Efficient Computations in Finite Fields with Cryptographic Significance

by

Huapeng Wu

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Electrical Engineering

Waterloo, Ontario, Canada, 1998

©Huapeng Wu 1998

National Library of Canada

Acquisitions and Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada

Acquisitions et services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-38293-1

Canada

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

The increasing use of cryptographic techniques in various communication and computer systems has inspired many researchers to find ways to perform fast computations over finite fields, especially over large finite fields of characteristic two. The central theme of the thesis is an investigation of finite field computations and their architectures, such as multiplication and exponentiation. The computation of point multiples on elliptic curves is also discussed.

Three new types of finite field multipliers are given in the thesis. New bit-serial and bit-parallel multipliers using redundant bases, which is a modification of certain normal bases, are proposed. Parallel weakly dual basis multipliers are presented in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ for any prime power q. For the polynomial basis, bit-parallel multiplication and squaring are discussed and their low-complexity constructions are investigated.

Exponentiation of a primitive element in finite fields is also considered. Structures for exponentiations using different representations of the exponent are given for both the polynomial basis and its weakly dual basis. A new signed-digit representation is proposed and used for the computation of $m_1P_1 + m_2P_2 + \cdots + m_kP_k$ for elliptic curve cryptosystems. The performance analysis for such computations on elliptic curves using the sliding window method is also given. Other related results include closed form expressions for the average Hamming weight and length of signed-digit representations, which correspond to the numbers of multiplications and squarings in an exponentiation operation.

Acknowledgements

I deeply appreciate my co-supervisor Professor M. Anwar Hasan, for introducing me to an extremely rich and beautiful area of research, for his competent and consistent guidance throughout my work, for the considerable effort and time he has given me, and for his financial support for my graduate studies at Waterloo. I am very grateful to Professor Ian F. Blake, my co-supervisor, for more than anything else providing me with confidence in my work. I would aspire to one day be like him. I thank them for introducing me so smoothly to the subject of Galois fields.

I would like to express my gratitude to the follwoing persons for their help and encouragement: Professor Gordon Agnew, who gives me a sense of the differece between finite field architecture and its VLSI implementation; Professor Scott Vanstone, every time I talk to him, I have a new future work topic; Professor Shuhong Gao, without his help I could not finish the third chapter of the thesis; Professor Alfred Menezes, with both his book and his comment I could work out Chapter 10 of this thesis; Dr. Minghua Qu, who lets me know how a mathematician thinks about implementation of cryptosystems.

I would like to thank all my friends who have provided me with the joy and warmth needed to keep me going through out my Ph.D. program.

Finally, my deepest gratitude goes to my mother for her unconditional love and support.

I am grateful to my examiners, Professor Tho Le-Ngoc, Professor Scott Vanstone, Professor Gordon Agnew, and Professor Amir. Khandani, for their comments and corrections.

Contents

1 Introduction 1

1.1 Motivation 1

1.1.1 Finite fields and their applications 1

1.1.2 Cryptography and finite field computations 2

1.2 VLSI Architecture and Complexities 4

1.2.1 Parallel VLSI architecture 4

1.2.2 Complexity measures 5

1.3 Thesis Outline 6

1.4 Research Contributions 7

2 Mathematical Preliminaries 8

2.1 Finite Fields 8

2.1.1 Groups, rings and fields 8

2.1.2 Extension fields 10

2.1.3 Polynomials over finite fields 11

2.1.4 Roots of unity 13

2.1.5 Finite field bases and arithmetic operations 14

2.1.6 Elliptic curves over finite fields 17

3 Normal Basis Multipliers 20

3.1 Gauss Period and Normal Basis 20

3.2 Bit-Serial NB Multipliers 22

3.2.1 Previous implementations 22

3.2.2 Bases generated with Gauss period of type $(m, 1)$ 24

3.2.3 Bases generated with Gauss period of type (m, k) 27

3.3 Bit Parallel NB Multipliers 29

3.3.1 Previous implementations 29

3.3.2 New bit-parallel multipliers 31

3.4 Discussions on Redundant Basis 32

4 Parallel Dual Basis Multipliers 34

4.1 A Brief Review of Dual Basis Multipliers 34

4.2 Parallel Multipliers in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ 35

4.2.1 WDB multiplication 35

4.2.2 A complexity bound 38

4.2.3 Algorithm and architecture 40

4.2.4 Architecture with reduced time delay 42

4.3 ESP Based Bit-Parallel Multiplier in $\mathbb{F}_{2^m}$ 44

4.3.1 Algorithm 44

4.3.2 Implementation 46

4.4 Trinomial Based Bit-Parallel Multiplier in $\mathbb{F}_{2^m}$ 51

4.4.1 Analysis of multiplier complexity 51

4.4.2 Construction with reduced propagation delay 53

4.5 Basis Conversion 58

4.6 Chapter Summary and Discussions 62

5 Parallel Polynomial Basis Multipliers 64

5.1 Polynomial Basis Multiplication in $\mathbb{F}_{2^m}$ 64

5.1.1 Polynomial multiplication 65

5.1.2 Reduction modulo a polynomial 66

5.2 Bit Parallel PB Multipliers 70

5.2.1 Previous implementations 70

5.2.2 Implementation with new method of modulo reduction 71

5.3 Low Complexity PB Squarer in $\mathbb{F}_{2^m}$ 76

5.3.1 Complexity of PB squaring in $\mathbb{F}_{2^m}$ 76

5.3.2 Implementation 83

6 Analysis of SD Form Exponents 84

6.1 Exponent Representations 84

6.1.1 Using conventional number systems 84

6.1.2 Using redundant number systems 85

6.2 Average Hamming Weight and Length of Non-adjacent Form 87

6.2.1 Hamming weight of radix-r NAF 87

6.2.2 Length of radix-r NAF 93

7 Realization of Finite Field Exponentiation 97

7.1 Brief Review 97

7.2 Efficient Representations of Exponent 99

7.2.1 Algorithm 99

7.2.2 Features of minimal radix-4 SD form 100

7.3 Exponentiation Algorithms 104

7.4 Implementation Using Polynomial Basis 105

7.5 Realization Using Weakly Dual Basis 115

7.6 Comparisons 120

7.7 Chapter Summary 122

8 Computations for Elliptic Curve Cryptosystems **125**

8.1 Introduction 125

8.2 Sliding Window Method for Non-Supersingular Curves 127

8.2.1 Modified Shamir's method 127

8.2.2 General sliding window methods 129

8.2.3 Window method with efficient computation of $2^l P$ 133

8.2.4 Results and features 133

8.3 Algorithms using a New SD number Representation 136

8.3.1 A New SD representation with fewer zero runs 136

8.3.2 Window method with the FZR form 140

8.4 Window Method for Koblitz Curves 140

8.5 Numerical Results 143

8.5.1 Comparison of computing mP 143

8.5.2 Comparison of computing $mP_1 + rP_2$ 144

8.6 Chapter Summary 146

9 Summary, Discussions and Future Work **149**

Bibliography **152**

List of Tables

2.1 r-ESP for $m \leqslant 1000$. 13

3.1 Comparison of bit-serial multipliers using type I ONB and RB. 26

3.2 Comparison of bit-serial multipliers using type II ONB and RB. 28

3.3 Comparison of bit-serial multipliers using NB and RB. 29

3.4 Comparison of bit-parallel multipliers using type I ONB and RB. 32

4.1 (a) Comparison of multipliers based on AOP. (b) Comparison of multipliers based on r-ESP ($r > 1$). 63

7.1 Radix-4 SD number encoding using 2-bit Booth algorithm [8]. 102

7.2 Table for bit-parallel fourth power complexity when $f(x)$ is a primitive trinomial or pentanomial ($k \leq \frac{m}{2}$) whose degree is a Mersenne exponent. 107

8.1 Comparison of the various binary number representations 139

8.2 Comparison of the algorithms for computing mP. 143

8.3 Comparison of the algorithms for computing mP (P fixed). 144

8.4 Comparison of the numbers of $16P$, $8P$, $4P$ and $2P$ operations required to compute mP for non-supersingular curves. 145

8.5 Comparison of algorithms for computing $mP_1 + rP_2$ 146

8.6 Comparison of the algorithms for computing $mP_1 + rP_2$ (P_1 fixed). . . 147

8.7 Comparison of algorithms for computing $mP_1 + rP_2$ (both P_1 and P_2 fixed). 148
8.8 Comparison of the numbers of $16P$, $8P$, $4P$ and $2P$ operations required to compute $mP_1 + rP_2$ for non-supersingular curves. 148

List of Figures

2.1 Subfields of $\mathbb{F}_{30}$. 11

3.1 Bit serial multiplier using redundant basis when there is a type I ONB. . 26

3.2 Bit serial multiplier using redundant basis. 28

3.3 Parallelization of the bit-serial multiplier using the redundant basis. . . 31

4.1 Architecture of a parallel multiplier in F_{7^5} over F_7 when $f(x) = x^5 + 2x^3 + 5 \in F_7[x]$. 45

4.2 Architecture for a parallel multiplier when $f(x)$ is an r-ESP. 49

4.3 Multiplier structure when $f(x) = x^6 + x^3 + 1$. 50

4.4 Multiplier structure when $f(x) = x^m + x^k + 1, 1 < k < \frac{m}{2}$. 55

4.5 Architecture of WDB multiplier when $f(x) = x^m + x + 1$. 58

5.1 Indication of the relation between terms in (5.9a) and (5.9b). 73

7.1 Bidirectional LFSR (BiLFSR) for multiplying a field element with a primitive root α and its inverse α^{-1}. 109

7.2 The Structure EXP1 for computing α^H with H being converted to a minimal binary SD number. 109

7.3 (a) Circuits for multipling a field element with α^2 and α^{-2}; (b) Circuits for multipling a field element with $\alpha^{\pm 1}$ and $\alpha^{\pm 2}$. 112

7.4 The Structure EXP2 for α^H with H represented as a minimal radix-4 SD number. 113
7.5 EXP1 with a converter that performs the canonical recoding. 113
7.6 (a) EXP2 with a converter that performs the extended canonical recoding; (b) Circuits for the extended canonical recoding (Algorithm 7.1). . . . 114
7.7 (a) The BiLFSR: LFSR for multiplying by $\alpha^{\pm 1}$; (b) The LFSR for multiplying by $\alpha^{\pm 2}$; (c) The XBiLFSR: LFSR for multiplying by $\alpha^{\pm 1}$and $\alpha^{\pm 2}$. 123
7.8 System diagram for exponentiation when the weakly dual basis is used. 124

8.1 Window and block 130

Chapter 1

Introduction

1.1 Motivation

1.1.1 Finite fields and their applications

A finite (or Galois) field is an algebraic structure with a finite number of elements where we can perform addition, subtraction, multiplication and division. A finite field of q elements is denoted by $\mathbb{F}_q$ or GF(q). Finite fields have been applied to finite geometries, optimal designs, linear recurring sequences, linear modular systems, and other mathematical disciplines. Over recent decades, finite fields have also gained wide spread practical applications.

For example, the study and design of systems for secret communication is the subject of cryptography, and finite fields are involved in many modern cryptographic systems or cryptosystems. More about the use of finite fields in cryptography is discussed in § 1.1.2.

The theory of finite fields and the theory of polynomials over finite fields have been applied to the design of good codes and efficient decoding methods. For example, BCH (Bose-Choudhuri-Hocquenghem) [37] codes and the related RS (Reed-Solomon) [66]

codes are widely used codes. A number of efficient algorithms are available for encoding and decoding these codes [11, 13], where finite fields are involved.

Finite fields can also be used in digital signal processing that makes extensive use of the discrete Fourier transform, convolution, and solution to Toeplitz systems of equations. The use of appropriate finite fields may simplify or speed-up these computations [14]. In certain digital testing schemes for integrated circuits, it is very important to select a good primitive polynomial over a finite field to be used in the implementation of the tester [31].

1.1.2 Cryptography and finite field computations

Communication networking or information highway has been one of the greatest technical achievements in this century, which has greatly promoted electronic information exchange, electronic commerce, and emerging electronic banking. The security requirements for these facilitations can include *privacy* or *confidentiality, user authentication, data authentication* or *integrity, key management*, and *non-repudiation*. Researchers have been developing tools for cryptosystems to meet these requirements, for instance, encryption/decryption system for privacy, hashing function for data integrity, and digital signature scheme for authentication and non-repudiation.

Cryptosystems can be categorized into public (or asymmetrical) key systems and private (or symmetrical) key systems. Public-key cryptosystems are such that they are based on some "hard" computational problems, for example, factoring the product of two large primes (*i.e.*, integer factorization cryptosystems) or the discrete logarithm in a finite cyclic group (*i.e.*, discrete logarithm cryptosystems and elliptic curve cryptosystems). Many of them work in $\mathbb{Z}_n^*$ or over $\mathbb{F}_q$, where n and q are usually large enough to promise adequate security of the system against possible cryptoanalytic attacks. For both a secure and an efficient implementation of the latter type of cryptosystems, the cyclic group G

should be chosen to satisfy the following two conditions:

- For efficiency, the group operation in G should be "easy" to apply;

- For security, the discrete logarithm problem in the cyclic group G should be "hard".

The group $G = \mathbb{F}_{2^m}^*$ is considered since operations in $\mathbb{F}_{2^m}$ are easy and more suitable for implementation. Besides the discrete logarithm cryptosystems over $\mathbb{F}_{2^n}$, the elliptic curve cryptosystems, which utilize the group of points on an elliptic curve over a field, can also be realized using finite fields of characteristic two. These groups are generally used to take advantage of their efficiency over multiprecision arithmetic for large prime fields. The elliptic curve cryptosystems also have the advantage of their high cryptographic strength relative to the key size, and thus they are especially attractive in applications such as the financial industry, smart cards and wireless areas where power and bandwidth are limited.

The computations in $\mathbb{F}_{2^n}$ with applications to cryptography have the following requirements:

- In logarithm and elliptic curve cryptosystems, the primary operation performed is exponentiation or addition of two points on an elliptic curve. Both operations involve extensive finite field multiplications especially when high indices are required for security considerations. As a result, there is a need for high-speed multipliers.

- Compared to software implementations, it is believed that a hardware implementation of a security algorithm may better resist attackers from tampering with the system, besides its advantage on speed it has.

- Network security protocols require high computation speed. For example, computation time probably should be within a second for on-line identification on the Internet.

There have been several excellent books and dissertations on finite fields, for example, [47, 55, 40, 52, 24]. Books on cryptography or number theory with special treatment of algorithms and computations over finite fields include [57, 73, 54, 9]. Some of the dissertations on efficient finite field computations and their VLSI implementations are [74, 50, 33, 63, 39].

In this thesis, we focus on finite field operations which are commonly used in cryptosystems, such as multiplication, exponentiation and computation of point multiples on elliptic curves.

1.2 VLSI Architecture and Complexities

1.2.1 Parallel VLSI architecture

Hardware architectures performing finite field arithmetic operations can be generally categorized into three classes: *parallel*, *serial*, and *hybrid* architectures. Correspondingly, architectures over the field $\mathbb{F}_2$ are usually classified as bit-parallel, bit-serial and hybrid architectures [50]. Generally speaking, bit-parallel finite field architectures discussed in this thesis may be characterized by the following features:

1. Consist of only combinational logic, and no memories required;
2. Have parallel input and output ports;
3. No sequential logic involved and thus no clock required.

In contrast, bit-serial architectures use sequential logic and have clock inputs, while both the input and output ports can be either parallel or serial type. Hybrid architectures are usually referred to those architectures that consist of a number of bit-parallel modules which are serially connected using sequential logic, or a number of bit-serial modules

working in parallel. For instance, bit-parallel word-serial architectures and bit-serial word-parallel architectures.

1.2.2 Complexity measures

Two parameters of a VLSI architecture are of vital importance, its *size* (or *space*) *complexity* and *time complexity*.

In this thesis, the complexity of the algorithms for computations in finite fields are usually evaluated by the number of operations in the ground field. If the characteristic of the underlying field is 2, then the count of bit operations is given as a complexity measure. An addition or multiplication in $\mathbb{F}_2$ can be realized with a two-input XOR or a two-input AND gate, respectively.[1] Consequently, the space complexity of the mapped VLSI architecture can be expressed in terms of the numbers of AND and XOR gates. In the sequel, the numbers of AND and XOR gates needed in an architecture are denoted by C_A and C_X, respectively. Paar and Lange have shown that this measure for size complexity is a good estimate for the chip area if it is implemented in VLSI technologies [64]. If the exact number of the logic gates or elementary logic cells is not available, we also use $O(n), O(n^2)$, *etc.*, to give the asymptotic size complexity, where n can be a measure of the field size.

The time complexity of a bit-parallel architecture consisting of only combinational logic is measured by the maximal propagation delay in the path from the input to the output. If we denote the time delays caused by one AND gate or one XOR gate by T_A or T_X, respectively, then the time complexity which is referred to as C_T is given as a sum of multiples of T_A and T_X.

For bit-serial or other types of architectures with clock inputs, the time complexity is decided by both the clock period and the number of clock cycles required to complete

[1] In the sequel, all XOR gates and AND gates are assumed to have only two inputs.

the operation. Then it could be understood that a computation requires a time delay not less than

$$\text{\# of clock cycles} \times \text{the clock period.}$$

1.3 Thesis Outline

The remainder of the thesis is organized as follows.

In Chapter 2, some mathematical preliminaries are reviewed. Definitions and fundamental theorems in finite fields which relate to the subsequent chapters are given.

Chapters 3, 4 and 5 discuss finite field multipliers and their implementations. Redundant basis, a modification of certain normal bases, is presented in Chapter 3. New bit-serial and bit-parallel multipliers using this basis are also developed in this chapter. Chapter 4 discusses parallel multipliers using weakly dual bases. New implementations of these multipliers with reduced propagation delay are given over $\mathbb{F}_q$ and over $\mathbb{F}_2$. In Chapter 5, first complexity bounds on bit-parallel polynomial basis multiplication and squaring are given, then new low-complexity bit-parallel multipliers and squarers over $\mathbb{F}_{2^m}$ are presented.

Chapters 6, 7, and 8 discuss exponentiation over finite fields and point multiples on elliptic curves. Methods of efficiently representing an exponent is discussed in Chapter 6. Efficient implementation of finite field exponentiation is presented in Chapter 7, where a novel linear feedback shift register is used to efficiently realize multiplication with multiple multiplicands. Two exponentiation architectures, using polynomial basis and dual basis, are also proposed. In Chapter 8, a new signed-digit representation is proposed and used in the general sliding-window algorithm to compute point multiples on elliptic curves. Various extensions of sliding-window method are discussed and their performance analyses are given.

A summary of our work is presented in Chapter 9, where we also give some suggestions on possible future work.

1.4 Research Contributions

The major contributions in this thesis are the formulation of algorithms and the development of architectures for finite field multiplication, exponentiation, and computing point multiples on an elliptic curve. Some specific contributions are as follows:

- new bit-serial and bit-parallel multipliers using redundant basis,
- new parallel multipliers using weakly dual basis in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$,
- new bit-parallel multipliers and squaring using polynomial basis,
- new bit-parallel squarings using polynomial basis,
- Closed form expressions for average number of nonzeros and average length of the NAF,
- New architectures of exponentiation using polynomial and dual bases,
- A minimal signed-digit representation with fewer zero runs.

Chapter 2

Mathematical Preliminaries

This chapter gives some preliminaries on finite fields to facilitate the discussions of the chapters to follow. A brief introduction of *elliptic curves* over finite fields is also provided. For detailed treatment of finite fields and elliptic curves, the readers are referred to [47, 52, 54].

2.1 Finite Fields

2.1.1 Groups, rings and fields

Definition 2.1 [47] A *group* is a set G together with a binary operation $*$ on G such that:

1. Binary operator $*$ is associative; *i.e.*, for any $a, b, c \in G$, $a * (b * c) = (a * b) * c$.

2. There is an identity (or unity) element e in G such that for all $a \in G$, $a*e = e*a = a$.

3. For each $a \in G$, there exists an inverse element $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$. □

The operation $*$ may be denoted as either ordinary multiplication $(\cdot)$ or ordinary addition $(+)$. If for all $a, b \in G, a * b = b * a$, then G is called *abelian group*. A group containing a finite number of elements is called a *finite group*. The number of elements in a finite group G is called its *order* and denoted as $|G|$.

A multiplicative group G is said to be *cyclic* if there is an element $a \in G$ such that for any $b \in G$ there is some integer j with $b = a^j$. Such an element a is called a *generator* of G and we write $G = \langle a \rangle$. If $\langle a \rangle$ is finite, then its order is called *the order* of the element a. A subset H of the group G is a *subgroup* of G if H is itself a group with respect to the operation of G.

A mapping $f : G_1 \Rightarrow G_2$ of the group G_1 into the group G_2 is called a *homomorphism* of G_1 into G_2 if f preserves the operation of G_1. If f is a one-to-one and onto homomorphism of G_1 onto G_2, then f is called an *isomorphism* and we say that G_1 and G_2 are *isomorphic*.

Definition 2.2 [47] A *ring* $(R, +, \cdot)$ is a set R together with two binary operations denoted by $+$ and $\cdot$ such that:

1. R is an abelian group with respect to $+$.
2. Binary operator $\cdot$ is associative, *i.e.*, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in R$.
3. The distribution law holds; that is, for all $a, b, c \in R$ we have $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$. □

The identity element of the abelian group R with respect to addition is called *zero element*, while the multiplicative identity (if it exists) is called *identity*. A ring is called *commutative* if the operator $\cdot$ is commutative. A ring is called *an integral domain* if it is a commutative ring with identity $e \neq 0$ in which $ab = 0$ implies either $a = 0$ or $b = 0$.

A subset S of a ring R is called a subring of R provided that S is closed under $+$ and $\cdot$ and forms a ring under these operations. A subring J of R is called *an ideal* provided that for all $a \in J$ and $r \in R$ we have $ar \in J$ and $ra \in J$. Let R be a

commutative ring, then an ideal J of R is said to be *principal* if there is an $a \in R$ such that $J = (a) = \{ra + na : r \in R, n \in \mathbb{Z}\}$.

If n is the least positive integer such that $nr = 0$ for every r belonging to finite ring R, then n is called the *characteristic* of R and R is said to have characteristic n.

Definition 2.3 [47] A *finite field* F is a set F together with two binary operations denoted by $+$ and $\cdot$ such that

1. F is a commutative ring under $+$ and $\cdot$.
2. Nonzero elements of F form a group under $\cdot$. □

2.1.2 Extension fields

A subset K of a field F is called a *subfield* of F provided K is a field under the operations of F. In this context, F is called an *extension field* of K. If F, considered as a vector space over K, is of finite-dimension, then F is called a *finite extension* of K. The dimension of the vector space F over K is then called the *degree* of F over K, in symbols $[F : K]$. If L is a finite extension of F, then L is a finite extension of K with $[L : K] = [L : F][F : K]$. Clearly, if there are q elements in F then L has q^n elements, where $n = [L : F]$.

A field containing no proper subfield is called a *prime field.* Let F be a finite field, then F has $q = p^n$ elements, (denoted by $\mathbb{F}_q$), where the prime p is the characteristic of F and n is the degree of F over its prime field. Every subfield of $\mathbb{F}_q$ has order p^m, where m is a positive divisor of n. Conversely, if m is a positive divisor of n, then there is exactly one subfield of $\mathbb{F}_q$ with p^m elements. For example, the subfields of $\mathbb{F}_{30}$ can be determined by finding all divisors of 30. The containment relations between the subfields are illustrated in Figure 2.1.

The number of elements of a finite field F can only be equal to a prime power. Given a prime power q, there exists one and only one finite field $\mathbb{F}_q$, up to an isomorphism.

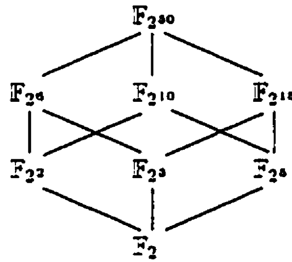

Figure 2.1: Subfields of $\mathbb{F}_{30}$.

2.1.3 Polynomials over finite fields

The ring formed by the polynomials over a field F is called the *polynomial ring* over F and denoted by $F[x]$. Let $f(x) = \sum_{i=0}^{n} f_i x^i \in F[x]$. Polynomial $f(x)$ is called a *monic polynomial* if $f_n = 1$.

Definition 2.4 [47] A polynomial $f(x) \in F[x]$ is said to be *irreducible* over F (or *irreducible in* $F[x]$, or *prime in* $F[x]$) if $f(x)$ has positive degree and $f(x) = g(x) \cdot h(x)$ with $g(x), h(x) \in F[x]$ implies that either $g(x)$ or $h(x)$ is a constant polynomial. □

$F[x]$ is a principal ideal domain. In fact, for every ideal $J \neq (0)$ of $F[x]$ there exists a uniquely determined monic polynomial $g \in F[x]$ with $J = (g)$. For $f \in F[x]$, the residue class ring $F[x]/(f)$ is a field if and only if f is irreducible over F. Moreover, if $F = \mathbb{F}_q$ and $f \in \mathbb{F}_q[x]$ is an irreducible polynomial of degree n, then $\mathbb{F}_q[x]/(f)$ is isomorphic to $\mathbb{F}_{q^n}$. And all the roots of f are in $\mathbb{F}_{q^n}$ and given by the n distinct elements $\alpha, \alpha^2, \alpha^{q^2}, \ldots, \alpha^{q^{n-1}}$ of $\mathbb{F}_{q^n}$.

For a nonzero polynomial $f(x) \in \mathbb{F}[x]$, the least positive integer e for which $f(x)$ divides $x^e - 1$ is called the *order* of f and denoted by $\mathrm{ord}(f)$. The number of monic irreducible polynomials in $\mathbb{F}_q[x]$ of degree m and of order e is equal to $\phi(e)/m$ if $e \geqslant 2$, equal to 2 if $m = e = 1$, and equal to 0 in all other cases. A monic irreducible polynomial $f(x) \in \mathbb{F}_q$ of degree n is *primitive* if $\mathrm{ord}(f) = q^n - 1$. A root of a primitive polynomial of degree n is a *primitive element* and generates the cyclic group $G = \mathbb{F}_{q^n}^*$.

Let K be a subfield of F. Then $\theta \in F$ is said to be *algebraic* over K if θ satisfies a nontrivial polynomial equation with coefficients in K. If $\theta \in F$ is algebraic over K, then the uniquely determined monic polynomial $g \in K[x]$ generating the ideal $J = \{f \in K[x] : f(\theta) = 0\}$ of K is called the *minimal polynomial* of θ over K. A field element is a primitive element if and only if its minimal polynomial is primitive.

Let $f \in F[x]$. The polynomial f is called a *trinomial* if it has three nonzero terms and it is called a *pentanomial* if it has five nonzero terms [47]. Because of their low Hamming weights, irreducible trinomials and pentanomials are often used in finite field arithmetic operations. Over $\mathbb{F}_2$, if $f(x)$ is of the form $f(x) = g(x^k)$ for $g(y) = 1 + y^{\lambda-1} + y^{\lambda}$, then all irreducible trinomials of the form $g(y)$ of degree up to $30,000$ are known [15]. By a theorem of Cohen [19], the condition under which $g(x^k)$ is irreducible given $g(x)$ is irreducible is also known. The same work in [15] has also given all the irreducible polynomials of the form $x^m + x^k + 1, 1 < k < \frac{m}{2}$ for all m up to $10,000$. Thus all irreducible trinomials are known for today's practical purposes. For $m \leqslant 10,000$, where irreducible trinomials do not exist, the recent work of [71] has shown that irreducible pentanomials exist.

The polynomial $f \in \mathbb{F}_2[x]$ is an *equally spaced polynomial* (or ESP) if for some integers t its degree $n = (t+1)r$ and, $f_i = 1$ if $i = jr$ and $j = 0, 1, \ldots, t$, and $f_i = 0$ otherwise. A 1-ESP is often called *all one polynomial* (AOP) [38]. Let r-ESP $f(x) = h(x^r)$, where $h(x) = \frac{x^{t+2}+1}{x+1}$. It can be seen that such an $f(x)$ is irreducible if and only if $t + 2 = p$, $r = p^n$, and $h(x)$ an irreducible AOP over $\mathbb{F}_2$, where p is a prime and n a non-negative integer [38]. This in turn exists if and only if 2 is a generator of $\mathbb{F}_p^*$ and for $n > 0$ p^2 does not divide $2^{p-1} - 1$ [35]. Table 2.1 shows all the irreducible ESPs for $m \leqslant 1000$ [38].

Examples of irreducible ESPs: m (r)								
1 (1)	28 (1)	100 (25)	172 (1)	342 (19)	460 (1)	558 (1)	700 (1)	828 (1)
2 (1)	36 (1)	106 (1)	178 (1)	346 (1)	466 (1)	562 (1)	708 (1)	852 (1)
4 (1)	52 (1)	110 (11)	180 (1)	348 (1)	486 (243)	586 (1)	756 (1)	858 (1)
6 (3)	54 (27)	130 (1)	196 (1)	372 (1)	490 (1)	612 (1)	772 (1)	876 (1)
10 (1)	58 (1)	138 (1)	210 (1)	378 (1)	500 (125)	618 (1)	786 (1)	882 (1)
12 (1)	60 (1)	148 (1)	226 (1)	388 (1)	508 (1)	652 (1)	796 (1)	906 (1)
18 (1)	66 (1)	156 (13)	268 (1)	418 (1)	522 (1)	658 (1)	812 (29)	940 (1)
18 (9)	82 (1)	162 (1)	292 (1)	420 (1)	540 (1)	660 (1)	820 (1)	946 (1)
20 (5)	100 (1)	162 (81)	316 (1)	442 (1)	546 (1)	676 (1)	826 (1)	

Table 2.1: r-ESP for $m \leqslant 1000$.

2.1.4 Roots of unity

Let K be a subfield of F and $\theta \in F$. Then the field $K(\theta)$ is defined as the intersection of all the subfields of F containing both K and θ, and is called the extension (field) of K obtained by *adjoining* the element θ. And θ is called a *defining element* of $L = K(\theta)$ over K.

Let $f \in K[x]$ be of positive degree and F an extension field of K. Then f is said to *split in* F if f can be written as a product of linear factors in $F[x]$—that is, if there exist elements $\alpha_1, \alpha_2, \ldots . \alpha_n \in F$ such that

$$f(x) = a(x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_n),$$

where a is the leading coefficient of f. The field F is a *splitting field* of f over K if f splits in F and if, moreover, $F = K(\alpha_1, \alpha_2, \ldots, \alpha_n)$. And it can be proven that such a splitting field of f over K always exists and is unique.

Let n be a positive integer. The splitting field of $x^n - 1$ over an arbitrary field K is called the nth *cyclotomic field* over K and denoted by $K^{(n)}$. The roots of $x^n - 1$ in $K^{(n)}$ are called the nth *roots of unity* over K and the set of all of these roots is denoted by $E^{(n)}$. Let the characteristic of K be p. Then there are two cases: (i) If p does not divide n, then $E^{(n)}$ is a cyclic group of order n with respect to multiplication in $K^{(n)}$. In this case, a generator of $E^{(n)}$ is called a *primitive* n^{th} *root of unity* over K. (ii) If p divides n, write

$n = mp^e$ with positive integers m and e, and m not divisible by p. Then $K^{(n)} = K^{(m)}$, $E^{(n)} = E^{(m)}$, and the roots of $x^n - 1$ in $K^{(n)}$ are the m elements of $E^{(m)}$, each with multiplicity p^e.

The nth *cyclotomic polynomial* over K is defined as

$$Q_n(x) = \prod_{\substack{s=1 \\ \gcd(s,n)=1}}^{n} (x - \xi^s),$$

where K has characteristic p, n a positive integer not divisible by p, and ξ a primitive nth root of unity over K. It is known that the degree of $Q_n(x)$ is $\phi(n)$ and its coefficients belong to the prime subfield of K [47]. If $K = \mathbb{F}_q$ with $\gcd(q, n) = 1$, then we have $x^n - 1 = \prod_{d|n} Q_n(x)$ and Q_n factors into $\phi(n)/d$ distinct monic irreducible polynomials in $K[x]$ of the same degree d, and furthermore, $K^{(n)}$ is the splitting field of any such irreducible factor over K, and $[K^{(n)} : K] = d$, where d is the least positive integer such that $q^d \equiv 1 \bmod n$.

2.1.5 Finite field bases and arithmetic operations

Let us consider the finite field $K = \mathbb{F}_q$ and its finite extension $F = \mathbb{F}_{q^n}$. Then F can be considered as an n-dimensional vector space over K, and if $\{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ is a basis of F over K, each element $A \in F$ can be uniquely represented in the form

$$A = a_0\alpha_0 + a_1\alpha_1 + \cdots + a_{n-1}\alpha_{n-1} \text{ with } a_j \in K, \ 0 \leqslant j \leqslant n-1.$$

Let $B = b_0\alpha_0 + b_1\alpha_1 + \cdots + b_{n-1}\alpha_{n-1}$ be another element in F. Then *addition* or *subtraction* of A with B is given by

$$A \pm B = (a_0 \pm b_0)\alpha_0 + (a_1 \pm b_1)\alpha_1 + \cdots + (a_{n-1} \pm b_{n-1})\alpha_{n-1},$$

where $a_j \pm b_j$ are taken as modulo p and p is the characteristic of F. The *multiplication* of A with B is given by

$$AB = \sum_{i=0}^{n-1} a_i\alpha_i \sum_{j=0}^{n-1} b_j\alpha_j = \sum_{0\leqslant i,j\leqslant n-1} a_i b_j \alpha_i \alpha_j.$$

Define multiplication matrices $T_k = (t_{i,j}^{(k)})_{n\times n}$, $t_{i,j}^{(k)} \in \mathbb{F}_q$, $k = 0, 1, \ldots, n-1$, such that

$$\alpha_k\alpha_i = \sum_{j=0}^{n-1} t_{i,j}^{(k)}\alpha_j \text{ for } k = 0, 1, \ldots, n-1. \tag{2.1}$$

Then coefficients of the product C can be written as a *bilinear form* of the coefficients of A and B:

$$c_k = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} t_{j,k}^{(i)} a_i b_j \text{ for } k = 0, 1, \ldots, n-1,$$

where $AB = C \triangleq \sum_{j=0}^{n-1} c_j\alpha_j$.

Below we give a few special types of bases of particular interest.

Definition 2.5 [47] A *polynomial basis* $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$, of F over K, is made up of the powers of an element α whose minimal polynomial f over K is of degree n. □

Since $\alpha_i\alpha_j = \alpha^{i+j}$, the multiplication matrix can be reduced into $T = (t_{i,j})_{(n-1)\times n}$ and, $\alpha_i\alpha_j = \alpha_{i+j}$ if $i + j \leqslant n-1$ and $\alpha_i\alpha_j = \sum_{k=0}^{n-1} t_{i+j-n,k}\alpha^k$ if $i + j \geqslant n$. Then from

$$AB = \sum_{\substack{0\leqslant i,j\leqslant n-1\\ i+j\geqslant n}} \sum_{k=0}^{n-1} a_i b_j t_{i+j-n,k}\alpha^k + \sum_{\substack{0\leqslant i,j\leqslant n-1\\ i+j\leqslant n-1}} a_i b_j \alpha^{i+j},$$

the product C can be obtained by

$$c_k = \sum_{\substack{0\leqslant i,j\leqslant n-1\\ i+j\geqslant n}} t_{i+j-n,k}a_ib_j + \sum_{i=0}^{k} a_ib_{k-i} \text{ for } k = 0,\dots,n-1.$$

Definition 2.6 [47] A set of n elements of F over K of the form $\{\alpha, \alpha^q, \dots, \alpha^{q^{m-1}}\}$, where α is a suitable element in F, is called a *normal basis* of F over K. □

Since $\alpha_j^q = \alpha_{(j+1)}$, where $(j+1) \triangleq j+1 \bmod n$, we raise by a power of q on both sides of (2.1) and it follows

$$\alpha_{(k+1)}\alpha_{(i+1)} = \sum_{j=0}^{n-1} t_{(i+1),j}^{((k+1))}\alpha_j = \sum_{j=0}^{n-1} t_{i,j}^{(k)}\alpha_{(j+1)}.$$

Then we have $t_{i,j}^{(k)} = t_{(i-1),j}^{(k-1)} = \cdots = t_{(i-k),j}^{(0)}$, and thus

$$c_k = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} t_{(j-i),k}^{(0)}a_ib_j \text{ for } k = 0,1,\dots,n-1. \tag{2.2}$$

Now it can be seen from (2.2) that each c_k, $k = 0,1,\dots,m-1$ has the same number of nonzero terms, and furthermore it is equal to the number of nonzeros in the matrix $\boldsymbol{T_0}$. Following Mullin, Onyszchuk, Vanstone and Wilson [62], we denote this number as C_N.

Theorem 2.1 [62] If N is a normal basis in $\mathbb{F}_{2^m}$, then $C_N \geqslant 2m-1$. N is called an *optimal normal basis* when $C_N = 2m-1$.

For $\alpha \in F = \mathbb{F}_{q^n}$ and $K = \mathbb{F}_q$, the *trace* $\mathrm{Tr}_{F/K}(\alpha)$ of α over K is given by $\mathrm{Tr}_{F/K}(\alpha) = \alpha + \alpha^q + \cdots + \alpha^{q^{m-1}}$. If K is the prime field of F, then $\mathrm{Tr}_{F/K}(\alpha)$ is usually denoted by $\mathrm{Tr}_F(\alpha)$ or simply, $\mathrm{Tr}(\alpha)$ if F is understood.

Definition 2.7 [47] Let K be a finite field and F a finite extension of K. Then two bases $\{\alpha_0, \ldots, \alpha_{m-1}\}$ and $\{\beta_0, \ldots, \beta_{m-1}\}$ of F over K are said to be *dual* to each other if for $1 \leqslant i, j \leqslant m$, we have

$$\mathrm{Tr}_{F/K}(\alpha_i\beta_j) = \begin{cases} 0 & \text{for } i \neq j, \\ 1 & \text{for } i = j. \end{cases}$$

□

A basis that is its own dual is called a *self-dual basis*.

2.1.6 Elliptic curves over finite fields

An elliptic curve over the finite field F_q is given as [54]

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \tag{2.4}$$

For an elliptic curve E/F_q defined over F_q, the set F_q-rational points of E/F_q, denoted by $E(F_q)$, is the set of points whose both coordinates lie in F_q, together with the point $\mathcal{O}$. By a theorem of Hasse, we know the number of points in $E(F_q)$ is: $\#E(F_q) = q + 1 - t, |t| \leq 2\sqrt{q}$. The elliptic curve E is said to be *supersingular* if p divides t, otherwise it is called *non-supersingular*.

A non-supersingular elliptic curve E over $\mathbb{F}_{2^n}$ is of the form [54]

$$E : y^2 + xy = x^3 + a_2x^2 + a_6,$$

where $a_2, a_6 \in \mathbb{F}_{2^n}$. The set of $\mathbb{F}_{2^n}$-rational points $E(\mathbb{F}_{2^n})$ on E together with $\mathcal{O}$ forms an abelian group under a certain operation which is usually called addition. Let $P = (x_1, y_1) \in E$, then $-P = (x_1, y_1 + x_1)$. If $Q = (x_2, y_2) \in E$ and $Q \neq -P$, then

$P + Q = (x_3, y_3)$, where

$$P \neq Q : \begin{cases} x_3 = (\frac{y_1 + y_2}{x_1 + x_2})^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a_2, \\ y_3 = (\frac{y_1 + y_2}{x_1 + x_2})(x_1 + x_3) + x_3 + y_1, \end{cases}$$

and

$$P = Q : \begin{cases} x_3 = x_1^2 + \frac{a_6}{x_1^2}, \\ y_3 = x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3. \end{cases}$$

Let $b \in \mathbb{F}_{2^n}$ be the square root of a_6, which always exists, then the point $(0, b)$ is on E and has order 2. Thus we know that the highest order p of a point on E can not be greater than half of $\#E(\mathbb{F}_{2^n})$. By the Hasse theorem, $\#E(\mathbb{F}_{2^n}) = 2^n + O(2^{\frac{n}{2}})$, then the possible highest order of a point is not greater than $2^{n-1} + O(2^{\frac{n}{2}-1})$.

Since the characteristic of the underlying field is 2, for non-supersingular curve the cost of an elliptic addition is roughly the same as that of a point doubling, which is about one field inversion and three or four field multiplications. Let $P \in E(\mathbb{F}_{2^n})$. If $P = (x, y)$, then it is worthy to note that $-P = (x, -y) = (x, x + y)$. Consequently, elliptic addition $P + Q$ has about the same difficulty as point subtraction $P - Q$, where $Q \in E(\mathbb{F}_{2^n})$.

In [43], Koblitz has shown that doubling a point on some non-supersingular curves over $\mathbb{F}_q$ with complex multiplication (CM) can be done almost as easily as in the case of supersingular curves. Such curves defined over $\mathbb{F}_2$ are given by [43]

$$E_c : \; y^2 + xy = x^3 + (1 - c)x + 1,$$

where $c = 0$ or 1. The Frobenius map τ of E_c over $\mathbb{F}_2$ is defined by $\tau(x, y) = (x^2, y^2)$ that satisfies: $2 = -\tau^2 + (-1)^c\tau$. Then given a multiple point mP one can write m as a

sum of τ powers. Since squaring in $\mathbb{F}_{2^n}$ is simply a circular shift of the coordinates in a normal basis, mP can be computed with a few elliptic additions.

Chapter 3

Normal Basis Multipliers

In this chapter we propose a new basis – redundant basis, which is a modification of certain normal bases. The redundant basis takes advantage of the elegant multiplicative structure of the set of $(mk+1)^{st}$ roots of unity over $\mathbb{F}_q$ that includes a basis of $\mathbb{F}_{q^m}$. It is shown that multiplication using redundant basis is simple.

The generation of a normal basis using the Gauss period is first reviewed (§3.1). Then bit-serial multipliers using redundant basis are proposed (§3.2). Parallelization of these multipliers is also discussed (§3.3). Discussions on redundant bases are given in §3.4.

3.1 Gauss Period and Normal Basis

The Gauss period was discovered by Gauss and is defined as follows: Let $m, k \geqslant 1$ be integers such that $r = mk + 1$ is a prime, and let q be a prime power with $\gcd(q, r) = 1$. Let $\mathcal{K}$ be the unique subgroup of order k of the multiplicative group of $\mathbb{Z}_r = \mathbb{Z}/r\mathbb{Z}$, then

for any primitive rth root β of unity in $\mathbb{F}_{q^{mk}}$, the element

$$\alpha = \sum_{\gamma \in \mathcal{K}} \beta^{\gamma} \tag{3.1}$$

is called a *Gauss period of type* (m, k) over $\mathbb{F}_q$. It can be checked that $\alpha \in \mathbb{F}_{q^m}$.

The Gauss periods have been used to construct normal bases with low complexity [62, 7]. A Gauss period of type (m, k) over $\mathbb{F}_q$ is a normal element of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ if and only if $\gcd(e, m) = 1$, where e is the index of q modulo r. Furthermore, such a normal basis has complexity at most $mk' - 1$ with $k' = k$ if $p|k$ and $k + 1$ otherwise, where p is the characteristic of $\mathbb{F}_q$ [7, 78, 26]. Clearly, for $q = 2$, Gauss periods of types $(m, 1)$ and $(m, 2)$ generate optimal normal bases with complexity $2m - 1$, which are usually called type-I and type-II optimal normal bases (ONB), respectively [62]. For small values of $k > 2$, the Gauss periods generate low complexity normal bases [7]. Other classes of low complexity normal bases can be generated by an extension of the Gauss period where $r = mk + 1$ is not a prime [21].

Gauss periods are also used to develop fast arithmetic in finite fields. Gao and Vanstone have proposed an exponentiation algorithm in $\mathbb{F}_{2^m}$ using normal bases generated with the Gauss period of type $(m, 2)$ [26]. Subsequent work has shown that an efficient realization of arithmetic operations can be obtained using normal bases generated with the Gauss period of type (m, k), $k > 2$ [27], and with the *general Gauss period* [25].

3.2 Bit-Serial NB Multipliers

3.2.1 Previous implementations

The first efficient implementation of normal basis multiplication was described by Massey and Omura in a US patent [49]. Let $A = \sum_{k=0}^{m-1} a_k\alpha^{2^k}$ and $B = \sum_{k=0}^{m-1} b_k\alpha^{2^k}$ be two elements in $\mathbb{F}_{2^m}$ represented with respect to the normal basis $\langle \alpha, \alpha^2, \ldots, \alpha^{2^{m-1}} \rangle$. Viewing the coefficient c_k of $C = A \cdot B$ as a bilinear form of $A = (a_0, a_1, \ldots, a_{m-1})$ and $B = (b_0, b_1, \ldots, b_{m-1})$, the f-function is defined as

$$c_{m-1} = f(a_0, a_1, \ldots, a_{m-1}; b_0, b_1, \ldots, b_{m-1}).$$

Since squaring of an element is just a cyclic shift of its coefficients, we have

$$(c_{m-1}, c_0, \ldots, c_{m-2}) = (a_{m-1}, a_0, \ldots, a_{m-2}) \times (b_{m-1}, b_0, \ldots, b_{m-2}).$$

Then it follows that

$$\begin{aligned} c_{m-2} &= f(a_{m-1}, a_0, \ldots, a_{m-2}; b_{m-1}, b_0, \ldots, b_{m-2}) \\ &\vdots \\ c_0 &= f(a_1, \ldots, a_{m-1}, a_0; b_1, \ldots\ldots, b_{m-1}, b_0) \end{aligned}$$

The f-function can be solved from (2.2) as

$$c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} t^{(0)}_{(j-i),k} a_{i+k} b_{j+k} = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} t^{(0)}_{(j-i),k} a_i b_j,$$

where $t^{(0)}_{ij}$ can be obtained from $\beta_0\beta_i = \sum_{j=0}^{m-1} t^{(0)}_{ij}\beta_j$. Clearly, realizing the f-function requires C_N multiplication operations and $C_N - 1$ addition operations, both in $\mathbb{F}_2$. Also,

two m-bit registers are required to supply the shifted coefficients of A and B.

An alternate structure with serial input and parallel output and its VLSI implementation has been proposed by Agnew, Mullin, Onyszchuk, and Vanstone [3]. Their basic idea is as follows [3]:

Let the function $F_j^{(k)}(\cdot)$ be defined by

$$F_j^{(k)}(t) = b_{j+k+t} \sum_{i=0}^{m-1} t_{j-i,0}^{(0)} a_{i+k+t},$$

where $t_{i,j}^{(k)}$ is given in (2.2). Then $c_k = \sum_{j=0}^{m-1} F_j^{(k)}(0)$. The coefficients of A and B are stored in registers $\mathbf{A}$ and $\mathbf{B}$, which are shifted cyclically. At time t, $\sum_{i=0}^{t} F_j^{(0)}(i), \ldots, \sum_{i=0}^{t} F_j^{(m-1)}(i)$ are computed from the previous contents in register $\mathbf{C}$ and current contents of $\mathbf{A}$, and the results are stored in $\mathbf{C}$. Then after m clock cycles, the contents of register $\mathbf{C}$ are $c_0, c_1, \ldots, c_{m-1}$.

Compared to the Massey-Omura multiplier, this multiplier offers advantages in the following aspects: fewer gates, potentially faster clock rate, highly regular structure and simpler cell connection. It is thus more suitable for large fields.

Other normal basis multipliers include the one presented by Feng in [22]. It has the same input and output style as Agnew *et al*'s [3], but its complexity seems not better than the one in [3] when implemented in VLSI technology.

Recently, Gao and Vanstone have proposed a novel multiplication algorithm for the field generated with the Gauss period of type $(m, 2)$ [26]. The resultant architecture is very simple and has low complexities.

3.2.2 Bases generated with Gauss period of type $(m, 1)$

Algorithm Let β be a primitive $(m+1)$st root of unity in $\mathbb{F}_{q^m}$ and q is a generator of the group $G = \mathbb{F}^*_{m+1}$, then $\langle \beta, \beta^q, \ldots, \beta^{q^{m-1}} \rangle \triangleq I_1$ is an optimal normal basis in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ generated with the Gauss period of type $(m, 1)$. From

$$\beta \cdot \beta^i = \begin{cases} \beta^{i+1} & i \neq m, \\ 1 = \sum_{j=1}^{m} \beta^j & i = m, \end{cases}$$

it can be shown that $I_2 \triangleq \langle \beta, \beta^2, \ldots, \beta^m \rangle$ is a set of m linearly independent elements in $\mathbb{F}_{q^m}$ and thus forms a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. It can also be verified that I_2 contains the same elements as I_1, since β is a primitive $(m+1)$st root of unity in $\mathbb{F}_{q^m}$.

Consider the set of the following $m+1$ ordered field elements of $\mathbb{F}_{q^m}$: $\langle 1, \beta, \beta^2, \ldots, \beta^m \rangle$, and denote it as I_3. Clearly, every element $A \in \mathbb{F}_{q^m}$ can be represented with I_3:

$$A = a_0 + a_1\beta + a_2\beta^2 \cdots + a_m\beta^m. \tag{3.2}$$

where $a_i \in \mathbb{F}_q, i = 0, 1, \ldots, m$. Then I_3 can serve as a representation basis for $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Since the elements of I_3 are not linearly independent, the representation of a field element with respect to I_3 is not unique. In the sequel, we will refer to I_3 as a *redundant basis.*

Now let us look at multiplication operation under the redundant basis I_3. Let $B \in \mathbb{F}_{q^m}$ be given as $B = b_0 + b_1\beta + b_2\beta^2 + \cdots + b_m\beta^m$. Then we have

$$\begin{aligned} \beta \cdot B &= b_0\beta + b_1\beta^2 + b_2\beta^3 + \cdots + b_{m-1}\beta^m + b_m\beta^{m+1} \\ &= b_m + b_0\beta + b_1\beta^2 + b_2\beta^3 + \cdots + b_{m-1}\beta^m. \end{aligned}$$

Obviously, the coordinates of βB is a cyclic shift of those of B, with respect to I_3. From

$$\begin{aligned}\beta^i \cdot B &= b_0\beta^i + b_1\beta^{i+1} + b_2\beta^{i+2} + \cdots + b_{m-i}\beta^m + b_{m-i+1} + b_{m-i+2}\beta + \cdots + b_m\beta^{i-1} \\ &= b_{m-i+1} + b_{m-i+2}\beta + \cdots + b_m\beta^{i-1} + b_0\beta^i + b_1\beta^{i+1} + \cdots + b_{m-i}\beta^m \\ &= \sum_{j=0}^{m} b_{(j-i)}\beta^j, \end{aligned} \tag{3.3}$$

where $(j-i) = (j-i) \bmod (m+1)$ denotes that $j-i$ is to be reduced modulo $m+1$, we have

$$A \cdot B = \sum_{i=0}^{m} a_i(\beta^i \cdot B) = \sum_{i=0}^{m} a_i \sum_{j=0}^{m} b_{(j-i)}\beta^j = \sum_{j=0}^{m} \Big(\sum_{i=0}^{m} a_i b_{(j-i)} \Big) \beta^j.$$

If we define $AB = C = \sum_{j=0}^{m} c_j\beta^j$, then it follows

$$c_j = \sum_{i=0}^{m} a_i b_{(j-i)}, \quad j = 0, 1, \ldots, m. \tag{3.4}$$

Architecture Figure 3.1 shows the multiplier structure to realize multiplication using I_3. The coordinates of B with respect to I_3 are loaded into a register of length $m+1$ bits whose contents can be shifted cyclically. The binary tree of m adders in $\mathbb{F}_q$ takes $m+1$ $a_i b_k$ terms as its inputs and generates a c_j term as output every clock cycle. All c_j's, $j = 0, 1, \ldots, m$, which are represented using I_3, are computed and obtained in $m+1$ clock cycles. When $p = 2$, it can be seen that $m+1$ AND gates, m XOR gates and $m+1$ 1-bit registers are required for constructing the multiplier. The clock period should not be less than $T_A + \lceil \log_2 m + 1 \rceil T_X$. Table 3.1 shows a comparison of the multipliers proposed in [49, 22, 3] and the presented here. It can be seen from the table that the new proposed architecture has a significantly lower complexity compared to the previous

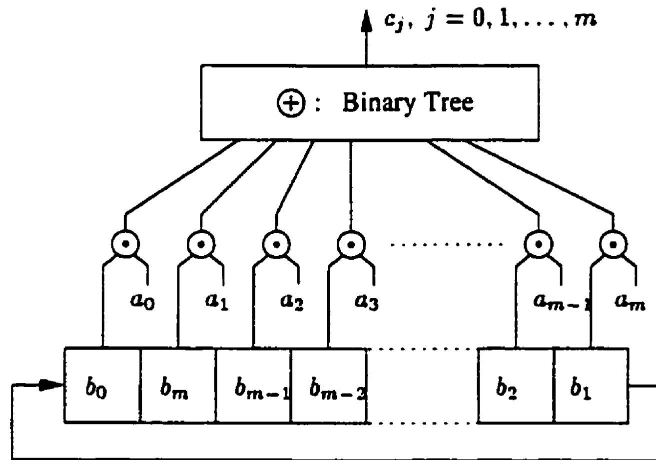

Figure 3.1: Bit serial multiplier using redundant basis when there is a type I ONB.

Multiplier	#AND	#XOR	#1-bit reg.	# clk cycles	Basis
Massey-Omura [49]	$2m-1$	$2m-2$	$2m$	m	normal
Feng [22]	$2m-1$	$3m-2$	$3m-2$	m	normal
Agnew *et al* [3]	m	$2m-1$	$3m$	m	normal
presented here	$m+1$	m	$m+1$	$m+1$	redundant

Table 3.1: Comparison of bit-serial multipliers using type I ONB and RB.

implementations.

Basis conversion The conversions between the normal basis I_2 and the redundant basis I_3 are simple: If $A = (a'_1, a'_2, \ldots, a'_m)$ with respect to basis I_2, then $(0, a'_1, a'_2, \ldots, a'_m)$ is a representation of A with respect to basis I_3; If $A = (a_0, a_1, a_2, \ldots, a_m)$ with respect to I_3, then with basis I_2 the representation of A is $(a_1 + a_0, a_2 + a_0, \ldots, a_m + a_0)$.

3.2.3 Bases generated with Gauss period of type (m, k)

Algorithm The idea of redundant basis can be easily applied to the normal basis generated with the Gauss period of type $(m, k), k > 1$. Let β be a primitive $(mk+1)$st root of unity in $\mathbb{F}_{2^{mk}}$ and γ be a primitive kth root of unity in $G = \mathbb{F}^*_{km+1}$. If $G = \langle 2, G^m \rangle$, then $\alpha = \sum_{i=0}^{k-1} \beta^{\gamma^i}$ is a normal element and $I_4 \triangleq \langle \alpha, \alpha^2, \ldots, \alpha^{2^{m-1}} \rangle$ is a normal basis in $\mathbb{F}_{2^m}$ with complexity not greater than $k'm - 1$.

Consider two sets of km elements in $\mathbb{F}_{2^{km}}$: $S_1 = \{\beta^{2^i\gamma^j}, i = 0, 1, \ldots, m-1; j = 0, 1, \ldots, k-1\}$ and $S_2 = \{\beta, \beta^2, \ldots, \beta^{km}\}$. For any element $\beta^{2^i\gamma^j} \in S_1$, we have $\beta^{2^i\gamma^j} = \beta^{2^i\gamma^j \bmod (mk+1)} \in S_2$, and thus, $S_1 \subseteq S_2$. Let $G = \mathbb{F}^*_{km+1}$ then $G = \langle 2, \gamma \rangle$. For any integer $l \in \{1, 2, \ldots, km\}$, there exist integers $i \in \{0, 1, \ldots, m-1\}$ and $j \in \{0, 1, \ldots, k-1\}$, such that $l = 2^i\gamma^j \bmod (km+1)$. Therefore, $S_2 \subseteq S_1 \Rightarrow S_2 = S_1$.

Since $I_4 = \langle \sum_{i=0}^{k-1} \beta^{\gamma^i}, \sum_{i=0}^{k-1} \beta^{2\gamma^i}, \ldots, \sum_{i=0}^{k-1} \beta^{2^{m-1}\gamma^i} \rangle$ and each element in I_4 is a sum of k elements in S_1, it can be seen that elements in $S_1 (= S_2)$ can serve as a basis in $\mathbb{F}_{2^m}$. Consider redundant basis I_5 by adding element '1' to the set S_2: $I_5 = \langle 1, \beta, \beta^2, \ldots, \beta^{km} \rangle$. Obviously, any element $B \in \mathbb{F}_{2^m}$ can be represented with I_5: $B = b_0 + b_1\beta + \cdots + b_{km}\beta^{km}$, where $b_0, \ldots, b_{km} \in \mathbb{F}_2$. Then the multiplication of B with β^l is actually an l-fold cyclic shift of the coordinates of B:

$$\beta^l B = b_0\beta^j + b_1\beta^{l+1} + \cdots + b_{mk}\beta^{mk+l} = \sum_{i=0}^{mk} b_{(i-l)}\beta^i.$$

Architecture A structure for multiplication in $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ using I_5 is shown in Figure 3.2. The structure is very similar to that shown in Figure 3.1 except that it requires more gates and registers. Its complexities are compared to those of other similar multipliers in Table 3.3 (When the Gauss period is of type $(m, 2)$, comparison is made and shown in Table 3.2).

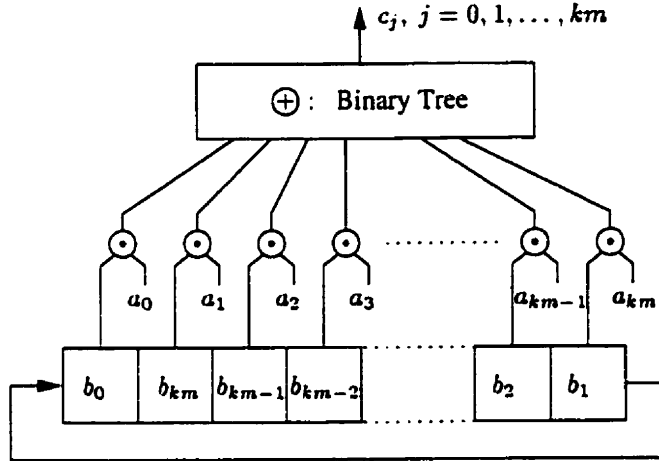

Figure 3.2: Bit serial multiplier using redundant basis.

Multipliers	#AND	#XOR	#1-bit reg.	# clk cycles	basis
Massey-Omura [49]	$2m-1$	$2m-2$	$2m$	m	normal
Feng [22]	$2m-1$	$3m-2$	$3m-2$	m	normal
Agnew *et al* [3]	m	$2m-1$	$3m$	m	normal
Gao-Vanstone [26]	m	$2m-1$	$2m+1$	m	normal [1]
presented here	$2m+1$	$2m$	$2m+1$	$2m+1$	redundant

[a] In fact, it is a certain permutation of a normal basis.

Table 3.2: Comparison of bit-serial multipliers using type II ONB and RB.

From the table it can be seen that the new structure suffers a lower throughput and when k is an odd integer not less than 3 it also has higher space complexity. However, it still has the advantage of simpler architecture over the other implementations.

Basis conversion Now let us look at the conversion from the normal basis $I_4 = \langle \alpha, \alpha^2, \ldots, \alpha^{2^{m-1}} \rangle$ to the basis I_5. As we have seen before, the conversion between redundant basis I_5 and the basis consisting of elements from $S_2 = S_1$ is simple. If $A = (a'_0, a'_1, \ldots, a'_{m-1})$ with

Multipliers	#AND	#XOR	#1-bit reg.	# clk cycles	basis
Massey-Omura [49]	C_N	$C_N - 1$	$2m$	m	normal
Feng [22]	$2m - 2$	$C_N + m - 1$ [b]	$3m - 2$	m	normal
Agnew *et al* [3]	m	C_N	$3m$	m	normal
presented here	$km + 1$	km	$km + 1$	$km + 1$	redundant

[b] In the example presented in [22], a technique of reusing partial sum was used to reduce the complexity. Thus the number of XOR gates should be not greater than $C_N + m - 1$ if a non-optimal normal basis is used.

Table 3.3: Comparison of bit-serial multipliers using NB and RB.

the normal basis, then with the basis from S_1,

$$A = (a''_{0,0}, a''_{0,1}, \dots, a''_{0,k-1}, \dots, a''_{m-1,k-1}),$$

where $a''_{i,j} = a'_i$ for $j = 0, 1, \dots, k - 1$ and $i = 0, 1, \dots, m - 1$.

3.3 Bit Parallel NB Multipliers

3.3.1 Previous implementations

Although the original Massey-Omura multiplier focuses on bit-serial form, its parallelization is straightforward. The architecture of a bit-parallel version of Massey-Omura multiplier can for instance be found in Wang, *et. al.*'s paper [76], The complexity is mC_N AND gates and $m(C_N - 1)$ XOR gates. If an optimal normal basis is chosen, the complexity is $m(2m - 1) + m(2m - 2) = 4m^2 - 3m$.

Later, Hasan, Wang and Bhargava proposed a modified Massey-Omura bit-parallel multiplier [36] using the type-I optimal normal basis which has accomplished the lowest complexity among bit-parallel normal basis multipliers reported so far in the literature. Let the normal basis $(\alpha, \alpha^2, \dots, \alpha^{2^{m-1}})$ be generated with the Gauss period of type

$(m, 1)$. Since α is a primitive $(m+1)$st root of unity in $\mathbb{F}_{2^m}$, then, $\alpha_i\alpha_j = 1$ for some $j = s(i)$, and $\alpha_i\alpha_j = \alpha_{t(i,j)}$, for $j \neq s(i)$, $i = 0, 1, 2, \ldots, m-1$, and

$$\begin{aligned} AB = C &= \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j \alpha_i \alpha_j \\ &= \sum_{i=0}^{m-1} a_i \Big(\sum_{\substack{0 \le j \le m-1 \\ j \neq s(i)}} b_j \alpha_{t(i,j)} + b_{s(i)} \cdot 1 \Big) \\ &= \sum_{i=0}^{m-1} a_i \Big(\sum_{\substack{0 \le j \le m-1 \\ j \neq s(i)}} b_j \alpha_{t(i,j)} + b_{s(i)} \sum_{k=0}^{m-1} \alpha_k \Big) \\ &= \sum_{i=0}^{m-1} \sum_{\substack{0 \le j \le m-1 \\ j \neq s(i)}} b_j \alpha_{t(i,j)} + \sum_{k=0}^{m-1} \Big(\sum_{i=0}^{m-1} a_i b_{s(i)} \Big) \alpha_k \\ &= C^1 + C^2. \end{aligned}$$

Since for fixed i_0, $t(i_0, j)$ runs through $0, 1, 2, \ldots, m-1$ except i_0, C^1 has $m(m-1)$ terms and α_l would appear in $m-1$ terms for $l = 0, 1, \ldots, m-1$. Therefore, in $c_k = c_k^1 + c_k^2$, c_k^1 has $m-1$ terms and requires $m-1$ AND gates and $m-2$ XOR gates. While $c_k^2 = \sum_{i=0}^{m-1} a_i b_{s(i)}$ has m terms but they are the same ones for $k = 0, 1, \ldots, m-1$, hence c_k^2 needs to be implemented only once which costs m AND gates and $m-1$ XOR gates. Considering another m XOR gates to realize $c_k = c_k^1 + c_k^2$ for $k = 0, 1, \ldots, m-1$, the complexity is $m(m-1)+m = m^2$ AND gates and $m(m-2)+(m-1)+m = m^2-1$ XOR gates.

Other parallel multipliers using type-I ONB include the one presented in [45]. It is a combination of a polynomial basis multiplier and bases conversion circuits.

Multipliers	#AND	#XOR	Time delay
Hasan et al [36]	m^2	$m^2 - 1$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$
Koc and Sunar [45]	m^2	$m^2 - 1$	$T_A + (2 + \lceil \log_2 m \rceil)T_X$
New proposal	$(m+1)^2$	$m(m+1)$	$T_A + \lceil \log_2(m+1) \rceil T_X$

Table 3.4: Comparison of bit-parallel multipliers using type I ONB and RB.

3.3.2 New bit-parallel multipliers

When a normal basis in $\mathbb{F}_{2^m}$ is generated with a Gauss period of type $(m, 1)$, a parallel version of the multiplier using a redundant basis is shown in Figure 3.3. On the left side of the figure inputs $\{a_i\}$ and $\{b_i\}$ are fed into m blocks (Block B). The detailed structure of Block B is shown on the right side of the figure. It can be seen that $(m+1)^2$ AND gates and $m(m+1)$ XOR gates are required. The time delay is $T_A + \lceil \log_2(m+1) \rceil T_X$. Compared to the bit-parallel multiplier proposed in [36], this one uses more gates, but has a simpler architecture (see Table 3.4 and Fig. 3.3). Moreover, it can be easily made for trade-offs between size and time complexities: If t Block B's are used to construct a multiplier and thus in one clock cycle t c_j's are computed and output, then one multiplication operation can be completed in $\lceil \frac{m}{t} \rceil$ clock cycles.

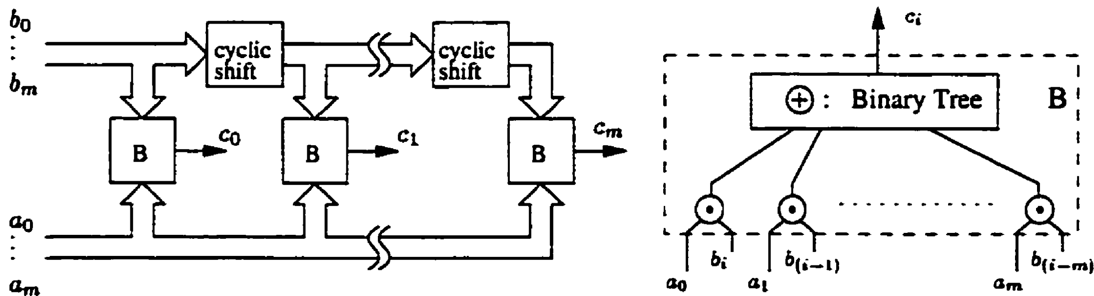

Figure 3.3: Parallelization of the bit-serial multiplier using the redundant basis.

When the redundant basis is generated with Gauss period of type (m, k), the parallelization of bit-serial multiplier can be obtained in a similar way.

3.4 Discussions on Redundant Basis

From the previous sections, it can be seen that a redundant basis is the set of all the nth roots of unity over $\mathbb{F}_q$ which includes m field elements in $\mathbb{F}_{q^m}$ which form a basis in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Clearly, any field $\mathbb{F}_{q^m}$ has a redundant basis if there is a cyclotomic field over $\mathbb{F}_q$ that contains $\mathbb{F}_{q^m}$ as a subfield. Thus the redundant basis can be the set of (q^m-1)st roots of unity if p does not divide $q^m - 1$,where p is the characteristic of $\mathbb{F}_{q^m}$. To efficiently represent the field elements, the redundant basis should be chosen such that its size is as small as possible. Now the question is: Given $\mathbb{F}_{q^m}$, what is the smallest cyclotomic field $\mathbb{F}_q^{(n)}$ that contains $\mathbb{F}_{q^m}$ as a subfield? A systematic algorithm for computing such an n is given below.

Algorithm 3.1 Computing the smallest cyclotomic field that includes $\mathbb{F}_{q^m}$ as a subfield

1. Find all the factors d_i of q^m-1 that are greater than m and list them in an increasing order: $d_1, d_2, \dots, d_k = q^m - 1$;
2. Do while$(i \leqslant k)$

   If $m \mid \phi(d_i)$, then output d_i as n, and Stop; Else $i \leftarrow i + 1$.

□

Since the $(q^m - 1)^{\text{th}}$ cyclotomic field has a degree of $\phi(q^m - 1)$ and contains the field $\mathbb{F}_{q^m}$ as a subfield, we have that m divides $\phi(q^m - 1)$.

A redundant basis consisting of all the nth roots of unity is an *optimal* redundant basis if $\mathbb{F}_q^{(n)}$ is the smallest cyclotomic field over $\mathbb{F}_2$ that contains $\mathbb{F}_{q^m}$ as a subfield.

Given a basis I in $\mathbb{F}_{q^m}$, the general case of basis conversion between I and the redundant basis R may not be trivial. If I is a normal basis generated with the Gauss period of type (m, k), then how to obtain R has been discussed in §3.2 and §3.3. If

$I = \langle 1, \alpha, \ldots, \alpha^{m-1} \rangle$ is the polynomial basis, and if we know that the order of element α is $\text{ord}(\alpha)$, then the redundant basis R can be obtained using the following algorithm:

Algorithm 3.2 Computing the RB from the PB $\langle 1, \alpha, \ldots, \alpha^{m-1} \rangle$

1. Compute n using Algorithm 3.1;
2. Compute the order of the irreducible polynomial $\text{ord}(\alpha)$;
3. Let $t = \text{ord}(\alpha)/n$, then the RB is given by $\langle 1, \alpha^t, \alpha^{2t}, \ldots, \alpha^{(n-1)t} \rangle$. □

Chapter 4

Parallel Dual Basis Multipliers

In this chapter, first a complexity bound for parallel *weakly dual basis* (WDB) multipliers in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ is given (§4.2). Then parallel multipliers using WDB in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ (§4.2) and $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ (§4.3 and §4.4) are presented, respectively. When the generating polynomial is $x^m + x^k + 1, 1 \leqslant k \leqslant \left\lfloor \frac{m}{2} \right\rfloor$, or an r-ESP ($r \geqslant 1$) over $\mathbb{F}_2$, low complexity bit-parallel multipliers are constructed with reduced propagation delays. Basis conversions between the WDB and the polynomial basis and vice versa are discussed in §4.5. Parts of this chapter were presented in [82] and [81].

4.1 A Brief Review of Dual Basis Multipliers

Implementation of dual basis (DB) multiplication is known as bit-serial Berlekamp multiplier which efficiently realizes the operation using a linear feedback shift register (LFSR) [12]. Recently many other DB type bases (which, in the sequel, will be referred to as weakly dual basis or WDB) have been found and used to achieve a LFSR style operation [60, 77]. A self-dual normal basis has also been considered and used for the implementation of multiplication in [75]. Bit-parallel DB multipliers have been discussed in [63, 23]. By

reusing some other previously generated signals, it has been shown in [23] that significant reduction of size complexity can be achieved for the fields of characteristic two.

4.2 Parallel Multipliers in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$

4.2.1 WDB multiplication

Definition 4.1 Two bases $\langle\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\rangle$ and $\langle\beta_0, \beta_1, \ldots, \beta_{m-1}\rangle$ of F_{q^m} over F_q are said to be *weakly dual* to each other if $\mathrm{Tr}(\gamma\alpha_i\beta_j) = \delta_{ij},\ i, j = 0, 1, 2, \ldots, m-1$, where $\gamma \in F^*_{q^m} = F_{q^m}\backslash 0$ and δ_{ij} is the Kronecker delta function, which is equal to 1 if $i = j$ and 0 otherwise.

In this chapter, we consider $\langle\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\rangle$ to be the polynomial basis $\langle 1, \alpha, \ldots, \alpha^{m-1}\rangle$ where α is a root of monic irreducible polynomial $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ with $f_i \in F_q$. For a field element $A \in F_{q^m}$, $A = \sum_{i=0}^{m-1} a_i\alpha^i = \sum_{i=0}^{m-1} a^*_i\beta_i$, where a_i's and a^*_i's are the coordinates of A with respect to the polynomial basis and its weakly dual basis (WDB), respectively. Then we have

$$\mathrm{Tr}(\alpha^j A) = \mathrm{Tr}(\alpha^j \sum_{i=0}^{m-1} a^*_i\beta_i) = \sum_{i=0}^{m-1} \mathrm{Tr}(a^*_i\alpha^j\beta_i) = a^*_j, \qquad 0 \leqslant j \leqslant m-1. \tag{4.1}$$

Let the field element $B \in F_{q^m}$ be given by $B = \sum_{i=0}^{m-1} b^*_i\beta_i$, where b^*_i's are the coordinates of B with respect to the WDB. Consider the product of A and B, in the WDB, given by

$$AB = C \triangleq \sum_{j=0}^{m-1} c^*_j\beta_j.$$

From (4.1), c_j^* can be expressed as a sum of a_i's as follows:

$$c_j^* = \mathrm{Tr}(\alpha^j AB) = \mathrm{Tr}\left(\alpha^j B \sum_{i=0}^{m-1} a_i \alpha^i\right) = \sum_{i=0}^{m-1} \mathrm{Tr}(\alpha^{i+j} B) a_i = \sum_{i=0}^{m-1} c_{j,i}^* a_i, \qquad (4.2)$$

where

$$c_{j,i}^* \triangleq \mathrm{Tr}(\alpha^{i+j} B) = \begin{cases} b_{i+j}^* & \text{for } 0 \leqslant i+j \leqslant m-1, & (4.3\text{a}) \\ \mathrm{Tr}(\alpha^{i+j} B) & \text{for } m \leqslant i+j \leqslant 2m-2. & (4.3\text{b}) \end{cases}$$

Define the reduction matrix $\boldsymbol{R} = (\ r_{i,j}\ )_{(m-1)\times m}$, $r_{i,j} \in F_q$, by

$$\begin{bmatrix} \alpha^m \\ \alpha^{m+1} \\ \vdots \\ \alpha^{2m-2} \end{bmatrix} = \begin{bmatrix} r_{0,0} & r_{0,1} & \cdots & r_{0,m-1} \\ r_{1,0} & r_{1,1} & \cdots & r_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m-2,0} & r_{m-2,1} & \cdots & r_{m-2,m-1} \end{bmatrix} \begin{bmatrix} 1 \\ \alpha \\ \vdots \\ \alpha^{m-1} \end{bmatrix}, \qquad (4.4)$$

then,

$$\mathrm{Tr}(\alpha^{m+l} B) = \mathrm{Tr}\left[\left(\sum_{i=0}^{m-1} r_{l,i}\alpha^i\right) B\right] = \sum_{i=0}^{m-1} r_{l,i}\mathrm{Tr}(\alpha^i B) = \sum_{i=0}^{m-1} r_{l,i} b_i^*. \qquad l = 0, 1, 2, \ldots, m-2. \qquad (4.5)$$

The rows of the reduction matrix give the representation of $\alpha^{m+i}, i = 0, 1, \ldots, m-2$, in terms of the polynomial basis. The first part of the above identity is obtained from (4.4), whereas the last part follows from (4.1). It can be seen that (4.2), (4.3) and (4.5) will decide the multiplication in F_{q^m} over F_q. The above can be summarized in the following algorithm.

Algorithm 4.1 Parallel multiplication in F_{q^m} over F_q

Input: Field elements $A = \langle a_0, a_1, \ldots, a_{m-1} \rangle$, $B = \langle b_0^*, b_1^*, \ldots, b_{m-1}^* \rangle$;

Output: The product $C = \langle c_0^*, c_1^*, \ldots, c_{m-1}^* \rangle$.

(Precompute $r_{i,j}$, $i, j = 0, 1, \ldots, m-1$.)

1. Compute $\mathrm{Tr}(\alpha^{m+l}) = \sum_{i=0}^{m-1} r_{l,i} b_i^*$, for $l = 0, 1, \ldots, m-2$;
2. Compute $c_{j,i}^* a_i$ for $i, j = 0, 1, \ldots, m-1$;
3. Compute $c_j^* = \sum_{i=0}^{m-1} c_{j,i}^* a_i$ for $j = 0, 1, \ldots, m-1$. □

Note that the above multiplication algorithm has been suggested in [12, 63, 23] for $q = 2$. Write (4.5) as

$$\mathrm{Tr}(\alpha^{m+l} B) = \mathrm{Tr}\left[\alpha^l \left(\sum_{i=0}^{m-1} f_i \alpha^i\right) B\right] = \sum_{i=0}^{m-1} f_i \mathrm{Tr}(\alpha^{i+l} B), \qquad l = 0, 1, 2, \ldots, m-2.$$

In Step 1 of Algorithm 4.1, if we first compute $\mathrm{Tr}(\alpha^{l'} B)$ with smaller l', then it can be seen that each $\mathrm{Tr}(\alpha^{m+l} B)$ needs $H(f) - 2$ bit additions when the field is of characteristic 2 [23], where $H(f)$ is the Hamming weight of $f(x)$. The consequent architecture requires only m^2 two-input AND gates and $(m-1)(m + H(f) - 2)$ two-input XOR gates [23].

4.2.2 A complexity bound

For parallel multiplication in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, it is natural to represent its complexity based on the numbers of multiplication and addition operations in the ground field $\mathbb{F}_q$. Consequently, the complexity of a parallel multiplier in $\mathbb{F}_{q^m}$ can be measured in terms of the numbers of multipliers and adders in the ground field $\mathbb{F}_q$.

Theorem 4.2 Let $f(x)$ be an irreducible polynomial of degree m over $\mathbb{F}_q$. If $f(x)$ has k nonzero terms, then a parallel WDB multiplier in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ can be constructed with at most m^2 multipliers and $(k-1)(m-1)$ constant multipliers in $\mathbb{F}_q$, $m^2+(k-3)m-(k-2)$ adders (or subtractors) and $m-1$ constant adders (or subtractors) in $\mathbb{F}_q$.

Proof: From the discussion of the previous section, a parallel WDB multiplier can be implemented based on Algorithm 4.1. Steps 1, 2 and 3 of Algorithm 4.1 will determine the complexity of this multiplier. It can be readily seen that realizations of Steps 2 and 3 require m^2 multiplication operations in $\mathbb{F}_q$ and $m(m-1)$ addition operations in $\mathbb{F}_q$, respectively. In the following we will prove that the complexity required in Step 3 is $(k-1)(m-1)$ constant multiplications in $\mathbb{F}_q$, $(k-2)(m-1)$ multiplications and $m-1$ constant additions in $\mathbb{F}_q$.

Let the monic irreducible polynomial

$$f(x) = x^m + \sum_{i=0}^{k-2} f_{e_i} x^{e_i}$$

have k nonzero terms, where $0 = e_0 < e_1 < \cdots\cdots < e_{k-2} < m$, $f_{e_i} \in F_q^* = F_q \backslash \{0\}$. We proceed by induction on l in $\mathrm{Tr}(\alpha^{m+l}B)$.

1. When $l = 0$,

$$\mathrm{Tr}(\alpha^m B) = \mathrm{Tr}\left[\left(-\sum_{i=0}^{k-2} f_{e_i}\alpha^{e_i}\right) B\right] = -\sum_{i=0}^{k-2} f_{e_i}\mathrm{Tr}(\alpha^{e_i}B) = -\sum_{i=0}^{k-2} f_{e_i}b^*_{e_i}.$$

   Since $f_{e_i}, b^*_{e_i} \in \mathbb{F}_q$, obviously, $k-1$ constant multiplications in $\mathbb{F}_q$ and $k-2$ additions in $\mathbb{F}_q$ are required to generate $\sum_{i=0}^{k-2} f_{e_i}b^*_{e_i}$, and one constant subtraction operation for $\mathrm{Tr}(\alpha^m B) = -\sum_{i=0}^{k-2} f_{e_i}b^*_{e_i}$.

2. Assume that $\mathrm{Tr}(\alpha^{m+l'}B)$ has been generated with $k-1$ constant multiplications in $\mathbb{F}_q$, $k-2$ additions and one constant subtraction in $\mathbb{F}_q$, for each of $l' = 0, 1, \ldots, l-1$. Then

$$\mathrm{Tr}(\alpha^{m+l}B) = \mathrm{Tr}[\alpha^l(\alpha^m)B] = \mathrm{Tr}\left[\alpha^l\left(-\sum_{i=0}^{k-2} f_{e_i}\alpha^{e_i}\right)B\right] = -\sum_{i=0}^{k-2} f_{e_i}\mathrm{Tr}(\alpha^{l+e_i}B).$$

For each of $i = 0, 1, \ldots, k-2$, when $0 < l + e_i < m$, $\mathrm{Tr}(\alpha^{l+e_i}) = b^*_{l+e_i}$ is available as a certain coordinate of input B; when $m \leqslant l + e_i < l + m$, $\mathrm{Tr}(\alpha^{l+e_i})$ has already been generated as assumed. Therefore, $\mathrm{Tr}(\alpha^{m+l})$ can be generated with $k-1$ additions (subtractions) in $\mathbb{F}_q$ and $k-1$ multiplications in $\mathbb{F}_q$.

We conclude that a realization of $\mathrm{Tr}(\alpha^{m+l})$, $l = 0, 1, \ldots, m-2$, requires at most $(k-1)(m-1)$ constant multiplications in $\mathbb{F}_q$, $(k-2)(m-1)$ additions and $m-1$ constant additions in $\mathbb{F}_q$. Thus the lemma holds. □

Corollary 4.1 Let $f(x)$ be given as in Lemma 4.2 and q a power of 2. Then a parallel weakly dual basis multiplier in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ can be constructed with m^2 multipliers and $(k-1)(m-1)$ constant multipliers in $\mathbb{F}_q$, and at most $m^2 + (k-3)m - (k-2)$ adders (or subtractors) in $\mathbb{F}_q$.

Proof: Since the field has characteristic of 2, $\mathrm{Tr}(\alpha^{m+l}B) = -\mathrm{Tr}(\alpha^{m+l}B)$ for $l = 0, 1, \ldots, m-2$. The corollary follows by noting that the $m-1$ constant subtractors in $\mathbb{F}_q$ are not needed, compared to the case of Lemma 4.2. □

When $q = 2$ we obtain the same results as Fenn, Benaissa and Taylor proposed in [23]:

Corollary 4.2 Let $f(x)$ be a k term irreducible polynomial over $\mathbb{F}_2$ of degree m. Then a bit-parallel weakly dual basis multiplier in $\mathbb{F}_{2^m}$ can be constructed with m^2 AND gates and at most $m^2 + (k-3)m - (k-2)$ XOR gates.

4.2.3 Algorithm and architecture

An algorithm for parallel multiplication in $\mathbb{F}_{q^m}$, which conforms to the complexity bound discussed in the previous section, can be given as follows.

Algorithm 4.2 Modified Parallel Multiplication in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$

Input: Coefficients of an irreducible polynomial: $\langle f_0, f_1, \ldots, f_{m-1}, f_m = 1\rangle$;

Two field elements $\langle a_0, a_1, \ldots, a_{m-1}\rangle$, and $\langle b_0^*, b_1^*, \ldots, b_{m-1}^*\rangle$;

Output: The product $\langle c_0^*, c_1^*, \ldots, c_{m-1}^*\rangle$.

(Precompute $r_{l,j}$ for $l = 0, 1, \ldots, m - e_{k-2}$ and $j = 0, 1, \ldots, m-1$;)

1a. Compute:

$$\mathrm{Tr}(\gamma\alpha^{m+l}) = \sum_{i=0}^{m-1} r_{l,i} b_i^*, l = 0, 1, \ldots, m - e_{k-2} - 1;$$

1b. Compute:

$t = 1$;

Do

For $i = 0, 1, \ldots, m - e_{k-2} - 1$, if $(ti < e_{k-2} - 1)$

Compute $\mathrm{Tr}(\gamma\alpha^{2m-e_{k-2}+ti})$ for $i = 0, 1, \ldots, m - e_{k-2} - 1$;

$t = t + 1$; } While $(ti < e_{k-2} - 1)$;

2. Compute:

$c_{j,i}^* a_i$ for $i, j = 0, 1, \ldots, m-1$;

3. Compute:

$$c_j^* = \sum_{i=0}^{m-1} c_{j,i}^* a_i \text{ for } j = 0, 1, \ldots, m-1.$$ □

In order to obtain $\mathrm{Tr}(\gamma\alpha^{m+j})$, Algorithm 4.2 allows us to use certain $\mathrm{Tr}(\gamma\alpha^{m+i})$ terms which have already been generated in the generation of some other signals $\mathrm{Tr}(\gamma\alpha^{m+j})$. Consequently, generating $\mathrm{Tr}(\gamma\alpha^{m+j})$ should be arranged as follows: First, terms $\mathrm{Tr}(\gamma\alpha^{m+j})$ with $j = 0, 1, \ldots, \min(m - e_{k-2} - 1, 2m-2)$ are generated. Then $\mathrm{Tr}(\gamma\alpha^{m+m-e_{k-2}+j})$ with $j = 0, 1, \ldots, \min(m - e_{k-2} - 1, 2m-2)$ are generated where the previously generated terms $\mathrm{Tr}(\gamma\alpha^{m+j})$ are used as available inputs, and so on. An implementation

procedure for a parallel multiplier in $\mathbb{F}_{q^m}$ can be summarized as the following three steps:

Scheme 4.1 Implementation of parallel multiplier in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$

1. Generate $c^*_{j,i} = \mathrm{Tr}(\gamma\alpha^{i+j}B)$ for $n \leqslant i+j \leqslant 2m-2$.

   1a. Generate those $\mathrm{Tr}(\gamma\alpha^{m+l}B)$ which are to be reused in some other $\mathrm{Tr}(\gamma\alpha^{m+l'}B)$.

   1b. Generate those $\mathrm{Tr}(\gamma\alpha^{m+l}B)$ which reuse some other $\mathrm{Tr}(\gamma\alpha^{m+l'}B)$.

2. Generate $c^*_{j,i}a_i$, for $i, j = 0, 1, \ldots, m-1$, with m multipliers in $\mathbb{F}_q$.

3. Generate $c^*_j = \sum_{i=0}^{m-1} c^*_{j,i}a_i$ for $j = 0, 1, \ldots, m-1$ by using m binary tree network of $m-1$ adders in $\mathbb{F}_q$.

□

When $q = 2$, Fenn, Benaissa and Taylor proposed the similar implementation procedure [23], where Step 1 is implemented with Module B and Steps 2 and 3 are realized with Module A (see [23]).

At Step 1 of Algorithm 4.2, the longest propagation delay occurs at $\mathrm{Tr}(\gamma\alpha^{2m-2})$ and it is $tT^c_{\otimes} + t\lceil\log_2(H(f)-1)\rceil T_{\oplus}$, where $T^c_{\otimes}$ and $T_{\oplus}$ denote the propagation delays incurred with a constant multiplier and an adder in $\mathbb{F}_q$, respectively, and t can be solved from Step 1b in Algorithm 4.2 as follows:

$$(2m-2) - t(m - e_{k-2}) \leqslant m-1$$

$$\Rightarrow \quad t \geqslant \frac{m-1}{m-e_{k-2}} \quad \Rightarrow \quad t = \left\lceil \frac{m-1}{m-e_{k-2}} \right\rceil .$$

Considering the time delay incurred at Steps 2 and 3, the time complexity of the parallel multiplier is

$$tT_{\otimes}^{c} + T_{\otimes} + \{t\lceil \log_2(H(f)-1)\rceil + \lceil \log_2 m\rceil\}T_{\oplus}, \tag{4.6}$$

where $T_{\otimes}$ denotes the time delay with a multiplier in $\mathbb{F}_q$.

4.2.4 Architecture with reduced time delay

The propagation delay of the multiplier presented above can be further reduced if the terms in the parallel multiplier are arranged properly. This can be illustrated in the following example.

Let two bases $\langle 1, \alpha, \alpha^2, \alpha^3, \alpha^4\rangle$ and $\langle \beta_0, \beta_1, \beta_2, \beta_3, \beta_4\rangle$ be weakly dual bases in F_{7^5} over F_7, where α is a root of irreducible polynomial $f(x) = x^5 + 2x^3 + 5$. Let A and B be two field elements $\in F_{7^5}$ given by $A = \sum_{i=0}^{4} a_i\alpha^i$ and $B = \sum_{i=0}^{4} b_i^*\beta_i$, where a_i and b_i^* are the ith coordinates of A with respect to the polynomial basis and of B with respect to the weakly dual basis, respectively. Let their product C be denoted by $C = \sum_{i=0}^{4} c_i^*\beta_i$, where c_i^* is the ith coordinate of C with respect to the weakly dual basis.

From (4.3) and Step 1 in Algorithm 4.2, we can write

$$\begin{aligned}
&c_{j,i}^* = b_{i+j}^*, \text{ for } i+j = 0, 1, 2, 3, 4;\\
&c_{1,4}^* = c_{2,3}^* = c_{3,2}^* = c_{4,1}^* = \mathrm{Tr}(\gamma\alpha^5 B) = 5b_0^* + 2b_3^*;\\
&c_{2,4}^* = c_{3,3}^* = c_{4,2}^* = \mathrm{Tr}(\gamma\alpha^6 B) = 5b_1^* + 2b_4^*;\\
&c_{3,4}^* = c_{4,3}^* = \mathrm{Tr}(\alpha^7 B) = 5b_2^* + 2c_{1,4}^*;\\
&c_{4,4}^* = \mathrm{Tr}(\alpha^8 B) = 5b_3^* + 2c_{2,4}^*.
\end{aligned}$$

When all $c^*_{j,i}a_i$, for $i, j = 0, 1, 2, 3, 4$, have been generated, then we have

$$\begin{aligned} c^*_0 &= b^*_0 a_0 + b^*_1 a_1 + b^*_2 a_2 + b^*_3 a_3 + b^*_4 a_4, \\ c^*_1 &= b^*_1 a_0 + b^*_2 a_1 + b^*_3 a_2 + b^*_4 a_3 + c^*_{1,4} a_4, \\ c^*_2 &= b^*_2 a_0 + b^*_3 a_1 + b^*_4 a_2 + c^*_{1,4} a_3 + c^*_{2,4} a_4, \\ c^*_3 &= b^*_3 a_0 + b^*_4 a_1 + c^*_{1,4} a_2 + c^*_{2,4} a_3 + c^*_{3,4} a_4, \\ c^*_4 &= b^*_4 a_0 + c^*_{1,4} a_1 + c^*_{2,4} a_2 + c^*_{3,4} a_3 + c^*_{4,4} a_4. \end{aligned} \tag{4.7}$$

If we implement (4.7) using Scheme 1, from (4.6) it has a time delay of $2T^c_\otimes + T_\otimes + 5T_\oplus$.

Categorizing all the terms occurred at the righthand side of (4.7) into three sets, one can obtain:

$$\begin{aligned} S_1 &\triangleq \{b^*_i a_j | i, j = 0, 1, 2, 3, 4\}, \\ S_2 &\triangleq \{c^*_{j,i} a_j | j = 1, 2, 3, 4; i + j = 5, 6\}, \\ S_3 &\triangleq \{c^*_{j,i} a_j | j = 3, 4; i + j = 7, 8\}. \end{aligned}$$

It can be seen that the generation of the signals belonging to S_1, S_2, and S_3 requires a time delay of $T_\otimes$, $T^c_\otimes + T_\otimes + T_\oplus$, and $2T^c_\otimes + T_\otimes + 2T_\oplus$, respectively. Then (4.7) can be written as

$$c^*_j = (c^*_{j,S_1} + c^*_{j,S_2}) + c^*_{j,S_3}, \quad j = 0, 1, 2, 3, 4,$$

where signal c^*_{j,S_i} denotes the sum of those terms that belong to S_i. It can be seen that the longest propagation delay of the multiplier occurs to generate c^*_4 which is $2T^c_\otimes + T_\otimes + 4T_\oplus$

(see Figure 4.1).

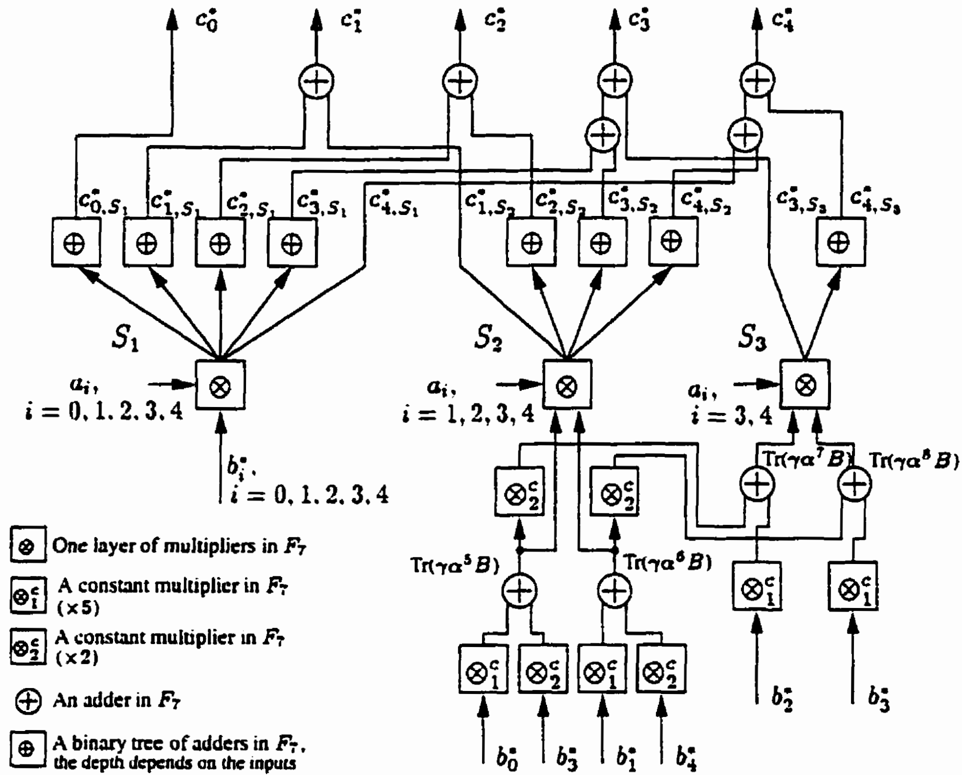

Figure 4.1: Architecture of a parallel multiplier in F_{7^5} over F_7 when $f(x) = x^5 + 2x^3 + 5 \in F_7[x]$.

4.3 ESP Based Bit-Parallel Multiplier in $\mathbb{F}_{2^m}$

4.3.1 Algorithm

Let $f(x)$ be an irreducible r-ESP of degree m, *i.e.*, $f(x) = 1 + x^r + x^{2r} + \cdots + x^{tr} + x^m$, where $m = (t+1)r$, for a root α of $f(x)$ we obtain the equations for α^{m+j}, $j =$

$0, 1, \ldots, m-2$, as follows.

$$\begin{aligned} \alpha^{m+j} &= \alpha^j + \alpha^{r+j} + \cdots + \alpha^{tr+j} && \text{for } 0 \leqslant j \leqslant r-1, \\ \alpha^{m+r+j} &= \alpha^j && \text{for } 0 \leqslant j \leqslant m-r-2. \end{aligned}$$

Applying the trace function $\mathrm{Tr}(\gamma \cdot B)$ to the above equations we have

$$\begin{aligned} \mathrm{Tr}(\gamma\alpha^{m+j}B) &= b_j^* + b_{r+j}^* + \cdots + b_{tr+j}^* && \text{for } 0 \leqslant j \leqslant r-1 \\ \mathrm{Tr}(\gamma\alpha^{m+r+j}B) &= b_j^* && \text{for } 0 \leqslant j \leqslant m-r-2. \end{aligned}$$

From (4.3), it follows

$$c_{j,i}^* = \begin{cases} b_{i+j}^*, & 0 \leqslant i+j \leqslant m-1, \quad (4.8a) \\ b_{i+j-m}^* + b_{i+j-m+r}^* + \cdots + b_{i+j-m+tr}^*, & m \leqslant i+j \leqslant m+r-1, \quad (4.8b) \\ b_{i+j-m-r}^*, & m+r \leqslant i+j \leqslant 2m-2 \quad (4.8c) \end{cases}$$

Clearly, the complexity of a bit-parallel WDB multiplier completely depends on identities (4.2) and (4.8). It can be readily seen that when $c_{j,i}^*$'s are given, a realization of (4.2) requires m^2 AND gates and $m(m-1)$ XOR gates and has a time delay[a] of $T_A + \lceil \log_2 m \rceil T_X$, where T_A and T_X denote the delays caused by one AND gate and by one XOR gate, respectively. Another time delay of $\lceil \log_2(t+1) \rceil T_X = \lceil \log_2 \frac{m}{r} \rceil T_X$ is required to generate those $c_{j,i}^*$'s with $m \leqslant i+j \leqslant m+r-1$. The number of XOR gates required is tr. For $0 \leqslant i+j \leqslant m-1$ and $m+r \leqslant i+j \leqslant 2m-2$, $c_{j,i}^*$'s are certain coordinates of B with respect to the WDB and are available without any cost of gates and gate delays.

[a]We assume that for the generation of c_j^*, the $m-1$ XOR gates are connected in the binary tree form.

We summarize the above discussions as follows: In the finite field $\mathbb{F}_{2^m}$ constructed with an irreducible r-ESP, at most m^2 AND gates and $m^2 - r$ XOR gates are required to build a bit-parallel multiplier. The time delay incurred is at most $T_A + \left(\lceil \log_2 m \rceil + \lceil \log_2 \frac{m}{r} \rceil\right) T_X$. In most cases, however, the time delay of the multiplier can be further reduced as it is discussed in the next section.

4.3.2 Implementation

Let us consider the case when $f(x)$ is an irreducible r-ESP. Then $c^*_{j,i}$ is given by (4.8) and a bit-parallel multiplier can be implemented using the following three steps.

1. Obtain $c^*_{j,i}$ for $i, j = 0, 1, \ldots, m-1$ using (4.8). In this step, logic gates are needed only to generate signals $\sum_{l=0}^{t} b^*_{i+j-m+lr} = \sum_{l=0}^{t} b^*_{k+lr}$, for $k = 0, 1, \ldots, r-1$, where $k = i + j - m$. Each of these signals can be reused for $m - k - 1$ $c^*_{j,i}$'s, since $c^*_{j,i} = c^*_{j,m-j+k}$ and j can be any value of $k+1, k+2, \ldots, m-1$. The number of XOR gates needed is $rt = m - r$ and the time delay incurred is $\lceil \log_2(t+1) \rceil$.

2. Generate $c^*_{j,i}a_i$, for $i, j = 0, 1, \ldots, m-1$, where m^2 two-input AND gates are used and the incurred time delay is T_A.

3. Obtain $c^*_j = \sum_{i=0}^{m-1} c^*_{j,i}a_i$ for $j = 0, 1, \ldots, m-1$. Each c^*_j can be implemented with a binary tree network of $m - 1$ XOR gates. The time delay is $\lceil \log_2 m \rceil T_X$.

Notice that for $0 \leqslant i+j \leqslant m-1$ and $m+r \leqslant i+j \leqslant 2m-2$, $c^*_{j,i}$'s are certain WDB coordinates of the element B and are already available. We may let each of these $c^*_{j,i}$'s enter an AND gate (Step 2) before the rest of the $c^*_{j,i}$'s are generated and thus the total

time delay of the multiplier might be reduced. This can be illustrated by the multiplier structure shown in Figure 4.2.

Block B1 generates $c^*_{j,m-j+k} = \sum_{l=0}^{t} b^*_{lr+k}$ for $k+1 \leqslant j \leqslant m-1$ and $0 \leqslant k \leqslant r-1$ in accordance with (4.8b). This block consists of $tr = m - r$ XOR gates and has a propagation delay of $\lceil \log_2(t+1) \rceil T_X = \lceil \log_2 \frac{m}{r} \rceil T_X$.

The $c^*_{j,i}$ terms obtained from B1 are ANDed with $a_i, 1 \leqslant i \leqslant m-1$ in block B2 to generate $c^*_{j,i}a_i$ for $m \leqslant i+j \leqslant m+r-1$. For these particular constraints on i and j, the number of $c^*_{j,i}a_i$ terms is $mr - \frac{r(r+1)}{2}$. Consequently, block B2 consists of $mr - \frac{r(r+1)}{2}$ AND gates and has a propagation delay of T_A only. For convenience, the outputs of B2 are shown at $m-1$ ports, viz., $P_1^{(2)}, P_2^{(2)}, \ldots, P_{m-1}^{(2)}$. The number of $c^*_{j,i}a_i$ terms at port $P_j^{(2)}$ is j if $1 \leqslant j \leqslant r-1$ and r if $r \leqslant j \leqslant m-1$.

From (4.3), the number of $c^*_{j,i}a_i$ terms for $0 \leqslant i, j \leqslant m-1$ is m^2. The $c^*_{j,i}a_i$ terms, which are not generated in B2, are generated in block B3. The latter takes a_i and b^*_i, $0 \leqslant i \leqslant m-1$ as inputs. As given in (4.8a) and (4.8c), the b^*_i's are directly related to $c^*_{j,i}$ for $0 \leqslant i+j \leqslant m-1$ and $m+r \leqslant i+j \leqslant 2m-2$. The B3 block outputs ANDed terms $c^*_{j,i}a_i$ for these values of i and j. This requires $m^2 - mr + \frac{r(r+1)}{2}$ AND gates and a propagation delay of T_A. Similar to B2, the outputs of B3 are shown at m ports, namely, $P_0^{(3)}, P_1^{(3)}, \ldots, P_{m-1}^{(3)}$. Let the number of $c^*_{j,i}a_i$ terms at port $P_j^{(3)}$ be denoted by $N_{P_j^{(3)}}$. Then

$$N_{P_j^{(3)}} = \begin{cases} m-j & \text{if } 0 \leqslant j \leqslant r-1, \\ m-r & \text{if } r \leqslant j \leqslant m-1. \end{cases}$$

The $c^*_{j,i}a_i$ terms obtained through port $P_j^{(3)}$ are then partially added in block B4 using XOR gates that are arranged in binary tree forms with $\lceil \log_2(t+1) \rceil$ levels. This is to ensure that the sum of the propagation delays of B3 and B4 is equal to that of B1 and B2. The outputs from B4 are presented at ports $P_0^{(4)}, P_1^{(4)}, \ldots, P_{m-1}^{(4)}$. If $N_{P_j^{(4)}}$ denotes the number of outputs at port $P_j^{(4)}$, then

$$N_{P_j^{(4)}} = \left\lceil \frac{N_{P_j^{(3)}}}{2^{\lceil \log_2(t+1) \rceil}} \right\rceil = \begin{cases} \left\lceil \frac{m-j}{2^{\lceil \log_2(t+1) \rceil}} \right\rceil & \text{if } 0 \leqslant j \leqslant r-1, \\ \\ \left\lceil \frac{m-r}{2^{\lceil \log_2(t+1) \rceil}} \right\rceil & \text{if } r \leqslant j \leqslant m-1. \end{cases}$$

The outputs from the corresponding ports of B2 and B4 are then added in block B5 using XOR gates to generate c_j^* (eqn (4.3)). The maximum propagation delay in B5 is

$$\max\left\{ \max_{0 \leqslant j \leqslant r-1} \left\{ \left\lceil \log_2 \left(j + \frac{m-j}{2^{\lceil \log_2(t+1) \rceil}} \right) \right\rceil T_X \right\}, \left\lceil \log_2 \left(r + \frac{m-r}{2^{\lceil \log_2(t+1) \rceil}} \right) \right\rceil T_X \right\}$$

$$= \left\lceil \log_2 \left(r + \frac{m-r}{2^{\lceil \log_2(t+1) \rceil}} \right) \right\rceil T_X.$$

Since m^2 ANDed terms, *i.e.*, $c^*_{j,i}a_i, 0 \leqslant i,j \leqslant m-1$, are added in B4 and B5 to generate m outputs $c_j^*, 0 \leqslant j \leqslant m-1$, the total number of XOR gates in B4 and B5 combined is $m(m-1)$.

As an example, the details of the blocks used in Figure 4.2, are shown in Figure 4.3 with irreducible 3-ESP $x^6 + x^3 + 1$ of degree 6.

The result of the above discussions can be summarized in a theorem as given below.

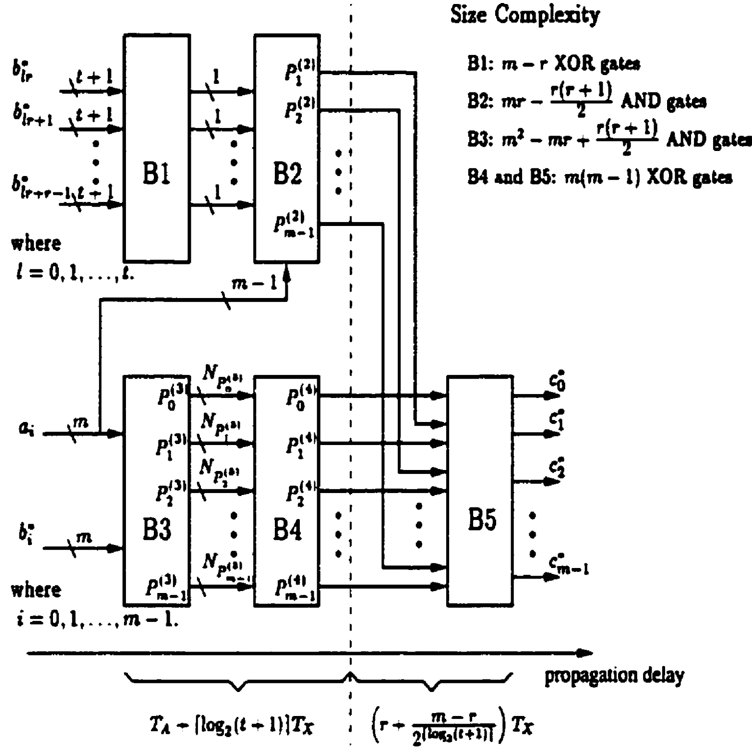

Figure 4.2: Architecture for a parallel multiplier when $f(x)$ is an r-ESP.

Theorem 4.3 If the finite field $\mathbb{F}_{2^m}$ is defined by an irreducible r-ESP, *i.e.*, $f(x) = 1 + x^r + x^{2r} + \cdots + x^{tr} + x^m$, where $m = (t+1)r$, then, a bit-parallel WDB multiplier can be implemented with m^2 AND gates, $m^2 - r$ XOR gates, and time delay of $T_A + \left\{\lceil\log_2 \frac{m}{r}\rceil + \left\lceil\log_2\left(r + \left\lceil\frac{m-r}{2^{\lceil\log_2 \frac{m}{r}\rceil}}\right\rceil\right)\right\rceil\right\} T_X$.

It can be seen that irreducible $\frac{m}{2}$-ESP is a trinomial $x^m + x^{\frac{m}{2}} + 1$. In this case, both the size and the time complexities achieve the minimum: Numbers of required AND and XOR gates are at most m^2 and $m^2 - \frac{m}{2}$, respectively, and incurred time delay is $T_A + \left\lceil\log_2\left(m + \left\lceil\frac{m}{2}\right\rceil\right)\right\rceil T_X$. In the AOP case, $r = 1$ and such a multiplier requires m^2 AND gates and $m^2 - 1$ XOR gates and has a time delay of $T_A + (1 + \lceil\log_2 m\rceil)T_X$.

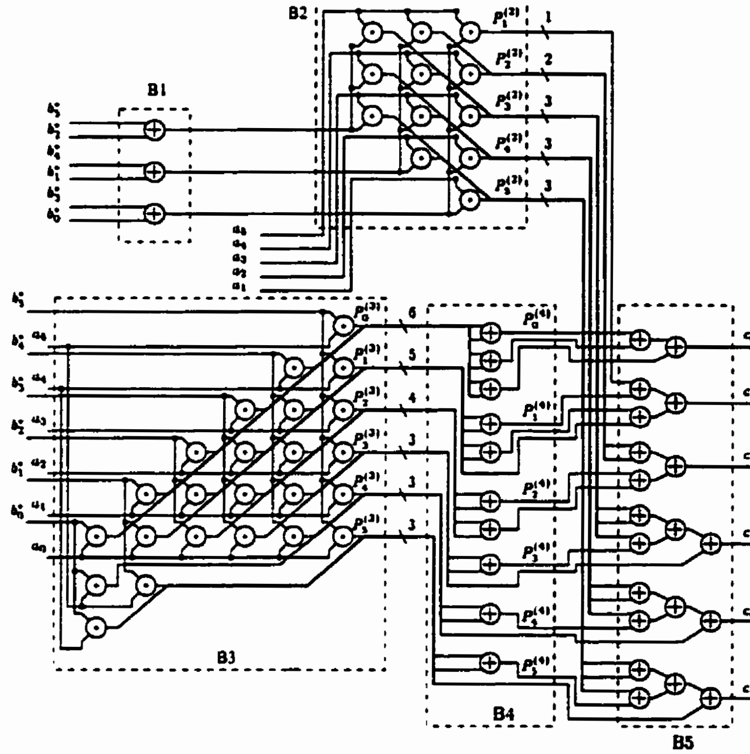

Figure 4.3: Multiplier structure when $f(x) = x^6 + x^3 + 1$.

4.4 Trinomial Based Bit-Parallel Multiplier in $\mathbb{F}_{2^m}$

4.4.1 Analysis of multiplier complexity

Corollary 4.2 has shown that a parallel WDB multiplier can be constructed with m^2 AND gates and at most $m^2 - 1$ XOR gates when $f(x)$ is an irreducible trinomial.

Since the size and time complexities of a bit-parallel WDB multiplier are determined by (4.5) and (4.2), we shall use $\mathrm{C}_{SA}^{(1)}$, $\mathrm{C}_{SX}^{(1)}$, $\mathrm{C}_{T}^{(1)}$ and $\mathrm{C}_{SA}^{(2)}$, $\mathrm{C}_{SX}^{(2)}$, $\mathrm{C}_{T}^{(2)}$ to denote the contributions to the size complexity and the time complexity of a circuitry realizing (4.5)

and (4.2), respectively.

It can be readily seen that an implementation of (4.2) requires m^2 AND gates and $m^2 - m$ XOR gates and has a time delay of $T_A + \lceil \log_2 m \rceil T_X$, or $C_{SA}^{(2)} = m^2, C_{SX}^{(2)} = m^2 - m, C_T^{(2)} = T_A + \lceil \log_2 m \rceil T_X$. Now consider eqn (4.4): $(\alpha^m, \alpha^{m+1}, \ldots, \alpha^{2m-2})^T = \boldsymbol{R}(1, \alpha, \ldots, \alpha^{m-1})^T$, where α is a root of $f(x) = 1 + x^k + x^m$.

(i) When $k = 1$, one can write

$$\alpha^{m+j} = \alpha^j + \alpha^{1+j} \quad \text{for } 0 \leqslant j \leqslant m-2. \tag{4.9}$$

Thus each row of $\boldsymbol{R}$ has only two nonzero entries. Then from (4.5) we have $\text{Tr}(\gamma\alpha^{m+j}B) = b_j^* + b_{1+j}^*$ for $j = 0, 1, \ldots, m-2$. Clearly, $C_{SX}^{(1)} = m - 1$, and $C_T^{(1)} = T_X$.

(ii) When $1 < k < \frac{m}{2}$ or $m - k > k$. According to arithmetic modulo a polynomial, $\alpha^{m+j}, j = 0, 1, \ldots, m-2$, can be computed as follows

$$\begin{aligned} \alpha^{m+j} &= \alpha^j + \alpha^{k+j}, & j \in S_a, \quad &(4.10a) \\ \alpha^{m+(m-k)+j} &= \alpha^j + \alpha^{k+j} + \alpha^{(m-k)+j}, & j \in S_b, \quad &(4.10b) \end{aligned}$$

where S_a denotes the set $\{0, 1, \ldots, m-k-1\}$, and $S_b \triangleq \{0, 1, \ldots, k-2\}$.

Applying trace function $\text{Tr}(\gamma \cdot B)$ to both sides of (4.10a) and (4.10b), we find

$$\begin{aligned} \text{Tr}(\gamma\alpha^{m+j}B) &= b_j^* + b_{k+j}^*, & j \in S_a, \quad &(4.11a) \\ \text{Tr}(\gamma\alpha^{m+(m-k)+j}B) &= \boldsymbol{b_j^* + b_{k+j}^*} + b_{(m-k)+j}^*, & j \in S_b. \quad &(4.11b) \end{aligned}$$

It is readily seen that the first two terms on the right side of (4.11b) (in boldface) are

exactly the same ones as those on the right side of (4.11a). Notice that $S_b \subset S_a$ ($\because m > 2k$), thus we can reuse the term $\mathrm{Tr}(\gamma\alpha^{m+j}B) = b_j^* + b_{k+j}^*, j \in S_a$, of (4.11a) in (4.11b) and it follows (reused terms in boldface)

$$
\begin{aligned}
\mathrm{Tr}(\gamma\alpha^{m+j}B) &= b_j^* + b_{k+j}^*, & j \in S_a, \quad (4.12a)\\
\mathrm{Tr}(\gamma\alpha^{m+(m-k)+j}B) &= \mathbf{Tr}(\boldsymbol{\gamma\alpha^{m+j}B}) + b_{(m-k)+j}^*, & j \in S_b. \quad (4.12b)
\end{aligned}
$$

Clearly, computing (4.12a) and (4.12b) requires $m-k$ and $k-1$ bit additions, respectively. Hence, $\mathbf{C}_S^{(1)} = m-k+k-1 = m-1$ XOR gates. The longest time delay occurs at (4.12b) with the first T_X time delay to carry out $\mathrm{Tr}(\gamma\alpha^{m+j}B) = b_j^* + b_{k+j}^*$ and a second T_X delay to accomplish $\mathrm{Tr}(\gamma\alpha^{m+j}B) + b_{(m-k)+j}^*$ for $0 \leqslant j \leqslant k-2$. Thus, $\mathbf{C}_T^{(1)} = 2T_X$.

(iii) When $k = \frac{m}{2}$, m even, we have

$$
\begin{aligned}
\alpha^{m+j} &= \alpha^j + \alpha^{m-k+j}, & 0 \leqslant j \leqslant k-1 \quad (4.13a)\\
\alpha^{m+k+j} &= \alpha^j, & 0 \leqslant j \leqslant k-2. \quad (4.13b)
\end{aligned}
$$

Applying trace function $\mathrm{Tr}(\gamma \cdot B)$ to both sides of (4.13a) and (4.13b), we obtain

$$
\begin{aligned}
\mathrm{Tr}(\gamma\alpha^{m+j}B) &= b_j^* + b_{m-k+j}^*, & 0 \leqslant j \leqslant k-1, \quad (4.14a)\\
\mathrm{Tr}(\gamma\alpha^{m+k+j}B) &= b_j^*, & 0 \leqslant j \leqslant k-2. \quad (4.14b)
\end{aligned}
$$

Clearly, only (4.14a) requires $k = \frac{m}{2}$ XOR gates and has a time delay T_X. Thus, $\mathbf{C}_{SX}^{(1)} = \frac{m}{2}$ and $\mathbf{C}_T = T_X$.

Now we can summarize the above discussions as follows. When $f(x) = 1 + x^k + x^m$, a WDB multiplier can be constructed with $\mathbf{C}_{SA} = m^2$, and

(i) $\mathbf{C}_{SX} = m^2 - 1$ and $\mathbf{C}_T = T_A + (\lceil \log_2 m \rceil + 1) T_X$ for $k = 1$;

(ii) $\mathbf{C}_{SX} = m^2 - 1$ and $\mathbf{C}_T = T_A + (\lceil \log_2 m \rceil + 2) T_X$ for $1 < k < \frac{m}{2}$;

(iii) $\mathbf{C}_{SX} = m^2 - \frac{m}{2}$ and $\mathbf{C}_T = T_A + (\lceil \log_2 m \rceil + 1) T_X$ for $k = \frac{m}{2}$.

4.4.2 Construction with reduced propagation delay

When $f(x) = x^m + x^k + 1, 1 < k < \frac{m}{2}$, from (4.3), (4.12a) and (4.12b) we obtain

$$\begin{aligned} c^*_{j,i} &= b^*_{i+j-m} + b^*_{i+j-m+k} && \text{for } m \leqslant i+j \leqslant 2m-k-1, \quad (4.15a) \\ c^*_{j,i} &= \mathrm{Tr}(\gamma\alpha^{i+j-m+k}B) + b^*_{i+j-m} \\ &= \mathbf{c^*_{j,i-m+k}} + b^*_{i+j-m} && \text{for } 2m-k \leqslant i+j \leqslant 2m-2. \quad (4.15b) \end{aligned}$$

The boldface terms in (4.15b) can be obtained by reusing the same terms in (4.15a). We put (4.3), (4.15a) and (4.15b) together and it yields

$$c^*_{j,i} = \begin{cases} b^*_{i+j} & \text{for } i+j \in S_c, \quad (4.16a) \\ b^*_{i+j-m} + b^*_{i+j-m+k} & \text{for } i+j \in S_d, \quad (4.16b) \\ \mathbf{c^*_{j,i-m+k}} + b^*_{i+j-m} & \text{for } i+j \in S_e, \quad (4.16c) \end{cases}$$

where $S_c \triangleq \{0, 1, \ldots, m-1\}$, $S_d \triangleq \{m, m+1, \ldots, 2m-k-1\}$, and $S_e \triangleq \{2m-k, 2m-k+1, \ldots, 2m-2\}$.

Notice that in (4.16) it may take different time delays to generate $c^*_{j,i}$ with different values of $i+j$. By exploiting this fact, we might expect that the whole time delay of the

multiplier can be further reduced (for some m and k).

It can be seen that no time delay is incurred for generating $c^*_{j,i}, i+j \in S_c$, but T_X, and $2T_X$ are required to generate $c^*_{j,i}$ for $i+j \in S_d$, and for $i+j \in S_e$, respectively. If those $c^*_{j,i}$'s that are generated earlier are allowed to multiply with a_i first, the overall time delay of the multiplier might be further reduced. A multiplier architecture with reduced time delay can be shown in Figure 4.4.

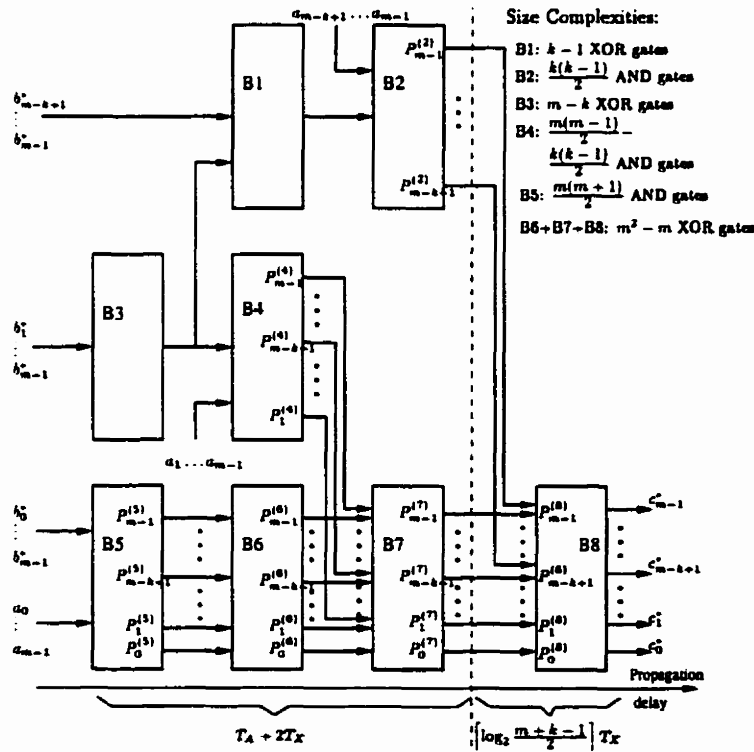

Figure 4.4: Multiplier structure when $f(x) = x^m + x^k + 1, 1 < k < \frac{m}{2}$.

Block B3 generates $c^*_{j,i} = b^*_{i+j-m} + b^*_{i+j-m+k}, i+j \in S_d$, in accordance with (4.16a). This block uses $m-k$ XOR gates and causes a time delay of T_X. The $c^*_{j,i}$ terms obtained from B3 are ANDed with $a_i, 1 \leqslant i \leqslant m-1$ in block B4 to generate $c^*_{j,i}a_i$ for $i+j \in S_d$.

Consequently, $\frac{m(m+1)}{2} - \frac{k(k-1)}{2}$ AND gates are required in B4 and they cause a time delay of T_A. The outputs of B4 are shown at $m-1$ ports, namely, $P_1^{(4)}, \dots, P_{m-1}^{(4)}$. The number of $c^*_{j,i}a_i$ terms at port $P_j^{(4)}$ is denoted by $N_{P_j^{(4)}}$ and

$$N_{P_j^{(4)}} = \begin{cases} j & \text{for } 0 \leqslant j \leqslant m-k, \\ m-k & \text{for } m-k+1 \leqslant j \leqslant m-1. \end{cases}$$

In the sequel, the output ports of block Bi, $i = 2, 4, 5, 6, 7$, are denoted by $P_0^{(i)}, \dots, P_{m-1}^{(i)}$, and the number of binary outputs at port $P_j^{(i)}$ is denoted by $N_{P_j^{(i)}}$.

Block B1 produces $c^*_{j,i} = c^*_{j,i-m+k} + b^*_{i+j-m}, i+j \in S_e$, where $c^*_{j,i-m+k}$ is from the output of B3 and reused by B1, see (4.16c). The number of XOR gates used is $k-1$ and the time delay incurred is $2T_X$ (since the input of the reused signals have already a delay of T_X). Then the output of B1 is multiplied with a_i in B2 to yield $c^*_{j,i}a_i$ for $i+j \in S_e$, where $\frac{k(k-1)}{2}$ AND gates are used and incurred delay is T_A. From (4.16c),

$$N_{P_j^{(2)}} = \begin{cases} 0 & \text{for } 0 \leqslant j \leqslant m-k, \\ j-m+k & \text{for } m-k+1 \leqslant j \leqslant m-1. \end{cases}$$

From (4.2), the number of $c^*_{j,i}a_i$ terms for $0 \leqslant i, j \leqslant m-1$ is m^2. The $c^*_{j,i}a_i$ terms, which are not generated in B2 or B4, are generated in block B5. The latter takes a_i and b^*_i, $0 \leqslant i \leqslant m-1$ as inputs. As given in (4.16a), the b^*_i's are directly related to $c^*_{j,i}$ for $i+j \in S_e$. The B5 outputs ANDed terms $c^*_{j,i}a_i$ for these values of i and j. This requires $\frac{m(m+1)}{2}$ AND gates and a propagation delay of T_A. Clearly, the number of $c^*_{j,i}a_i$ terms at $P_j^{(5)}$ is $N_{P_j^{(5)}} = m-j$.

The $c^*_{j,i}a_i$ terms at port $P_j^{(5)}$ are then partially added in block B6 using one layer of XOR gates. This is to ensure that the sum of the propagation delay of B5 and B6 is equal to that of B3 and B4. The outputs from B6 are presented at ports $P_0^{(6)}, P_1^{(6)}, \ldots, P_{m-1}^{(6)}$ and $N_{P_j^{(6)}} = \left\lceil \frac{N_{P_j^{(5)}}}{2} \right\rceil, j = 0, 1, \ldots, m-1$.

To ensure that the sum of time delays of B5, B6 and B7, or B3, B4 and B7 is equal to that of B3, B1 and B2, block B7 must consist of one layer of XOR gates. Then, we have

$$N_{P_j^{(7)}} = \left\lceil \frac{N_{P_j^{(4)}} + N_{P_j^{(6)}}}{2} \right\rceil, \; j = 0, 1, \ldots, m-1.$$

Finally, the outputs from the corresponding ports of B2 and B7 are added together in block B8 to get c_j. Let the input ports of B8 be $P_0^{(8)}, P_1^{(8)}, \ldots, P_{m-1}^{(8)}$ and the number of binary inputs at $P_j^{(8)}$ be $N_{P_j^{(8)}}$. Then, we have $N_{P_j^{(8)}} = N_{P_j^{(2)}} + N_{P_j^{(7)}}$. Thus, the propagation delay caused in B8 can be computed as

$$\max_{0 \leqslant j \leqslant m-1}\{\lceil \log_2(N_{P_j^{(2)}} + N_{P_j^{(7)}})\rceil T_X\} = \left\lceil \log_2\left(\frac{m+k-1}{2}\right)\right\rceil T_X.$$

When $f(x) = x^m + x^{\frac{m}{2}} + 1$, the if-and-only-if condition for that $f(x)$ is irreducible is that $m = 2 \cdot 3^l, l = 0, 1, 2, \cdots$. This is so because $g(x) = 1 + x + x^2$ is irreducible and by Cohen's theorem [19], $f(x) = g(x^k)$ is irreducible iff $k = 3^l$. This is also a special case of r-ESP when $r = \frac{m}{2}$ which has been discussed in the previous section.

When $f(x) = x^m + x + 1$, from (4.3) and (4.9),

$$c^*_{j,i} = \begin{cases} b^*_{i+j} & \text{for } 0 \leqslant i+j \leqslant m-1, \\ b^*_{i+j-m} + b^*_{i+j-m+1} & \text{for } m \leqslant i+j \leqslant 2m-2. \end{cases}$$

In this case, a block diagram for parallel multiplier is shown in Figure 4.4. It is easy to see that the total time delay is $T_A + (\lceil \log_2 m \rceil + 1)\, T_X$.

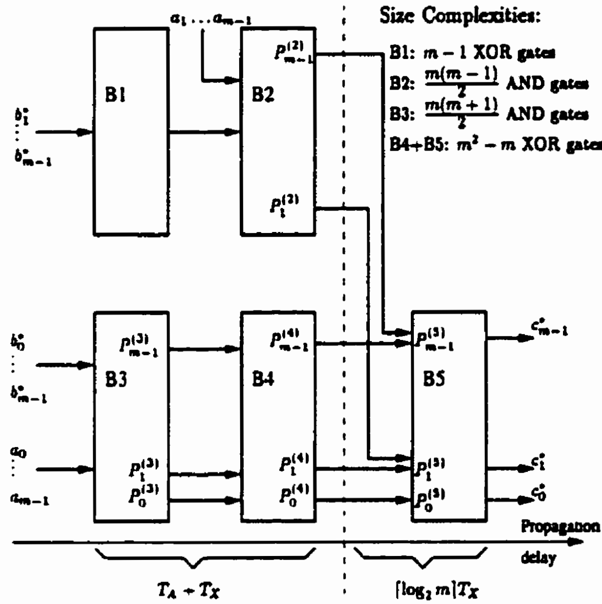

Figure 4.5: Architecture of WDB multiplier when $f(x) = x^m + x + 1$.

We summarize the results for the reduced time delay multipliers in the following theorem.

Theorem 4.4 If finite field $\mathbb{F}_{2^m}$ is generated with an irreducible trinomial $f(x) = 1 + x^k + x^m$, $1 \leqslant k \leqslant \left\lfloor \frac{m}{2} \right\rfloor$, then, a bit-parallel WDB multiplier can be constructed with

$\mathbf{C}_{SA} = m^2$,

(i) $\mathbf{C}_{SX} = m^2 - 1$ and $\mathbf{C}_T = T_A + (\lceil \log_2 m \rceil + 1) T_X$, for $k = 1$;

(ii) $\mathbf{C}_{SX} = m^2 - 1$ and $\mathbf{C}_T = T_A + \left(\left\lceil \log_2 \left\lceil \frac{m+k-1}{2} \right\rceil \right\rceil + 2 \right) T_X$, for $1 < k < \frac{m}{2}$;

(iii) $\mathbf{C}_{SX} = m^2 - \frac{m}{2}$ and $\mathbf{C}_T = T_A + \left\lceil \log_2 \left(m + 2 \left\lceil \frac{m}{4} \right\rceil \right) \right\rceil T_X$, for $k = \frac{m}{2}$.

4.5 Basis Conversion

Since two bases are involved in the WDB multiplication, sometimes it may be necessary to have a basis conversion between the polynomial basis and the WDB, and vice versa. Given an element $A = \sum_{i=0}^{m-1} a_i \alpha^i \in \mathbb{F}_{2^m}$, from (4.1) its jth coordinate with respect to the WDB is given by

$$a_j^* = \mathrm{Tr}(\gamma \alpha^j A) = \sum_{i=0}^{m-1} a_i \mathrm{Tr}(\gamma \alpha^{i+j}),$$

or

$$[a_0^*, \cdots, a_{m-1}^*]^T = \mathbf{T} \cdot [a_0, \cdots, a_{m-1}]^T, \tag{4.18}$$

where basis conversion matrix $\mathbf{T}$ is defined by

$$\mathbf{T} \triangleq \begin{bmatrix} \mathrm{Tr}(\gamma) & \mathrm{Tr}(\gamma\alpha) & \cdots & \mathrm{Tr}(\gamma\alpha^{m-1}) \\ \mathrm{Tr}(\gamma\alpha) & \mathrm{Tr}(\gamma\alpha^2) & \cdots & \mathrm{Tr}(\gamma\alpha^m) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Tr}(\gamma\alpha^{m-1}) & \mathrm{Tr}(\gamma\alpha^m) & \cdots & \mathrm{Tr}(\gamma\alpha^{2m-2}) \end{bmatrix}. \tag{4.19}$$

Lemma 4.1 [60, 77] Let the polynomial basis be defined by an irreducible trinomial $f(x) = x^m + x^k + 1$, then γ can be chosen so that the basis conversion matrix is a permutation back circulant matrix. □

When the field is defined with an irreducible trinomial, from Lemma 4.1 we know that the coordinates of a field element represented with the dual basis is simply a permutation of its polynomial basis coordinates. Then, with a little added cost, a dual basis multiplier can be used as a polynomial basis multiplier.

Lemma 4.2 Let the field be defined with a $d+3$ term irreducible polynomial $f(x) = x^m + x^{k+d} + x^{k+d-1} + \cdots + x^k + 1$. Then we can choose γ such that $\mathbf{T}$ has $m+1+\frac{d(d-1)}{2}$ nonzero entries. □

Proof: Let α be a root of $f(x)$ and let $t_{0,i} = \mathrm{Tr}(\gamma\alpha^i), \gamma \in \mathbb{F}_{2^m}, t_{0,i} \in \mathbb{F}_2$, and $i = 0, 1, \ldots, m-1$. Then when γ runs through all the 2^m elements in $\mathbb{F}_{2^m}$, $(t_{0,0}, t_{0,1}, \ldots, t_{0,m-1})$ will give each of the possible 0-1 sequences of length m once.

Choose $\gamma \in \mathbb{F}_{2^m}^*$ such that $t_{0,0} = t_{0,k} = 1, t_{0,i} = 0$ for $i = 1, 2, \ldots, m-1$ and $i \neq k$. Then we have

$$\begin{aligned}
\mathrm{Tr}(\gamma\alpha^{m+i}) &= \mathrm{Tr}(\gamma\alpha^i) + \mathrm{Tr}(\gamma\alpha^{k+i}) + \cdots + \mathrm{Tr}(\gamma\alpha^{k+d+i}) \\
&= 0 \quad \text{for } 0 \leqslant i \leqslant k-1; \\
\mathrm{Tr}(\gamma\alpha^{m+k}) &= \mathrm{Tr}(\gamma\alpha^k) + \mathrm{Tr}(\gamma\alpha^{2k}) + \cdots + \mathrm{Tr}(\gamma\alpha^{2k+d}) \;=\; 1; \\
\mathrm{Tr}(\gamma\alpha^{m+k+i}) &= \mathrm{Tr}(\gamma\alpha^{k+i}) + \mathrm{Tr}(\gamma\alpha^{2k+i}) + \cdots + \mathrm{Tr}(\gamma\alpha^{2k+d+i}) \\
&= 0 \quad \text{for } 1 \leqslant i \leqslant m-k-d-1; \\
\mathrm{Tr}(\gamma\alpha^{2m-d}) &= \mathrm{Tr}(\gamma\alpha^{m-d}) + \mathrm{Tr}(\gamma\alpha^{m+k-d}) + \cdots + \mathrm{Tr}(\gamma\alpha^{m+k}) \;=\; 1.
\end{aligned}$$

If we assume that $\mathrm{Tr}(\gamma\alpha^{2m-d+i}) = 1$ for $i = 1, 2 \ldots, d-2$ and let the number of nonzero entries in $\mathbf{T}$ be N. Then $N \leqslant 1 + (k-1) + (2m-1-m-k) + (1+2+\cdots+d-1) = m+1+\frac{d(d-1)}{2}$. The '=' holds when $m+k+d-2 < 2m-d$, or $m > k+2d-2$. □

Clearly, for $d = 2$, $f(x) = x^m + x^{k+2} + x^{k+1} + x^k + 1$ is a pentanomial and $N = m+2$, which was discussed by Morii, Kasahara and Whiting [60].

Lemma 4.3 Let the polynomial basis be given by an irreducible r-ESP $f(x) = 1 + x^r + x^{2r} + \cdots + x^{tr} + x^m$ $(r \geqslant 1)$. Then we can choose γ such that the basis conversion matrix $\mathbf{T}$ has $2m - 2r$ ones. □

Proof: Choose $\gamma \in \mathbb{F}_{2^m}$ such that $t_{0,r-1} = 1, t_{0,i} = 0$ for $i = 0, 1, \ldots, m-1$ and $i \neq r-1$. Then we have

$$\begin{aligned}
\mathrm{Tr}(\gamma\alpha^{m+i}) &= \mathrm{Tr}(\gamma\alpha^{i}) + \mathrm{Tr}(\gamma\alpha^{i+r}) + \cdots + \mathrm{Tr}(\gamma\alpha^{i+tr}) \\
&= 0 \quad \text{for } 0 \leqslant i \leqslant r-2; \\
\mathrm{Tr}(\gamma\alpha^{m+r-1}) &= \mathrm{Tr}(\gamma\alpha^{r-1}) + \mathrm{Tr}(\gamma\alpha^{2r-1}) + \cdots + \mathrm{Tr}(\gamma\alpha^{(t+1)r-1}) \;=\; 1; \\
\mathrm{Tr}(\gamma\alpha^{m+i}) &= \mathrm{Tr}(\gamma\alpha^{i-r}) \;=\; 0 \quad \text{for } r \leqslant i \leqslant 2r-2; \\
\mathrm{Tr}(\gamma\alpha^{m+2r-1}) &= \mathrm{Tr}(\gamma^{r-1}) \;=\; 1; \\
\mathrm{Tr}(\gamma\alpha^{m+i}) &= \mathrm{Tr}(\gamma^{i-r}) \;=\; 0 \quad \text{for } 2r \leqslant i \leqslant m-2.
\end{aligned}$$

Clearly, $\mathbf{T}$ has $r + m - r + m - 2r = 2m - 2r$ 1's. □

Wang and Blake have given a simple $\mathbf{T}$ for any polynomial basis [77].

Lemma 4.4 [77] Let the polynomial basis be given by an irreducible polynomial $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$. Then a basis conversion matrix $\mathbf{T}$ can be given as

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & b_m \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & b_m & b_{m+1} & \cdots & b_{2m-4} & b_{2m-3} & b_{2m-2} \end{bmatrix},$$

where $b_{m-1} = 1, b_{i+1} = \sum_{j=m-1}^{i} b_j f_{j-i+(m-1)}, i = m-1, m, \ldots, 2m-2$. □

If we choose $f(x)$ with its second highest order term being x^e, then it can be seen that the number of nonzero entries will not be greater than $m + \frac{(e+1)(e+2)}{2}$. More interesting is that the entries b_i's can be computed serially with a linear feed-forward shift register which can be easily implemented in hardware [77, 34].

4.6 Chapter Summary and Discussions

In this chapter, we have first presented an upper bound on the size complexity of bit-parallel multipliers using an arbitrary field $\mathbb{F}_{2^m}$. Then for classes of fields which are generated with irreducible trinomials, or irreducible ESPs, we have given both the size and time complexities of the bit-parallel multiplier. Implementation issues have been discussed, especially to reduce the time delay incurred by the multiplier. These results compare favorably with those of the recently proposed multipliers of the same classes [35, 23].

Multipliers presented in this chapter are suitable for the cases where the weakly dual

basis representation of one input is available. When the polynomial basis is defined by an irreducible trinomial, our results match exactly those of Fenn, *et al* [23], who have implemented the bit parallel multiplier for small size fields ($m \leqslant 15,\ m \neq 8, 12$). When the polynomial basis is defined by an irreducible AOP, which is a special case of r-ESP with $r = 1$, the complexities of our bit-parallel multiplier in terms of both size and time complexities are equal to those of M_MOM proposed by Hasan, *et. al.* [36] (see Table 4.1(a)). When the finite field is defined by an irreducible r-ESP ($r > 1$), the complexities of our multiplier are significantly lower than those of the previously reported ESP based multipliers as shown in Table 4.1(b). If the field generating polynomial is a trinomial, the results presented in this chapter compare favorably with those of recently proposed multipliers of the same class [23].

Multipliers	Basis used	Number of two-input AND gates	Number of XOR gates	Time delay due to gates
MOM [10]	Normal	$2m^2 - m$	$2m^2 - 2m$	$T_A + (\lceil \log_2 m \rceil + 1) T_X$
ITM [6]	Polynomial	$m^2 + 2m + 1$	$m^2 + 2m$	$T_A + (\lceil \log_2 m \rceil + \lceil \log_2 (m+2) \rceil) T_X$
HWBM [4]	Polynomial	m^2	$m^2 + m - 2$	$T_A + (m + \lceil \log_2 m \rceil) T_X$
M_MOM [5]	Normal	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 m \rceil + 1) T_X$
WDBM	Weakly dual	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 m \rceil + 1) T_X$

(a)

Multipliers	Basis used	Number of two-input AND gates	Number of XOR gates	Time delay due to gates
ITM [6]	Polynomial	$(m+r)^2$	$(m+r)^2 - r$	$T_A + (\lceil \log_2 m \rceil + \lceil \log_2 (m+r+1) \rceil) T_X$
HWBM [4]	Polynomial	m^2	$m^2 + m - 2r$	$T_A + \left(\frac{m}{r} + \lceil \log_2 m \rceil\right) T_X$
WDBM	Weakly dual	m^2	$m^2 - r$	$T_A + \left\{\lceil \log_2 \frac{m}{r} \rceil + \left\lceil \log_2 \left(r + \left\lceil \frac{m-r}{2^{\lceil \log_2 \frac{m}{r} \rceil}} \right\rceil\right) \right\rceil \right\} T_X$

(b)

Table 4.1: (a) Comparison of multipliers based on AOP. (b) Comparison of multipliers based on r-ESP ($r > 1$).

The scope of the proposed architecture, like all other bit-parallel multipliers, appears to be constrained to relatively small-to-medium size fields since the size complexity of $O(m^2)$ makes hardware realization difficult with large values of m which are of interest for cryptographic algorithms, especially, those which are based on the discrete logarithm. The recent advances in the elliptic curve cryptosystems, however, have been making it possible to use relatively small fields for attaining similar level of data security [32]. Moreover, as VLSI and packaging technologies, such as multi-chip-module continues to improve, the proposed bit-parallel multipliers are expected to find potential use in practical applications.

Chapter 5

Parallel Polynomial Basis Multipliers

In this chapter, we present a low complexity algorithm for computing reduction modulo a polynomial. Implementations of polynomial basis multipliers using the new method of modular reduction is proposed in § 5.2.2. New algorithms for squaring in $\mathbb{F}_{2^m}$ are also presented and their implementations are discussed in §5.3.

5.1 Polynomial Basis Multiplication in $\mathbb{F}_{2^m}$

Let the finite field $\mathbb{F}_{2^m}$ be generated with an irreducible r-term polynomial $f(x) = x^m + \sum_{i=0}^{r-2} x^{e_i}$, where $0 = e_0 < e_1 < \cdots < e_{r-2} < m$. Let $A(x) = \sum_{i=0}^{m-1} a_i x^i$ and $B(x) = \sum_{i=0}^{m-1} b_i x^i$ be any two elements in $\mathbb{F}_{2^m}$. Then, $C(x) = \sum_{i=0}^{m-1} c_i x^i \in \mathbb{F}_{2^m}$, the product of $A(x)$ and $B(x)$ can be obtained in two steps:

1. Polynomial multiplication:

$$S(x) = A(x)B(x), \tag{5.1}$$

where $S(x) = \sum_{k=0}^{2m-2} s_k x^k$, and s_k is given by

$$s_k \triangleq \sum_{\substack{i+j=k \\ 0 \leqslant i,j \leqslant m-1}} a_i b_j, \; k = 0, 1, 2, \ldots, 2m-2.$$

2. Reduction modulo the irreducible polynomial:

$$C(x) = S(x) \bmod f(x), \tag{5.2}$$

where $C(x) = \sum_{i=0}^{m-1} c_i x^i$, $c_i \in \mathbb{F}_2$.

Obviously, the complexities of the polynomial basis multiplication in $\mathbb{F}_{2^m}$ are determined by these two steps. The complexity of the first step (polynomial multiplication) is independent of choice of the irreducible polynomial $f(x)$, and it has been shown to be $O(m \log m \log\log m)$ in bit operations [68]. We will show that the complexity of the second step (modular reduction) is $O(rm)$, where r is the Hamming weight of the irreducible polynomial $f(x)$.

5.1.1 Polynomial multiplication

In the first step of PB multiplication (5.1), if $S(x)$ is computed from $A(x)$ and $B(x)$ by the conventional polynomial multiplication method, it requires m^2 multiplications and

$(m-1)^2$ additions in the ground field and the time delay is $T_A + \lceil \log_2 m \rceil T_X$. However, there are some asymptotically faster methods for polynomial multiplication over finite fields [5], such as, the Fast Fourier Transform method [5, 42] and the Karatsuba-Ofman algorithm [41, 2, 63]. They can result in asymptotically fewer bit operations at the expense of longer time delay and certain costly pre- and post-computations. Another technique for polynomial basis multiplication that can combine polynomial multiplication with modulo reduction into one single step is called the Montgomery method [58, 44].

5.1.2 Reduction modulo a polynomial

For modular reduction $C(x) = S(x) \bmod f(x)$, where $\deg f = m$, $\deg S \leqslant 2m-2$ and $\deg C \leqslant m-1$, if the conventional polynomial division method is used, the complexity is $O(m^2)$ in ground field operations. Mastrovito [50] has found that if the irreducible polynomial is chosen properly for $m \leqslant 15, m \neq 8$, the complexity of modulo reduction can be greatly reduced by using some partial sums. Paar [63] has also discussed this issue for certain small values of m. However, their methods are based on computer based exhaustive search and available for only moderately small size fields. In the following, we will present a new algorithm that can perform modulo reduction in $O(rm)$ ground field operations for any irreducible polynomial $f(x)$ with the Hamming weight r.

Theorem 5.1 If the Hamming weight of the irreducible polynomial $f(x)$ is r, then the modular polynomial reduction (5.2) can be done with $(r-1)(m-1)$ bit operations.

Proof: Define

$$\sum_{i=0}^{m+l} s_i x^i \mod f(x) \triangleq \sum_{i=0}^{m-1} t_i^{(l)} x^i, \; l = -1, 0, 1, \ldots, m-2. \tag{5.3}$$

$t_i^{(l)}$'s have the initial values $t_i^{(-1)} = s_i$, and, we try to solve for the 'final' values $t_i^{(m-2)} = c_i$, $i = 0, 1, 2, \ldots, m-1$.

In the following, we shall prove by induction that the complexity of solving $t_i^{(m-2)}, i = 0, 1, 2, \ldots, m-1$, is $(r-1)(m-1)$ bit operations.

When $l = 0$, from (5.3) we have

$$\begin{aligned} \sum_{i=0}^{m-1} t_i^{(0)} x^i = \sum_{i=0}^{m} s_i x^i &= \sum_{i=0}^{m-1} s_i x^i + s_m x^m \\ &= \sum_{i=0}^{m-1} t_i^{(-1)} x^i + s_m[1 + x^{e_1} + \cdots + x^{e_{r-2}}] \end{aligned}$$

Clearly, $t_i^{(0)} = \begin{cases} t_i^{(-1)} + s_m, & \text{if } i = 0, e_1, e_2, \ldots, e_{r-2}, \\ t_i^{(-1)}, & \text{if } 1 \le i \le m-1, \text{ and } i \ne e_1, e_2, \ldots, e_{r-2}. \end{cases}$

It can be seen that $r-1$ bit additions are required for obtaining $t_i^{(0)}$ from $t_i^{(-1)}, i = 0, 1, \ldots, m-1$.

Assume when $l < l'$, $r-1$ bit-additions are required for obtaining $t_i^{(l)}$ from $t_i^{(l-1)}, i = 0, 1, \ldots, m-1$. Then, when $l = l'$, we have

$$\sum_{i=0}^{m-1} t_i^{(l')} s^i = \sum_{i=0}^{m+l'} s_i x^i = \sum_{i=0}^{m+l'-1} s_i x^i + s_{m+l'} x^{m+l'}$$
$$= \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + s_{m+l'} x^{l'} x^m$$
$$= \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + s_{m+l'} x^{l'} [1 + x^{e_1} + \cdots + x^{e_{r-2}}]$$

If $m > l' + e_{r-1}$, then

$$t_i^{(l')} = \begin{cases} t_i^{(l'-1)} + s_{m+l'}, & \text{if } i = l', l' + e_1, \ldots, l' + e_{r'-1}, \\ t_i^{(l'-1)}, & \text{otherwise.} \end{cases}$$

Obviously, $t_i^{(l')}$ can be computed from $t_i^{(l'-1)}$ using $r - 1$ bit additions. Now suppose that $l' + e_{r'-1} < m \leqslant l' + e_{r'}, r' \in \{1, \ldots, r-2\}$ and $e_0 \triangleq 0$, thus it follows

$$\sum_{i=0}^{m+l'} s_i x^i = \{\sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + s_{m+l'} x^{l'} [1 + x^{e_1} + \cdots + x^{e_{r'-1}}]\}$$
$$+ s_{m+l'} x^{l'} [x^{e_{r'}} + x^{e_{r'+1}} + \cdots + x^{e_{r-2}}]$$
$$= \sum_{i=0}^{m-1} t_i^{(l',0)} x^i + s_{m+l'} x^{l'} [x^{e_{r'}} + x^{e_{r'+1}} + \cdots + x^{e_{r-2}}]$$
$$= \sum_{i=0}^{m-1} t_i^{(l',0)} x^i + s_{m+l'} x^{l'+e_{r'}} + \cdots + s_{m+l'} x^{l'+e_{r-2}} \quad (5.4)$$

where $\sum_{i=0}^{m-1} t_i^{(l',0)} \triangleq \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + s_{m+l'} x^{l'}[1 + x^{e_1} + \cdots + x^{e_{r'-1}}]$. Since we have

$$t_i^{(l',0)} = \begin{cases} t_i^{(l'-1)} + s_{m+l'}, & \text{if } i = l', l'+e_1, \ldots, l'+e_{r'-1}, \\ t_i^{(l'-1)}, & \text{otherwise,} \end{cases}$$

it can be seen that r' bit additions are required to obtain $t_i^{(l',0)}$ from $t_i^{(l'-1)}, i = 0, 1, \ldots, m-1$.

In the following we shall prove that $t_i^{(l')}$, $i = 0, 1, \ldots, m-1$, can be obtained from $t_i^{(l',0)}$ with $r - r' - 1$ bit additions. Define

$$\sum_{i=0}^{m-1} t_i^{(l',1)} x^i \triangleq \sum_{i=0}^{m-1} t_i^{(l',0)} x^i + s_{m+l'} x^{l'+e_{r'}} \mod f(x), \; i = 0, 1, \ldots, m-1. \tag{5.5}$$

Since $l' + e_{r'} - m < l'$, we have

$$\sum_{i=0}^{m-1} t_i^{(l'+e_{r'}-m)} x^i = \sum_{i=0}^{m-1} t_i^{(l'+e_{r'}-m-1)} x^i + s_{l'+e_{r'}} x^{l'+e_{r'}} \mod f(x), \; i = 0, 1, \ldots, m-1. \tag{5.6}$$

Since $t_i^{(l'+e_{r'}-m)}$ has been obtained from $t_i^{(l'+e_{r'}-m-1)}$ with $r-1$ bit additions as assumed, comparing (5.5) to (5.6), we can see that (5.5) and (5.6) can be combined together to save bit operations. That is, when $l = l' + e_{r'} - m$, instead of performing (5.6), we perform

$$\sum_{i=0}^{m-1} t_i^{(l'+e_{r'}-m-1)} x^i + (s_{l'+e_{r'}} + s_{m+l'}) x^{l'+e_{r'}} \mod f(x) = \sum_{i=0}^{m-1} t_i^{(l'+e_{r'}-m,*)} x^i \tag{5.7}$$

with r bit additions, while (5.5) can be saved. In the sense of the count of bit operations, we may equivalently say that (5.5) requires one bit addition, while (5.6) still needs $r-1$ bit operations. Similar arguments can be applied to the remaining $r-r'-2$ terms in (5.4). Thus for $l=l'$, $r-1$ bit additions are required for obtaining $t_i^{(l')}$ from $t_i^{(l'-1)}$, for $i=0,1,\ldots,m-1$. Therefore, to compute $t_i^{(l)}$ from $t_i^{(l-1)}, i=0,1,\ldots,m-1$, needs $r-1$ bit additions for any integer l. We conclude that computing $c_i=t_i^{(m-2)}$ from $s_i, i=0,1,\ldots,2m-2$ requires $(m-1)(r-1)$ bit additions. □

Theorem 5.1 can be easily extended to $\mathbb{F}_{q^m}$ as it is stated in Theorem 5.2. A proof for Theorem 5.2 is analogous to that of Theorem 5.1.

Theorem 5.2 If the monic irreducible polynomial $f(x)\in\mathbb{F}_q[x]$ of degree m has the Hamming weight of r, then the modular polynomial reduction in polynomial basis multiplication can be done with $(r-1)(m-1)$ multiplications and $(r-1)(m-1)$ additions in $\mathbb{F}_q$.

5.2 Bit Parallel PB Multipliers

5.2.1 Previous implementations

The earliest parallel polynomial basis (PB) multiplier was suggested by Bartee and Schneider [10]. Depending on the irreducible polynomial, the implementation requires as many as m^3-m two-input adders over $\mathbb{F}_2$ [11]. Some proposals on bit-parallel PB multipliers are suitable for VLSI implementation by using cellular array, systolic array, or other highly regular structures [83, 70], while others with less complexity are based on some specific class of fields such as all one polynomials and equally spaced polynomials which

can potentially simplify the multiplication circuitry [38, 35].

Bit-parallel PB multipliers based on the irreducible trinomial $x^m + x^k + 1$ with $1 \leq k \leq \left\lfloor \frac{m}{2} \right\rfloor$ are attractive because they require fewer gates for modular reduction. Mastrovito has proposed a multiplication algorithm and architecture when $f(x)$ is a trinomial [50]. He has shown that the number of both AND and XOR gates needed is proportional to $2m^2$ when the degree of $f(x)$ is no greater than 15 and not equal to 8. The KOA has also been considered for building bit-parallel finite field multipliers [63, 2]. Paar's implementation has shown that bit-parallel multiplication architectures using the KOA in certain composite fields can have significantly lower complexity, compared to that of Mastrovito's. However, the time delay of the architectures using the KOA can be longer.

5.2.2 Implementation with new method of modulo reduction

If the conventional method for polynomial multiplication is used, then the complexity of a bit parallel multiplier in $\mathbb{F}_{2^m}$ can be described as follows.

Theorem 5.3 Let $f(x)$ be an irreducible r-term polynomial of degree m over $\mathbb{F}_2$. Then PB multiplication in $\mathbb{F}_{2^m}$ can be performed with at most m^2 bit multiplications and $m^2 + (r-3)m - (r-2)$ bit additions.

Proof: It is a direct consequence from Theorem 5.1 when the conventional polynomial multiplication is used for (5.1). □

In the following, we will present an analysis of propagation delay for the bit parallel multiplier when the irreducible polynomial is a trinomial.

Lemma 5.1 If the finite field $\mathbb{F}_{2^m}$ is defined with an irreducible trinomial $f(x) = 1+x^k+x^m$, $1 \leqslant k < \frac{m}{2}$, then a bit-parallel PB multiplier can be constructed with $\mathbf{C}_{SA} = m^2$,

$\mathbf{C}_{SX} = m^2 - 1$ and

$$\mathbf{C}_T = \begin{cases} T_A + (\lceil \log_2 m \rceil + 1)\,T_X, & \text{for} \quad k = 1; \\ T_A + (\lceil \log_2 m \rceil + 2)\,T_X, & \text{for} \quad 1 < k < \frac{m}{2}. \end{cases}$$

Proof: If the conventional polynomial multiplication method is used, an implementation of the first step (5.1) requires m^2 AND gates, $(m-1)^2$ XOR gates and a time delay of $T_A + \lceil \log_2 m \rceil T_X$.

In the following we solve the complexities required for implementing the second step (5.2). Define

$$\sum_{i=0}^{m-1} u_i x^i = \sum_{i=m}^{2m-2} s_i x^i \bmod f(x).$$

(i) When $k = 1$, $f(x) = x^m + x + 1$, we have

$$x^{m+j} \;=\; x^j + x^{1+j}, \qquad 0 \le j \le m-2. \tag{5.8}$$

It can be seen that term $x^0 = 1$ occurs once on the right hand side of (5.8) when $j = 0$, term x^{m-1} occurs once on the right hand side of (5.8) when $j = m-2$, and term x^i, $i = 1, 2, \ldots, m-3$, occurs twice on the right hand side of (5.8) when $j = i-1, i$. Thus, we obtain

$$\begin{aligned} u_0 &= s_m, & j &= 0, \\ u_j &= s_{m+j-1} + s_{m+j}, & j &= 1, 2, \ldots, m-2, \\ u_{m-1} &= s_{2m-2}, & j &= m-2. \end{aligned}$$

Clearly, $C_S^2 = m - 2$, and $C_T^2 = T_X$. The longest delay occurs at the terms $u_j, j = 1, 2, \ldots, m-2$.

(ii) When $k \geqslant 2$, since $f(x) = 1 + x^k + x^m$ and $k < \frac{m}{2}$ or $m - k > k$, we can write the terms x^j, $j = 0, 1, \ldots, m-2$ as follows

$$x^{m+j} = x^j + x^{k+j}, \qquad 0 \le j \le m-k-1, \tag{5.9a}$$

$$x^{m+(m-k)+j} = x^j + x^{k+j} + x^{(m-k)+j}, \quad 0 \le j \le k-2. \tag{5.9b}$$

Let T_1 and T_2 denote the first and second terms on the right hand side of (5.9a), respectively. Let T_3, T_4, and T_5 denote the first, second and third terms on the right hand side of (5.9b), respectively. Note the range of degree of each term. We illustrate their relationship in Fig. 5.1.

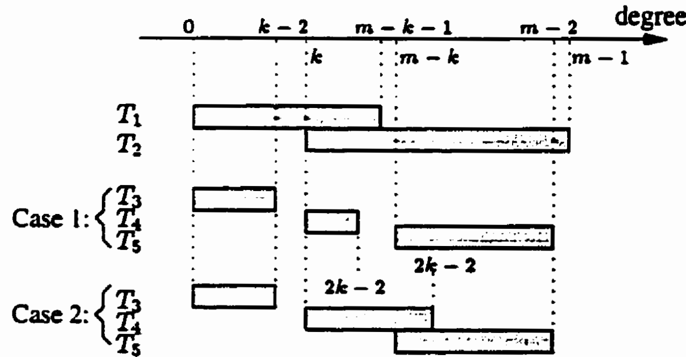

Figure 5.1: Indication of the relation between terms in (5.9a) and (5.9b).

It can be easily seen from Fig. 5.1 that term x^j, $0 \le j \le k-2$, only exists in T_1 and T_3, and term x^{k-1} exists only in T_1. Thus we have

$$u_j = s_{m+j} + s_{m+(m-k)+j}, 0 \le j \le k-2, \text{ and } u_{k-1} = s_{m+k-1}.$$

Case 1: When $2k-2 \leq m-k-1$, we have

$$u_{k+j} = s_{m+j} + s_{m+k+j} + s_{m+(m-k)+j}, \quad 0 \leq j \leq k-2,$$
$$u_{k+j} = s_{m+j} + s_{m+k+j}, \quad k-1 \leq j \leq m-2k-1,$$
$$u_{m-k+j} = s_{m+(m-k)+j} + s_{m+(m-2k)+j}, \quad 0 \leq j \leq k-2,$$
$$u_{m-1} = s_{2m-k-1}.$$

Case 2: When $2k-2 \geq m-k$, we have

$$u_{k+j} = s_{m+j} + s_{m+k+j} + s_{m+(m-k)+j}, \quad 0 \leq j \leq m-2k-1,$$
$$u_{k+j} = s_{m+j} + s_{m+(m-k)+j} + s_{m+k+j}, \quad m-2k \leq j \leq k-2,$$
$$u_{2k-1+j} = s_{m+k-1+j} + s_{m+2k-1+j}, \quad 0 \leq j \leq m-2k-1,$$
$$u_{m-1} = s_{2m-k-1}.$$

Rewrite the above equations to obtain

$$\text{Case 1}: \begin{cases} u_j &= s_{m+j} + s_{m+(m-k)+j}, & 0 \leq j \leq k-2, \\ u_{k-1} &= s_{m+k-1}, & \\ u_{k+j} &= s_{m+j} + s_{m+k+j} + s_{m+(m-k)+j} & \\ &= (s_{m+j} + s_{m+(m-k)+j}) + s_{m+k+j} & \\ &= u_j + s_{m+k+j}, & 0 \leq j \leq k-2, \\ u_{k+j} &= s_{m+j} + s_{m+k+j}, & k-1 \leq j \leq m-2k-1, \\ u_{m-k+j} &= s_{m+(m-k)+j} + s_{m+(m-2k)+j}, & 0 \leq j \leq k-2, \\ u_{m-1} &= s_{2m-k-1}, & \end{cases}$$

and

$$\text{Case 2}: \begin{cases} u_j &= s_{m+j} + s_{m+(m-k)+j}, & 0 \le j \le k-2, \\ u_{k-1} &= s_{m+k-1}, & \\ u_{k+j} &= s_{m+j} + s_{m+k+j} + s_{m+(m-k)+j}, & \\ &= (s_{m+j} + s_{m+(m-k)+j}) + s_{m+k+j}, & \\ &= u_j + s_{m+k+j}, & 0 \le j \le k-2, \\ u_{2k+j} &= s_{m+j} + s_{m+k+j}, & -1 \le j \le m-2k-2, \\ u_{m-1} &= s_{2m-k-1}. & \end{cases}$$

Thus, for both cases we have $\mathbf{C}_S^2 = m - 2$ XOR gates, and $\mathbf{C}_T^2 = 2T_X$. The longest delay occurs when generating the terms u_{k+j}, $0 \le j \le k-2$. The lemma follows by noting $\mathbf{C}_{SA} = \mathbf{C}_S^1 + \mathbf{C}_S^2$ and $\mathbf{C}_T = \mathbf{C}_T^1 + \mathbf{C}_T^2$. □

Lemma 5.2 If the finite field $\mathbb{F}_{2^m}$ is generated with an irreducible trinomial $f(x) = 1 + x^{\frac{m}{2}} + x^m$, then a bit-parallel PB multiplier can be constructed with $\mathbf{C}_{SA} = m^2$, $\mathbf{C}_{SX} = m^2 - \frac{m}{2}$ and $\mathbf{C}_T = T_A + (\lceil \log_2 m \rceil + 1)\, T_X$.

Proof: Since $x^m = 1 + x^{\frac{m}{2}}$, we have

$$\begin{aligned} x^{m+j} &= x^j + x^{\frac{m}{2}+j} & 0 \leqslant j \leqslant \tfrac{m}{2} - 1, \\ x^{\frac{3}{2}m+j} &= x^j & 0 \leqslant j \leqslant \tfrac{m}{2} - 2. \end{aligned}$$

From the above equations, we have

$$\begin{aligned} u_j &= s_{\frac{3}{2}m-1}, & j = \tfrac{m}{2} - 1, \\ u_j &= s_{m+j} + s_{\frac{3}{2}m+j}, & 0 \le j \le \tfrac{m}{2} - 2, \\ u_{k+j} &= s_{m+j}, & 0 \le j \le \tfrac{m}{2} - 1. \end{aligned}$$

Clearly, $C_S^2 = \frac{m}{2} - 1$ XOR gates and $C_T^2 = T_X$. Then, the lemma follows by noting $C_{SA} = C_S^1 + C_S^2$ and $C_T = C_T^1 + C_T^2$. □

Since the above proofs are constructive, architectural implementation of the bit-parallel multipliers is straightforward. We summarize the results in the following theorem:

Theorem 5.4 If the finite field $\mathbb{F}_{2^m}$ is generated with an irreducible trinomial $f(x) = 1 + x^k + x^m$, $1 \leqslant k \leqslant \left\lfloor \frac{m}{2} \right\rfloor$, then a bit-parallel PB multiplier can be constructed with $C_{SA} = m^2$,

(i) $C_{SX} = m^2 - 1$ and $C_T = T_A + (\lceil \log_2 m \rceil + 1) T_X$ for $k = 1$;

(ii) $C_{SX} = m^2 - 1$ and $C_T = T_A + (\lceil \log_2 m \rceil + 2) T_X$, for $1 < k < \frac{m}{2}$;

(iii) $C_{SX} = m^2 - \frac{m}{2}$ and $C_T = T_A + (\lceil \log_2 m \rceil + 1) T_X$ for $k = \frac{m}{2}$.

5.3 Low Complexity PB Squarer in $\mathbb{F}_{2^m}$

5.3.1 Complexity of PB squaring in $\mathbb{F}_{2^m}$

Let $f(x)$ be the irreducible polynomial over $\mathbb{F}_2$ generating the field $\mathbb{F}_{2^m}$. Let $A(x) = \sum_{i=0}^{m-1} a_i x^i$ be the polynomial representation of an arbitrary element of $\mathbb{F}_{2^m}$. The squaring operation of $A(x)$ is $C(x) = A^2(x) \bmod f(x)$, where $0 \leq \deg C(x) \leq m-1$. In $A^2(x)$, the terms with degree equal to or higher than x^m are transformed to lower degree terms using the $(m - \left\lceil \frac{m}{2} \right\rceil - 1)$-by-$m$ reduction matrix $\boldsymbol{R}_s$ as follows [50].

$$\boldsymbol{R}_s \, (1, x, \ldots, x^{m-1})^T = (x^{2\lceil \frac{m}{2} \rceil}, x^{2(\lceil \frac{m}{2} \rceil + 1)}, \ldots, x^{2m-2})^T \bmod f(x)$$

or,

$$\begin{bmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,m-1} \\ q_{1,0} & q_{1,1} & \cdots & q_{1,m-1} \\ \vdots & \vdots & \vdots & \vdots \\ q_{m-\lceil \frac{m}{2} \rceil -1,0} & q_{m-\lceil \frac{m}{2} \rceil -1,1} & \cdots & q_{m-\lceil \frac{m}{2} \rceil -1,m-1} \end{bmatrix} \begin{bmatrix} 1 \\ x \\ \vdots \\ x^{m-1} \end{bmatrix} = \begin{bmatrix} x^{2\lceil \frac{m}{2} \rceil} \\ x^{2(\lceil \frac{m}{2} \rceil +1)} \\ \vdots \\ x^{2m-2} \end{bmatrix} \bmod f(x).$$

Then the square $C(x) \triangleq \sum_{i=0}^{m-1} c_i x^i = A^2(x) \bmod f(x) = a_0 + a_1 x^2 + a_2 x^4 + \ldots + a_{\lceil \frac{m}{2} \rceil} x^{2\lceil \frac{m}{2} \rceil} + \ldots + a_{m-1} x^{2m-2} \bmod f(x)$ can be described as

$$c_i = \begin{cases} a_{\frac{i}{2}} + a_{\lceil \frac{m}{2} \rceil} q_{0,i} + a_{\lceil \frac{m}{2} \rceil +1} q_{1,i} + \cdots + a_{m-1} q_{\lfloor \frac{m}{2} \rfloor -1,i}, & i \text{ even}, \\ a_{\lceil \frac{m}{2} \rceil} q_{0,i} + a_{\lceil \frac{m}{2} \rceil +1} q_{1,i} + \cdots + a_{m-1} q_{\lfloor \frac{m}{2} \rfloor -1,i}, & \text{otherwise}, \end{cases}$$

for $i = 0, 1, 2, \ldots, m-1$.

The complexity of the polynomial basis bit-parallel squaring depends on the number of 1's in matrix $\boldsymbol{R}_s$. The coordinates c_i is the sum of at most $\lfloor \frac{m}{2} \rfloor + 1$ terms which can be realized using a binary tree of at most $\lfloor \frac{m}{2} \rfloor$ XOR gates when i is even. Otherwise, c_i has at most $\lfloor \frac{m}{2} \rfloor$ terms and needs $\lfloor \frac{m}{2} \rfloor - 1$ XOR gates or fewer. The total number of gates is less than $\lfloor \frac{m}{2} \rfloor m - \lfloor \frac{m}{2} \rfloor$, or $\frac{m^2}{2} - \frac{m}{2}$ for m even, and $\frac{m^2}{2} - m + \frac{1}{2}$ for m odd. The maximal time delay is $\lceil \log_2 \left(\lfloor \frac{m}{2} \rfloor + 1 \right) \rceil T_X$. The above can be summarized as follows: Let the field $\mathbb{F}_{2^m}$ be generated with the irreducible polynomial $f(x)$ of degree m. Then squaring a field element can be performed with at most $\lfloor \frac{m}{2} \rfloor m - \lfloor \frac{m}{2} \rfloor$ bit operations.

It can be seen that squaring in $\mathbb{F}_{2^m}$ is actually a case of polynomial modular reduction that has been discussed in §5.1.2, where the degree of each squared terms in $A^2(x)$ is an even integer between 0 and $2m-2$. From the discussion in § 5.1.2 (Theorem 5.1), the following corollary is obvious.

Corollary 5.1 Let the field $\mathbb{F}_{2^m}$ be generated with the irreducible r-term polynomial $f(x)$ of degree m. Then squaring a field element can be performed with at most $(r-1)(m-1)$ addition operations in $\mathbb{F}_2$.

When $f(x)$ is an irreducible trinomial, however, both the size complexity and time complexity can be further reduced.

Theorem 5.5 Let the field $\mathbb{F}_{2^m}$ be generated with the irreducible trinomial $f(x) = x^m + x^k + 1$, where m is even and k odd. Then squaring a field element can be performed with at most $\frac{m+k-1}{2}$ bit operations. □

Proof: Define

$$\sum_{i=0}^{m+2l} a'_i x^i \bmod f(x) \triangleq \sum_{i=0}^{m-1} t_i^{(l)} x^i,$$

where $a'_i \triangleq a_{\frac{i}{2}}$ if i even, and 0 if i odd, and $l = -1, 0, 1, \ldots, \frac{m}{2} - 1$. The terms $t_i^{(l)}$'s have their initial values $t_i^{(-1)} = a'_i$, and we try to solve the final values $t_i^{(\frac{m}{2}-1)} = c_i, i = 0, 1, \ldots, m-1$. In the following we shall prove the theorem by induction.

When $l = 0$,

$$\sum_{i=0}^{m-1} t_i^{(0)} x^i = \sum_{i=0}^{m} a'_i x^i = \sum_{i=0}^{m-1} a'_i x^i + a'_m x^m = \sum_{i=0}^{m-1} a'_i x^i + a'_m(1 + x^k).$$

$$\therefore\ t_i^{(0)} = \begin{cases} a'_i + a'_m, & i = 0; \\ a'_m, & i = k. \\ a'_i, & 0 < i \leqslant m-1, i \text{ even}; \\ 0, & i \text{ odd}, i \neq k; \end{cases}$$

Clearly, one bit addition is needed to compute $t_i^{(0)}$ from $t_i^{(-1)}$, $i = 0, 1, \ldots, m-1$. For $l > 0$, we have

$$\begin{aligned} \sum_{i=0}^{m-1} t_i^{(l)} x^i &= \sum_{i=0}^{m+2l} a'_i x^i \\ &= \sum_{i=0}^{m+2(l-1)} a'_i x^i + a'_{m+2l} x^{m+2l} \\ &= \sum_{i=0}^{m-1} t_i^{(l-1)} x^i + a'_{m+2l} x^{2l}(1 + x^k) \\ &= \sum_{i=0}^{m-1} t_i^{(l-1)} x^i + a'_{m+2l} x^{2l} + a'_{m+2l} x^{k+2l}. \end{aligned}$$

If $k + 2l < m$ or $l < \frac{m-k}{2}$, then

$$t_i^{(l)} = \begin{cases} t_i^{(l-1)}, & 0 \leqslant i \leqslant m-1, i \neq 2l, i \text{ even; or } i = k, k+2, \ldots, k+2(l-1); \\ 0, & i \text{ odd}, i \neq k, k+2, \ldots, k+2l; \\ t_i^{(l-1)} + a'_{m+2l}, & i = 2l; \\ a'_{m+2l}, & i = k+2l. \end{cases} \tag{5.10}$$

It can be seen that only one bit addition is required to compute $t_i^{(l)}$ from $t_i^{(l'-1)}$ for $0 < l < \frac{m-k}{2}$ and $i = 0, 1, \ldots, m-1$.

In the following we proceed with induction. When $l = \frac{m-k+1}{2}$ ($\because m-k$ is odd), we have

$$\begin{aligned}\sum_{i=0}^{m-1} t_i^{(l)} x^i &= \sum_{i=0}^{m-1} t_i^{(l-1)} x^i + a'_{m+2l} x^{m+2l} \\ &= \sum_{i=0}^{m-1} t_i^{(l-1)} x^i + a'_{m+2l} x^{2l} + a'_{m+2l} x^{k+2l} \\ &= \sum_{i=0}^{m-1} t_i^{(l-1)} x^i + a'_{m+2l} x^{2l} + a'_{m+2l} x + a'_{m+2l} x^{k+1}.\end{aligned}$$

Then,

$$t_i^{(l)} = \begin{cases} t_i^{(l-1)} + a'_{m+2l}, & i = 2l \text{ or } i = k+1; \\ a'_{m+2l}, & i = 1; \\ t_i^{(l-1)}, & i \text{ even}, i \neq 2l, k+1; \\ & \text{or } i = k, k+2, \ldots, k+2(l-1); \\ 0, & i \text{ odd}, i \neq k, k+2, \ldots, k+2(l-1) \text{ and } i \neq 1. \end{cases} \tag{5.11}$$

Obviously, two bit additions are required to compute $t_i^{(l)}$ from $t_i^{(l-1)}$, $i = 0, 1, \ldots, m-1$.

Assume that for $\frac{m-k+1}{2} \leqslant l < l'$, (5.11) holds, then for $l = l' < \frac{m}{2}$, we have

$$\begin{aligned}
\sum_{i=0}^{m-1} t_i^{(l')} x^i &= \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + a_{m+2l'} x^{m+2l'} \\
&= \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + a'_{m+2l'} x^{2l'} [1 + x^k] \\
&= \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + a'_{m+2l'} x^{2l'} + + a'_{m+2l'} x^{2l'+k} \\
&= \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + a'_{m+2l'} x^{2l'} + a'_{m+2l'} x^{k+2l'-m} + a'_{m+2l'} x^{2k+2l'-m} \\
&= \sum_{i=0}^{m-1} t_i^{(l',0)} x^i + a'_{m+2l'} x^{2k+2l'-m}, \qquad (5.12)
\end{aligned}$$

where $t_i^{(l',0)}$ is defined by $\sum_{i=0}^{m-1} t_i^{(l',0)} x^i \triangleq \sum_{i=0}^{m-1} t_i^{(l'-1)} x^i + a'_{m+2l'} x^{2l'} + a'_{m+2l'} x^{k+2l'-m}$. Since $2l' < m$, and $k + 2l' - m$ is odd and less than k,

$$t_i^{(l',0)} = \begin{cases} t_i^{(l'-1)} + a'_{m+2l'}, & i = 2l'; \\ a'_{m+2l'}, & i = k + 2l' - m; \\ t_i^{(l'-1)}, & 0 \leqslant i \leqslant m-1, i \neq 2l', i \text{ even; or } i = k, k+2, \dots, k+2(l'-1); \\ 0, & i \text{ odd}, i \neq k, k+2, \dots, k+2(l'-1). \end{cases} \qquad (5.13)$$

Thus it requires one bit addition to obtain $t_i^{(l',0)}$ from $t_i^{(l'-1)}$, $i = 0, 1, \dots, m-1$.

When $2k + 2l' - m < m$, we have $t_i^{(l')} = t_i^{(l',0)}$ if $i \neq 2k + 2l' - m$, otherwise $t_i^{(l')} = t_i^{(l',0)} + a'_{m+2l'}$. It is therefore two bit operations are required to compute $t_i^{(l')}$ from $t_i^{(l'-1)}$ for $i = 0, \dots, m-1$.

When $2k + 2l' - m \geqslant m$, consider

$$\sum_{i=0}^{m-1} t_i^{(k+l'-m)} = \sum_{i=0}^{m-1} t_i^{(k+l'-m-1)} + a'_{2k+2l'-m} x^{2k+2l'-m}. \tag{5.14}$$

It can be seen that the last terms of the right hand side of (5.12) and (5.14) are the same except for the coefficients. If we perform

$$\sum_{i=0}^{m-1} t_i^{(k+l'-m,*)} = \sum_{i=0}^{m-1} t_i^{(k+l'-m-1)} + (a'_{2k+2l'-m} + a'_{m+2l'}) x^{2k+2l'-m} \tag{5.15}$$

at step $l = k + l' - m$ at the cost of one more bit operation, then at step $l = l'$, the term $t_i^{(l')}$ can be computed from $T_i^{(l'-1)}, i = 0, \ldots, m-1$ with only one bit operation. Equivalently, we might say that at step $l = l'$, term $t_i^{(l')}$ can be computed from $t_i^{(l'-1)}$, $i = 0, 1, \ldots, m-1$, at the cost of two bit operations. Thus for $\frac{m-k+1}{2} \leqslant l < \frac{m}{2} - 1$, it requires two bit additions at each step.

We conclude that the total cost for computing $c_i = t_i^{(\frac{m}{2}-1)}$ from $t_i^{(-1)} = a'_i, i = 0, 1, \ldots, m-1$ is $\frac{m-k+1}{2} + 2(\frac{m}{2} - 1 - \frac{m-k-1}{2}) = \frac{m+k-1}{2}$ bit operations. □

Theorem 5.6 Let the field $\mathbb{F}_{2^m}$ be generated with the irreducible trinomial $f(x) = x^m + x^k + 1$, where m is odd and k even. Then squaring in $\mathbb{F}_{2^m}$ can be performed with at most $\frac{m+k-1}{2}$ bit additions.

Theorem 5.7 Let the field $\mathbb{F}_{2^m}$ be generated with the irreducible trinomial $f(x) = x^m + x^k + 1$, where both m and k are odd. Then squaring in $\mathbb{F}_{2^m}$ can be performed with at most $\frac{m-1}{2}$ bit additions. □

Proofs of Theorems 5.6 and 5.7 are similar to that of Theorem 5.5.

5.3.2 Implementation

Theorem 5.8 Let the field $\mathbb{F}_{2^m}$ be generated with the irreducible trinomial $f(x) = x^m + x^k + 1$, where $m + k$ is odd. Then a bit-parallel squarer can be implemented with at most $\frac{m+k-1}{2}$ XOR gates. For $k = 1$ and 2, the incurred time delay is T_X, and for $2 < k \leqslant \frac{m}{2}$, it is $2T_X$.

Proof: For a certain value of i, $i = 0, 1, \ldots, m-1$, it can be seen from (5.10) and (5.13) that that every bit addition occurs for a different value of i. Thus the time delay incurred with (5.10) and (5.13) is at most T_X. The longest time delay is incurred with (5.12). A simple method to estimate this delay is to count the number of times that (5.15) is used when computing (5.12). When $l' = \frac{m}{2} - 1$, $x^{2k+2l'-m} = x^{2k-2}$. Applying the mapping $x^m = x^k + 1$ to x^{2k-2} until all the terms with degree less than m. Let the number of the mappings is y, then

$$2k - 2 - y(m-k) < m \Rightarrow y > \frac{2k-m-2}{m-k}.$$

$\therefore$ The longest time delay $= \left\lceil \frac{2k-m-2}{m-k} \right\rceil + 2 = \left\lceil \frac{m-2}{m-k} \right\rceil$.

For $k = 1, 2$, we have $\left\lceil \frac{m-2}{m-k} \right\rceil = 1$, and when $2 < k \leqslant \frac{m}{2}$, $\left\lceil \frac{m-2}{m-k} \right\rceil = 2$. □

Theorem 5.9 Let the field $\mathbb{F}_{2^m}$ be generated with the irreducible trinomial $f(x) = x^m + x^k + 1$, where both m and k are odd. Then a bit-parallel squarer can be implemented with at most $\frac{m-1}{2}$ XOR gates. The incurred time delay is T_X if $k = 1$, and $2T_X$ if $2 < k < \frac{m}{2}$.

Proof: The proof is similar to that of Theorem 5.8.

Chapter 6

Analysis of SD Form Exponents

The primary operation in most discrete logarithm and elliptic curve public-key cryptosystems is to raise an element in the group to large powers, *i.e.*, exponentiation and point multiplication on an elliptic curve. This chapter deals with the efficient representations of the exponent. First a brief review of signed-digit (SD) number systems (§6.1), then closed form formula for the number of nonzeros and the length of the (SD) non-adjacent form (NAF) are derived in § 6.2.

6.1 Exponent Representations

6.1.1 Using conventional number systems

The conventional number systems are non-redundant and fixed-radix number systems. In a non-redundant number system, every number has a unique representation. An integer N is uniquely represented with a radix-r number system as $N = \sum_{i=0}^{n-1} a_i r^i$, where $a_i \in$

$\{0, 1, \ldots, r-1\}$. The ordered string $(a_{n-1}, \ldots, a_0)$ is called the radix-r representation of N. A binary or radix-m representation of N naturally introduces a method to compute α^N (*square and multiply* method or *m-ary* method [57]).

6.1.2 Using redundant number systems

In a fixed-radix system, if we allow the digit set to be extended to $\{\overline{a}, \ldots, \overline{1}, 0, 1, \ldots, a\}$,[a] where $\overline{a} = -a$. then the resultant number system is called the ***signed-digit (SD) number system***. A radix-r SD representation of N is given by:

$$N = (b_m b_{m-1} \ldots b_0)$$

where $b_i \in \{0, \pm 1, \ldots, \pm(r-1)\}$ and $\sum_{i=0}^{m} b_i r^i = N$.

The SD number system is redundant since some numbers have more than one representation. Note that the number of nonzeros digits of an SD form is actually the length of the corresponding addition/subtraction chain [59], then we can present the following definition:

Definition 6.1 [67, 18, 6] $N = (b_m b_{m-1} \ldots b_0)$ is referred to as a *minimal weight radix-r SD representation* of N if the sum $\sum_{i=0}^{m} x_i$ is minimized, where

$$x_i \triangleq \begin{cases} 1 & \text{if } b_i \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

[a]The positive integer a can be in the range of $\lceil \frac{r-1}{2} \rceil \leqslant a \leqslant r-1$. When $a = \lceil \frac{r-1}{2} \rceil$, the set of $\{\overline{a}, \ldots, \overline{1}, 0, 1, \ldots, a\}$ is the *minimal signed digit set* [51] and the system is called the *minimally redundant signed-digit number system*.

It can be shown that a minimal weight radix-r SD representation is of length not greater than $n+1$, where n is the binary length of N.

Among the minimal weight radix-r SD representations of N, a canonical radix-r SD form can be defined as follows which is a generization of the canonical binary SD (BSD) form [67].

Definition 6.2 [18] Let $N = (b_n b_{n-1} \ldots b_0)$ be a minimal weight radix-r SD representation. It is referred to as a *canonical radix-r SD representation* of N if b_i's satisfy the following two conditions:

$$\begin{array}{lll} \text{(i)} & |b_{i+1} + b_i| < r & \text{for all } i, \\ \text{(ii)} & |b_i| < |b_{i+1}| & \text{if } b_{i+1} b_i < 0. \end{array}$$

where $|b|$ denotes the absolute value of b.

In Clark and Liang's paper [18], where the canonical SD representation is referred to as a *generalized nonadjacent form (GNAF)*, since it has the property that there is no two consecutive nonzero digits in this representation when $r = 2$, proofs have been provided for minimality, uniqueness and existence of the canonical SD representation for any integer N. An algorithm from Theorem 3 of [18] that converts the conventional radix-r representation of an integer into the canonical radix-r SD form is given below.

Algorithm 6.1 [18]

Input: The radix-r form of an integer: $N = (a_{n-1} a_{n-2} \cdots a_0)$,
where $a_i \in \{0, 1, \ldots, r-1\}$.

Output: The canonical radix-r SD form: $N = (b_n b_{n-1} \ldots b_0)$,
where $b_i \in \{0, \pm 1, \ldots, \pm(r-1)\}$.

Step 1: $a_n := 0; a_{n+1} := 0; c_0 := 0;$

Step 2: for $i = 0$ to n do

if $a_i + a_{i+1} + c_i \geqslant r$

$b_i := a_i + c_i - r;$

$c_{i+1} := 1;$

else

$b_i := a_i + c_i;$

$c_{i+1} := 0;$

End.

6.2 Average Hamming Weight and Length of Non-adjacent Form

Minimal weight signed-digit (SD) representations have been used in many arithmetic operations, such as, computation of exponentiations in the integer domain and in a cyclic group [17, 80], computation of multiple of a point on an elliptic curve [59], *etc.* In these algorithms, the number of certain basic arithmetic operations, for example, multiplication and squaring, depends on both the *Hamming weight* and the *length* of the SD representations used. It is thus important to know their precise formulae to determine the number of underlying basic arithmetic operations.

6.2.1 Hamming weight of radix-r NAF

Lemma 6.1 Let $\rho(n, r)$ denote the average number of nonzero digits in the minimal weight radix-r SD representation of an integer which is between 0 and $r^n - 1$, inclusive.

Assuming $\rho(0,r)=0$, we have ($n \geqslant 1$)

$$\rho(n,r) = \begin{cases} \rho(n-1,r) + \dfrac{r-1}{r+1}(1+\dfrac{1}{r^{n-1}}) & \text{for } n \text{ even,} \\ \rho(n-1,r) + \dfrac{r-1}{r+1}(1+\dfrac{1}{r^{n}}) & \text{for } n \text{ odd.} \end{cases}$$

Proof:

(i) When $n=1$, obviously, $\rho(1,r)=\frac{r-1}{r}$.

(ii) When $n=2$, it can be verified that the number of nonzero digits cannot be reduced in the minimal weight SD form, and thus $\rho(2,r)$ is equal to the average number of nonzero digits in the radix-r form of an integer N, $0 \leqslant N \leqslant r^2-1$. Then $\rho(2,r)=\dfrac{2(r-1)}{r}=\rho(1,r)+\frac{r-1}{r}$.

(iii) When $n \geqslant 3$, let $N=(a_{n-1}a_{n-2}\dots a_0)$ be an integer in radix-r form between 0 and r^n-1, inclusive. We have

$$\begin{aligned} \rho(n,r) &= \Pr(a_{n-1}=0)\rho(n-1,r) + \Pr(a_{n-1}\neq 0)\sigma(n,r) \\ &= \frac{1}{r}\rho(n-1,r) + \frac{r-1}{r}\sigma(n,r), \end{aligned} \tag{6.2}$$

where $\sigma(n,r)$ denotes the average number of nonzero digits in the minimal weight radix-r SD form of an integer whose radix-r representation is of length n with the most significant digit being nonzero.

Let each of $u_j, v_j, \tilde{v}_j$ and w_j, $j=0,1,2,\dots$, denote a digit and be given by

$$\begin{aligned} u_j &\in \{1,2,\dots,r-1\}, \\ v_j &\in \{0,1,\dots,r-2\}, \\ \tilde{v}_j &\triangleq r-1. \\ w_j &\in \{0,1,\dots,r-1\}. \end{aligned}$$

Let $N' = (a_{n-2}a_{n-3}\dots a_0)$, and let $N'' = u_0 \cdot r^{n-1} + N' = (u_0a_{n-2}a_{n-3}\dots a_0)$. Then all the possible combinations of the three most significant digits in the radix-r form of N'' can be divided into three cases A_0, B_0, and C_0:

$$\begin{cases} (A_0) & u_0|v_1w_2\cdots \\ (B_0) & u_0|\tilde{v}_1 0\cdots \to \begin{cases} (A_1) & u_0|\tilde{v}_1 0 v_3 w_4\cdots \\ (B_1) & u_0|\tilde{v}_1 0\tilde{v}_3 0\cdots \to \begin{cases} (A_2) & u_0|\tilde{v}_1 0 \tilde{v}_3 0 v_5 w_6\cdots \\ (B_2) & u_0|\tilde{v}_1 0\tilde{v}_3 0 \tilde{v}_5 0\cdots \to \cdots \\ (C_2) & u_0|\tilde{v}_1 0\tilde{v}_3 0\tilde{v}_5 u_6\cdots \end{cases} \\ (C_1) & u_0|\tilde{v}_1 0 \tilde{v}_3 u_4\cdots \end{cases} \\ (C_0) & u_0|\tilde{v}_1 u_2\cdots \end{cases}$$

Now apply Algorithm 6.1 to both N' and N'' and let the resultant canonical SD forms be $b'_{n-1}\cdots b'_0$ and $b''_n\cdots b''_0$, respectively. Obviously, $b'_i = b''_i$ for $i = 0, 1, \dots, n-3$ and $c'_{n-2} = c''_{n-2}$.

When the case A_0 is considered, one clearly has $b'_{n-2} = v_1 + c'_{n-2}$ and $b'_{n-1} = 0$. If $u_0 + v_1 + c''_{n-2} \geqslant r$ then $b'_{n-2} = v_1 + c'_{n-2} \neq 0$ and $b''_{n-2} = v_1 + c''_{n-2} - r \neq 0$ and, $b''_{n-1} = u_0 + 1 \neq 0$ and $b''_n = 0$ if $u_0 < r-1$, or $b''_{n-1} = 0$ and $b''_n = 1$ if $u_0 = r-1$. If $u_0 + v_1 + c''_{n-2} < r$ then $b''_{n-2} = v_1 + c''_{n-2} = b'_{n-2}$ and $b''_{n-1} = u_0 \neq 0$. Thus, in A_0, the SD form of N'' has one more nonzero digit than that of N'.

When the case C_0 is considered, then $c'_{n-2} = c''_{n-2} = 1$ since $\tilde{v}_1 + u_1 \geqslant r$. Also $b'_{n-2} = 0$, $b'_{n-1} = 1$, and $b''_{n-2} = 0$ and, $b''_{n-1} = u_0 + 1 \neq 0$ if $u \leqslant r-2$, or $b''_{n-1} = 0$ and $b''_n = 1$ if $u_0 = r-1$. Thus, in the case of C_0 the SD forms of N' and of N'' have the same number of nonzero digits.

In case B_0, it can be easily checked that the SD form of N'' has as many nonzero digits as that of N' if $c'_{n-2} = c''_{n-2} = 1$, and the SD form of N'' has one more nonzero digit than that of N' if $c'_{n-2} = c''_{n-2} = 0$. Further including next two digits so that it can be divided into three subcases A_1, B_1, and C_1, then in subcase A_1 we have $c'_{n-2} = c''_{n-2} = 0$,

and in subcase C_1 we have $c'_{n-2} = c''_{n-2} = 1$ because $c'_{n-3} = c''_{n-3} = 1$. In subcase B_1, the above dividing procedure is repeated. It can be verified that in all subcases $A_j, j = 0, 1, 2, \ldots$, we have $c'_{n-2} = c''_{n-2} = 0$, and in all subcases $C_j, j = 0, 1, 2, \ldots$, we have $c'_{n-2} = c''_{n-2} = 1$. If n is odd, the repeated procedure stops when it reaches subcases $A_{\frac{n-3}{2}}, B_{\frac{n-3}{2}}$, and $C_{\frac{n-3}{2}}$. Obviously, $c'_{n-2} = c''_{n-2} = 0$ in subcase $B_{\frac{n-3}{2}}$. If n is even, the final subcases to consider are subcases $A_{\frac{n-4}{2}}, B_{\frac{n-4}{2}}$, and $C_{\frac{n-4}{2}}$. In subcase $B_{\frac{n-4}{2}}$ further including the last remaining digit a_0, it is easy to see that $c'_{n-2} = c''_{n-2} = 0$ when a_0 either is 0 or belongs to $\{1, 2, \ldots, r-1\}$.

Following the above argument and noting that

$$\begin{aligned}
\Pr(A_0) = \Pr(A_1|B_0) = \Pr(A_2|B_1) = \cdots = \frac{r-1}{r}, \\
\Pr(B_0) = \Pr(B_1|B_0) = \Pr(B_2|B_1) = \cdots = \frac{1}{r^2}, \\
\Pr(C_0) = \Pr(C_1|B_0) = \Pr(C_2|B_1) = \cdots = \frac{r-1}{r^2},
\end{aligned}$$

when n is odd one can write

$$\begin{aligned}
\sigma(n,r) &= \rho(n-1,r) + \Pr(A_0) \times 1 + \Pr(B_0) \times \Big(\Pr(A_1|B_0) \times 1 + \Pr(B_1|B_0) \times \\
&\quad \Big(\Pr(A_2|B_1) \times 1 + \Pr(B_2|B_1) \times \big(\cdots\big)\Big)\Big) \\
&= \rho(n-1,r) + \frac{r-1}{r}\Big(1 + \frac{1}{r^2} + \frac{1}{(r^2)^2} + \cdots + \frac{1}{(r^2)^{\frac{n-3}{2}}}\Big) + \frac{1}{(r^2)^{\frac{n-1}{2}}} \\
&= \rho(n-1,r) + \frac{r-1}{r}\Big(\frac{\frac{1}{r^{n-1}} - 1}{\frac{1}{r^2} - 1}\Big) + \frac{1}{r^{n-1}} \\
&= \rho(n-1,r) + r(r-1)\frac{1 - \frac{1}{r^{n-1}}}{r^2 - 1} + \frac{1}{r^{n-1}} \\
&= \rho(n-1,r) + \frac{r}{r+1}\Big(1 - \frac{1}{r^{n-1}} + \frac{r+1}{r^n}\Big) \\
&= \rho(n-1,r) + \frac{r}{r+1}\Big(1 + \frac{1}{r^n}\Big). \qquad (6.3)
\end{aligned}$$

Then by noting (6.2), we obtain

$$\rho(n,r) \;=\; \rho(n-1,r) + \frac{r-1}{r+1}\Big(1 + \frac{1}{r^n}\Big).$$

When n is even, we have

$$\begin{aligned}
\sigma(n,r) \;&=\; \rho(n-1,r) + \Pr(A_0) \times 1 + \Pr(B_0) \times \Big(\Pr(A_1|B_0) \times 1 + \Pr(B_1|B_0) \times \\
&\qquad \Big(\Pr(A_2|B_1) \times 1 + \Pr(B_2|B_1) \times \big(\cdots\big)\Big)\Big) \\
&=\; \rho(n-1,r) + \frac{r-1}{r}\Big(1 + \frac{1}{r^2} + \frac{1}{(r^2)^2} + \cdots + \frac{1}{(r^2)^{\frac{n-4}{2}}}\Big) + \frac{1}{(r^2)^{\frac{n-2}{2}}} \\
&=\; \rho(n-1,r) + \frac{r-1}{r}\Big(\frac{\frac{1}{r^{n-2}} - 1}{\frac{1}{r^2} - 1}\Big) + \frac{1}{r^{n-2}} \\
&=\; \rho(n-1,r) + r(r-1)\frac{1 - \frac{1}{r^{n-2}}}{r^2 - 1} + \frac{1}{r^{n-2}} \\
&=\; \rho(n-1,r) + \frac{r}{r+1}\Big(1 - \frac{1}{r^{n-2}} + \frac{r+1}{r^{n-1}}\Big) \\
&=\; \rho(n-1,r) + \frac{r}{r+1}\Big(1 + \frac{1}{r^{n-1}}\Big). \qquad (6.4)
\end{aligned}$$

From (6.2) and (6.4), we obtain

$$\rho(n,r) \;=\; \rho(n-1,r) + \frac{r-1}{r+1}\Big(1 + \frac{1}{r^{n-1}}\Big).$$

□

From the recursive relation in Lemma 6.1, we can derive a closed form expression for $\rho(n,r)$. When n is odd, by recalling that $\rho(1,r) = \frac{r-1}{r}$ and using Lemma 6.1 we

can write

$$\begin{aligned}
\rho(n,r) &= \rho(n-1,r) + \frac{r-1}{r+1} + \frac{r-1}{(r+1)r^n} \\
&= \rho(n-2) + \frac{r-1}{r+1} + \frac{r-1}{(r+1)r^n} + \frac{r-1}{r+1} + \frac{r-1}{(r+1)r^{n-2}} \\
&\vdots \\
&= \rho(1,r) + \frac{(n-1)(r-1)}{r+1} \\
&\quad + \frac{r-1}{r+1}\left(\frac{1}{r^n} + \frac{1}{r^{n-2}} + \frac{1}{r^{n-2}} + \frac{1}{r^{n-4}} + \frac{1}{r^{n-4}} + \cdots + \frac{1}{r^3} + \frac{1}{r^3} + \frac{1}{r}\right) \\
&= \frac{r-1}{r} + \frac{(n-1)(r-1)}{r+1} + \frac{r-1}{r+1}\left(\frac{1}{r} + \frac{1}{r^3}\right)\frac{1-\frac{1}{r^{n-1}}}{1-\frac{1}{r^2}} \\
&= \frac{r-1}{r+1}n + \frac{r-1}{(r+1)r} + \frac{(r^2+1)(r^{n-1}-1)}{(r+1)^2 r^n} \\
&= \frac{r-1}{r+1}n + \frac{2r}{(r+1)^2} - \frac{r^2+1}{(r+1)^2 r^n}.
\end{aligned}$$

When n is even, it yields

$$\rho(n,r) = \frac{r-1}{r+1}n + \frac{2r}{(r+1)^2} - \frac{2}{(r+1)^2 r^{n-1}}.$$

We summarize the above results in the following theorem.

Theorem 6.1 Let $\rho(n,r)$ denote the average number of nonzero digits in the minimal weight radix-r SD representation of an integer whose radix-r form is of length n (between 0 and $r^n - 1$, inclusive, and $n \geqslant 1$). Then a closed form expression on $\rho(n,r)$ is given

by

$$\rho(n,r) = \begin{cases} \dfrac{r-1}{r+1}n + \dfrac{2r}{(r+1)^2} - \dfrac{2}{(r+1)^2 r^{n-1}} & \text{for } n \text{ even,} \\ \\ \dfrac{r-1}{r+1}n + \dfrac{2r}{(r+1)^2} - \dfrac{r^2+1}{(r+1)^2 r^n} & \text{for } n \text{ odd.} \end{cases}$$

For the case of the minimal weight BSD form, we have the following expression for the average number of nonzero digits in the minimal weight BSD form of an integer whose radix-r form is of length n ($n \geqslant 0$):

$$\rho(n,2) = \begin{cases} \dfrac{n}{3} + \dfrac{4}{9}(1 - \dfrac{1}{2^n}) & \text{for } n \text{ even,} \\ \\ \dfrac{n}{3} + \dfrac{1}{9}(4 - \dfrac{5}{2^n}) & \text{for } n \text{ odd.} \end{cases}$$

6.2.2 Length of radix-r NAF

We have also obtained the average length of the canonical SD form and it is summarized in the following theorem.

Theorem 6.2 Let $\lambda(n,r)$ denote the average length of the canonical radix-r SD representation of an integer whose radix-r form is of length n ($n \geqslant 1$) with the most significant digit being nonzero. Then $\lambda(n,r)$ can be given by the following expression,

$$\lambda(n,r) = \begin{cases} n + \dfrac{r}{r^2-1}\left(1 - \dfrac{1}{r^n}\right) & \text{for } n \text{ even,} \\ \\ n + \dfrac{r}{r^2-1}\left(1 - \dfrac{1}{r^{n-1}}\right) & \text{for } n \text{ odd.} \end{cases}$$

□

Proof:

(i) When $n = 1, \lambda(1, r) = 1$.

(ii) When $n = 2$, the SD form has length 3 only when $a_1 = r - 1$ and $a_0 \neq 0$, otherwise the SD form has length 2. Thus, $\lambda(2, r) = 2 + \frac{r-1}{r(r-1)} = 2 + \frac{1}{r}$.

(iii) When $n \geqslant 3$, it is easy to see that $c_n = 1$ is a necessary and sufficient condition for that the canonical SD form is longer than its radix-r form. If $c_n = 0$, the canonical SD form keeps the same length as its radix-r form.

Let $u_j, v_j, \bar{v}_j$, and $w_j, j = 0, 1, \ldots$, be defined as in the proof of Lemma 5.1 and $x_j, j = 0, 1, \ldots$, be a digit $\in \{1, 2, \ldots, r-2\}$. Let T be an integer between r^{n-1} and $r^n - 1$, inclusive. Consider all the possible combinations of the leading three digits (the most significant digits) in the radix-r form of T (shown below) and apply Algorithm 8.2 to them:

$$
\begin{cases}
(A_0) & x_0 w_1 w_2 \cdots \\
(B_0) & \bar{v}_0 0 v_2 \cdots \\
(C_0) & \bar{v}_0 0 \bar{v}_2 \cdots \rightarrow \begin{cases}
(B_1) & \bar{v}_0 0 \bar{v}_2 0 v_4 \cdots \\
(C_1) & \bar{v}_0 0 \bar{v}_2 0 \bar{v}_4 \cdots \rightarrow \begin{cases}
(B_2) & \bar{v}_0 0 \bar{v}_2 0 \bar{v}_4 0 v_6 \cdots \\
(C_2) & \bar{v}_0 0 \bar{v}_2 0 \bar{v}_4 0 \bar{v}_6 \cdots \rightarrow \cdots \\
(D_2) & \bar{v}_0 0 \bar{v}_2 0 \bar{v}_4 u_5 w_6 \cdots
\end{cases} \\
(D_1) & \bar{v}_0 0 \bar{v}_2 u_3 w_4 \cdots
\end{cases} \\
(D_0) & \bar{v}_0 u_1 w_2 \cdots
\end{cases}
$$

It can be seen that $c_n = 0$ in cases A_0 and B_0, and $c_n = 1$ in case D_0. In case C_0, include the next two digits and consider all possible combinations of the leading five digits, it is easy to see that $c_n = 0$ in case B_1, and in case D_1 we have $c_n = 1$ ($\because c_{n-2} = 1$ and $a_{n-1} + c_{n-2} = \bar{v}_0 + c_{n-2} \geqslant r, \therefore c_{n-1} = 1$ and $a_{n-1} + c_{n-1} \geqslant r$). For C_1, we can further divide it into three sub-cases B_2, C_2, and D_2 and a similar discussion can be applied.

Obviously we have

$$
\begin{aligned}
\Pr(A_0) &= \frac{r-2}{r-1}, \\
\Pr(B_0) &= \frac{1}{r-1} \cdot \frac{1}{r} \cdot \frac{r-1}{r} = \frac{1}{r^2}, \\
\Pr(C_0) &= \frac{1}{r-1} \cdot \frac{1}{r} \cdot \frac{1}{r} = \frac{1}{r^2(r-1)}, \\
\Pr(D_0) &= \frac{1}{r-1} \cdot \frac{r-1}{r} = \frac{1}{r},
\end{aligned}
$$

and

$$
\begin{aligned}
\Pr(B_1|C_0) = \Pr(B_2|C_1) = \Pr(B_3|C_2) &= \cdots = \frac{r-1}{r^2}, \\
\Pr(C_1|C_0) = \Pr(C_2|C_1) = \Pr(C_3|C_2) &= \cdots = \frac{1}{r^2}, \\
\Pr(D_1|C_0) = \Pr(D_2|C_1) = \Pr(D_3|C_2) &= \cdots = \frac{r-1}{r}.
\end{aligned}
$$

When n is even, the procedure would continue until the least significant digit is reached, where the last digit is a 0 with a probability of $\frac{1}{r}$ and a nonzero with a probability of $\frac{r-1}{r}$, corresponding to $c_n = 0$ and $c_n = 1$, respectively. Then,

$$
\begin{aligned}
\lambda(n,r) &= n + \Pr(D_0) + \Pr(C_0)\Big(\Pr(D_1|C_0) + \Pr(C_1|C_0)\Big(\Pr(D_2|C_1) + \cdots\Big)\Big) \\
&= n + \frac{1}{r} + \frac{1}{r^3} + \frac{1}{r^5} + \cdots + \frac{1}{r^{n-1}} \\
&= n + \frac{r}{r^2-1}\Big(1 - \frac{1}{r^n}\Big).
\end{aligned}
$$

When n is odd, this procedure can be repeated to the end of the original radix-r form and the length of the canonical SD form is,

$$\begin{aligned}\lambda(n,r) &= n + \Pr(D_0) + \Pr(C_0)\Big(\Pr(D_1|C_0) + \Pr(C_1|C_0)\Big(\Pr(D_2|C_1) + \cdots\Big)\Big) \\ &= n + \frac{1}{r} + \frac{1}{r^3} + \frac{1}{r^5} + \cdots + \frac{1}{r^{n-2}} \\ &= n + \frac{r}{r^2-1}\Big(1 - \frac{1}{r^{n-1}}\Big).\end{aligned}$$

□

When the radix $r = 2$, we have the following formula for the average length of a canonical BSD form whose binary expansion is of length n ($n \geqslant 0$):

$$\lambda(n,2) = \begin{cases} n + \frac{2}{3}\Big(1 - \frac{1}{2^n}\Big) & \text{for } n \text{ even,} \\ n + \frac{2}{3}\Big(1 - \frac{1}{2^{n-1}}\Big) & \text{for } n \text{ odd.} \end{cases}$$

Chapter 7

Realization of Finite Field Exponentiation

In this chapter, we present efficient architectures for exponentiation of a primitive element of a finite field. We consider two different representations of the element – one using the polynomial basis and the other using its weakly dual basis. Parts of this chapter have been presented in [80, 79].

7.1 Brief Review

Many cryptosystems require extensive exponentiation in finite fields [20, 3]. High-speed computation of this function in large finite fields, which are necessary to achieve a high level of security, requires hardware implementation. A special case of exponentiation in finite fields is that the base α is a root of the primitive polynomial $F(x)$ generating the field $\mathbb{F}_{2^m}$ [69, 50]. Given a primitive element $\alpha \in \mathbb{F}_{2^m}$ and an integer $H = \sum_{i=0}^{m-1} h_i 2^i, 0 \leq H \leq 2^m - 2$, the exponentiation function of α is given as $y = \alpha^H \in \mathbb{F}_{2^m}^* = \mathbb{F}_{2^m} - \{0\}$.

The basic scheme for computing the exponentiation function α^H in $\mathbb{F}_{2^m}$ is the *Square and Multiply* algorithm. The number of multiplications involved in this algorithm is determined by the number of 1's in the binary representation of the exponent H.

When a polynomial basis or its weakly dual basis is used to represent the field elements, multiplication with α is simple and squaring can be implemented in a bit-parallel fashion [50]. If a normal basis is adopted, the squaring is trivial; the multiplication is however quite complicated. The multiplication, however, can be simplified when an optimal normal basis [7] is used. Using the square and multiply algorithm, architectures for exponentiation of a primitive root with polynomial basis and normal basis can be found in [69, 50]. Since α is a fixed element, one method to compute α^H is to precompute the conjugates of α and store them in a memory, and then multiply together some of the conjugates according to the binary representation of H [69]. The multiplications can be performed in a parallel fashion with a number of multipliers arranged in a binary tree form. Processor-time tradeoffs can be made by adopting a subset of the full multiplier tree.

Other methods for exponentiation of a primitive root are based on lookup tables (LUTs) [50] and linear feedback shift registers (LFSRs) [50]. The former is advantageous only for small m, since the size of the LUT is proportional to $m \times 2^m$. The LFSR based method requires H multiplications to compute $\alpha^H = \underbrace{\alpha\alpha\ldots\alpha}_{H}$. It is suitable for small values of m and under the condition that the computation time is not critical [50]. It is worth mentioning here that the binary representation of H is used in all the above algorithms.

On the other hand, binary signed digit (SD) representation of H whose symbols belong to $\{-1, 0, 1\}$ has been used in exponentiation b^H when the base b is a conventional real number [17]. In the binary SD number system, H may have several representations; however, the minimal binary SD representation, which has the least number of nonzero symbols, can reduce the average number of nonzero symbols to $\frac{m}{3}$ from $\frac{m}{2}$ contained in the conventional binary representation [28]. . Consequently, the use of the minimal SD

representation results in fewer multiplications required for the computation of exponentiations in the real number field.

However, introducing SD number system, especially with a higher radix, into exponentiation in $\mathbb{F}_{2^m}$ would involve a multiplicative inversion operation, which is commonly known to be difficult. To solve this problem, in this chapter, we present novel architectures as well as new algorithms for exponentiation of a primitive root in $\mathbb{F}_{2^m}$. Using the minimal binary SD representation and bidirectional LFSR, an architecture for the exponentiation is developed. This architecture has a low size complexity and can effectively reduce the number of underlying operations. Consequently, it is suitable for low power implementation using VLSI technologies. Furthermore, the use of the minimal radix-4 SD representation of the exponent is investigated and an extended bidirectional LFSR is devised to perform multiple operations that arise due to the use of the radix-4 representation. The attractive feature of the extended bidirectional LFSR is that a multiplication with a primitive root, or its square or its inverse or its inverse-and-square can be performed with one single shift operation. Using this LFSR, a second architecture for exponentiation in $\mathbb{F}_{2^m}$ is developed. With a modest extra size complexity, this architecture has the potential of significantly reducing the total computation time as well as power consumption, when implemented using VLSI technologies. As a result, the proposed exponentiation architectures are suitable when low power and compact configurations are of prime concern, such as personal communication systems.

7.2 Efficient Representations of Exponent

7.2.1 Algorithm

One special case of the redundant SD number representations (discussed in §6.1.1) is $r = 4$, and $a = 2$, which uses radix 4 and the minimal signed digit set $\{-2, -1, 0, 1, 2\}$. This class of SD representations is well known and has been applied to the design of computers

[8]. In this section, we will give an explicit definition of the reduced redundancy minimal radix-4 SD form and an algorithm to generate the canonical representation, as well as some properties.

Definition 7.1 The reduced redundancy radix-4 SD number $K = k_{n-1}k_{n-1}...k_0$, $k_i \in \{\bar{2}, \bar{1}, 0, 1, 2\}$, is a *minimal radix-4 SD representation*, if $\sum_{i=0}^{n-1} s_i$ is minimized, where $\bar{x} \triangleq -x$ and

$$s_i = \begin{cases} 1 & \text{for } k_i \neq 0 \\ 0 & \text{otherwise} \end{cases}.$$

A method called the *extended canonical recoding* to obtain the radix-4 SD representation from a binary number $H = h_{m-1}h_{m-2}\dots h_0$ is given below.

Algorithm 7.1

Step 1. Use the canonical recoding to obtain the canonical binary SD representation of H [67]:

$$\{h_{m-1}h_{m-2}\dots h_0\} \rightarrow \{g_m g_{m-1}\dots g_0\};$$

Step 2. Compute $k_i = g_{2i} + 2g_{2i+1}$ for $i = 0, 1, 2, \dots, \left\lfloor \frac{m}{2} \right\rfloor$ ($g_{m+1} = 0$ for m even).

The radix-4 SD form $H = k_{\lfloor \frac{m}{2} \rfloor}k_{\lfloor \frac{m}{2} \rfloor - 1}\dots k_0$ computed from Algorithm 7.1 is called the *(reduced redundancy) canonical radix-4 SD* representation. Circuits to transform a binary number to its canonical radix-4 SD representation using Algorithm 7.1 are shown in Fig. 7.6. An example of using the extended canonical recoding to obtain the canonical radix-4 SD form is given in Example 7.1 in the next subsection.

7.2.2 Features of minimal radix-4 SD form

Lemma 7.2 Every integer has a unique canonical radix-4 SD representation.

Proof: $\{g_m g_{m-1} \ldots g_0\}$ computed in Step 1 is a canonical binary SD number and has the property: $g_{j+1}g_j = 0$, for $j = 0, 1, \ldots, m-1$ [67]. Then the possible values of $g_{2i+1}g_{2i}$, in Step 2, are $00, 01, 10, 0\bar{1}$, and $\bar{1}0$, and correspondingly, $k_i = 0, 1, 2, \bar{1}$, and $\bar{2}$. It is obvious that there is a one-to-one correspondence between $g_{2i}g_{2i+1}$ and k_i. Thus, $\{k_{\lfloor \frac{m}{2} \rfloor} k_{\lfloor \frac{m}{2} \rfloor - 1} \ldots k_0\}$ is uniquely decided by the $\{g_m g_{m-1} \ldots g_0\}$. The lemma follows by noting that any binary number H has a unique canonical binary SD form $\{g_m g_{m-1} \ldots g_0\}$ [67]. □

Lemma 7.3 Canonical radix-4 SD form is a minimal radix-4 SD representation.

Proof: From the proof of Lemma 7.2, we know that $k_i = 0$ if and only if both g_{2i} and g_{2i+1} are zero, and $k_i \neq 0$ if and only if either g_{2i} or g_{2i+1} is a nonzero digit, and moreover, g_{2i} and g_{2i+1} cannot be both nonzero. Thus, the canonical radix-4 SD form $\{k_{\lfloor \frac{m}{2} \rfloor} k_{\lfloor \frac{m}{2} \rfloor - 1} \ldots k_0\}$ has the same number of nonzero digits as its canonical binary SD form $\{g_m g_{m-1} \ldots g_0\}$.

For the sake of contradiction assume that the canonical radix-4 SD number $H = k_{\lfloor \frac{m}{2} \rfloor} k_{\lfloor \frac{m}{2} \rfloor - 1} \ldots k_0$ is not a minimal SD representation. Let $\{k'_n k'_{n-1} \ldots k'_0\}$ be another radix-4 SD representation of H, which is a minimal radix-4 SD number. Then there are fewer nonzero digits in $\{k'_n k'_{n-1} \ldots k'_0\}$ than those in $\{k_{\lfloor \frac{m}{2} \rfloor} k_{\lfloor \frac{m}{2} \rfloor - 1} \ldots k_0\}$ or in $\{g_m g_{m-1} \ldots g_0\}$. Applying the conversion rules $\bar{2} = \bar{1}0$, $\bar{1} = 0\bar{1}$, $0 = 00$, $1 = 01$, and $2 = 10$ to $\{k'_n k'_{n-1} \ldots k'_0\}$, we obtain a binary SD form $\{g'_{2n+1} g'_{2n} \cdots g'_0\}$ with the same number of nonzero digits as $\{k'_n k'_{n-1} \ldots k'_0\}$. Then the number of nonzero digits in $\{g'_{2n+1} g'_{2n} \cdots g'_0\}$ is less than that in $\{g_m g_{m-1} \ldots g_0\}$, which is however impossible since $H = g_m g_{m-1} \ldots g_0$ is a minimal binary SD representation. Thus the lemma holds. □

Note that 2-bit Booth [8] form uses the same digit set $\{\bar{2}, \bar{1}, 0, 1, 2\}$. The following lemma states a comparison between these two SD representations.

Lemma 7.4 The average number of nonzero digits in a minimal radix-4 SD number is asymptotically 11% less than that of its 2-bit Booth form.

Proof: First, we obtain the number of nonzero digits in the 2-bit Booth form. Let $X = x_{m-1}x_{m-2}\dots x_0$, $x_j \in \{0,1\}, j = 0,1,\dots,m-1$, be any binary number of length m, and Y its 2-bit Booth form. Let the two consecutive bits $x_{2i+1}x_{2i}$ in X correspond to the digit y_i in Y, where $y_i \in \{\bar{2},\bar{1},0,1,2\}$. The conversion rules for the 2-bit Booth form are given in Table 7.1 [8].

k	x_{2i+1} x_{2i}	x_{2i-1}	y_i
1	0 0	0	0
2	0 1	0	1
3	1 0	0	$\bar{2}$
4	1 1	0	$\bar{1}$
5	0 0	1	1
6	0 1	1	2
7	1 0	1	$\bar{1}$
8	1 1	1	0

Table 7.1: Radix-4 SD number encoding using 2-bit Booth algorithm [8].

(*a*) When $i = 0$; $x_{2i-1} = x_{-1} = 0$, and either of x_1 and x_0 with equal probability can be 0 or 1, corresponding to the cases $k = 1,2,3,4$. Then it is obvious from Table 7.1 that $\Pr(y_0 \neq 0) = \frac{3}{4}$, since each case of $k = 1,2,3,4$, has equal probability.

(*b*) When $1 \leq i \leq \frac{m}{2} - 1$ (m even), or $1 \leq i \leq \frac{m-1}{2} - 1$ (m odd); each of x_{2i+1}, x_{2i} and x_{2i-1} has equal probability to be 0 or 1. Consider in Table 7.1 all the values of k *i.e.*, $1,2,3,4,5,6,7,8$, each of which has the same probability, then we have $\Pr(y_i \neq 0) = \frac{6}{8} = \frac{3}{4}$.

(*c*) When $i = \frac{m}{2}$ (m even); $x_{2i} = x_{2i+1} = 0$, and x_{2i-1} can be 0 or 1, with equal probability. Consider the cases $k = 1,5$ with equal probability in Table 7.1, obviously, $\Pr(y_i \neq 0) = \frac{1}{2}$.

(*d*) When $i = \frac{m-1}{2}$ (m odd); $x_{2i+1} = 0$, and with equal probability, either of x_{2i} and x_{2i-1} can be 0 or 1. Consider in Table 7.1 the cases $k = 1,2,5,6$, which are equally likely, thus, $\Pr(y_i \neq 0) = \frac{3}{4}$.

Let L_Y be the average number of nonzeros in Y, then we have

$$L_Y = \begin{cases} \frac{3}{4} \times 1 + \frac{3}{4} \times (\frac{m}{2} - 1) + \frac{1}{2} \times 1 = \frac{3}{8}m + \frac{1}{2}, & m \text{ even}, \\ \frac{3}{4} \times 1 + \frac{3}{4} \times (\frac{m-1}{2} - 1) + \frac{3}{4} \times 1 = \frac{3}{8}m + \frac{3}{8}, & m \text{ odd}. \end{cases}$$

Therefore, for any large m, $L_Y = \frac{3}{8}m + \frac{7}{16} \to \frac{3}{8}m$.

Thus, the lemma follows by noting that the radix-4 minimal SD number of length $\frac{m}{2}$ has average $\frac{m}{3}$ nonzero digits. □

Below is an example which shows that the nonzero digits in a minimal radix-4 SD number are fewer than those in its 2-bit Booth form. It is worth noting here that the (reduced redundancy) minimal radix-4 SD number has fewer nonzero digits and shorter length than the corresponding 2-bit Booth form, since they both use the same symbol set $\{\bar{2}, \bar{1}, 0, 1, 2\}$.

Example 7.1 Consider the binary number 1001110010000110110. Its canonical binary SD form is $10100\bar{1}00100100\bar{1}0\bar{1}0$. Then, from the extended canonical recoding, its canonical radix-4 SD representation is $2\,2\,\bar{1}\,0\,2\,1\,0\,\bar{2}\,\bar{2}$. According to the conversion table for 2-bit Booth algorithm shown in Appendix A, however, its 2-bit Booth form is $1\,\bar{2}\,2\,\bar{1}\,1\,\bar{2}\,1\,\bar{1}\,2\,\bar{2}$.

Thus the 2-bit Booth form has three more nonzero digits than the canonical radix-4 SD representation. (Also notice that the 2-bit Booth form is longer than the canonical radix-4 SD representation by one digit.)

Lemma 7.5 Let a number H, in the conventional binary representation, be of length m. Then, for large m, the average length of its canonical radix-4 SD representation is $\frac{m}{2} + \frac{7}{12}$.

Proof: Let $H = h_{m-1}h_{m-2}\ldots h_0$ be a binary number of length m with its leading bit $h_{m-1} = 1$. Let L_1 and L_2 be the lengths of the corresponding canonical binary SD form and canonical radix-4 SD form of H, respectively. From Theorem 6.2 and for $m \gg 1$,

$L_1 \doteq m + \frac{2}{3}$. Or, $\Pr(L_1 = m) = \frac{1}{3}$ and $\Pr(L_1 = m+1) = \frac{2}{3}$. When m is even, $L_2 = \frac{1}{3} \times \frac{m}{2} + \frac{2}{3}(\frac{m}{2} + 1) = \frac{m}{2} + \frac{2}{3}$. When m is odd, $L_2 = \frac{m+1}{2}$.

Therefore, for any binary number of length m (large m), its canonical radix-4 SD representation has the average length of $m_2 = \frac{1}{2}\left(\frac{m}{2} + \frac{2}{3} + \frac{m}{2} + \frac{1}{2}\right) = \frac{m}{2} + \frac{7}{12}$. □

The use of the minimal radix-4 SD representation of H is advantageous over the 2-bit Booth form as stated in Lemma 7.4. The former can also reduce the computation time for exponentiation by about half of what is needed using the minimal binary SD representation (Lemma 7.5).

7.3 Exponentiation Algorithms

In the squaring and multiply scheme of exponentiation, a multiplication operation results from each nonzero digit in the exponent. Thus an exponent in minimal SD representation would require a minimum number of multiplications. Minimal binary SD representation can be obtained from its conventional binary form using the *canonical recoding* [67]. An alternative method for the conversion is Booth's algorithm which usually yields sub-optimal but not minimal SD number representation [16].

The algorithm for computing α^H, where α is a primitive root in $\mathbb{F}_{2^m}$ and H is represented as a minimal SD number, is given below.

Algorithm 7.2

```
X = 1;                    \* 1 ∈ F_{2^m} *\
FOR i = m  TO 0 DO
  {
  X = X * X;
  X = X * α^{g_i};        \* g_i ∈ {-1,0,1} *\
  }
```

The final value of X is α^H.

When the exponent is represented in the minimal radix-4 SD form, an analogue of the "square and multiply" algorithm is given below.

Algorithm 7.3

```
X = 1;                          \* 1 ∈ F_{2^m} *\
FOR i = ⌊m/2⌋ TO 0 DO
  {
   X = X^4;
   X = X * α^{k_i};             \* k_i ∈ {2̄, 1̄, 0, 1, 2} *\
  }
```

The final value of X is α^H.

Notice that the number of the fourth power operations in Algorithm 7.3 is only about half of the number of squaring operations in Algorithm 7.2. This feature can potentially reduce the dynamic power dissipation of the overall exponentiation structure, when implemented in VLSI technologies. However, Algorithm 7.3 requires a few finite field operations (e.g. multiplication with $\alpha^{\pm 2}$) which were not required in Algorithm 7.2. Efficient realization of these operations is discussed in the next section.

7.4 Implementation Using Polynomial Basis

Since α is primitive, there is a one-to-one correspondence between H and $y = \alpha^H$ in the range $0 \leq H \leq 2^m - 2$. In practical applications, H usually satisfies $\gcd(H, 2^m - 1) = 1$ for security considerations [69]. Recently proposed structures for exponentiation are to use the "square and multiply" scheme. However, the exponentiation of a primitive root, as we show here, can be calculated with the exponent represented by a minimal SD number which can effectively reduce the underlying multiplication operations.

Bit-parallel squarer and bit-parallel fourth power A polynomial basis bit-parallel squarer has been discussed in §5.3. When $f(x)$ is chosen as an irreducible trinomial

$x^m + x^k + 1$, by Theorems 5.7 and 5.8 we know that a bit-parallel squarer in $\mathbb{F}_{2^m}$ can be implemented with fewer than $\frac{3}{4}m$ XOR gates and a propagation delay of not greater than $2T_X$.

One way to obtain a bit-parallel fourth-power (FP) is to cascade two bit-parallel squarers. Then the complexities of the resultant architecture double those of a bit-parallel squarer.

If we consider $f(x)$ of a general form, the fourth power is $D(x) = A^4(x) \bmod F(x) = a_0 + a_1x^4 + a_2x^8 + \ldots + a_{\lceil \frac{m}{4} \rceil}x^{4\lceil \frac{m}{4} \rceil} + \ldots + a_{m-1}x^{4m-4} \bmod F(x)$, $0 \le \deg D(x) \le m-1$, and the corresponding $(m - \left\lceil \frac{m}{4} \right\rceil - 1)$-by-$m$ reduction matrix $\mathbf{P}$ is defined by

$$(x^{4\lceil \frac{m}{4} \rceil}, x^{4(\lceil \frac{m}{4} \rceil + 1)}, \ldots, x^{4m-4})^T = \mathbf{P}\,(1, x, \ldots, x^{m-1})^T,$$

$$\text{where } \mathbf{P} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,m-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,m-1} \\ \vdots & \vdots & \vdots & \vdots \\ p_{m-\lceil \frac{m}{4} \rceil -1,0} & p_{m-\lceil \frac{m}{4} \rceil -1,1} & \cdots & p_{m-\lceil \frac{m}{4} \rceil -1,m-1} \end{bmatrix}$$

The coefficients of $D(x) \triangleq \sum_{i=0}^{m-1} d_i x^i$ can be obtained by

$$d_i = \begin{cases} a_k + a_{\lceil \frac{m}{4} \rceil}p_{0,i} + a_{\lceil \frac{m}{4} \rceil + 1}p_{1,i} + \cdots + a_{m-1}p_{\lfloor \frac{3m}{4} \rfloor -1,i}, & i \text{ is a multiple of } 4, \\ a_{\lceil \frac{m}{4} \rceil}p_{0,i} + a_{\lceil \frac{m}{4} \rceil + 1}p_{1,i} + \cdots + a_{m-1}p_{\lfloor \frac{3m}{4} \rfloor -1,i}, & \text{otherwise}, \end{cases}$$

where $i = 0, 1, 2, \ldots, m-1$.

When $m = 4n + i$, $i = 0, 1, 2, 3$, and $n \in \mathbf{N}$, the upper limit of the size complexity of the fourth power is $\frac{3}{4}m^2 - \frac{3}{4}m$, $\frac{3}{4}m^2 - \frac{3}{2}m + \frac{3}{4}$, $\frac{3}{4}m^2 - \frac{5}{4}m + \frac{1}{2}$, and $\frac{3}{4}m^2 - m + \frac{1}{4}$, XOR gates, respectively, and the time complexity for all cases is at most $\left\lceil \log_2 \left(\left\lfloor \frac{3}{4}m \right\rfloor + 1 \right) \right\rceil$ T_X for arbitrary polynomial $F(x)$.

The complexities can be significantly reduced when we choose $f(x)$ as an irreducible trinomial or an irreducible pentanomial of the form $f(x) = x^m + x^{k+2} + x^{k+1} + x^k + 1$. When such a polynomial is of degree which is a Mersenne exponent, the bit-parallel fourth power complexities are given in Table 7.2.

For the fourth power (FP) module, let S_{FP} denote the size complexity in terms of the number of its 2-input XOR gates and T_{FP} the time complexity in terms of the number of levels of XOR gates needed to realize the FP module. Table 7.2 shows the values of S_{FP} and T_{FP} when $f(x)$ is an irreducible trinomial or pentanomial whose degree is a Mersenne exponent for $M_6 \leq \deg(f(x)) \leq M_{15}$, where M_j denotes the jth Mersenne exponent. Note that an irreducible polynomial is also a primitive polynomial if the degree is a Mersenne exponent.

$x^m + x^k + 1$			$x^m + x^{k+2} + x^{k+1} + x^k + 1$		
m,k	S_{FP}/m	T_{FP}	m,k	S_{FP}/m	T_{FP}
17,3	1.35	2	17,1	2.24	3
17,5	1.35	3	17,6	5.00	3
17,6	1.53	2	19,5	4.84	4
31,3	1.32	3	31,1	2.36	3
31,6	1.36	2	31,13	6.74	4
31,7	1.42	3	89,10	4.55	3
31,13	1.68	2	89,37	6.90	4
89,38	1.76	2	107,14	4.57	3
127,1	1.24	2	107,19	4.69	4
127,7	1.29	3	107,23	4.77	4
127,15	1.34	3	107,39	6.21	4
127,30	1.42	2	107,45	7.02	4
521,32	1.29	2	127,35	5.19	4
521,48	1.32	2	127,52	6.80	4
521,158	1.53	2	521,18	4.34	3
521,168	1.56	2	521,174	5.27	4
607,105	1.38	2	521,204	6.42	4
607,147	1.42	3	607,73	4.53	4
607,273	1.85	3			
1279,216	1.38	2			
1279,418	1.57	2			

Table 7.2: Table for bit-parallel fourth power complexity when $f(x)$ is a primitive trinomial or pentanomial ($k \leq \frac{m}{2}$) whose degree is a Mersenne exponent.

From Table 7.2, one can choose certain primitive trinomial $f(x)$ for which the FP module needs only $1\frac{3}{4}m$ XOR gates or fewer and has a time delay of at most two levels of XOR gates. However, if a pentanomial is used, according to Table 7.2, the FP module would require as many as about $5m$ XOR gates and cause a time delay of four layers of XOR gates.

Structure for exponentiation with a binary SD exponent Let $F(x)$ and α be given as above, then $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$ is a polynomial basis. Let A be a field element in $\mathbb{F}_{2^m}$ and $A = \sum_{i=0}^{m-1} a_i\alpha^i$. Since $F(\alpha) = 0$, it yields $\alpha^m = \sum_{i=0}^{m-1} f_i\alpha^i$ and $\alpha^{-1} = \sum_{i=0}^{m-1} f_{i+1}\alpha^i$. Then one can easily obtain

$$A\alpha = \sum_{i=0}^{m-1}[(1 - \delta_{i,0})a_{i-1} + a_{m-1}f_i]\alpha^i, \tag{7.1}$$

$$A\alpha^{-1} = \sum_{i=0}^{m-1}[(1 - \delta_{i,m-1})a_{i+1} + a_0 f_{i+1}]\alpha^i, \tag{7.2}$$

where $\delta_{i,j}$ is the Kronecker delta function which is 1 when $i = j$, and 0 otherwise.

Equations (1) and (2) show how a field element can be multiplied with α and α^{-1}. The corresponding realization using shift register is shown in Fig. 7.1. The LFSR is bidirectional which is referred to as BiLFSR and initially loaded with A. If a rightward shift is applied, the BiLFSR will have $A\alpha$, while a leftward shift will result in $A\alpha^{-1}$. It can be seen later that this BiLFSR is a building block of the structure for exponentiation of a primitive root α whose exponent is represented as a binary SD number. More on the BiLFSR can be found in [34].

The algorithm for the exponentiation of a primitive root with the exponent represented as a binary SD number is shown in Algorithm 7.2. The corresponding structure,

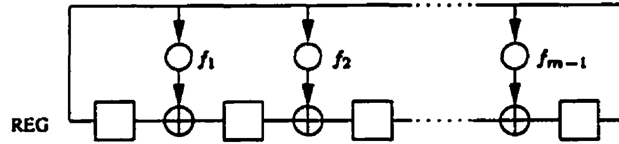

Figure 7.1: Bidirectional LFSR (BiLFSR) for multiplying a field element with a primitive root α and its inverse α^{-1}.

or EXP1 as we call it, is shown in Fig. 7.2. EXP1 has a BiLFSR for multiplication with $\alpha^{\pm 1}$. The direction of a shift depends on the sign of the nonzero digit of the SD exponent H. The bit-parallel squaring is employed for faster operation. Since EXP1 uses, the minimal binary SD form of the exponent, the average number of multiplications is $\frac{m}{3}$ as opposed to $\frac{m}{2}$ when the exponent is in the conventional binary form.

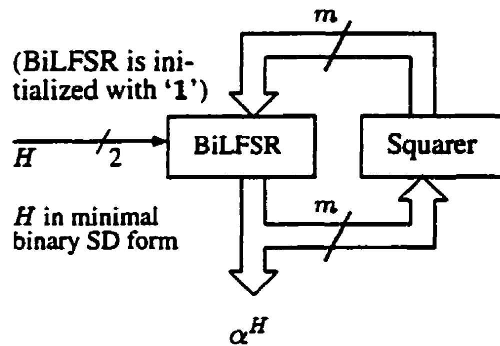

Figure 7.2: The Structure EXP1 for computing α^H with H being converted to a minimal binary SD number.

To further reduce the number of operations in the exponentiation in finite fields, a simple technique can be used on the exponent H before the exponentiation. If H is greater than half of $2^m - 1$, let $H \leftarrow H - 2^m + 1$, which results in performing square and multiply algorithm at *the other end of* '1' ($= \alpha^{2^m-1}$), rather than $1 = \alpha^0$. This technique would save one square and multiply operation on average. More significantly, when H is very close to 2^m, or $2^m - H$ is very small, only about $\lfloor \log_2(2^m - H + 1) \rfloor$ square and multiply operations are needed using the modified method. For example, to compute $\alpha^{2^{1000}-5}$, where α is a primitive element in $\mathbb{F}_{2^{1000}}$, the modified method requires

only about $\lfloor \log_2(2^{1000} - H + 1) \rfloor = \lfloor \log_2 6 \rfloor = 2$ square and multiply operations, rather than about $\lfloor \log_2 H \rfloor = 999$ square and multiply operations. To use this method, all that is needed is to reduce the exponent as $H \leftarrow H - 2^m + 1$ before applying Algorithm 7.2 to it to compute the exponentiation.[a] This simple operation can be combined into the exponent conversion with little extra hardware, since all that is needed is to change $H = h_{m-1}h_{m-2}\dots h_0$ into its two's complement if $h_{m-1} \neq 0$.

Complexity of EXP1 Here we give the size and the time complexities of EXP1. For the BiLFSR and Squaring modules, the total number of XOR gates is at most $\left\lfloor \frac{m}{2} \right\rfloor m - \left\lfloor \frac{m}{2} \right\rfloor + W_F - 2$, where W_F is the Hamming weight of $F(x)$, together with m 1-bit registers. If we choose the clock cycle period as the same as the delay of the squaring operation T_S (for example, exponentiation in $\mathbb{F}_{2^{1000}}$, $T_S \leq \left\lceil \log_2 \left(\left\lfloor \frac{1000}{2} \right\rfloor + 1 \right) \right\rceil \times T_X = 9T_X$, where T_X is the delay in one XOR gate), the exponentiation can be completed in $\lfloor \log_2 H \rfloor + \frac{2}{3}$ clock cycles on average.

If $F(x) = 1 + x^k + x^m$, $1 \leq k \leq \left\lfloor \frac{m}{2} \right\rfloor$, at most $\frac{3m}{4}$ XOR gates besides m 1-bit registers are required, and the delay of the squaring operation $T_S \leq 2T_X$. It both increases the computing speed and reduces the size complexity significantly. Since a multiplication operation (either with α or α^{-1}) is due to a nonzero digit in the SD number representation of the exponent, the number of multiplication operations would be minimized using the minimal SD number representation and the structure can potentially reduce the dynamic power dissipation when implemented in VLSI technologies.

Structure for exponentiation with a radix-4 SD exponent Since $F(\alpha) = 0$, we have

$$\alpha^{m+1} = \sum_{i=0}^{m-1} [f_{m-1} f_i + (1 - \delta_{i,0}) f_{i-1}] \alpha^i \tag{7.3}$$

[a] A similar idea can be applied to Algorithm 7.3 where radix-4 exponents are used.

$$\alpha^{-2} = \sum_{i=0}^{m-1}[f_1 f_{i+1} + (1-\delta_{i,m-1})f_{i+2}]\alpha^i. \tag{7.4}$$

Thus from (1), (2), (3) and (4), we obtain

$$A\alpha^2 = \sum_{i=0}^{m-1}[(1-\delta_{i,0}-\delta_{i,1})a_{i-2} + a_{m-2}f_i + a_{m-1}((1-\delta_{i,0})f_{i-1} + f_{m-1}f_i)]\alpha^i$$

$$A\alpha^{-2} = \sum_{i=0}^{m-1}[(1-\delta_{i,m-2}-\delta_{i,m-1})a_{i+2} + a_1 f_{i+1} + a_0((1-\delta_{i,m-1})f_{i+2} + f_1 f_{i+1})]\alpha^i$$

Fig. 7.3(a) shows the structure for multiplications with α^2 and α^{-2} using the polynomial basis. The field element, whose coordinates are stored in *REG*, is multiplied with α^2 when a right-shift is applied and multiplied with α^{-2} when a left-shift is applied. Comparing Fig. 7.3(a) with Fig. 7.1, one can see that the former has a BiLFSR for $\alpha^{\pm 1}$ multiplication embedded in it. Thus, with minor modification it can be used for both $\alpha^{\pm 1}$ and $\alpha^{\pm 2}$ multiplications as shown in Fig. 7.3(b), which is referred to as the Extended BiLFSR or XBiLFSR. When the switches are at upper positions (solid lines), the circuits are configured to perform $\alpha^{\pm 2}$ multiplications. When they are at lower positions (dotted lines), the upper branch of the circuits is disconnected and the circuits are able to multiply with $\alpha^{\pm 1}$. The switches are controlled by the signed digits of the exponent. For simplicity, the control circuitry is omitted from the figure.

When the exponent is represented as a minimal radix-4 SD number, the XBiLFSR can be used to realize Algorithm 7.3. The corresponding structure (referred to as EXP2) is shown in Fig. 7.4. Here the XBiLFSR is used for the multiplication with $\alpha^{\pm 1}$ or $\alpha^{\pm 2}$, and the bit-parallel fourth power replaces the squaring for performing power of four. The sign of the digit in the radix-4 SD exponent would control the shifting directions while the absolute value of the nonzero digit would decide switch positions in the XBiLFSR. The number of clock cycles needed in EXP2 is about half of that of EXP1.

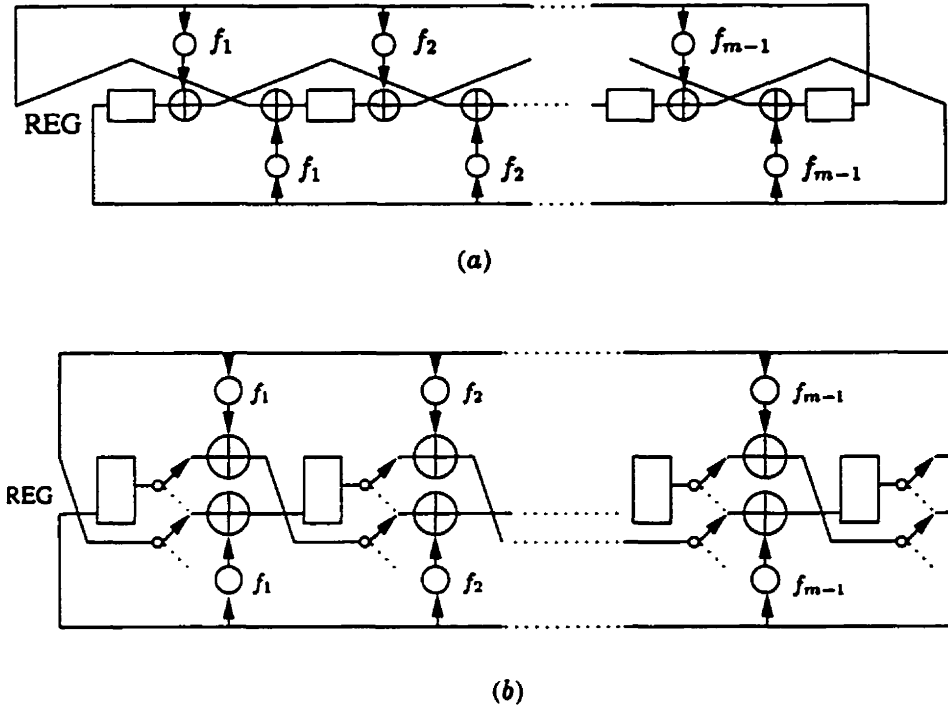

Figure 7.3: (a) Circuits for multipling a field element with α^2 and α^{-2}; (b) Circuits for multipling a field element with $\alpha^{\pm 1}$ and $\alpha^{\pm 2}$.

Complexity of EXP2 There are at most $2m - 2$ XOR gates, together with m 1-bit registers in the XBiLFSR. If $F(x)$ is chosen as a trinomial, XBiLFSR would only need two XOR gates and m 1-bit registers. The size complexity of the fourth power module is at most $\left\lfloor \frac{3}{4}m \right\rfloor m - \left\lfloor \frac{3}{4}m \right\rfloor$ (XOR gates), which also heavily depends on the choice of the field-generating polynomial. The system clock cycle period should be no shorter than $T_{FP} \leq \left\lceil \log_2 \left(\left\lfloor \frac{3}{4}m \right\rfloor + 1 \right) \right\rceil \times T_X$, where T_{FP} is the delay of a fourth power operation. The time required for an exponentiation would be $\frac{1}{2}\lfloor \log_2 H \rfloor + \frac{7}{12}$ clock cycles on average. If a primitive trinomial is chosen, the size complexity would be not greater than $1\frac{3}{4}m$ gates and propagation delay is within $4T_X$.

Exponent conversion The exponent H is represented as a minimal binary SD number in EXP1, and as a minimal radix-4 SD number in EXP2. Where EXP1 or EXP2 is only a

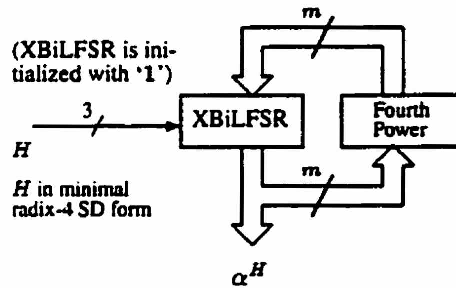

Figure 7.4: The Structure EXP2 for α^H with H represented as a minimal radix-4 SD number.

part of a larger system, H might be available in the form of a conventional binary number. In such cases, simple extra circuitry can be used to obtain the required SD representation as briefly explained below.

Assume that bits $h_0h_1 \ldots h_{m-1}$ are stored in an m stage shift register R (Fig. 7.5) from where these bits sequentially enter Converter1. The latter simply realizes the canonical recoding, and can be readily implemented using 2 flip flops and a few logic gates arranged in two levels. The outputs of Converter1, each of which consists of two bits, are pushed into the stack S1 from where EXP1 gets H in the required SD form. The stack allows the SD symbols enter EXP1 in the reverse order.

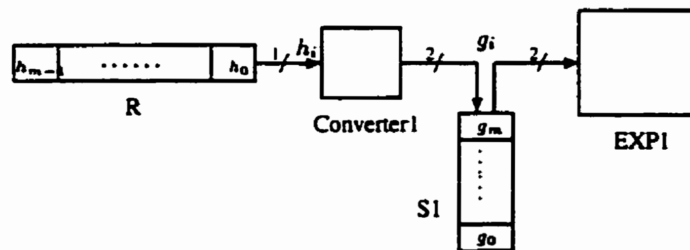

Figure 7.5: EXP1 with a converter that performs the canonical recoding.

For EXP2, to take the advantage of its lower computation time which is about half of that of EXP1, bits $h_0h_1 \ldots h_{m-1}$ are stored into two $\left\lceil \frac{m}{2} \right\rceil$ stage shift registers — one register with $h_0h_2h_4 \ldots$ and the other with $h_1h_3h_5 \ldots$ (see Fig. 7.6(a)). These two registers are shifted in parallel to allow two consecutive bits to enter Converter2 every clock

cycle. Converter2 realizes the extending canonical recoding to generate the sequence of symbols $k_0 k_1 \ldots k_{\lfloor \frac{m}{2} \rfloor}$ each of which consists of three bits. These symbols are stacked in S2 and then enter EXP2 in the reverse order. Converter2 can be implemented using 4 flip flops and a few logic gates as shown in Fig. 7.6(*b*). Compared to Converter1, the gate count is almost doubled in Converter2.

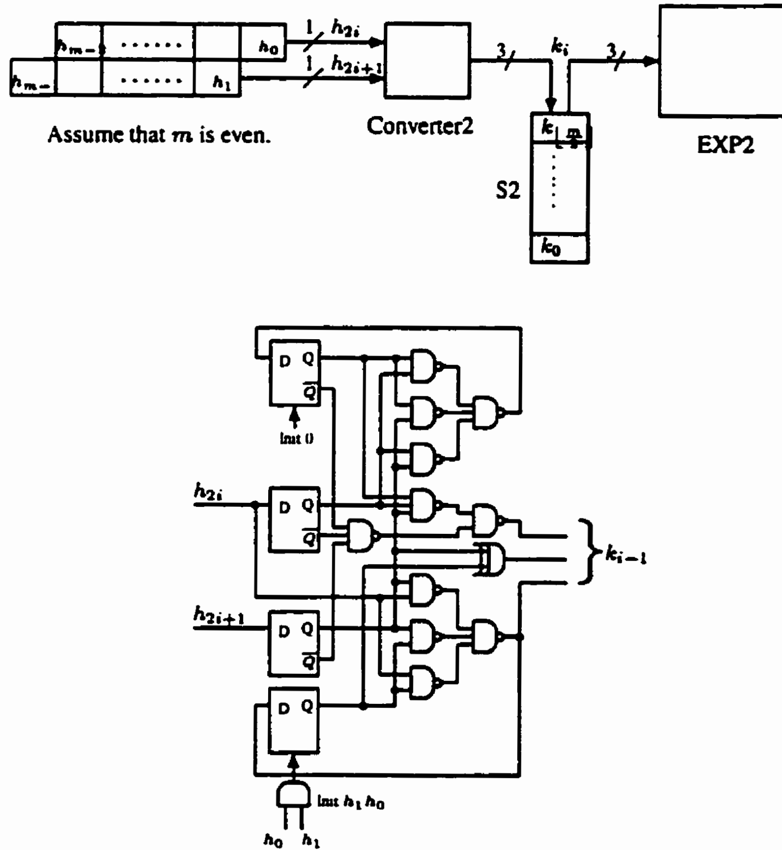

Figure 7.6: (*a*) EXP2 with a converter that performs the extended canonical recoding; (*b*) Circuits for the extended canonical recoding (Algorithm 7.1).

To pipeline exponentiation operations, an extra stack (like S1 with Converter1 or S2 with Converter2) can be used. Notice that the structures of Converter1 as well as Converter2 are independent of the values of m and do not slow down the clock speed at which EXP1 and EXP2 operate.

If one could use algorithms for obtaining the required minimal SD representations by scanning the sequence $h_0h_1 \dots h_{m-1}$ from the most significant bit, there would not be any need for stacks S1 and S2; consequently, the time to reverse the order of the SD symbols could be saved. However, it seems the implementation of such algorithms may not be simple. As a trade-off, one can use the 2-bit Booth algorithm which, however, generates about 11% more non-zero symbols on average as compared to the extended canonical recoding as stated in Lemma 7.6.

7.5 Realization Using Weakly Dual Basis

Exponentiation algorithm An analogue to Algorithm 7.1 by using WDB is as follows:

Algorithm 7.4

```
X = 1;                          \* 1 ∈ F_{2^m} in PB *\
FOR i = ⌊m/2⌋ TO 0 DO
  {
   X = X^4;
   X* = T · X;                  \* basis conversion: from PB to WDB *\
   X* = (X · α^{k_i})*;         \* k_i ∈ {2̄, 1̄, 0, 1, 2} *\
   X = T^{-1} · X*;             \* basis conversion: from WDB to PB *\
  }
```

The final value of X is α^H.

Since multiplication by α^{k_i} is achieved with weakly dual bases, basis conversions before and after multiplication operation at each iteration are necessary. Basis conversions are usually realized by multiplying by the conversion matrix $\boldsymbol{T}$ or the inverse of $\boldsymbol{T}$. From Lemma 4.1 we know that those matrix operations can be a simple permutation of the coordinates if a primitive trinomial is chosen as the generating polynomial.

If the multiplication operations are efficiently implemented, we can then increase the computing speed as well as potentially reduce the dynamic power dissipation of the overall exponentiation structure when implemented in VLSI technologies.

Multiplication in weakly dual bases Following the discussion in §4.2.1, multiplications of A by some powers of α can be obtained below.

$$(\alpha A)_j^* = \begin{cases} \mathrm{Tr}(\gamma\alpha^{j+1}A) &= a_{j+1}^* \quad 0 \le j \le m-2, \\ \mathrm{Tr}(\gamma\alpha^{m}A) &= \sum_{i=0}^{m-1} f_i a_i^* \quad j = m-1; \end{cases} \tag{7.5}$$

$$(\alpha^{-1} A)_j^* = \begin{cases} \mathrm{Tr}(\gamma\alpha^{j-1}A) &= a_{j-1}^* \quad 1 \le j \le m-1, \\ \mathrm{Tr}(\gamma\alpha^{-1}A) &= \sum_{i=0}^{m-1} f_{i+1} a_i^* \quad j = 0; \end{cases} \tag{7.6}$$

$$(\alpha^{2} A)_j^* = \begin{cases} \mathrm{Tr}(\gamma\alpha^{j+2}A) &= a_{j+2}^* \quad 0 \le j \le m-3, \\ \mathrm{Tr}(\gamma\alpha^{m}A) &= \sum_{i=0}^{m-1} f_i a_i^* \quad j = m-2, \\ \mathrm{Tr}(\gamma\alpha^{m+1}A) &= \sum_{i=1}^{m-1} (f_{i-1} + f_{m-1} f_i) a_i^* + f_0 f_{m-1} a_0^* \quad j = m-1; \end{cases} \tag{7.7}$$

$$(\alpha^{-2}A)^*_j = \begin{cases} \mathrm{Tr}(\gamma\alpha^{j-2}A) = a^*_{j-2} & 2 \le j \le m-1, \\ \mathrm{Tr}(\gamma\alpha^{-1}A) = \sum_{i=0}^{m-1} f_{i+1}a^*_i & j = 1, \\ \mathrm{Tr}(\gamma\alpha^{-2}A) = \sum_{i=0}^{m-2}(f_{i+2} + f_1 f_{i+1})a^*_i + f_1 a^*_{m-1} & j = 0. \end{cases} \tag{7.8}$$

Equations (7.5) and (7.6) can be readily realized as shown in Fig. 7.7(a). The LFSR is bidirectional which is referred to as (Fibonacci type) BiLFSR. The coordinates of A with respect to the weakly dual basis are initially loaded into BiLFSR. If a rightward shift is applied, the BiLFSR will have $A\alpha$, while a leftward shift will result in $A\alpha^{-1}$. BiLFSR can be extended for realizing multiplication by α^2 and α^{-2} (Fig. 7.7(b)). By combining these two LFSRs, we obtain a structure for the multiplication of A with both $\alpha^{\pm 1}$ and $\alpha^{\pm 2}$, which is referred to as (Fibonacci type) *extended bidirectional LFSR* or XBiLFSR and shown in Fig. 7.7(c). When the switches are at upper positions (solid lines), the circuits are configured to perform $\alpha^{\pm 2}$ multiplications. When they are at lower positions (dotted lines), the upper branch of the circuits is disconnected and the circuits are able to multiply with $\alpha^{\pm 1}$. The switches are controlled by the signed digits of the exponent. For simplicity, the control circuitry is omitted from the figure. When the exponent is represented as a minimal radix-4 SD number, the XBiLFSR can be used to realize the multiplication operation step in Algorithm 7.4.

When the generating polynomial $f(x)$ is a trinomial, the XBiLFSR can be built with only two two-input XOR gates and m 1-bit registers and minimal clock period can be chosen as no shorter than the time delay of one layer of XOR gates.. When $f(x)$ is chosen as a pentanomial, four more two-input XOR gates are required to construct the XBiLFSR, while the clock period should be equal to or longer than the time delay caused

by two layers of XOR gates.

Basis conversions Since the product obtained from the XBiLFSR is in weakly dual basis and the fourth power operation requires polynomial basis, intermediate results should be converted between the weakly dual basis and the polynomial basis before and after the multiplication operation. The conversions of bases can be greatly simplified if we choose a proper generating polynomial $f(x)$.

From the discussion in § 4.5, we have the following two lemmas:

Lemma 7.6 [60, 77] Let $f(x) = x^m + x^k + 1$ be an irreducible trinomial in $\mathbb{F}_{2^m}$ and α its root. Then for the polynomial basis $\{\alpha^i\}$, there exists a weakly dual basis and it has the form: $\{\alpha^{\pi(0)}, \alpha^{\pi(1)}, \ldots, \alpha^{\pi(m-1)}\}$, where $\pi(j) = k - 1 - j \bmod m, 0 \leq j \leq m-1$.

In Lemma 7.6, the permutation can be done with a cyclic shift of lines with no time delay. Since $\pi(\pi(i)) = \pi(k - i - 1 \bmod m) = [k - 1 - (k - i - 1)] \bmod m = i$, we have $\pi^{-1}(i) = \pi(i) = k - i - 1 \bmod m$. Therefore the same permutation can also be used to perform the conversion from the weakly dual basis back to the polynomial basis.

Lemma 7.7 [60] Let $f(x) = x^m + x^{k+2} + x^{k+1} + x^k + 1$ be an irreducible pentanomial and α its root. Then for polynomial basis $\{\alpha^i\}$, there exists a weakly dual basis which has the form: $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$, where

$$\beta_j = \begin{cases} 1 + \alpha^k & \text{for } j = 0, \\ \alpha^{k-j} & \text{for } 1 \leq j \leq k, \\ \alpha^{m-j+k} & \text{for } k+1 \leq j \leq m-2, \\ \alpha^{k+1} + \alpha^{m-1} & \text{for } j = m-1. \end{cases}$$

When $f(x) = x^m + x^{k+2} + x^{k+1} + x^k + 1$, it can be checked that two two-input XOR gates are required to realize the basis conversion from the polynomial basis to the weakly dual basis or from the dual basis back to the polynomial basis.

System architecture and its complexities When α is a root of a trinomial, the architecture for exponentiation using weakly dual basis is shown in Fig. 7.8, which is called EXP3. Here the XBiLFSR is used for the multiplication with $\alpha^{\pm 1}$ or $\alpha^{\pm 2}$, and the bit-parallel fourth power for performing power of four. The sign of the digit in the radix-4 SD exponent would control the shifting directions while the absolute value of the nonzero digit would decide switch positions in the XBiLFSR. The permutation block is to realize basis conversions between the polynomial basis and the weakly dual basis, which is a simple re-arrangement of lines. If the polynomial basis is generated by a primitive pentanomial of the form $f(x) = x^m + x^{k+2} + x^{k+1} + x^k + 1$, the two permutation blocks would be replaced by two slightly more complicated basis conversion blocks which can be implemented with four two-input XOR gates. Fourth power module is implemented in bit-parallel fashion with combinational logic. Its time delay would determine the minimum period of the system clock.

The time required for an exponentiation would be about $\frac{1}{2}\log_2 H$ clock cycles on average if the exponent H is available in its minimum radix-4 SD form. When the generating polynomial is a primitive trinomial whose degree is a Mersenne exponent, the FP module needs fewer than $1\frac{3}{4}m$ XOR gates and has a time delay of at most two levels of XOR gates. Consequently, the size complexity of the proposed system is less than $(1\frac{3}{4}m + 2)$ XOR gates, together with m 1-bit registers, and the system clock period can be chosen as $T_{\text{clock}} \geq 2T_{\text{XOR}}$, where T_{XOR} is the time delay of one layer of XOR gates. When one primitive pentanomial in Table 7.2 is to be used as the generating polynomial, substantially more gates are required to implement FP module and XBiLFSR. The system complexity can be as much as $(5m + 6)$ XOR gates and m 1-bit registers. The system clock period should be chosen as no shorter than the time delay of four layers of XOR gates.

When the generating polynomial $f(x)$ is an irreducible trinomial, the proposed weakly dual basis method in this section achieves time and size complexities which are equivalent to those described in the last section. Using weakly dual basis rather than polynomial

basis, the proposed method provides an example of the equivalence in the complexities of implementation of many finite field operations by using different bases.

7.6 Comparisons

In this section, the proposed exponentiation schemes are compared with the existing schemes for similar operations which have been briefly reviewed in §7.1. In the following discussion, it is assumed that for an arbitrary $\mathbb{F}_{2^m}$ the gate count for a bit-parallel multiplier using either a polynomial or normal basis is proportional to m^2 and that for a bit-parallel squarer using the polynomial basis is proportional to m. (It has already been mentioned that squaring using normal basis is free of cost.)

In the full-parallel scheme of stored conjugates method [69, 50], let L be the delay due to an m-bit multiplier, then the time complexity of an exponentiation is proportional to $L \times \lceil \log_2(m-1) \rceil$, where $\lceil \log_2(m-1) \rceil$ is the depth of the multiplier tree. If the bit-parallel multipliers are employed, L would be greater than both T_S and T_{FP}, where T_S and T_{FP} are the time delays of the squaring operation and the fourth power operation proposed in this paper, respectively. While $\lceil \log_2(m-1) \rceil$ would be much less than $\log_2 H$, and the total time of exponentiation might be less than those of our proposed structures, the gate count of the bit-parallel multiplier tree is, for an arbitrary $F(x)$, proportional to m^3 for both polynomial and normal basis multipliers, which is much higher than those of our proposed methods (whose complexities are proportional to m^2). When trade-off is made in the stored conjugates methods by using a smaller multiplier tree, the time complexity would increase while the size complexity is still much higher than those of our proposed methods, since the number of gates in a polynomial or normal basis bit-parallel multiplier is proportional to m^2. If the bit-sequential multipliers are employed, the multiplier tree has a gate count of about $4m^2$ for a polynomial basis multiplier or $5m^2$ for normal basis multiplier and the time for performing an exponentiation is at least $m\lceil \log_2(m-1) \rceil$ clock cycles. In this case, all size and time complexity and

consequently power consumption are higher than those of our proposed structures. An additional memory of m^2 bits is also required for the stored conjugates methods which we do not take into consideration for comparison.

If the squaring and multiply scheme (Algorithm 7.2) is adopted, both the polynomial basis and normal basis can be used. An exponentiation structure with the polynomial basis is presented in [50]. The squaring module is the same as that of the structure in this paper. The multiplication with α has a simple non-LFSR structure with complexity of $W_F - 2$ XOR gates, where W_F is the Hamming weight of $F(x)$. At least one m-bit register is required to temporarily store the intermediate results and support the iterations. The time complexity is about the same as that of EXP1, which is $(\lfloor \log_2 H \rfloor + 1)T_S$, where T_S is the delay caused by one squaring operation. However more multiplication operations would be performed and therefore more power dissipation would be required compared with EXP1. If the normal basis is used, the squaring is readily implemented with a shift of the coefficients, while the multiplier is more complicated. It is shown in [50] that the complexity of multiplier (multiplication with a constant) and squaring pair in the normal basis is higher than that in the polynomial basis. Even when an optimal normal basis is chosen, the size complexity is proportional to $3m$, which is still higher than that in the polynomial basis when the field-generating polynomial $F(x) = x^m + x^k + 1, k \leq \left\lfloor \frac{m}{2} \right\rfloor$ is used.

Some specific classes of fields can be used to reduce the multiplier complexity (for example, the field-generating polynomial is a trinomial [50] or m is a power of 2 [63]), and the comparisons can be made in a similar way. However, it is worth noting here that the base α is a primitive element and hence $F(x)$ should be primitive. As a result, multipliers based on all one polynomials (AOP) or equally spaced polynomials (ESP) as proposed in [35, 36] cannot be used since AOPs and ESPs are non-primitive irreducible polynomials (except when the polynomial is of degree two).

Simple schemes for exponentiation are to use LFSR or LUT. The time needed to compute an exponentiation using LFSR is H clock cycles, which is much more than

those of all the other methods discussed above. This method is adequate only for small m. The LUT method is simple in design, and it requires the use of a memory of size proportional to $m \times 2^m$, which might be unacceptable even when m is moderately large.

In our proposed structures using PB or WDB, the gate count for EXP1 is about 66.7% of that of EXP2 or EXP3, while the computation time for EXP2 or EXP3 is 50% of that of EXP1. The number of multiplications is reduced to minimum in both structures since the minimal SD representations are employed.

7.7 Chapter Summary

Exponentiation of a primitive element has applications in cryptography. In this chapter, we have presented architectures for realizing this computation for the cases where the element is represented with respect to a polynomial basis or a weakly dual basis. Compared to previous proposals, the new proposals have lower size complexity, shorter propagation delay, and thus they are expected to require less power when implemented in VLSI technologies.

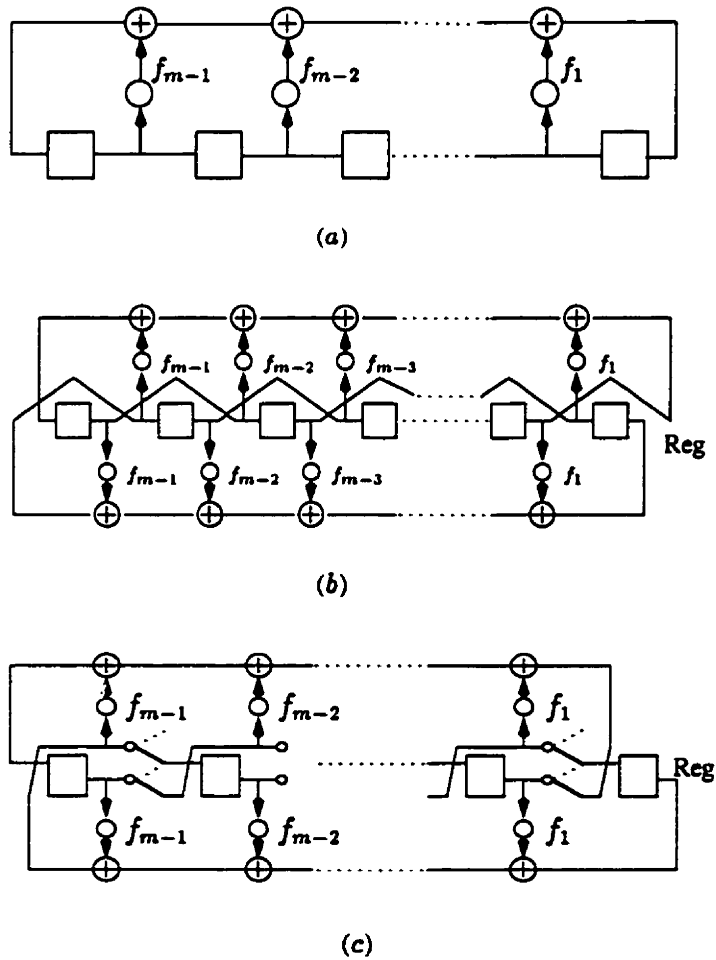

Figure 7.7: (*a*) The BiLFSR: LFSR for multiplying by $\alpha^{\pm 1}$; (*b*) The LFSR for multiplying by $\alpha^{\pm 2}$; (*c*) The XBiLFSR: LFSR for multiplying by $\alpha^{\pm 1}$and $\alpha^{\pm 2}$.

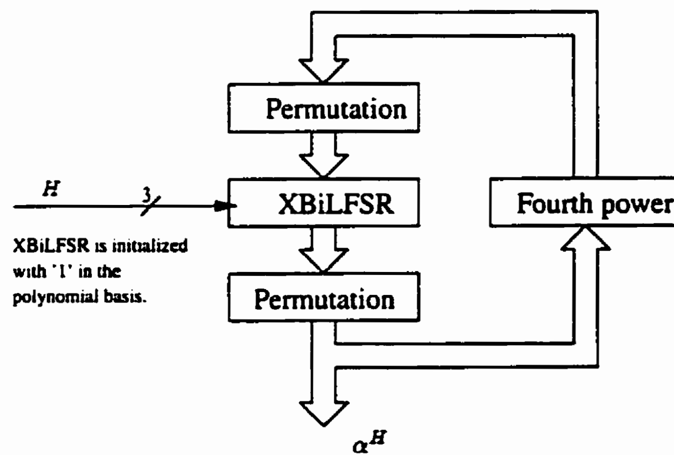

Figure 7.8: System diagram for exponentiation when the weakly dual basis is used.

Chapter 8

Efficient Computations for Elliptic Curve Cryptosystems

8.1 Introduction

In this chapter, the computation of $m_1P_1 + m_2P_2 + \cdots + m_kP_k$ is considered, where m_i is an integer and P_i a point on an elliptic curve for $i = 1, 2, \ldots, k$. When $k = 1$, mP is the primary operation in most EC cryptosystems [56]. When $k = 2$, computation of $m_1P_1 + m_2P_2$ has applications in various elliptic curve signature schemes [56]. When $k = 3$, the computation of $m_1P_1 + m_2P_2 + m_3P_3$ is required in verifying ElGamal signature [57]. With $k = 4$, we have $m_1P_1 + \cdots + m_4P_4$ which has application in the Burmester-Desmedt keying scheme [57]. In the recent past, several algorithms have been proposed for efficiently computing mP, *e.g.*, [59, 43, 53, 46, 32, 30, 72]. The general idea behind these algorithms for computing mP is to find ways to minimize the number of point operations (*i.e.*, elliptic addition or doubling for non-supersingular curve).

In order to compute mP, Agnew, Mullin and Vanstone have applied the ***double-and-add*** method [4], while Morain and Olivos have used the binary signed-digit (SD)

non-adjacent form (NAF) to compute mP [59]. The use of the SD form is based on the fact that obtaining an additive inverse of a given point is at little cost. Koyama and Tsuruoka have speeded up the computation using a binary SD window scheme [46]. In their method, first m is transformed into a binary SD form with fewer zero runs than that of the NAF, and certain point multiples are computed and stored. Then computation of mP is performed with elliptic doublings and additions of some of the stored values alone.

On the other hand, Shamir has developed a novel scheme to compute multiple exponentiation of the form $M_1^{e_1} M_2^{e_2} \cdots M_k^{e_k}$ [57], which can be easily modified to compute $m_1 P_1 + \cdots + m_k P_k$. With a few extra stored values, his method can compute multiple exponentiation in the same way as performing a single exponentiation operation using square-and-multiply method. An extension of Shamir's method proposed by Yen and Laih requires more stored points by using a window method, and thus fewer steps are needed for obtaining the final result [84]. An alternative to the window method is the 'comb algorithm' which is proposed in [48] and described in [57].

Efficient computation of mP has also been proposed for a class of non-supersingular curves – *e.g.*, anomalous or Koblitz curves [43, 53, 72, 29]. The recent work by Solinas shows that computation of mP on a Koblitz curve requires only about $\frac{n}{3}$ elliptic additions, where n is the degree of the finite field over which the group of points is defined [72, 29].

In the sequel, the set of $\mathbb{F}_{2^n}$-rational points $E(\mathbb{F}_{2^n})$ on a nonsupersingular curve E is considered. We assume that the point P_1 has prime order p and thus m_1 a positive integer less than p. While $P_2, \ldots, P_k$ are other points on the curve of order not greater than that of P_1 and integers $m_2, \ldots, m_k$ are also not greater than p. Since $p \leqslant 2^{n-1} + O(2^{\frac{n}{2}-1})$, then the binary form of any one of $m_1, \ldots, m_k$ is of length not greater than n. In the sequel, for generality, we, however, denote the binary length of m_i by h.

The organization of this chapter is as follows. In § 8.2, a general sliding window method for computing $m_1 P_1 + m_2 P_2 + \cdots + m_k P_k$ on nonsupersingular curves is pro-

posed. A new SD representation with fewer zero runs is proposed and its application to the computation of point multiples is discussed in 2S 8.3. Such computation of addition of point multiples on a Koblitz curve is considered in § 8.4. Finally, numerical comparisons are made in § 8.5.

8.2 Sliding Window Method for Non-Supersingular Curves

8.2.1 Modified Shamir's method

Since negating a point is equally expensive as adding a point, a simple modification of Shamir's method can be used to compute $m_1P_1 + m_2P_2 + \cdots + m_kP_k$.

Let us consider the case of $k = 2$. First compute points $P_1 + P_2$ and $P_1 - P_2$ and store them along with the points P_1 and P_2. While m_1 and m_2 are converted into their NAFs:

$$m_1 = m_h^{(1)}m_{h-1}^{(1)}\cdots m_0^{(1)} \text{ and } m_2 = m_h^{(2)}m_{h-1}^{(2)}\cdots m_0^{(2)},$$

where $m_j^{(i)} \in \{-1, 0, 1\}$. for $i = 1, 2$ and $j = 0, 1, \ldots, h$.

Now we have

$$\begin{aligned}
& m_1P_1 + m_2P_2 \\
= \;& [2^hm_h^{(1)} + 2^{h-1}m_{h-1}^{(1)} + \cdots + 2^0m_0^{(1)}]P_1 + [2^hm_h^{(2)} + 2^{h-1}m_{h-1}^{(2)} + \cdots + 2^0m_0^{(2)}]P_2 \\
= \;& 2^h[m_h^{(1)}P_1 + m_h^{(2)}P_2] + 2^{h-1}[m_{h-1}^{(1)}P_1 + m_{h-1}^{(2)}P_2] + \cdots + 2^0[m_0^{(1)}P_1 + m_0^{(2)}P_2] \\
= \;& \underbrace{2(\cdots 2(2}_{h}(m_h^{(1)}P_1 + m_h^{(2)}P_2) + m_{h-1}^{(1)}P_1 + m_{h-1}^{(2)}P_2) + \cdots) + m_0^{(1)}P_1 + m_0^{(2)}P_2
\end{aligned}$$

Since $m_j^{(1)}P_1 + m_j^{(2)}P_2$, for $0 \leqslant j \leqslant h$, is either the point $\mathcal{O}$ or one of the stored points or the negative of one stored points, $m_1P_1 + m_2P_2$ can be computed in a double-and-add

fashion.

Algorithm 8.1 Modified Shamir's Method

Input: Integers m_i, w, and points P_i, $i = 1, 2, \ldots, k$.

Output: Point $P = m_1P_1 + \cdots + m_kP_k$.

1 Compute and store the points $l_1P_1 + l_2P_2 + \cdots + l_kP_k$, where $l_i \in \{-1, 0, 1\}$, not all l_i's are zeros and first nonzero l_i is positive.

2 Convert m_i into the NAF, for $i = 1, 2, \ldots, k : m_i = m_h^{(i)} \ldots m_1^{(i)} m_0^{(i)}$, where $m_j^{(i)} \in \{-1, 0, 1\}$, for $j = 0, 1, \ldots, h$; Place them as an array of size $k \times (h + 1)$.

3 Starting from the leftmost end, find the first nonzero column $[m_j^{(1)}, \ldots, m_j^{(k)}]^T$ and its corresponding point stored as P_0 (or $-P_0$); Set $L \leftarrow j$ and $P \leftarrow P_0$ (or $P \leftarrow -P_0$).

4 **Do while** $L \geqslant 0$ **Begin:**

4.1 Set $P \leftarrow 2P$.

4.1 Set $P \leftarrow P + P'$ (or $P \leftarrow P - P'$) if the next column is not a zero column and the corresponding point being stored as P' (or $-P'$).

4.1 Set $L \leftarrow L - 1$.

End.

The performance of Algorithm 8.1 can be summarized in the following theorem.

Theorem 8.1 For computing $m_1P_1 + m_2P_2 + \cdots + m_kP_k$, the modified Shamir's method requires $h - \frac{1}{3} - \frac{1}{2^k - 1}$ elliptic doublings and $[1 - (\frac{2}{3})^k][h - \frac{1}{3} - \frac{1}{2^k - 1}] + \frac{3^k - 1}{2} - k$ additions on average, with $N_{\text{store}} = \frac{3^k - 1}{2}$ stored points. The worst case performance is: $h + \frac{3^k - 1}{2} - k$ elliptic additions and h doublings.

A sketch of proof:

1. The number of the stored points is $\frac{3^k - 1}{2}$.

2. The number of elliptic additions required for stored points is $\frac{3^k - 1}{2} - k$. This can be shown as follows: Let S denote the set of the stored points. Let A_1 be any point $\in S$ except for $P_1, P_2, \ldots, P_k$. Write $A_1 = l_1 P_1 + l_2 P_2 + \cdots + l_k P_k$, without loss of generality, assume that $l_1 > 0$. Then $l_1 = 1$, and point A_1 can be computed from the point $A_2 = l_2 P_2 + \cdots + l_k P_k \in S$ by one addition. Perform this process on A_2 and repeat until A_j is one of $P_1, P_2, \ldots, P_k$. Then the statement follows by noting that the k points $P_1, P_2, \ldots, P_k \in S$ are already available.

3. The average length[a] of the NAF of m_i is $h - \frac{1}{3}$ for $0 \leqslant m_i < 2^h$ and $i = 1, 2, \ldots, k$ (see § 6.2).

4. The average length[b] of an $k \times (h+1)$ array of binary signed digits is $\overline{L} = h + \frac{2}{3} - \frac{1}{2^k - 1}$, and thus the number of doublings is $h - \frac{1}{3} - \frac{1}{2^k - 1}$.

5. Let $\overline{d}$ denote the average number of all-zero columns between two nonzero columns, then $\overline{d} = \frac{(\frac{2}{3})^k}{1 - (\frac{2}{3})^k}$.

6. The number of elliptic additions except those for the stored points is

$$\frac{\overline{L} - 1}{1 + \overline{d}} = [1 - (\frac{2}{3})^k][h - \frac{1}{3} - \frac{1}{2^k - 1}].$$

□

8.2.2 General sliding window methods

Window method with double-and-add

If we view the modified Shamir's method as a sliding window algorithm with window

[a]The length of a number representation is equal to the number of the digits between the most significant nonzero digit and the least significant digit, inclusive.

[b]The length of an array is equal to the number of the columns between the leftmost nonzero column and the rightmost column, inclusive.

size $w = 1$, then further improvements may be explored using a window of size $w > 1$.

Let us write $m_1, m_2, \ldots, m_k$ in their NAFs and put them as an $k \times (h+1)$ array of binary signed-digits ($\{\bar{1}, 0, 1\}$), see Figure 8.1(a). Consider a window of size $k \times w$ horizontally sliding along the array from the leftmost end. Then the array can be split into *blocks* of size $k \times w$ whose leftmost column is not all zero. If we compute and store the points $l_1P_1 + l_2P_2 + \cdots + l_kP_k$, where $l_1, l_2, \ldots, l_k$ are all possible integers whose NAF is of length w or less, the required point can be computed with elliptic doublings and additions performed only on some stored points. The number of elliptic additions can be less than that in Shamir's method for $w > 1$ if the computation complexity of the stored points is comparatively low.

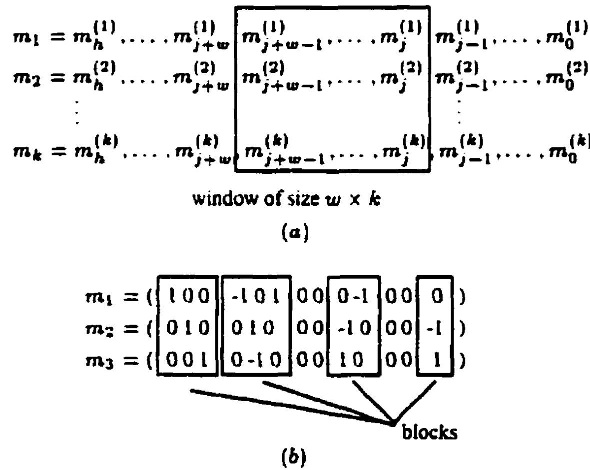

Figure 8.1: (a) A window of size w scanning along an $k \times (h+1)$ array of binary signed digits; (b) An example showing the blocks resulted from the scanning and splitting.

The number of stored points can be reduced if we allow the use of variable block widths, If blocks are of maximum width w, then the block of the smallest width $w' \leqslant w$ is used provided that it is still able to contain all the nonzero digits as a block of width w does. When such a block is chosen, its both rightmost and leftmost columns are nonzero columns. See Figure 8.1(b), where a window of size 3×3 used to scan and split the array into four blocks of size not greater than 3. The main steps in this method can be

illustrated in the following example.

Example 8.2 Compute $60454P_1 + 127453P_2$ using window size of two.

First 8 points are computed and stored that costs 6 elliptic additions:

$$P_1, P_2, P_1 \pm P_2, P_1 \pm 2P_2, 2P_1 \pm P_2.$$

While the two multiples are converted into the NAFs and they are placed as a $2 \times (h+1)$ array of binary signed digits. Then, a window of maximal length 2 is used to scan over the array and to split the array into a number of blocks. Each block is of width two or one and there is at least one nonzero digit in both the leftmost and the rightmost columns. The array is split into a number of blocks of maximal length 2 as shown below.

$$\begin{array}{r} 60454 = \\ 127453 = \end{array} \begin{bmatrix} & 1 \\ 1 & 0 \end{bmatrix} \begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array} \begin{bmatrix} 0 & \bar{1} \\ \bar{1} & 0 \end{bmatrix} \begin{array}{c} 0 \\ 0 \end{array} \begin{bmatrix} \bar{1} & 0 \\ 0 & 1 \end{bmatrix} \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \end{array} \begin{bmatrix} 1 \\ \bar{1} \end{bmatrix} \begin{array}{c} 0 \\ 0 \end{array} \begin{bmatrix} 1 & 0 \\ 0 & \bar{1} \end{bmatrix} \begin{bmatrix} \bar{1} & 0 \\ 0 & 1 \end{bmatrix}$$

For each block we can find the corresponding point in the storage. The solution can finally be obtained with

$$\begin{aligned} 60454P_1 + 127453P_2 &= 2^2 \cdot [2^3 \cdot [2^4 \cdot [2^3 \cdot [2^4 \cdot (P_1 + 2P_2) - (P_1 + 2P_2)] - (2P_1 - P_2)] + \\ &\quad (P_1 - P_2)] + (2P_1 - P_2)] - (2P_1 - P_2). \end{aligned}$$

It can be seen that five elliptic additions and sixteen doublings are required for the above equation. Then the total cost for computing $60454P_1 + 127453P_2$ is 11 elliptic additions and 16 doublings.

We summarize the general window method in the following algorithm.

Algorithm 8.2 Window method

Input: Integers m_i, w, and points P_i, $i = 1, 2, \ldots, k$.

Output: Point $P = m_1 P_1 + \cdots + m_k P_k$.

1 Convert m_i into a certain signed-digit (SD) form, and place them as a $k \times n$ array of binary signed digits.

2 Compute the points $l_1 P_1 + \cdots + l_k P_k$, $-Y \leqslant l_i \leqslant Y$, Y is the largest integer whose SD form is of length w. The first nonzero $l_i > 0$, and at least one l_i is odd (for the binary case).

3 Find the first block of size $\leqslant w$ from the leftmost end of the array. Find the corresponding point P_0 or its negative $-P_0$ among the stored points. Set $P \leftarrow P_0$ (or $P \leftarrow -P_0$); Set $L \leftarrow$ the length of the remaining array.

4 **Do while** $(L > 0)$ **Begin:**

4.1 Let the window of size w slide along the array rightward and find the next block if such a block exists. Let the stored point P' be the corresponding point of the block (or the negative of the corresponding point).

4.2 Set $P \leftarrow 2^d P$, where d denotes the distance between the rightmost column of the current block and that of the previous block. If there is no block to be found in Step 4.1, then let d be the length of the remaining array.

4.3 Set $P \leftarrow P + P'$ (or $P \leftarrow P - P'$).

4.4 Set $L \leftarrow L - d$.

End.

8.2.3 Window method with efficient computation of $2^l P$

Recently, it has been shown that under certain circumstances it is advantageous to directly compute $16P$, $8P$ and $4P$ instead of performing consecutive doublings [30, 61]. This idea can also be applied to the algorithms presented in this article by performing a $4P$, $8P$ or $16P$ operation whenever there are two, three or four consecutive doublings are required.

Algorithm 8.3 Window method with efficient computation of $2^l P$

Input: Integers m_i, w, and points P_i, $i = 1, 2, \ldots, k$.

Output: Point $P = m_1 P_1 + \cdots + m_k P_k$.

(Same as Algorithm 8.2 except Step 4.2 should be divided into 4 sub-steps:)

4.2.1 Set $P \leftarrow (16)^{t_1} P$, where $t_1 = \left\lfloor \frac{d}{4} \right\rfloor$;

4.2.2 Set $P \leftarrow (8)^{t_2} P$, where $t_2 = \left\lfloor \frac{d - 4t_1}{3} \right\rfloor$;

4.2.3 Set $P \leftarrow (4)^{t_3} P$, where $t_3 = \left\lfloor \frac{d - 4t_1 - 3t_2}{2} \right\rfloor$;

4.2.4 Set $P \leftarrow (2)^{t_4} P$, where $t_4 = d - 4t_1 - 3t_2 - 2t_3$.

8.2.4 Results and features

Some of the salient features of Algorithm 8.2 and 8.3 are presented in the following theorems.

Theorem 8.2 For $w \geqslant 1$, the number of stored points required in both Algorithms 8.2 and 8.3 is

$$N_{\text{store}} = \begin{cases} \sum_{k'=1}^{k} [Y(2Y+1)^{k'-1} - \frac{Y-1}{2} Y^{k'-1}] & \text{if } w \text{ is odd,} \\ \sum_{k'=1}^{k} [Y(2Y+1)^{k'-1} - \frac{Y}{2} (Y+1)^{k'-1}] & \text{if } w \text{ is even,} \end{cases}$$

where Y is the largest integer whose NAF is of length w, and $Y = \frac{1}{6}[2^{w+2}-3+(-1)^{w+1}]$.

Proof: For the computation of $m_1P_1 + \cdots + m_jP_j, j \geqslant 1$, and a certain window size $w \geqslant 1$, let the points to be stored be denoted by $l_1P_1 + \cdots + l_jP_j$, where l_i is of length not great than w in its NAF, and the number of points to be stored be denoted as $N_{\text{store}}^{(j)}$. (Note that $N_{\text{store}}^{(k)} = N_{\text{store}}$).

In the case of $j = k$: When w is even, the largest integer whose NAF is of length w is $Y = 2^{w-1} + 2^{w-3} + \cdots + 2$. Obviously there are $Y+1$ integers in $[0, Y]$ and $\frac{Y}{2}$ of them are odd numbers.

1. When l_1 is one of those $\frac{Y}{2}$ odd numbers, l_i can be any number in $[-Y, Y]$, for $i = 2, 3, \ldots, k$ for this case. Thus we have to store $\frac{Y}{2}(2Y+1)^{k-1}$ points.
2. When l_1 is nonzero even number in $[1, Y]$, l_i can be any number in $[-Y, Y]$ for $i = 2, 3, \ldots, k$, except at least one of l_i's is an odd number. Since there are $Y+1$ even numbers in $[-Y, Y]$, $\frac{Y}{2}[(2Y+1)^{k-1} - (Y+1)^{k-1}]$ points need to be stored.
3. When l_1 is zero, clearly, $N_{\text{store}}^{(k-1)}$ points should be computed and stored.

Thus we have

$$\begin{aligned} N_{\text{store}}^{(k)} &= \frac{Y}{2}(2Y+1)^{k-1} + \frac{Y}{2}[(2Y+1)^{k-1} - (Y+1)^{k-1}] + N_{\text{store}}^{(k-1)} \\ &= \sum_{k'=1}^{k}[Y(2Y+1)^{k'-1} - \frac{Y}{2}(Y+1)^{k'-1}] \end{aligned}$$

When w is odd, similar analysis can be applied. □

Theorem 8.3 For $h \gg w$ and $w \geqslant 1$, the average number of elliptic additions required in both Algorithms 8.2 and Algorithm 8.3 is

$$N_{\text{add}} = N_{\text{store}} - k + \frac{(h - \frac{1}{2^k-1})[1-(\frac{2}{3})^k]^2 - w[1-(\frac{2}{3})^k] + \frac{1}{3}[2-(\frac{2}{3})^k] + (\frac{2}{3})^{2k+1} - (\frac{2}{3})^{(w+1)k}}{w[1-(\frac{2}{3})^k] - (\frac{2}{3})^{2k} + (\frac{2}{3})^{(w+1)k}},$$

while the average number of doublings required in Algorithm 8.2 is

$$N_{2P} = h - \frac{w}{1-(\frac{2}{3})^k} + \frac{2}{3} - \frac{1}{2^k-1} + \frac{(\frac{2}{3})^k - (\frac{2}{3})^{(w+1)k}}{[1-(\frac{2}{3})^k]^2}.$$

A sketch of proof:

1. The number of elliptic additions required for the stored points is $N_{\text{store}} - k$.
2. The average length of an $k \times (h+1)$ array of binary signed digits is $\overline{L} = h + \frac{2}{3} - \frac{1}{2^k - 1}$.
3. The average length of blocks is

$$\overline{L}_b = \left[w - \frac{(\frac{2}{3})^k - (\frac{2}{3})^{(w+1)k}}{1-(\frac{2}{3})^k}\right] \Big/ \left[1 - \left(\frac{2}{3}\right)^k\right].$$

   The number of elliptic doublings is $\overline{L} - \overline{L}_b$.

4. The average number of all-zero columns between two nonzero columns is $\overline{d} = \frac{(\frac{2}{3})^k}{1-(\frac{2}{3})^k}$.
5. The number of elliptic additions except those for computing the stored points is $\frac{\overline{L} - \overline{L}_b}{\overline{L}_b + \overline{d}}$.

□

Theorem 8.4 The worst-case performance of Algorithm 8.2 for $h \gg w$ and $w \geqslant 1$ requires $N_{\text{store}} - k + \left\lceil \frac{h}{w} \right\rceil$ elliptic additions and h doublings. The number of elliptic additions needed in the worst case for Algorithm 8.3 is $N_{\text{store}} - k + \left\lceil \frac{h}{w} \right\rceil$.

Proof: By direct inspection. □

From Theorem 8.2, the optimal length of the window w_{opt} for the average performance of Algorithm 8.2 can be obtained by solving the equation

$$\frac{\partial(N_{\mathrm{add}} + \alpha N_{\mathrm{doubling}})}{\partial w} = 0$$

for w, where we assume that the complexity of an elliptic doubling is α times that of an elliptic addition. If the solution $w = w_{\mathrm{opt}}$ is not an integer, then we can try both $w = \lceil w_{\mathrm{opt}} \rceil$ and $w = \lfloor w_{\mathrm{opt}} \rfloor$, and choose the one with smaller $N_{\mathrm{add}} + \alpha N_{\mathrm{doubling}}$.

In Algorithm 8.3, if we further assume that the complexities of $4P$, $8P$, and $16P$ are β_1, β_2, and β_3 times that of a doubling, then the optimal value of window size w can be obtained by solving

$$\frac{\partial\{N_{\mathrm{add}} + \alpha(N_{2P} + \beta_1 N_{4P} + \beta_2 N_{8P} + \beta_3 N_{16P})\}}{\partial w} = 0,$$

where N_{2P}, N_{4P}, N_{8P}, and N_{16P} denote the numbers of operations of $2P, 4P, 8P$ and $16P$, respectively.

8.3 Algorithms using a New SD number Representation

8.3.1 A New SD representation with fewer zero runs

It is well known that a binary NAF of length n has the minimal number of nonzeros which is approximately $\frac{n}{3}$. However, Koyama and Tsuruoka have observed that the NAF is not necessarily the best representation to use for computing point multiples on an elliptic curve. It does have minimal weight, but allowing a few adjacent nonzeros may increase the length of zero runs, reducing the total number of elliptic additions. They also have given an algorithm for computing an SD representation with such features [46].

In the following, we will present a new SD number representation which is similar

to that of Koyama and Tsuruoka [46] in the sense that they both have a much reduced number of zero runs, compared to that of the NAF.

Given a bit string $A = [\cdots, a_i, a_{i-1}, \cdots, a_1, a_0]$, $a_i \in \{0,1\}$ for $i = 0,1,2,\cdots$, define a flag function $g(\cdot)$ over A as $g(0) = a_0$, and for $j > 0$ as

$$g(j) = \begin{cases} g(j-1)+1 & \text{if } a_j = 1; \\ g(j-1)-1 & \text{if } a_j = 0 \text{ and } g(j-1) > 0; \\ 0 & \text{if } a_j = 0 \text{ and } g(j-1) = 0; \\ 0 & \text{if } a_{j-1} \text{ is the rightmost bit in a convertible block.} \end{cases}$$

A *convertible block* $[a_{l_0+l-1} a_{l_0+l-2} \ldots a_{l_0}]$ *of length* l is defined by

1. $g(l_0 - 1) = 0$ or $l_0 = 0$;
2. $g(l_0) = 1$;
3. $a(j) + a(j+1) > 0$ for $l_0 \leqslant j \leqslant l_0 + l - 2$;
4. If $g(j) = 0$, then $g(j-1) \leqslant 2$ for $l_0 + 1 \leqslant j \leqslant l_0 + l - 2$;
5. $g(l_0 + l - 1) > 2$ and $a_{l_0+l} = 0$.

The new recoding algorithm takes a binary form as a bit string and the operation starts at the least significant end. The transformation from binary digits into signed digits is performed only on convertible blocks while the ordinary binary bits are retained outside the blocks. The transform operation applied to a convertible block is the Booth algorithm [16]. Note that the transformation is made block by block starting from the least significant end of the input since the result from transforming a convertible block could affect the calculation of the next convertible block.

Then the details of the transformation of an integer into the new SD form are shown in Algorithm 8.4. The function $g(\cdot)$ is denoted by `flag`. Lines 4 to 8 are to obtain the binary form of an integer while the Booth algorithm [16] is used in Lines 13 to 18. A

convertible block is found in Lines 24 to 32. In the sequel, the SD form with fewer zero runs obtained from Algorithm 8.4 is denoted as the FZR form.

Algorithm 8.4 Recoding into binary SD form with fewer zero runs (FZR form)

```
Input:  n:  Integer;
Output: S:  FZR form of n.
1. Begin:
2.    k1 <= n;  k2<= n;  i <= 0;  flag <= 0;  t <= 0;  st <= 0;
3.    do while k1>0 Begin:
4.       if k1 even
5.           then u <= 0;
6.           else u <= 1;
7.       k1 <= (k1-u)/2;
8.       S[i] <= u;
9.       if u=1
10.          then flag <= flag+1;
11.          else flag <= flag-1;
12.      if flag>=0
13.          then
14.              if k2 even
15.                  then v <= 0;
16.                  else v <= -1;
17.              k2 <= (k2-v)/2;
18.              T[i] <= v;
19.          else
20.              j from st to i do: Begin: T[j] <= S[j]; End;
21.              k2 <= k1;
22.              t <= 0;
23.              flag <= 0;
24.      if flag=1
25.          then
26.              if t=0 then st <= i;
27.              t <= 1;
28.          elseif flag>2 and k1 even
29.                  then
30.                      j from st to i do:
                             Begin: S[j] <= T[j]; End;
31.                      k1 <= k2;
32.                      flag <= 0;
```

```
33.     i <= i+1;
34.  End;
35.End.
```

□

	Binary form	Nonadjacent form	KT form [46]	Proposed form
	128 bits			
Average length	125.92	126.66	127.27	126.29
Average # of nonzeros	64.00	43.09	43.47	43.09
Average # of zero runs	31.16	42.06	34.15	33.96
Average zero run length	1.9911	1.9877	2.4488	2.4519
	256 bits			
Average length	253.85	254.65	255.31	254.25
Average # of nonzeros	128.00	85.75	86.40	85.75
Average # of zero runs	63.25	84.72	68.90	68.71
Average zero run length	1.9930	1.9941	2.4526	2.4542
	512 bits			
Average length	509.63	510.59	511.10	510.11
Average # of nonzeros	156.00	171.07	171.72	171.07
Average # of zero runs	127.17	170.17	138.17	137.98
Average zero run length	1.9979	1.9958	2.4583	2.4591

Table 8.1: Comparison of the various binary number representations

Table 8.1 shows some statistical parameters of the FZR form, along with those of the binary form, the NAF and the SD form obtained with the Koyama and Tsuruoka algorithm [46]. It can be seen from the table that the FZR form not only has the minimal number of nonzeros but also has an average length which is less than that of the NAF. One can also see that the FZR form is as good as the SD form obtained using the Koyama-Tsuruoka algorithm. This new form was conjectured having not only the minimal number of nonzero digits but also the fewest zero runs among all minimal binary SD representations. Later, Proos has found an example to show that the conjecture does not hold [65].

8.3.2 Window method with the FZR form

If the FZR form is used in Algorithm 8.2 and Algorithm 8.3, we have the following evaluation:

Evaluation 8.1 For $w \geqslant 1$, the number of stored points required in Algorithm 8.2 and Algorithm 8.3 using the FZR form is

$$N_{store} = \begin{cases} \sum_{k'=1}^{k} [Y(2Y+1)^{k'-1} - \frac{Y-1}{2} Y^{k'-1}] & \text{if } w \text{ is odd,} \\ \sum_{k'=1}^{k} [Y(2Y+1)^{k'-1} - \frac{Y}{2} (Y+1)^{k'-1}] & \text{if } w \text{ is even,} \end{cases}$$

where Y which is the largest number whose FZR form is of length w is given by $\frac{1}{6}[5 \cdot 2^w - 3 + (-1)^w]$.

Proof
From Algorithm 8.4, it can be seen that $Y = 2^{w-1} + 2^{w-2} + 2^{w-4} + \cdots + 1$ if w is even, and $Y = 2^{w-1} + 2^{w-2} + 2^{w-4} + \cdots + 2$ if w is odd. Thus we have $Y = \frac{1}{6}[5 \cdot 2^w - 3 + (-1)^w]$. The rest is similar to the proof of Evaluation 8.2.

8.4 Window Method for Koblitz Curves

Based on the results obtained by Merier and Staffelbach [53], Solinas [72] and Gordon [29], we have developed a new algorithm for computing $m_1P_1 + m_2P_2 + \cdots + m_kP_k$, where m_i is an integer and P_i is a point on a Koblitz curve. The algorithm is stated below.

Algorithm 8.5 Window method for Koblitz curves

Input: Integers m_i, w, and points P_i, $i = 1, 2, \ldots, k$.

Output: Point $P = m_1P_1 + \cdots + m_kP_k$.

1 Compute and store the points $l_1P_1 + \cdots + l_kP_k$, where $l_i = \langle l^{(i)}_{w-1} l^{(i)}_{w-2} \cdots l^{(i)}_0 \rangle$ are all possible τ-adic NAFs of length not greater than w, $l^{(i)}_j \in \{-1, 0, 1\}$ for $i = 1, \ldots, k$, and $j = 0, 1, \ldots, w-1$, and the nonzero l_i with smallest i has a positive τ-adic NAF and at least one of the τ-adic NAFs is odd ($l^{(j)}_0 \neq 0$).

2 Use the method presented in Theorem 3, 4 and 5 in [29] to obtain the τ-adic NAFs for $m_i = \sum_{j=0}^{h} e^{(i)}_j \tau^j (\bmod\ \tau^n - 1)$ and $e^{(i)}_j \in \{-1, 0, 1\}$ for $i = 1, \ldots, k$ and $j = 0, 1, \ldots, h$, where the elliptic curve is defined over $\mathbb{F}_{2^n}$, and, $h = n + 1$ as it is shown in Theorem 5 in [29].

3 Set $L \leftarrow 0$ and $P \leftarrow \mathcal{O}$;

4 **Do while ($L \leqslant h$) Begin:**

4.1 Let the window of size w scan leftward along the array to locate the next block and its corresponding point in the storage be P' (or $-P'$). If no block is found then exit the iteration and stop the program.

4.2 Set $P \leftarrow P + \tau^d P'$ (or $P \leftarrow P - \tau^d P'$), where d is the distance between the rightmost columns of the current block and that of the previous block.

4.3 Set $L \leftarrow L + d$;

End.

Some features of Algorithm 8.5 can be summarized in the following theorems.

Theorem 8.5 For $w \geqslant 1$, the number of stored points required in Algorithm 8.5 is

$$N_{\text{store}} = \begin{cases} \sum_{k'=1}^{k} [Y(2Y+1)^{k'-1} - \frac{Y-1}{2} Y^{k'-1}] & \text{if } w \text{ is odd,} \\ \sum_{k'=1}^{k} [Y(2Y+1)^{k'-1} - \frac{Y}{2} (Y+1)^{k'-1}] & \text{if } w \text{ is even,} \end{cases}$$

where Y is the largest number whose NAF is of length w, and $Y = \frac{1}{6}[2^{w+2} - 3 + (-1)^{w+1}]$.

A proof can be similar to that of Theorem 8.2.

Theorem 8.6 For $h \gg w$ and $w \geqslant 1$, on average Algorithm 8.5 requires

$$N_{\text{add}} = N_{\text{store}} - k + \frac{(h - \frac{1}{2^k - 1})[1 - (\frac{2}{3})^k]^2 - w[1 - (\frac{2}{3})^k] + \frac{1}{3}[2 - (\frac{2}{3})^k] + (\frac{2}{3})^{2k+1} - (\frac{2}{3})^{(w+1)k}}{w[1 - (\frac{2}{3})^k] - (\frac{2}{3})^{2k} + (\frac{2}{3})^{(w+1)k}},$$

elliptic additions. Here we assume that the statistical properties of the NAF of m_i and those of the τ-adic NAF of m_i are the same except that the τ-adic NAF is longer than the NAF by one digit on average.

A proof can be similar to that of Theorem 8.3. Note that in Theorem 8.6 we do not include the cost of some up-front calculations required in Step 2 of Algorithm 8.5, which can be estimated to be a couple n-bit binary multiplications and divisions and is negligible compared to the cost of elliptic addition and doubling [53, 29].

Theorem 8.7 The worst-case performance of Algorithm 8.5 for $h \gg w$ and $w \geqslant 1$ requires $N_{\text{store}} - k + \left\lceil \frac{h}{w} \right\rceil$ elliptic additions.

Proof: By direct inspection. □

8.5 Numerical Results

In this section, we give some numerical comparisons for the proposed algorithms. The underlying field is chosen to be $\mathbb{F}_{2^{163}}$ which has been considered for implementing elliptic curve cryptosystems [1].

8.5.1 Comparison of computing mP

Algorithms	$n = 163$					
	N_{add}	N_{2P}	N_{store}	N_{add}	N_{2P}	N_{store}
	$w = 2$			$w = 3$		
Algorithm 8.2 (NAF)	53.76	161.70	1	37.65	160.67	3
Algorithm 8.2 (FZR)	44.52	161.09	2	38.35	160.68	3
Algorithm 8.5 (τ-adic NAF)	54.10	0	1	37.88	0	3
	$w = 4$			$w = 5$		
Algorithm 8.2 (NAF)	34.49	159.93	5	35.02	158.94	11
Algorithm 8.2 (FZR)	34.96	159.80	7	36.66	158.81	13
Algorithm 8.5 (τ-adic NAF)	34.68	0	5	35.17	0	11
	$w = 6$			$w = 7$		
Algorithm 8.2 (NAF)	41.74	158.01	21	70.98	157.01	53
Algorithm 8.2 (FZR)	47.19	157.84	27	120.94	156.84	105
Algorithm 8.5 (τ-adic NAF)	41.88	0	21	71.09	0	53

Table 8.2: Comparison of the algorithms for computing mP.

From the data shown in Table 8.2, we can see that if the number of elliptic operations is all what we are concerned of, then Algorithm 8.2 using the NAF and Algorithm 8.5 with window size of 4 or 5 may be the best options for computing mP on a non-supersingular, or on a Koblitz curve in $\mathbb{F}_{2^{163}}$, respectively. When the memory size for the stored points is also an issue of consideration or the window size is small, we may choose Algorithm 8.2 using the FZR form to compute mP on a non-supersingular curve in $\mathbb{F}_{2^{163}}$.

When the point P is fixed, then all the possible point multiples, whose corresponding binary SD form is within a window of size w, can be precomputed and stored. Conse-

Algorithms	$n = 163$					
	N_{add}	N_{2P}	N_{store}	N_{add}	N_{2P}	N_{store}
	$w = 2$			$w = 3$		
Algorithm 8.2 (NAF)	53.76	161.70	1	35.65	160.67	3
Algorithm 8.2 (FZR)	43.52	161.09	2	36.35	160.68	3
Algorithm 8.5 (τ-adic NAF)	54.10	0	1	35.88	0	3
	$w = 4$			$w = 5$		
Algorithm 8.2 (NAF)	30.49	159.93	5	25.02	158.94	11
Algorithm 8.2 (FZR)	28.96	159.80	7	24.66	158.81	13
Algorithm 8.5 (τ-adic NAF)	30.68	0	5	25.17	0	11
	$w = 6$			$w = 7$		
Algorithm 8.2 (NAF)	21.74	158.01	21	18.98	157.01	53
Algorithm 8.2 (FZR)	21.19	157.84	27	16.94	156.84	105
Algorithm 8.5 (τ-adic NAF)	21.88	0	21	19.09	0	53

Table 8.3: Comparison of the algorithms for computing mP (P fixed).

quently, the computational cost for computing mP may exclude the precomputations. From Table 8.3, it can be seen that it is advantageous to use Algorithm 8.2 with the FZR form to compute mP for the cases when the point P is fixed.

Instead of performing consecutive doublings, when the operations like $16P$, $8P$, and $4P$ are utilized, the efficiency of the Algorithm 8.3 is shown in Table 8.4. It can be seen that the algorithm using the FZR form sometimes has more $16P$ and $8P$ operations than using the NAF.

8.5.2 Comparison of computing $mP_1 + rP_2$

The best option for computing $mP_1 + rP_2$ is to use Algorithm 8.2 with the NAF for non-supersingular curve and Algorithm 8.5 for Koblitz curve with window size of 2, when the curve is defined in $\mathbb{F}_{2^{163}}$. additions is

If either P_1 or P_2 is fixed, say, P_1 is fixed, which is also the case practically used

Algorithms	$n = 163$					
	N_{add}	N_{16P}	N_{8P}	N_{4P}	N_{2P}	N_{store}
	$w = 2$					
Algorithm 8.3 (NAF)	53.76	14.22	14.38	28.94	3.81	1
Algorithm 8.3 (FZR)	44.52	20.56	14.97	13.96	6.02	2
	$w = 3$					
Algorithm 8.3 (NAF)	37.65	26.56	9.07	9.06	9.12	3
Algorithm 8.3 (FZR)	38.35	25.76	10.94	7.38	10.05	3
	$w = 4$					
Algorithm 8.3 (NAF)	34.49	29.74	5.27	8.73	7.68	5
Algorithm 8.3 (FZR)	34.96	28.57	7.72	7.33	7.68	7
	$w = 5$					
Algorithm 8.3 (NAF)	35.02	29.75	5.90	7.92	6.39	11
Algorithm 8.3 (FZR)	36.66	30.06	6.37	6.36	6.72	13
	$w = 6$					
Algorithm 8.3 (NAF)	41.74	30.53	6.33	6.21	4.47	21
Algorithm 8.3 (FZR)	47.19	31.11	5.86	5.28	5.26	27
	$w = 7$					
Algorithm 8.3 (NAF)	70.98	32.32	4.80	4.34	4.63	53
Algorithm 8.3 (FZR)	120.94	32.15	4.66	4.57	5.13	105

Table 8.4: Comparison of the numbers of $16P$, $8P$, $4P$ and $2P$ operations required to compute mP for non-supersingular curves.

in the Elliptic Curve Digital Signature Algorithm [56], then all the P_1 multiples whose binary SD form has length within w can be precomputed. In this case the data are given in Table 8.6, and it is still Algorithm 8.2 using the NAF with a window of size 2 which yields the best results for non-supersingular curves.

If both P_1 and P_2 are fixed, then Algorithm 8.2 using the NAF with window of size 3 and using the FZR form with window of size 2 seem to be good choices for computing $mP_1 + rP_2$ on a non-supersingular curve (see Table 8.7). Also from Tables 8.5, 8.6 and 8.7, the computation of $mP_1 + rP_2$ on a Koblitz curve, no matter whether the points are fixed or not, instead of using Algorithm 8.5 with $k = 2$, it is more advantageous to

Algorithms	$n = 163$					
	N_{add}	N_{2P}	N_{store}	N_{add}	N_{2P}	N_{store}
	$w = 1$			$w = 2$		
Algorithm 8.2 (NAF)	92.22	162.42	4	72.01	162.05	8
Algorithm 8.2 (FZR)	92.19	162.08	4	76.09	161.51	20
Algorithm 8.5 (τ-adic NAF)	92.77	0	4	72.43	0	8
	$w = 3$			$w = 4$		
Algorithm 8.2 (NAF)	90.14	160.89	48	193.31	160.03	160
Algorithm 8.2 (FZR)	102.27	160.80	60	312.41	159.77	280
Algorithm 8.5 (τ-adic NAF)	90.42	0	48	193.53	0	160

Table 8.5: Comparison of algorithms for computing $mP_1 + rP_2$

compute mP_1 and rP_2 separately using Algorithm 8.5 with $k = 1$ and then add them up.

When the operations like $16P$, $8P$, and $4P$ can be performed more efficiently than by consecutive doublings, it can be seen from Table 8.8 that with the FZR form Algorithm 8.3 has significantly more operations like $16P$ and $8P$ for $w = 1$ and $w = 2$, while for $w = 4$ the NAF has better performance.

It is also worth noting that when n is large, say, $n \geqslant 400$, the algorithm using the FZR form for computing both mP_1 and $mP_1 + rP_2$ perform significantly better than that using the NAF.

For computing $m_1P_1 + \cdots + m_kP_k, k \geqslant 3$, it is common to choose $w = 1$ since the cases of $w \geqslant 2$ often require large memory to store precomputed points. For example, when $k = 3$, Algorithm 8.2 using the NAF requires 13 stored points for $w = 1$ and 49 s tored points for $w = 2$, and when $k = 4$, this algorithm requires 40 stored points for $w = 1$ and even much more stored points for $w > 1$.

8.6 Chapter Summary

In this chapter, the general sliding window method and its performance analysis are presented for computing $m_1P_1 + m_2P_2 + \cdots + m_kP_k$ for nonsupersingular and Koblitz

Algorithms	$n = 163$					
	N_{add}	N_{2P}	N_{store}	N_{add}	N_{2P}	N_{store}
	$w = 1$			$w = 2$		
Algorithm 8.2 (NAF)	92.22	162.42	4	72.01	162.05	8
Algorithm 8.2 (FZR)	92.19	162.08	4	75.09	161.51	20
Algorithm 8.5 (τ-adic NAF)	92.77	0	4	72.43	0	8
	$w = 3$			$w = 4$		
Algorithm 8.2 (NAF)	86.14	160.89	48	185.31	160.03	160
Algorithm 8.2 (FZR)	98.27	160.80	60	300.41	159.77	280
Algorithm 8.5 (τ-adic NAF)	86.42	0	48	185.53	0	160

Table 8.6: Comparison of the algorithms for computing $mP_1 + rP_2$ (P_1 fixed).

curves. For computing a point multiple on an EC over a certain field, the numerical results have given a hint on how to choose a recoded SD representation and window parameters for efficient computations.

Algorithms	$n = 163$					
	N_{add}	N_{2P}	N_{store}	N_{add}	N_{2P}	N_{store}
	$w = 1$			$w = 2$		
Algorithm 8.2 (NAF)	90.22	162.42	4	66.01	162.05	8
Algorithm 8.2 (FZR)	90.19	162.08	4	59.09	161.51	20
Algorithm 8.5 (τ-adic NAF)	90.77	0	4	66.43	0	8
	$w = 3$			$w = 4$		
Algorithm 8.2 (NAF)	44.14	160.89	48	35.31	160.03	160
Algorithm 8.2 (FZR)	44.27	160.80	60	34.41	159.77	280
Algorithm 8.5 (τ-adic NAF)	44.42	0	48	35.53	0	160

Table 8.7: Comparison of algorithms for computing $mP_1 + rP_2$ (both P_1 and P_2 fixed).

Algorithms	$n = 256$					
	N_{add}	N_{16P}	N_{8P}	N_{4P}	N_{2P}	N_{store}
	$w = 1$					
Algorithm 8.3 (NAF)	92.22	3.35	10.14	40.78	37.04	4
Algorithm 8.3 (FZR)	92.19	5.23	13.08	28.50	44.93	4
	$w = 2$					
Algorithm 8.3 (NAF)	72.01	6.71	20.24	33.51	7.48	8
Algorithm 8.3 (FZR)	76.09	12.41	21.08	19.13	10.38	20
	$w = 3$					
Algorithm 8.3 (NAF)	90.14	24.74	12.34	8.05	8.85	48
Algorithm 8.3 (FZR)	102.27	23.57	13.23	7.81	11.24	60
	$w = 4$					
Algorithm 8.3 (NAF)	193.31	30.13	5.40	5.51	12.29	160
Algorithm 8.3 (FZR)	312.41	28.36	6.99	7.51	10.34	280

Table 8.8: Comparison of the numbers of $16P$, $8P$, $4P$ and $2P$ operations required to compute $mP_1 + rP_2$ for non-supersingular curves.

Chapter 9

Summary, Discussions and Future Work

Computations in finite fields play an important role in cryptography, coding theory, sequence generation, signal processing and VLSI testing. In this thesis, a number of efficient algorithms and architectures for finite field multiplication have been presented. Efficient realizations of finite field exponentiation and point multiples on an elliptic curve have also been proposed.

A normal basis is commonly used in many cryptographic systems, since a squaring operation using the normal basis is simply a cyclic shift. This can potentially simplify exponentiation, elliptic addition and doubling, and Frobenius mapping. Since the invention of the Massey-Omura multiplier, a few alternative normal basis multipliers have been proposed, *e.g.*, [22, 3]. The redundant basis presented in this thesis takes advantage of the elegant multiplicative structure of the set of $(mk+1)^{st}$ roots of unity over $\mathbb{F}_q$ that includes a basis of $\mathbb{F}_{q^m}$. The resultant multiplier architectures using redundant basis are extremely simple and also have a lower complexity when $k = 1$ and 2. Further work on this topic might include the investigation of low complexity multipliers when k has a

small value but greater than 2. It will also be interesting to study efficient representation of the field elements when the normal basis is generated by the general Gauss period.

Weakly dual basis multiplication architectures have also been considered in this thesis. For the classes of finite fields generated with an irreducible trinomial or an irreducible ESP, low complexity bit-parallel multipliers have been presented. Basis conversion has also been discussed such that we have given its complexity bound when the field is generated with an ESP or a polynomial of form $f(x) = x^m + x^{k+d} + x^{k+d-1} + \cdots + x^k + 1$. Future work might include the investigation of multiplier architectures with reduced time delay when the field is generated with an irreducible pentanomial or a polynomial of arbitrary form. Multipliers using normal dual bases might also be worthy to further study.

A polynomial basis probably has been most commonly used in various applications. In this thesis, we have given a size complexity bound for a polynomial basis multiplier in an arbitrary finite field. When the field is of characteristic 2 and generated with an irreducible trinomial, both the size complexities and time delays of the multiplier and the squarer are analyzed. Given the irreducible trinomial, we can easily build a multiplier or a squarer conforming to the complexity parameters following the steps in the proofs of the theorems. However, general diagrams of the multiplier and the squarer architectures have not been available because they seem quite complicated and could be considered for further investigation. Future work should also include study of using the FFT and the KOA methods and seeking the possibility to combine these methods with other techniques to yield more efficient architectures for finite field computations.

Finite field exponentiation and elliptic curve operation have received considerable attention recently for their uses in cryptography. To efficiently perform an exponentiation operation, on the one hand, fast multiplication and squaring must be provided; on the other hand, efficient representation of the exponent should be investigated. Precomputation can be done if certain information about the base and/or the exponent is known before hand, and consequently, speed and memory size trade-offs can be made to obtain the maximum efficiency.

In this thesis, architectures for exponentiation of a primitive element have been proposed. LFSR-style structures are presented to realize multiplication operation, while squaring or power operation is performed in a bit-parallel module. On the other hand, minimal weight signed-digit forms have been utilized to efficiently represent the exponent. For computation of point multiples for elliptic curve cryptosystems, algorithms have been proposed for efficient representation of the multiples. Future work in this area should include research on efficient computation of elliptic point operation. Investigation can be carried out for the design of a finite field processor especially for performing elliptic point operations, which can coordinate the data flows between its sub-modules, such as the inverter and the multiplier.

Bibliography

[1] http://www.certicom.com/ecc/.

[2] V. B. Afanasyev. On the complexity of finite field arithmetic. In *Proc 5th Joint Soviet-Swedish Intern. Workshop on Information Theory*, pages 9–12, 1991. Moscow, USSR.

[3] G. B. Agnew, R. C. Mullin, I. Onyszchuk, and S. A. Vanstone. An implementation for a fast public key cryptosystem. *J. Cryptology*, 3:63–79, 1991.

[4] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. Fast exponentiation in $GF(2^m)$. In *Eurocrypt'88*. Springer-Verlag, 1989.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publ. Co., Reading, MA, 1974.

[6] S. Arno and F. S. Wheeler. Signed digit representation of minimum Hamming weight. *IEEE Trans. Comput.*, 42:1007–1010, 1993.

[7] D. W. Ash, I. F. Blake, and S. A. Vanstone. Low complexity normal bases. *Disc. Appl. Math.*, 25:191–210, 1989.

[8] D. E. Atkins. Design of the arithmetic units of Illiac III: Use of redundancy and higher radix methods. *IEEE Trans. Comput.*, 9:720–733, 1970.

[9] E. Bach and J. Shallit. *Algorithmic Number Theory*. MIT Press, 1996.

[10] T. C. Bartee and D. I. Schneider. Computation with finite fields. *Inform. and Comput.*, 6:79–98, 1963.

[11] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill Book Company, New York, 1968.

[12] E. R. Berlekamp. Bit-serial Reed-Solomon encoders. *IEEE Trans. IT*, 28:869–974, 1982.

[13] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley Pub. Co., Reading, Mass., 1983.

[14] R. E. Blahut. *Algebraic Methods for Signal Processing and Communications Coding*. Springer-Verlag, New York, 1992.

[15] I. F. Blake, S. Gao, and R. Lambert. Constructive problems for irreducible polynomials over finite fields. In *Canadian Workshop on Information Theory*. Springer-Verlag, 1993.

[16] A. D. Booth. A signed binary multiplication technique. *Quart. J. Mech. Appl. Math.*, 4:236–240, 1951.

[17] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation (extended abstract). In *EUROCRYPT'92*, pages 200–207. Springer-Verlag, 1992.

[18] W. E. Clark and J. J. Liang. On arithmetic weight for a general radix representation of integers. *IEEE Trans. IT*, 19:823–826, 1973.

[19] S. D. Cohen. On irreducible polynomials of certain types in finite fields. *Proc. Camb. Phil. Soc.*, 66:335–344, 1969.

[20] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. IT*, 22:644–654, 1976.

[21] S. Feisel, J. von zur Gathen, and M. A. Shokrollahi. Normal bases via general Guass periods. *Math. Comp.*, 1998. To appear.

[22] M. Feng. A VLSI architecture for fast inversion in GF(2^m). *IEEE Trans. Comput.*, 38:1383–1386, 1989.

[23] S. T. J. Fenn, M. Benaissa, and D. Taylor. GF(2^m) multiplication and division over the dual basis. *IEEE Trans. Comput.*, 45(3):319–327, 1996.

[24] S. Gao. *Normal Bases over Finite Fields.* PhD thesis, University of Waterloo, Waterloo,Ontario,Canada, 1993.

[25] S. Gao. Gauss periods, groups, and normal bases. Preprint, August 1997.

[26] S. Gao and S. Vanstone. On orders of optimal normal basis generators. *Math. Comp.*, 64(2):1227–1233, 1995.

[27] S. Gao, von zur Gathen, and D. Panario. Gauss periods and fast exponentiation in finite fields. *Lecture Notes in Computer Science*, 911:311–322, 1995.

[28] H. L. Garner. Number systems and arithmetic. In *Advan. Computers 6*, pages 131–194. Academic Press, 1965.

[29] D. M. Gordon. A survey of fast exponentiation methods. Preprint, August 1996.

[30] J. Guajardo and C. Paar. Efficient algorithms for elliptic curve cryptosystems. In *CRYPT97*, pages 342–356. Springer-Verlag, 1997.

[31] T. A. Gulliver, M. Serra, and V. K. Bhargava. The generation of primitive polynomials in GF(q) with independent roots and their applications for power residue codes, VLSI testing and finite field multipliers using normal basis. *Int. J. Electronics*, 71:559–576, 1991.

[32] G. Harper, A. Menezes, and S. Vanstone. Public-key cryptosystems with very small key lengths. In *Advances in Cryptology -Crypto'93*, pages 163–173. Springer-Verlag, 1993.

[33] M. A. Hasan. *Efficient Computations over Galois Fields*. PhD thesis, University of Victoria, Victoria, B.C., Canada, 1992.

[34] M. A. Hasan and V. K. Bhargava. Architecture for a low complexity rate adaptive Reed-Solomon encoder. *IEEE Trans. Comput.*, 44:938–942, 1995.

[35] M. A. Hasan, M. Wang, and V. K. Bhargava. Modular construction of low complexity parallel multipliers for a class of finite fields GF(2^m). *IEEE Trans. Comput.*, 41(8):962–971, 1992.

[36] M. A. Hasan, M. Wang, and V. K. Bhargava. A modified Massey-Omura parallel multiplier for a class of finite fields. *IEEE Trans. Comput.*, 42(8):1278–1280, 1993.

[37] A. Hocquenghem. Codes corecteurs d'erreurs. *Chiffres*, 2:147–156, 1959.

[38] T. Itoh and S. Tsujii. Structure of parallel multipliers for a class of fields GF(2^m). *Inform. and Comput.*, 83:21–40, 1989.

[39] Y. Jeong. *VLSI Algorithms and Architectures for Real-time Computation over Finite Fields*. PhD thesis, University of Massachusetts Amherst, Amherst, Massachusetts, USA, 1995.

[40] D. Jungnickel. *Finite Fields: Structure and Arithmetics*. B. I. Wissenschaftsverlag, Mannheim, Germany, 1993.

[41] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Sov. Phys.-Dokl. (English translation)*, 7(7):595–596, 1963.

[42] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley Publishing Company, Reading, MA, 1981.

[43] N. Koblitz. CM-curves with good cryptographic properties. In *CRYPTO91*, pages 279–287. Springer-Verlag, 1992.

[44] C. K. Koc and T. Acar. Montgomery multiplication in GF(2^k). *Designs, Codes and Cryptography*, 14:57–69, 1998.

[45] C. K. Koc and B. Sunar. Low-complexity bit-parallel canonical and normal multipliers for a class of finite fields. *IEEE Trans. Comput.*, 47(3):353–256, 1998.

[46] K. Koyama and Y. Tsuruoka. Speeding up elliptic cryptosystems by using a signed binary window method. In *CRYPTO93*, pages 345–357. Springer-Verlag, 1993.

[47] R. Lidl and H. Niederreiter. *Finite Fields*. Addison-Wesley Publishing Company, Reading, MA, 1983.

[48] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology -Crypto'94*, pages 95–107. Springer-Verlag, 1994.

[49] J. L. Massey and J. K. Omura. Computational method and apparatus for finite field arithmetic. U.S. Patent No.4587627, 1984.

[50] E. D. Mastrovito. *VLSI Architectures for Computations in Galois Fields*. PhD thesis, Linköping University, Linköping, Sweden, 1991.

[51] D. W. Matula. Basic digit sets for radix representation. *J. of the ACM*, 29:1131–1143, 1982.

[52] R. J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 1987.

[53] W. Meier and O. Staffelbach. Efficient multiplication on certain nonsupersingular elliptic curves. In *CRYPTO'92*, pages 333–344. Springer-Verlag, 1993.

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

[55] 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.

[56] A. J. Menezes, M. Qu, and S. A. Vanstone. Working draft: IEEE P1363 standard. 1995.

[57] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

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

[59] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theoretical Informatics and Applications*, pages 531–543, 1990.

[60] M. Morii, M. Kasahara, and D. L. Whiting. Efficient bit-serial multiplication and discrete-time Wiener-Hoph equation over finite fields. *IEEE Trans. IT*, 35:1177–1184, 1989.

[61] V. Muller. A survey of fast exponentiation methods. Preprint, August 1997.

[62] R. Mullin, I. Onyszchuk, S. A. Vanstone, and R. Wilson. Optimal normal bases in GF(p^n). *Disc. Appl. Math.*, 22:149–161, 1988.

[63] C. Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*. VDI-Verlag, Düsseldorf, 1994. Ph.D Thesis.

[64] C. Paar and N. Lange. A comparative VLSI synthesis of finite field multipliers. In *Proc. 3rd Intern. Symp. Comm. Theory and Its Applications*, 1995. Lake District, UK.

[65] J. Proos. Private communication, 1998.

[66] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Ind. Appl. Math.*, 8:300–304, 1960.

[67] G. W. Reitwiesner. Binary arithmetic. In *Advan. Computers 1*, pages 232–308. Academic Press, 1960.

[68] A. Schönhage. Schnelle Multiplikation von Polynomen uber Korpern der Charakteristik 2. *Acta Inf.*, 7:395–398, 1977.

[69] P. A. Scott, S. J. Simmons, S. E. Tavares, and L. E. Peppard. Architectures for exponentiation in GF(2^m). *IEEE J. Selected Areas in Comm.*, 6:578–586, 1988.

[70] P. A. Scott, S. E. Tavares, and L. E. Peppard. A fast VLSI multiplier for GF(2^m). *IEEE J. Selected Areas in Comm.*, 4:62–66, 1986.

[71] G. Seroussi. Table of low-weight binary irreducible polynomials. Technical Report HPL-98-135, Hewlett-Packard Laboratories, Palo Alto, CA, August 1998.

[72] J. A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In *CRYPT'97*, pages 357–371. Springer-Verlag, 1997.

[73] D. R. Stinson. *Cryptography: Theory and Practice*. CRC Press, Inc., Boca Raton, Florida, 1995.

[74] C. C. Wang. *Exponentiation in Finite Fields*. PhD thesis, University of California at Los Angeles, Los Angeles, CA, USA, 1985.

[75] C. C. Wang. An algorithm to design finite field multipliers using a self-dual normal basis. *IEEE Trans. Comput.*, 38(10):1457–1459, October 1989.

[76] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed. VLSI architectures for computing multiplications and inverses in GF(2^m). *IEEE Trans. Comput.*, 34(8):709–717, 1985.

[77] M. Wang and I. F. Blake. Bit serial multiplication in finite fields. *SIAM Discrete Mathematics*, 3(1):140–148, 1990.

[78] A. Wassermann. Konstruktion von Normalbasen. *Bayreuther Mathematische Schriften*, pages 155–164, 1990.

[79] H. Wu and M. A. Hasan. Exponentiation using dual basis. In *Proceedings of 18th Biennial Communication Symposium*, pages 204–207, 1996. Kingston, Canada.

[80] H. Wu and M. A. Hasan. Efficient exponentiation of a primitive root in GF(2^m). *IEEE Trans. Comput.*, 46(2):162–172, Feburary 1997.

[81] H. Wu and M. A. Hasan. Low complexity bit-parallel multipliers for a class of finite fields. *IEEE Trans. Comput.*, 47(8):883–887, August 1998.

[82] H. Wu, M. A. Hasan, and I. F. Blake. Low complexity weakly dual basis bit-parallel multiplier over finite fields. *IEEE Trans. Comput.*, November 1998. scheduled to appear.

[83] C.-S. Yeh, I. S. Reed, and T. K. Truong. Systolic multipliers for finite fields GF(2^m). *IEEE Trans. Comput.*, 33:357–360, 1984.

[84] S-M. Yen and C-S. Laih. The fast cascade exponentiation algorithm and its applications on cryptography. In *ASIACRYPT'92*, pages 447–456. Springer-Verlag, 1992.