by a three letter codename, such as 0HM. The first letter of this codename determines which level operations (high, medium, low 2, low 1, or none (0)) are implemented in C on a general-purpose microprocessor. The second letter determines which operations are described as a C function for the MAP, and the third letter determines which operations are implemented as HDL macros. For example, the codename 0HM means that no operations are implemented in C for the microprocessor, a high-level operation (kP) is implemented as a C function for MAP, and medium level operations (point addition and doubling) are implemented as VHDL macros. In the 0HL1 scheme, only the lowest level functions (those at the level L1) needs to be implemented in

VHDL. In the 0HL2 scheme, the low level 2 function, $\mathbb{F}_{2^m}$ inversion, is implemented in VHDL, and the majority of the lowest level functions (those at the level L1) are also implemented in VHDL, as they are the components of the medium level operations. In some cases, it appeared to be beneficial to group several instantiations of the same lowest level operation into a single macro. This observation led to the implementation of macros MUL2 and MUL4, shown in Figs. 5.6(a) and 5.6(b). These macros implement respectively two and four instantiations of the $\mathbb{F}_{2^m}$ multiplier.



(a) 0HL1 Partitioning

(b) 0HL2 Partitioning

(c) 0HM Partitioning

(d) 00H Partitioning

Figure 5.6: Four alternative program partitioning schemes

In the most straightforward partitioning approach, HML1, the C MAP function performs in parallel two medium-level operations, $P + Q$ and $2P$. The results of both of these operations are returned to kP. Due to the SRC programming model, if $P + Q$ and $2P$ were implemented as separate high-level MAP functions, then the

reconfiguration of the User FPGA would need to take place each time we switch execution between $P + Q$ and $2P$. Since the time of the reconfiguration of the User FPGA has been measured to be equal about 47 ms, and kP implemented in VHDL executes within only 200 μs, even a single reconfiguration time by far exceeds the total execution time of kP in hardware. The existence of an integrated $P + Q/2P$ function and calling this function once as a part of the application setup eliminates the reconfiguration overhead. Unfortunately, an additional timing overhead is introduced during each MAP function call because of the control, input, and output transfer between the microprocessor board and the MAP board. In the current generation of the SRC system, this overhead has been measured to be in the range of 390 μs per function call. $m - 2$ such calls would be necessary to complete the entire kP operation. The inferred overhead would be orders of magnitudes larger than the average execution time of the entire kP operation in hardware. Therefore, this scheme was rejected at the early stage of our analysis. In order to minimize the large overhead of the HML1 scheme, the 0HL1 partitioning scheme (shown in Fig. 5.6(a) has been investigated and implemented. In this scheme, the MAP function is called only once and executes the entire high level operation kP. As a result, the control, input, and output overheads occur only once. The kP operation includes point addition, point doubling, and the conversion from the projective to affine coordinates. Point doubling requires two multiplications, which can be done in parallel. Point addition requires four multiplications in total. However, because of the data dependencies, these multiplications cannot be all performed in parallel, and need to be done in two iterations, with two independent multiplications per each iteration. The macros implemented in VHDL include MUL4, MUL2, MUL, SQR, and V_SQR. MUL4 is the macro which includes four multipliers working in parallel. Two of these

multipliers are used for point doubling and two for point addition. MUL2 consists of two multipliers working in parallel. It is used in the projective-to-affine (P2A) routine where 10 interdependent multiplications are scheduled using two multipliers, with the maximum possible use of parallelism. MUL is a single multiplier macro used to implement inversion. SQR is a one bit rotation for normal basis, and XOR arrays for polynomial basis. V_SQR is a variable exponentiation macro used in the inverse operation (INV) in P2A. The primary advantage of the 0HL1 partitioning scheme is that it makes use of only elementary operations in $\mathbb{F}_{2^m}$, which can either exist in the library, or be implemented using relatively small amount of the hardware designer effort. 0HL2 is similar to 0HL1 except that the inversion (INV) is implemented entirely as a VHDL macro. More hardware code needs to be generated in this case with some performance gain due to manual optimization. A further reduction in the execution time can be accomplished in the 0HM partitioning shown in Fig. 5.6(c), by implementing medium level operations, $P+Q$, $2P$, and $P2A$ as VHDL macros. The disadvantage of this approach is the required hardware knowledge, the level of HDL programming experience and the increased effort necessary to develop VHDL code in place of the C function for the MAP. The advantage is the opportunity for manual optimization of the VHDL code versus the HDL generated by the SRC MAP compiler. This hardware-oriented approach can be taken to its extreme by implementing the entire kP operation as VHDL macro (see partitioning scheme 00H shown in Fig. 5.6(d)).

Our implementation supports elliptic curve operations over $\mathbb{F}_{2^m}$ for $m = 233$, which is one of the sizes recommended by NIST [17]. Additionally, other sizes can be supported by changing the implementation of the $\mathbb{F}_{2^m}$ multipliers, the MSD-serial architecture shown in Fig. 6.3 , and the input/output size of other functional units.

All hardware macros have been developed first using standard tools for simulation and synthesis of digital circuits, Aldec Active-HDL and Synplicity Synplify Pro. All macros have been optimized to work at the clock frequency of 100 MHz. The XOR operation did not need to be implemented as a user macro, as it is a standard macro in the SRC library. This macro is invoked automatically when compiler encounters the XOR operator within a C MAP function. Our VHDL implementations have been tested for correct functionality using the public domain software, LiDIA [71]. The execution time of operations within the C MAP function has been measured in the number of clock cycles using the standard SRC macro, read_timer(). The end-to-end time of C functions has been measured in time units using the C timer function of the Linux operating system, gettimeofday().

Table 5.2: Results of the timing measurements for several investigated partitioning schemes and implemenation approaches.

System level architecture	End-to-end time (μs)	DMA data-in time (μs)	FPGA computation time (μs)	DMA data-out time (μs)	Total overhead (μs)	Speedup vs. software	Slowdown vs. VHDL macro
H00 (Software)	9710	NA	NA	NA	NA	1	30.2
0HL1	524.2	14.7	258.9	7.6	265.3	18.5	1.43
0HL2	519.7	14.5	254.4	7.5	265.3	18.7	1.43
0HM	366.3	14.7	101.1	7.4	265.2	26.5	1.00
00H (VHDL)	365.7	14.6	100.5	7.6	265.2	26.6	1.00

The results of the timing measurements for all investigated partitioning schemes are summarized in Table 5.2. The FPGA computation time includes only the time spent performing computations using User FPGAs. The end-to-end time includes the FPGA computation time and all overheads associated with the data and control transfers between the microprocessor board and the FPGA board. The total overhead is the difference between the end-to-end time and the FPGA computation time. Two

Table 5.3: Resource utilization for several investigated partitioning schemes and implementation approaches.

System level architecture	% of CLB slices (out of 33,792)	CLB increase vs. pure VHDL	% of LUTs (out of 67,584)	LUT increase vs. pure VHDL	% of FFs (out of 67,584)	FF count increase vs. pure VHDL
0HL1	99	1.86	55	1.25	66	3.30
0HL2	99	1.86	57	1.30	62	3.10
0HM	68	1.28	51	1.16	32	1.60
00H	53	1.00	44	1.00	20	1.00

specific components of the total overhead listed in Table 5.2 are DMA data-in time, and DMA data-out Time. They represent, respectively, the time spent to transfer inputs from the common memory to the on-board memory, and the time spent to transfer outputs from the on-board memory to the common memory. The end-to-end time and the total overhead do not include the reconfiguration time, which is equal to about 47 ms. This is justified by the fact that reconfiguration takes place only during the first function call, and does not effect the execution time for subsequent function calls. On two extremes, Table 5.2 shows the end-to-end time for the purely software implementation (Architecture H00), equal to about 9710 μs, and the FPGA computation time for the purely VHDL implementation (Architecture 00H), equal to about 365.7 μs. The speed-up by a factor of 30 has been demonstrated. It should be noted, that this speed-up could be greater, if we considered throughput (number of scalar multiplications per unit of time), instead of latency, and used all resources available in both user FPGAs for implementation of multiple computational units working in parallel. Additionally, our comparison assumes that the general purpose microprocessor is dedicated entirely to performing cryptographic transformations, which is rarely the case.

The scheme that requires the smallest amount of hardware expertise and effort, 0HL1, is still 18.5 times faster than software and less than 50% slower than pure VHDL macro. Implementing inversion in VHDL, in the 0HL2 scheme, does not give any significant gain in performance and only small reduction in terms of the resource usage. The 0HM scheme is more difficult to implement than 0HL1 and 0HL2 schemes, because of the additional operations that need to be expressed in VHDL. Nevertheless, using this scheme gives substantial advantages in terms of both performance (about 43% improvement) and resource usage (e.g., reduction in the number of CLB slices by 46% compared to the 0HL1 scheme). The most difficult to implement, the 00H scheme (the entire kP operation described in VHDL) appears to have the same speed as 0HM, but it provides an additional substantial reduction in terms of the amount of required FPGA resources. The MAP compiler optimizes performance over resource utilization. As it matures the compiler should be expected to balance high performance, ease of coding, and resource utilization to yield a truly optimized logic.

5.4 Macro Library for Public Key Schemes based on Elliptic Curves over Binary Fields

Building libraries in the SRC environment allows the reconfigurable community with access to SRC machines to share efforts [72]. Since those public key schemes based on elliptic curves over binary fields, such as elliptic curve cryptography (ECC), hyper-elliptic curve cryptography (HECC) and pairing based cryptography have the same field operations, see Fig. 5.7, it is beneficial to build binary field arithmetic macro libraries by which higher level operations for different cyrptographic schemes can be constructed via MAP C function, particularly suitable for those schemes with complicated group operations such as HECC. In this approach, we implement three types

of macro libraries, field arithmetic in normal basis representation for those fields recommended by NIST standard [17], field arithmetic in trinomial/pentanomial basis representation for arbitrary binary fields, and elliptic curve arithmetic in both basis representations. Several issues need to be considered along the process of building

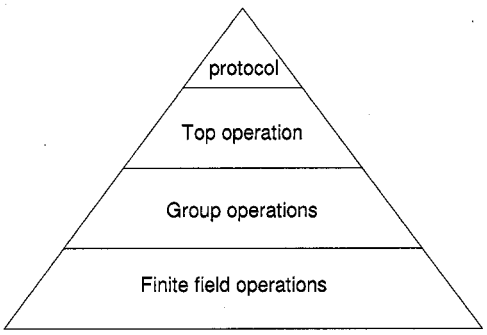


Figure 5.7: The hierarchy of those public key schemes based on elliptic curves

macro libraries for ECC over Binary Fields. First of all, variable operand sizes should be supported by those macros with generic parameters, such as trinomial/pentanomial multipliers, considering that the operand size required by SRC environment must be a multiple of 8, usually chosen as 32 or 64. One solution to this problem is to instantiate the generic component inside the wrapper whose interface satisfies the requirements of SRC programming, see Fig. 5.8. However it is required that users must be able to access the original VHDL codes. Alternatively, we can develop a tool written in C so that users need only specify macro type, field size and other generic parameters and the tool will generate the wrapper, the 'info' file and the black box (interface) accordingly. As a result, users will not necessarily access the VHDL codes.

Another consideration that will need to be made is the types of external dependencies needed by the library components provided, see [72]. This is usually of concern when providing debug code for macro development that requires use of software libraries such as LiDIA [71]. When running the debug mode (or emulation mode),

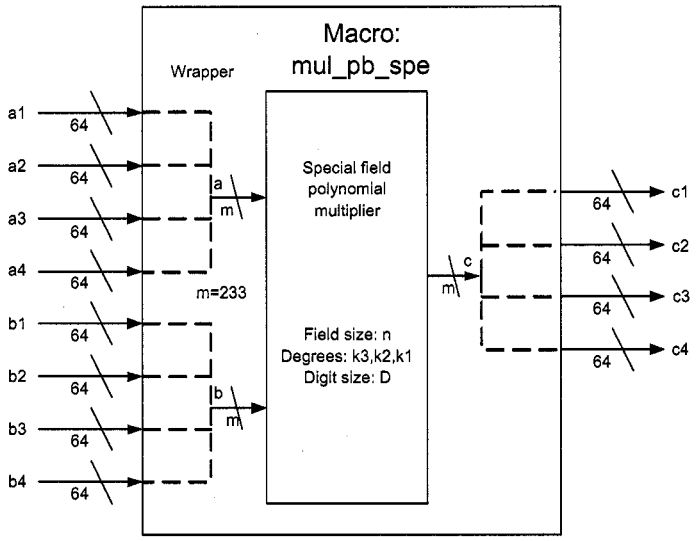


Figure 5.8: Block diagram of a multiplier macro

the macro in VHDL will be replaced with the corresponding LiDIA code to verify its functionality.

The third consideration is related to the issue of efficient usage of ECC library. Since the computation time for one field operation such as square, multiplication and inversion is much shorter than the overhead of communications between the microprocessor and the MAP board, it is suitable to transfer a bunch of data instead of one chunk to the MAP board and call the macro in a loop in MAP C so that high throughput can be achieved.

5.5 Summary

Reconfigurable computers are suitable for solving computationally intensive problems with limited input/output requirements and intrinsic parallelism. Therefore we choose SRC-6, an available reconfigurable computer on market, as our target platform for elliptic curve cryptosystems. We investigated several partition schemes to determine

the optimum boundary between hardware and software, and between the descriptions of hardware in VHDL vs. C for the four-level hierarchy of operations constituting the elliptic curve scalar multiplication. This boundary has to be considered to balance the trade-off between the end-to-end execution time, the resource utilization and the designer's productivity and ability. Assuming as a primary criterion the increased application developer productivity and an attempt to minimize involvement of hardware designers and traditional HDL-based design methodology, we have determined an optimum solution. In this solution, referred to as 0HL1 scheme, the entire scalar multiplication is implemented in hardware, but only low-level operations, $\mathbb{F}_{2^m}$ multiplication, squaring and addition needed to be described in VHDL. This partitioning scheme is shown to increase the execution time by less than 50% compared to the scheme based on implementing the entire scalar multiplication in VHDL. This result was accomplished at the cost of the increased use of FPGA resources, such as CLB slices, used mostly as a source of additional flip-flops. Finally, we address the issues relevant to building the macro libraries for those public-key cryptosystems based on elliptic curves, including variable operand sizes consistent with the SRC programming environment, external dependency of software library used in debug mode and efficient usage of such libraries.

Chapter 6: FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields

6.1 Introduction

Pairing based cryptography has become a subject of active research recently because it is the basis of a new class of public key cryptosystems called identity based cryptosystems. In these cryptosystems, a sender can derive a public key of a receiver directly from an ID of the receiver, without the need for any additional information. The basic idea of identity based cryptosystems was proposed by Shamir [46]. Applying pairing techniques to identity based cryptography was suggested first by Boneh and Franklin [47], and then extended by Sakai et al. [36].

Initially, the implementations of pairing based cryptosystems were commonly believed to be slow because of the heavy cost of underlying computations. A significant progress in this area has been accomplished by the works of Galbraith et al. [11], Barreto et al. [9], Granger et. al. [66, 67], and Duursma and Lee [68]. The optimizations introduced by these authors involved delicate techniques of deleting unnecessary operations from the Miller's algorithm [6]. In particular, the work of Duursma and Lee [68] promoted the study of efficient pairing computations for elliptic curves over Galois fields of characteristic three, $\mathbb{F}_{3^m}$. Subsequently, their idea was applied to the case of binary fields in [32], and generalized by Barreto et al. [10] to encompass different characteristics of the underlying field, as well as computations over hyperelliptic curves.

To the authors' knowledge, Kerins et al. [33,34], and Grabher and Page [35] were the first to report hardware implementations of pairing based cryptosystems. Both publications considered Duursma-Lee algorithm for elliptic curves over cubic fields. Recently, Ronans et al. [65] proposed the dedicated hardware for computing pairing on hyperelliptic curves using the algorithm introduced in [10].

Though the cubic elliptic and binary hyperelliptic cases have strong merits with the high security they offer for relatively low operand sizes, they also have some drawbacks for hardware implementations. First, the arithmetic circuits over cubic fields, $\mathbb{F}_{3^m}$, are more complex and costly in terms of the area and power compared to the circuits for computations over $\mathbb{F}_{2^m}$. Secondly, the binary hyperelliptic case [65] involves more complicated higher-level operations than the binary elliptic case, and the data path may be very complex for hardware implementations.

6.2 Overview of Tate Pairing Computation

Let E be an elliptic curve over a finite field $\mathbb{F}_q$ where q is a power of a prime. Let $l > 0$ be an integer relatively prime to q and let k be the least positive integer satisfying $q^k \equiv 1 \pmod{l}$. Such k is called a *security multiplier* or *embedding degree* of E. Let $E[l] = \{P | lP = O\}$ and $E(\mathbb{F}_q)[l] = \{P \in E(\mathbb{F}_q) | lP = O\}$.

A divisor D on E is a formal (finite) sum of the points P on the curve, $D = \sum n_p(P), \quad n_p \in \mathbb{Z}$. We call D a degree 0 divisor if $\sum n_p = 0$. A principal divisor is a divisor of the form $(f) = \sum n_p(P)$, where f is a rational function on E and P is a point of E with n_P the order of multiplicity of f at P. One can refer [45] for elementary introduction of divisor theories. The (reduced) Tate pairing τ_l on the set $E[l]$ is defined as follows.

Let $P \in E[l](\mathbb{F}_q)$ and $Q \in E[l](\mathbb{F}_{q^k})$. The Tate pairing is a map

$$\tau_l : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \longrightarrow \mathbb{F}_{q^k}^{\times}/(F_{q^k}^{\times})^l$$

with $\tau_l(P,Q) = f_P(D_Q)^{\frac{q^k-1}{l}}$, where f_P is a rational function satisfying $(f_P) = l(P) - l(O)$ and D_Q is a degree 0 divisor equivalent to $(Q) - (O)$ such that D_Q and (f_P) have disjoint supports.

It is well known that τ_l is a non-degenerate bilinear pairing [45]. An effective algorithm for finding a rational function f_P satisfying $(f_P) = l(P) - l(O)$ with $P \in E[l]$ was proposed by Miller [6]. The Miller's algorithm was further improved by the works of [9–11, 32, 66, 67].

Let E be a supersingular elliptic curve over $\mathbb{F}_{2^m}$ with $gcd(m,2) = 1$ defined by

$$E_b : Y^2 + Y = X^3 + X + b, \ b = 0, 1.$$

Then it is well known that the corresponding elliptic curves have the embedding degrees $k = 4$ and have orders dividing $2^{2m} + 1$. More precisely we have

$$|E_b(\mathbb{F}_{2^m})| = 2^m + 1 + (-1)^b 2^{\frac{m+1}{2}}, \text{ if } m \equiv 1, 7 \pmod 8$$

$$= 2^m + 1 - (-1)^b 2^{\frac{m+1}{2}}, \text{ if } m \equiv 3, 5 \pmod 8.$$

6.3 Algorithms for Pairing for Supersingular Elliptic Curves over Binary Fields

Inspired by the work of Duursma and Lee [68], a nice formula for the Tate paring computation of supersingular elliptic curve over binary field was proposed in [10, 32]. Moreover by introducing eta pairing technique, revised version of [10] contains an improved formula which reduces the number of iterations by half. The algorithms in

[10, 32] use repeated product of the term $g_{2^iP}(\psi Q)$. Here P, Q are points on E_b and $g_P(X, Y)$ denotes the tangent line at P. That is if $P = (\alpha, \beta)$, then g_P is given by the equation $g_P(x, y) = (\alpha^2+1)x+\beta^2+b+y$. Also ψ is a distortion map (automorphism) defined by

$$\psi : E_b \longrightarrow E_b, \quad \text{with} \quad \psi(x, y) = (x + s^2, y + sx + t),$$

where $s^2 + s + 1 = 0$ and $t^2 + t + s = 0$.

The results in [10, 32] imply that the Tate pairing $\tau(P, Q)$ is given by

$$\tau(P, Q) = \left(\prod_{i=0}^{m-1} g_{2^iP}(\psi Q)^{2^{2m-i}} \right)^{2^{2m}-1}.$$

Algorithm 9 A modified algorithm from [10, 32] for parallel computation of Tate pairing.

Require: $P = (\alpha, \beta), Q = (x, y)$

Ensure: $C = \tau(P, Q)$

1: $C \leftarrow 1$,

2: $\alpha \leftarrow \alpha^4$, $\beta \leftarrow \beta^4$, $v \leftarrow x^2 + 1$, $\theta \leftarrow \alpha v, u \leftarrow x^2 + y^2 + b + \frac{m-1}{2}$ {Initialize}

3: **for** $i = 0$ to $m - 1$ **do**

4: $A \leftarrow \beta + \theta + u + (\alpha + v)s + t$

5: $C \leftarrow C^2$

6: $C \leftarrow C \cdot A$

7: $\alpha \leftarrow \alpha^4, \quad \beta \leftarrow \beta^4, \quad u \leftarrow u + v, \quad v \leftarrow v + 1, \quad \theta \leftarrow \alpha v$

8: **end for**

9: $C \leftarrow C^{2^{2m}-1}$ {Final exponentiation}

For a point $P = (\alpha, \beta)$ on a supersingular curve, it is straightforward to verify that point doublings follow a nice formula $2^iP = \phi^i(\alpha^{(2i)}, \beta^{(2i)})$, where ϕ is defined as

$\phi(x,y) = (x+1, y+x)$ and $\alpha^{(i)}$ denotes $\alpha^{(i)} = \alpha^{2^{i'}}$ with $i' \equiv i \pmod{m}$ and $i' \geq 0$. One can show inductively that $\phi^i(x,y) = (x+i, y+ix+\epsilon_i)$, where $\epsilon_i = 0$ if $i \equiv 0,1 \pmod 4$, and $\epsilon_i = 1$ if $i \equiv 2,3 \pmod 4$

Thus

$$g_{2^iP}(x,y) = (\alpha_i^{(2i+1)} + 1)x + \beta_i^{(2i+1)} + b + y \tag{6.1}$$

where $(\alpha_i^{(j)}, \beta_i^{(j)}) = \phi^i(\alpha^{(j)}, \beta^{(j)})$. Note that $\alpha_i^{(j)} = (\alpha_i)^{(j)} = (\alpha^{(j)})_i$ since the automorphism ϕ and the Frobenius map are commutative to each other.

By refining eta pairing approach, Barreto et al. [10] successfully reduced the number of loop iterations by half so that they showed

$$\tau(P,Q) = \left(\ell(\psi Q) \prod_{i=0}^{\frac{m-1}{2}} g_{2^iP}(\psi Q)^{2^{\frac{m-1}{2}-i}} \right)^{MT} \tag{6.2}$$

where $MT = (2^{2m}-1)(2^m \mp 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} \pm 1)$, and $\ell(X,Y)$ is an equation of line passing $2^{\frac{m+1}{2}}P$ and ϵP with $\epsilon = (-1)^{b+\epsilon_{\frac{m+1}{2}}}$. $\ell(X,Y)$ is given by

$$\ell(X,Y) = Y + \beta + b + \epsilon_{\frac{m+1}{2}} + (\alpha + \frac{m-1}{2})(X+\alpha).$$

Here we computed the product by $\ell(\psi Q) = \ell(x+s^2, y+sx+t)$ after the for-loop, unlike in the original algorithm [10]. This is possible because the last element $g_{2^{\frac{m-1}{2}}P}(\psi Q)$ of the product in Equation 6.2 is related to $\ell(\psi Q)$ by the relation

$$\ell(\psi Q) = g_{2^{\frac{m-1}{2}}P}(\psi Q) + \frac{m+1}{2} + \alpha^2 + x + s.$$

In Alg. 10, the values of α and β can be recovered after the accumulative multiplication stage without additional memories. After reviewing previous works [33,35,65]

Algorithm 10 A modified algorithm from [10] for parallel computation of Tate pairing.

Require: $P = (\alpha, \beta), Q = (x, y)$

Ensure: $C = \tau(P, Q)$

1: $C \leftarrow 1$

2: $\alpha \leftarrow \alpha^2 + 1, \quad \beta \leftarrow \beta^2 + 1, \quad u \leftarrow y + b + 1, \quad v \leftarrow x + 1, \quad \theta \leftarrow \alpha v$ {Initialize}

3: **for** $i = 0$ to $\frac{m-1}{2}$ **do**

4: $A \leftarrow \beta + \theta + u + (\alpha + v + 1)s + t$

5: $C \leftarrow C^2$

6: $C \leftarrow C \cdot A$

7: **if** $i < \frac{m-1}{2}$ **then**

8: $\alpha \leftarrow \alpha^4, \quad \beta \leftarrow \beta^4, \quad u \leftarrow u + v + 1, \quad v \leftarrow v + 1, \quad \theta \leftarrow \alpha v$

9: **end if**

10: **end for**

11: $A \leftarrow A + (\alpha^2 + v + 1) + s$

12: $C \leftarrow C \cdot A$

13: $C \leftarrow C^{MT}, \quad MT = (2^{2m} - 1)(2^m \mp 2^{\frac{m+1}{2}} + 1)(2^{\frac{m+1}{2}} \pm 1)$ {Final exponentiation}

on FPGA implementations of Tate pairing and analyzing comparable finite fields of equivalent security levels, we choose two finite fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ for our single FPGA implementations. Our modified Algorithms 1 and 2 have the following characteristics.

1. They are parallel algorithms in the sense that the two crucial operations $C \leftarrow C^2A$ and $\theta \leftarrow \alpha v$ can be done in parallel.

2. We use a polynomial basis for our implementation of the above algorithms since a polynomial basis has an advantage over a normal basis for computing multiplications, even though a normal basis has a simple squaring and square root operation.

3. We do not compute square root as in the original algorithms, because in the pentanomial case where $m = 283$, we found that square root operation in hardware is rather complicated unlike the trinomial case, where square root operation is as fast as squaring [28].

6.4 Choice of Underlying Fields

The following is the table for applicable curves (having large prime order subgroups) with corresponding MOV security levels. Cofactor means that the order of the given curve divided by the cofactor is a prime. Cubic elliptic and binary hyperelliptic cases in Table 6.1 are taken from [10, 67] and the binary elliptic cases are computed using MAPLE. In Table 6.1, the **bolded** numbers in MOV security are the closest security levels to $\mathbb{F}_q$ with $q \approx 2^{1024}$. At the current state of cryptographic standards, it is reasonable to choose a field for FPGA implementation whose MOV security is comparable to 1024-bit RSA. In the cubic elliptic case, it is the field $\mathbb{F}_{3^{163}}$ that gives

the equivalent security level. Due to the nature of a cubic field, in which two bits are necessary to represent an element of $\mathbb{F}_3$, this field requires 326 bits to represent an element of the underlying field $\mathbb{F}_{3^{163}}$ and there is no known FPGA implementations for $\mathbb{F}_{3^{163}}$. Instead the implementation results for the cases $\mathbb{F}_{3^{79}}$ and $\mathbb{F}_{3^{97}}$ can be found in [33, 35]. For binary hyperelliptic case, the field which insures the security level of $\mathbb{F}_q$ with $q \approx 2^{1024}$ is $\mathbb{F}_{2^{103}}$, and its implementation can be found in [65].

Table 6.1: Applicable curves for cubic elliptic, binary hyperelliptic and binary elliptic cases.

	Fields	Curves	Co-Factors	MOV Security
Cubic Elliptic	$\mathbb{F}_{3^{79}}$	$Y^2 = X^3 - X - 1$	1	750
	$\mathbb{F}_{3^{97}}$	$Y^2 = X^3 - X + 1$	7	**922**
	$\mathbb{F}_{3^{163}}$	$Y^2 = X^3 - X - 1$	1	**1548**
	$\mathbb{F}_{3^{193}}$	$Y^2 = X^3 - X - 1$	1	1830
	$\mathbb{F}_{3^{239}}$	$Y^2 = X^3 - X - 1$	1	2268
	$\mathbb{F}_{3^{353}}$	$Y^2 = X^3 - X - 1$	1	3354
Binary Hyperelliptic	$\mathbb{F}_{2^{79}}$	$Y^2 + Y = X^5 + X^3 + 1$	151681	**948**
	$\mathbb{F}_{2^{103}}$	$Y^2 + Y = X^5 + X^3$	$13 \cdot 1237$	**1236**
	$\mathbb{F}_{2^{127}}$	$Y^2 + Y = X^5 + X^3 + 1$	198168459411337	1524
	$\mathbb{F}_{2^{199}}$	$Y^2 + Y = X^5 + X^3 + 1$	$2389 \cdot 121789$	2388
	$\mathbb{F}_{2^{239}}$	$Y^2 + Y = X^5 + X^3 + 1$	1	2868
	$\mathbb{F}_{2^{313}}$	$Y^2 + Y = X^5 + X^3 + 1$	1	3756
Binary Elliptic	$\mathbb{F}_{2^{239}}$	$Y^2 + Y = X^3 + X + 1$	1	**956**
	$\mathbb{F}_{2^{241}}$	$Y^2 + Y = X^3 + X + 1$	1	**964**
	$\mathbb{F}_{2^{283}}$	$Y^2 + Y = X^3 + X$	5	**1132**
	$\mathbb{F}_{2^{353}}$	$Y^2 + Y = X^3 + X + 1$	1	1412
	$\mathbb{F}_{2^{367}}$	$Y^2 + Y = X^3 + X + 1$	1	1468
	$\mathbb{F}_{2^{379}}$	$Y^2 + Y = X^3 + X + 1$	1	1516
	$\mathbb{F}_{2^{457}}$	$Y^2 + Y = X^3 + X + 1$	1	1828
	$\mathbb{F}_{2^{557}}$	$Y^2 + Y = X^3 + X$	5	2228

6.5 Software Results

We have implemented Alg. 9 and 10 for Tate pairing over binary fields listed in Table 6.1. The generating polynomials for these fields are provided in Table 6.2. The subfield arithmetic was realized via a public domain C++ library named LiDIA, and we developed the high level operations. All the codes were compiled with g++

3.0.4 and simulations were performed on a Xeon station working at 2.8 GHz. With our software implementations, we can generate the bilinear test-vectors for our FPGA realizations of pairing. Additionally, the comparisons of timing between hardware and software are performed in the following sections. In Table 6.3, we found that Alg 10 is 1.5 times faster than Alg. 9 on average because half iterations in the accumulative multiplication stage are performed. The speedup of Alg. 10 vs. Alg. 9 is lower for $\mathbb{F}_{2^{379}}$ because more squarings in $\mathbb{F}_{2^{4m}}$ will be performed sequentially in the final exponentiation stage of Alg. 10.

Table 6.2: Generating polynomials for those selected binary fields.

Fields	Generating Polynomials $f_m(x)$	Fields	Generating Polynomials $f_m(x)$
$\mathbb{F}_{2^{239}}$	$x^{239}+x^{36}+1$	$\mathbb{F}_{2^{367}}$	$x^{367}+x^{21}+1$
$\mathbb{F}_{2^{241}}$	$x^{241}+x^{70}+1$	$\mathbb{F}_{2^{379}}$	$x^{379}+x^{10}+x^{8}+x^{5}+1$
$\mathbb{F}_{2^{283}}$	$x^{283}+x^{12}+x^{7}+x^{5}+1$	$\mathbb{F}_{2^{457}}$	$x^{457}+x^{16}+1$
$\mathbb{F}_{2^{353}}$	$x^{353}+x^{69}+1$	$\mathbb{F}_{2^{557}}$	$x^{557}+x^{7}+x^{6}+x^{5}+1$

Table 6.3: The timing of both algorithms of the Tate pairing (in ms) over binary fiels, on a Xeon workstation at 2.8 GHz.

Finite Fields $\mathbb{F}_{2^m}$	Latency of Alg. 9	Latency of Alg. 10	Speedup Alg. 10 vs. Alg. 9	Finite Fields $\mathbb{F}_{2^m}$	Latency of Alg. 9	Latency of Alg. 10	Speedup Alg. 10 vs. Alg. 9
$\mathbb{F}_{2^{239}}$	10.8	7.5	1.44	$\mathbb{F}_{2^{367}}$	28.8	18.4	1.57
$\mathbb{F}_{2^{241}}$	11.0	7.7	1.43	$\mathbb{F}_{2^{379}}$	31.0	22.5	1.38
$\mathbb{F}_{2^{283}}$	18.1	12.2	1.48	$\mathbb{F}_{2^{457}}$	42.5	26.7	1.59
$\mathbb{F}_{2^{353}}$	27.0	16.9	1.60	$\mathbb{F}_{2^{557}}$	84.3	52.3	1.61

Although the arithmetic of paring computations on binary elliptic curves is very simple, there is no known FPGA implementation at this moment, and the reason might be low security multiplier (embedding degree). However since the hardware implementation of the cubic field is not so efficient compared with a binary field, and the binary hyperelliptic curve has complex arithmetic operations for point additions (which make the data path complicated for FPGAs), it is desirable to design an FPGA

circuit for binary elliptic case and compare it with existing architectures. From the previous results on cubic elliptic and binary hyperelliptic cases, we chose two fields $\mathbb{F}_{2^{239}}$ for the comparison with the cubic case [33,35] and $\mathbb{F}_{2^{283}}$ for the comparison with the binary hyperelliptic case [65].

6.6 FPGA implementations

In this section, we focus on the FPGA implementations of Tate pairing on supersingular elliptic curves over binary fields, $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$. Both Alg. 9 and 10 contain mainly two stages, accumulative multiplication and final exponentiation, in both of which the operations in $\mathbb{F}_{2^{4m}}$ are involved. The best approach is to represent $\mathbb{F}_{2^{4m}}$ as an extension of $\mathbb{F}_{2^m}$ with a convenient basis, and work over the smaller field whenever possible. We use the basis $\{1, s, t, st\}$ for $\mathbb{F}_{2^{4m}}$ over $\mathbb{F}_{2^m}$, with $s \in \mathbb{F}_{2^{2m}}$, $t \in \mathbb{F}_{2^{4m}}$ satisfying:

$$s^2 + s + 1 = 0 \text{ and } t^2 + t + s = 0. \tag{6.3}$$

To obtain a high-efficiency pairing accelerator, one must consider issues such as: algorithm selection, top architecture, parallelism, resource sharing and efficient realization of the underlying field arithmetic.

Algorithm comparison: The most attractive advantage of using Alg. 10 instead of Alg. 9 is that it takes a half of iterations in the accumulative multiplication stage. However, a lower complexity of the data-path for final exponentiation can be gained when using Alg. 9 considering that its final exponentiation is much simpler than that of Alg. 10. Both algorithms are realized in our experiments.

Top architecture: We adopt the similar top architecture for our pairing accelerator as ECC, shown in Fig. 6.1. One can refer to Section 4.3.1 for more details of

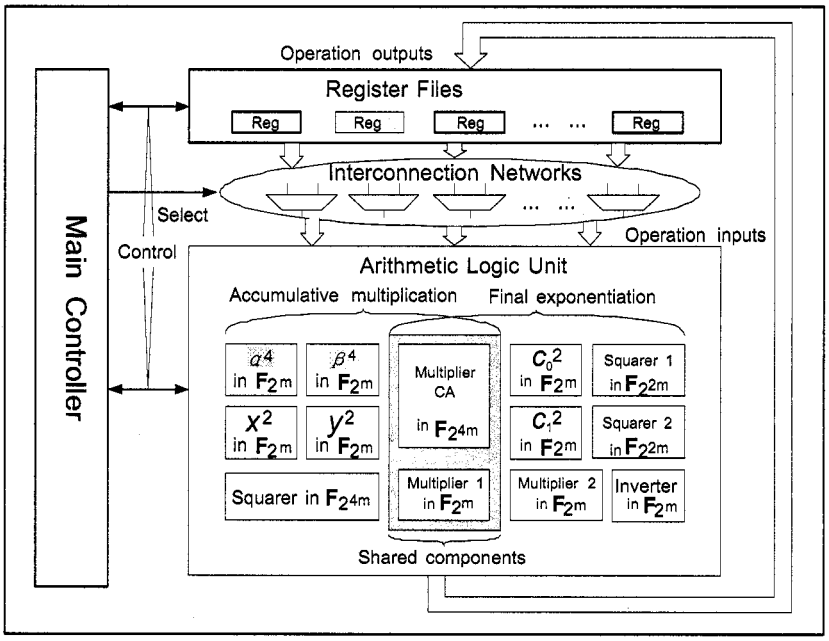


Figure 6.1: Top architecture of the accelerator of Tate pairing over binary fields.

reasoning.

Parallelism and sharing resources: In the first stage of both algorithms, the computations $C \cdot A$ and $\alpha \cdot v$ can be completed simultaneously. Additionally, in the second stage of both algorithms, by multiplying the conjugates of the elements in extension fields $\mathbb{F}_{2^{4m}}$ and $\mathbb{F}_{2^{2m}}$, the inversion in extension fields can be transformed into one inversion in $\mathbb{F}_{2^m}$ and several multiplications in $\mathbb{F}_{2^m}$, $\mathbb{F}_{2^{2m}}$ and $\mathbb{F}_{2^{4m}}$. We use one special extension field multiplier, namely CA in Fig. 6.1, to perform the multiplications involved in both stages. The multiplier CA can be optimized to obtain a compact design by sharing some combinational circuits among several multipliers in $\mathbb{F}_{2^m}$ in case of the same operand. The multiplier computing $\alpha \cdot v$ in the first stage performs multiplications in $\mathbb{F}_{2^m}$ involved in final exponentiation as well.

Underlying field arithmetic: The underlying field $\mathbb{F}_{2^m}$ is constructed via the low Hamming weight irreducible polynomial, such as a trinomial or a pentanomial, by

which reductions become simple. Squarer should not be shared since the multiplexers introduced are more expensive. Multiplier is the most significant component directly determining the performance of the accelerator, so it is imperative to implement it with high efficiency. Linear feedback shift register (LFSRs) structure is adopted in our MSD-serial multipliers. The multiplicative inversion in $\mathbb{F}_{2^m}$ is computed using Itoh-Tsujii algorithm [27]. Since most intensive computations are performed in the first stage, the multipliers inside the component CA and the multiplier performing $\alpha \cdot v$ should have large digit sizes to achieve high operation speed. On the other hand, in order to decrease the resource utilization without loss of performance, the multipliers working only at the final exponentiation stage can be relatively slow, with small digit size.

6.6.1 Design of Arithmetic Logic Unit

In the following section, we briefly review the traditional technique computing squaring over the underlying field. Our main interest is to compute the multiplications $C \cdot A$ efficiently. In particular, we propose a method optimizing individual subfield multiplier to obtain a compact design of the extension field multiplier CA. Furthermore, we present two schemes for the multiplier CA in which a different number of multipliers are used. These two schemes are ported to an FPGA device. The optimal choices are made based on the product of latency by area. Finally, the simplifying technique for the final exponentiation in both algorithms is explained.

Squarer

Squaring an element $a = \sum_{i=0}^{m-1} a_i x^i \in \mathbb{F}_{2^m}$, where $a_i \in \mathbb{F}_2$, is given by the equation $a^2 = \sum_{i=0}^{m-1} a_i x^{2i}$. Since the underlying fields are constructed via irreducible trinomials

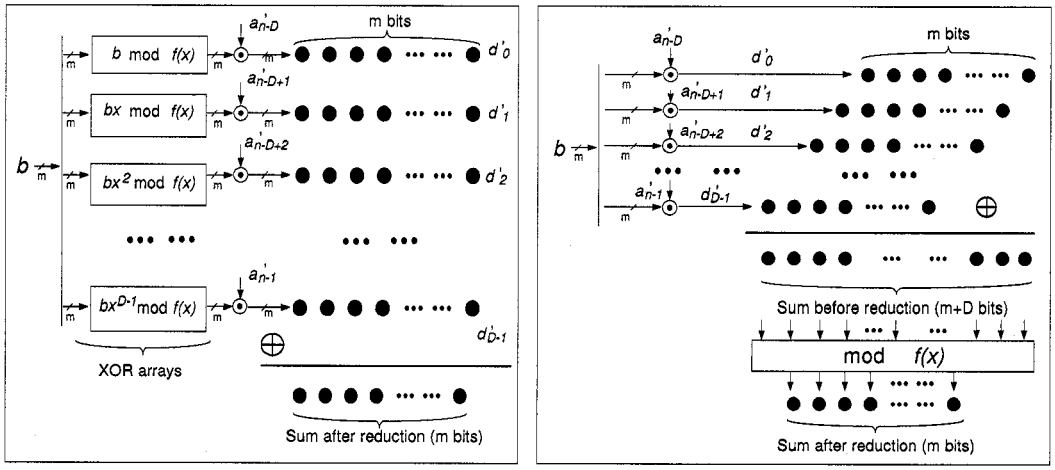
or pentanomials, by replacing x^m with $x^k + 1$ or $x^{k_3} + x^{k_2} + x^{k_1} + 1$, we can get the formulae for computing the coefficients of a^2. The circuit complexity in terms of gate count is proportional to m. Squaring over the extension fields $\mathbb{F}_{2^{2m}}$ and $\mathbb{F}_{2^{4m}}$ is relatively easy and can be decomposed into several squarings in $\mathbb{F}_{2^m}$.

Multiplier

Digit serial multiplier, allowing the trade-off between timing and area, is more suitable for cryptographic applications with large operand sizes. There are two basic algorithms computing multiplications with polynomial basis representation in $\mathbb{F}_{2^m}$, left-to-right and right-to-left. It is claimed that the first algorithm is superior in term of low power [74]. Additionally, we find that fewer registers are needed when using the first algorithm because only partial product needs to be updated in each iteration apart from the shift-in digits. However, both partial product and one operand must be updated in each iteration when using the second one. Therefore we adopt the left-to-right algorithm to derive the digit-serial multiplier.

Let $a(x)$ and $b(x)$ be the two operands of the multiplication in $\mathbb{F}_{2^m}$ and let $c(x)$ be the product. Let $n = min\{l \mid l \in \mathbb{Z},\ l \geq m,\ \text{and}\ D \mid l\}$. Let $a' = \sum_{i=0}^{n-1} a'_i x^i$, where $a'_i = a_i$ if $0 \leq i \leq m-1$, otherwise $a'_i = 0$. The multiplier contains mainly two parts, XOR-AND arrays for computing $s(x) = \sum_{i=0}^{D-1} a'_{n-D+i} x^i b(x) \bmod f_m(x)$ and LFSRs for computing $c(x) \leftarrow c(x) + s(x)$. There are two candidates for the second part, see Fig. 6.2. The first approach is to compute $x^i b(x) \bmod f_m(x)$ separately and the partial sum is kept in m bits. For the second one, the partial sum is kept in $m+D$ bits before reduction. Even though fewer XOR gates are used in the second structure for an individual multiplier, these XOR-AND arrays cannot be shared among different multipliers in $\mathbb{F}_{2^m}$. Additionally, the wire density is increased significantly if D is

chosen large. On the contrary, the first approach is more suitable to construct the extension field multiplier CA in $\mathbb{F}_{2^{4m}}$ considering that the XOR arrays for $x^i b(x)$ mod $f_m(x)$ can be shared among different multipliers in $\mathbb{F}_{2^m}$ in case they share one operand, see Equation 6.5 and Fig. 6.3. The second advantage is its low wire density which makes it easy for placing and routing.



(a) Structure 1 (b) Structure 2

Figure 6.2: Alternative structures for $\sum_{i=0}^{D-1} a_{n-D+i} x^i b(x)$ mod $f_m(x)$.

With the basis $\{1, s, t, st\}$ of $\mathbb{F}_{2^{4m}}$ over $\mathbb{F}_{2^m}$, we may write $A = w + zs + et$ where $w, z \in \mathbb{F}_{2^m}$ and $e \in \mathbb{F}_2$. We set $e = 1$ in the accumulative stage and $e = 0$ in the final exponentiation stage. Let $C = c_0 + c_1 s + c_2 t + c_3 st$, $c_i \in \mathbb{F}_{2^m}$, be the partial product of $C \leftarrow C \cdot A$. It it not suitable to apply Karatsuba-Ofman algorithm directly to compute this extension field multiplication recursively since more underlying field multiplications would need to be calculated. However, we can use the same idea to simplify the computations of coefficients c'_1 and c'_3 (see Equation 6.5).

$$C \cdot (w + zs + et) = (c_0 + c_1 s + c_2 t + c_3 st)(w + zs + et) = c'_0 + c'_1 s + c'_2 t + c'_3 st, \quad (6.4)$$

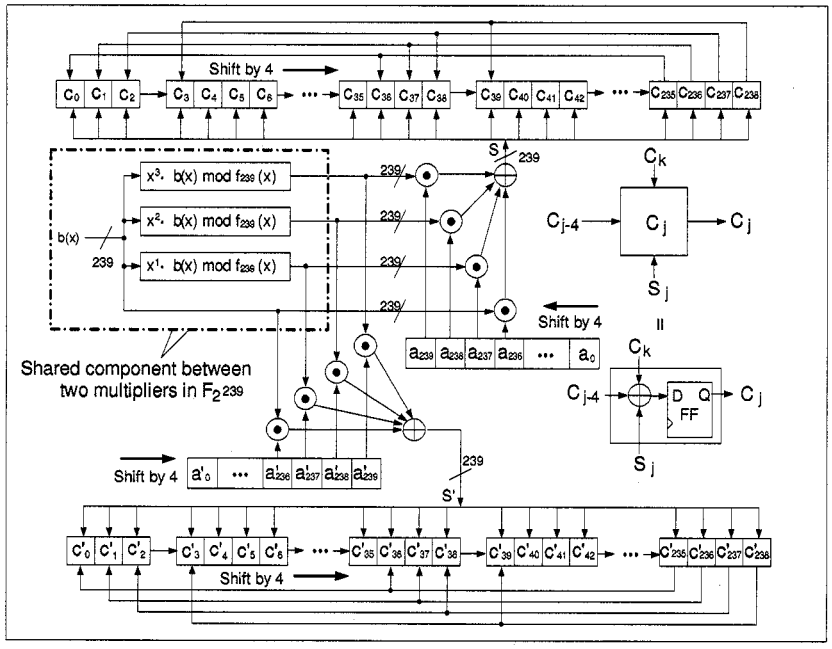
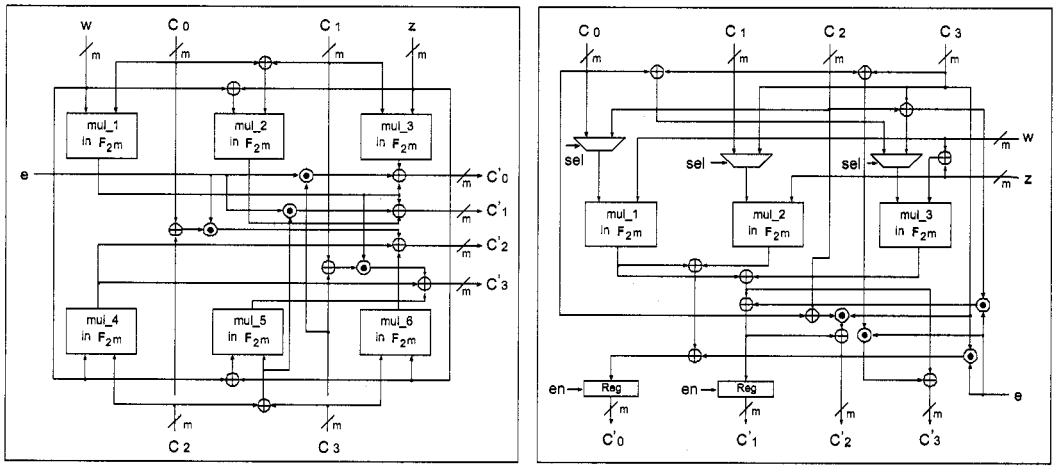


Figure 6.3: Two digit serial multipliers in $\mathbb{F}_{2^{239}}$ with $D = 4$ sharing the component for $\sum_{i=0}^{3} x^i b(x) \bmod f_{239}(x)$.



(a) Scheme 1: 6 multipliers adopted

(b) Scheme 2: 3 multipliers adopted

Figure 6.4: Alternative schemes for CA.

Table 6.4: FPGA implementation results for the multipliers CA over $\mathbb{F}_{24\times239}$ and $\mathbb{F}_{24\times283}$, and the target device is Xilinx XC2VP100-6FF-1704.

Scheme 1: 6 multipliers adopted for $\mathbb{F}_{2^{239}}$				Scheme 1: 6 multipliers adopted for $\mathbb{F}_{2^{283}}$			
	$D=4$	$D=8$	$D=16$		$D=4$	$D=8$	$D=16$
# FF	3664(4%)	3664(4%)	3664(4%)	# FF	4324(4%)	4348(4%)	4348(4%)
# LUT	7799(8%)	13508(15%)	20744(23%))	# LUT	9273(10%)	16060(18%)	24729(28%)
# CLB slices	4268(9%)	7094(16%)	10722(24%))	# CLB slices	5070(11%)	8416(19%)	12856(29%)
Clock period (ns)	9.61	9.98	9.99	Clock period (ns)	9.84	9.98	9.99
Latency (ns)	576.6	299.4	149.9	Latency (ns)	698.6	359.3	179.8
Scheme 2: 3 multipliers adopted for $\mathbb{F}_{2^{239}}$				Scheme 2: 3 multipliers adopted for $\mathbb{F}_{2^{283}}$			
	$D=4$	$D=8$	$D=16$		$D=4$	$D=8$	$D=16$
# FF	2675(3%)	2675(3%)	2675(3%)	# FF	3159(3%)	3171(3%)	3171(3%)
# LUT	5580(6%)	8439(9%)	12075(13%)	# LUT	6632(7%)	10031(11%)	14428(16%)
# CLB slices	3617(8%)	5331(12%)	7536(17%)	# CLB slices	3046(6%)	4483(10%)	6306(14%)
Clock period (ns)	9.86	9.98	9.76	Clock period (ns)	8.89	9.99	9.98
Latency (ns)	1182.8	598.8	292.8	Latency (ns)	1262.4	719.3	359.3

where

$$c'_0 = c_0w + c_1z + ec_3 \qquad c'_1 = (c_0 + c_1)(w + z) + c_0w + e(c_2 + c_3)$$

$$c'_2 = c_2w + c_3z + e(c_0 + c_2) \qquad c'_3 = (c_2 + c_3)(w + z) + c_2w + e(c_1 + c_3) \qquad (6.5)$$

Therefore, it takes only 6 $\mathbb{F}_{2^m}$-multiplications for the computation of $C \cdot A$, and all these 6 multiplications can be done simultaneously if 6 multipliers in $\mathbb{F}_{2^m}$ are adopted, see Fig. 6.4(a). Alternatively, if 3 multipliers are adopted in case of limited resources, the computation of $C \cdot A$ will take two multiplication rounds. In the first round, the first two coefficients of the product, namely c'_0 and c'_1, will be computed and stored in the registers. In the second round, the other two coefficients, namely c'_2 and c'_3, will be computed, see Fig. 6.4(b). To get the optimal choice in terms of low product of latency by area, both schemes of the special multipliers CA over $\mathbb{F}_{24\times239}$ and $\mathbb{F}_{24\times283}$ are ported to the same FPGA device, Xilinx XC2VP100-6FF-1704. The optimization goal is speed instead of area. Second, the hierarchy is not kept so that registers and XOR arrays can be saved considering that several underlying field multiplications

share the same operands. Comparisons in terms of timing and area are demonstrated in Table 6.4.

According to Table 6.4, Scheme 1 is always superior in terms of low product of latency by area. Hence we adopt 6 multipliers in the extension field multiplier CA for our pairing accelerator so that the multiplication $C \cdot A$ can be completed in one underlying field multiplication round.

Inverter

The inversion in $\mathbb{F}_{2^{4m}}$ involved in the final exponentiation can be transformed into one inversion in $\mathbb{F}_{2^m}$ and several multiplications in $\mathbb{F}_{2^m}$ (the transforming procedure is explained in details in Sec. 6.6.2). Therefore, we only need to implement one inverter in $\mathbb{F}_{2^m}$. Itoh-Tsujii's algorithm is adopted in our realizations.

6.6.2 Simplifying the Data-path for the Final Exponentiation

Let $C = C_1 + C_2 t$ with $C_1, C_2 \in \mathbb{F}_{2^{2m}}$. Using subfield arithmetic and repeated norm calculations, we may compute $C^{2^{2m}-1}$. Letting $C_1^2 + C_1 C_2 + C_2^2 s = c_0 + c_1 s$ with $c_0, c_1 \in \mathbb{F}_{2^m}$, a straight calculation shows

$$C^{2^{2m}-1} = \frac{1}{c_0^2 + c_0 c_1 + c_1^2} \cdot (c_0 + c_1(s+1)) \cdot$$
$$(C_1^2 + C_2^2(1+s) + C_2^2 t),$$

where the total cost of the above computations is one $\mathbb{F}_{2^m}$-inversion plus 12 $\mathbb{F}_{2^m}$-multiplications. That is, we need 4 $\mathbb{F}_{2^m}$-multiplications (1 $\mathbb{F}_{2^{2m}}$-multiplication for

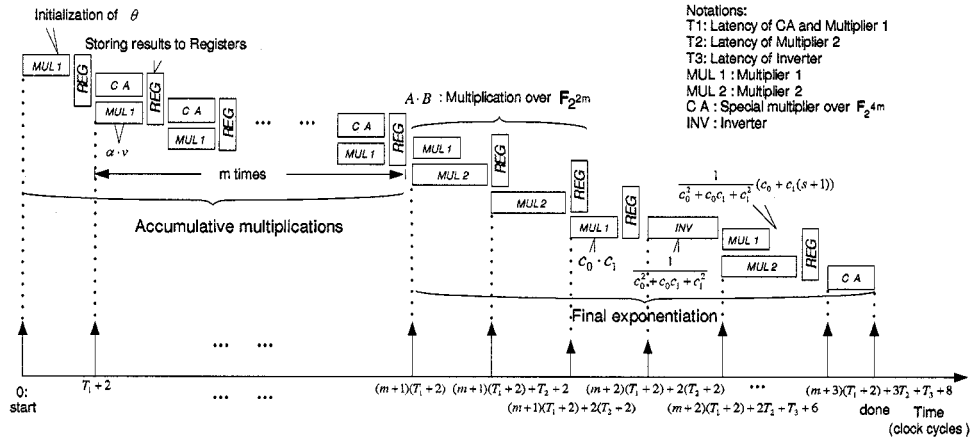


Figure 6.5: Timing diagram of pairing computations according to Alg. 9.

C_1C_2 and 1 $\mathbb{F}_{2^m}$-multiplication for c_0c_1) for the denominator $c_0^2 + c_0c_1 + c_1^2$, 2 $\mathbb{F}_{2^m}$-multiplications for $\frac{1}{c_0^2+c_0c_1+c_1^2} \cdot (c_0 + c_1(s+1))$, and 6 $\mathbb{F}_{2^m}$-multiplications (2 $\mathbb{F}_{2^{2m}}$-multiplications) which come from the final multiplication by $C_1^2 + C_2^2(1+s) + C_2^2 t$. Therefore only six multiplication units (for parallel processing) are needed here and one can reuse the same arithmetic units as shown in Fig. 6.5 for multiplications, and thus the additional cost of the final exponentiation is the inversion circuit for computing $\frac{1}{c_0^2+c_0c_1+c_1^2}$.

In Algorithm 2, we have more complicated final exponentiation, where $M = \frac{2^{4m}-1}{|E_b(\mathbb{F}_{2^m})|} = \frac{2^{4m}-1}{2^m \pm 2^{\frac{m+1}{2}}+1} = (2^{2m}-1)(2^m \mp 2^{\frac{m+1}{2}}+1)$ and $T = 2^{\frac{m+1}{2}} \pm 1$. Here we should be cautious about appropriate sign of T. The original definition of T [10] is $T = 2^m - N = \mp 2^{\frac{m+1}{2}} - 1$. However when $T = -2^{\frac{m+1}{2}} - 1$, one computes $(-T)\{(P) - (O))\}$ which gives an inverse of the rational function corresponding to $T\{(P) - (O))\}$, so the exponents should be changed to $-T$ in this case. Suppose z is in $GF(2^{4m})$ such that $z = w^{2^{2m}-1}$ for some $w \in \mathbb{F}_{2^{4m}}$. Then $z^{2^{2m}+1} = w^{2^{4m}-1} = 1$ and thus we get $z^{-1} = z^{2^{2m}}$. Moreover from $1 = z^{2^{2m}+1} = z^{(2^m+1+2^{\frac{m+1}{2}})(2^m+1-2^{\frac{m+1}{2}})}$,

we get

$$z^{(2^m+1+2^{\frac{m+1}{2}})(2^{\frac{m+1}{2}}-1)} = z^{(2^m+1+2^{\frac{m+1}{2}})2^m}$$

Therefore the exponent MT can be interpreted as:

$$MT = (2^{2m}-1)(2^m+2^{\frac{m+1}{2}}+1)(2^{\frac{m+1}{2}}-1)$$

$$= (2^{2m}-1)(2^m+2^{\frac{m+1}{2}}+1)2^m \tag{6.6}$$

6.6.3 Parameter Choices for ALU

In Fig. 6.5, we provide the timing diagram of scheduling computations for pairing according to Algorithm 9. Additions and squarings realized via combinational circuits need not to be taken into account for estimating the latency.

Let Δ_1 denote the latency for one computation of Tate pairing using Alg. 9 and let T_{clk} denote the clock period. Let D_1 denote the digit size of multipliers inside CA and Multiplier 1; D_2 denote the size of Multiplier 2; and D_3 denote the digit size of multiplier inside the inverter. The notations of T_1, T_2, T_3 are specified in Fig. 6.5. Then we can estimate the latency for computing one Tate pairing using Alg. 9 as follows:

$$\Delta_1 = (m+3)(T_1+2) + 3T_2 + T_3 + 8 \tag{6.7}$$

where $T_1 = \lceil m/D_1 \rceil T_{clk}$, $T_2 = \lceil m/D_2 \rceil T_{clk}$. Two extra clock cycles are required for the register read/write operations. In case of $\mathbb{F}_{2^{239}}$, it takes around 239 clock cycles for exponentiation and 12 multiplications in case of computing c^2 each cycle. However the latency for exponentiation needs to be shortened. We compute c^{2^4} in each cycle so that 60 cycles are necessary to complete the exponentiation for the inversion in $\mathbb{F}_{2^{239}}$.

Then $T_3 \approx (60 + 12 \cdot \lceil 239/D_3 \rceil)T_{clk}$. Similarly $T_3 \approx (71 + 11 \cdot \lceil 283/D_3 \rceil)T_{clk}$ for $\mathbb{F}_{2^{283}}$. If we choose $D_1 = 16$, $D_2 = 4$ and $D_3 = 8$, the time spent on final exponentiation is 10.3% of accumulative multiplications. However if we choose $D_3 = 4$ and keep the same value of D_1 and D_2, the time for final exponentiation is 23.8% of accumulative multiplication. So we choose $D_1 : D_2 : D_3 = 4 : 1 : 2$ for both $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$.

For Alg. 10, the computation of C^{MT} needs to be additionally taken into account for latency estimation, see Equation 6.6. We use the component computing C^{2^4} each cycle for the extension field exponentiation, and one extension field multiplication with two operands in $\mathbb{F}_{2^{4m}}$ can be completed by CA in two underlying field multiplication rounds. Then, we use the following equation to estimate the latency of pairing for Alg. 10,

$$\Delta_2 = \frac{m+9}{2}(T_1 + 2) + 3T_2 + T_3 + T_4 + 8;$$

$$T_4 = \frac{m+1}{2}T_{clk} + 4(T_1 + 2) \tag{6.8}$$

where T_4 denotes the latency for C^{MT}. We use the same ratio of D_1, D_2 and D_3 as Alg. 9.

6.6.4 Results and Comparisons with Previous Work

The target device for our implementations is Xilinx XC2VP100-6FF-1704. For Alg. 9, the digit size of multipliers inside CA is chosen as 16 and 32 for both $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$. For Alg. 10, due to the limitation of resources, the digit size D_1 is chosen as 16. Our FPGA accelerators have been fully tested via the bilinear test-vectors generated by LiDIA. The results after placing and routing via Xilinx ISE-7.1 are summarized in Table 6.5. Compared with software implementations, our FPGA accelerator can

Table 6.5: Performance and cost of Tate pairing accelerators on elliptic curves over $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$.

		# FF	# LUT	# CLB slices	f (MHz)	Latency (μs)	Speedup over software
Alg. 9	$\mathbb{F}_{2^{239}}$, $D_1 = 16$	10,981 (12%)	34,499 (12%)	18,202 (41%)	100	55	196
	$\mathbb{F}_{2^{239}}$, $D_1 = 32$	11,077 (12%)	59,971 (68%)	31,719 (71%)	83	43	251
	$\mathbb{F}_{2^{283}}$, $D_1 = 16$	12,995 (14%)	42,997 (48%)	22,726 (51%)	84	87	208
	$\mathbb{F}_{2^{283}}$, $D_1 = 32$	13,007 (14%)	72,961 (82%)	37,803 (85%)	72	61	297
Alg. 10	$\mathbb{F}_{2^{239}}$, $D_1 = 16$	14,226 (16%)	48,895 (55%)	25,487 (57%)	84	41	183
	$\mathbb{F}_{2^{283}}$, $D_1 = 16$	16,563 (18%)	64,845 (73%)	33,252 (75%)	56	78	156

Table 6.6: Comparison with earlier FPGA implementations.

	Curves	Underlying fields	MOV Security	Xilinx FPGA device	Controller	# CLB slices	Digit size D	f(MHz)	Latency (μs)
[33]	Elliptic	$\mathbb{F}_{3^{97}}$	922	XC2VP125	Hardwired logic	55,616	4	10	850
[35]	Elliptic	$\mathbb{F}_{3^{97}}$	922	XC2VP4FF672	Microprocessor	4,481	4	150	432
[65]	Hyperelliptic	$\mathbb{F}_{2^{103}}$	1236	XC2VP125	Hardwired logic	43,986	16	32	749
Alg. 10	Elliptic	$\mathbb{F}_{2^{239}}$	956	XC2VP100	Hardwired logic	25,287	16	84	41
Alg. 9	Elliptic	$\mathbb{F}_{2^{283}}$	1132	XC2VP100	Hardwired logic	37,803	32	72	61

run 150-to-300 times faster. In case of the same digit sizes of both algorithms, the latency of Alg. 10 is shorter than Alg. 9. However more resources are utilized for Alg. 10 because the multiple squarers in $\mathbb{F}_{24m}$ are adopted for C^{MT} and more complicated datapath in final exponentiation cannot be avoided so that the critical path is longer than Algorithm 9.

Compared with other researchers' works [33, 35, 65], almost at the same level of security strength, our Tate pairing accelerators can run 10-to-20 times as fast as theirs on average with smaller product of latency by area. See Table 6.6, where D denotes the digit size of multipliers working in the accumulative stage. In [33], the pairing accelerator for the elliptic curve over $\mathbb{F}_{3^{97}}$ with $k = 6$ is realized via a larger FPGA device, Xilinx XC2VP125 with 55,616 slices. Karatsuba-Ofman's method is used to construct the multiplier in $\mathbb{F}_{36m}$. To achieve the full power of parallel computation for such a large extension field multiplication, 18 multipliers in $\mathbb{F}_{3^m}$ are necessary. Due

to the limitation of resources, the digit sizes of underlying field multipliers cannot be large, so $D = 4$ is selected. The cost is 60% of 55616 slices for the multiplier in $\mathbb{F}_{36\times97}$. For comparison it costs only 10,722 slices for our multiplier CA in $\mathbb{F}_{24\times239}$, where $D = 16$. No exact results for the whole accelerator of pairing are provided in [33], but it is claimed that 100% of resources are utilized. So the cost is around 55,616 CLB slices. The operation frequency is 10MHz and it takes 850 μs for one pairing. Our pairing accelerator on the elliptic curve over $\mathbb{F}_{2^{239}}$ can run 20 times faster.

In [35], a smaller device, Xilinx XC2VP4FF672-6 with 4928 slices is chosen as the target device. The cubic field arithmetic is realized as an FPGA-based co-processor, which is controlled by a general-purpose processor, i.e., the top architecture is a stored-program machine (SPM), as described previously. The field arithmetic co-processor contains only one polynomial basis multiplier with digit size $D = 4$, so that multiplications are performed sequentially, i.e., at least 18 subfield multiplication rounds are necessary for each iteration of the accumulative multiplication. In contrast, our processor supports parallel computation of multiplications in $\mathbb{F}_{2^m}$ and the digit sizes we used are much larger than the ones in [35]. The latency for one pairing of our accelerator over $\mathbb{F}_{2^{239}}$ is only 41 μs. The latency of the design by Grabher et al. [35] is at least 432 μs. At the same time, our resource utilization is only 5 times larger as theirs.

Ronan et al. [65] have implemented Tate pairing accelerator on the hyperelliptic curve over binary field using the device Xilinx XC2VP125, which is the same as the one used in [33]. A smaller underlying field $\mathbb{F}_{2^{103}}$ and a larger embedding degree $k = 12$ are selected. However, in each iteration of accumulative multiplication stage, 16 multiplications in $\mathbb{F}_{2^m}$ and 1 multiplication in $\mathbb{F}_{2^{12m}}$ need to be computed. These 16 subfield multiplications must be completed before the accumulative multiplication in

$\mathbb{F}_{2^{12m}}$. Note that, in [65], one multiplication in $\mathbb{F}_{2^{12m}}$ is realized as 54 multiplications in $\mathbb{F}_{2^m}$ using Karatsuba's method. In our case, we choose a larger underlying field $\mathbb{F}_{2^{283}}$ and smaller embedding degree $k = 4$, so that the computation of accumulative multiplications becomes much simpler. Only 7 multiplications in $\mathbb{F}_{2^m}$ are involved each round and these multiplications can be performed in parallel. The shortest latency for one pairing of Ronan et al's is 749 μs and 43,986 slices are used. Our accelerator can run 12 times faster, whereas the resource utilization is only 37,803 slices, see Table 6.6.

6.7 Summary

We investigated the FPGA implementations of Tate pairing on supersingular elliptic curves over binary fields with embedding degree $k = 4$. We adopted two top algorithms computing pairing by which full power of parallel computations can be exploited with less resource utilization than in designs of other researchers. We performed security analysis for different curves over binary and cubic fields by which we can choose two binary fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ for our single FPGA implementations to achieve the same security strength as others. Besides the superiority of top algorithms, we also proposed an optimization method to obtain a compact design of the extension field multiplier CA by sharing some combinational circuits among several individual subfield multipliers in $\mathbb{F}_{2^m}$ in case of one shared operand. Furthermore we compared two schemes with a different number of multipliers in $\mathbb{F}_{2^m}$ for CA to get the optimal choice. The controller is implemented via hardwired logic to eliminate the overhead of instruction fetching and decoding, particularly for the simple operations such as additions and squarings. The technique simplifying the final exponentiations for both algorithms are addressed. Additionally, the implementations

are further optimized by optimal parameter choices for ALU. Compared with earlier implementations of Tate pairing computations based on the cubic and binary hyperelliptic shemes, our pairing processors are more efficient in terms of the product of latency by area. In particular, our accelerators can outperform earlier designs by a factor of 10-to-20 in terms of the total execution time.

Chapter 7: Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields

We choose SGI Altix-4700, a leading reconfigurable computer from Silicon Graphics, Inc., as the target platform for our experiments with Tate pairing cryptosystems over 8 selected binary fields ranging from $\mathbb{F}_{2^{239}}$ to $\mathbb{F}_{2^{557}}$ listed in Table 6.1. Finally, comparisons of performance and cost between pairing and scalar multiplication on Altix-4700 are demonstrated.

7.1 RASC Hardware/Software Overview

The Altix 4700 series [1] is a family of multiprocessor distributed shared memory (DSM) computer systems. All processors and memory are tied together into a single logic system with special crossbar switches. This combination of processors, memory, and crossbar switches constitutes the interconnect fabric called NUMAlink. RASC (Reconfigurable Application-Specific Computing) blades, which are powered by dual Xilinx Virtex-4 LX200 FPGAs, 80 MB QDR SRAM and dual NUMAlink 4 ports, can be integrated into SGI Altix 4700 system. In Fig. 7.1, the RASC hardware blade contains two computational FPGAs, two TIO ASICs which attach to the Altix system NUMAlink interconnect directly, and a loader FPGA for loading bitstreams into the computational FPGAs. Fig. 7.2(a) shows a block diagram of the RASC FPGA with core services and the algorithm block. The computational FPGAs connect directly

[1] Most of the contents related to Altix system is from [82]

into the NUMAlink fabric via Scalable System Ports (SSP) on the TIO ASICs. The FPGA is loaded with a bitstream that contains two major functional blocks:

- The reprogrammable algorithm
- The core services that facilitate running the algorithm

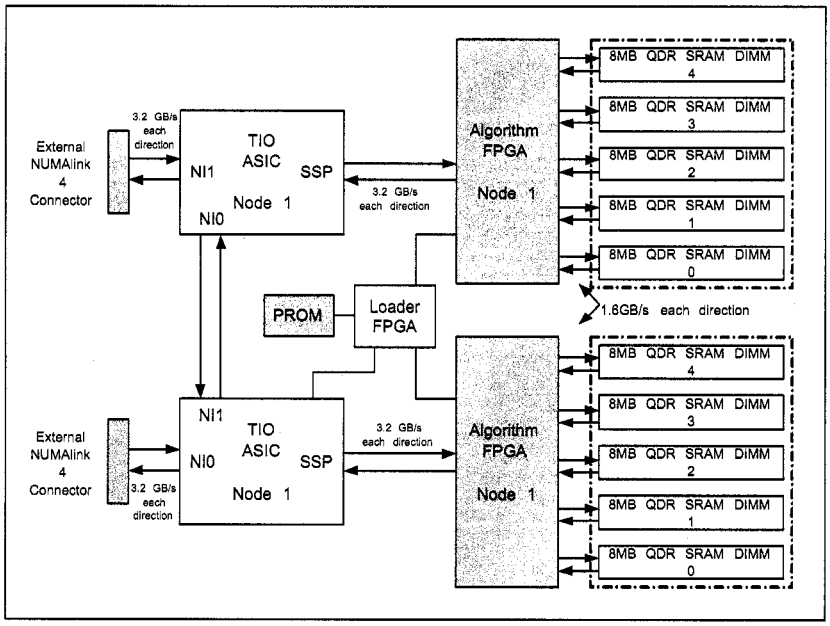
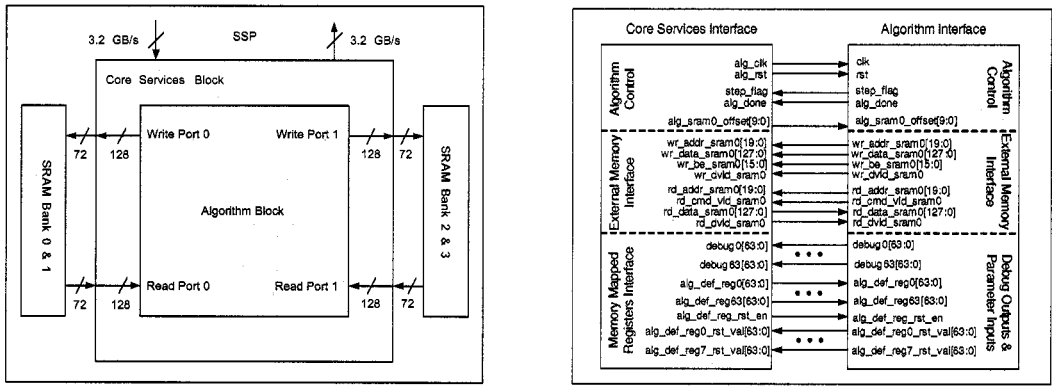


Figure 7.1: RASC blade hardware

There are three groups of interface signals between the algorithm block and the core services logic:

- **General algorithm control interface** provides the algorithm with a clock, reset, triggering and stepping control. When the algorithm is done, it should pulse its **alg_done** output shown in Fig. 7.2(b).
- **External memory interface** provides the algorithm with a simple interface to the read and write ports of two banks of SRAMs connected to the FPGA.



(a) RASC FPGA architecture

(b) Algorithm/core services interface diagram

Figure 7.2: The diagram of core services block and algorithm block

Access to the SRAM ports by the algorithm block is specified by macros defined during the synthesis of the FPGA bitstream.

- **Debug port interface** contains 64 64-bit debug outputs with respect to the algorithm block and 8 optional algorithm defined registers which can be written by software and read by hardware. The algorithm block makes its internal signals visible to the GNU debugger software by connecting them to one or more debug output signals. The use of those additional control registers is determined by algorithm need.

The RASC abstraction layer provides an application programming interface (API) for the kernel device driver and the RASC hardware. It is intended to provide a similar level of support for application development as the standard open/close/read/write/io_control calls for IO peripherals.

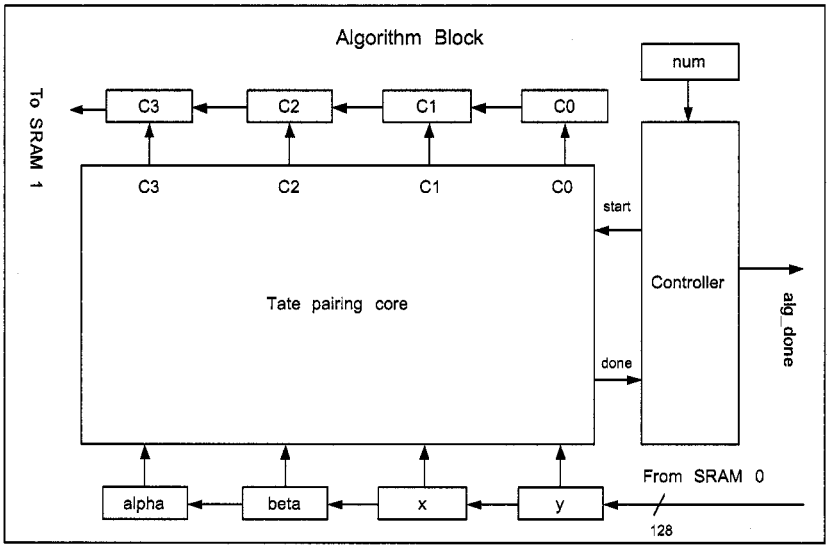


Figure 7.3: Tate pairing algorithm diagram

7.2 Port Tate Pairing Cryptosystems to SGI Altix 4700

We chose one of the two FPGAs, shown in Fig. 7.1, as our target device. We developed the generic IP core of Tate pairing in VHDL and its architecture has been proposed in previous section. To port our pairing design, we need consider the following issues.

- Specify the signals of the algorithm interface of the pairing core for different field sizes. For instance, if the underlying field size is 239, then each element of α, β, x, and y will be stored in two 128-bit registers considering that the word size of SRAM is 128 bits. We use serial-in-parallel-out registers for data input and parallel-in-serial-out registers for data output of the pairing core, shown in Fig. 7.3.

- Memory (SRAM) allocations. We use two SRAMs for inputs and outputs separately in our experiments. The SRAM for inputs are divided into 4 segments

for α, β, x, and y accordingly. The SRAM for outputs are also divided into 4 segments for c_3, c_2, c_1 and c_0. Each segment for both inputs and outputs contains *num* sets of data, where *num* denotes the number of pairing computations to be performed. All these data are generated by the software LiDIA. We use an algorithm-defined-register to store the value of *num* which can be accessed by the controller, so that the *alg_done* will be pulsed when all the computations are completed. One advantage of adopting two SRAMs is that it allows to read from and write to SRAMs simultaneously.

- The controller schedules computations and communications between external SRAM and algorithm FPGA. In our experiments, multiple pairings are computed sequentially, i.e., when one computation is done, a pulse named *done* in the pairing core will be asserted by which the controller will be informed to write the results to SRAM 1 and read next set of data from SRAM 0 at the same time.

The process of generating the bitstream for our Tate pairing cryptosystems is described as follows. There are two supported ways to create a bitstream. The first method is to implement the Algorithm FPGA using the pre-synthesized core services block EDIF netlist generated by Synplicity Synplify Pro. This allows faster synthesis time. The second method, which allows more optimization between the core services and the algorithm block, is to synthesize from the top-down, including re-synthesis of the core services block source codes. In our experiments, we choose the second method.

Apart from the bitstream file, we need two configuration files to register the algorithm to the device manager. The first, for core services, is generated automatically by the RASC tool. The second is the user space configuration file, generated by

the RASC extractor, which can extract information, such as the name of SRAM and algorithm-defined-registers referred to in the application software, from the comments of the VHDL codes of the algorithm block.

7.3 Performance and Cost of Tate Pairing Cryptosystem on SGI Altix 4700

Table 7.1: Performance and cost of Tate pairing on binary elliptic curves ported on SGI Altix 4700

Underlying fields	Digit sizes of multiplier in CA	Frequency (MHz)	Algorithm block resource utilization # slices / (%)	Total resource Utilization # slices / (%)	Latency of Software LiDIA (ms)	Latency of SGI Altix 4700 (μs)	Speed-up Altix vs. Software
$\mathbb{F}_{2^{239}}$	32	100	29,920 (34%)	41,641 (46%)	11.00	36.47	302
$\mathbb{F}_{2^{241}}$	32	100	30,286 (34%)	41,989 (47%)	11.10	37.23	298
$\mathbb{F}_{2^{283}}$	32	100	36,481 (41%)	48,202 (56%)	18.20	46.14	394
$\mathbb{F}_{2^{353}}$	32	100	45,543 (51%)	57,264 (64%)	26.90	67.26	400
$\mathbb{F}_{2^{367}}$	32	100	47,271 (53%)	58,992 (66%)	28.40	70.47	403
$\mathbb{F}_{2^{379}}$	32	100	49,819 (56%)	61,540 (69%)	30.07	72.70	414
$\mathbb{F}_{2^{457}}$	32	100	58,956 (66%)	70,677 (79%)	42.70	100.76	424
$\mathbb{F}_{2^{557}}$	8	66	37,931 (43%)	49,652 (55%)	83.8	675.50	124

We have ported the Tate pairing cryptosytem based on elliptic curves over the 8 binary fields listed in Table 6.1 to SGI Altix 4700. Placing and routing are completed by Xilinx ISE. As a result, our pairing processor can work at 100 MHz for the first 7 field sizes. As the field size become larger, e.g. 557, it is difficult for EDA tools to place and route to meet timing such as 100 MHz. Instead, we decreased the circuit complexity by using smaller digit size of multipliers and loosened the timing constraint to 66 MHz for the size 557. The FPGA device integrated into RASC blade is Xilinx XC4VLX200, with a total of 89,088 slices. As claimed before, the total resource utilization includes two parts, core services, which cost 11,721 slices (13%), and Tate

pairing algorithm block. In Table 7.1, the latency of hardware computation is end-to-end. The FPGA computational time is approximately several thousand clock cycles. In contrast, the overhead of communication between SRAMs and FPGAs is only 20-to-40 clock cycles. Second, the FPGA chip need to be configured only once for several hundred pairing computations and the configuration latency is approximately 2 ms. Therefore the end-to-end latency is almost equal to the FPGA computational time. Compared with the software (LiDIA) results, our pairing processor, ported on SGI reconfigurable platform, can run 120-to-420 times faster.

7.4 Performance and Cost Comparisons between Tate Pairing and ECC

It is noted that computations of both Tate pairing and scalar multiplication are similar as each other, since either of them contains mainly two stages: accumulative multiplication and final powering for pairing, or add-and-double and coordinate conversion for scalar multiplication. Most computations for both schemes concentrate on the first stage, and the second stage can be completed via several multiplications and one inversion. Though there exist several algorithms for both pairing and scalar multiplication, we can choose Algorithm 9 and López-Dahab algorithm for our comparisons without loss of correctness considering that both algorithms are fastest for their corresponding schemes by now. There are 6 multiplications for each add-and-double iteration of scalar multiplication, and all these multiplications can be completed as fast as possible in 2 multiplication rounds. However each accumulative multiplication iteration of pairing contains 7 multiplications, and all these multiplications can be performed simultaneously if 7 multipliers are used. To compare both schemes fairly

in term of hardware implementation efficiency, we ported both accelerators to Altix-4700. Four underlying fields are selected for both schemes at the same level of security strength, see Table 7.2. Additionally, the performance comparisons for software are also provided. We use the underlying field functions of LiDIA to implement the high level operations of pairing and scalar multiplications, and run both schemes on Intel Xeon 2.8 GHz, see Table 7.3.

Table 7.2: Hardware performance/cost comparisons for one operation between pairing and ECC, on SGI Altix-4700.

Tate Pairing					Elliptic Curve Cryptosystems					Speed-up
Field sizes	Digit sizes	Frequency (MHz)	CLB # slices (%)	Latency (μs)	Field sizes	Digit sizes	Frequency (MHz)	CLB # slices (%)	Latency (μs)	Pairing vs. ECC
239	32	100	41461 (46%)	36.47	233	64	100	41284 (46%)	43.84	1.20
283	32	100	48202 (54%)	46.14	283	64	100	49518 (55%)	58.87	1.28
457	32	100	70677 (79%)	100.76	409	64	100	64249 (72%)	101.39	1.10
557	8	66	59652 (55%)	675.50	571	32	66	62522 (70%)	388.90	0.58

Table 7.3: Software performance comparisons for one operation between pairing and ECC, implemented via LiDIA and run on Intel Xeon 2.8 GHz.

Tate Pairing		Elliptic Curve Cryptosystems		Speed-up
Field sizes	Latency (ms)	Field sizes	Latency (ms)	Pairing vs. ECC
239	11.00	233	11.36	1.03
283	18.20	283	21.85	1.20
457	42.70	409	45.19	1.06
557	83.80	571	116.80	1.39

In accordance with Table 7.2 and 7.3, we note that Tate pairing based cryptosystems are comparable with ECC in terms of software/hardware performance and cost.

7.5 Conclusion

We ported the Tate pairing cryptosystems for 8 field sizes, ranging from 239 to 557, to the reconfigurable computer, SGI Altix 4700. The issues related to data communications, memory allocation and controller for the reconfigurable approach of pairing cryptosystems are discussed in details. Consequently, our pairing processor on SGI reconfigurable platform can run 120-to-420 times faster than the software implementation based on LiDIA. To the author's knowledge, this is the first complete investigation of pairing cryptosystems over binary elliptic curves on a reconfigurable system. Furthermore, we performed comparisons between pairing and ECC in terms of software/hardware implementation efficiency. We note that the computational efficiency of pairing is comparable with ECC.

Chapter 8: Conclusions and Further Research

In this chapter, we will provide a summary for this dissertation as well as several recommendations for further research.

8.1 Conclusions

Today, RSA is widely used in public key cryptographic protocols. However, there is a trend that ECC is becoming a replacement for RSA in practical applications due to its efficiency in both software and hardware realizations. In recent years, pairing in ECC has drawn more attention of scholars because it can be exploited to build several cryptographic schemes that can not be constructed in any other way. This dissertation concentrates on efficient hardware architectures of elliptic curve based cryptosystems. We have investigated different target platforms, such as a single FPGA device and reconfigurable computers (SRC-6 and SGI Altix-4700) for ECC and Tate pairing.

We started our approach with efficient design for finite field arithmetic, mainly focusing on multipliers. We have shown how to exploit good algebraic properties to design efficient arithmetic components. For example we have adopted such as low/maximum Hamming weight irreducible polynomials for polynomial basis multipliers, and symmetry to decrease the complexity of timing and area for normal basis multipliers. We derived a novel composite field multiplier by choosing special irreducible polynomials to construct the ground field, so that the low circuit complexity can be achieved and the basis conversion can be completed easily. We performed

comparisons between the hybrid multipliers and the conventional digit-serial multipliers constructed via low Hamming weight irreducible polynomials. Both kinds of multipliers are ported to the same FPGA device, and the hybrid multipliers are 10%-to-33% more efficient than the conventional ones in terms of the product of latency by area.

We addressed several optimization techniques for ECC processors built using a single FPGA device. First, the top architecture is realized via hardwired logic instead of a stored-program machine due to the simplicity of group operations when using López-Dahab algorithm. Parallel computations are supported by adopting multiple underlying field multipliers in the point adder, the point doubler, and the coordinate converter. The optimal choices of the digit sizes of multipliers give a further optimization of the processor. We selected two fields $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$ for our experiments. As a result, our ECC processor can run 1.5-to-3 times faster than the processors designed by Gura et al. from Sun Microsystems with approximately the same cost. Alternatively, we chose reconfigurable computers as the second target platform for elliptic curve cryptosystems, because it allows parallel computations in hardware without the loss of software flexibility. Several partitioning schemes are investigated to determine the optimal boundary between software and hardware. We also built the macro libraries for binary field arithmetic and ECC for the SRC-6 reconfigurable computer, which allow the researchers to build the high level public key cryptographic protocols as well to perform cryptanalysis.

The third part of this dissertation is efficient hardware architecture of Tate pairing cryptosystems over binary elliptic curves. We are the first to port the binary elliptic case of Tate pairing to a single FPGA device. We derived a compact design of the accumulative multiplier in large extension field by sharing parts of the combinational

circuits among several individual underlying field multipliers. Second, two schemes for the accumulative multiplier are compared in terms of timing and area. The technique for simplifying the data-path for final powering is also addressed. We selected two underlying fields $\mathbb{F}_{2^{239}}$ and $\mathbb{F}_{2^{283}}$ for our FPGA implementations. Compared with the other two schemes, cubic elliptic and binary hyperelliptic cases, implemented by other research groups, our pairing processor can run 10-to-20 times faster at the same level of security strength. At the same time, an improvement in the product of latency by area by a factor between 12 and 46 has been achieved. In general, the speed-up of our implementation of the binary elliptic Tate pairing scheme compared to the other researchers' implementations of the cubic and hyperelliptic binary schemes is a function of multiple factors. These factors mainly include,

1. Computational efficiency at the group level, which can be evaluated by counting the number of subfield multiplications in each iteration of the accumulative multiplication.

2. Parallelization for the accumulative multiplication.

   - How many subfield multiplication rounds will be taken in case of full power parallelization.
   - How many subfield multipliers will be adopted to achieve the full power parallelization.

3. Efficiency of subfield arithmetic: one advantage of the cubic elliptic scheme is that the field size can be smaller than the binary elliptic scheme, however it takes $2m$ bits to represent an element in $\mathbb{F}_{3^m}$. In addition, the operations in $\mathbb{F}_3$ are much more complicated than the operations in $\mathbb{F}_2$.

4. Choice of top architecture: There are mainly two kinds of architectures, SPM and hardwired logic as claimed previously. Stored Program Machine (SPM) is suitable for the cubic elliptic and the binary hyperelliptic schemes because the group level operations are much more complicated than the binary elliptic case. On the other hand, hardwired logic can be chosen for the binary elliptic case due to the simplicity of its group operations.

Finally, we ported Tate pairing cryptosystems for the 8 field sizes ranging from 239 to 557 to the reconfigurable computer SGI Altix-4700. Compared with software implementation results via LiDIA, our pairing processor on this reconfigurable computer can run 120-to-420 times faster. We also compared performance and cost of pairing and ECC in both software and hardware. We found that the computational efficiency of pairing is comparable with ECC.

8.2 Further Research

There are two possible approaches for future work. The first is for Tate pairing cryptosystems. Since there have been mainly three different schemes: binary elliptic, cubic elliptic and hyper-elliptic schemes. It is necessary to decide the computational efficiency for hardware realizations at the same level of security strength. We have compared our results of binary elliptic case with the other two cases realized by other research groups. However the detailed determination of the relative influence of various factors would require an implementation of all three schemes by the same team, using identical assumptions, design techniques, optimization schemes, tools, and coding style.

Second, it is possible to develop a general processor for elliptic curve based cryptosystems, including ECC, HECC and pairing, via the combination of ASIC and

FPGA technologies. Since these public key cryptographic schemes share the same finite field arithmetic (for binary fields), it is necessary to obtain a finite field arithmetic unit, which can work for any underlying field. However, the operand sizes are variable and depend on the chosen fields, i.e., the circuits of the components such as multipliers can be totally different for different fields. Therefore such a finite field arithmetic unit can be realized via reconfigurable technology, e.g., FPGAs. On the other hand, the operations at group level can be completed by a stored-programmable machine, which can be realized via ASIC technology. Especially, such a stored-programmable machine is suitable for those public key cryptographic schemes, in which more complicated group operations are involved. Several considerations need to be taken into account. Some of these factors are listed below:

- **Clock Domain:** At least two kinds of clock signals need to be adopted for ASIC and FPGA parts separately. In general the ASIC parts can work at a faster frequency than the FPGA parts. Hence it is imperative to synchronize those signals passing from one clock domain to the other.

- **Instruction Set:** One instruction contains the operation code and the operands. Since the operand size is variable and more than 150 bits in general, the size of the memory allocated for each instruction can be variable depending on the underlying field size.

- **Data Bus Length:** In case of wider data bus, the cycles for fetching/stores data can be shortened, however, the area as well as the clock period will increase.

Here we only provided the sketch of this idea. To make it realized, more research work needs to be conducted.

Bibliography

Bibliography

[1] ANSI X9.62-1999. The Elliptic Curve Digital Signature Algorithm. Technical report, ANSI, 1999

[2] G.B. Agnew, R.C. Mullin, and S.A. Vanstone. An Implementation of Elliptic Curve Cryptography over $GF(2^{155})$, *IEEE Journal on Selected Areas in Communications*, vol. 11, 1993, pp. 804 - 813.

[3] G.B. Agnew, R.C. Mullin, I.M. Onyszchuk, and S.A. Vanstone. An implementation for a fast public-key cryptosystem, *Journal of Cryptology (Historical Archive)*, vol. 3, Issue 2, Jan 1991, pp. 63 - 79.

[4] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation* 48:203-209, 1987

[5] V. Miller. Uses of Elliptic Curves in Cryptography. In H.C. Williams, editor, *Advances in Cryptography - CRYPTO'85*, LNCS 218, pages 417-426, Berlin, Germany, 1986. Springer-Verlag.

[6] V. Miller, "Short programs for functions on curves," *unpublished manuscript*, 1986.

[7] IEEE. *IEEE P1363 Standard Specifications for Public Key Cryptography*, November 1999. Last Preliminary Draft.

[8] E.R. Berlekamp. *Algebraic Coding Theory*, McGraw-Hill Press, 1968.

[9] P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing based cryptosystems," *Crypto 2002, Lecture Notes in Computer Science*, vol. 2442, pp. 354–368, 2002.

[10] P. Barreto, S. Galbraith, C. OhEigeartaigh, and M. Scott, "Efficient pairing computation on supersingular abelian varieties," preprint *available at* http://eprint.iacr.org/2004/375.pdf, 2004.

[11] S. Galbraith, K. Harrison, and D. Soldera, "Implementing the Tate pairing," *ANTS 2002, Lecture Notes in Computer Science*, vol. 2369, pp. 324–337, 2002.

[12] W. Diffie and M.E. Hellman. New Directions in Cryptography, *IEEE Transactions on Information Theory*, IT-22: 644-654, 1976

[13] H. Fan and Y. Dai. Fast Bit-Parallel $GF(2^n)$ Multiplier for All Trinomials. *IEEE Trans. Computers*, vol. 54, pp. 485-490, 2005.

[14] S.T.J. Fenn, M.G. Parker, M. Benaissa, and D. Taylor. Bit-Serial Multiplication in $GF(2^m)$ Using Irreducible All-One Polynomials, *IEE Proc. Comput. Digit. Tech.*, vol. 144, No. 6, Novemver 1997, pp. 391 -393.

[15] S.T.J. Fenn, M. Benaissa, and D. Taylor. Bit-Serial Berlekamp-like Multipliers for $GF(2^m)$, *IEE Electronics Letters*, 1995 vol. 31, no. 22, pp. 1893 -1894.

[16] G. Frey. Applications of Arithmetic Geometry to Cryptographic Constructions. in D. Jungnickel and H. Niederreiter, editors, *Finite Fields and Applications* - 5, 128-161. Springer, 2001.

[17] FIPS 186-2, Digital Signature Standard (DSS), http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf.

[18] I.F. Blake, G. Seroussi, and N. P. Smart. Elliptic Curves in Cryptography, Cambridge University Press, 1999.

[19] I.F. Blake, G. Seroussi, and N.P. Smart. Advances in Elliptic Curve Cryptography, Cambridge University Press, 2005.

[20] X. Gao. Normal Bases over Finite Fields, Ph. D Thesis, University of Waterloo, 1993.

[21] R.M. Avanzi, C. Heuberger, and H. Prodinger. Scalar Multiplication on Koblitz Curves Using the Frobenius Endomorphism and its Combination with Point Halving Extensions and Mathematical Analysis, *http://finanz.math.tu-graz.ac.at/ prodinger/pdffiles/tauext.pdf*

[22] P. Gaudry, F. Hess, and N.P. Smart. Constructive and Destructive Facets of Weil Descent on Elliptic Curves, *Journal of Cryptology*, 15(1):19-46, 2002.

[23] J.S. Rose. A Course on Group Theory, Cambridge University Press, 1978.

[24] R. LiDl and H. Niederreiter. Introduction to Finite Fields and their Applications, Cambridge University Press, 1986.

[25] G. Seroussi. Table of Low-Weight Binary Irreducible Polynomials, Hewlett Packard, 1998.

[26] D. Hankerson, A.J. Menezes, and S. Vanstone. Guide to Elliptic Curve Cryptography, Springer-Verlag, 2003.

[27] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases, *Information and Computation*, vol. 78, 1988, pp. 171 - 177.

[28] K. Fong, D. Hankerson, J. López, and A. Menezes. Field Inversion and Point Halving Revisited, *IEEE Transactions on Computers*, vol. 53, pp. 1047-1059, 2004.

[29] F.R. Henríquez and Ç.K. Koç. Parallel Multipliers Based on Special Irreducible Pentanomials, *IEEE Trans. Computers*, vol. 52, no. 12, pp. 1535 - 1542, December 2003.

[30] Ç.K. Koç and B. Sunar. Low-Complexity Bit-Parallel Cannonical and Normal Basis Multipliers for a Class of Finite Fields, *IEEE Trans. Computers*, vol. 47, no. 3, pp. 353 - 356, March 1998.

[31] S. Kwon, K. Gaj, C.H. Kim, and C.P. Hong. Efficient Linear Array for Multiplication in $GF(2^m)$ Using a Normal Basis for Elliptic Curve Cryptography, *International Workshop on Cryptographic Hardware and Embedded Systems - CHES'04*, LNCS 3156, pp. 76 - 91.

[32] S. Kwon. Efficient Tate Pairing Computation for Elliptic Curves over Binary Fields, *ACISP'05*, LNCS 3574, pp. 134 - 145, 2005

[33] T. Kerins, W.P. Marnane, E.M. Popovici, and P.S.L.M. Barreto. Efficient Hardware for the Tate Pairing Calculation in Charateristic Three, *International Workshop on Cryptographic Hardware and Embedded Systems - CHES'05*, LNCS 3659, pp. 412 - 426.

[34] T. Kerins, E.M. Popovici, and W.P. Marnane, "Algorithms and architectures for use in FPGA implementations of Identity Based Encryption Schemes. In *Field Programmable Logic and Applications - FPL 2004* volume 3203 of *Lecture Notes in Computer Science*, pp. 74-83, Springer-Verlag 2004.

[35] P. Grabher and D. Page. Hardware Acceleration of the Tate Pairing in Charateristic Three, *International Workshop on Cryptographic Hardware and Embedded Systems - CHES'05*, LNCS 3659, pp. 398 - 411.

[36] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems Based on Pairing. In *2000 Symposium on Cryptography and Information Security - SCIS 2000*, 2000.

[37] A. Joux. A One Round Protocol for Tripartite Diffe-Hellman. In [**A-4**], 385-394.

[38] D. Bonel and M. Franklin. Identity Based Encryption from the Weil Pairing. *SIAM J. Comp.*, 32, 586-615, 2003.

[39] C.Y. Lee, E.H. Lu, and J.Y. Lee, Bit-Parallel Systolic Multipliers for $GF(2^m)$ Fields Defined by All-One and Equally Spaced Polynomials, *IEEE Transactions on Computers*, vol. 50, No. 5, May 2001, pp. 385 - 393.

[40] LiDIA, A C++ Library For Computational Number Theory, http://www.informatik.tu-darmstadt.de/TI/LiDIA.

[41] J. López and R. Dahab, Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation. *Cryptographic Hardware and Embedded Systems - CHES'99*, LNCS 1717, pp. 316 - 327.

[42] E. Mastrovito, VLSI Design for Multipliers for a Class of Fields $GF(2^k)$, *Lecture Notes in Computer Sciences 357*, Pages 297 - 309. Berlin: Springer-Verlag, Mar. 1989.

[43] R. Mullin, I. Onyszchuk, S. Vanstone and R. Wilson, "Optimal Normal Bases in $GF(p^n)$", *Discrete Applied Math.*, 22(1988/1989), 149-161.

[44] A.J. Menezes, I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone, and T. Yaghoobian, Applications of Finite Fields, Kluwer Academic Publishers, 1993.

[45] A.J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publisher, 1993.

[46] A. Shamir, "Identity-based cryptosystems and signature schemes," *Crypto 1985, Lecture Notes in Computer Science*, vol. 196, pp. 47–53, 1985.

[47] D. Boneh and M. Franklin, "Identity based encryption from the Weil pairing," *Crypto 2001, Lecture Notes in Computer Science*, vol. 2139, pp. 213–229, 2001.

[48] A.J. Menezes, An Introduction to Pairing-Based Cryptography, *http://www.math.uwaterloo.ca/ ajmeneze/publications/pairings.pdf*, 2005

[49] A.R. Masoleh and M.A. Hasan, Low Complexity Word-Level Sequential Normal Basis Multipliers, *IEEE Transactions on Computers*, vol. 54, No. 2, pp.98-110, 2005

[50] N.C. Nguyen, Implementation of Elliptic Curve Cryptosystems on a Reconfigurable Computer, Master Thesis, George Mason University, 2003

[51] S. Oh, C.H. Kim, J. Lim, and D.H. Cheon, Efficient Normal Basis Multipliers in Composite Fields, *IEEE Transactions on Computers*, vol. 49, No. 10, October 2000, pp. 1133 - 1138.

[52] O. Ahmadi and A. Menezes, Irreducible Polynomials of Maximum Weight, *http://www.math.uwaterloo.ca/ ajmeneze/publications/weightn.pdf.*

[53] J. Guajardo and C. Paar. Efficient Algorithms for Elliptic Curve Cryptosystems, *Advances in Cryptology - CRYPTO '97*, LNCS 1294, Pages 342 - 356, 1997

[54] C. Paar. A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields, *IEEE Transactions on Computers*, Vol 45, No. 7, Pages 856 - 861, July 1996

[55] C. Paar, P. Fleischmann and P. Soria-Rodriguez, Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents, *IEEE Trans. Computers*, vol. 48, pp. 1025-1034, 1999.

[56] G. Orlando, and C. Paar, A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$, *Cryptographic Hardware and Embedded Systems - CHES'00*, LNCS 1965, pages 41-56.

[57] S. Okada, N. Torii, K. Itoh, and M. Takenaka, Implementation of Elliptic Curve Cryptographic Coprocessor over $GF(2^m)$ on an FPGA, International Workshop on Cryptographic Hardware and Embedded Systems - CHES'00, LNCS 1965, pp. 25-40, 2000.

[58] P.H.W. Leong and I.K.H. Leung, A Microcoded Elliptic Curve Processor using FPGA Technology, *IEEE Trans. on VLSI*, 10(5), 2002.

[59] J. Lutz and A. Hasan, High Performance FPGA Based Elliptic Curve Cryptographyic Co-Processor *IEEE International Conference on Information Technology (ITCC)*, 2004.

[60] J. Solinas, Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, 19:195-249, 2000.

[61] L. Song and K.K. Parhi, Efficient Finite Field Serial/Parallel Multiplication, *Proceedings of International Conference on Application Specific Systems, Achitectures and Processors - ASAP'96*, pp. 72-82, 1996.

[62] A. Karatsuba and Y. Ofman, "Multiplication on many-digit numbers by automatic computers", *Translation in Physics-Doklady*, vol 7, pp. 595-596, 1963.

[63] N. Gura, S.C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, An End-to-End Systems Approach to Elliptic Curve Cryptography, *International Workshop on Cryptographic Hardware and Embedded Systems - CHES'02*, LNCS 2523, pp. 349 - 365.

[64] M.D. Ciletti, "Advanced digital design with the Verilog HDL", 2004.

[65] R. Ronan, C.O. Eigeartaigh, C. Murphy, M. Scott, T. Kerins, and W.P. Marnane, A Dedicated Processor for the Eta Pairing, *Cryptology ePrint Archive*, http://eprint.iacr.org/2005/330.pdf.

[66] R. Granger, D. Page, and M. Stam, "Hardware and software normal basis arithmetic for pairing based cryptography in characteristic three," preprint, *available at* http://eprint.iacr.org/ 2004/157.pdf, 2004.

[67] R. Granger, D. Page, and M. Stam, "On small characteristic algebraic tori in pairing based cryptography," preprint *available at* http://eprint.iacr.org/2004/132.pdf, 2004.

[68] I. Duursma and H. Lee, "Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$," *Asiacrypt 2003, Lecture Notes in Computer Science*, vol. 2894, pp. 111–123, 2003.

[69] S. Bajracharya, C. Shu, K. Gaj, and T. El-Ghazawi, Implementation of Elliptic Curve Cryptosystems over $GF(2^n)$ in Optimal Normal Basis on a Reconfigurable Computer, *Proc. IEEE 2004 Conference on Field Programmable Logic and Applications - FPL'04.*

[70] C. Shu, K. Gaj, and T. El-Ghazawi, Low Latency Elliptic Curve Cryptography Accelerators for NIST Curves over Binary Fields, *Proc. IEEE 2005 Conference on Field Programmable Technology - FPT'05.*

[71] LiDIA, A C++ Library For Computational Number Theory, http://www.informatik.tu-darmstadt.de/TI/LiDIA

[72] P. Saha, Building Libraries in the SRC environment, Technical Report, 2006.

[73] N.C. Nguyen, K. Gaj, D. Caliga, and T. El-Ghazawi, Implementation of Elliptic Curve Cryptosystems on a Reconfigurable Computer, *Proc. IEEE International Conference on Field-Programmable Technology - FPT'03.*

[74] L. Song and K.K. Parhi, Efficient Finite Field Serial/Parallel Multiplication, *Proceedings of International Conference on Application Specific Systems, Achitectures and Processors - ASAP'96*, pp. 72-82, 1996.

[75] J. Großschädl, A Low-Power Bit-Serial Multiplier For Finite Fields $GF(2^m)$, *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS'01*, vol. 4, pp. 37-40, 2001.

[76] SRC Inc. Web Page, *http://www.srccomp.com.*

[77] B. Sunar and C.K. Koç, Mastrovito Multiplier for All Trinomials, *IEEE Trans. Computers*, vol. 48, no. 5, May 1999, pp. 522 -527.

[78] B. Sunar, E. Savas, C.K. Koç, Constructing Composite Field Representations for Efficient Conversion, *IEEE Trans. Computers*, vol. 52, no. 11, November 2003, pp. 1391 - 1398.

[79] B. Sunar, An Efficient Basis Conversion Algorithm for Composite Fields with Given Representations, *IEEE Transas. Computers*, vol. 54, no. 8, August 2005, pp. 992 - 997.

[80] T. Wollinger, Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems, Ph. D Thesis, Bochum : Europäischer University, 2004.

[81] H. Wu, Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial Basis, *IEEE Trans. Computers*, vol. 42, pp. 750-758, 2002.

[82] Silicon Graphics, Inc., Reconfigurable Application-Specific Computer User's Guide.

Appendix A: VHDL Methodology for Polynomial Basis Multipliers

Here we provide two Functions in VHDL for pentanomial multipliers (trinomial multipliers can be constructed via the same method). Function 1 can be used to generate LFSR structures and AND-XOR arrays in the pentonomial digit-serial multipliers. And Function 2 can be used to generate the pentanomial bit-parallel multiplier.

Let the quadruple nm, k_3, k_2, k_1 denote the pentanomial. We suppose that $k_3 < \lfloor n/2 \rfloor$ considering that choosing $k_3 < \lfloor n/2 \rfloor$ is always possible.

Function 1 Reduction for $c = b \cdot \alpha^r$ implemented in VHDL

```
function reduce (
  b : in std_logic_vector(n-1 downto 0);
  r : in integer)
return std_logic_vector is variable c : std_logic_vector(n-1 downto 0)
          := (others=>'0');
begin
  if r=0 then
    c:=b;
  else
    c(n-1 downto r):=b(n-1-r downto 0);
    c(r-1 downto 0):=(others=>'0');
    for i in 1 to r loop
      c(i-1):=c(i-1) xor b(n-r+i-1);
      c(k3+i-1):=c(k3+i-1) xor b(n-r+i-1);
      c(k2+i-1):=c(k2+i-1) xor b(n-r+i-1);
      c(k1+i-1):=c(k1+i-1) xor b(n-r+i-1);
    end loop;
  end if;
  return c;
end function reduce;
```

Function 2 Pentanomial bit-parallel multiplication implemented in VHDL

```
function mul_parallel (
   a: in std_logic_vector(n-1 downto 0);
   b: in std_logic_vector(n-1 downto 0)) return std_logic_vector is
   variable c : std_logic_vector(n-1 downto 0) := (others=>'0');
   variable p_s : std_logic_vector(2*n-2 downto 0 := (others => '0');
begin
   for j in 0 to 2*n-2 loop
      if j<n then
         for t in 0 to j loop
            p_s(j):=p_s(j) xor (a(t) and b(j-t));
         end loop;
      else
         for t in j+1-n to n-1 loop
            p_s(j):=p_s(j) xor (a(t) and b(j-t));
         end loop;
      end if;
   end loop;
   for i in 0 to 2*n-2 loop
      if i<n then
         null;
      elsif i-n+k3<n then
         p_s(i-n):=p_s(i-n) xor p_s(i); p_s(i-n+k3):=p_s(i-n+k3) xor p_s(i);
         p_s(i-n+k2):=p_s(i-n+k2) xor p_s(i);
         p_s(i-n+k1):=p_s(i-n+k1) xor p_s(i);
      elsif i-n+k2<n then
         p_s(i-n):=p_s(i-n) xor p_s(i); p_s(i-n+k2):=p_s(i-n+k2) xor p_s(i);
         p_s(i-n+k1):=p_s(i-n+k1) xor p_s(i);
         p_s(i-2*n+k3):=p_s(i-2*n+k3) xor p_s(i);
         p_s(i-2*n+2*k3):=p_s(i-2*n+2*k3) xor p_s(i);
         p_s(i-2*n+k3+k2):=p_s(i-2*n+k3+k2) xor p_s(i);
         p_s(i-2*n+k3+k1):=p_s(i-2*n+k3+k1) xor p_s(i);
      elsif i-n+k1<n then
         p_s(i-n):=p_s(i-n) xor p_s(i); p_s(i-n+k1):=p_s(i-n+k1) xor p_s(i);
         p_s(i-2*n+k3):=p_s(i-2*n+k3) xor p_s(i);
         p_s(i-2*n+2*k3):=p_s(i-2*n+2*k3) xor p_s(i);
         p_s(i-2*n+k3+k1):=p_s(i-2*n+k3+k1) xor p_s(i);
         p_s(i-2*n+k2):=p_s(i-2*n+k2) xor p_s(i);
         p_s(i-2*n+2*k2):=p_s(i-2*n+2*k2) xor p_s(i);
         p_s(i-2*n+k2+k1):=p_s(i-2*n+k2+k1) xor p_s(i);
      else
         p_s(i-n):=p_s(i-n) xor p_s(i); p_s(i-2*n+k3):=p_s(i-2*n+k3) xor p_s(i);
         p_s(i-2*n+2*k3):=p_s(i-2*n+2*k3) xor p_s(i);
         p_s(i-2*n+k2):=p_s(i-2*n+k2) xor p_s(i);
         p_s(i-2*n+2*k2):=p_s(i-2*n+2*k2) xor p_s(i);
         p_s(i-2*n+k1):=p_s(i-2*n+k1) xor p_s(i);
         p_s(i-2*n+2*k1):=p_s(i-2*n+2*k1) xor p_s(i);
      end if;
   end loop;
   c := p_s(n-1 downto 0);
   return c;
end function mul_parallel;
```

Curriculum Vitae

Chang Shu received his Bachelor of Science Degree and Master of Science Degree in Electrical Engineering from Tsinghua University and Graduate School of Chinese Academy of Sciences, China, in 1999 and 2002, respectively. He was a research assistant in the State Key Lab of Information Security at the Graduate School, Chinese Academy of Sciences from 1999 to 2002.

He commenced his Ph.D. studies in the Department of Electrical & Computer Engineering at George Mason University in 8/2002, where he served as a research assistant, developing the cryptographic macro library in VHDL for the reconfigurable computer SRC-6, and a teaching assistant for several undergraduate/graduate courses.

He worked as a digital design intern at ST Microelectronics, Inc. from 6/2004 to 9/2004 and participated in various stages of the ASIC design cycle for Elliptic Curve Cryptosystems. From 6/2006 to 8/2006, he worked as a reconfigurable application intern at Silicon Graphics, Inc., where he developed the generic IP core of pairing cryptosystems on binary elliptic curves.

His research interests include finite field arithmetic, hardware/software implementations of elliptic curve based cryptosystems, and reconfigurable computing for scientific algorithms.

Publications

1. S. Bajracharya, C. Shu, K. Gaj, and T. El-Ghazawi, Implementation of Elliptic Curve Cryptosystems over $GF(2^n)$ in Optimal Normal Basis on a Reconfigurable Computer, *14th International Conference on Field Programmable Logic and Applications FPL04*, Antwerp, Belgium, Aug. 2004.

2. C. Shu, K. Gaj, and T. El-Ghazawi, Low Latency Elliptic Curve Cryptography Accelerator for NIST Curves over Binary Fields, *IEEE International Conference on Field Programmable Technology FPT05*, Singapore, Dec. 2005.

3. C. Shu, S. Kwon, and K. Gaj, FPGA Accelerated Tate Pairing Based Cryptosytems over Binary Fields, *IEEE International Conference on Field Programmable Technology FPT06*, Thailand, Dec. 2006.

4. C. Shu, S. Kwon, and K. Gaj, FPGA Accelerated Multipliers over Binary Composite Fields Constructed via Low Hamming Weight Irreducible Polynomials, *submitted to IEE Proceeding of Computer & Digital Techniques.*

5. C. Shu, S. Kwon, and K. Gaj A Hybrid Multiplier of Binary Composite Field with Efficient Basis Conversion, *submitted to IEEE Transactions on Computers.*

6. C. Shu, S. Kwon, and K. Gaj Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields *submitted to IEEE Transactions on Computers.*