University of Windsor
Scholarship at UWindsor

Electronic Theses and Dissertations | Theses, Dissertations, and Major Papers

2004

Bit-parallel word-serial polynomial basis finite field multiplier in GF(2(233)).

Wenkai Tang
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
Tang, Wenkai, "Bit-parallel word-serial polynomial basis finite field multiplier in GF(2(233))." (2004). *Electronic Theses and Dissertations*. 1310.
https://scholar.uwindsor.ca/etd/1310

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Bit-Parallel Word-Serial Polynomial Basis Finite Field Multiplier in $GF(2^{233})$

by

Wenkai Tang

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the Department of Electrical and Computer Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario, Canada
May, 2004

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services

Acquisisitons et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-92454-8
Our file Notre référence
ISBN: 0-612-92454-8

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 aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

© May, 2004 Wenkai Tang

All Rights Reserved. No Part of this document may be reproduced, stored or otherwise retained in a retreival system or transmitted in any form, on any medium by any means without prior written permission of the author.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

Smart card gains extensive uses as a cryptographic hardware in security applications in daily life. The characteristics of smart card require that the cryptographic hardware inside the smart card have the trade-off between area and speed.

There are two main public key cryptosystems, these are RSA cryptosystem and elliptic curve (EC) cryptosystem. EC has many advantages compared with RSA such as shorter key length and more suitable for VLSI implementation. Such advantages make EC an ideal candidate for smart card.

Finite field multiplier is the key component in EC hardware. In this thesis, bit-parallel word-serial (BPWS) polynomial basis (PB) finite field multipliers are designed. Such architectures trade-off area with speed and are very useful for smart card.

An ASIC chip which can perform finite field multiplication and finite field squaring using the BPWS PB finite field multiplier is designed in this thesis. The proposed circuit has been implemented using TSMC 0.18 CMOS technology.

A novel 8×233 bit-parallel partial product generator is also designed. This new partial product generator has low circuit complexity. The design algorithm can be easily extended to $w \times m$ bit-parallel partial product generator for $GF(2^m)$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To my parents for their constant encouragement, my wife for her support and my daughter Xina.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

I would like to express my sincere gratitude to my graduate supervisors Dr. Huapeng Wu and Dr. Majid Ahmadi for their constant support, guidance and motivation. I am also grateful to the committee members Dr. Kemal E. Tepe and Dr. Jessica Chen for providing valuable feedback at all times.

I would like to thank Till Kuendiger for helpful support on the utilization of VLSI CAD tools. I would also like to thank Minyi Fu, Bijan Ansari, Zheng Li and Zhong Zheng for helpful discussions and suggestions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Contents

Abstract **iv**

Dedication **v**

Acknowledgements **vi**

List of Figures **viii**

List of Tables **x**

Abbreviations **xi**

1 Introduction **1**

1.1 Research motivations 1

1.1.1 Smart card and its applications 1

1.1.2 Cryptography and cryptosystems 3

1.1.3 Elliptic curve cryptography (ECC) 4

1.2 Research goals 6

1.3 Thesis organization 6

2 Arithmetic over Finite Field **7**

2.1 Group, ring, field and finite field 7

2.1.1 Group 7

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1.2 Ring . . . 8
2.1.3 Field . . . 9
2.1.4 Finite field . . . 10
2.2 Finite field element representations . . . 11
2.2.1 Polynomial basis . . . 12
2.3 Finite field operation . . . 12
2.3.1 Addition . . . 12
2.3.2 Multiplication . . . 13
2.3.3 Comparisons among the multiplications with different basis . . 14
2.4 Galois type linear feedback shift register (LFSR) . . . 15
2.5 Elliptic curve . . . 15
2.6 Polynomial basis (PB) finite field multipliers . . . 17
2.6.1 Bit-parallel PB finite field multipliers . . . 18
2.6.2 Bit-serial PB finite field multipliers . . . 19
2.6.3 Bit-parallel PB finite field squarer . . . 21
2.7 Summary . . . 22

3 Design of Bit-Parallel Word-Serial PB Finite Field Multipliers 25
3.1 NIST recommendations . . . 26
3.2 Design of BPWS PB finite field Multiplier . . . 27
3.2.1 Multiplication algorithm . . . 27
3.2.2 Bit-parallel word-serial multiplier architecture . . . 28
3.2.3 M3: Constant Finite Field Multiplier $Z = x^8 Y$ . . . 31
3.2.4 M1: 8×233 Bit-parallel partial product generator . . . 32
3.3 Alternative BPWS PB finite field multiplier . . . 36
3.4 General BPWS PB finite field multipliers . . . 40
3.5 Comparisons . . . 41
3.6 Summary . . . 44

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4 Hardware Design **45**

4.1 Hardware architecture . . . 45

4.2 Hardware specifications . . . 46

4.3 VLSI implementation technology and design flow . . . 47

4.4 Front-end design . . . 47

4.4.1 Stimuli files . . . 49

4.4.2 Hardware modeling . . . 49

4.4.3 Logical synthesis . . . 50

4.4.4 DFT synthesis . . . 50

4.5 Back-end design . . . 51

4.5.1 Floorplanning and Placement . . . 51

4.5.2 Clock tree synthesis . . . 53

4.5.3 Golden netlist . . . 54

4.5.4 Routing . . . 54

4.6 Physical verification and modification . . . 58

4.6.1 Layout versus schematic (LVS) . . . 58

4.6.2 Design rule checking (DRC) . . . 59

4.7 Chip Layout . . . 60

4.8 Comparisons . . . 63

4.9 Summary . . . 63

5 Summaries of Contributions **65**

A Program 1 **66**

B Program 2 **68**

References **79**

VITA AUCTORIS **82**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

1.1 Smart card 3

2.1 Galois type LFSR when $F(x) = x^m + x^k + 1$ 15

2.2 MSB first bit-serial PB finite field multiplier when $F(x) = x^5 + x^2 + 1$ 20

2.3 LSB first bit-serial PB finite field multiplier when $F(x) = x^5 + x^2 + 1$ 21

3.1 Proposed hybrid finite field multiplier 29

3.2 M3: The constant finite field multiplier $Z = x^8 Y$ 31

3.3 8×233 bit-parallel partial product generator 33

3.4 The architecture of the general constant finite field multiplier $Z = x^w Y$ 33

3.5 The architecture of AND network 34

3.6 The architecture of XOR network 34

3.7 The architecture of sub XOR network 35

3.8 Alternative BPWS PB finite field multiplier over $GF(2^{233})$ 37

3.9 The BPWS PB finite field multiplier in $GF(2^m)$ 40

3.10 The alternative BPWS PB finite field multiplier in $GF(2^m)$ 41

4.1 The schematic of the hardware 46

4.2 CMC digital design flow 48

4.3 Illustration of placement 53

4.4 Functional multiplication test 55

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5 Waveform of functional multiplication test 55
4.6 Functional squaring test 56
4.7 Waveform of functional squaring test 56
4.8 Timing limit checking 57
4.9 The result of LVS 59
4.10 The result of phantom level DRC 60
4.11 The result of standard DRC from CMC 61
4.12 The result of anntenna DRC from CMC 61
4.13 The layout of the chip 62

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

2.1 Addition rule for $GF(2)$ 11
2.2 Multiplication rule for $GF(2)$ 11
2.3 Close form representation of the product coefficient c_i 19
2.4 The summary for MSB and LSB bit-serial finite field multipliers . . . 21
2.5 Close form representation for the squaring coefficient c_i 23

3.1 NIST recommendations 26
3.2 The output and intermediate results upon each clock cycle 30
3.3 Circuit and timing complexities of the 8×233 partial product generator 35
3.4 Circuit and timing complexities of the BPWS PB finite field multiplier 35
3.5 The values of output and other modules on each clock cycle 39
3.6 The circuit and timing complexities of alternative BPWS PB finite field multiplier 39
3.7 The comparisons among bit-parallel, bit-serial and BPWS finite field multipliers 42

4.1 Specifications 47
4.2 Results of logic synthesis 50
4.3 The hardware parameters 63
4.4 Comparisons among VLSI implementation of finite field multipliers . 64

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abbreviations

AOP	All One Polynomial
ASIC	Application-Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BPWS	Bit-Parallel Word-Serial
CMC	Canadian Microelectronics Corporation
CMOS	Complementary Metal Oxide Semiconductor
DFT	Design For Testbility
DRC	Design Rule Check
DSS	Digital Signature Standard
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
GF	Galois Field (Finite Field)
HDL	Hardware Description Language
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LSW	Least Significant Word
LVS	Layout Versus Schematic
MSB	Most Significant Bit
MSW	Most significant Word
NIST	National Institute of Standard and Technology
PB	Polynomial Basis
RC	Resistance-Capacitance
RSA	Rivest, Shamir, and Adleman
	(public key encryption technology)
RSPF	Regular Standard Parasitic Format
RTL	Register Transfer Level
SOC	System On Chip
TSMC	Taiwan Semiconductor Manufacture Company

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

In this chapter, the research motivations are introduced in the first section, the followings are our research goals and the thesis organization.

1.1 Research motivations

Our research is originated from smart card applications.

1.1.1 Smart card and its applications

A smart card is a credit card sized plastic card embedded with an integrated circuit (IC) chip. It provides not only memory capacity, but computational capability as well. The self-containment of smart card makes it resistant to attack as it does not need to depend upon potentially vulnerable external resources. Because of this characteristic, smart cards are often used in different security applications which require strong security protection and authentication.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The success of the smart cards in Europe began in the early eighties, between 1982 and 1984 when Carte Bancaire (the French Bank Card Group) had the first smart card pilot running [9]. Together with Bull (A French company), Philips (An international company) and Schlumberger (An international company), Carte Bancaire launched trials in the French cities of Blois, Caen, and Lyon. The trials were a great success. Following these trials, French banks launched the use of smart cards for banking. This was the first mass rollout of smart cards in the banking industry.

Today, smart cards are used for many different purposes in daily life. Smart card can be a phone card, people can use it to make local or long distance call in a phone booth; Smart card can also act as an identification card which is used to prove the identity of the card holder. For example, it can be used as campus access card. In Finland, smart cards are used as the Finnish National Electronic Identity (FINEID) cards; Smart card can be a medical card which stores the medical history of a person; Furthermore, the smart card can be used as a credit/debit bank card which allows off-line transactions. In the near future, the traditional magnetic strip card will be replaced and integrated together into a single card by using the multi-application smart card, which is known as an electronic purse or wallet in the smart card industry. All of these applications require sensitive data to be stored in the card, such as biometrics information of the card owner, personal medical history, and cryptographic keys for authentication, etc.

A smart card [9] shown in Figure 1.1 consists of a microprocessor, ROM (Read Only Memory), EEPROM (Electrical Erasable Programmable Read Only Memory), and RAM (Random Access Memory).

Today's smart cards have approximately the same computing power as the first IBM PC [9]. At present, most smart cards have an inexpensive 8-bit microprocessor, but the high-end cards can have a 16-bit or 32-bit processor. An optional cryptographic coprocessor (security processor) increases the performance of cryptographic

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

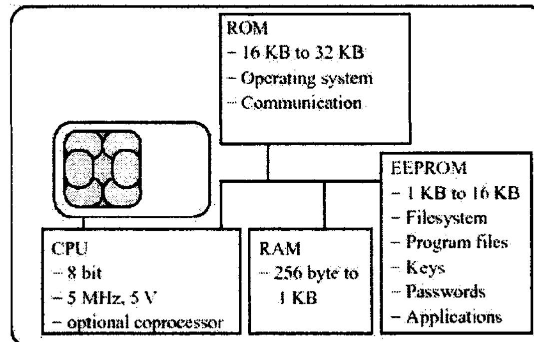

Figure 1.1: Smart card

operations. The working frequency of smart card normally is 5 MHz. The RAM size of most smart cards is 256 bytes to 1 kilobyte. The chip size is at most $25mm^2$ and there are also a card operating system and might have some applications in smart card [9].

Since the working frequency is relatively slow, furthermore, the memory inside the smart card is very limited and the card operating system is not for security purpose, software implementation of any security application in smart card is normally very slow and considered insecure. We usually solve the high load of cryptographic computations by means of the cryptographic coprocessor. Due to the above features of the smart card and its area constraint which we cannot make the chip very large, the coprocessor hardware inside the smart card should trade-off area for speed.

1.1.2 Cryptography and cryptosystems

Cryptography and cryptosystems gain extensive uses in all kinds of security applications.

Cryptography is the study of mathematical techniques related to the aspects of information security such as confidentiality, data integrity, and data origin authenti-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cation, etc. Cryptosystems can normally be classified into two groups : *symmetric* cryptosystems [1] and *asymmetric* cryptosystems [1] (also called *public key* cryptosystems).

Symmetric cryptosystems use the same key to encrypt and decrypt information. Implementations of symmetric key encryption/decryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. But symmetric cryptosystems have a problem of low security for their key. It is generally very difficult to transmit the secret key from the sender to the recipient securely and in a tamperproof fashion. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can also encrypt new messages and send them as if they came from one of the two parties who were originally using the key. Frequently, trusted couriers are used as a solution to this problem. A more efficient and reliable solution is a public key cryptosystem.

Public key cryptosystems involve a pair of keys (a **public key** and a **private key**) which are associated with an entity that needs to authenticate its identity or to sign or to encrypt data. Such public key cryptosystems have the abilities to perform the functions of key exchange, digital signature, encryption and decryption. Nowadays there are two main public key cryptosystems which are RSA cryptosystems and Elliptic curve (EC) cryptosystems.

1.1.3 Elliptic curve cryptography (ECC)

In 1985, N. Koblitz [11] and V.S. Miller [16] independently proposed elliptic curves (EC) for public key cryptosystems. Their proposal however was not considered as a new cryptographic algorithm with elliptic curves over finite fields, as they implemented existing algorithms, like Diffie-Hellman, using elliptic curves [23].

Over the past two decades, elliptic curve has been well researched by many schol-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ars. These cryptosystems need a much shorter key than RSA cryptosystems to provide the same security strength. It appears that an elliptic curve cryptosystem implemented over the 160-bit field $GF(2^{160})$ currently offers roughly the same resistance to side channel attack as would a 1024-bit RSA [20] and an elliptic curve cryptosystem over a 136-bit field $GF(2^{136})$ gives us roughly the same security as 768-bit RSA [22]. The basic operations in RSA cryptosystems are integer modular operations, while in EC cryptosystems, finite field operations are the basic operations. When elliptic curve is over finite field $GF(2^m)$, the implementation of EC cryptosystems will save more hardware resources than RSA cryptosystems since the field elements in $GF(2^m)$ can be represented by m-bit binary numbers and the binary number is well adopted by computer arithmetic. All these advantages make EC an ideal candidate for smart card applications.

The finite field operations in EC cryptosystems can be broken into finite field additions, multiplications, squarings and inversions. Finite field addition can be simply implemented by XOR gates and normally considered as almost free. These finite field adders are carry-free, and thus are faster than normal carry ripple adders. The finite field inversion can be further broken into finite field multiplications and finite field squarings [26, 4] and finite field squaring is a special case of finite field multiplication, thus, the finite field multiplier becomes the key component in EC hardware.

A number of finite field multiplier architectures have been proposed with different emphasis for various security applications. Full bit-parallel finite field multipliers [28, 12, 13, 14, 15, 19, 25, 27] can yield high throughputs, bit-serial finite field multipliers [2, 8, 24, 27] only need small area. These finite field multiplier architectures can satisfy nearly all security applications. However, the full bit-parallel finite field multipliers are still too large for smart cards because they have the chip area constraint; On the other hand, finite field multiplier with bit-serial structure is too slow since the frequency is low for smart card and too many clock cycles are needed to perform one

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

multiplication. We may need a hybrid bit-parallel word-serial (BPWS) finite field multiplier architecture to balance the trade-off between area and speed.

1.2 Research goals

One of our research goals is to design a new hybrid BPWS finite field multiplier architecture for smart card, such a finite field multiplier should trade-off between area with speed; The final goal is to design an ASIC (Application Specified Integrated Circuit) chip which can perform finite field multiplication using this BPWS architecture and finite field squaring using a bit-parallel finite field squarer.

1.3 Thesis organization

Chapter 2 introduces the basic concepts of finite field, finite field element representations, finite field operation and a few state-of-art polynomial basis finite field multiplier architectures.

A BPWS finite field multiplier architecture is designed in Chapter 3, a novel 8×233 bit-parallel partial product generator is developed, an alternative BPWS finite field multiplier architecture and the general architectures are also introduced in this chapter.

The design of an ASIC chip which has the BPWS finite field multiplier together with a full bit parallel squarer is presented in Chapter 4. The VLSI implementation technology is TSMC 0.18 CMOS technology. The design flow is the CMC digital design flow. The results at each design stage are also shown in this chapter.

Chapter 5 presents summary and conclusions of this research and provides future works.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Arithmetic over Finite Field

In this chapter, concept of field, irreducible polynomial, finite field and finite field element representations are introduced. Furthermore Elliptic Curve (EC) and why finite field multiplier is so important for EC are explained. Using these background knowledge, several state-of-the-art polynomial basis finite field multiplier architectures are discussed. At the end of this chapter, a bit parallel finite field squarer is introduced.

2.1 Group, ring, field and finite field

2.1.1 Group

A group [31] $(G, *)$ is defined as a set G together with a binary operation $* : G * G \rightarrow G$. We write "$a * b$" for the result of applying the operation $*$ to the two elements a and b of G. To have a group, $*$ must satisfy the following axioms :

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Associativity: For all a, b and c in G, $(a * b) * c = a * (b * c)$.
- Identity element: There is an element e in G such that for all a in G, $e * a = a = a * e$.
- Inverse element: For all a in G, there is an element b in G such that $a * b = e = b * a$, where e is the identity element from the previous axiom.
- Closure: For all a and b in G, $a * b$ belongs to G.

An **abelian group** is a group $(G, *)$ that is commutative, i.e., $a * b = b * a$ holds for all elements a and b in G.

Examples

1. The set of integers under addition forms a group and also an abelian group.
2. The set of nonzero rational numbers under multiplication is a group and also an abelian group.
3. The set of integers under multiplication is NOT a group.

2.1.2 Ring

A ring [32] is an abelian group $(R, +)$, together with a second binary operation $*$ such that for all a, b, and c in R,

$$\begin{aligned} a * (b * c) &= (a * b) * c \\ a * (b + c) &= (a * b) + (a * c) \\ (a + b) * c &= (a * c) + (b * c) \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and such that there exists a multiplicative identity, or unity, that is, an element 1 so that for all a in R,

$$a * 1 = 1 * a = a$$

The identity element with respect to + is called the zero element of the ring and written as 0.

A **commutative ring** is a ring in which the multiplication operation obeys the commutative law, i.e., if a and b are any elements of the ring, and if the multiplication operation is written as $*$, then $a * b = b * a$.

Examples

Integers, rational numbers, real numbers and complex number under addtion and multiplication are all examples of rings.

2.1.3 Field

A field [33] is a commutative ring $(F, +, *)$ such that additive identity element 0 does not equal multiplicative identity 1 and all elements of F except 0 have a multiplicative inverse. Besides the above axioms of group and ring, a field also obey the following rules:

- Existence of an additive identity
  There exists an element 0 in F, such that for all a belonging to F, $a + 0 = a$.
- Existence of a multiplicative identity
  There exists an element 1 in F different from 0, such that for all a belonging to F, $a * 1 = a$.
- Existence of multiplicative inverses

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For every $a \neq 0$ belonging to F, there exists an element a^{-1} in F, such that $a * a^{-1} = 1$.

Examples

Some examples of fields are listed below:

- The rational numbers $\mathcal{Q} = \{a/b \mid a, b \text{ in } \mathcal{Z}, b \neq 0\}$, where $\mathcal{Z}$ is the set of integers.
- The real numbers $\mathcal{R}$.
- The complex numbers $\mathcal{C}$.
- The smallest field has only two elements: 0 and 1. It is sometimes denoted by $\mathbf{F}_2$ or $\mathbf{GF}(2)$. It has important uses in cryptography and coding theory.

2.1.4 Finite field

Finite field is also called Galois field (so named in honor of Evariste Galois). Finite field is a field that contains only finite number of elements.
All finite fields have prime characteristic. The number (or the order) of the elements in a finite field is always a prime or a power of a prime [3].

- If p is a prime, the integers modulo p form a field with p elements, denoted by $GF(p)$. Every other field with p elements is isomorphic to this one.
- If $q = p^m$ is a prime power, then there exists up to isomorphism exactly one field with q elements, written as $GF(q)$ or $GF(p^m)$.

The finite field that is used in this thesis is $GF(2^m)$. When we say finite field in this thesis, we refer to $GF(2^m)$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finite field $GF(2^m)$ can be defined (or generated) by an irreducible polynomial $F(x)$ of degree m with its coefficients in $GF(2)$,

$$F(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \ldots + f_1 x + 1$$

where $f_i \in GF(2)$, for $i = 1, 2, \ldots, m-1$.

The elements in this finite field can be treated as the polynomials of degree n ($0 \leq n < m$) with the coefficients in $GF(2)$ or the m-bit binary numbers.

The finite field $GF(2)$ consists of only two elements which are 0 and 1 and satisfies the following addition and multiplication rules which are summarized in Table 2.1 and Table 2.2.

+	0	1
0	0	1
1	1	0

Table 2.1: Addition rule for $GF(2)$

*	0	1
0	0	0
1	0	1

Table 2.2: Multiplication rule for $GF(2)$

2.2 Finite field element representations

Like vectors in linear algebra can be represented by various vector spaces, we use bases to represent the field element. There are three main bases used to represent

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the elements in $GF(2^m)$, they are polynomial basis, normal basis and dual bases respectively.

2.2.1 Polynomial basis

Assume x be a root of the irreducible polynomial $F(x)$ which generates the finite field $GF(2^m)$, then $\{1, x, x^2, x^3, \ldots, x^{m-1}\}$ forms a polynomial basis. Any element A in the finite field can be represented as

$$A = \sum_{i=0}^{m-1} a_i x^i = (a_0, a_1, a_2, \ldots, a_{m-1}), \text{ where } a_i \in GF(2) \ .$$

Normal basis and dual bases are other two main bases. The detail discussion can be found in [2].

2.3 Finite field operation

Given a finite field $GF(2^m)$ which is generated by an irreducible polynomial $F(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \ldots + f_1 x + 1$, where $f_i \in GF(2)$ for $i = 1, 2, \ldots, m-1$, let A and B be any two elements in $GF(2^m)$ and $\{1, x, x^2, \ldots, x^{m-1}\}$ be the polynomial basis, A and B can be expressed as

$$A = \sum_{i=0}^{m-1} a_i x^i \ , \quad \text{and}$$
$$B = \sum_{i=0}^{m-1} b_i x^i \ ,$$

where $a_i, b_i \in GF(2)$ for $i = 0, 1, \ldots, m-1$.

2.3.1 Addition

Let S be the sum of A and B and S be expressed as

$$S = \sum_{i=0}^{m-1} s_i x^i \ ,$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where $s_i \in GF(2)$, for $i = 0, 1, \ldots, m-2$, then

$$\begin{aligned} S = A + B &= \sum_{i=0}^{m-1} a_i x^i + \sum_{i=0}^{m-1} b_i x^i \\ &= \sum_{i=0}^{m-1} (a_i + b_i) x^i \ . \end{aligned}$$

Thus, we can get

$$s_i = a_i + b_i \ , \tag{2.1}$$

where s_i, a_i, $b_i \in GF(2)$, for $i = 0, 1, \ldots, m-1$.

The addition expressed in Formula 2.1 obeys the addition rule for $GF(2)$ which is described in Table 2.1 and can be implemented by an XOR gate. Hence, the addition in $GF(2^m)$ can be implemented by m XOR gates.

2.3.2 Multiplication

Let C be the product of A and B and C be expressed as

$$C = \sum_{i=0}^{m-1} c_i x^i \ ,$$

where $c_i \in GF(2)$, for $i = 0, 1, \ldots, m-1$, then

$$C = AB = \sum_{i=0}^{m-1} a_i x^i \sum_{j=0}^{m-1} b_j x^j \mathrm{mod} F(x) \ , \tag{2.2}$$

where i, $j = 0, 1, \ldots, m-1$. Formula 2.2 involves two operations. One is polynomial multiplication which is straightforward; The other is the reduction modulo the irreducible polynomial $F(x)$. When the irreducible polynomial is trinomial, the coefficients c_i has a close form of expression in terms of $\{a_i\}$ and $\{b_i\}$ [28] which will be introduced later.

Finite field squaring is a special case of finite field multiplication. Let C be the squaring of A and C be expressed as

$$C = \sum_{i=0}^{m-1} c_i x^i \ ,$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where $c_i \in GF(2)$, for $i = 0, 1, \ldots, m-1$, then

$$\begin{aligned} C = \sum_{i=0}^{m-1} c_i x^i &= A^2 \bmod F(x) \\ &= a_0 + a_1 x^2 + a_2 x^4 + \ldots + a_{m-1} x^{2m-2} \bmod F(x) \\ &= \sum_{i=0}^{m-1} a_i x^{2i} \bmod F(x) \\ &= \sum_{i=0}^{2m-2} a'_i x^i, \end{aligned}$$

where a'_i is given by

$$a'_i = \begin{cases} a_{\frac{i}{2}} & \text{if } i \text{ is even;} \\ 0 & \text{if } i \text{ is odd.} \end{cases} \qquad (2.3)$$

When $F(x)$ is an irreducible trinomial, the coefficient c_i has close form representations and the architecture of finite field squarer is much simpler than that of finite field multiplier. We will discuss this in detail later.

The detail discussions about finite field multiplications based on normal basis and dual bases can be found in [2, 10, 27].

2.3.3 Comparisons among the multiplications with different basis

Three finite field multipliers which are based on polynomial basis, normal basis and dual bases respectively were compared by I.S. Hsu et al in [10], which are the dual basis multiplier, the normal basis multiplier, and the polynomial basis multiplier. The dual basis multiplier occupies the smallest amount of chip area in VLSI implementation if the basis conversion is not included; The area of the normal basis multiplier however grows dramatically as the order of field goes up; The polynomial basis multiplier does not require basis conversion, it is readily matched to any input or output system, the design and expansion to higher order finite fields are easier to realize than the dual or normal basis multipliers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.4 Galois type linear feedback shift register (LFSR)

Galois type LFSR are widely used in bit-serial finite field multiplier architectures. Galois type LFSR is simple and the architecture of Galois type LFSR can be easily obtained from the irreducible polynomial $F(x)$ which generates $GF(2^m)$. Figure 2.1 shows the Galois type LFSR architecture when the irreducible polynomial is $F(x) = x^m + x^k + 1$.

Galois type LFSR serves as a constant multiplier, i.e., if the current value of Galois

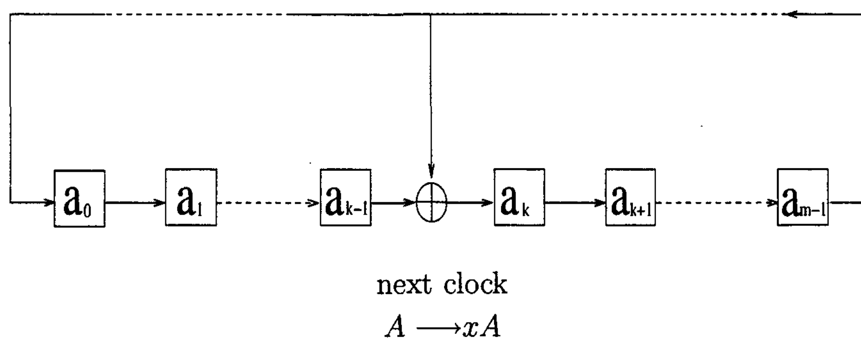

Figure 2.1: Galois type LFSR when $F(x) = x^m + x^k + 1$

type LFSR is A, the value of this Galois type LFSR during the next clock cycle will be xA.

2.5 Elliptic curve

Elliptic curve cryptography (ECC) was proposed by Victor Miller and Neal Koblitz in the mid 1980s [11, ?]. EC over $GF(2^m)$ has the following form,

$$E : y^2 + xy = x^3 + a_2x^2 + a_0 \tag{2.4}$$

where a_0 and a_2 are the elements in finite field $GF(2^m)$ and E represents the elliptic curve.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The elliptic curve is the set of points (x, y) which are the solutions to Formula 2.4 together with an extra point $\mathcal{O}$ which is called the **point at infinity**. The coordinate values x and y of the point are also the elements in $GF(2^m)$. The number of such points is finite.

This set of points on an EC forms a **group** under a certain addition rule (or it is called **addition law**), which is written using the notation $+$. The point $\mathcal{O}$ is the identity element of the group.

Given a point $P = (x, y)$ and a positive integer t, we define $[t]P = P + P + \ldots + P$(t times). The **order** of a point $P = (x, y)$ is the smallest positive integer n such that $[n]P = \mathcal{O}$.

We denote $< P >$ as the **group generated by** P, i.e.

$$< P >= \{\mathcal{O}, P, P+P, P+P+P, \ldots, \overbrace{P+P+\ldots+P}^{n-1}\}$$

The security of ECC relies on **elliptic curve discrete logarithm problem (ECDLP)**: Let E be an elliptic curve over $GF(2^m)$, let P be a point on the elliptic curve, let Q be a point in $< P >$. Finding an integer l such that $Q = [l]P$ is the ECDLP.

It is widely believed that the l in ECDLP is hard to computationally solve when the point P has large prime order.

Point operations on EC conform the addition law which is defined below.

Assume $P_1(x_1, y_1)$, $P_2(x_2, y_2) \in E$ and $P_1 + P_2 = P_3(x_3, y_3)$, we define

$$P_1 \neq P_2 : \begin{cases} x_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + a_2 \\ y_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1 + x_3) + x_3 + y_1 \end{cases}, \quad (2.5)$$

$$P_1 = P_2 : \begin{cases} x_3 = {x_1}^2 + \frac{a_0}{{x_1}^2} \\ y_3 = {x_1}^2 + \left(x_1 + \frac{y_1}{x_1}\right) x_3 + x_3 \end{cases}. \quad (2.6)$$

When $P_1 \neq P_2$, we can find the point P_3 using Formula 2.5, this is also called point addition; When $P_1 = P_2$, we can obtain P_3 by the formula 2.6, this is also called

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

point doubling. All the arithmetic operations in these two formulae are finite field operations.

From above Formula 2.5 and Formula 2.6 we can see that point operations always can be broken into **finite field multiplications**, **finite field squarings**, **finite field inversions** and **finite field additions**. Finite field addition in $GF(2^m)$ can be implemented by m XOR gates, so the finite field addition is considered as almost free. We usually calculate the finite field inversion by means of extended Euclidian algorithm or Fermat theorem [26, 4]. From Fermat theorem, the inversion of any field element A can be obtained by the following formula

$$A^{-1} = A^{2^m-2} . \tag{2.7}$$

Equation 2.7 can be further broken into finite field multiplications and finite field squarings. Thus, the finite field inversion can be obtained from finite field multiplication and finite field squaring. Since finite field squaring is a special case of finite field multiplication, in addition, finite field squarer is much simpler than finite field multiplier (as will be seen later), the finite field multiplier is our focus. As we have already discussed in Section 2.3.2, the polynomial basis multiplier has the advantages over other basis multipliers, the polynomial basis (PB) finite field multiplier is the focus in this thesis.

2.6 Polynomial basis (PB) finite field multipliers

A number of PB finite field multipliers have been proposed [2, 6, 8, 13, 14, 25, 21, 28, 24]. Two typical kinds of PB finite field multipliers are bit-parallel PB finite field multipliers and bit-serial PB finite field multipliers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.6.1 Bit-parallel PB finite field multipliers

There are many bit-parallel finite field multipliers which have been proposed so far. A bit-parallel systolic multiplier has been proposed in [13] for the $GF(2^m)$ using the polynomial basis representation. The finite field is generated by the irreducible trinomial $x^m + x^n + 1$ of degree m. The permutation polynomial and Horner's algorithm are applied to create a low complexity systolic multiplier. The circuit includes m^2 2-input AND gates, $m^2 + m - 1$ 2-input XOR gates and $3m^2 + 2m - 2$ 1-bit latches. The latency of the systolic multiplier over $GF(2^m)$ is only $2m - 1$ clock cycles with a throughput rate of one result per clock cycle.

In [14], a bit-parallel systolic AOP-based (All One Polynomial based) multiplier for $GF(2^m)$ has been presented. The architectures of the two AOP-based multipliers can also be adopted to implement ESP-based (Equal Space Polynomial based) multipliers.

In [25], an architecture based on a new formulation of the multiplication matrix is described and circuit complexities are analyzed when the finite field is generated by trinomial $x^m + x^n + 1$.

In [21], a new bit-parallel structure for a multiplier with low complexity in Galois fields is introduced. The multiplier operates over composite fields $GF((2^n)^m)$, with $k = nm$. It is shown that this operation has complexity of order $\mathcal{O}(k^{\log_2 3})$ under certain constraints regarding k.

A bit-parallel finite field multiplier based on polynomial basis is discussed in [28]. Let A and B be any two field elements represented by polynomial basis as follow

$$A = \sum_{i=0}^{m-1} a_i x^i \text{ , and}$$

$$B = \sum_{i=0}^{m-1} b_i x^i \text{ ,}$$

where $a_i, b_i \in GF(2)$, for $i = 0, 1, 2, \ldots, m-1$. Let C be their product

$$C = \sum_{i=0}^{m-1} c_i x^i = AB \text{ .}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

When $F(x)$ is trinomial i.e. $F(x) = x^m + x^k + 1$, c_i has the following close form representations shown in Table 2.3.

$F(x) = x^m + x + 1$	c_0	$=$	$s_0 + s_m$	
	c_i	$=$	$s_i + s_{m+i-1} + s_{m+i}$,	$i = 1, 2, \ldots, m-1$
	c_{m-1}	$=$	$s_{m-1} + s_{2m-2}$	
$F(x) = x^m + x^k + 1$	c_i	$=$	$s_i + s_{m+i} + s_{2m-k+i}$	$i = 0, 1, \ldots, k-2$
$1 < k < m/2$	c_{k-1}	$=$	$s_{k-1} + s_{m+k-1}$	
	c_i	$=$	$s_i + s_{m+i} + s_{m-k+i} + s_{2m-2k+i}$	$i = k, \ldots, 2k-2$
	c_i	$=$	$s_i + s_{m+i} + s_{m-k+i}$	$i = 2k-1, \ldots, m-2$
	c_{m-1}	$=$	$s_{m-1} + s_{2m-k-1}$	
$F(x) = x^m + x^{m/2} + 1$	c_i	$=$	$s_i + s_{m+i} + s_{3m/2+i}$	$i = 0, 1, \ldots, m/2-2$
	$c_{m/2-1}$	$=$	$s_{m/2-1} + s_{3m/2-1}$	
	c_i	$=$	$s_i + s_{m/2+i}$	$i = m/2, \ldots, m-2$
	c_{m-1}	$=$	$s_{m-1} + s_{3m/2-1}$	

Table 2.3: Close form representation of the product coefficient c_i

2.6.2 Bit-serial PB finite field multipliers

Thomas Beth et al presented two basic architectures for PB bit serial finite field multiplier in [2]. Leilei Song et al have the similar design in [24] and Johann Grobschadl has the same idea with low power implementation in [8]. All the bit serial PB finite field multipliers use Galois type LFSRs.

Most significant bit (MSB) first bit serial PB finite field multiplier

The architecture of MSB first bit serial finite field multiplier [2] is very simple. The Figure 2.2 shows the architecture when irreducible polynomial is $F(x) = x^5 + x^2 + 1$.

There is a Galois type LFSR used in Figure 2.2. The initial value of this Galois type LFSR is 0. One operand B is input in parallel. The other operand A is input in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

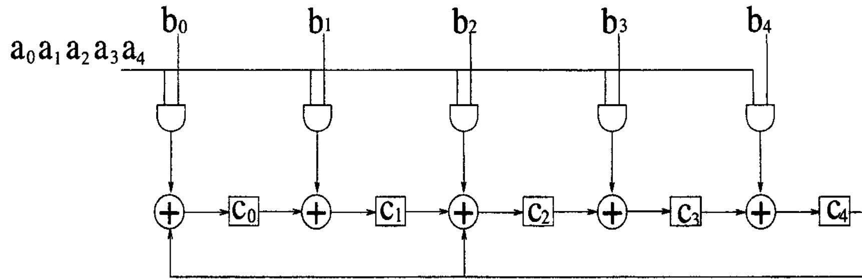

Figure 2.2: MSB first bit-serial PB finite field multiplier when $F(x) = x^5 + x^2 + 1$

serial, upon each clock cycle, one bit a_i in A is fed into the circuit, the input method used is the most significant bit (MSB) first. The final result can be obtained from the outputs c_is after 6 clock cycles. It takes 6 clock cycles to perform one finite field multiplication using this MSB first bit-serial finite field multiplier.

Least significant bit (LSB) first bit-serial PB finite field multiplier

The above MSB first PB finite field multiplier has an alternative form which is LSB first PB finite field multiplier [2]. When irreducible polynomial is still $F(x) = x^5 + x^2 + 1$, the architecture of the LSB first bit-serial PB multiplier is shown in Figure 2.3.

There is a Galois type LFSR in this architecture and it is initially set to one of the operands. The other operand A is input in serial. The input method is the least significant bit (LSB) first. There is one additional 5-bit register which is initially set to 0. The final product can be obtained from the outputs c_is after 6 clock cycles. Like the MSB first bit-serial finite field multiplier, there are 6 clock cycles needed in order to perform one finite field multiplication using this LSB architecture.

Assume all the AND gates and XOR gates have two inputs, the delay of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

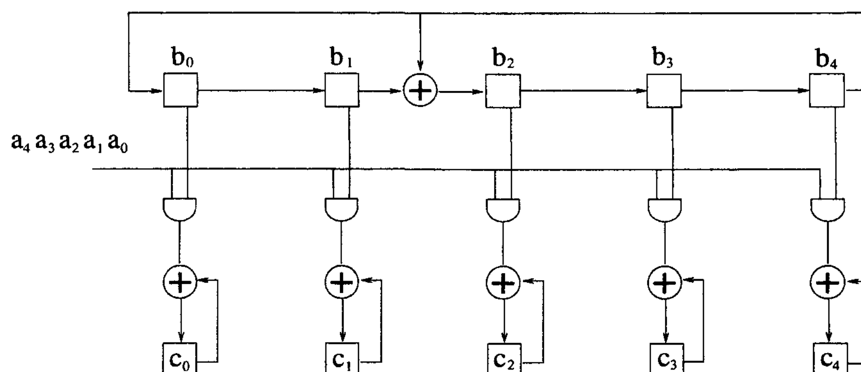

Figure 2.3: LSB first bit-serial PB finite field multiplier when $F(x) = x^5 + x^2 + 1$

AND gate is T_A and the delay of the XOR gate is T_X. When $GF(2^m)$ is generated by irreducible trinomial $F(x) = x^m + x^k + 1$, we can summarize the circuit complexity and speed to perform one multiplication for the above bit-serial PB finite field multipliers in Table 2.4.

Multiplier	Speed (clock cycle)	Circuit complexity	Critical path
MSB first	m	m AND gates $m+1$ XOR gates 1 m-bit register	$2T_X$
LSB first	m	m AND gates $m+1$ XOR gates 2 m-bit registers	$T_A + T_X$

Table 2.4: The summary for MSB and LSB bit-serial finite field multipliers

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.6.3 Bit-parallel PB finite field squarer

Finite field squaring is a special case of finite field multiplication. The architecture of bit-parallel PB finite field squarer is much simpler than that of bit-parallel PB finite field multiplier. A trinomial based bit-parallel PB finite field squarer is introduced in [28].

Suppose that $GF(2^m)$ is generated by the irreducible polynomial $F(x)$ over $GF(2)$, an arbitrary field element A can be expressed by the polynomial basis as

$$A = \sum_{i=0}^{m-1} a_i x^i \ .$$

Let C be the squaring of A, we have

$$\begin{aligned} C = \sum_{i=0}^{m-1} c_i x^i &= A^2 \bmod F(x) \\ &= a_0 + a_1 x^2 + a_2 x^4 + \ldots + a_{m-1} x^{2m-2} \bmod F(x) \\ &= \sum_{i=0}^{m-1} a_i x^{2i} \bmod F(x) \\ &= \sum_{i=0}^{2m-2} a'_i x^i, \end{aligned}$$

where a'_i is given by

$$a'_i = \begin{cases} a_{\frac{i}{2}} & \text{if } i \text{ is even;} \\ 0 & \text{if } i \text{ is odd.} \end{cases} \tag{2.8}$$

When $F(x)$ is an irreducible trinomial, i.e., $F(x) = x^m + x^k + 1$, where $1 \leq k \leq \frac{m}{2}$, the coefficient c_i has close form representations which are summarized in Table 2.5.

2.7 Summary

Some basic concepts about finite field, field element representation and finite field arithmetic are introduced in the first three sections. In Section 2.5, Elliptic curve cryptography is briefly touched and we know that the finite field multiplier is the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$F(x) = x^m + x + 1$	m is even	$c_i = a'_i + a'_{m+i}$,	$i = 0, 2, \ldots, m-2$,
		$c_i = a'_{m+i-1}$,	$i = 1, 3, \ldots, m-1$.
	m is odd	$c_0 = a'_0$,	
		$c_i = a'_{m+i}$,	$i = 1, 3, \ldots, m-2$,
		$c_i = a'_i + a'_{m+i-1}$,	$i = 2, 4, \ldots, m-1$.
$F(x) = x^m + x^k + 1$	k is even m is odd	$c_i = a'_i + a'_{2m-k+i}$,	$i = 0, 2, \ldots, k-2$,
		$c_i = a'_{m+i}$,	$i = 1, 3, \ldots, k-1$,
		$c_i = a'_i + a'_{2m-2k+i}$,	$i = k, k+2, \ldots, 2k-2$,
		$c_i = a'_{m+i} + a'_{m-k+i}$,	$i = k+1, k+3, \ldots, m-2$,
		$c_i = a'_i$,	$i = 2k, 2k+2, \ldots, m-1$.
	k is odd m is odd	$c_i = a'_i$,	$i = 0, 2, \ldots, k-1$,
		$c_i = a'_{m+i} + a'_{2m-k+i}$,	$i = 1, 3, \ldots, k-2$,
		$c_i = a'_i + a'_{m-k+i} + a'_{2m-2k+i}$,	$i = k+1, k+3, \ldots, 2k-2$,
		$c_i = a'_{m+i}$,	$i = k, k+2, \ldots, m-2$,
		$c_i = a'_i + a'_{m-k+i}$,	$i = 2k, 2k+2, \ldots, m-1$.
	k is odd m is even	$c_i = a'_i + a'_{m+i}$,	$i = 0, 2, \ldots, k-1$,
		$c_i = a'_{2m-k+i}$,	$i = 1, 3, \ldots, k-2$,
		$c_i = a'_i + a'_{m+i} + a'_{2m-2k+i}$,	$i = k+1, k+3, \ldots, 2k-2$,
		$c_i = a'_{m-k+i}$,	$i = k, k+2, \ldots, m-1$,
		$c_i = a'_i + a'_{m+i}$,	$i = 2k, 2k+2, \ldots, m-2$.
$F(x) = x^m + x^{\frac{m}{2}} + 1$		$c_i = a'_i + a'_{m+i}$,	$i = 0, 2, \ldots, \frac{m}{2}-1$,
		$c_i = a'_{\frac{3m}{2}+i}$,	$i = 1, 3, \ldots, \frac{m}{2}-2$,
		$c_i = a'_i$,	$i = \frac{m}{2}+1, \frac{m}{2}+3, \ldots, m-2$,
		$c_i = a'_{\frac{m}{2}+i}$,	$i = \frac{m}{2}, \frac{m}{2}+2, \ldots, m-1$.

Table 2.5: Close form representation for the squaring coefficient c_i

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

basic key component in elliptic curve hardware. Two typical architectures of finite field multipliers are introduced in Section 2.6. At last, we mentioned a bit-parallel PB finite field squarer.

In the next chapter, we will design bit-parallel word-serial PB finite field multiplier architectures which have the trade-off between gate counts (area) and speed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

Design of Bit-Parallel Word-Serial PB Finite Field Multipliers

In this chapter, accepted standard for EC cryptosystems is introduced at the beginning. Next, the bit-parallel word-serial (BPWS) PB finite field multiplier is designed. An alternative form of BPWS PB finite field multiplier architecture is introduced in the following section. At the end of this chapter, general forms of BPWS PB multiplier architectures and the comparisons are presented.

It is known that, ECC devices require less storage, less power, less memory, and less bandwidth than other systems [29, 22, 34]. This allows implementation of cryptography in platforms that are constrained, such as wireless devices, handheld computers, smart cards. Several organizations such as NIST (National Institute of Standard and Technology), ANSI, IEEE etc. have standardized ECC [34]. NIST issues

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

standards that are mandatory for US Federal Government agencies to follow. NIST recommends not only key establishment schemes, key management in Special Publication 800-56 and 800-57 for EC cryptosystems, but also digital signature standard (DSS) in Federal Information Processing Standards (FIPS) 186-2 [29] with elliptic curve domain parameters. We follow NIST recommendations in this thesis as NIST recommendations for ECC are well and widely adopted.

3.1 NIST recommendations

National Institute of Standard and Technology (NIST) recommends five finite fields which are generated by five irreducible polynomials for EC cryptosystems [29]. These five polynomials are shown in Table 3.1.

In this thesis, $GF(2^{233})$ which is generated by the irreducible trinomial $F(x) =$

Degree	Irreducible Polynomial $F(x)$
163	$F(x) = x^{163} + x^7 + x^6 + x^3 + 1$
233	$F(x) = x^{233} + x^{74} + 1$
283	$F(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
409	$F(x) = x^{409} + x^{87} + 1$
571	$F(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

Table 3.1: NIST recommendations

$x^{233} + x^{74} + 1$ is our choice since $GF(2^{233})$ can satisfy the security requirements for smart card applications and the irreducible trinomial can significantly reduce the circuit complexity.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2 Design of BPWS PB finite field Multiplier

3.2.1 Multiplication algorithm

Let the irreducible polynomial be

$$F(x) = x^{233} + x^{74} + 1 ,$$

then the polynomial basis for $GF(2^{233})$ can be given as $\{1, x, x^2, \ldots, x^{232}\}$.
Let $A, B \in GF(2^{233})$ be any two field elements and C be their product. We can write A and B as

$$A = \sum_{i=0}^{232} a_i x^i, \quad \text{and}$$
$$B = \sum_{i=0}^{232} b_i x^i.$$

Then the product C is

$$C = AB \bmod F(x) . \tag{3.1}$$

We can divide the operand A into 30 groups (words) from the least significant bit of A and let each word contain 8 bits. In the $30th$ word, we append seven "0"s as the most significant seven bits. This can be shown as follow,

$$A = (\underbrace{0,0,0,0,0,0,0,a_{232}}_{A_{29}}, \underbrace{a_{231}, a_{230}, \ldots, a_{224}}_{A_{28}}, \ldots, \underbrace{a_{15}, a_{14}, \ldots, a_8}_{A_1}, \underbrace{a_7, a_6, \ldots, a_0}_{A_0}) ,$$

where A_j is the word, for $j = 0, 1, \ldots, 29$.
Let us denote A_j as

$$A_j = a_{8j+7}x^7 + a_{8j+6}x^6 + \ldots + a_{8j+1}x + a_{8j} , \tag{3.2}$$

for $j = 0, 1, 2, \ldots, 29$, where $a_i = 0$ for $i = 233, 234, \ldots, 239$.
Then A can be expressed as

$$A = (\ldots(A_{29}x^8 + A_{28})x^8 + \ldots + A_1)x^8 + A_0 . \tag{3.3}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thus, the product C can be expressed in the following formula,

$$\begin{aligned} C &= AB \bmod F(x) \\ &= ((\ldots(A_{29}x^8 + A_{28})x^8 + \ldots + A_1)x^8 + A_0)B \bmod F(x) \\ &= (\ldots(A_{29}Bx^8 + A_{28}B)x^8 + \ldots + A_1B)x^8 + A_0B \bmod F(x) \end{aligned} \quad (3.4)$$

Let

$$C_j = C_{j-1}x^8 + D_j \text{ , for } j = 0, 1, \ldots, 29, \quad (3.5)$$

where

$$D_j = A_{29-j}B \text{ , for } j = 0, 1, \ldots, 29 . \quad (3.6)$$

If we assume $C_{-1} = 0$, then it can be seen from (3.4) (3.5) and (3.6) that

$$C = C_{29} . \quad (3.7)$$

3.2.2 Bit-parallel word-serial multiplier architecture

The above iterative processes (3.5) and (3.6) can be mapped into the architecture shown in Figure 3.1.

The multiplier architecture has two input ports $I1$, $I2$ and one output port. Input port $I2$ is 8 bits wide and used to serially input the words A_{29-j}, $j = 0, 1, \ldots, 29$. The other input port $I1$ is 233 bits wide and used to input the other operand B. The output port is 233 bits wide which is used to output the product.

There are four modules in this architecture. These are :

Module M1 : Module M1 is a 8×233 bit-parallel partial product generator. M1 implements the equation (3.6) by taking inputs of 8-bit word A_{29-j} and 233-bit B and yielding the partial product $A_{29-j} \times B$ after the jth clock cycle. The detailed architecture of M1 will be discussed later in Section 3.2.4.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

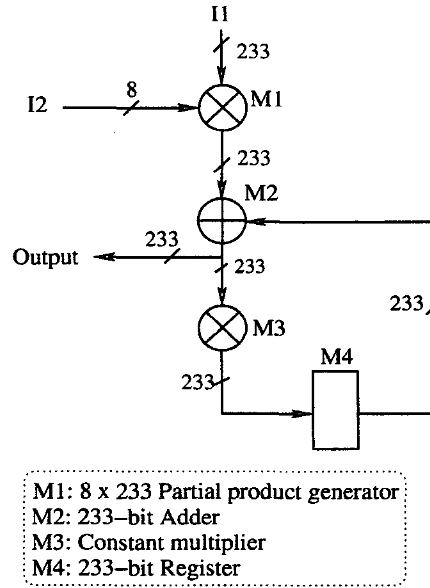

Figure 3.1: Proposed hybrid finite field multiplier

Module M2 : Module M2 is a bit-parallel finite field adder which realizes the equation (3.5) by taking inputs D_j from M1 and $C_{j-1}x^8$ from M4. This adder can be easily implemented by 233 XOR gates. The output of M2 is actually the output of the proposed BPWS PB finite field multiplier after 30 clock cycles.

Module M3 : Module M3 is a bit-parallel finite field constant multiplier. The two operands include the constant x^8 and the output of module M2 which is C_j. The output of module M3 is C_jx^8.

Module M4 : Module M4 is a 233-bit register which is used to store the intermediate results. The output of module M4 is $C_{j-1}x^8$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Let the content of M4 be initialized as 0. Assume the operand B be available at $I1$ from the beginning until the product C is generated at Output.

After the first clock cycle (Clock 0), the input word at $I2$ is A_{29}, the output of M1 is $D_0 = A_{29}B$. Since the output of M4 is 0, the output of M2 is $C_0 = D_0$ and the output of M3 is C_0x^8.

After the second clock cycle (Clock 1), the input word at $I2$ is A_{28} and the output of M1 is given by $D_1 = A_{28}B$. The adder M2 takes the inputs D_1 and C_0x^8, where C_0x^8 was stored in the register M4 in the previous clock cycle, and yields $C_1 = C_0x^8 + D_1$.

$\vdots$

This process is continued till the $30th$ clock cycle (Clock 29), the input word at $I2$ is A_0 and the output of M1 is given by $D_{29} = A_0B$. The adder M2 takes the inputs D_{29} and $C_{28}x^8$, where $C_{28}x^8$ was stored in the register M4 in the previous clock cycle, and yields $C_{29} = C_{28}x^8 + D_{29}$. This is the exact product C from equation (3.7).

Thus one finite field multiplication in $GF(2^{233})$ needs 30 clock cycles in this BPWS PB finite field multiplier.

Assume that B is available at $I1$ throughout the multiplication. Table 3.2 shows the main intermediate results after each clock cycle..

Clock	Input at $I2$	Output of M1	Output of M4	Output
0	A_{29}	$D_0 = A_{29}B$	0	$C_0 = D_0$
1	A_{28}	$D_1 = A_{28}B$	C_0x^8	$C_1 = C_0x^8 + D_1$
2	A_{27}	$D_2 = A_{27}B$	C_1x^8	$C_2 = C_1x^8 + D_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
29	A_0	$D_{29} = A_0B$	$C_{28}x^8$	$C_{29} = C_{28}x^8 + D_{29}$

Table 3.2: The output and intermediate results upon each clock cycle

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.3 M3: Constant Finite Field Multiplier $Z = x^8 Y$

In general, we call a finite field multiplier a constant finite field multiplier when one of two operands is a constant field element. The constant finite field multiplier has a much simpler architecture than a regular multiplier since it removes all the AND gates and significantly reduces the number of XOR gates.

The module M3 in Figure 3.1 is a constant finite field multiplier which performs the multiplication of $Z = x^8 Y$ in $GF(2^{233})$, where x^8 is a constant and Y is any element in $GF(2^{233})$. This constant finite field multiplier can be simply implemented by only 8 XOR gates.

If we express Y and Z as

$$\begin{aligned} Y &= \sum_{i=0}^{232} y_i x^i, \quad \text{and} \\ Z &= \sum_{i=0}^{232} z_i x^i, \end{aligned}$$

the coefficients z_i of the product Z can be expressed as,

$$z_i = \begin{cases} y_{225+i} & \text{i=0,1,...,7} \\ y_{i-8} & \text{i=8,9,...,73} \\ y_{i-8} + y_{151+i} & \text{i=74,75,...,81} \\ y_{i-8} & \text{i=82,83,...,232} \end{cases} \tag{3.8}$$

The architecture of the constant finite field multiplier M3 is shown in Figure 3.2.

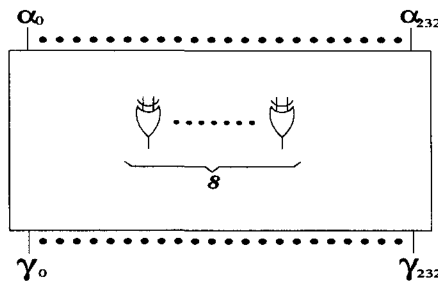

Figure 3.2: M3: The constant finite field multiplier $Z = x^8 Y$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.4 M1: 8×233 Bit-parallel partial product generator

Module M1 in the proposed BPWS PB finite field multiplier is a 8×233 partial product generator which performs the function of A_jB, where B is the 233-bit operand and A_j is the input word of 8 bits. Note that A_j can be viewed as a field element in $GF(2^{233})$ with the most significant 225 bits being 0s, i.e.,

$$A_j = a_7x^7 + a_6x^6 + \ldots + a_0 \ , \tag{3.9}$$

where $a_7, a_6, \ldots, a_0 \in GF(2)$. Thus,

$$\begin{aligned} A_jB &= (a_7x^7 + a_6x^6 + \ldots + a_0)B \\ &= a_7x^7B + a_6x^6B + \ldots + a_0B \ . \end{aligned} \tag{3.10}$$

In this expression, $x^7B, x^6B, \ldots, xB$ are seven constant finite field multipliers. Each result from the seven constant finite field multipliers is multiplied by the coefficient a_i correspondingly, this step can be done by an AND network which is introduced later. Finally, the accumulation of the eight results can be simply obtained from an XOR network.

This 8×233 partial product generator can be implemented by the following full bit-parallel architecture shown in Figure 3.3.

The seven constant finite field multipliers have the similar architectures as the module M3 in the proposed BPWS finite field multiplier in Figure 3.1.

Let $Z = x^wY$ be the constant multiplier, where $w = 1, 2, \ldots, 7$, such constant multiplier needs w XOR gates. The architecture of the constant finite field multiplier $Z = x^wY$ is shown in Figure 3.4.

In the 8×233 bit-parallel partial product generator architecture shown in Figure 3.3, the AND networks are used to multiply a 233-bit field element by the coefficient a_i and can be implemented by 233 AND gates. The architecture is shown in Figure 3.5.

The outputs from eight AND networks are accumulated by an XOR network as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

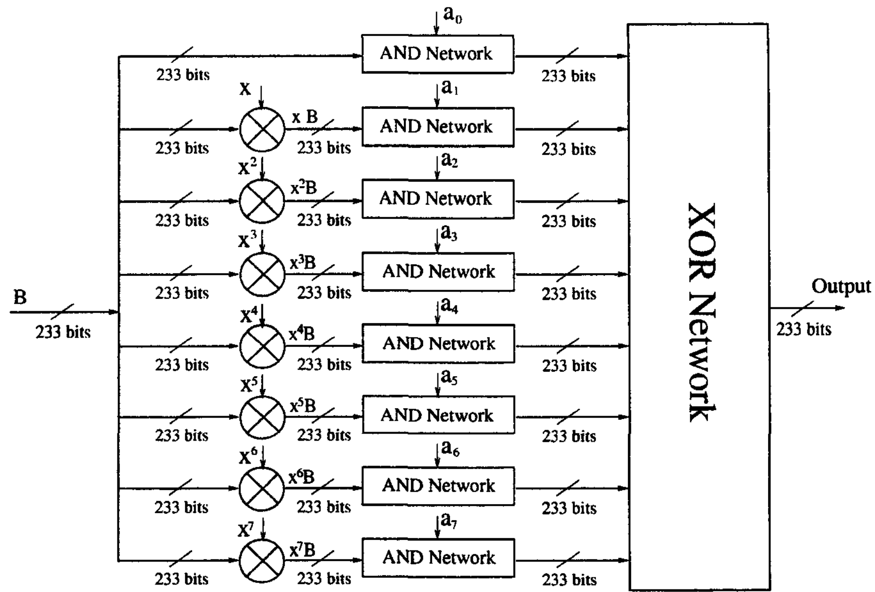

Figure 3.3: 8×233 bit-parallel partial product generator

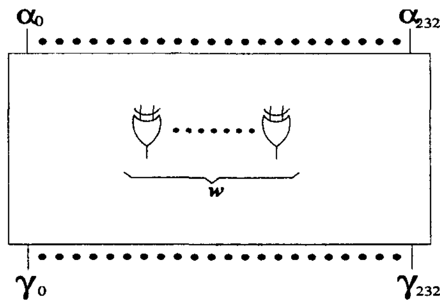

Figure 3.4: The architecture of the general constant finite field multiplier $Z = x^w Y$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

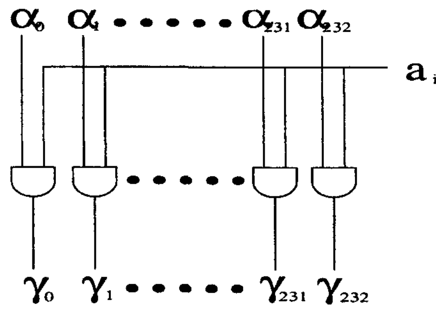

Figure 3.5: The architecture of AND network

shown in Figure 3.6.

In Figure 3.6, the module M is sub XOR network which can be implemented by 233

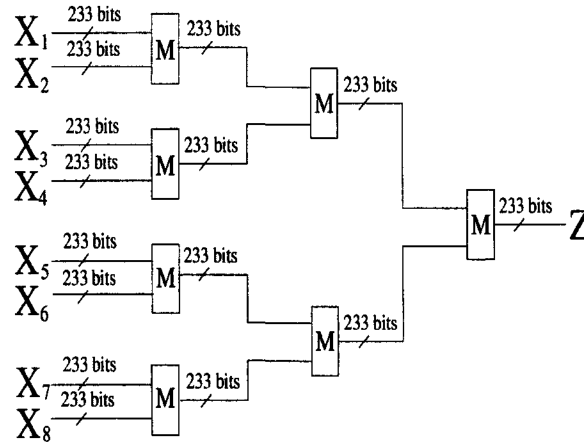

Figure 3.6: The architecture of XOR network

XOR gates. The architecture is shown in Figure 3.7.

Assume all AND and XOR gates have only two inputs, the delay of AND gate is T_A, the delay of XOR gate is T_X, the circuit complexity and timing complexity of the 8×233 bit-parallel partial product generator are summarized in the Table 3.3.

The circuit complexity and timing complexity of the BPWS PB finite field multiplier in $GF(2^{233})$ are summarized in Table 3.4.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

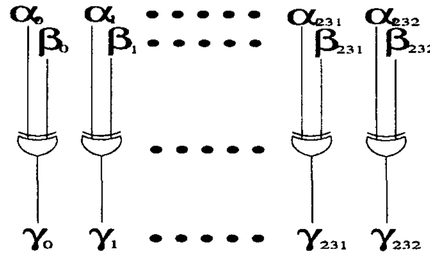

Figure 3.7: The architecture of sub XOR network

# of AND gates	$8 * 233$
# of XOR gates	$7 * 233 + (1 + 7) * 7/2$
Critical path	$T_A + 4T_X$

Table 3.3: Circuit and timing complexities of the 8×233 partial product generator

# of AND gates	$8 * 233$
# of XOR gates	$8 * 233 + (1 + 8) * 8/2$
# of 233-bit registers	1
Critical path	$T_A + 6T_X$

Table 3.4: Circuit and timing complexities of the BPWS PB finite field multiplier

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.3 Alternative BPWS PB finite field multiplier

As defined in Section 3.2, There is an alternative form of architecture for the above BPWS PB finite field multiplier.

we still divide one operand A into 30 words from the least significant bit of A and each word contains 8 bits. In the $30th$ word, we append seven "0"s as the most significant seven bits.

Let A_j denote each word, A_j can be expressed as

$$A_j = a_{8j} + a_{8j+1}x + a_{8j+2}x^2 + \ldots + a_{8j+7}x^7 \ ,$$

where $j = 0, 1, 2, \ldots, 29$ and $a_i = 0$ for $i = 233, 234, \ldots, 239$.

Then A can be rewritten as

$$A = A_0 + A_1x^8 + A_2(x^8)^2 + \ldots + A_{29}(x^8)^{29} \ . \quad (3.11)$$

Thus the product $C = AB \bmod F(x)$ can be expressed as follows,

$$\begin{aligned} C &= AB \bmod F(x) \\ &= (A_0 + A_1x^8 + A_2(x^8)^2 + \ldots + A_{29}(x^8)^{29})B \bmod F(x) \\ &= A_0B + A_1Bx^8 + A_2B(x^8)^2 + \ldots + A_{29}B(x^8)^{29} \bmod F(x) \end{aligned} \quad (3.12)$$

We can further let

$$D_j = D_{j-1}x^8 \ , \quad (3.13)$$

for $j = 1, 2, \ldots, 29$, and $D_0 = B$;

$$C_j = C_{j-1} + A_jD_j \ , \quad (3.14)$$

for $j = 0, 1, \ldots, 29$, and $C_{-1} = 0$.

From Equations 3.13, 3.14 and 3.12, we will find that the product $C = AB \bmod F(x)$ is

$$C = C_{29} \ . \quad (3.15)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The above alternative BPWS PB finite field multiplier can be implemented by the architecture shown in Figure 3.8.

This multiplier architecture has two input ports and one output port. One input

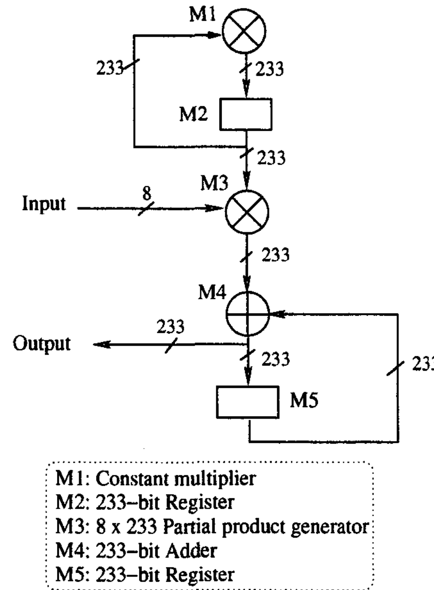

Figure 3.8: Alternative BPWS PB finite field multiplier over $GF(2^{233})$

port is 8 bits wide and is used to serially input the word A_j which is part of the operand A, the other input port is 233 bits wide and used to input the other operand B in parallel. The output port is 233 bits wide which is used for the output product. There are five modules in this alternative multiplier architecture, which are given below.

Module M1 : Module M1 is a constant finite field multiplier which performs the same function as Module M3 described in Section 3.2.

Module M2 : Module M2 is a 233-bit register which has the initial value of $B(x)$ and store the intermediate results. Let the output of M2 be D_j, and the function that the module M1 and M2 perform be $D_j = D_{j-1}x^8$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Module M3 : Module M3 is a 8x233 bit-parallel PB finite field multiplier which is the same as Module M1 described in 3.1. One input A_j is the word from the operand A, the other input is D_j, the output of the module M3 is $A_j D_j$.

Module M4 : Module M4 is an adder which has the same function as Module M2 described in 3.1. The output of module M4 is also the output of the alternative BPWS PB finite field multiplier.

Module M5 : Module M5 is another 233-bit register which is used to keep the intermediate results. Let the output of module M4 be C_j, then the function that the module M4 and M5 perform is $C_j = C_{j-1} + A_j D_j$.

Let the initial value of M2 be B, the initial value of M5 be 0, the word input method used is the least significant word (LSW) first.

During the first clock cycle (Clock 0), the input word is A_0, the output of module M2 is B which is D_0, the output of module M3 is $A_0 B$ which is $A_0 D_0$, the output module M4 is $A_0 D_0$ which is C_0 since the initial value of M5 is 0 which is mapped by $C_{-1} = 0$;

During the second clock cycle (Clock 1), the output of module M2 is Bx^8 which is D_1, the input word is A_1, the output of module M3 is $A_1 D_1$, the output of module of M5 is C_0, therefore the output of module M4 is $C_0 + A_1 D_1$ which is C_1;

During the third clock cycle (Clock 2), the output of module M2 is $B(x^8)^2$ which is $D_2 = D_1 x^8$, the input word is A_2, the output of module M3 is $A_2 D_2$, the output of module M5 is C_1, therefore the output of module M4 is $C_1 + A_2 D_2$ which is C_2;

$\vdots$

During the $30th$ clock cycle (Clock 29), the output of module M2 is $D_{29} = D_{28} x^8$, the input word is A_{29}, the output of module M3 is $A_{29} D_{29}$, the output of module M5 is C_{28}, therefore the output of M4 is $C_{28} + A_{29} D_{29}$ which is C_{29}.

The product $C = AB \bmod F(x)$ can be obtained from the output of module M4

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Clock	Input	M2	M5	Output
0	A_0	$D_0 = B$	0	$C_0 = A_0 B$
1	A_1	$D_1 = D_0 x^8$	C_0	$C_1 = C_0 + A_1 D_1$
2	A_2	$D_2 = D_1 x^8$	C_1	$C_2 = C_1 + A_1 D_1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
29	A_{29}	$D_{29} = D_{28} x^8$	C_{28}	$C_{29} = C_{28} + A_{29} D_{29}$

Table 3.5: The values of output and other modules on each clock cycle

during the $30th$ clock cycle. Table 3.5 shows the values of input words, M2, M5 and output of the alternative BPWS PB finite field multiplier upon each clock cycle.

The 8×233 bit-parallel partial product generator, the constant multiplier, the adder and registers are the same as those described in Section 3.2. Using this architecture 30 clock cycles are needed in order to perform one multiplication. With the same assumption in Section 3.2, The circuit complexity and timing complexity for this alternative BPWS PB finite field multiplier are summarized in Table 3.6.

# of AND gates	$8 * 233$
# of XOR gates	$8 * 233 + (1 + 8) * 8/2$
# of 233-bit registers	2
Critical path	$T_A + 5T_X$

Table 3.6: The circuit and timing complexities of alternative BPWS PB finite field multiplier

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.4 General BPWS PB finite field multipliers

General BPWS PB finite field multipliers in $GF(2^m)$ can be simply derived by extending the proposed BPWS PB finite field multiplier described in Section 3.2 and Section 3.3. Assume the finite field is $GF(2^m)$ and the size of the input word A_j is w, then the corresponding architectures are shown in Figure 3.9 and Figure 3.10.

The modules in the two general architectures serve the similar functions as those

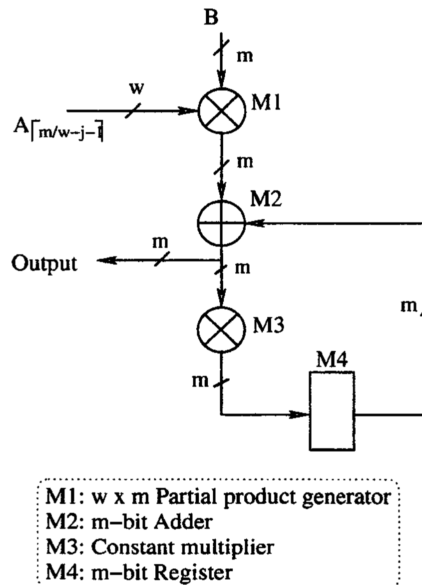

Figure 3.9: The BPWS PB finite field multiplier in $GF(2^m)$

described in Section 3.2 and Section 3.3. It needs $\lceil m/w \rceil$ clock cycles to perform one multiplication in $GF(2^m)$ using these general architectures.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

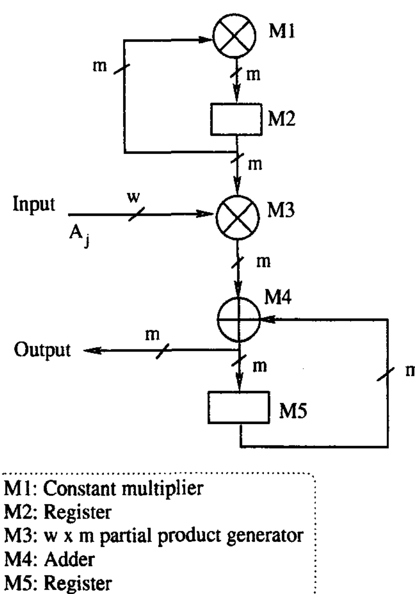

Figure 3.10: The alternative BPWS PB finite field multiplier in $GF(2^m)$

3.5 Comparisons

Assume all AND gates and XOR gates have only 2 inputs and the delays of AND gate and XOR gate are T_A and T_X respectively. When the irreducible polynomial $F(x)$ which generates the finite field $GF(2^m)$ is trinomial, i.e. $F(x) = x^m + x^k + 1$, with $1 < k < m/2$, circuit and timing complexities of bit-parallel, bit-serial and the general BPWS multipliers are shown in Table 3.7.

From Table 3.7 we can see that the numbers of AND gates and XOR gates in proposed BPWS PB finite field multipliers are between those in bit-parallel PB finite field multiplier and those in bit-serial PB finite field multiplier. There is no sequential element needed in bit-parallel finite field multiplier. The number of registers in proposed BPWS PB finite field multipliers is the same as that in bit-serial PB finite field multipliers. The critical path for proposed BPWS PB finite field multipliers is also between that of bit-parallel and bit-serial PB finite field multipliers. The number

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Multiplier	Speed (Clock Cycles)	Circuit complexity	Critical path
bit-parallel [28]	1	m^2 AND gates $m^2 - 1$ XOR gates	$T_A + (\lceil \log_2(m-1) \rceil + 2)T_X$
bit-serial (MSB) [2]	m	m AND gates $m+1$ XOR gates One m-bit register	$T_A + T_X$
bit-serial (LSB) [2]	m	m AND gates $m+1$ XOR gates Two m-bit registers	$T_A + T_X$
Proposed BPWS MSW first	$\lceil (m/w) \rceil$	$w * m$ AND gates $w * m + (1+w) * w/2$ XOR gates One m-bit register	$T_A + (\lceil \log_2 w \rceil + 3)T_X$
Proposed BPWS LSW first	$\lceil (m/w) \rceil$	$w * m$ AND gates $w * m + (1+w) * w/2$ XOR gates Two m-bit registers	$T_A + (\lceil \log_2 w \rceil + 2)T_X$

Table 3.7: The comparisons among bit-parallel, bit-serial and BPWS finite field multipliers

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of clock cycles to perform one finite field multiplication in proposed BPWS PB finite field multiplier is also bigger than that of bit-parallel PB finite field multiplier and smaller than that of bit-serial PB finite field multipliers. This proposed BPWS PB finite field multiplier is the trade-off between bit-parallel finite field multiplier and bit-serial finite field multiplier.

When m is far bigger than w, there is a rough relation between the number of AND gates, the number of XOR gates and the number of clock cycles to perform one multiplication among these multipliers, which is that the products of speed (in clock cycles) and circuit complexities for these multipliers are approximately same, i.e., for AND gates:

$$1 * m^2(\text{bit parallel}) = m * m(\text{bit serial}) \approx \lceil (m/w) \rceil * w * m \ , \tag{3.16}$$

for XOR gates:

$$1 * (m^2 - 1)(\text{bit parallel}) \approx m * (m+1)(\text{bit serial}) \approx \lceil (m/w) \rceil * (w * m + (1+w) * w/2) \ . \tag{3.17}$$

Carefully choosing the number w, the proposed BPWS PB finite field multipliers can achieve the desired trade-off between area and speed.

Now we can have further analysis conducted using this word size w,

$w = m$, this actually is a full bit parallel architecture, which can be simplified to the bit parallel PB finite field multiplier [28] by removing all other modules except for the module of $w \times m$ finite field multiplier.

$w = 1$, this directly becomes a bit-serial multiplier as reported in [2].

Thus, our design algorithm can be treated as a general design algorithm for finite field multiplier.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.6 Summary

In this chapter, we first introduced the finite fields recommended by NIST for EC cryptosystems. After choosing a finite field $GF(2^{233})$ which is generated by an irreducible trinomial $F(x) = x^{233} + x^{74} + 1$, we designed and analyzed the BPWS PB finite field multiplier and the alternative BPWS PB finite field multiplier in Section 3.2 and Section 3.3 respectively. At the end of this chapter, we designed a general form of BPWS PB finite field multiplier which is MSW first multiplier and its alternative form which is LSW first multiplier, we also made the comparisons among the bit-parallel PB finite field multiplier in [28], the bit-serial PB finite field multiplier in [2] and our general BPWS finite field multipliers. The proposed architecture is suitable for the application which requires to balance the trade-off between speed and area, it is extremely useful for smart card applications.

In next chapter, we will design an ASIC chip which is capable of performing the finite field multiplication and squaring using the proposed BPWS PB finite field multiplier described in Section 3.2 and the bit-parallel PB finite field squarer described in [28].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Hardware Design

The final aim of this thesis is to design an application specific integrated circuit (ASIC) chip which can perform multiplication and squaring in $GF(2^{233})$. In this chapter, the detail design methodology of such a chip is introduced. Issues during the design are addressed. The multiplication is implemented by applying the proposed BPWS PB finite field multiplier described in Section 3.2. The squaring is achieved by applying the bit-parallel PB finite field squarer described in [28].

4.1 Hardware architecture

The proposed hardware architecture is shown in Figure 4.1.

In this figure, the module **Multiplier** is the BPWS PB finite field multiplier described in Section 3.2, the module **Squarer** is the bit-parallel PB finite field squarer described in [28], the width of the data bus (which is also size of the word in BPWS PB finite field multiplier in Section 3.2) is 8, the width of the address is 5, *clk* is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

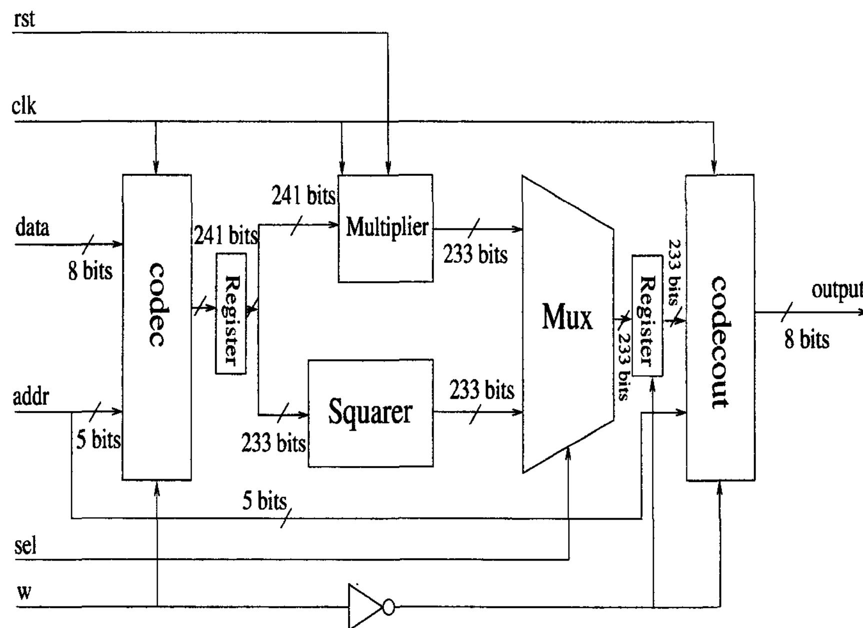

Figure 4.1: The schematic of the hardware

clock signal, *rst*, *sel* and *w* are control signals, the module **Mux** is used to select either finite field multiplication or finite field squaring, the modules of **Register**s are used to store the input operands or the result, the modules of **codec** and **codecout** are used to decode the address to write the input data into or read the data out of the registers. Except for the modules of **Multiplier** and **Squarer**, all others can be modeled in the control part of the data path in a processor.

4.2 Hardware specifications

The specifications for the ASIC chip are summarized in Table 4.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Frequency	50MHz
Area	$300000\mu m^2$
Power consumption	20 mw

Table 4.1: Specifications

4.3 VLSI implementation technology and design flow

CMC (Canada Microelectronic Corporation) supports all Canadian universities with industry level VLSI design tools and technical support. CMC also provides several design flows for different kinds of ASIC designs. The design flow followed in this thesis is the CMC digital design flow. Figure 4.2 shows the CMC digital design flow. The VLSI implementation technology used in this project is TSMC (Taiwan Semiconductor Manufacture Company) $0.18\mu m$ CMOS technology. Compared with TSMC $0.35\mu m$ technology, $0.18\mu m$ technology has the advantages of small area and low power consumption etc.

Digital chip design can be partitioned into front-end design, back-end design and post design verification and modification.

4.4 Front-end design

Front-end design of an ASIC chip includes the tasks of hardware modeling, testbench and stimuli file creations, logic synthesis and design-for-testability (DFT) synthesis etc.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

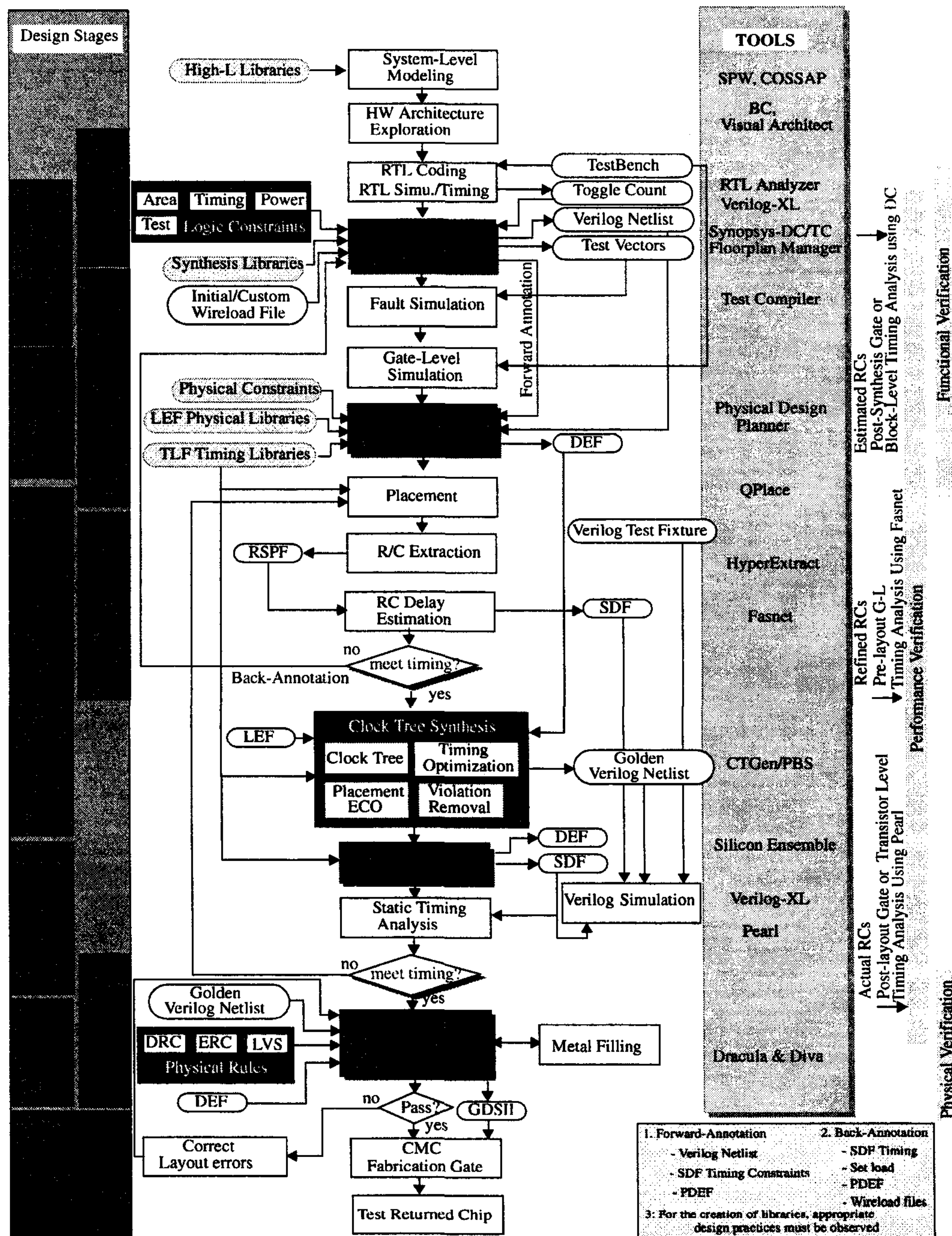

Figure 4.2: CMC digital design flow

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4.1 Stimuli files

In ASIC digital design, we need stimuli files to verify the logic function of the modeled hardware circuits. In this project, a software program is developed using Borland C++ Builder to create the stimuli files. This program is actually a finite field multiplier over $GF(2^{233})$ and can create two kinds of stimuli files to test finite field multiplication and finite field squaring respectively. There are a number of vectors in the stimuli files. In the stimuli file for the finite field multiplication test, each vector contains three 233-bit binary numbers, the first two are the two input operands and the last one is the result used to compare with the output of the modeled circuits; In the stimuli file for the finite field squaring test, each vector contains two 233-bit binary numbers, the first is the input operand and the last is the squaring. The number of the vectors is 1000 in this program.

4.4.2 Hardware modeling

The circuit shown in Figure 4.1 and its modules are modeled in Verilog which is an industry level hardware description language (HDL). The circuit is modeled at register transfer level (RTL). In order to verify that the circuit perform the desire logic, two testbenches which are written in Verilog to perform functional finite field multiplication and finite field squaring tests are also needed. The detail Verilog files are listed in Appendix A and Appendix B. The Verilog simulation tool used in this thesis is **Verilog-XL**. In the testbenches, the stimuli files which are created by the software are used to exercise the circuit, no functional error has occurred during simulation. The modeled circuit performs the desired logic.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4.3 Logical synthesis

The Verilog files that we modeled at RTL level to describe the behavior of the hardware circuit are also called **RTL netlists**, while the physical layout design needs gate level netlists. The task of logic synthesis is to convert the RTL netlists to gate netlists. The cells in the gate netlists are referenced by cell libraries (target libraries) which are provided by ASIC vendors. The logic synthesizer is a software to perform the logic synthesis task. The logic synthesizer used to perform logical synthesis in this project is Design Compiler from Synopsys, and the target libraries used here are TSMC 0.18 micron CMOS technology libraries. The Table 4.2 summarizes the results of logic synthesis.

Circuit/module	# of Cells	Cell Area (μm^2)	# of equivalent gates
BPWS multiplier core	3029	124936.062500	4893
Squarer core	293	4248.527832	293
Whole circuit	4539	495452.468750	12113

Table 4.2: Results of logic synthesis

4.4.4 DFT synthesis

There are two types of test in our project. One is functional test which verifies that the circuit performs the correct logic as expected. The other is manufacturing test which verifies that the circuit does not have manufacturing defects by focusing on circuit structure rather than functional behavior. Manufacturing defects might remain undetected by functional testing yet cause undesirable behavior during circuit operation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Design for testability (DFT)

DFT is a manufacturing test technique that we can adopt to thoroughly test our integrated circuit. Detailed introduction of DFT can be found in [30] and other relevant books.

In this project, the DFT design technique is **internal full scan**. The tool which is used for DFT synthesis is DFT Compiler from Synopsys, the ATPG tool which is used to create test patterns and to perform fault simulation is TetraMax from Synopsys. In our design, all sequential cells are all valid, no violation exists. The number of the test patterns is 154 and the fault coverage is 100%.

4.5 Back-end design

In this section, we start the physical IC layout design. Back-end design includes the tasks of floorplanning, placement, clock synthesis and routing.

4.5.1 Floorplanning and Placement

Floorplanning

The objectives of floorplanning are to minimize the chip area and minimize delay. The input to a floorplanning tool is the gate netlist that describes the modeled circuit. The gate netlist in our design is the output from logical synthesis and DFT synthesis which is a logical description of the ASIC. The floorplan is a physical description of an ASIC. Floorplanning is thus a mapping between the logical description (the netlist) and the physical description (the floorplan).

The tasks of floorplanning are to

- arrange the blocks on a chip,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- decide the location of the I/O pads,
- decide the location and number of the power pads,
- decide the type of power distribution, and

In our design, the gate netlist does not contain any blocks. In TSMC 0.18μm technology, the power supplies for I/O ring and core cells are different. There are four pairs of power pads added in our design, two pairs are used for I/O ring power supply, two pairs are used for the core power supply. The aspect ratio is set to 1 which means the shape of the chip is square. A pair of power ring is placed around the core which contains all the standard cells and three pairs of vertical power strips are placed across the core.

Placement

After completing a floorplan we can begin placement of the logic cells. The objectives of placement are to

- guarantee the router can complete the routing step,
- minimize all the critical net delays,
- the chip as dense as possible,
- minimize the total estimated interconnect length,
- meet the timing requirements for the critical nets,
- minimize the interconnect congestion.

Compare with floorplanning, placement is more suitable for automation.
Our design is row-based ASIC design. All logic cells are placed in rows which are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

defined in the floorplanning and placement tool. Rows are separated by channels which are used for horizontal routing. Figure 4.3 illustrates that the logic cells are placed into rows. Carefully selecting the channel offset, we can avoid the later design rule checking (DRC) problems. In our design, the row utilization is set to 90%, the channel offset of 4*0.56 micron is obtained from many experiments and other designers' experience.

In our design, the tool used to perform floorplanning and placement is Physical

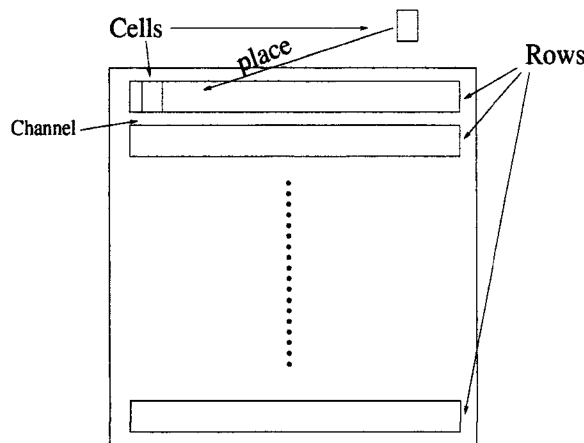

Figure 4.3: Illustration of placement

Design Planner (or called AreaPdp) which is a Cadence tool. Placement is done by the timing driven Qplace tool which is integrated in AreaPdp. Timing constraint files which are obtained during logic synthesis are fed into AreaPdp as constraint files. In our design, timing requirements are met and there is no congestions in the geometry report.

4.5.2 Clock tree synthesis

The major task of clock tree synthesis is developing the interconnect geometry that connects the clock to all the cells on the chip that use a clock. These cells consist of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

latches, flip-flops, and other logic elements that are needed to synchronize with the system clock. In this thesis, clock tree synthesis is done in CTGen which is integrated with First Encounter (FE) Ultra, a Cadence tool. The generated clock tree meets the timing requirements and 23 buffers are added to balance the clock tree in our design.

4.5.3 Golden netlist

The gate netlist generated from Design compiler has been modified at stages of DFT compiler by replacing all sequential cell with corresponding scanned enable sequential cells and clock tree synthesis by adding buffers into it. This modified gate netlist should perform the same logic as the RTL netlist. Before doing routing, we should run the functional test again to verify if this modified netlist perform the desire logic. The test tool is still Verilog-XL. Without considering the timing requirement, the functional test at this step is the last time to verify if the circuit can perform the desire logic. Any failure in functional test will result in the iterations of floorplan, placement and clock tree synthesis. The results of functional multiplication test are shown in Figure 4.4 and Figure 4.5. The functional squaring test results are shown in Figure 4.6 and Figure 4.7.

There is no functional error during simulation and this modified gate netlist is called golden netlist which can also be used in later LVS checking.

4.5.4 Routing

After the chip is floorplanned and the logic cells have been placed, it is time to make the connections by routing the chip. Routing is usually split into global routing followed by detailed routing. In this project, the tool (router) which is used to perform routing is Silicon Ensemble which is a Cadence tool. After detailed routing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

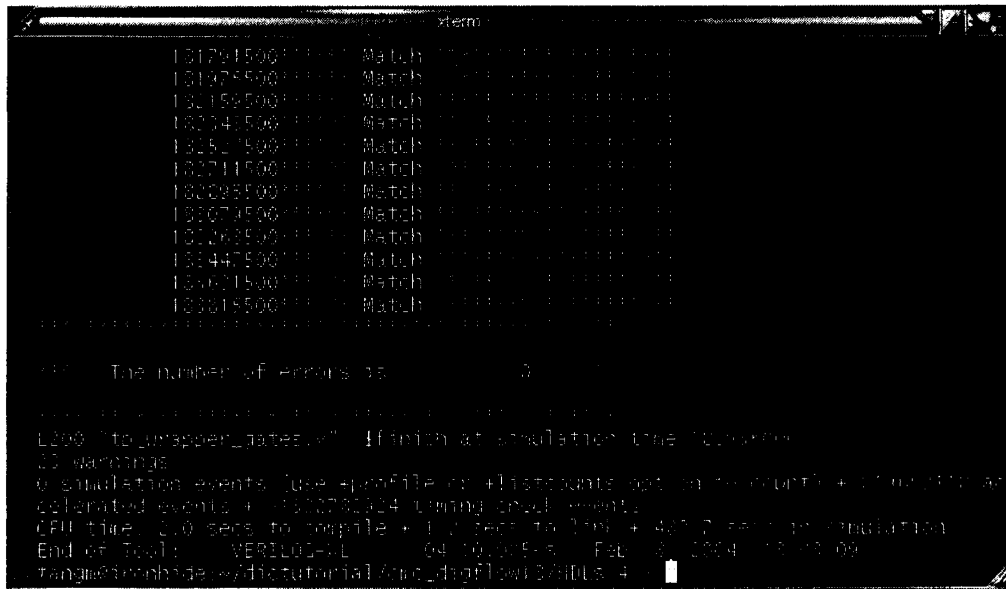

Figure 4.4: Functional multiplication test

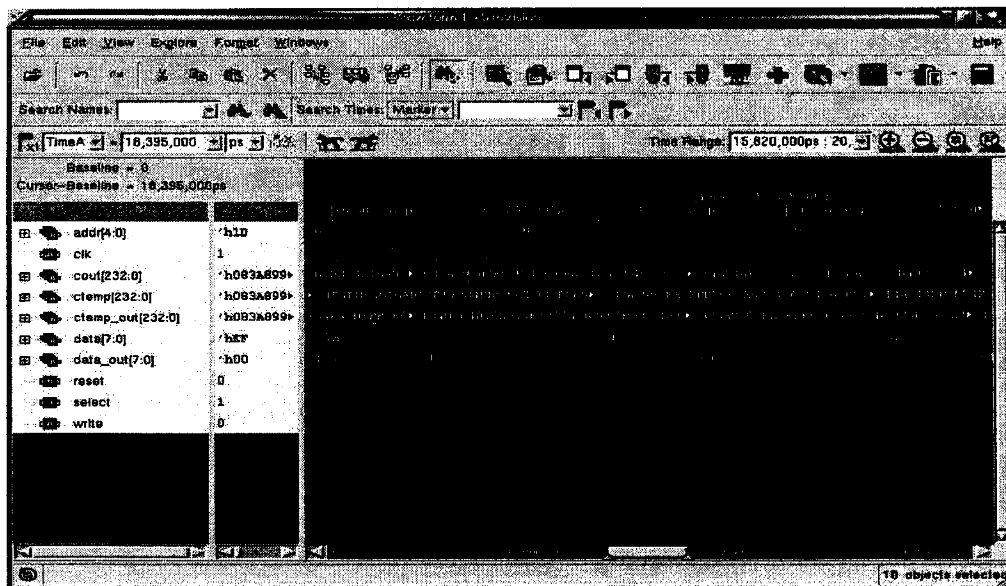

Figure 4.5: Waveform of functional multiplication test

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.6: Functional squaring test

Figure 4.7: Waveform of functional squaring test

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is complete, the exact length and position of each interconnect for every net is known and the parasitic capacitance and resistance associated with each interconnect, via, and contact can be calculated by RC extraction tool. Interconnect delay and load due to parasitic resistance and capacitance are written in regular standard parasitic format (RSPF) format and static timing analysis is done in perl scripts in our design. The timing analysis at this step is the last time to verify timing. Any failure to meet the timing requirements will result in the iterations of floorplanning, placement and routing until the timing requirements are thoroughly met. The result of timing limit checking is shown in Figure 4.8.

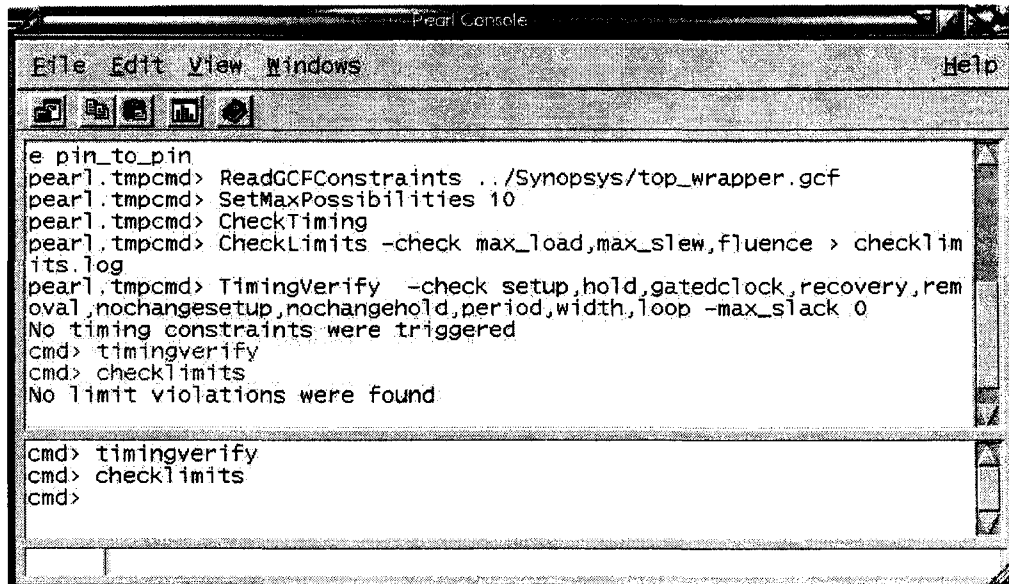

Figure 4.8: Timing limit checking

Our design meets the timing requirements and there is no timing violation in our design.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.6 Physical verification and modification

After detail routing is complete and timing analysis shows that the design meet the timing requirement, we can perform physical verification and even modification which is usually needed before the chip is fabricated. There are two major kinds of checking, layout versus schematic (LVS) and design rule checking (DRC).

4.6.1 Layout versus schematic (LVS)

The timing analysis we perform after routing just shows whether the design meets the timing requirements. The netlist might be modified during routing. One of our concerns is if the physical layout after routing performs the same logic as the golden netlist. LVS essentially compares the physical netlist (the netlist after routing) to the golden reference (golden netlist) to ensure that what is about to be committed to silicon is what is really wanted.
In our design, the tool to perform LVS is Diva LVS from Cadence which is integrated in Cadence Design Framework II (dfII). Diva LVS in dfII is used to compare:

1. the final layout in the form of a DEF (Design Exchange Format) file created from Silicon Ensemble after routing to

2. the golden netlist

to verify the physical (Placed & Routed) version of the design contains the same instances, nets, and connectivity as the verified "golden" netlist. The LVS result is shown in Figure 4.9.

In our design, the layout and schematic match each other. Since the physical layout meets the timing requirements, now we know the physical layout can perform the desire logic.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
The net-lists match.

                        layout   schematic
                          instances
        un-matched          0         0
        rewired             0         0
        size errors         0         0
        pruned              0         0
        active              4622      4622
        total               4622      4622

                            nets
        un-matched          0         0
        merged              0         0
        pruned              0         0
        active              4651      4651
        total               4651      4651

                          terminals
        un-matched          0         0
        matched but
        different type      0         0
        total               0         37
End comparison:      Feb  20 10:34:32 2004

Comparison program completed successfully.
```

Figure 4.9: The result of LVS

4.6.2 Design rule checking (DRC)

DRC ensures that nothing has gone wrong in the process of placing the logic cells and routing.

The DRC may be performed at two levels. Since the detailed router normally works with logic-cell phantoms, the first level of DRC is a phantom level DRC , which checks for shorts, spacing violations, or other design-rule problems between logic cells. This is principally a check of the detailed router. In our design, Dracula DRC which is a cadence tool performs the phantom level DRC. The result from Dracula is shown in Figure 4.10.

There is no any DRC violation during phantom level design rule checking.

If we have access to the real library-cell layouts (sometimes called **hard layout**), we can instantiate the phantom cells and perform a second-level DRC at the transistor level. This is principally a check of the correctness after replacing the library cells

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
File  Tools  Options  CMC Gateway  CMOSP18-Documentation                          Help
executing: drcAntenna(met5AntError (gate (sum gateM5)) (antenna (sum met5A)) (ignore < rule) "V...
executing: gateV5 = geomStamp(gate poly1AntennaV5)
executing: diffV5AntV5 = drcAntenna(v5AntennaV5 (gate (sum diffAntennaV5)) (antenna (sum v5Ante...
executing: v5AntV5 = geomAndNot(v5AntennaV5 diffV5AntV5)
executing: v5AntV5 = geomStamp(v5AntV5 v5AntennaV5)
executing: v5AntError = geomCat(gateV5 v5AntennaV5)
executing: drcAntenna(v5AntError (gate (sum gateV5)) (antenna (sum v5AntV5)) (ignore < rule) "V...
DRC started......Mon Feb  9 15:57:45 2004
   completed ....Mon Feb  9 16:05:22 2004
   CPU TIME = 00:07:02  TOTAL TIME = 00:07:37
*********   Summary of rule violations for cell "top_wrapper layout"   *********
  Total errors found: 0
t
t

mouse L: prsMouseSelPt()          M: mousePopUp()          R: ivHiDRC()
>
```

Figure 4.10: The result of phantom level DRC

with detail cell layouts. Since we don't have the detail library-cell layouts due to the confidentiality of TSMC technology, CMC will perform this check as a type of incoming inspection. The results of the second-level DRC from CMC are shown in Figure 4.11 and Figure 4.12.

There is no any DRC violation in this design. Now our design passed the CMC inspection and is ready for fabrication. The design name is *ICFWRWTK*, the fabrication run code is *0402CF*. The chip is expected to return on Oct. 2004.

4.7 Chip Layout

The layout of the chip is shown in Figure 4.13.

The total die size of chip is $2533190.5\mu m^2$ including pads. There are 39 pads in the chip. The hardware parameters are summarized in Table 4.3. From this table, we

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

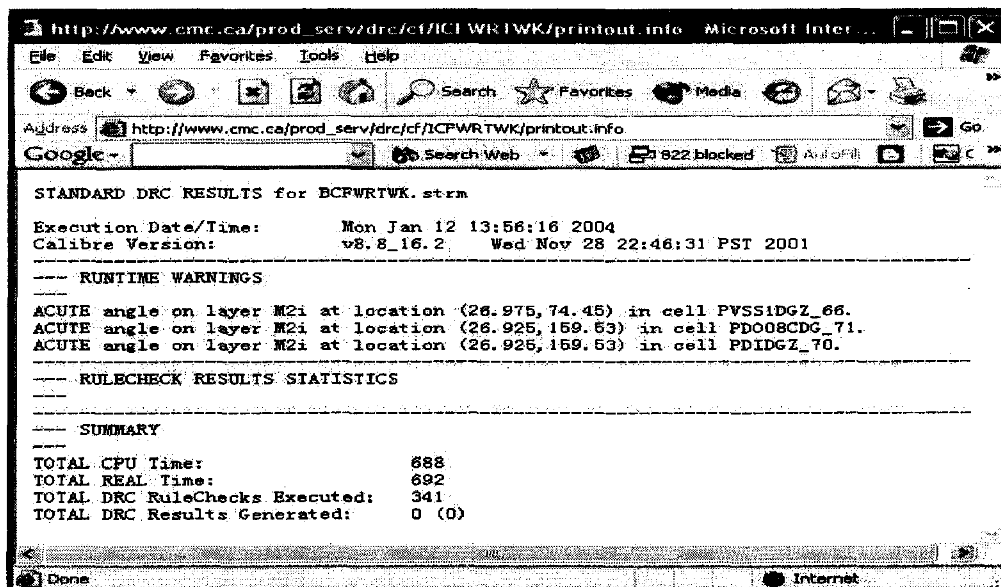

Figure 4.11: The result of standard DRC from CMC

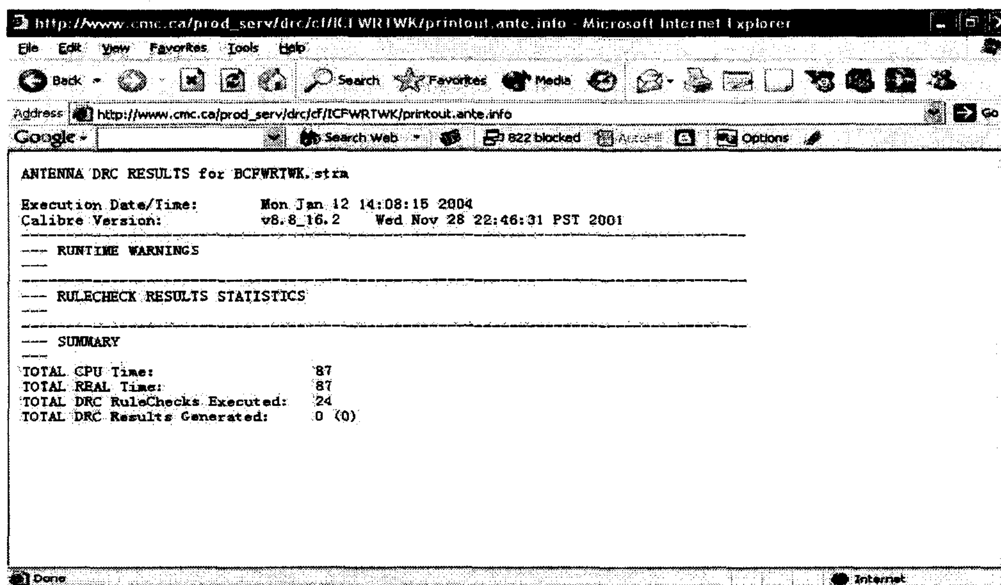

Figure 4.12: The result of anntenna DRC from CMC

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

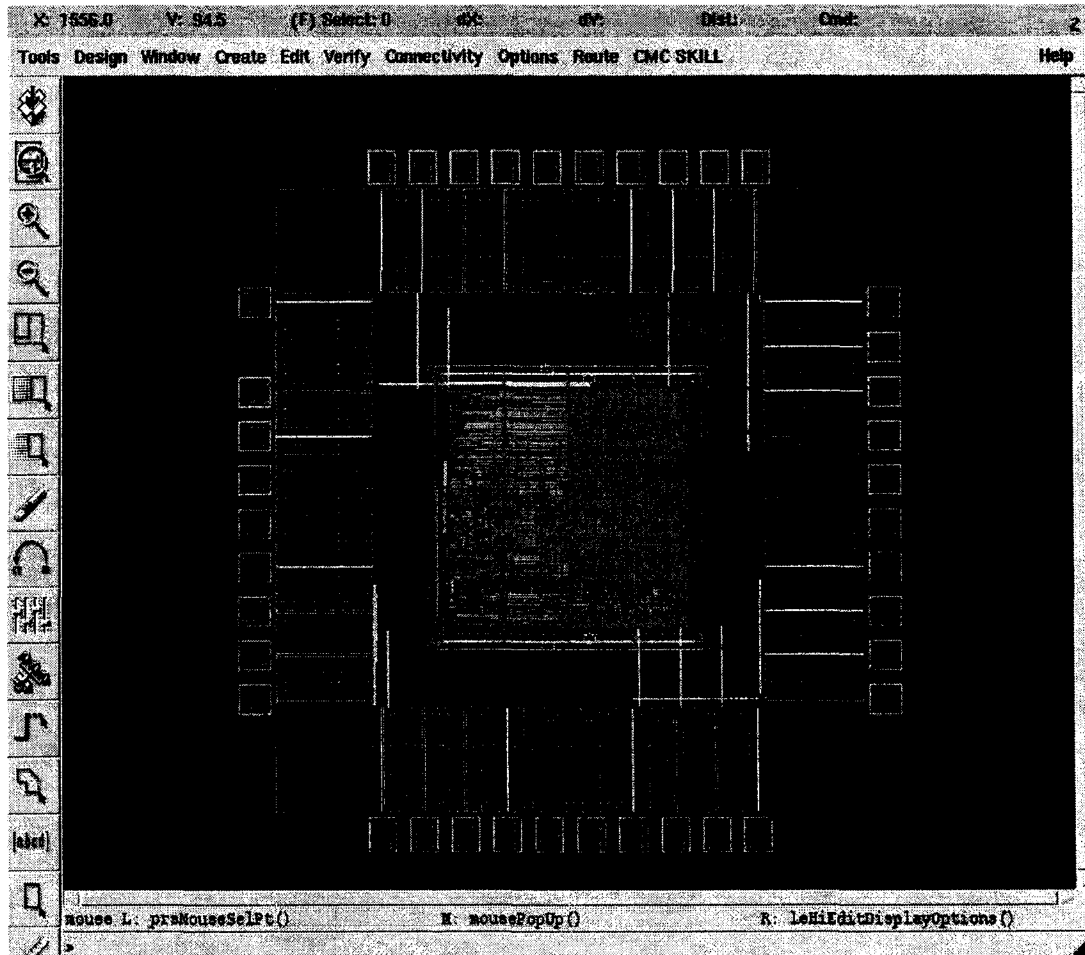

Figure 4.13: The layout of the chip

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can see that our designed chip meets the design specifications.

Specification	BPWS core	Squarer core	Whole circuit
# of cells	3029	293	4570
# of equivalent gates	4893	293	12154
Area (μm^2)	189297.06439	6437.15746	2533190.5
Power consumption (mw)	< 9.2178		31.7421
Frequency (MHz)	50 (max. 130)		

Table 4.3: The hardware parameters

4.8 Comparisons

It is known that the finite field multiplier is a key component in an EC security processor. The comparisons among the designed chip and other VLSI implementation of finite field multipliers are made in Table 4.4.

Even though the frequency is set to 50 MHz during the chip design, the maximum frequency that the chip can work on is 130 MHz. For our designed chip, there is only one clock cycle needed to perform one finite field squaring, while for all other multipliers, the number of clock cycles to perform one finite field squaring is the same as that to perform one finite field multiplication. From Table 4.4 we can see that our design meets the design saves hardware resources.

4.9 Summary

The full hardware design methodology is introduced in this chapter. The CMC digital design flow is followed, the VLSI technology adopted is TSMC 0.18μm CMOS

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Multiplier	Frequency (MHz)	Field size	# of Cells	Gate counts	VLSI technology
BPWS 8x233	50 (max.130)	2^{233}	3029	4893	TSMC 0.18μm CMOS
Squarer			293	293	
Classical 233 x 233 [7]	77	2^{233}	37296 LUTs 37552 FFs	528427	Xilinx FPGA XC2V6000-ff1517-4
Hans et al MSD 64 x 256 [5]	66.4	$\leq 2^{256}$	14797 LUTs 2948 FFs	136064	Xilinx FPGA Virtex-II XCV2000E-7
Souichi et al 8 x 288 [18]	3	$\leq 2^{576}$	$2 * 8 * 288$ ANDs $2 * 8 * 288$ XORs $3 * (8 + 288)$ FFs	14544	ALTERA FPGA EPF10K250-AGC5992

Table 4.4: Comparisons among VLSI implementation of finite field multipliers

technology. There is no any error existed in the final layout. The design name is *ICFWRWTK*, the fabrication run code is *0402CF*. The chip is expected to return on Oct. 2004. The designed chip is ready for fabrication.

In next chapter, my contributions and the expected future works are summarized.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Summaries of Contributions

The summaries of my contributions are:

- Two BPWS PB finite field multipliers in $GF(2^{233})$ are designed in this thesis and the proposed BPWS PB finite field multipliers have the trade-off between area and speed.
- The maximum frequency that the designed chip can work is 130 Mhz. The area of the BPWS multiplier is $189297.06439 \mu m^2$, the power consumption is less than 9.2178 mw. These results meet the design specifications.
- Compared with other finite field multipliers in Table 4.4, the proposed BPWS finite field multiplier saves the hardware resource.
- A novel 8×233 bit-parallel partial product generator is designed in this thesis.

The expected future work is to design an EC security processor for smart card using this proposed BPWS finite field multiplier.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A

Program 1

```
/*************************************************************
***                   top_wrapper.v                       ***
***                   Wenkai Tang                         ***
***                   Aug. 16 2003                        ***
*************************************************************/
// The circuit hierachy includes two level. The top_wrapper.v is
// the top level description, it defines the I/O pad cells.

'timescale 1ns/10ps module
 top_wrapper(data,addr,clk,write,reset,select,data_out,
             wclk,rstclk,test_se,test_in, test_in1,test_in2);
     input[7:0] data;
     input[4:0] addr;
     input test_se, test_in, test_in1,test_in2;
     input clk,write,reset,select,wclk,rstclk;
     output[7:0] data_out;
     wire [7:0] data_top,data_out_top;
     wire [4:0] addr_top;
     wire clk_top,write_top,reset_top,select_top,wclk_top,
          rstclk_top,test_se_top,test_in_top,test_in1_top,
          test_in2_top;

  top_sys topsys (data_top,addr_top,clk_top,write_top,reset_top,
                select_top,data_out_top,wclk_top,rstclk_top,
                test_se_top,test_in_top,test_in1_top,test_in2_top);

  PD008CDG pdata_out00 ( .PAD(data_out[0]), .I(data_out_top[0]) );
  PD008CDG pdata_out01 ( .PAD(data_out[1]), .I(data_out_top[1]) );
  PD008CDG pdata_out02 ( .PAD(data_out[2]), .I(data_out_top[2]) );
  PD008CDG pdata_out03 ( .PAD(data_out[3]), .I(data_out_top[3]) );
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  PD008CDG pdata_out04 ( .PAD(data_out[4]), .I(data_out_top[4]) );
  PD008CDG pdata_out05 ( .PAD(data_out[5]), .I(data_out_top[5]) );
  PD008CDG pdata_out06 ( .PAD(data_out[6]), .I(data_out_top[6]) );
  PD008CDG pdata_out07 ( .PAD(data_out[7]), .I(data_out_top[7]) );
  PDIDGZ pdata00 ( .C(data_top[0]), .PAD(data[0]) );
  PDIDGZ pdata01 ( .C(data_top[1]), .PAD(data[1]) );
  PDIDGZ pdata02 ( .C(data_top[2]), .PAD(data[2]) );
  PDIDGZ pdata03 ( .C(data_top[3]), .PAD(data[3]) );
  PDIDGZ pdata04 ( .C(data_top[4]), .PAD(data[4]) );
  PDIDGZ pdata05 ( .C(data_top[5]), .PAD(data[5]) );
  PDIDGZ pdata06 ( .C(data_top[6]), .PAD(data[6]) );
  PDIDGZ pdata07 ( .C(data_top[7]), .PAD(data[7]) );
  PDIDGZ paddr00 ( .C(addr_top[0]), .PAD(addr[0]) );
  PDIDGZ paddr01 ( .C(addr_top[1]), .PAD(addr[1]) );
  PDIDGZ paddr02 ( .C(addr_top[2]), .PAD(addr[2]) );
  PDIDGZ paddr03 ( .C(addr_top[3]), .PAD(addr[3]) );
  PDIDGZ paddr04 ( .C(addr_top[4]), .PAD(addr[4]) );
  PDIDGZ pclk ( .C(clk_top), .PAD(clk) );
  PDIDGZ pwrite ( .C(write_top), .PAD(write) );
  PDIDGZ pwclk ( .C(wclk_top), .PAD(wclk) );
  PDIDGZ prstclk ( .C(rstclk_top), .PAD(rstclk) );
  PDIDGZ pselect ( .C(select_top), .PAD(select) );
  PDIDGZ preset (.C(reset_top), .PAD(reset) );
  PDIDGZ ptest_se (.C(test_se_top), .PAD(test_se) );
  PDIDGZ ptest_in (.C(test_in_top), .PAD(test_in) );
  PDIDGZ ptest_in1 (.C(test_in1_top), .PAD(test_in1) );
  PDIDGZ ptest_in2 (.C(test_in2_top), .PAD(test_in2) );
endmodule
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix B

Program 2

```
/**************************************************************
***                    topall.v                          ***
***                   Wenkai Tang                        ***
***                   Aug. 16 2003                       ***
**************************************************************/
// The circuit hierachy includes two level. The topall.v is
// the second level description, it models the circuit and still
// contains a hierachy.

'timescale 1ns/10ps module
top_sys(a,addr,clk,w,rst,sel,c,wclk,rstclk,test_se,
              test_in,test_in1,test_in2);
   input [7:0] a;
   input [4:0] addr;
   input clk,wclk,rstclk;
   input w;
   input rst;
   input sel;
   input test_in, test_se, test_in1,test_in2;
   output [7:0] c;
   wire [255:0] w1, w2;
   wire [7:0] w81;
   wire [232:0] w11,w12,w13,w14,w15;

   codec c1(a,addr,clk,w,w1);
   assign w11=w1[240:8];
   assign w81=w1[7:0];
   BPWSMSBFFM b1(w81,w11,clk,rst,rstclk,w12);
   fbp_squarer f1(w11,w13);
   multiplixer2to1 m1(w12,w13,sel,w14);
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    latch233 l1(w14,wclk,1'b0,w15);
    assign w2[232:0]=w15;
    assign w2[255:233]='b0;
    codecout c2(w2,addr,clk,w,c);

endmodule

// Module codec is a component in module top_sys, it decodes address
// and write the data into a part of register.
module codec(data,addr,clk,w,data_out);
    input [7:0] data;
    input [4:0] addr;
    input clk;
    input w;
    output [255:0] data_out;
    reg[255:0] data_out;

    always @(posedge clk)
    begin
       if(w)
          case (addr)
           5'b00000: data_out[7:0]=data;
            5'b00001: data_out[15:8]=data;
            5'b00010: data_out[23:16]=data;
            5'b00011: data_out[31:24]=data;
           5'b00100: data_out[39:32]=data;
            5'b00101: data_out[47:40]=data;
            5'b00110: data_out[55:48]=data;
            5'b00111: data_out[63:56]=data;
           5'b01000: data_out[71:64]=data;
            5'b01001: data_out[79:72]=data;
            5'b01010: data_out[87:80]=data;
            5'b01011: data_out[95:88]=data;
           5'b01100: data_out[103:96]=data;
            5'b01101: data_out[111:104]=data;
            5'b01110: data_out[119:112]=data;
            5'b01111: data_out[127:120]=data;
           5'b10000: data_out[135:128]=data;
            5'b10001: data_out[143:136]=data;
            5'b10010: data_out[151:144]=data;
            5'b10011: data_out[159:152]=data;
           5'b10100: data_out[167:160]=data;
            5'b10101: data_out[175:168]=data;
            5'b10110: data_out[183:176]=data;
            5'b10111: data_out[191:184]=data;
           5'b11000: data_out[199:192]=data;
            5'b11001: data_out[207:200]=data;
            5'b11010: data_out[215:208]=data;
            5'b11011: data_out[223:216]=data;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
           5'b11100: data_out[231:224]=data;
            5'b11101: data_out[239:232]=data;
            5'b11110: data_out[247:240]=data;
            5'b11111: data_out[255:248]=data;
        endcase
    end
endmodule

//Module codecout is a component in module top_sys, it decodes
//the address and read the data out of the register.
module codecout(data,addr,clk,w,data_out);
    input [255:0] data;
    input [4:0] addr;
    input w;
    input clk;
    output [7:0] data_out;
    reg [7:0] data_out;

    always @(posedge clk)
     begin
       if(!w)
          case (addr)
           5'b00000: data_out=data[7:0];
            5'b00001: data_out=data[15:8];
            5'b00010: data_out=data[23:16];
            5'b00011: data_out=data[31:24];
           5'b00100: data_out=data[39:32];
            5'b00101: data_out=data[47:40];
            5'b00110: data_out=data[55:48];
            5'b00111: data_out=data[63:56];
           5'b01000: data_out=data[71:64];
            5'b01001: data_out=data[79:72];
            5'b01010: data_out=data[87:80];
            5'b01011: data_out=data[95:88];
           5'b01100: data_out=data[103:96];
            5'b01101: data_out=data[111:104];
            5'b01110: data_out=data[119:112];
            5'b01111: data_out=data[127:120];
           5'b10000: data_out=data[135:128];
            5'b10001: data_out=data[143:136];
            5'b10010: data_out=data[151:144];
            5'b10011: data_out=data[159:152];
           5'b10100: data_out=data[167:160];
            5'b10101: data_out=data[175:168];
            5'b10110: data_out=data[183:176];
            5'b10111: data_out=data[191:184];
           5'b11000: data_out=data[199:192];
            5'b11001: data_out=data[207:200];
            5'b11010: data_out=data[215:208];
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
            5'b11011: data_out=data[223:216];
           5'b11100: data_out=data[231:224];
            5'b11101: data_out=data[239:232];
            5'b11110: data_out=data[247:240];
            5'b11111: data_out=data[255:248];
        endcase
    end
endmodule

//Module BPWSMSBFFM is a component in module top_sys. It is
//the proposed BPWS PB finite field multiplier.
module BPWSMSBFFM(a,b,clk,rst,rstclk,c);
    input [7:0] a;
    input [232:0] b;
    input clk;
    input rst,rstclk;
    output [232:0] c;
    wire [232:0] w1,w2,w3,w4,w5,w6,w7;
    wire w0;
    assign w0=1'b0;
    latch233 l1 (b,rstclk,w0,w1);//clk
    multiplier8x233 m8x233 (a,w1,w2);
    xor_network xn1 (w2,w5,w3);
    assign c=w3;
    const_multiplier cm8 (w3,w4);
    latch233 l2 (w4,clk,rst,w5);
endmodule

//Module fbp_squarer is a component in module top_sys. It is
//the full bit parallel PB squarer.
module fbp_squarer(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
    integer m;
    integer i;

   always @ (a)
    begin
         k=74;
         m=233;
         for(i=0;i<k;i=i+2)
             b[i]=a[i/2]^a[m-k/2+i/2];
         for(i=1;i<k;i=i+2)
             b[i]=a[(m+i)/2];
         for(i=k;i<2*k;i=i+2)
             b[i]=a[i/2]^a[m-k+i/2];
         for(i=k+1;i<m;i=i+2)
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
            b[i]=a[(m+i)/2]^a[(m-k+i)/2];
        for(i=2*k;i<m;i=i+2)
            b[i]=a[i/2];
   end
endmodule

//Module latch233 is a component in both module top_sys and
//module BPWSMSBFFM. It serve as a 233-bit register.
module latch233(a,clk,rst, q);
    input [232:0] a;
    input clk;
    input rst;

    output [232:0] q;
    reg [232:0] q;
    integer k;

    always @ ( posedge clk) //or posedge rst
     if(rst)
       q=233'b0;
      else
       q=a;
endmodule

//Module multiplixer2to1 is a component in module top_sys.
//It is the multiplixer used to select the output from either
//multiplier or squarer.
module multiplixer2to1(a,b,sel,c);
    input [232:0] a;
    input [232:0] b;
    input sel;
    output [232:0] c;
    reg [232:0] c;

    always@(a or b or sel)
     begin
      if(sel)
        c=a;
      else
        c=b;
     end
endmodule

//Module multiplier8x233 is a component in module BPWSMSBFFM.
//It is the 8x233 partial product generator.
module multiplier8x233(a,b,c);
    input [7:0] a;
    input [232:0] b;
    output [232:0] c;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    wire [232:0] x1,x2,x3,x4,x5,x6,m1,m2,m3,m4,m5,m6,m7,
                 a0,a1,a2,a3,a4,a5,a6,a7;
    const_m1 cm1 (b,m1);
    const_m2 cm2 (b,m2);
    const_m3 cm3 (b,m3);
    const_m4 cm4 (b,m4);
    const_m5 cm5 (b,m5);
    const_m6 cm6 (b,m6);
    const_m7 cm7 (b,m7);

    and_network am0 (a[0],b,a0);
    and_network am1 (a[1],m1,a1);
    and_network am2 (a[2],m2,a2);
    and_network am3 (a[3],m3,a3);
    and_network am4 (a[4],m4,a4);
    and_network am5 (a[5],m5,a5);
    and_network am6 (a[6],m6,a6);
    and_network am7 (a[7],m7,a7);

    xor_network xa1 (a0,a1,x1);
    xor_network xa2 (a2,a3,x2);
    xor_network xa3 (a4,a5,x3);
    xor_network xa4 (a6,a7,x4);
    xor_network xa5 (x1,x2,x5);
    xor_network xa6 (x3,x4,x6);
    xor_network xa7 (x5,x6,c);
endmodule

//Module xor_network is a component in both module BPWSMSBFFM
// and module multiplier8x233. It is the 233-bit adder.
module xor_network(a,b,c);
    input [232:0] a;
    input [232:0] b;
    output [232:0] c;
    reg [232:0] c;
    integer k;

    always @(a or b)
     for (k=0;k<233;k=k+1)
     c[k]=a[k]^b[k];
endmodule

//Module const_multiplier is a component in module BPWSMSBFFM.
//It is the constant multiplier.
module const_multiplier(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    always @(a)
     begin
         b[0]=a[225];
         b[1]=a[226];
         b[2]=a[227];
         b[3]=a[228];
         b[4]=a[229];
         b[5]=a[230];
         b[6]=a[231];
         b[7]=a[232];
         for(k=8;k<74;k=k+1)
             b[k]=a[k-8];
             b[74]=a[225]^a[66];
             b[75]=a[226]^a[67];
             b[76]=a[227]^a[68];
             b[77]=a[228]^a[69];
             b[78]=a[229]^a[70];
             b[79]=a[230]^a[71];
             b[80]=a[231]^a[72];
             b[81]=a[232]^a[73];
         for(k=82;k<233;k=k+1)
             b[k]=a[k-8];
     end
endmodule

//Module const_m1 is a component in module multiplier8x233.
//It is one constant multiplier.
module const_m1(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
    always @(a)
     begin
         b[0]=a[232];
         for(k=1;k<74;k=k+1)
             b[k]=a[k-1];
         b[74]=a[232]^a[73];
         for(k=75;k<233;k=k+1)
             b[k]=a[k-1];
     end
endmodule

//Module const_m2 is a component in module multiplier8x233.
//It is one constant multiplier.
module const_m2(a,b);
    input [232:0] a;
    output [232:0] b;
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    reg [232:0] b;
    integer k;
    always @(a)
     begin
        b[0]=a[231];
        b[1]=a[232];
        for(k=2;k<74;k=k+1)
            b[k]=a[k-2];
        b[74]=a[231]^a[72];
        b[75]=a[232]^a[73];
        for(k=76;k<233;k=k+1)
            b[k]=a[k-2];
     end
endmodule

//Module const_m3 is a component in module multiplier8x233.
//It is one constant multiplier.
module const_m3(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
    always @(a)
     begin
         b[0]=a[230];
         b[1]=a[231];
         b[2]=a[232];
         for(k=3;k<74;k=k+1)
             b[k]=a[k-3];
         b[74]=a[230]^a[71];
         b[75]=a[231]^a[72];
         b[76]=a[232]^a[73];
         for(k=77;k<233;k=k+1)
             b[k]=a[k-3];
     end
endmodule

//Module const_m4 is a component in module multiplier8x233.
//It is one constant multiplier.
module const_m4(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
    always @(a)
     begin
         b[0]=a[229];
         b[1]=a[230];
         b[2]=a[231];
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        b[3]=a[232];
        for(k=4;k<74;k=k+1)
             b[k]=a[k-4];
        b[74]=a[229]^a[70];
        b[75]=a[230]^a[71];
        b[76]=a[231]^a[72];
        b[77]=a[232]^a[73];
        for(k=78;k<233;k=k+1)
             b[k]=a[k-4];
    end
endmodule

//Module const_m5 is a component in module multiplier8x233.
//It is one constant multiplier.
module const_m5(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
    always @(a)
     begin
        b[0]=a[228];
        b[1]=a[229];
        b[2]=a[230];
        b[3]=a[231];
        b[4]=a[232];
        for(k=5;k<74;k=k+1)
            b[k]=a[k-5];
        b[74]=a[228]^a[69];
        b[75]=a[229]^a[70];
        b[76]=a[230]^a[71];
        b[77]=a[231]^a[72];
        b[78]=a[232]^a[73];
        for(k=79;k<233;k=k+1)
            b[k]=a[k-5];
    end
endmodule

//Module const_m6 is a component in module multiplier8x233.
//It is one constant multiplier.
module const_m6(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
    always @(a)
     begin
        b[0]=a[227];
        b[1]=a[228];
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        b[2]=a[229];
        b[3]=a[230];
        b[4]=a[231];
        b[5]=a[232];
        for(k=6;k<74;k=k+1)
            b[k]=a[k-6];
        b[74]=a[227]^a[68];
        b[75]=a[228]^a[69];
        b[76]=a[229]^a[70];
        b[77]=a[230]^a[71];
        b[78]=a[231]^a[72];
        b[79]=a[232]^a[73];
        for(k=80;k<233;k=k+1)
            b[k]=a[k-6];
    end
endmodule

//Module const_m7 is a component in module multiplier8x233.
//It is one constant multiplier.
module const_m7(a,b);
    input [232:0] a;
    output [232:0] b;
    reg [232:0] b;
    integer k;
    always @(a)
     begin
        b[0]=a[226];
        b[1]=a[227];
        b[2]=a[228];
        b[3]=a[229];
        b[4]=a[230];
        b[5]=a[231];
        b[6]=a[232];
        for(k=7;k<74;k=k+1)
            b[k]=a[k-7];
        b[74]=a[226]^a[67];
        b[75]=a[227]^a[68];
        b[76]=a[228]^a[69];
        b[77]=a[229]^a[70];
        b[78]=a[230]^a[71];
        b[79]=a[231]^a[72];
        b[80]=a[232]^a[73];
        for(k=81;k<233;k=k+1)
            b[k]=a[k-7];
    end
endmodule

//Module and_network is a component in module BPWSMSBFFM.
//It is the AND network used to multiply an element by a
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
//coefficient.
module and_network(a,b,c);
    input a;
    input [232:0] b;
    output [232:0] c;
    reg [232:0] c;
    integer k;

    always @(a or b)
     begin
         for(k=0;k<233;k=k+1)
             c[k]=a&b[k];
     end
endmodule
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

References

[1] M. Benantar "Introduction to the Public Key Infrastructure for the Internet" Prentice Hall PRT 2002

[2] T. Beth, D. Gollmann,"Algorithm Engineering for Public Key Algorithm", IEEE Journal on Selected Areas in Communications, VOL. 7, NO. 4, May 1989

[3] G. Birkhoff, S. Mac Lane, "A Survey of Modern Algebra", 5th ed. New York: Macmillan, p. 413, 1996

[4] H. Brunner, A. Curiger, M.Hofstetter, "On Computing Multiplicative Inverses in $GF(2^m)$" IEEE Trans. Computers, VOL 42, NO. 8, August 1993

[5] H. Eberle, S. Chang, N. Gura, S. Gupta, D. Finchelstein, E. Goupy, D. Stebila, " An End-to-End Systems Approach to Elliptic Curve Cryptography" Sun Microsytems Laboratories 2002-2003

[6] D. Gollmann, "qually Spaced Polynomials, Dual Bases, and Multiplication in F_{2^m}" IEEE Trans. Computers, VOL.51, NO.5, May 2002

[7] C. Grabbe, M. Bednara, J. Teich, J. Von Zur Gathen, J. Shokrollahi, "FPGA designs of parallel high performance $GF(2^233)$ multipliers", Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on , VOL 2 , 25-28 May 2003

[8] J. Grobschadl, "A LOW-POWER BIT-SERIAL MULTIPLIER FOR FINITE FIELDS $GF(2^m)$" IEEE International Symposium on Circuits and Systems ISCAS 2001, Sydney, Australia, May 6-9, 2001

[9] U. Hansmann, M. S. Nicklous, T. Schack, F. Seliger, "Smart Card Application Development Using Java" Springer, first edition, 2000.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[10] I.S. Hsu, T.K. Truong, L.J. Deutsch, I.S. Reed, "A comparison of VLSI architecture of finite field multipliers using dual, normal, or standard bases" IEEE Trans. Computers, VOL. 37, Issue 6, June 1988

[11] N. Koblitz, "Elliptic curve cryptosystems", Mathematics of Computation, American Mathematical Society, 48(177):203-209, 1987.

[12] C.K. Koc, B. Sunar "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields" IEEE Trans. Computers, VOL. 47, NO.3, March 1998

[13] C.Y. Lee, "Low complexity bit-parallel systolic multiplier over $GF(2^m)$ using irreducible trinomial", IEE Proc. Comput. Digit. Tech., Vol 150, No. 1, January 2003

[14] C.Y. lee, E.H. Lu, J.Y. Lee, "New Bit-Parallel systolic multipliers for a class of $GF(2^m)$", Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on , 11-13 June 2001 Pages:51 - 58

[15] E.D. Mastrovito, "VLSI Architectures for Multiplication Over Finite Field $GF(2^m)$" Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes, Proc. Sixth Intl Conf., AAECC-6, T. Mora, ed., pp. 297-309, Rome, July 1988. New York: Springer-Verlag.

[16] V.S. Miller, "Use of elliptic curves in cryptography" CRYPTO'85 Proceedings of Crypto, pages 417-426, Springer, 1985.

[17] T. Nagell, "Irreducibility of the Cyclotomic Polynomial." 47 in Introduction to Number Theory. New York: Wiley, pp. 160-164, 1951.

[18] S. Okada, N. Torii, K. Itoh, M. Takenaka, "Implementation of Elliptic Curve Cryptographic Coprocessor over $GF(2^m)$ on an FPGA", C.K. Koc and C. Paar (Eds.): CHES 2000, LNCS 1965, pp. 25-40, 2000. Springer-Verlag Berlin Heidelberg 2000

[19] A. Reyhani-Masoleh, M. Anwar Hasan,"A New Construction of Massey-Omura Parallel Multiplier over $GF(2^m)$"IEEE Trans. Computers, VOL. 51, NO. 5, May 2002

[20] A. Menezes, "Elliptic Curve Cryptosystems", CryptoBytes, Vol.1 No.2, Summer 1995.

[21] C. Paar, "A new architecture for a parallel finite field multiplier with low complexity based on composite fields", IEEE Trans. Computers, VOL. 45, NO. 7, July 1996

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[22] M.J.B. Robshaw, Y.L. Yin, "Elliptic Curve Cryptosystems" An RSA Laboratories Technical Note, Revised Jun2 27, 1997

[23] B. Schneier, "Applied Cryptography" John Wiley and Sons, Inc., 1994.

[24] L. Song, K. K. Parhi, "Efficient Finite Field Serial/Parallel Multiplication" Application Specific Systems, Architectures and Processors, 1996. ASAP 96. Proceedings of International Conference on , 19-21 Aug. 1996 Pages:72 - 82.

[25] B. Sunar, C.K. Koc"Mastrovito Multiplier for All Trinomails"IEEE Trans. Computers, VOL. 48, NO. 5, May 1999

[26] N. Takagi, J. Yoshiki, K. Takagi, " A Fast Algorithm for Multiplicative Inversion in $GF(2^m)$ Using Normal Basis", IEEE Trans. Computers. VOL. 50, NO. 5, May, 2001

[27] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and L.S. Reed,"VLSI Architectures for computing multiplications and Inverses in $GF(2^m)$" IEEE Trans. Computers, VOL. 46, NO. 2, Feb. 1997

[28] H. Wu, "Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial Basi" IEEE Trans. Computers, VOL. 51, NO. 7, July 2002

[29] National Institute of Standard and Technology, FIPS PUB 186-2, Jan 2000

[30] Synopsys online manual "Test automation" SOLD

[31] http://en.wikipedia.org/wiki/Group_(mathematics)#Basic_definitions

[32] http://en.wikipedia.org/wiki/Ring_(mathematics)

[33] http://en.wikipedia.org/wiki/Field_(mathematics)

[34] http://www.certicom.com/index.php?action=res,ecc_faq

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

VITA AUCTORIS

Wenkai Tang was born in 1969 in P.R. China. He received his Bachelor's Degree in Optoelectronics from Electronic Engineering Department in Tsinghua University in 1992. He is currently a candidate for the Master of Applied Science Degree in the Department of Electrical and Computer Engineering at the University of Windsor and hopes to graduate in Winter 2004.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.