國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

通用型橢圓曲線密碼系統純量乘法之實現

An Implementation of Universal Dual-Field Scalar Multiplication on Elliptic Curve Cryptosystems

研究生：劉耀仁

指導教授：張錫嘉 教授

中 華 民 國 九 十 六 年 一 月

通用型橢圓曲線密碼系統純量乘法之實現

An Implementation of Universal Dual-Field Scalar Multiplication on Elliptic Curve Cryptosystems

研 究 生：劉耀仁　　　　Student：Yao-Jen Liu

指導教授：張錫嘉　　　　Advisor：Hsie-Chia Chang

國 立 交 通 大 學
電 子 工 程 學 系　電 子 研 究 所 碩 士 班
碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering

January 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年一月

通用型橢圓曲線密碼系統純量乘法之實現

學生：劉耀仁　　　　　　　　　　　　　　　　　　　指導教授：張錫嘉

國立交通大學電子工程學系 電子研究所碩士班

摘　　　要

這篇論文中介紹了一個同時適用在有限質數場和 GF(2^m)有限場的橢圓曲線純量乘法器的通用型硬體架構。這個所提出的純量乘法器能支援最多 256 位元任意長度的有限質數場，且它也能應付 GF(2^m)有限場不同的場多項式 degree 和 p(x)。實現此可變的通用型硬體架構是根據蒙哥馬利的技術，包括蒙哥馬利的乘法器以及除法器。而所提出的蒙哥馬利模數除法理論也可以用來取代在蒙哥馬利域中的一個模數反元素運算和一個模數乘法運算。這個理論在計算模數除法時，比原本橢圓曲線所使用的方法效能較好，且設計成硬體時也比其他模數除法理論需要較小的面積。而用所提出的純量乘法器架構來計算橢圓曲線上的純量乘法也有合理的速度，例如它只需 3.3 毫秒就可以完成一個 192 位元的純量乘法運算。

An Implementation of Universal Dual-Field Scalar Multiplication on Elliptic Curve Cryptosystems

student：Yao-Jen Liu　　　　　　Advisors：Hsie-Chia Chang

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

ABSTRACT

An universal hardware architecture of scalar multiplier on elliptic curves suitable for both GF(p) and GF(2^m) is introduced in this thesis. The proposed scalar multiplier can work in arbitrary field lengths within a maximum 256-bit length in GF(p), and it also supports various field degrees and primitives in GF(2^m). The flexible universal hardware architecture is based on the Montgomery techniques, including the Montgomery multiplier and divider. The Montgomery modular division algorithm is also proposed to replace the inversion followed by a multiplication in the Montgomery domain. It provides a better performance on modular division operations than previous ECC techniques and also has a smaller area size than other modular division architectures. The proposed scalar multiplier architecture can perform the scalar multiplication on elliptic curves at a reasonable speed. For a 192-bit scalar multiplication operation, it takes about 3.3 ms.

誌　　　謝

很珍惜在交大電子所的這兩年研究所生活，在這短短的兩年中除了學到了許多做研究的方法，更讓人難忘的是在這兩年中給予我指導的師長及帶給我歡樂的朋友。首先，要感謝的是我最敬愛的指導教授張錫嘉博士，大學時曾經修過老師的邏輯設計，便結下了這段不解之緣。老師給人的感覺就像是一位親切的學長般，帶著幽默且平易近人的方式與我們相處，不僅是良師也是益友，在研究所這兩年生涯中不斷地給我許多專業的指導和研究方向，生活上的大小事都會給予幫助；在這也要感謝老師與師母在我生病住院時還親自到醫院來探望，讓我實在倍感溫馨。還有建青學長、彥欽學姊每個禮拜都撥冗幫忙主持實驗室會議，而且都能及時給我們很多在學術上的建議與幫助。特別要感謝的是陳榮傑教授，橢圓曲線的課程給了我很多啟發，以及半年的學術研討交流，使我在論文上得到很多的幫助。再來要感謝的是曾經相處一年的已畢業學長們，年薪百萬、寫了三篇論文就一直拿來說嘴的小科學長，不辭辛勞擔任導遊、招待實驗室宜蘭花蓮五日遊然後猛收回扣的小安學長，任勞任怨、替實驗室管理主機跟郵件伺服器的毛律學長，大學一起玩社團、教導我橢圓曲線的維均學長，時常在實驗室放閃光的小天學長，口味與眾不同、蟒蛇也能當寵物的周毓堂學長；此外，還有陪伴我一起度過兩年研究所生涯的同袍，說話總是格外中肯、現在是老師博士班愛將的大頭，當了三年室友、下雨天時常讓我搭便車的胖威，熱心招待實驗室台中美食之旅的祐徵，從泰國回來就變了個樣、成天打嘴砲的大師兄；再來就是還有一年要努力的學弟妹們，能說善道、騙了很多專題小學妹進來的博元學弟，熬夜趕期中報告時會一邊熱心播報王建民即時戰況的佳瑋學妹，也是系籃明星球員而且選過模特兒的修齊學弟，會溫馨送宵夜、半夜一起在實驗室奮鬥的大嘴學弟，感謝所有一起陪伴過我的朋友們。最後要感謝的是我的家人，一直寵愛著我的父母、找不到詞可以形容的哥哥，還有很愛胡思亂想的妹妹，感謝你們一直在背後給我莫大的支持與鼓勵。

An Implementation of Universal Dual-Field Scalar Multiplication on Elliptic Curve Cryptosystems

Student: Yao-Jen Liu

Advisor: Dr. Hsie-Chia Chang

Department of Electronics Engineering
National Chiao Tung University

Abstract

An universal hardware architecture of scalar multiplier on elliptic curves suitable for both $GF(p)$ and $GF(2^m)$ is introduced in this thesis. The proposed scalar multiplier can work in arbitrary field lengths within a maximum 256-bit length in $GF(p)$, and it also supports various field degrees and primitives in $GF(2^m)$. The flexible universal hardware architecture is based on the Montgomery techniques, including the Montgomery multiplier and divider. The Montgomery modular division algorithm is also proposed to replace the inversion followed by a multiplication in the Montgomery domain. It provides a better performance on modular division operations than previous ECC techniques and also has a smaller area size than other modular division architectures. The proposed scalar multiplier architecture can perform the scalar multiplication on elliptic curves at a reasonable speed. For a 192-bit scalar multiplication operation, it takes about 3.3 ms.

Contents

List of Figures

List of Tables

Chapter 1

introduction

1.1 Background

Since the public-key cryptography was introduced by Diffe and Hellman [1] in 1976, the use of discrete logarithm problem in public-key cryptosystems has been recognized. This method of exponential key exchange came to be known as Diffie-Hellman key exchange. RSA and El-Gamal are two of the popular public-key cyrptosystems widely used nowadays. The RSA algorithm based on the difficult of factoring large numbers was published by Rivest, Shamir and Adleman [2] at MIT[1] in 1978. Further, the El-Gamal algorithm based on Diffie-Hellman key agreement describes the public-key system and digital signature schemes, and it was proposed by Taher ElGamal [3] in 1985.

The public-key cryptosystem such as RSA is still widely used in electronic commerce protocols and it is believed to be secure enough as long as it has sufficiently long keys. However, there are many efficient attacks known for both RSA and modular p discrete log based cryptosystems such as the Number Field Sieve [4] attacks for RSA and the index calculus attacks for the modular p systems.

The elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The ECC was independently proposed by Victor S. Miller of IBM[2] in 1986 [5] and Neal Koblitz of the University of Washington in 1987 [6]. There are no subexponential algorithms known

[1]Massachusetts Institute of Technology, located in Cambridge, MA, USA. http://web.mit.edu/

[2]International Business Machines Corporation. http://www.ibm.com/

for the elliptic curve discrete logarithm problem (ECDLP) and denotes that there are no efficient attacks known on it. Consequently, the parameters for ECC can be chosen to be much smaller than the paramters for RSA with the same level of resistance against the best known attacks. Table 1.1 shows each different parameter size with the same level of security strengths compared with given cryptography [7].

Table 1.1: Comparable security strength for given cryptography

ECC (e.g., ECDSA)	IFC (e.g., RSA)	Symmetric key algorithms
$f = 160 - 223$	$k = 1024$	-
$f = 224 - 255$	$k = 2048$	-
$f = 256 - 383$	$k = 3072$	AES-128
$f = 384 - 511$	$k = 7680$	AES-192
$f = 512 \uparrow$	$k = 15360$	AES-256

[1] ECDSA (Appendix A.3).

[2] IFC denotes integer factorization cryptography.

[3] f is the size of n, where n is the order of the base point G.

[4] k is the size of the modulus p.

[5] Advanced Encryption Standard (AES) [8].

Note that in Table 1.1, the difference of the size between ECC and RSA becomes more enormous as the security level increases. It is attractive that the ECC has much smaller parameters leads to more significant performance advantages contrast to RSA. Therefore, the ECC takes advantages for wireless applications where the computing power, memory and battery life are limited such as smart cards and wireless devices.

Furthermore, the performance of ECC mainly depends on the efficiency of its mathematical arithmetics, namely, scalar multiplication. Given p, a positive integer, and a point P on an elliptic curve. The scalar multiplication kP can easily be defined as adding the $(k-1)$ copies of P to itself. There are some algorithm to compute the multiple of points on elliptic curves. More details will be discussed in chapter 2.3 later.

At the end of this thesis in appendix, some schemes for ECC are listed. Appendix A.1 is El-Gamal on elliptic curves, Appendix A.2 is Elliptic Curve Diffie-Hellman (ECDH) [9] and Appendix A.3 is Elliptic Curve Digital Signature Algorithm (ECDSA) [10].

1.2 Motivation

In recent years, security issues on communications are more and more significant as the wireless industry explodes. The ECC has become an important role in public-key cryptographic systems. There are more and more applications using ECC as authentication for transactions and encryption or signature for secure messaging. For example, ECDSA has been used to sign the product key by Microsoft[3] since Windows 95[4].

Since the scalar multiplications on elliptic curves are needed, there are much advanced research on modular arithmetic operations over finite fields such as the Montgomery's technique [11] for modular multiplication which will be discussed in chapter 3.1.1. Then the Montgomery's technique for modular inversion has been described in [12] and will be mentioned later in chapter 3.2.2. However, scalar multiplication on elliptic curves needs modular division operations in affine coordinates, and there are few implementations on ECC applications using dedicated modular division component. There were still no efficient modular division algorithms known for past hundreds of years. Instead, modular division is achieved by computing the inversion followed by multiplication, but it takes longer latency between domain transformation using Montgomery's technique in this way. Thus, a Montgomery modular division algorithm is developed in this thesis to shorten the timing for scalar multiplication on elliptic curves using Montgomery's technique.

In Freescale[5] MPC190 security processor, it includes elliptic curve operations in either $GF(p)$ or $GF(2^m)$ in its features, and its programmable field size is from 55 to 511 bits. The point operations (addition, doubling and multiplication) involve one or more finite field operations, which are addition, multiplication, inverse and squaring.

Therefore, in this thesis, an approach is also provided to compute the scalar multiplication on elliptic curves in both $GF(p)$ and $GF(2^m)$, and the Montgomery technique can be used to deal with various finite field degrees and different primitive polynomials in $GF(2^m)$. Further, in this feature, one more finite field operation, division, is involved here to replace the inversion followed by a multiplication, and it is used in hardware to accelerate the scalar multiplication operations in ECC applications.

[3]Microsoft Corporation. http://www.microsoft.com/

[4]A consumer-oriented graphical user interface-based operating system developed by Microsoft in 1995.

[5]Freescale Semiconductor, Inc. http://www.freescale.com/

1.3 Thesis Organization

In this thesis, the total solution to elliptic curve operations in hardware and software is given. In Chapter 2, the preliminary mathematical background of elliptic curves is introduced. In Chapter 3, the Montgomery's techniques for the finite field arithmetic are detailed in this chapter. It shows each algorithm in both prime field and binary extension field versions. Additionally, an algorithm for Montgomery modular division is proposed for the combination of Montgomery inversion and Montgomery multiplication. In Chapter 4, all the proposed universal dual-field architectures are described in this chapter. An universal architecture of Montgomery multiplication for both prime field and binary extension field is proposed first. Then the implementation of the proposed Montgomery modular division algorithm is presented. The division hardware is the major part of the scalar multiplier on elliptic curves. In Chapter 5, it shows the hardware implementation results and test consideration for the scalar multiplier on elliptic curves. Finally, the conclusion is given in Chapter 6.

Chapter 2

Elliptic Curves

Elliptic curves [13] [14] are not ellipses as shown in literal. In mathematics, an elliptic curve is an algebraic curve defined by a cubic equation such as $y^2 = x^3 + ax + b$, which is non-singular, i.e. its graph has no cusps or self-intersections. Elliptic curves received their name from their relation to elliptic integrals such as

$$\int_{z_1}^{z_2} \frac{dx}{\sqrt{x^3 + ax + b}} \text{ and } \int_{z_1}^{z_2} \frac{x\,dx}{\sqrt{x^3 + ax + b}} \tag{2.1}$$

that arose in connection with the computation of the circumference of ellipses.

2.1 Basic Facts

Let $\mathbb{F}$ be an algebraically closed field and $\mathbb{F}^2$ denote the affine plane $\mathbb{A}^2$, the usual plane, $\mathbb{A}^2(\mathbb{F}) = \{(x, y)|x, y \in \mathbb{F}\}$. Let $C(x, y)$ be an irreducible polynomial over $\mathbb{F}$, and the curve C means the set of zeros of C in the affine plane $\mathbb{F}^2$, i.e. $\{(x, y) \in \mathbb{F}^2|C(x, y) = 0\}$. Assume that P is a point (x_p, y_p) on the curve C. If both of the partial derivatives vanish at P, that is $\frac{\partial C(x_p,y_p)}{\partial x} = \frac{\partial C(x_p,y_p)}{\partial y} = 0$, then the point P is called a singular point on the curve C. A curve is called a singular curve if and only if it has at least one singular point on it, otherwise it is called a non-singular curve. An elliptic curve commonly used in cryptography is a non-singular curve because of its better security level relative to a singular curve. A singular elliptic curve is thought of insecure in general. Definition 2.1 shows the algebraic equation of the elliptic curve in a more general form.

Definition 2.1. *An elliptic curve E over the field $\mathbb{F}$ defined by an affine Weierstrass equation is an equation of the form*

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, \quad \forall a_i \in \mathbb{F} \tag{2.2}$$

let $E(\mathbb{F})$ denote the elliptic curve E over $\mathbb{F}$, i.e. the set of points $(x, y) \in \mathbb{F}^2$ that satisfy this equation, along with the point at infinity denoted by $\mathcal{O}$.

Definition 2.2. *The point at infinity called $\mathcal{O}$ is the intersection of the y-axis and the line at infinity. The line at infinity is the set of points on the projective plane for which $Z = 0$. Therefore, the point at infinity $\mathcal{O}$ is $(0, 1, 0)$ in the projective plane, i.e. the equivalence class with $X = Z = 0$.*

No further details about projective plane are shown in this thesis since only affine coordinates are discussed in the remaining chapters.

In order to describe a singular or non-singular curve clearly, an important quantity Δ related to the elliptic curve called the discriminant of E is defined.

Definition 2.3. *Δ is the discriminant of E and is given by*

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6, \quad \textit{where} \begin{cases} b_2 &= a_1^2 + 4a_2 \\ b_4 &= 2a_4 + a_1a_3 \\ b_6 &= a_3^2 + 4a_6 \\ b_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 \\ & \quad +a_2a_3^2 - a_4^2 \end{cases} \tag{2.3}$$

and the symbols above correspond to (2.2).

Theorem 2.1. *A cubic curve defined by a Weierstrass equation (2.2) is singular if and only if its discriminant Δ is zero.*

The Definition 2.1 is feasible for any field $\mathbb{F}$. However, the elliptic curves commonly used in cryptography are over the finite field $GF(q)$, where q is either a large prime p or a power of p. If q is a large prime p, the prime field $GF(p)$, also labeled as $\mathbb{F}_p$ or $\mathbb{Z}_p$, is a field of characteristic p where $p \neq 2, 3$, that is, $\sum_{i=1}^{k} 1 = k \neq 0$ for $1 \leq k < p$ and $\sum_{i=1}^{p} 1 = 0$. If q is a power of p, denoted by p^m, the Galois field $GF(p^m)$ is an extension field of $GF(p)$, where p is typically chosen as 2 for the sake of binary property in hardware. The finite fields are also called Galois fields, in honor of their discoverer.

On the basis of various characteristics, the Weierstrass equation (2.2) can be simplified into different forms by a linear change of variables. The following paragraphs shows the equation for a field of characteristic $\neq 2, 3$ and a field of characteristic 2.

Let $\mathbb{F}$ be a field of characteristic $\neq 2, 3$ and $char(\mathbb{F})$ denote the characteristic of $\mathbb{F}$. Since the $char(\mathbb{F}) \neq 2$, substitute (X, Y) by $(X, Y - \frac{a_1X+a_3}{2})$ on the left hand side in (2.2).

$$\begin{array}{rl} & Y^2 + a_1XY + a_3Y \quad \text{substitute} \quad (X,Y) \to (X, Y - \dfrac{a_1X+a_3}{2}) \\ \Rightarrow & (Y - \frac{a_1X+a_3}{2})^2 + a_1X(Y - \frac{a_1X+a_3}{2}) + a_3(Y - \frac{a_1X+a_3}{2}) \\ = & Y^2 - \frac{a_1^2}{4}X^2 - \frac{a_1a_3}{2}X - \frac{a_3^2}{4} \end{array} \tag{2.4}$$

Notice that both XY and Y term are eliminated so the coefficients a_1 and a_3 should be zero. Thus the equation (2.4) results in Y^2 by substitution for $a_1 = a_3 = 0$. Further, the $char(\mathbb{F}) \neq 3$ so substitute (X, Y) by $(X - \frac{a_2}{3}, Y)$ on the right hand side in equation (2.2).

$$\begin{array}{rl} & X^3 + a_2X^2 + a_4X + a_6 \quad \text{substitute} \quad (X,Y) \to (X - \dfrac{a_2}{3}, Y) \\ \Rightarrow & (X - \frac{a_2}{3})^3 + a_2(X - \frac{a_2}{3})^2 + a_4(X - \frac{a_2}{3}) + a_6 \\ = & X^3 + (\frac{-1}{3}a_2^2 + a_4)X + (\frac{2}{27}a_2^3 - \frac{1}{3}a_2a_4 + a_6) \end{array} \tag{2.5}$$

Then again, the X^2 term is eliminated so that the coefficient a_2 should be zero and the equation (2.5) results in $X^3 + a_4X + a_6$ by setting $a_2 = 0$. According to (2.4) and (2.5), let $a_1 = a_2 = a_3 = 0, a_4 = a, a_6 = b$ and the equation (2.2) is modified as follows

$$Y^2 = X^3 + aX + b, \quad a, b \in \mathbb{F} \tag{2.6}$$

where $char(\mathbb{F}) \neq 2, 3$. Note that the elliptic curve is a smooth curve, i.e. the curve is non-singular. Review in Theorem (2.1), an elliptic curve should have its discriminant nonzero. Therefore, the discriminant of the cubic curve (2.6) can be derived through (2.3) by substitution for $a_1 = a_2 = a_3 = 0, a_4 = a, a_6 = b$. Thus $\Delta = -16(4a^3 + 27b^2) \neq 0$.

For a field of characteristic 2, only the non-supersingular case is considered. In brief, non-supersingular has the result of the coefficient $a_1 \neq 0$. Since $a_1 \neq 0$, substitute (X, Y) by $(a_1^2X + \frac{a_3}{a_1}, a_1^3Y + \frac{a_1^2a_4+a_3^2}{a_1^3})$ in (2.2) likewise. A simplified form is obtained as follows

$$Y^2 + XY = X^3 + aX^2 + b, \quad a, b \in \mathbb{F} \tag{2.7}$$

where $char(\mathbb{F}) = 2$. There is no need to care whether or not the cubic polynomial on the right hand side in (2.7) has multiple roots.

2.2 Elliptic Curves Arithmetic

Elliptic curve cryptography makes use of elliptic curves where the variables and coefficients are belong to a finite field. Two kinds of elliptic curves are commonly used in cryptographic applications. They are prime curves over $GF(p)$ and binary curves over $GF(2^m)$ respectively. Before discussion on the above curves, the elliptic curves over the reals are first introduced because some of the basic concepts are easier to visualize.

2.2.1 Elliptic Curves over the Reals

According to equation (2.6), a definition for elliptic curves over the reals is given below.

Definition 2.4. *A non-singular elliptic curve E over the reals is an equation of the form*

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

where $a, b \in \mathbb{R}$ are constants such that $4a^3 + 27b^2 \neq 0$.

It can be shown that the condition $4a^3 + 27b^2 \neq 0$ is necessary and sufficient to ensure that the equation (2.8) has three distinct roots which may be real or complex numbers. Figure 2.1 shows two non-singular elliptic curves and one singular elliptic curve whose equation are $y^2 = x^3 - 4x$, $y^2 = x^3 + 73$, and $y^2 = x^3 - 3x - 2$ respectively.

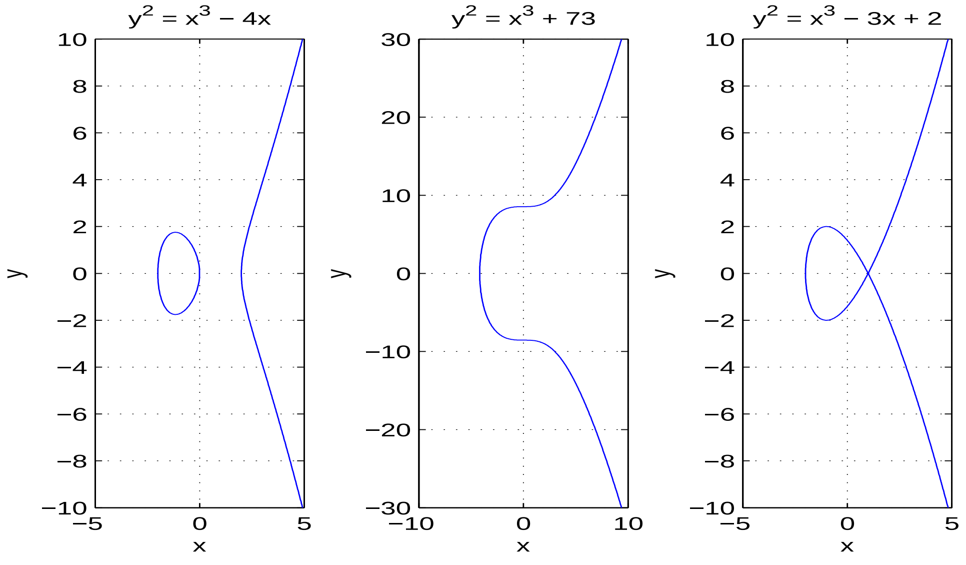

Figure 2.1: Elliptic curves over the reals

Let E be a non-singular elliptic curve over the reals. Given two points P and Q on E, the negative of P, denoted by $-P$, and the sum $P+Q$ is defined as follows:

1. If P is the point at infinity $\mathcal{O}$, then $-P$ is $\mathcal{O}$ and $P+Q$ is Q; that is, $\mathcal{O}$ is the additive identity which is also called zero element of the group of points.

2. If P is not the point at infinity $\mathcal{O}$, then $-P$ is the symmetry point of P on the curve E; that is, $-P$ is the point with the same x-coordinate and negative the y-coordinate of P, i.e. $-(x,y)=(x,-y)$. According to equation (2.8), if (x,y) is a point on the curve E, then the point $(x,-y)$ is consequently on the curve E.

3. If P and Q are different points on E with different x-coordinates, then let l be the line through P and Q, and the line l intersects the curve E in exactly one more point R. Then the sum $P+Q=-R$ is defined and is illustrated in Figure 2.2.

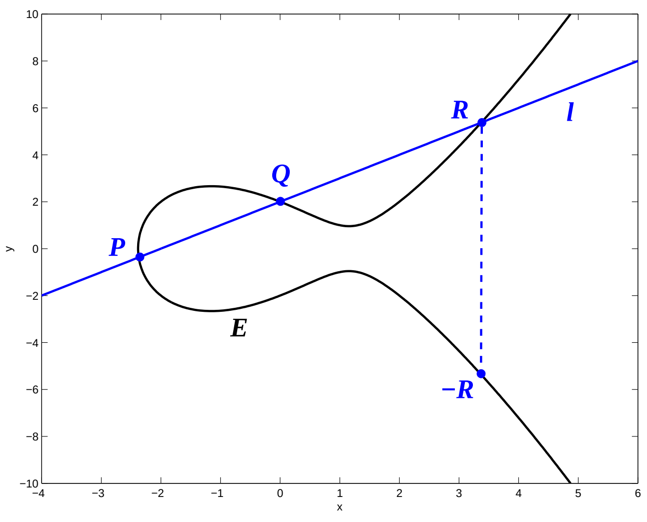

Figure 2.2: Adding two distinct points $P+Q=-R$

4. If P and Q are different points on E with the same x-coordinates, that is, Q is a symmetry point of P equal to $-P$, then the sum $P+Q=P+(-P)=\mathcal{O}$ is defined.

5. If P and Q are the same points on E, then let the line l be the tangent line to the curve at P and the point R be the only other point of intersection of l with the curve E. Thus the sum $P+Q=P+P=2P=-R$ is defined and is illustrated in

Figure 2.3. Furthermore, if the tangent line has a double tangency at P, that is, P is a point of inflection, then the sum $P+Q=P+P=2P=-P$ is defined.

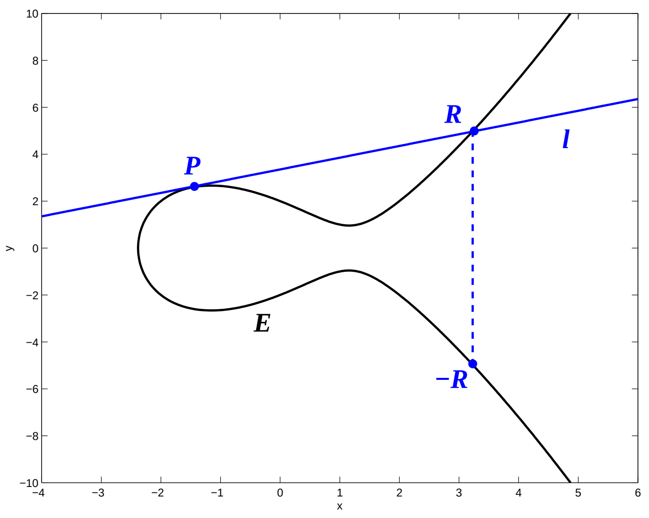

Figure 2.3: Doubling a point $2P=-R$

In figure 2.2, let (x_1,y_1), (x_2,y_2), $(x_3,-y_3)$ and (x_3,y_3) denote the coordinates of P, Q, R and $P+Q$ respectively. Let $l: y=\lambda x+\beta$ be the equation of the line through P and Q then $\lambda=\frac{y_2-y_1}{x_2-x_1}$ is the slope of the line l and $\beta=y_1-\lambda x_1=y_2-\lambda x_2$ is the consequence of the point P lying on the line l. Assume that t is a variable and $(t,\lambda t+\beta)$ denotes the coordinates of arbitrary points on the line l. The point on l simultaneously lies on the elliptic curve E if and only if $(t,\lambda t+\beta)$ satisfies equation (2.8) so that $(\lambda t+\beta)^2=t^3+at+b$ and rearrange it below by order of t.

$$t^3+(-\lambda^2)t^2+(a-2\lambda\beta)t+(b-\beta^2)=0 \tag{2.9}$$

Note that the equation has exactly three distinct roots and two of them are known as x_1 and x_2. Remember the relation between roots and coefficient mentioned in Viéte formula first proposed by François Viéte (1540–1603), a French mathematician.

Theorem 2.2. *(**Viéte's Formula**) Assume $P(x)$ is a polynomial of degree n with roots $x_1,x_2,\ldots,x_n$. For $1\le i\le n$, let S_i be the sum of the products of distinct polynomial roots x_j of the polynomial*

$$P(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0=0 \tag{2.10}$$

where the roots are taken i at a time, i.e. S_i is defined as the symmetric polynomial $\Pi_i(x_1, \ldots, x_n)$ for $i = 1, \ldots, n$, where

$$S_i = \Pi_i(x_1, \ldots, x_n) = \sum_{1 \le \alpha_1 < \alpha_2 < \ldots < \alpha_k \le n} x_{\alpha_1} x_{\alpha_2} \cdots x_{\alpha_k} \tag{2.11}$$

For example, the first few values of S_i are

$$\begin{aligned}
S_1 &= \Pi_1(x_1, \ldots, x_n) &=& \sum_{1 \le i \le n} x_i &=& x_1 + x_2 + x_3 + x_4 + \cdots \\
S_2 &= \Pi_2(x_1, \ldots, x_n) &=& \sum_{1 \le i < j \le n} x_i x_j &=& x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + \cdots \\
S_3 &= \Pi_3(x_1, \ldots, x_n) &=& \sum_{1 \le i < j < k \le n} x_i x_j x_k &=& x_1x_2x_3 + x_1x_2x_4 + x_2x_3x_4 + \cdots
\end{aligned}$$

and so on. Then Viéte's formula states that

$$S_i = (-1)^i \frac{a_{n-i}}{a_n} \tag{2.12}$$

Proof. The polynomial $P(x)$ can also be written as

$$\begin{aligned}
P(x) &= a_n(x - x_1)(x - x_2) \cdots (x - x_n) \\
&= a_n(x^n - S_1 x^{n-1} + S_2 x^{n-2} - \cdots + (-1)^n S_n)
\end{aligned} \tag{2.13}$$

According to equation (2.10), setting the coefficients equal yields

$$a_n(-1)^i S_i = a_{n-i}$$

which is what the Viéte's formula states for. **Q.E.D.**

The Viéte formula was proved by Viéte (1579) for positive roots only, and the general theorem was proved by Gérard Desargues (1591–1661). Therefore the sum of the roots s_1 of a monic polynomial shown in (2.9) is equal to minus the coefficient of the second-to-highest order. A monic polynomial or normed polynomial is a polynomial whose leading coefficient is equal to 1. It concludes that the third root x_3 in (2.9) is equal to $\lambda^2 - x_1 - x_2$ since the sum of the three distinct roots s_1 is λ^2. Then the y-coordinate of R is $\lambda x_3 + \beta$ and y_3 is minus the y-coordinate of R. Therefore the coordinate of $P + Q$ in terms of x_1, x_2, y_1, y_2 is shown below.

$$\begin{aligned}
x_3 &= (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \\
y_3 &= (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1
\end{aligned} \tag{2.14}$$

In figure 2.3, let (x_1, y_1), (x_2, y_2), $(x_3, -y_3)$ and (x_3, y_3) denote the coordinates of P, Q, R and $P + Q$ respectively. Since P and Q are the same point, $x_2 = x_1$ and $y_2 = y_1$. Let $l : y = \lambda x + \beta$ be the equation of the tangent line to the curve E at P. The slope of the tangent line at P can be derived by differentiation of the equation (2.8) as follows.

$$\begin{aligned} \frac{d}{dx}(y^2) &= \frac{d}{dx}(x^3 + ax + b) \\ (\frac{dy}{dx}) &= \frac{3x^2 + a}{2y} \end{aligned} \tag{2.15}$$

So the slope of the tangent line $\lambda = \frac{3x_1^2+a}{2y_1}$. According to (2.14), substitute $(\frac{y_2-y_1}{x_2-x_1})$ for $(\frac{3x_1^2+a}{2y_1})$ and $x_2 = x_1$, $y_2 = y_1$. A formula for doubling a point is obtained.

$$\begin{aligned} x_3 &= (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \\ y_3 &= (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3) - y_1 \end{aligned} \tag{2.16}$$

Table 2.1 shows the addition formula mentioned above.

	Point Addition ($P \neq Q$)	Point Doubling ($P = Q$)
$P + Q$	$x_3 = \lambda^2 - x_1 - x_2$ $y_3 = \lambda(x_1 - x_3) - y_1$ $\lambda = \frac{y_2-y_1}{x_2-x_1}$	$x_3 = \lambda^2 - 2x_1$ $y_3 = \lambda(x_1 - x_3) - y_1$ $\lambda = \frac{3x_1^2+a}{2y_1}$

Table 2.1: Point addition formula over reals

2.2.2 Elliptic Curves over Prime Fields

Let $p > 3$ be a prime. Elliptic curves over $GF(p)$ are defined almost the same as they are over the reals and the operations over the reals are replaced by modulus operations.

Definition 2.5. *Let $p > 3$ be a prime. A non-singular elliptic curve E over the finite field $GF(p)$ is an equation of the form*

$$y^2 \equiv x^3 + ax + b \pmod p \tag{2.17}$$

where $a, b \in GF(p)$ are constants such that $4a^3 + 27b^2 \not\equiv 0 \pmod p$.

Assume that (x_1, y_1), (x_2, y_2) and (x_3, y_3) denote the coordinates of P, Q and $P + Q$ respectively. Then the coordinate of $-P$ is defined as $(x_1, -y_1)$ and $P + (-P) = \mathcal{O}$.

According to equation (2.14) and (2.16), the point addition formula of the elliptic curves over $GF(p)$ is shown in Table 2.2.

	Point Addition ($P \neq Q$)	Point Doubling ($P = Q$)
$P + Q$	$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod{p}$ $y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$ $\lambda \equiv \frac{y_2 - y_1}{x_2 - x_1} \pmod{p}$	$x_3 \equiv \lambda^2 - 2x_1 \pmod{p}$ $y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$ $\lambda \equiv \frac{3x_1^2 + a}{2y_1} \pmod{p}$

Table 2.2: Point addition formula over $GF(p)$

2.2.3 Elliptic Curves over Extension of Binary Fields

Definition 2.6. *Let $p(x)$ be a primitive polynomial of degree m. A non-supersingular elliptic curve E over the extension of binary field $GF(2^m)$ is an equation of the form*

$$y^2 + xy = x^3 + ax^2 + b \tag{2.18}$$

where $a, b \in GF(2^m)$ are constants.

Note that in this subsection, all of the arithmetic operations are defined over $GF(2^m)$ and all of the parameters are belong to $GF(2^m)$, too. Assume that (x_1, y_1), (x_2, y_2) and (x_3, y_3) denote the coordinates of P, Q and $P + Q$ respectively. Then the coordinate of $-P$ is defined as $(x_1, x_1 + y_1)$ and $P + (-P) = \mathcal{O}$.

If $P \neq Q$, let $l : y = \lambda x + \beta$ be the equation of the line through P and Q then $\lambda = \frac{y_2 + y_1}{x_2 + x_1}$ is the slope of the line l and $\beta = y_1 + \lambda x_1 = y_2 + \lambda x_2$ is the consequence. The following equation shows all of the points $(t, \lambda x + \beta)$ on l simultaneously lies on the curve E.

$$t^3 + (\lambda^2 + \lambda + a)t + (\beta^2 + b) = 0 \tag{2.19}$$

Thus the third root $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ and the corresponding y-coordinate is $\lambda x_3 + \beta$. So the negative of the y-coordinate $y_3 = (\lambda x_3 + \beta) + x_3 = \lambda(x_1 + x_3) + x_3 + y_1$.

If $P = Q$, let $l : \lambda x + \beta$ be the equation of the tangent line to the curve E at P. The slope of the tangent line at P can be derived by differentiation of the equation (2.18).

$$\begin{gathered} \frac{d}{dx}(y^2 + xy) = \frac{d}{dx}(x^3 + ax^2 + b) \\ (\frac{dy}{dx}) = x + \frac{y}{x} \end{gathered} \tag{2.20}$$

So the slope of the tangent line $\lambda = x_1 + \frac{y_1}{x_1}$. Since $P = Q$, $x_3 = \lambda^2 + \lambda + a$ and $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$; moreover, there is another formula commonly used for y_3 by changing varibale. Given $\lambda = x_1 + \frac{y_1}{x_1} = \frac{x_1^2+y_1}{x_1}$ that leads to $\lambda x_1 + y_1 = x_1^2$. Thus rearrange $y_3 = (\lambda+1)x_3 + \lambda x_1 + y_1$ and adapt it for $y_3 = (\lambda+1)x_3 + x_1^2$. Table 2.3 lists all obtained formulas of the above together for $GF(2^m)$.

	Point Addition ($P \neq Q$)	Point Doubling ($P = Q$)
$P+Q$	$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ $\lambda = \frac{y_2+y_1}{x_2+x_1}$	$x_3 = \lambda^2 + \lambda + a$ $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ $= (\lambda + 1)x_3 + x_1^2$ $\lambda = x_1 + \frac{y_1}{x_1}$

Table 2.3: Point addition formula over $GF(2^m)$

2.3 Elliptic Curves Scalar Multiplication

Scalar multiplication is used to compute a multiple of an Elliptic curve point kP, where P is an elliptic curve point and k is a positive integer smaller than the order of P, then kP is the point obtained by adding together k copies of P and this operation dominates the execution time of elliptic curve cryptographic schemes.

2.3.1 Double-and-Add Algorithm

Algorithm 2.1. ***(Double-and-Add Algorithm)***

Input: *A positive integer $k<n$, where n is the order of P; and an elliptic curve point P.*

Output: *The elliptic curve point kP.*

```
1. Let $k_n k_{n-1} \ldots k_1 k_0$ be the binary representation of $k$, where the leftmost bit $k_n$ is 1.
2. Set $R = P$.
3. For $i$ from $n-1$ down to 1 do
    3.1 Set $R = 2R$.
    3.2 If $k_i = 1$, then set $R = R + P$.
4. Output $R$.
```

The double-and-add algorithm is a basic method for calculating scalar multiplication. It achieves by repeated point double and add operations. The expected number of ones in the binary representation of k is $\frac{m}{2}$, where m is the length of the integer k. The number of ones in k indicates the number of times that point addition performs and the number of times that point doubling operation performs is approximately equal to m. Thus Algorithm 2.1 averagely takes $\frac{m}{2}$ times point addition and m times point doubling to perform m-bit elliptic curve scalar multiplication once.

2.3.2 Addition-Subtraction Method

If $P(x, y) \in E(\mathbb{F}_p)$ then $-P = (x, -y)$; else if $P(x, y) \in E(\mathbb{F}_{2^m})$ then $-P = (x, x + y)$. Thus the point subtraction is as efficient as point addition. Then Algorithm 2.1 is replaced by using addition-subtraction method and shown in Algorithm 2.2.

Algorithm 2.2. ***(Addition-Subtraction Method)***
Input: *A positive integer $k < n$, where n is the order of P; and an elliptic curve point P.*
Output: *The elliptic curve point kP.*

```
1. Let $e_n e_{n-1} \dots e_1 e_0$ be the binary representation of $3k$, where the leftmost bit $e_n$ is 1.
2. Let $k_n k_{n-1} \dots k_1 k_0$ be the binary representation of $k$.
3. Set $R = P$.
4. For $i$ from $n-1$ down to 1 do
    4.1 Set $R = 2R$.
    4.2 If $e_i = 1$ and $k_i = 0$, then set $R = R + P$.
    4.3 If $e_i = 0$ and $k_i = 1$, then set $R = R - P$.
5. Output $R$.
```

2.3.3 Binary NAF Method

Owing to point subtraction is as efficient as point addition, the signed digit representation $k = \sum k_i 2^i$ is used, where $k_i \in \{0, \pm 1\}$. A non-adjacent form (NAF) is a useful signed representation which has the property that no two consecutive bits in k are nonzero and has the fewest nonzero bits of any signed digit representation of k. Each positive integer k has its unique NAF, denoted by NAF(k). The NAF of an integer k can be computed efficiently by using Algorithm 2.3 [15].

Algorithm 2.3. ***(NAF of a Positive Integer)***

Input: *A positive integer k.*

Output: *NAF(k).*

```
1. Set $i = 0$.
2. While $k \geq 1$ do
    2.1 If $k$ is odd, then set $k_i = 2 - (k \bmod 4)$ and then set $k = k - k_i$; else set $k_i = 0$.
    2.2 Set $k = \frac{k}{2}$ and $i = i + 1$.
3. Output $k$, whose binary representation is $(k_{i-1}k_{i-2} \ldots k_1k_0)$.
```

Note that the length of NAF(k) is at most one bit longer than the binary representation of k and the average density of nonzero bits in NAF(k) is approximately $\frac{m}{3}$ [16], where m is the length of the integer k.

Algorithm 2.4. ***(Binary NAF Method)***

Input: *NAF(k) and an elliptic curve point P.*

Output: *The elliptic curve point kP.*

```
1. Let $k_nk_{n-1} \ldots k_1k_0$ be signed digit representation of $k$, where the leftmost bit $k_n$ is $1$.
2. Set $R = P$.
3. For $i$ from $n - 1$ down to $0$ do
    3.1 Set $R = 2R$.
    3.2 If $k_i = 1$, then set $R = R + P$.
    3.3 If $k_i = -1$, then set $R = R - P$.
4. Output $R$.
```

Then the Algorithm 2.4 modifies Algorithm 2.1 by using NAF(k) instead of the binary representation of k and averagely takes approximately $\frac{m}{3}$ times point addition and m times point doubling to perform m-bit elliptic curve scalar multiplication once. Furthermore, it follows that the expected running time of Algorithm 2.2 and Algorithm 2.4 are the same.

Chapter 3

Finite Field Arithmetic

This chapter describes the basic arithmetic used in elliptic curve over $GF(p)$ and $GF(2^m)$. Modular arithmetic such as modular multiplication is especially an important part in many cryptographic systems, so there are still many approaches to its improvement nowadays. Since affine coordinate is used for elliptic curve cryptography in this thesis, modular inverse and division arithmetics are also discussed in this chapter.

3.1 Modular Multiplication

Modular multiplication is widely used in many applications including public key cryptography such as RSA [2] algorithm. The RSA algorithm requires the computation of modular exponentiation and this modular exponentiation is achieved through a series of modular multiplications. Given an m-bit integer p, called the modulus, and two m-bit operands a and b, modular multiplication computes the result of $a \times b \pmod{p}$.

3.1.1 Montgomery Multiplication Algorithm

The Montgomery multiplication algorithm, which was proposed by P. L. Montgomery in 1985 [11], computes the modular multiplication without trial division. It turns the modular multiplication into iterations of addition and shift operations. Thus the Montgomery multiplication is quite appropriate for implementing modular multiplication with hardware. Let the modulus p be an m-bit integer with $2^{m-1} \leq p < 2^m$, and let r be equal to 2^m where p and r are relatively prime, i.e. $\gcd(p, r) = \gcd(p, 2^m) = 1$. Thus the p-residue of an integer a with $0 \le a < p$ is defined as $A \equiv a \cdot r \pmod{p}$. Given two p-residues, A and B, the Montgomery product of the two p-residues is shown below,

$$\begin{aligned} C &\equiv A \cdot B \cdot r^{-1} \pmod{p} \qquad (3.1) \\ &\equiv (a \cdot r) \cdot (b \cdot r) \cdot r^{-1} \pmod{p} \\ &\equiv (a \cdot b) \cdot r \pmod{p} \\ &\equiv c \cdot r \pmod{p}, \text{ where } 0 \le c < p \qquad (3.2) \end{aligned}$$

where r^{-1} is the inverse of $r \pmod{p}$, i.e. $r \cdot r^{-1} \equiv 1 \pmod{p}$. The Montgomery reduction algorithm also involves another quantity, p', which is an integer that satisfies $r \cdot r^{-1} - p \cdot p' = 1$. Here r^{-1} and p' can both be derived by the extended Euclidean algorithm described later in chapter 3.2.1. Therefore, the Montgomery product shown in equation 3.1 can be obtained by Montgomery multiplication algorithm.

Algorithm 3.1. ***(Montgomery Multiplication Algorithm)***

```
1. $T = A \cdot B$
2. $Q = T \cdot p' \bmod r$
3. $U = (T + Q \cdot p)/r$
4. if $U \ge p$ then $C = U - p$, else $C = U$
```

Proof. Assume that the lengths of A and B are both m-bit and the value T is a product having double length equal to $2m$. Step 2 simply converts Q to single length m. Note that in step 3, given $r \cdot r^{-1} - p \cdot p' = 1$ which leads to $p \cdot p' \equiv -1 \pmod{r}$ so that

$$Q \cdot p \equiv T \cdot p' \cdot p \pmod{r} \equiv -T \pmod{r} \qquad (3.3)$$

where $Q \cdot p$ is also a product having the same length as T and $T + Q \cdot p \equiv 0 \pmod{r}$ which concludes that it can be divisable by r and U is its quotient.

$$U \cdot r = T + Q \cdot p \equiv T \pmod{p} \qquad (3.4)$$

The result $U \equiv T \cdot r^{-1} \pmod{p} \equiv A \cdot B \cdot r^{-1} \pmod{p}$ is derived. Seeing that the modulus r is a little bigger than p and $0 \le A, B < p$, the value of $T/r = A \cdot (B/r)$ is smaller than p and such is the same case with $(Q \cdot p)/r$. Since U is an additive result of two p-residue value, step 4 ensure that the result C is in p-residue format, too. **Q.E.D.**

Among Montgomery multiplication algorithm, the operations of modulus r and division by r are both trivial operations since r is a constant with power of two. Thus Montgomery multiplication has the advantage of hardware implementation, and it's simpler and faster than traditional modular multiplication.

Observe in (3.2), the resulting value C of the Montgomery product is still in the p-residue format, that is to say, the output value from the Montgomery product can be used as input value to the next without converting from an ordinary residue to a p-residue. To convert an ordinary residue to a p-residue, let the integer a multiply by $r^2 \pmod{p}$, i.e. $2^{2m} \pmod{p}$, with the Montgomery product operation.

$$\begin{aligned} A &= \mathrm{MonMul}(a, r^2) \\ &\equiv a \cdot r^2 \cdot r^{-1} \pmod{p} \\ &\equiv a \cdot r \pmod{p} \end{aligned} \tag{3.5}$$

Note that the constant $r^2 \pmod{p}$ needs to be precomputed externally. Similarly, in order to convert the result A back to the ordinary residue, it can be achieved that A multiplies by a constant 1 with the Montgomery product operation.

$$\begin{aligned} a &= \mathrm{MonMul}(A, 1) \\ &\equiv A \cdot 1 \cdot r^{-1} \pmod{p} \\ &\equiv (a \cdot r) \cdot 1 \cdot r^{-1} \pmod{p} \end{aligned} \tag{3.6}$$

Here no extra constants need to be precomputed. Moreover, the p-residue can also be called Montgomery domain. However, it's not efficient to perform one single modular multiplication using Montgomery product since the conversion between Montgomery and real domain needs one more Montgomery product operation, and otherwise computation for the value p' also takes much time.

3.1.2 Modified Montgomery Multiplication Algorithm

There are a variety of ways to realize the Montgomery multiplication [17]. The radix-2 Montgomery multiplication algorithm [18] over $GF(p)$ is shown in Algorithm 3.2 and it can easily adapt the field $GF(p)$ for the field $GF(2^m)$. Algorithm 3.3 shows the binary version of the radix-2 Montgomery multiplication algorithm and it has been proven by [19].

Algorithm 3.2. *(**Montgomery Multiplication over** $GF(p)$)*

Input: *A, B and P, where $A, B \in GF(p)$ and P is the modulus p.*

Output: *U, where $U \equiv A \times B \times 2^{-m} \pmod{P}$.*

```
1. Let $a_{m-1}a_{m-2}\ldots a_1a_0$ be the binary representation of $A$.
2. Set $U = 0$.
3. For $i$ from $0$ to $m-1$ do
   3.1. Set $T = (U + a_iB)$.
   3.2. Set $U = (T + t_0P)/2$.
4. If $U \geq P$, then set $U = U - P$.
5. Output $U$.
```

Algorithm 3.3. *(**Montgomery Multiplication over** $GF(2^m)$)*

Input: *$A(x)$, $B(x)$ and $P(x)$, where $A(x)$, $B(x)$ and $P(x) \in GF(2^m)$, and $GF(2^m)$ is generated by $P(x)$.*

Output: *$U(x)$, where $U(x) \equiv A(x) \cdot B(x) \cdot x^{-m} \pmod{P(x)}$.*

```
1. Let $A(x) = \sum_{i=0}^{m-1} a_ix^i$, $\forall a_i \in GF(2)$, be the polynomial representation of $A(x)$.
2. Set $U(x) = 0$.
3. For $i$ from $0$ to $m-1$ do
   3.1. Set $T(x) = (U(x) + a_iB(x))$.
   3.2. Set $U(x) = (T(x) + t_0P(x))/x$.
4. Output $U(x)$.
```

The suffix i of the variable indicates the ith bit in the binary or polynomial representation of the variable, i.e. t_0 denotes the least significant bit of T. Note that the coefficients of the polynomial representation of $A(x)$, i.e. $a_{m-1}a_{m-2}\ldots a_1a_0$, are also the binary representation of the integer $A(2)$ since $A(2) = \sum_{i=0}^{m-1} a_i2^i$. The addition of two elements in $GF(2^m)$ is shown in the following equation,

$$A(x) + B(x) = \sum_{i=0}^{m-1} a_ix^i + \sum_{i=0}^{m-1} b_ix^i = \sum_{i=0}^{m-1}(a_i \oplus b_i)x^i \tag{3.7}$$

and the subtraction in $GF(2^m)$ is the same as addition in $GF(2^m)$. Furthermore, division by 2 in $GF(p)$ and division by x in $GF(2^m)$ are both shift operations.

3.2 Modular Inversion

Modular inversion is used in cryptographic applications such as the Diffie-Hellman key exchange [20], the public and private key pair generations in RSA and point addition operations in ECC. Given an m-bit modulus p, modular inversion computes the inversion of a non-zero field element $a \in GF(p)$. The inversion of a is denoted by $a^{-1} \pmod{p}$, where $a \cdot a^{-1} \equiv 1 \pmod{p}$. Furthermore, the multiplicative inverse of a exists if and only if a and p are relatively prime and its proof will be shown later.

3.2.1 Extended Euclidean Algorithm

In number theory, the Euclidean algorithm determines the greatest common divisor (GCD) of two integers. The GCD of a and b, written as $\gcd(a, b)$, is the largest positive integer that divides both a and b without remainder. Two integers are called relatively prime if and only if their GCD equals 1.

The extended Euclidean algorithm (EEA) [21] is an extension to the Euclidean algorithm and it can be used to solve the Bézout's identity, a linear diophantine equation. In number theory, Bézout's identity, named after Étienne Bézout (1730–1783), states that if a and b are non-negative integers, there exist integers x and y such that

$$ax + by = \gcd(a, b) \tag{3.8}$$

where x and y can be obtained by the EEA, but they are not uniquely determined. Set $x' = x - kb$ and $y' = y + ka$, then (x', y') is another solution to (3.8) since $ax' + by' = a(x - kb) + b(y + ka) = ax + by = \gcd(a, b)$. The following is the proof of Bézout's identity.

Proof. Let S be the set of all positive integers of $ax + by$, where x and y are integers. Since S is not empty, it has a smallest element by the well-ordering principle. Let $s = ax_t + by_t$ be the smallest element of the set S. According to the division algorithm, there are unique integers q and r that satisfy $a = sq + r$ with $0 \leq r < s$. Then

$$r = a - sq = a - (ax_t + by_t)q = a(1 - qx_t) + b(-qy_t) \tag{3.9}$$

Note that $(1 - qx_t)$ and $(-qy_t)$ are both integers so that r should be in the set S. But the condition $0 \leq r < s$ contradicts the choice of s as the smallest element of S. Thus r must be equal to 0 and it leads to $a = sq$ which indicates that a is divisible by s. Similarly, b is also divisible by s. Therefore s is one of the common divisor of a and b. Assume that c is another common divisor of a and b. Let $a = cq_1$ and $b = cq_2$, then

$$c|s = ax + by = c(q_1x + q_2y) \tag{3.10}$$

Since $c|s$ leads to $c \leq s$, it implies that s is the GCD of a and b, i.e. $s = \gcd(a, b)$. **Q.E.D.**

The EEA can solve the equation $ax + by = \gcd(x, y)$ efficiently. When a and b are relatively prime, i.e. $ax + by = 1$, then x is the multiplicative inverse of $a \pmod b$. The following algorithm is a variant of the EEA.

Algorithm 3.4. ***(Modified Extended Euclidean Algorithm over** $GF(p)$**)***
Input: *A and P, where $A \in GF(p)$ and P is the modulus p.*
Output: *R, where $R \equiv A^{-1} \pmod P$.*

```
1. Set $U = P$, $V = A$, $R = 0$ and $S = 1$.
2. While $V \neq 0$ do
   2.1. While $U$ is even do
        2.1.1. Set $U = U/2$.
        2.1.2. If $R$ is even, then set $R = R/2$.
        2.1.3. Else set $R = (R + P)/2$.
   2.2. While $V$ is even do
        2.2.1. Set $V = V/2$.
        2.2.2. If $S$ is even, then set $S = S/2$.
        2.2.3. Else set $S = (S + P)/2$.
   2.3. If $U > V$, then set $U = U - V$, $R = R - S$.
   2.4. Else if $V \geq U$, then set $V = V - U$, $S = S - R$.
3. Output $R \pmod P$.
```

The algorithm described above works by using the elimination method for solving the simultaneous equations below, where d and e are not really computed.

$$\begin{cases} S \cdot A & + & d \cdot P & = & V \\ R \cdot A & + & e \cdot P & = & U \end{cases} \tag{3.11}$$

The algorithm terminates when $V = 0$, in which case $U = 1$, and then $RA + eP = 1$ leads to $RA \equiv 1 \pmod P$, hence $R \equiv A^{-1} \pmod p$.

3.2.2 Montgomery Modular Inverse Algorithm

Based on the EEA, the algorithm proposed by Kaliski [12] computes the Montgomery modular inverse. Given an m-bit modulus p, the Montgomery modular inverse of a non-zero integer $a \in GF(p)$ is defined as the integer x,

$$x \equiv a^{-1}2^m \pmod{p} \tag{3.12}$$

The following algorithm rewrites the Kaliski Montgomery inverse algorithm with combination of the two phases in one algorithm. It is alternative that the output can be in the Montgomery domain or in the integer domain.

Algorithm 3.5. ***(Montgomery Modular Inverse Algorithm over*** $GF(p)$***)***

Input: *A and P, where $A \in GF(p)$ and P is the modulus p.*

Output: *U, where* $\begin{cases} U \equiv A^{-1}2^m \pmod{P}. \\ U \equiv A^{-1} \pmod{P}. \end{cases}$

```
1. Set $U = P$, $V = A$, $R = 0$ and $S = 1$.
2. Set $k = 0$, where $k$ is an integer with $m \le k < 2m$.
3. While $V > 0$ do
   3.1. If $U$ is even, then set $U = U/2$, $S = 2S$.
   3.2. Else if $V$ is even, then set $V = V/2$, $R = 2R$.
   3.3. Else if $U > V$, then set $U = (U - V)/2$, $R = R + S$ and $S = 2S$.
   3.4. Else if $V \ge U$, then set $V = (V - U)/2$, $S = S + R$ and $R = 2R$.
   3.5. Set $k = k + 1$.
4. For $i$ from 1 to $\begin{cases} k - m \\ k \end{cases}$ do
   4.1. If $R$ is even, then set $R = R/2$.
   4.2. Else set $R = (R + P)/2$.
5. If $R \ge P$, then set $R = 2P - R$, else set $R = P - R$.
6. Output $R$.
```

Note that the upper in the braces derives Montgomery inverse result and the lower derives modular inverse result, that is to say, output U is equivalent to $A^{-1}2^m \pmod{P}$ or $A^{-1} \pmod{P}$ depends on that the iteration count is $(k - m)$ or k in Step 4.

The Step 3 is iterative and it steadily reduces U or V by one bit in each iteration. Obviously, U and V initially have at most $2m$ bits in total since $2^{m-1} \le U < 2^m$ and

$0 < V < U$. But U equals 1 and V equals 0 in the last iteration, therefore, the iteration count in Step 3 takes no more than $(2m - 1)$ iterations. Similarly, U and V initially have at least $m + 1$ bits in total while V is equal to 1, thus, the iteration count in Step 3 takes no less than m iterations. So the boundary of k is $m \leq k < 2m$.

If the input is originally in the Montgomery domain, i.e. $A \equiv a2^m \pmod{P}$, then the output of the Montgomery inverse is given below.

$$\begin{aligned} U &\equiv (a2^m)^{-1}2^m \pmod{P} \\ &\equiv a^{-1} \pmod{P} \end{aligned} \tag{3.13}$$

In order to convert the output to the Montgomery domain, the conversion between integer and Montgomery domain needs an additional Montgomery multiplication operation. If the input is originally in the integer domain, i.e. $A \equiv a \pmod{P}$, then the output in the integer domain needs m iterations more than output in the Montgomery domain.

Table 3.1: Latency of Montgomery modular inverse from and to both domain

Domain			Latency (cycles)		
From	→	To	MonInv	MonMul	Total
Int	→	Int	$4m + 1$	-	$4m + 1$
Int	→	Mont	$3m + 1$	-	$3m + 1$
Mont	→	Int	$3m + 1$	-	$3m + 1$
Mont	→	Mont	$3m + 1$	$m + 1$	$4m + 2$

[1] Int means integer and Mont means Montgomery.

[2] MonInv denotes Montgomery inverse.

[3] MonMul denotes Montgomery multiplication.

[4] m is the bit length of the modulus p.

Table 3.1 shows each latency in worst case of the Montgomery inverse from and to both domain. In worst case, the latencies of the result for an m-bit Montgomery inverse and for real modular inverse are $(3m + 1)$ and $(4m + 1)$ clock cycles respectively, and the Montgomery multiplication operation over $GF(p)$ requires $(m + 1)$ clock cycles. Further, the Montgomery multiplication operation over $GF(2^m)$ requires only m clock cycles.

3.3 Modular Division

The modular division operation is traditionally accomplished by modular inversion followed by modular multiplication since the modular division is believed to be slow. It can be applied to the computation of the parameter λ in ECC.

3.3.1 Modular Division Algorithm

Given an m-bit modulus p, the modular division of two integers $a, b \in GF(p)$, where $b \neq 0$, is defined as the integer x,

$$x \equiv ab^{-1} \pmod{p} \tag{3.14}$$

The following algorithm shows a binary add-and-shift algorithm proposed by Sheueling Chang Shantz [22] for modular divison in a residue class.

Algorithm 3.6. *(**Modular Division Algorithm over** $GF(p)$)*

Input: *A, B and P, where $A, B \in GF(p)$ and P is the modulus p.*

Output: *U, where $U \equiv AB^{-1} \pmod{P}$.*

```
1. Set $U = P$, $V = B$, $R = 0$ and $S = A$.
2. While $U \neq V$ do
   2.1. If $U$ is even, then set $U = U/2$.
      2.1.1. If $R$ is even, then set $R = R/2$.
      2.1.1. Else set $R = (R + P)/2$.
   2.2. Else if $V$ is even, then set $V = V/2$.
      2.2.2. If $S$ is even, then set $S = S/2$.
      2.2.2. Else set $S = (S + P)/2$.
   2.3. Else if $U - V > 0$, then set $U = (U - V)/2$, and $R = R - S$.
      2.3.3. If $R < 0$, then set $R = R + P$.
      2.3.3. If $R$ is even, then set $R = R/2$. Else set $R = (R + P)/2$.
   2.4. Else if $V - U \geq 0$, then set $V = (V - U)/2$, and $S = S - R$.
      2.4.4. If $S < 0$, then set $S = S + P$.
      2.4.4. If $S$ is even, then set $S = S/2$. Else set $S = (S + P)/2$.
3. Output $R$.
```

The modular division algorithm above is an iterative process of additions, parity-testings, and shifts. Like Montgomery modular inverse algorithm, it reduces U or V by one bit. But it is different that in the last iteration, U and V are both equal to 1. Thus the entire division routine takes no more than $2(m-1)$ iterations.

Remember that in modular inverse algorithm, solving the simultaneous equation (3.11) can derive the inverse of the term A modular P. In order to solve this equation, it can be easily observed that an identical equation fits this equation and shown below.

$$\begin{cases} 1 \cdot A & + & 0 \cdot P & = & A \\ 0 \cdot A & + & 1 \cdot P & = & P \end{cases} \tag{3.15}$$

Therefore the initial value of the variable U, V, R and S in modular inverse algorithm are set to be P, A, 0 and 1, respectively.

Further, the modular division algorithm is similar to the modular inverse algorithm that it also works by using the elimination method for solving another simultaneous equations below, where d and e are not really computed, too.

$$\begin{cases} S \cdot (A^{-1}B) & + & d \cdot P & = & V \\ R \cdot (A^{-1}B) & + & e \cdot P & = & U \end{cases} \tag{3.16}$$

Note that the modular division algorithm terminates when $U = V = 1$, in which case $R \cdot (A^{-1}B) + eP = 1$ and $S \cdot (A^{-1}B) + dP = 1$. Since P is a prime, i.e. $\gcd(A^{-1}B, P) = 1$, the equation above definitely exists integer solutions that satisfy $R \cdot (A^{-1}B) + e \cdot P = 1$, where $R \cdot (A^{-1}B) \equiv 1 \pmod{P}$, that is, $R \equiv AB^{-1} \pmod{P}$. Similarly, $S \equiv AB^{-1}$ $\pmod{P}$, too. And it also can be easily obtained that an identical equation that fits equation (3.16) is written below,

$$\begin{cases} A \cdot (A^{-1}B) & + & d \cdot P & = & B \\ 0 \cdot (A^{-1}B) & + & 1 \cdot P & = & P \end{cases} \tag{3.17}$$

Thus the two algorithms of modular inverse and division only differ from the initial value of the variable S with $S = A$ instead. Although d is not really computed, in this case, $d = (-kB)$, where k is an integer that $AA^{-1} = 1 + kP$ since $AA^{-1} \equiv 1 \pmod{P}$.

$$A \cdot (A^{-1}B) + d \cdot P = (1 + kP)B + (-kB) \cdot P = B \tag{3.18}$$

Thus, there exists an integer d satisfy the equation (3.17).

3.3.2 Montgomery Modular Division Algorithm

Given an m-bit modulus p, the Montgomery modular division of the two integers $a, b \in GF(p)$, where $b \neq 0$, is defined as the integer q, where

$$q \equiv ab^{-1}2^m \pmod{p} \tag{3.19}$$

And given a primitive polynomial $p(x)$ with degree m which generates the $GF(2^m)$, the Montgomery modular division of the two element $a(x), b(x) \in GF(2^m)$, where $b(x) \neq 0$, is defined as the polynomial $q(x)$, where

$$q(x) \equiv a(x)b^{-1}(x)x^m \pmod{p(x)} \tag{3.20}$$

An alternative algorithm for calculating the Montgomery modular division or real modular division is proposed below and it is suitable for both $GF(p)$ and $GF(2^m)$.

Algorithm 3.7. *(**Montgomery Modular Division Algorithm over** $GF(p)$)*

Input: *A, B and P, where $A, B \in GF(p)$ and P is the modulus p.*

Output: *U, where* $\begin{cases} U \equiv AB^{-1}2^m \pmod{P}. \\ U \equiv AB^{-1} \pmod{P}. \end{cases}$

```
1. Set $U = P$, $V = B$, $R = 0$ and $S = A$.
2. Set $k = 0$, where $k$ is an integer with $m \leq k < 2m$.
3. While $V > 0$ do
    3.1. If $U$ is even, then set $U = U/2$, $S = 2S$.
    3.2. Else if $V$ is even, then set $V = V/2$, $R = 2R$.
    3.3. Else if $U - V > 0$, then set $U = (U - V)/2$, $R = R + S$ and $S = 2S$.
    3.4. Else if $V - U \geq 0$, then set $V = (V - U)/2$, $S = S + R$ and $R = 2R$.
    3.5. If $R \geq P$, then set $R = R - P$.
    3.6. If $S \geq P$, then set $S = S - P$.
    3.7. Set $k = k + 1$.
4. For $i$ from $1$ to $\begin{cases} k - m \\ k \end{cases}$ do
    4.1. If $R$ is even, then set $R = R/2$.
    4.2. Else set $R = (R + P)/2$.
5. Set $R = P - R$.
6. Output $R$.
```

Algorithm 3.8. *(**Montgomery Modular Division Algorithm over** $GF(2^m)$)*

Input: *$A(x)$, $B(x)$ and $P(x)$, where $A(x)$, $B(x)$ and $P(x) \in GF(2^m)$, and $GF(2^m)$ is generated by $P(x)$.*

Output: *$U(x)$, where* $\begin{cases} U \equiv A(x)B^{-1}(x)x^m \pmod{P(x)}. \\ U \equiv A(x)B^{-1}(x) \pmod{P(x)}. \end{cases}$

```
1. Set $U(x) = P(x)$, $V(x) = B(x)$, $R(x) = 0$ and $S(x) = A(x)$.
2. Set $k = 0$, where $k$ is an integer with $m \le k < 2m$.
3. While $V(x) \neq 0$ do
   3.1. If $U(2)$ is even, then set $U(x) = U(x)/x$, $S(x) = xS(x)$.
   3.2. Else if $V(2)$ is even, then set $V(x) = V(x)/x$, $R(x) = xR(x)$.
   3.3. Else if $U(2) - V(2) > 0$,
        then set $U(x) = (U(x) + V(x))/x$, $R(x) = R(x) + S(x)$ and $S(x) = xS(x)$.
   3.4. Else if $V(2) - U(2) \ge 0$,
        then set $V(x) = (V(x) + U(x))/x$, $S(x) = S(x) + R(x)$ and $R(x) = xR(x)$.
   3.5. If $deg(R) = deg(P)$, then set $R(x) = R(x) + P(x)$.
   3.6. If $deg(S) = deg(P)$, then set $S(x) = S(x) + P(x)$.
   3.7. Set $k = k + 1$.
4. For $i$ from 1 to $\begin{cases} k - m \\ k \end{cases}$ do
   4.1. If $R(2)$ is even, then set $R(x) = R(x)/x$.
   4.2. Else set $R(x) = (R(x) + P(x))/x$.
5. Output $R(x)$.
```

The proposed Montgomery modular division algorithm above is based on EEA and the binary GCD algorithm [23]. It mainly modifies the Montgomery modular inverse algorithm by setting the dividend to the initial value of S or $S(x)$. Note that all of the condition statements are almost the same in hardware design among these two algorithms, so it can compute not only Montgomery division but also modular division in both $GF(p)$ and $GF(2^m)$ fields. The two algorithms only differ from the field addition operations and the modular condition statements in Step 3.5 and Step 3.6.

Although the Montgomery division can be performed by Montgomery inverse followed by Montgomery multiplication, it takes longer latency and needs extra one more Montgomery multiplication operation while the output is in Montgomery domain. Table 3.2

shows the latency of traditional method using a Montgomery inversion followed by a Montgomery multiplication.

Table 3.2: Latency of modular division using traditional method from and to both domain

Domain			Latency (cycles)			
From →		To	MonInv	MonMul	MonMul	Total
Int	→	Int	$3m+1$	$m+1$	-	$4m+2$
Int	→	Mont	$3m+1$	$m+1$	$m+1$	$5m+3$
Mont	→	Int	$3m+1$	$m+1$	-	$4m+2$
Mont	→	Mont	$3m+1$	$m+1$	$m+1$	$5m+3$

The total latency of the method above seems worse in some cases especially in Montgomery domain, so the Montgomery modular division algorithm combine the inversion with multiplication to improve this shortcoming. The maximum number of iterations in Montgomery modular division algorithm is $3m$ or $4m$ depends on the output is in Montgomery or integer domain. Table 3.3 shows each latency in worst case of the Montgomery division from and to both domain. However, there are no additional Montgomery multiplication operations needed in each case.

Table 3.3: Latency of Montgomery modular division from and to both domain

Domain			Latency (cycles)	
From →		To	$GF(p)$	$GF(2^m)$
Int	→	Int	$4m$	$4m-1$
Int	→	Mont	$3m$	$3m-1$
Mont	→	Int	$4m$	$4m-1$
Mont	→	Mont	$3m$	$3m-1$

According to Table 4.3, the Montgomery modular division algorithm is consequently suitable for Montgomery domain implementation design. Therefore, the proposed algorithm is consistent with the way of implementing ECC on affine coordinate using Montgomery architecture in this thesis.

Chapter 4

Proposed Universal Architectures

The proposed architecture of the universal dual-field scalar multiplier on ECC is presented in this chapter. It is adapted to arbitrary lengths of the fields within a fixed length of the given design, and it is also adapted to both the prime finite fields, $GF(p)$, and the binary extension fields, $GF(2^m)$. All of the required materials for mathematical theorems have been mentioned in early chapters. Then all of these main components used in the scalar multiplier are detailed in the following subsections.

In this thesis, all of the design in hardware is implemented using RTL (Register-Transfer-Level) Verilog HDL (hardware description language) and synthesized on both application-specific integrated circuit (ASIC) and field-programmable gate arrays (FPGAs). The technology of ASIC design is using UMC[1] $0.18\mu m$ 1P6M CMOS process and the technology of FPGA design is using Xilinx[2] Virtex-II XC2V8000 platform FPGAs.

4.1 Universal Dual-field Montgomery Multiplier

In order to design an universal dual-field multiplier, the main problem is to deal with the variable binary extension field since the primitive polynomial and field degree are unfixed. An universal multiplier in $GF(2^m)$ using Montgomery multiplication algorithm is proposed [24] using bit-parallel architecture, but this architecture for bit-parallel computation is suitable for small field degree. Thus a series shift and addition operations is chosen to implement Montgomery multiplication algorithm.

[1]United Microelectronics Corporation. The SoC solution foundry. http://www.umc.com

[2]Xilinx, Inc. The developer and fabless manufacturer of FPGAs. http://www.xilinx.com

Back to the Algorithm 3.2 and Algorithm 3.3 which are mentioned in subsection 3.1.2. It is known that the Montgomery multiplication in $GF(p)$ always takes $(m+1)$ iterations, but the Montgomery multiplication in $GF(2^m)$ takes only m iterations which is one cycle less than the former since there is no modular operation in the last step. Besides, the addition operations in these two algorithms are also different that the adder in $GF(2^m)$ is the adder in $GF(p)$ without carry propagation.

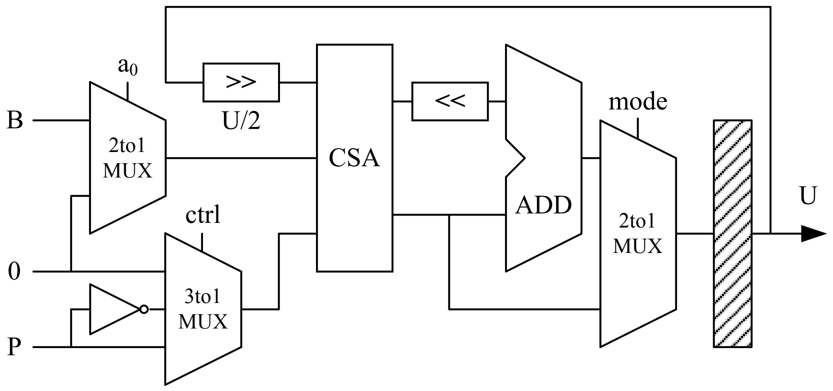

Figure 4.1: Universal dual-field Montgomery multiplier

Figure 4.1 shows the proposed architecture of the universal dual-field Montgomery multiplier. The finite field adder is achieved by a carry-save adder (CSA) follow by an adder. The sum and carry are separated by CSA, so the carry can independently be computed. The sum of CSA is directly chosen as the addition result in $GF(2^m)$ when the mode is in binary extension field. The carry of CSA is shifted left by one bit and added to the sum as the addition result in $GF(p)$ when the mode is in prime field.

However, the negative of P is needed at the end of the Algorithm 3.2 and $(-P)$ is represented by its 2's complement, P^*, where $P^* = 2^m - P$. Then

$$P^* = 2^m - P = (2^m - 1 - P) + 1 = \bar{P} + 1 \tag{4.1}$$

Note that $\bar{P}$ denotes the 1's complement of the positive integer P, where $\bar{P} = (2^m - 1) - P$. The 1's complement is to simply complement P bit-by-bit by replacing 0's with 1's and 1's with 0's. According to (4.1), the 2's complement of the integer P can be formed by complementing P bit-by-bit and then adding 1. It can be achieved in hardware by a bit-wise inverter except the least significant bit (LSB) of P since P is odd, that is, the

2's complement of P is always an odd, too. Moreover, the values of register U and A are shifted right by one bit and back to the input of CSA after each iteration finished.

There are some sample area and speed results for the Montgomery multiplier over $GF(p)$ given in [25]. Its architecture is based on [26] and implemented on FPGAs using Xilinx Virtex-E XCV2000e. Using this architecture, each multiplication operation requires $(m+5)$ clock cycles. Further, the architecture for the Montgomery multiplier over $GF(p)$ based on Algorithm 3.2 is almost the same as Figure 4.1 without the 2-to-1 multiplexer at the end. Each multiplication requires $(m+1)$ clock cycles. For more general comparison, the proposed design here is synthesized using the same technology with [25]. Table 4.1 lists the detailed area and speed results for both.

Table 4.1: Comparison of Montgomery multiplier in $GF(p)$

Bit-length		Proposed	A. Daly [25]
128-bit	Area (Slices)	629	646
	Frequency (MHz)	65.90	81.23
	Latency (cycles)	129	133
	Throughput (Mbit/s)	65.4	78.2
256-bit	Area (Slices)	1164	1292
	Frequency (MHz)	47.43	58.24
	Latency (cycles)	257	261
	Throughput (Mbit/s)	47.2	57.1
512-bit	Area (Slices)	2176	2588
	Frequency (MHz)	29.35	56.05
	Latency (cycles)	513	517
	Throughput (Mbit/s)	29.29	55.5

Note that [25] uses pipelined mux/add architecture to improve in clock speed when the bit-length of its multiplier exceeds the maximum carry chain length, i.e. 1 column of the FPGA. Its architecture is especially designed for the specific type of FPGA platforms. Since the clock period is not under constrain in this thesis, the proposed architecture is designed for smaller scale to simultaneously adapt to both $GF(p)$ and $GF(2^m)$.

The implementation results of the proposed universal dual-field Montgomery multiplier is given in Table 4.2. It shows the probably gate count synthesized at 100MHz and shows the area and speed results on FPGAs. Each number of gates in the field Gatecount consists of two parts which are non-combinational logic and combinational logic respectively. It doubles when the bit length doubles. The area is approximately in proportion to the bit length m of the field.

Table 4.2: Synthesize results for proposed universal dual-field Montgomery multiplier on ASIC and FPGA design

	ASIC		FPGA	
Bit-length	Gatecount (Gates)	Frequency (MHz)	Area (Slices)	Frequency (MHz)
64-bit	$4.2k$ $(1.9k + 2.3k)$	100	397	83.265
128-bit	$8.3k$ $(3.8k + 4.5k)$		733	65.811
256-bit	$16.3k$ $(7.4k + 8.9k)$		1387	47.432
512-bit	$32.1k$ $(14.5k + 17.6k)$		2266	27.118
1024-bit	$63.3k$ $(28.8k + 34.5k)$		4664	16.120

4.2 Universal Dual-field Montgomery Divider

The universal dual-field Montgomery divider is mainly used in Montgomery domain. It shortens the latency of the division in Montgomery domain by combination of an inversion and a multiplication. Table 4.3 shows the latency of modular division from and to both domain using Algorithm 3.6 method.

Table 4.3: The latency for Algorithm 3.6

Domain			Latency (cycles)		
From	→	To	MonDiv	MonMul	Total
Int	→	Int	$2(m-1)$	-	$2(m-1)$
Int	→	Mont	$2(m-1)$	$m+1$	$3m-1$
Mont	→	Int	$2(m-1)$	-	$2(m-1)$
Mont	→	Mont	$2(m-1)$	$m+1$	$3m-1$

In comparison with Table 3.3, the total latency for either domain to Montgomery domain is similar but it needs additional Montgomery multiplier in hardware. Table 4.4 shows the FPGA results of Algorithm 3.7 in comparison with other implementations [27] using Algorithm 3.6 with the same technology.

Table 4.4: A Comparison with FPGA results for modular division in $GF(p)$

	Proposed		A. Daly [27]	
Bit-length	Area (Slices)	Freq. (MHz)	Area (Slices)	Freq. (MHz)
64-bit	1163	30.361	1212	45
128-bit	2132	26.091	2215	31
256-bit	3262	15.429	3872	17

According to Algorithm 3.7 and Algorithm 3.8, note that all of the condition statements are almost the same in hardware design among these two algorithms. It reduces the complexity of the control flow for the two different fields. The control flow is simultaneously suitable for $GF(p)$ and $GF(2^m)$.

Figure 4.2 shows the circuit diagram for the registers U and V in the universal dula-field Montgomery divider. There are two subtracters in this circuit which compute $(U-V)$ and $(V-U)$ separately. One of them is reused to compute $(P-R)$ in order to share the subtracters. The term $(P-R)$ is needed to derive the correct result in $GF(p)$ when the division is finished at the last step. Further, it can use only one subtracter in only $GF(p)$ by substituting U for $-U$ in Algorithm 3.7.

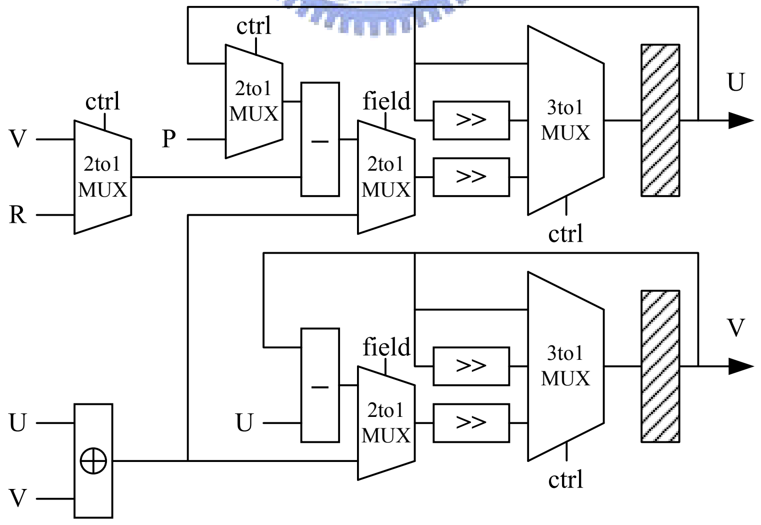

Figure 4.2: Universal dual-field Montgomery Divider Architecture for U and V

Figure 4.3 shows the circuit diagram for the registers R and S. Note that the source U denotes $(P - R)$ and is derived from the circuit mentioned above. Since the registers U and R will not simultaneously be equal to $(R + S)$ in the division algorithm, they can share the same hardware in this term $(R + S)$. The decoder is used to determine the modular operation in $GF(2^m)$ but also costs around 6% extra portion of area in FPGA. However, it is unnecessary in specific length design.

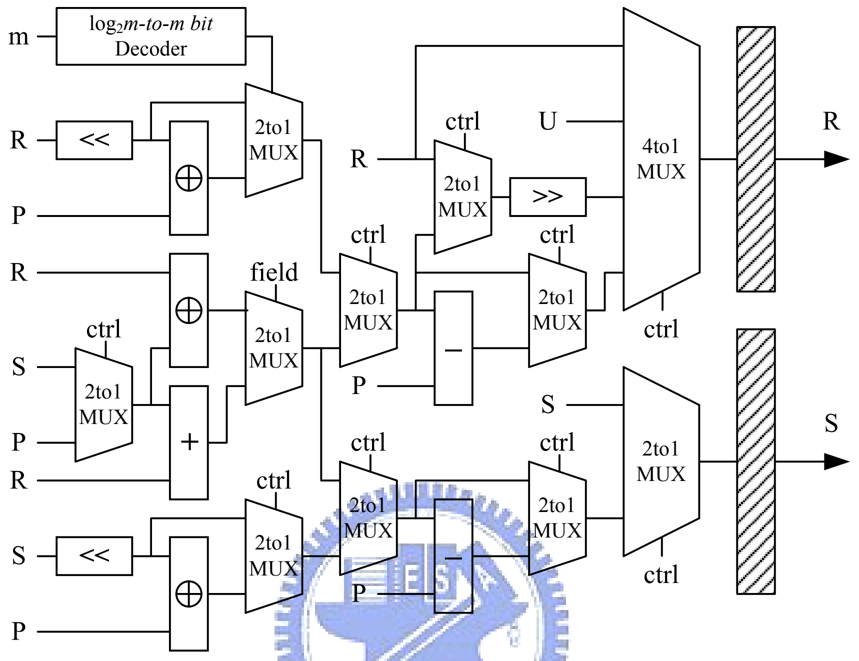

Figure 4.3: Universal dual-field Montgomery Divider Architecture for R and S

Table 4.5 shows the synthesize results for the universal dual-field Montgomery divider on ASIC and FPGA. The area is also approximately in proportion to the bit length m.

Table 4.5: Synthesize results for proposed universal dual-field Montgomery divider on ASIC and FPGA design

	ASIC		FPGA	
Bit-length	Gatecount (Gates)	Frequency (MHz)	Area (Slices)	Frequency (MHz)
64-bit	$10.3k$ $(2.6k + 7.7k)$	100	1095	52.366
128-bit	$20.8k$ $(5.0k + 15.8k)$		2164	41.917
256-bit	$42.1k$ $(10.0k + 32.1k)$		4535	28.243

Another algorithm for unified modular division in $GF(p)$ and $GF(2^m)$ is proposed in [28] and is implemented in [29]. Table 4.6 shows a comparison of the number of gates for two algorithms. The UMD algorithm in [28] is also suitable for both $GF(p)$ and $GF(2^m)$ but still needs additional multiplication for Montgomery domain.

Table 4.6: A Comparison with ASIC results for universal modular division algorithm

	Proposed		L. A. Tawalbeh [29]	
Bit-length	Gatecount (Gates)	Frequency (MHz)	Gatecount (Gates)	Frequency (MHz)
128-bit	$20.8k$	100	$22.8k$	100
256-bit	$42.1k$		$45.6k$	92

4.3 Universal Dual-field Scalar Multiplier

The most important arithmetic on elliptic curve applications is the scalar multiplication that computes kP, where k is an arbitrary integer and P is a point on elliptic curve. In scalar multiplication, the computation of the parameter λ is the most time consuming.

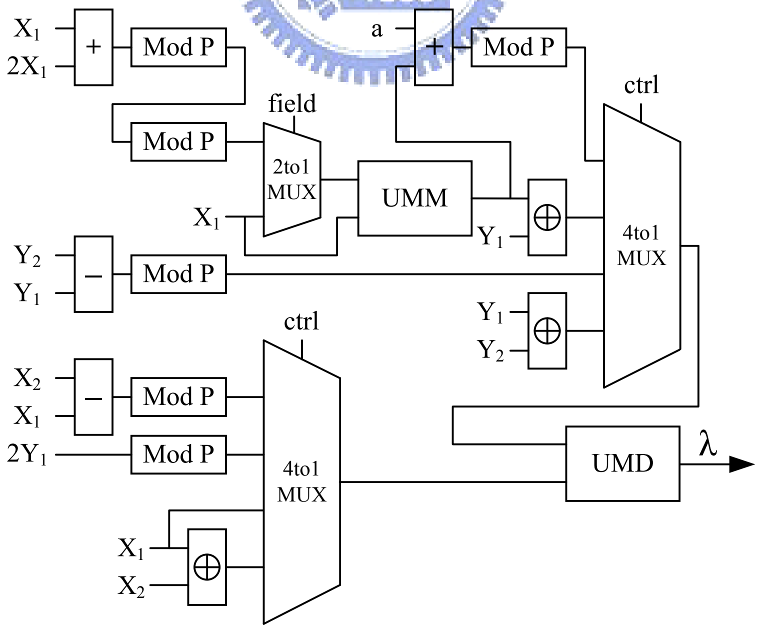

Figure 4.4: Circuit diagram for calculating λ

Figure 4.4 shows the circuit diagram for calculating the parameter λ. The block UMM and UMD denote an universal dual-field Montgomery multiplier and an universal dual-field Montgomery divider respectively. The block Mod P denotes the modular p operation once and requires one subtracter or adder and one multiplexer. Thus it is a great consumption of area in hardware.

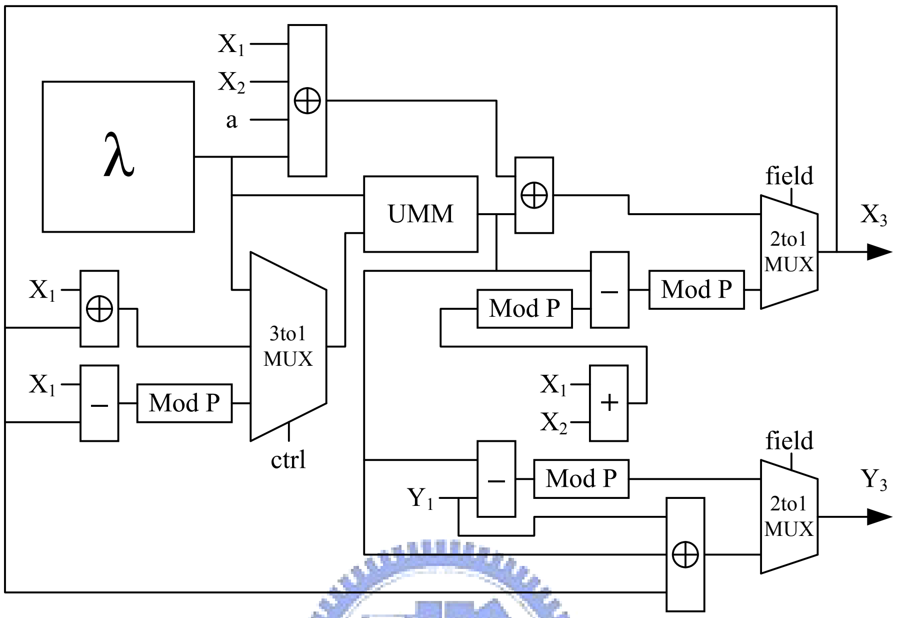

Figure 4.5: Universal dual-field point adder

Figure 4.5 shows the architecture of the universal dual-field point adder on elliptic curves. Except the λ block, there is one more UMM needed for both the outputs X_3 and Y_3. Since the multiplication does not simultaneous occur while calculating X_3 and Y_3, they can share the multiplication hardware.

Table 4.7 show the synthesize results for the proposed elliptic curve point adder. Note that the design here only works in Montgomery domain. The transformation between domains requires other Montgomery multipliers.

Table 4.7: Synthesize results for proposed elliptic curve point adder

	ASIC		FPGA	
Bit-length	Gatecount (Gates)	Frequency (MHz)	Area (Slices)	Frequency (MHz)
256-bit	$198.3k$ $(27.6k + 170.7k)$	75	12366	27.512

Figure 4.6 shows the entire architecture of the universal dual-field scalar multiplier. The block ECPA denotes the point adder. It takes five UMM blocks for domain transformation, however, the additional UMM blocks cost about 27% area of the entire design. It multiplies the constant $2^{2n} \pmod{p}$ to transform to Montgomery domain and multiplies the constant 1 to transform back to integer domain.

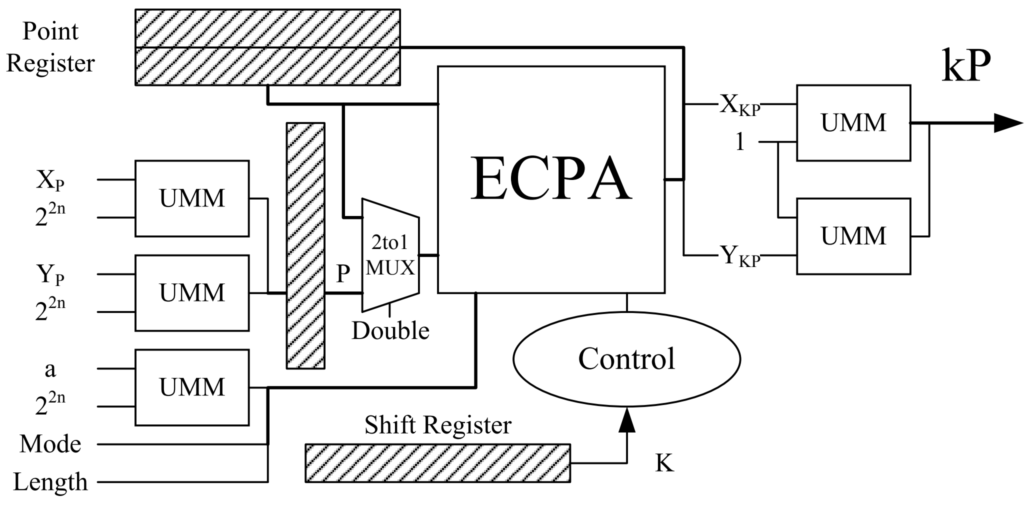

Figure 4.6: Universal dual-field scalar multiplier

The main function of the control circuit is the scalar multiplication. The Double-and-Add algorithm is used here and takes about 7% area of the design. This control circuit can be replaced by other more efficient algorithms mentioned in Chapter 2.3, but it certainly will take more circuits. Therefore, the control unit can be implemented using software for more flexibility. In most ECC applications, the base point or some parameters are already known, so that some variables can be pre-computed in advance and strored in additional memories in order to accelerate the computation. Table 4.8 shows the final results for the total design in this thesis. It performs the scalar multiplication on elliptic curves.

Table 4.8: Synthesize results for proposed elliptic curve scalar multiplier

	ASIC		FPGA	
Bit-length	Gatecount (Gates)	Frequency (MHz)	Area (Slices)	Frequency (MHz)
256-bit	$292.5k$ $(80.8k + 211.7k)$	75	18146	18.768

Chapter 5

Implementation Results

A total solutions for elliptic curve arithmetics in both software and hardware are given in this work. This chapter shows the hardware implementation results. The software simulation environment is constructed in both C and C++ programing languages. The design and test consideration are discussed in Chpater 5.1. The hardware implementation results and design flow are described in Chapter 5.2. The RTL synthesizer uses Synopsys[1] Design Compiler for ASIC and Xilinx XST or Synplicity[2] Synplify Pro for FPGA.

5.1 Design and Test Consideration

The hardware is designed to accelerate the operations on elliptic curves and it deals with different field parameters using Montgomery technique. The main part in hardware is the point operation on elliptic curves and the implementation of scalar multiplication on hardware uses only Double-and-Add algorithm which averagely takes more latency. The performance of scalar multiplication algorithm is not taken into consideration in this work since it can be improved efficiently by software or uses additional memories.

The Verilog code for this design was generated using the parameterized module for different values of m. The test patterns are generated randomly by software. The verification for the design uses not only hardware-software co-simulation but also confirms with the examples of NIST[3] publications for more confidence.

[1]Synopsys, Inc. http://www.synopsys.com/

[2]Synplicity, Inc. http://www.synplicity.com/

[3]National Institute of Standards and Technology. http://www.nist.gov/

5.2 Hardware Implementation

5.2.1 ASIC Implementation

Figure 5.1 illustrates the entire ASIC design and testing flow with various CAD (Computer Aided Design) tools. The design is done by pre-layout gate-level simulation but the pre-layout simulation can not calculate the circuit speed precisely. The results for post-layout gate-level simulation will be worse than the results shown in former.

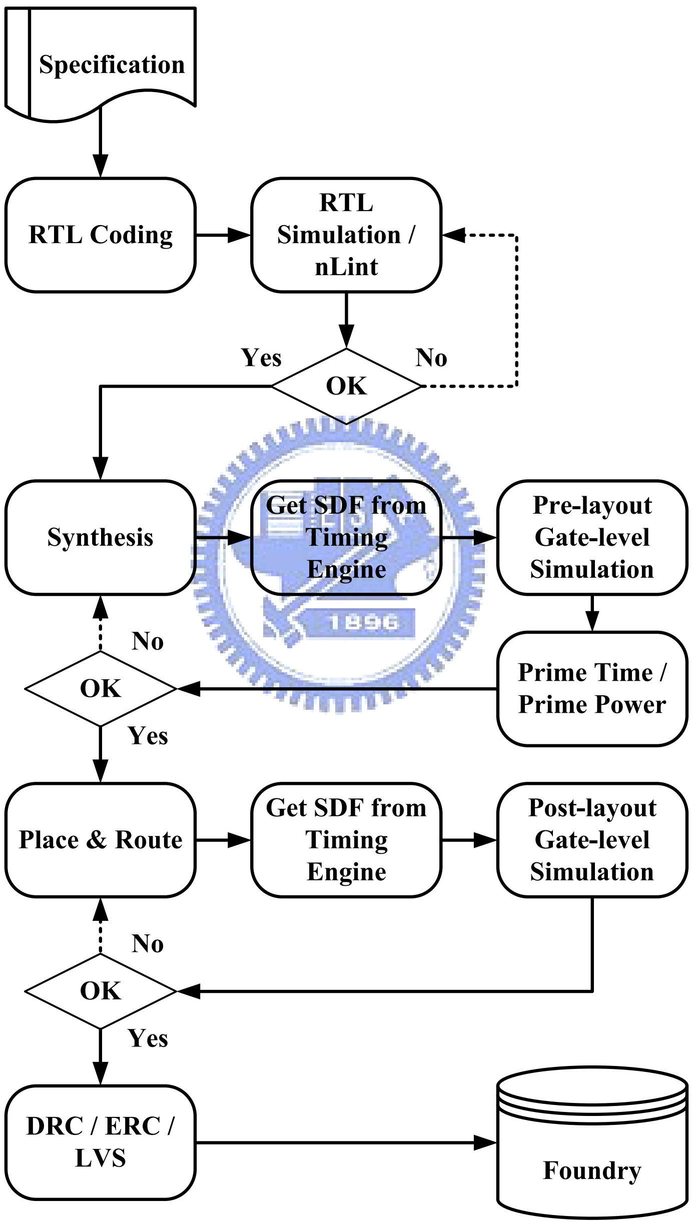

Figure 5.1: ASIC design flow

Table 5.1 shows a comparison for the ASIC performance of scalar multiplication. In this work, the execution time for computing kP in $GF(P_{192})$ is average 3.3 ms. The execution is probably range from 2.4 to 4.2 ms for the best and worst case, that is, the latency is probably range from $180k$ to $320k$ clock cycles.

In contrast to proposed design, the work [30] shows a great performance using a elliptic curve cryptographic processor. It has a Montgomery multiplier and uses projective plane to avoid inversion operations. In scalar multiplication, it uses software NAF method to reduce the number of 1 terms in k. However, the proposed design mainly shows a powerful dual-field arithmetic operator on elliptic curves by ASIC method.

In work [31], the design uses Fermat's Little Theorem for the modular inversion operation. However, it is not considered efficient in a large field design since the computation complexity increases significantly. The proposed Montgomery modular inversion or division algorithm based on EEA has an obviously improvement on the computation time for inversion computation. The division algorithm is chosen in this thesis because it always needs a division in elliptic curve point operations.

In software simulation on C, it takes around 300 ms averagely to do scalar multiplication once. Then the ECDSA takes about 1290 ms including signature and verification. The signature and verification have total four scalar multiplication operations. Therefore, the scalar multiplication spends the most time in ECDSA and requires extra hardware to accelerate its speed. The simulation results below show significant improvement on the computation time for scalar multiplication.

Table 5.1: Elliptic Curve Scalar Multiplication ASIC Performance Comparison

Author	A. Satoh [30]	G. Z. Lu [31]	Proposed
Field	$GF(P_{192})/GF(2^{160})$	$GF(P_{192})/GF(2^{192})$	$GF(P_{256})/GF(2^{256})$
Platform	.13μm CMOS	.25μm CMOS	.18μm CMOS
Gatecount (Gates)	$120.2k$	$26.7k$	$292.5k$
Frequency (MHz)	137.7	285.7	75
EC mult. (ms)	1.44/0.19	9.75/6.75	3.3
Note	64-bit multiplier	8PEs with $w = 8$bits	Universal dual-field architecture

[1] The timing for EC mult. of the proposed design is for 192-bit length.

5.2.2 FPGA Implementation

Figure 5.2 illustrates the FPGA design and testing flow in contrast to the ASIC design flow. In this thesis, since this work is mainly implemented on ASIC design, there is not any technique used to improve the performance on FPGA. Thus, there is no block RAM and specific length multiplier used to accelerate the speed on FPGA. Thus, the implementation results on FPGA is slightly worse in timing performance, but it is helpful in fast verification and gives reliable hardware information.

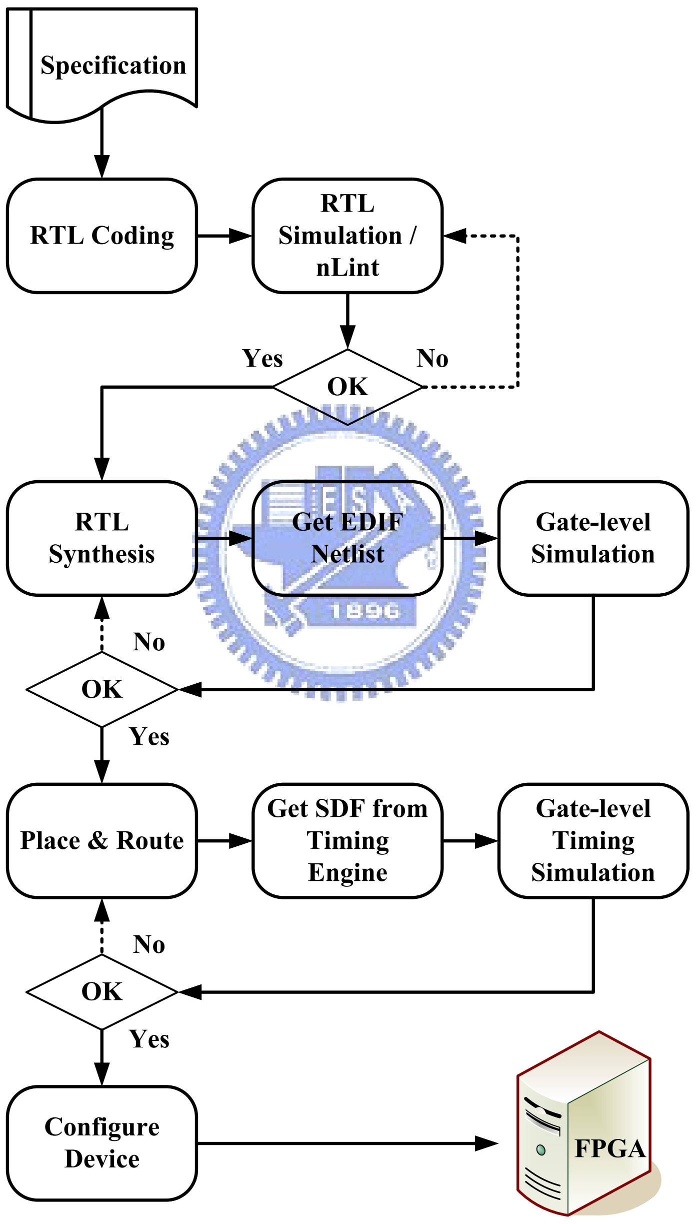

Figure 5.2: FPGA design flow

Table 5.2 shows a comparison for the FPGA performance of scalar multiplication. There are few similar parallel architecture of universal dual-field elliptic curve scalar multiplier, so the following table just lists some implementations for reference.

Table 5.2: Elliptic Curve Scalar Multiplication FPGA Performance Comparison

Author	Field	Platform	Area (Slices)	Freq. (MHz)	Latency (Cycles)	EC mult. (ms)
C. J. McIvor [32]	$GF(P_{256})$	Xilinx Pro XC2VP125	15755	39.46	$151.3k$	3.86
W. C. Hsu [33]	$GF(2^{163})$	Xilinx XC2V8000	8815 LUTs	90	$37k$	0.41
N. A. Saqib [34]	$GF(2^{191})$	Xilinx XCV3200E	18314 24BRAM	9.99	573	0.05
Leong [35]	$GF(2^{173})$	Microcoded Processor	-	-	$310k$	11.1
Proposed	$GF(2^{256})$	Xilinx XC2V8000	18146	18.768	$250k$	13.32

[1] The timing for EC mult. of the proposed design is for 192-bit length.

The authors in [32] have proposed much about the Montgomery techniques recent years. The latency of the Montgomery multiplication is especially shorter than the proposed design in this thesis. It takes only 32 clock cycles to perform one 256-bit multiplication and achieves by cascading 16×16-bit multipliers. The trade-off is the cost of area. The Montgomery multiplier here requires 11992 Slices. Despite of the amazing area consumption, it performs a fast operation speed for 256-bit scalar multiplication.

The work [33] and [34] shows the higher frequency and fewer latency design respectively. They both have a good performance on scalar multiplications. In [33], the design works on normal basis and uses projective plane method to avoid inversion operations. [35] shows the performance using microcoded EC processor.

Chapter 6

Conclusion

A total solution in hardware and software to the scalar multiplication on elliptic curves in both $GF(p)$ and $GF(2^m)$ is given in this thesis. In order to deal with various field conditions, the Montgomery techniques are employed. In affine coordinates, due to the slow division while calculating the parameter λ in point additions, a Montgomery modular division algorithm based on EEA is proposed instead of the inversion followed by a multiplication. The Montgomery divider plays an important role in elliptic curve scalar multiplication since it dominates 40% of total latency. Besides, the Montgomery multiplier is also an important operation. The implementation of these two functions in this work shows a considerable trade-off on area and speed, so it is suitable for hardware design to accelerate most complicated operations on elliptic curves.

According to the implementation result, it is synthesized using .18μm CMOS technology with 285k gates and using Xilinx Virtex-II XC2V8000 with 18146 slices in FPGA design. It takes about 300 ms to accomplish a scalar multiplication in software but takes only 3 ms in hardware. It is 100 times fast in speed. The result of proposed full parallel architecture for the scalar multiplier on elliptic curves seems a great consumption of area in comparison with others. However, the total area of entire scalar multiplier is not take into consideration in this thesis. It mainly shows the computation time in scalar multiplication using the proposed Montgomery method. It provides another way of implementing point additions in affine coordinates.

Appendix A

Elliptic Curve Cryptography

A.1 Elliptic Curve El-Gamal

The elliptic curve group forms finite cyclic group so that the elliptic curve group over finite field can be used to implement El-Gamal public cryptosystem.

1. Given the base point G, a key pair consists of a private key d, where $1 \leq d \leq n-1$, and a public key Q, where $Q = dG$.
2. Given Alice's key pair Q_A and d_A, where Q_A is public key and d_A is private key.
3. Given Bob's key pair Q_B and d_B, where Q_B is public key and d_B is private key.
4. Given a message P, Alice sends the point $(x_1, y_1) = (kG, P + kQ_B)$ to Bob using Bob's public key where k is a random integer chosen by Alice.
5. Bob uses his secret key d_B to derive the message P by computing $(y_1 - d_B x_1)$, where $y_1 - d_B x_1 = (P + kQ_B) - d_B(kG) = (P + kQ_B) - k(d_B G) = P + kQ_B - kQ_B = P$.

A.2 Elliptic Curve Diffe-Hellman

Elliptic Curve Diffe-Hellman (ECDH) is a variant of the Diffie-Hellman protocol using elliptic curve cryptography. It is a key agreement protocol that allows two parties to estabilish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

A.2.1 Key Exchange

1. Given the base point G, a key pair consists of a private key d, where $1 \leq d \leq n-1$, and a public key Q, where $Q = dG$.
2. Given Alice's key pair Q_A and d_A, where Q_A is public key and d_A is private key.
3. Given Bob's key pair Q_B and d_B, where Q_B is public key and d_B is private key.
4. Alice computes the point $(x_k, y_k) = d_A Q_B$ using Bob's public key.
5. Bob computes the point $d_B Q_A$ using Alice's public key.
6. Alice and Bob have the same key since $d_A Q_B = d_A(d_B G) = d_B(d_A G) = d_B Q_A$.

A.3 Elliptic Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the Digital Signature Algorithm (DSA) using elliptic curve cryptography.

A.3.1 Key Pair Generation

1. Select a statistically unique and unpredictable integer d in the interval $1 \leq d \leq n-1$, where n is the order of the point G, i.e. $nG = \mathcal{O}$.
2. Compute the point $Q(x_Q, y_Q) = dG$.
3. The key pair is Q and d, where Q is the public key, and d is the private key.

A.3.2 Signature Generation

1. Given a bit string, M, with arbitrary length as the message to be signed.
2. Message digesting:
   2.1. Compute the hash value $e = H(M)$ of the message using SHA-1.
   2.2. SHA-1 is specified in [36]. The output e is an integer with a length of 160 bits.
3. Elliptic curves computations:
   3.1. Select a statistically unique and unpredictable integer k, where $1 \leq k \leq n-1$.
   3.2. Compute the point $(x_1, y_1) = kG$.
4. Modular computations:
   4.1. Set $r = x_1 \bmod n$.
   4.2. Compute $s = k^{-1}(e + dr) \bmod n$.
5. The signature for M shall be the two integers, r and s.

A.3.3 Signature Verification

1. Given a received message M', and a received signature of two integers, r' and s'.
2. Message digesting:
   - 2.1. Compute the hash value $e' = H(M')$ of the message using SHA-1.
3. Modular computations:
   - 3.1. Compute $c = (s')^{-1} \bmod n$.
   - 3.2. Compute $u_1 = e'c$.
   - 3.3. Compute $u_2 = r'c$.
4. Elliptic curves computations:
   - 4.1. Compute the point $(x_2, y_2) = u_1G + u_2Q$.
5. Signature checking:
   - 5.1. Compute $v = x_2 \bmod n$.
   - 5.2. If $r' = v$, then the signature is verified.
   - 5.3. If $r' \neq v$, then reject the signature.

Bibliography

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in cryptology.* New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

[4] J. Cowie, B. Dodson, R. M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, and J. Zayer, "A world wide number field sieve factoring record: On to 512 bits." in *ASIACRYPT '96: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, 1996, pp. 382–394.

[5] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology - CRYPTO '85*, ser. Lecture Notes in Computer Science, H. C. Williams, Ed., vol. 218. Springer-Verlag, 1986, pp. 417–426.

[6] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, January 1987.

[7] *Recommendation on Key Management*, NIST Special Publications Std. 800-57, 2005.

[8] *Advanced Encryption Standard (AES)*, FIPS PUBS Std. 197, 2001.

[9] *Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, ANSI Std. X9.63, 2001.

[10] *Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, ANSI Std. X9.62, 2005.

[11] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, April 1985.

[12] B. S. Kaliski, Jr., "The Montgomery inverse and its applications," *IEEE Transactions on Computers*, vol. 44, no. 8, pp. 1064–1065, Auguest 1995.

[13] N. Koblitz, *A course in number theory and cryptography.* New York, NY, USA: Springer-Verlag New York, Inc., 1987.

[14] J. H. Silverman, *The Arithmetic of Elliptic Curves.* New York, NY, USA: Springer-Verlag New York, Inc., 1986.

[15] J. A. Solinas, "Efficient arithmetic on koblitz curves," *Des. Codes Cryptography*, vol. 19, no. 2-3, pp. 195–249, 2000.

[16] F. Morain and J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtraction chains," *Informatique théorique et Applications*, vol. 24, pp. 531–544, 1990.

[17] Çetin Kaya Koç, T. Acar, and B. S. Kaliski, Jr., "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, June 1996.

[18] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, October 1999.

[19] Ç. K. Koç and T. Acar, "Fast software exponentiation in $GF(2^k)$," in *ARITH '97: Proceedings of the 13th Symposium on Computer Arithmetic (ARITH '97).* Washington, DC, USA: IEEE Computer Society, 1997, p. 225.

[20] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Info. Theory*, vol. IT-22, pp. 644–654, November 1976.

[21] D. E. Knuth, *The Art of Computer Programming*, 3rd ed. Addison-Wesley, 1998, vol. 2, ch. Seminumerical Algorithms.

[22] S. C. Shantz, "From euclid's gcd to montgomery multiplication to the great divide," Sun Microsystems laboratories, Tech. Rep. TR-2001-95, June 2001.

[23] N. Takagi, "A vlsi algorithm for modular division based on the binary GCD algorithm," *IEICE Trans. Fundamentals*, vol. E81-A, no. 5, pp. 724–728, May 1998.

[24] C. C. Lin, F. K. Chang, H. C. Chang, and C. Y. Lee, "An universal VLSI architecture for bit-parallel computation in $GF(2^m)$," *IEEE Asia-Pacific Conference on Circuits and Systems*, vol. 1, pp. 125–128, December 2004.

[25] A. Daly, L. Marnane, and E. Popovici, "Fast modular inversion in the montgomery domain on reconfigurable logic," in *Irish signals and systems Conference (ISSC 2003)*, July 2003, pp. 363–367.

[26] A. Daly and W. Marnane, "Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. New York, USA: ACM Press, 2002, pp. 40–49.

[27] A. Daly, W. P. Marnane, T. Kerins, and E. M. Popovici, "Fast modular division for application in ecc on reconfigurable logic." in *FPL*, 2003, pp. 786–795.

[28] A. Tenca and L. Tawalbeh, "Algorithm for unified modular division in $GF(p)$ and $GF(2^n)$ suitable for cryptographic hardware," *Electronics Letters*, vol. 40, no. 5, pp. 304–306, 2004.

[29] L. A. Tawalbeh, A. F. Tenca, S. Park, and C. K. Koç, "Use of elliptic curves in cryptography," in *Thirty-Eighth Asilomar Conference on Signals, Systems, and Computers*, vol. 1, November 2004, pp. 483–487.

[30] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, 2003.

[31] G. Z. Lu, "Hardware implementation of elliptic curve cryptosystem over finite fields $GF(p)$ and $GF(2^m)$," Master's thesis, National Chiao Tung University, 2004.

[32] C. J. McIvor, M. McLoone, and J. V. McCanny, "Hardware elliptic curve cryptographic processor over $GF(p)$," *IEEE Transactions on Circuits and Systems*, vol. 53, no. 9, pp. 1946–1957, September 2006.

[33] W. C. Hsu, "Design and implementation for elliptic curve cryptosystems," Master's thesis, National Chiao Tung University, 2005.

[34] N. A. Saqib, F. Rodriguez-Henriquez, and A. Diaz-Perez, "A parallel architecture for fast computation of elliptic curve scalar multiplication over $GF(2^m)$," *ipdps*, vol. 04, p. 144a, 2004.

[35] P. H. W. Leong and I. K. H. Leung, "A microcoded elliptic curve processor using fpga technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 5, pp. 550–559, October 2002.

[36] *Public Key Cryptography Using Irreversible Algorithms - Part 2: The Secure Hash Algorithm (SHA-1)*, ANSI Std. X9.30, 1997.

作者簡介

姓名：劉耀仁

出生：台北市

學歷：建安國小、大安國中、建國中學

89.9 ~ 93.6 國立交通大學電子工程學系

93.9 ~ 96.1 國立交通大學電子研究所系統組

得獎

93 學年度大學校院積體電路設計競賽不分組 FPGA Xilinx 類 佳作