DESIGN AND IMPLEMENTATION OF A CONSTANT-TIME FPGA ACCELERATOR FOR FAST ELLIPTIC CURVE CRYPTOGRAPHY

by
Atıl Utku Ay

Submitted to the Graduate School of Engineering and
Natural Sciences in partial fulfilment of the requirements
for the degree of Master of Science

Sabancı University
August, 2016

DESIGN AND IMPLEMENTATION OF A CONSTANT-TIME FPGA ACCELERATOR FOR FAST ELLIPTIC CURVE CRYPTOGRAPHY

Approved by:

Prof. Dr. Erkay Savaş
(Thesis Supervisor) .....

Assoc. Prof. Dr. Ayhan Bozkurt .....

Asst. Prof. Dr. Erdinç Öztürk .....

Date of Approval: 10/09/2016

ABSTRACT

DESIGN AND IMPLEMENTATION OF A CONSTANT-TIME FPGA ACCELERATOR FOR FAST ELLIPTIC CURVE CRYPTOGRAPHY

ATIL UTKU AY

M.Sc. Thesis, August 2016

Supervisor: Prof. Dr. Erkay Savaş

Keywords: GLS curves, scalar multiplication hardware accelerators, digit-based multipliers, Karatsuba multipliers, FPGA

Elliptic Curve Cryptography (ECC) is one of the most popular public-key cryptosystems (PKC) today. Relatively shorter key lengths used in ECC compared to other popular PKCs and its potential for faster and more efficient implementations, both in software and in hardware, make it popular in industry and academia. In this thesis, we propose a scalar multiplication hardware accelerator that computes a constant-time variable-base point multiplication over the Galbraith-Lin-Scott (GLS) family of binary elliptic curves. Our hardware design is specifically customized for the quadratic extension field $\mathbb{F}_{2^{2n}}$, with $n = 127$, which provides a security level close to 128 bits. We experiment with digit-based and Karatsuba multipliers for performing $\mathbb{F}_{2^{127}}$ arithmetic used in GLS elliptic curves and report the time and area performances obtained by these two classes of multipliers. The real hardware implementation of our design achieves a delay of about 3.98 μs for computing one scalar multiplication on a XILINX KINTEX-7 FPGA device. This result clearly demonstrates that the proposed design claims the current speed record for this operation at or around the 128-bit security level for any hardware or software implementation reported in the literature.

ÖZET

HIZLI ELİPTİK EĞRİ KRİPTOGRAFİ İÇİN SABİT ZAMANLI, ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ HIZLANDIRICISININ TASARIMI VE GERÇEKLENMESİ

ATIL UTKU AY

Yüksek Lisans Tezi, Ağustos 2016

Tez Danışmanı: Prof. Dr. Erkay Savaş

Anahtar Kelimeler: GLS eğrileri, eliptik eğri nokta çarpımı için donanım hızlandırıcıları, basamak-tabanlı çarpıcılar, Karatsuba çarpıcıları, Alanda Programlanabilir Kapı Dizileri

Eliptik Eğri Kriptografi (EEK) günümüzde en sık kullanılan Açık Anahtarlı Şifreleme (AAŞ) türlerinden birisidir. Diğer AAŞ türlerine kıyasla EEK'nin kısa anahtar boyu ve daha hızlı ve verimli gerçekleme imkanı vermesi, onu hem endüstriyel hem de akademik çevrelerde popüler hale getirmektedir. Bu tez kapsamında, Galbraith-Lin-Scott (GLS) ailesine mensup eliptik eğriler üzerinde, eliptik eğri nokta çarpımı için, sabit zamanda çalışan bir donanım hızlandırıcı mimarisi tasarımı öneriyoruz. Bu donanım mimarisi, yaklaşık 128-bitlik güvenlik düzeyi sağlayan, $n = 127$ ile ikinci dereceye genişlemiş $\mathbb{F}_{2^{2n}}$ cebrik cismi için özelleştirilmiştir. Tez kapsamında, $\mathbb{F}_{2^{2n}}$ cebrik cismi üzerinde tanımlanmış GLS eğrileri aritmetiğini gerçekleyen basamak-temelli ve Karatsuba çarpma devreleri üzerinde denemeler gerçekleştirilmiş ve elde edilen alan ve zaman başarımları rapor edilmiştir. Tasarımın XILINX KINTEX-7 Alanda Programlanabilir Kapı Dizileri cihazı üzerinde gerçek donanım gerçeklemesi, bir eliptik eğri nokta çarpım işlemini, 3.98μ saniyede tamamlayabilmektedir. Bu süre, bu tezdeki tasarımın, bu işlem için literatürde rapor edilmiş 128 bit ve 128

bit'e yakın güvenlik düzeylerindeki tüm yazılım ve donanım uygulamalarından daha hızlı çalıştığını göstermektedir.

to my beloved family

Acknowledgements

First of all, I am grateful to my thesis advisor Prof. Dr. Erkay Savaş for his support throughout my academic life. This thesis is presented with the help of his immense knowledge, perfect guidance and endless patience. I also would like to thank to my thesis jury, Assoc. Prof. Dr. Ayhan Bozkurt and Asst. Prof. Dr. Erdinç Oztürk for their valuable time.

I would like to express my gratitude to Assoc. Prof. Francisco Rodríguez-Henríquez and Asst. Prof. Dr. Erdinç Oztürk for their cooperation throughout the research process. I will always remember and appreciate their help.

I am thankful to all my friends from Cryptography and Information Security Lab. They provide a perfect research and friendship environment. I survive all hardships with their support. In addition, I thank everybody who support me materially and spiritually during my educational life.

Last, but not least, I would like to thank my parents Ayşe Ay and Rasim Ay. I am thankful for their unlimited support and endless love throughout my life.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Elliptic curve cryptography (ECC), which is the most popular public-key cryptosystem after RSA, was proposed by Miller [26] and Koblitz [20] independently in mid-1980s. It uses relatively shorter keys for the same security level compared to RSA, and offers faster and efficient implementations, both in software and hardware. As a result, ECC implementations are highly popular in industry and academia. For instance, the Internet Engineering Task Force has recently announced that the Transport Layer Security (TLS) protocol version 1.3, will not use cipher suites based on RSA key transport primitives anymore [36]. From now on, the Ephemeral Diffie-Hellman and the Elliptic Curve Ephemeral Diffie-Hellman are convenient methods for establishing a TLS shared secret on secure client-server communications. The main reason of this change is that both of these methods provide the *perfect forward secrecy* feature.

Elliptic curves over binary extension fields, $\mathbb{F}_{2^n}$, are special curves as they are suitable for fast implementations by taking advantage of the carry-free nature of the $\mathbb{F}_{2^n}$ arithmetic. Implementation of quadratic extensions over $\mathbb{F}_{2^n}$ is also an efficient way to increase the security level.

The Galbraith-Lin-Scott (GLS) elliptic curves [10, 12] over $\mathbb{F}_{q^2}$, where $q = 2^n$, offer very profitable curve arithmetic, which requires only five multiplication, five squaring and three addition operations in $\mathbb{F}_{q^2}$ per each iteration of the Montgomery

Ladder algorithm used to implement an elliptic curve point multiplication, which is the most time consuming operation in ECC. Moreover, the fact that these operations can be performed in parallel and that there is an efficiently computable endomorphism make the GLS curves favorable to implement in hardware. Furthermore, certain field operations such as squaring and addition can be scheduled in such a way that they are computed by fully combinational circuits, without increasing the clock count and cycle time. In other words, they can be virtually computed free of cost. The security of a specific instance of a GLS curve against the gGHS attack can be verified [4].

There are a number of works on accelerating ECC arithmetic both in hardware [1, 16, 17, 19, 21, 33, 35, 40, 41] and in software [2, 5, 8, 29, 30] implementations, which report highly competitive timing results. The lower unit cost, when small volume of chip is needed, and simpler design process compared to ASIC makes FPGA devices highly attractive as target hardware platforms for cryptographic hardware accelerators. Extensive optimizations at each levels from the $\mathbb{F}_q$ arithmetic to ECC arithmetic, careful parameter selection and a holistic approach during the integration of all components are required to implement the best design because of the multi-leveled nature of ECC operations. The architectural design choices (e.g., the digit size of polynomial multipliers, data path arrangements such as the number of pipeline stages) can be made depending on the given specifics of the target device in order to obtain best performing implementations in hardware.

In this thesis, we propose a hardware accelerator for elliptic curve scalar point multiplication on FPGA. Our implementation offers approximately 128–bit security level and a first-level side-channel protection by using the quadratic field arithmetic and the two-dimensional endomorphism specific to binary GLS curves. After analyzing different algorithms in the literature and experimenting with various architectural design options for our target device, we implemented the fastest design for 128–bit security level in the literature, both in hardware and in software. Moreover, the efficiency of our design, which is measured in terms of area-time product

per bit, is highly competitive.

The organization of the thesis can be outlined as follows. Chapter 2 provides background information on finite field arithmetic, elliptic curve cryptography, specifically GLS elliptic curves and its security, and also a brief information about FPGA. In Chapter 3, the design of the proposed architecture is presented in detail. Then, our implementation results are reported in Chapter 4. In addition, recent elliptic curve scalar multiplication hardware accelerators and some software implementations in high-end microprocessors in the literature are reported and compared with our implementation in the same chapter. Lastly, the thesis is concluded by summarizing the achievements and pointing out directions for further research in the field in Chapter 5.

2 Background

In this section, we firstly provide mathematical background on binary extension finite field arithmetic and information on public key cryptography. Then, we give details on elliptic curve cryptography. Finally, we give introductory background information about FPGA devices.

2.1 Binary Extension Finite Field Arithmetic

A Finite Field is a mathematical object with finite number of elements, in which addition, subtraction, multiplication and division (except divison by zero) operations are defined. $GF(q)$ or $\mathbb{F}_q$, which is preferred in this thesis, are two alternative notations of finite fields. The integer q stands for the order of field, which basically means the number of elements in the field. The order of a field is always either a prime number or a power of a prime number. In other words, the order of a field has to be p^k, where p is a prime number and k is a positive integer.

$\mathbb{F}_{2^n}$ is called as binary extension field. The elements of $\mathbb{F}_{2^n}$ can be considered as binary polynomials of degree at most $n-1$, whose coefficients are either 0 or 1. In other words, the polynomial $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} \dots a_1x^1 + a_0x^0$, where $a_i \in \{0,1\}$ for $i = 0, \dots, n-1$ is an element of $\mathbb{F}_{2^n}$. In digital systems the polynomials are represented as the binary string of their coefficients. For instance, a 5 bit binary sequence 11011 is used to represent $x^4 + x^3 + x + 1 \in \mathbb{F}_{2^5}$.

All extension fields can be defined using an irreducible polynomial, which is used for reduction operation when the degree of the resulting polynomial after an arithmetic operation exceeds $n-1$. Reduction operation is simply a polynomial division operation that yields only the remainder. For instance, polynomial $x^7 + x^3 + x + 1$ can be reduced using the irreducible polynomial $x^5 + x^3 + 1$ of $\mathbb{F}_{2^5}$ into $x^2 + x$. As mentioned before, the coefficients of polynomials are either 0 or 1 and operations on coefficients are performed modulo 2. Thus, if a coefficient becomes greater than 1 in any operation, it is reduced to modulo 2.

2.1.1 Addition in $\mathbb{F}_{2^n}$

Addition operation in $\mathbb{F}_{2^n}$ can be performed as basic polynomial addition and applying modulo 2 arithmetic on the coefficients of the polynomials. Suppose $a = x^4 + x^2 + x + 1$ and $b = x^3 + x^2$ are two polynomials in $\mathbb{F}_{2^5}$. Polynomial addition, $a+b$, outputs $x^4 + x^3 + 2x^2 + x + 1$. $2x^2$ is equal to 0 in binary field. Consequently, $a + b = x^4 + x^3 + x + 1$. In digital systems, a, b and $a + b$ are represented as 10111_2, 01100_2 and 11011_2, respectively. Note that, polynomial addition can be computed by bitwise XORing of binary representations of polynomials.

2.1.2 Subtraction in $\mathbb{F}_{2^n}$

Subtraction and addition are inverse operations. As mentioned in 2.1.1, addition operation can be performed as XOR in $\mathbb{F}_{2^n}$ and the inverse of XOR operation is itself. Therefore, subtraction can also be computed by XOR of two numbers. In other words, addition and subtraction are identical operations in $\mathbb{F}_{2^n}$.

2.1.3 Multiplication in $\mathbb{F}_{2^n}$

Let $a = x^4 + x^2 + x + 1$ and $b = x^2 + 1$ be two polynomials in $\mathbb{F}_{2^5}$ with irreducible polynomial $x^5 + x^2 + 1$. In order to compute the multiplication of a and b, the polynomial multiplication is performed at first. This operation outputs $a \times b = x^2(x^4 + x^2 + x + 1) + x^4 + x^2 + x + 1 = x^6 + x^3 + x + 1$. Note that coefficients are

reduced to modulo 2. Then, the output is reduced by the irreducible polynomial of $x^5 + x^2 + 1$. Consequently the result is obtained as 1.

2.1.4 Division and Multiplicative Inverse in $\mathbb{F}_{2^n}$

Let a and b be two polynomial in $\mathbb{F}_{2^n}$. The division, $\frac{a}{b} \bmod f(x)$, can be expressed as $a \times b^{-1} \bmod f(x)$, where b^{-1} is the multiplicative inverse of b with respect to $f(x)$. In other words, $b \times b^{-1} = 1 \bmod f(x)$. In order to compute the multiplicative inverse of an element in $\mathbb{F}_{2^n}$, there are various methods such as extended Euclidean algorithm. Computing multiplicative inverse in $\mathbb{F}_{2^n}$ is more expensive than the other arithmetic operations in $\mathbb{F}_{2^n}$.

2.2 Public Key Cryptography

In cryptography, cryptosystems are categorized according to their encryption and decryption methods as Private Key (or Symmetric Key) Cryptosystem and Public Key (or Asymmetric Key) Cryptosystem. In Private Key Cryptosystems, the same key is used by all communicating parties. On the other hand, a public and private key pair is generated in Public Key Cryptosystems [PKCs] for encryption and decryption operations. The working principle of PKCs can be explained as follows. A party, which is called as the key owner, shares its public key with other parties. The other parties use this public key for encryption before they send their messages to the key owner who decrypts the encrypted message by using his/her private key. As the message encrypted by a public key can only be decrypted using the corresponding private key, the key owner is the only party that can decrypt the message. Public and private keys are not the same but related with each other and generated by using a key generation algorithm. It is computationally infeasible to compute the private key given the corresponding public key. This is the foundation of the security arguments for public key cryptography.

PKCs are commonly used to overcome today's security problems such as key distribution, authentication and integrity. RSA [34], Diffie-Hellman [6], Elliptic

Curve Cryptography [20], El-Gamal [7], Digital Signature Standart [28] and Paillier Cyrptosystem [31] are some examples of of the PKCs.

2.3 Elliptic Curve Cryptography

Elliptic Curve Cryptography [ECC], which is proposed by Miller [26] and Koblitz [20] independently, is one of the most popular public key cryptosystems in use today. The elliptic curve algebra using finite field arithmetic is the base of ECC. The security of ECC is based on the hardness of computing computing discrete logarithm in elliptic curve group. This problem can basically be defined as follows. Consider an elliptic curve, defined over a field $\mathbb{F}_q$. The point P be generates the elliptic curve group, whose order is r. Let Q be another point on the elliptic curve group such that $Q = kP$, where $k \in [0, r-1]$. The Elliptic Curve Discrete Logarithm Problem(ECDLP) is the problem of computing k for given P and Q.

ECC is advantageous to implement on platforms, which have limited resource such as energy and memory, thanks to its relatively shorter key size. ECC can provide the same level of security by using much smaller key size compared to RSA.

2.3.1 Elliptic Curves over Binary Fields $\mathbb{F}_{2^n}$

The Weierstrass equation [39], which is shown in Eq 1, is used in most of ECC systems.

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{1}$$

For binary curves, we use the simplified equation

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

where, $b \neq 0$. For the field, $\mathbb{F}_q$, where $q = 2^n$, the elements of the binary field are represented using at most n bits. The arithmetic operations in $\mathbb{F}_q$ are defined in Section 2.1.

A solution (x, y) to Eq 2 in $\mathbb{F}_{2^n}$ is called an elliptic curve point and all such solutions (i.e., points) along with an abstract point referred as point at infinity form an algebraic group of finite elements under addition operation. The point at infinity serves as the identity element of the elliptic curve group. For the rules for adding two elliptic curve points, one can profitably refer to [25]. Choosing a larger n provides higher level of security as it also increases the number of points on the curve, as a larger group generally implies harder discrete logarithm problem.

2.3.1.1 Point Addition

The point addition is one of the elliptic curve operations. This operation takes two different points on the curve as input and gives a point, which is also on the curve. The addition operation can be performed for any two different points on the curve.

Let P, Q and Z be three different points on a binary elliptic curve such that $P = (x_p, y_p)$, $Q = (x_q, y_q)$ and $Z = (x_z, y_z)$. The point addition operation, $Z = P + Q$, is performed as follows. Firstly, we need to calculate the slope of the line, t, which passes through P and Q as

$$t = \frac{y_p + y_q}{x_p + x_q}.$$

Then, x_z and y_z are computed as follows

$$x_z = t^2 + t + x_p + x_q + a$$

$$y_z = t \times (x_p + x_z) + x_z + y_p.$$

Note that, a is a parameter of the binary elliptic curve. One special case in point addition operation is subtracting a point from itself. In other words, Let P and Q be two points on an elliptic curve and $P = -Q$, i.e., $Q = (x_p, x_p + y_p)$. In this case, the slope of the line becomes infinity and $P + Q = \mathcal{O}$, where $\mathcal{O}$ is the point at infinity.

2.3.1.2 Point Doubling

For additive group of elliptic curve points, addition operation must be defined for all all points. However, the slope is computed as $\frac{0}{0}$, when the two points are selected as the same, and the point addition operation cannot be performed. In order to overcome this problem, the point doubling operation, which is the addition of a point to itself, is defined as a variant of point addition operation.

Let P and Z be two points on a binary elliptic curve such that $P = (x_p, y_p)$ and $Z = (x_z, y_z)$. For point doubling operation, $Z = 2P$, the slope, t, must be computed at first as follows.

$$t = \frac{x_p + y_p}{x_p}$$

Here, t is the slope of the line, which intersects with the curve at point Z and is tangent to the elliptic curve at point P. Then, x_z and y_z are computed as follows

$$x_z = s^2 + s + a$$

$$y_z = {x_p}^2 + (s + 1) \times x_z.$$

2.3.2 GLS Binary Elliptic Curves

Let $\mathbb{F}_{q^2}$ be a quadratic extension of the field $\mathbb{F}_q$, with $q = 2^n$. As mentioned in 2.3.1, the binary elliptic curve over $\mathbb{F}_{2^n}$ is generated by using the equation

$$E/\mathbb{F}_q : y^2 + xy = x^3 + ax^2 + b, \tag{3}$$

with $Tr(a) = 1$, and $b \neq 0$, where $Tr : a \in \mathbb{F}_{2^n} \rightarrow \sum_{i=0}^{n-1} a^{2^i}$ is defined as the trace function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$. The size of the the elliptic curve group, i.e. $\#E(\mathbb{F}_q)$, is $q + t - 1$, where t represents the trace of Frobenious of elliptic curve over $\mathbb{F}_q$. In addition, it is given that for the same elliptic curve over the quadratic extension field $\mathbb{F}_{q^2} = \mathbb{F}_{2^{2n}}$, we have $\#E(\mathbb{F}_{q^2}) = (q + 1)^2 - t^2$. Let a' be an element in $\mathbb{F}_{q^2}$, and selected such that $Tr(a') = 1$ and b be chosen as an element in $\mathbb{F}_q^*$ (i.e., $b \neq 0$).

Then a GLS curve can be defined as

$$\tilde{E}/\mathbb{F}_{q^2} : y^2 + xy = x^3 + a'x^2 + b. \tag{4}$$

The curve $\tilde{E}$ is the quadratic twist of E. In other words, the curves $\tilde{E}$ and E are isomorphic over $\mathbb{F}_{q^4}$ under the endomorphism [12],

$$\phi : E \to \tilde{E}, (x, y) \mapsto (x, y + sx),$$

with $s \in \mathbb{F}_{q^4} \backslash \mathbb{F}_{q^2}$ satisfying $s^2 + s = a + a'$. Let $\pi : E \to E$ be the Frobenius map defined as $(x, y) \mapsto (x^q, y^q)$, and let ψ be the composite endomorphism $\psi = \phi\pi\phi^{-1}$ given as,

$$\psi : \tilde{E} \to \tilde{E}, (x, y) \mapsto (x^q, y^q) + s^q x^q + sx^q.$$

By setting a' to u, there exists $b \in \mathbb{F}_q$, so that $\#\tilde{E}(\mathbb{F}_{q^2}) = hr$, with $h = 2$ and r is a prime number, whose bit length is about $2n-1$ bits. In this case, the endomorphism ψ acting over the affine point

$$P = (x_0 + x_1 u, y_0 + y_1 u) \in \tilde{E}(\mathbb{F}_{q^2}),$$

can be computed by using four additions in $\mathbb{F}_q$ as,

$$\psi(P) \mapsto ((x_0 + x_1) + x_1 u, (x_0 + y_0 + y_1) + (x_0 + x_1 + y_1)u). \tag{5}$$

2.3.2.1 Security of GLS Curves

As explained at the beginning of Section 2.3, the security of elliptic curve cryptography is based on the hardness of the elliptic curve discrete logarithm problem(ECDLP). There are two algorithms for solving the ECDLP: the Baby Step Giant Step and the Pollard Rho algorithms. The time complexity of these two algorithms are $O(\sqrt{q})$, hence exponential in n.

In [32], the most powerful specialized attack for solving the ECDLP on binary

elliptic curves, with an associated computational complexity of $O(2^{c \cdot n^{2/3} \log n})$, where $c < 2$ and n is a prime number, was presented (see also [14, 38, 9]). This complexity is higher than the generic algorithms for all prime field extensions $n < 2000$. This bound is much higher than the range for order of elliptic curves used in practical applications using elliptic curve cryptography [32].

In [9], a survey of recent progress in the computation of the ECDLP is provided by Galbraith and Gaudry. Some recent papers (see [37, 15, 18]) argue that the summation polynomial approach can lead to sub-exponential algorithms for the ECDLP in characteristic two. On the other hand, binary elliptic curve cryptography is still safe according to Galbraith and Gaudry. In [9], they state the following opinion with respect to the possibility of finding a sub-exponential time algorithm for ECDLP in characteristic 2,

> "*Finally, it must be emphasized that for the moment none of the [above] approaches are practical at all: even with the most optimistic assumptions, the running time and the memory usage would be extremely high for any key size currently in use. The difficulty to make practical experiments with non-tiny examples is an explanation why the assumptions are hard to (in-)validate*"

There may be a concern that some form of the Weil descent family of attacks [11, 13] can be efficiently applied on the GLS curves, which are defined over quadratic extension fields. On the other hand, in [24], it is shown that, the gGHS attack cannot be applied on elliptic curves defined over binary extension fields $\mathbb{F}_q$, with $q = 2^n$, and n being a prime number in $[160, \ldots, 600]$. In [12], it is proved that the only vulnerable prime extension in the range $[80, \ldots, 256]$, is $n = 127$ for GLS binary curves, which are defined over $\mathbb{F}_{q^2}$, with $q = 2^n$. Therefore, it is important to select other parameters to obtain a secure curve. Selecting the constant b of $E(\mathbb{F}_{q^2})$, which is not weak, is the recipe to prevent this attack. Firstly, the probability of $b \in \mathbb{F}_{q^2}^*$ being a weak constant is negligible when it is chosen randomly [12]. Moreover, a procedure, which analyzes that the vulnerability of a binary GLS elliptic curve,

which uses a concrete parameter $b \in \mathbb{F}_{q^2}^*$, to the gGHS attack, is presented in [4, Algorithm 1]. By using Algorithm 1 of [4], it is verified that the specific instance of the GLS curve, which used in this work (see Appendix), is not vulnerable to Weil descent attacks.

Algorithm 1 Left-to-right Montgomery Ladder [27]

Input: $P = (x, y), k = (1, k_{l-2}, \ldots, , k_1, k_0)$
Output: $Q = kP$
1: $R_0 \leftarrow P$; $R_1 \leftarrow 2P$;
2: **for** $i = l - 2$ `downto 0` **do**
3: **if** $k_i = 1$ **then**
4: $R_0 \leftarrow R_0 + R_1; R_1 \leftarrow 2R_1$
5: **else**
6: $R_1 \leftarrow R_0 + R_1; R_0 \leftarrow 2R_0$
7: **end if**
8: **end for**
9: **return** $Q = R_0$

2.3.3 Scalar Point Multiplication

Scalar point multiplication, which is the main cryptographic operation in ECC, can be defined as the calculation of a point Q by multiplying a point P with a scalar k, i.e. $Q = kP$ ($k < r$, where r is the order of the elliptic curve). In order to clarify the definition, let $\tilde{E}$ be the GLS elliptic curve of Eq. (4), which is defined over the field $\mathbb{F}_{q^2}$, with $q = 2^n$ and n a prime number. In addition, let us consider that $\#\tilde{E}(\mathbb{F}_{q^2}) = hr$, with $h = 2$ and where r is a $2n - 1$-bit prime number. In elliptic curve multiplication $Q = kP$, Q is a point which can be obtained by adding P to itself $k - 1$ times, with $k \in [0, r - 1]$. However, this method is not a feasible way to compute kP for a large k. Therefore, algorithms such as Montgomery ladder and Double-and-Add algorithms, are used to perform the scalar point multiplication operation efficiently.

In this work, the Montgomery ladder approach, which is described in Algorithm 1, is used. The average cost of performing a multiplication by using this approach is $\ell D + \ell A$, where ℓ is bit size of k and D and A are the costs of the point doubling and addition operations, respectively.

The main idea behind the Montgomery ladder algorithm can be explained as follows. Suppose two points R_0 and R_1 for a given base point P whose difference is exactly P, *i.e.*, $R_0 - R_1 = P$. Then, we can compute the x-coordinates of the elliptic curve points, $2R_0$, $2R_1$ and $R_0 + R_1$, by using P, R_0 and R_1. By taking the advantage of this feature, López and Dahabin [23] introduced a compact formulae for the point addition and point doubling operations in Steps 4 and 6 of Algorithm 1. The computational cost of each ladder step in Algorithm 1 by using the formulae given in [23] is of 5 multiplications, 1 multiplication by the curve b-constant, 5 squarings and 3 additions over the binary field where the elliptic curve is defined [29].

Moreover, thanks to the two-dimensional endomorphism ψ of the GLS curve $\tilde{E}$, there exists an integer $\delta \in [2, r-2]$, such that $\psi(P) = \delta P \in \langle P \rangle$. Consequently, the point multiplication can be performed by using the GLV approach as follows,

$$Q = kP = k_1 P + k_2 \cdot \delta P = k_1 P + k_2 \psi(P), \tag{6}$$

where the sub-scalars k_1 and k_2, whose sizes are approximately $\ell/2$, can be computed by solving a closest vector problem in a lattice [10]. The GLS curve endomorphism allows to exploit extra levels of concurrency in computation of the scalar multiplication as the operations $k_1 P$ and $k_2 \psi(P)$ can be assigned to different cores in hardware, thus computed in parallel.

2.4 FPGA

Application specific integrated circuits (ASICs) and Field Programmable Gate Arrays (FPGAs), which are also integrated circuits, are two common target platforms in the market for implementing hardware designs. ASICs are manufactured according to design specifications. Therefore, the design can be optimized in terms of area and critical path delay. As a result, they can have smaller form factors and very high clock frequencies. Moreover, the unit cost of ASICs are can be very low when

they are manufactured in very high volumes.

On the other hand, FPGAs, also known as reconfigurable hardware in more generic terms, have shorter and simpler design cycle thanks to automated computer-aided software tool chains. For instance, user can generate a bitstream file, which is used to program the FPGA, from an HDL code by using Xilinx Vivado Design Suite hiding the complexities of intermediate steps such as design entry, synthesis and optimization. Furthermore, they can be (dynamically and remotely) programmed many times without any cost. This feature allows users to update the their designs such as updating bug and adding some new features.

FPGAs consist of configurable logic blocks (CLBs), which contain look-up tables (LUTs), multiplexers, gates, and flip flops. LUTs are programmable units and used to implement any logical function. As they are volatile devices based on RAM technology FPGAs have to be programmed by loading a bitstream file after every power loss.

3 Architecture

In this chapter, we provide details of the architectural design of the base field arithmetic $\mathbb{F}_q$ with $q = 2^n$, the quadratic field arithmetic $\mathbb{F}_{q^2}$ and the elliptic curve arithmetic.

3.1 Arithmetic in Binary Extension Field $\mathbb{F}_q$

In binary extension field $\mathbb{F}_q$, we use three different arithmetic operations, which are squaring, addition and multiplication. The details of these operations are given below.

3.1.1 Squaring operation in $\mathbb{F}_q$

Squaring operation in finite field $\mathbb{F}_q$ consists of two parts: polynomial squaring and the reduction by the irreducible polynomial of $\mathbb{F}_q$. As the polynomials in $\mathbb{F}_q$ are binary, the squaring operation can be obtained by simple re-wiring in hardware. For instance, suppose that we will compute polynomial squaring of $a(x) = x^4+x^3+x+1$ in $\mathbb{F}_{2^5}$ with irreducible polynomial x^5+x^2+1. We simply have $a^2(x) = x^8+x^6+x^2+1$, which can be computed free of cost in hardware. Also, $a^2(x) = x^8 + x^6 + x^2 + 1$ can be reduced by the irreducible polynomial $x^5 + x^2 + 1$, which result in $a^2(x) = x^8 + x^6 + x^2 + 1 = x \bmod f(x)$. Thanks to the sparse irreducible polynomial (e.g.,

a trinomial) in the proposed design for $\mathbb{F}_{2^{127}}$ arithmetic, i.e. $f(x) = x^{127} + x^{63} + 1$, the reduction part of squaring operation can also be computed easily. Squaring operation for the parameters in the proposed design can be performed as defined in Algorithm 2.

Algorithm 2 Squaring operation in $\mathbb{F}_{2^{127}}$

Input: $a(x)$
Output: $b(x) = a^2(x) \bmod f(x)$

1: $c(x) = a^2(x) = \sum_{i=0}^{126} a_i x^{2i}$ ▷ $c(x)$ is a degree-252 polynomial to compute $a^2(x)$
2: **for** $i = 252$ `downto` 127 **do**
3: $\quad c_{i-127} = c_i \oplus c_{i-127}$
4: $\quad c_{i-64} = c_i \oplus c_{i-64}$
5: **end for**
6: **for** $i = 126$ `downto` 0 **do**
$\quad b_i = c_i$
7: **end for**
8: **return** $b(x)$

As can be seen in Algorithm 2, the reduction operation is performed in the first `for` loop. The squaring circuit is not too complex as a consequence of the chosen sparse irreducible polynomial, which is a trinomial in our work. It is implemented as a fully combinational circuit and its effect on the scalar multiplication circuit is negligible in terms of the critical path delay and area requirement with no cost in terms of clock cycle count.

3.1.2 Addition operation in $\mathbb{F}_q$

Addition in $\mathbb{F}_q$ is another operation that is used in elliptic curve arithmetic. As mentioned in Section 2.1.1, it consists of only XOR operations on the corresponding binary coefficients of two polynomials. The addition circuit is also implemented as fully combinational and its effect on the scalar multiplication circuit is negligible in terms of the critical path delay and area like the squaring circuit with no cost in terms of clock cycle count.

3.1.3 Multiplication operation in $\mathbb{F}_q$

For multiplication operation in $\mathbb{F}_q$, we use two different approach; digit-based and Karatsuba multipliers. These approaches are explained below.

3.1.3.1 Digit-Based Multipliers

For multiplication operations, we use a digit-based multiplication algorithm in our first approach. A digit-based multiplication algorithm performs the multiplication operation in more than one iteration. In each iteration, the multiplicand is multiplied by one digit of the multiplier and the result of this operation is added to the running sum from the previous iterations after shifting by digit size. The performance of multiplication varies depending on the digit size. For instance, the operation takes more clock cycles for a relatively small digit size. On the other hand, it requires smaller amount of area and has relatively shorter critical path delay. In other words, the number of clock cycles is inversely proportional with area requirement and critical path delay.

There are different digit-based multiplication algorithms such as different variants of most-significant element first (MSE) algorithms and least-significant element first (LSE) algorithms. These algorithms have been explained in [3] in detail. Furthermore in [3], different multiplication algorithms and architectures for $\mathbb{F}_q$ have been compared in terms of various performance metrics such as area, delay, multiplication time, throughput and throughput/slice for the target device Spartan-3 XC3S1500 using ISE WebPACK 8.2.03i. According to the comparison results of different architectures, Algorithm 3 in [3] is the best choice for our target device and the chosen finite field, i.e. $\mathbb{F}_{2^{127}}$.

The description of the selected algorithm is given in Algorithm 3 and the corresponding hardware architecture is shown in Figure 1. In each iteration of this multiplication algorithm, one digit of $a(x)$ (i.e., $a_{Di+(D-1)}x^{D-1} + a_{Di+(D-2)}x^{D-2} + \ldots + a_{Di+1}x^1 + a_{Di}x^0$, where D is digit size, is multiplied by $b(x)$. The result of this

Algorithm 3 MSE multiplication over $\mathbb{F}_q$ [3].

Require: A degree-n monic polynomial $f(x) = x^n + f_{n-1}x^{n-1} + ... + f_1x + f_0$ and two degree-$(n-1)$ polynomials $a(x)$ and $b(x)$, where $a_{-j} = 0$, $1 \leqslant j \leqslant D$.

Ensure: $p(x) = a(x)b(x) \bmod f(x)$

1: $s(x) \leftarrow 0$;
2: **for** i `from` $\lceil n/D \rceil$ `- 1 downto -1` **do**
3: $\quad t(x) \leftarrow \sum_{j=0}^{D-1} a_{Di+j}x^j b(x)$;
4: $\quad s(x) \leftarrow t(x) + x^D \cdot (s(x) \bmod f(x))$;
5: **end for**
6: $p(x) \leftarrow s(x)/x^D$;

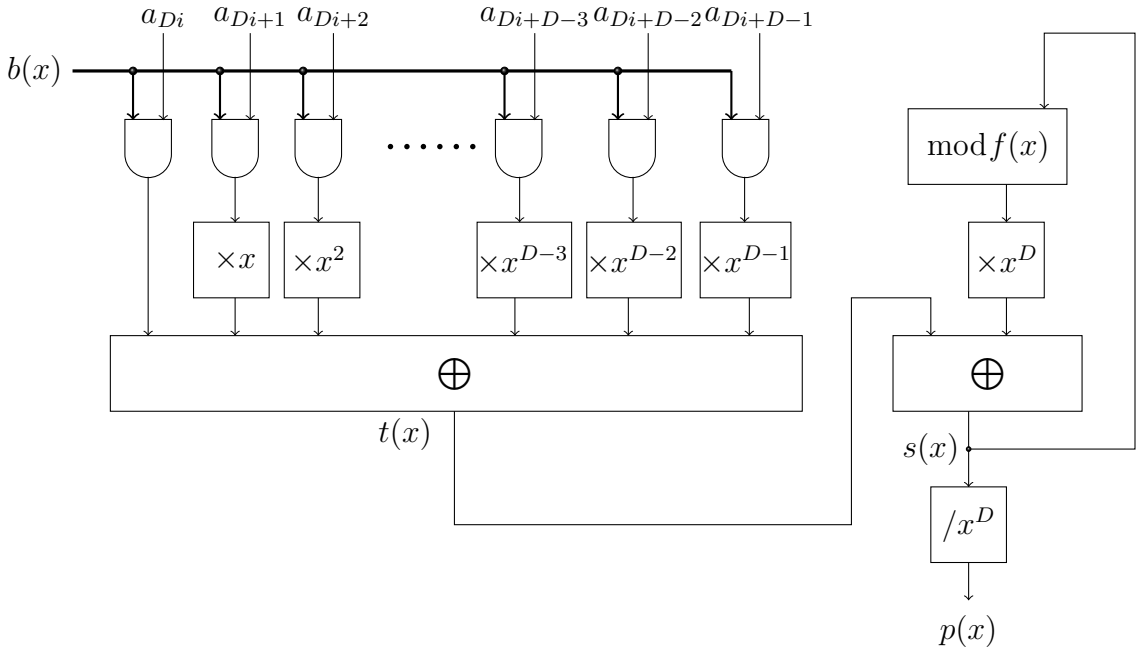

Figure 1: Digit-Based Multiplier Architecture for $\mathbb{F}_q$

multiplication is the polynomial $t(x)$, whose degree is $n + D - 2$. In addition, the polynomial $s(x)$, which is of degree $n + D - 1$, keeps the running sum of previous iterations. After $s(x)$ is reduced by the irreducible polynomial, it is shifted by digit size and the result is added to $t(x)$. The reduction operation (i.e. $\bmod f(x)$ in the algorithm) is implemented in a low complexity circuit as $f(x) = x^{127} + x^{63} + 1$ is trinomial. Consequently the reduction operation is designed and implemented as a fully combinational circuit.

Consequently, one multiplication in $\mathbb{F}_{2^n}$ takes $\lceil \frac{n}{D} \rceil + 2$ clock cycles in our design. More precisely, Algorithm 3 needs $\lceil \frac{n}{D} \rceil + 1$ clock cycles to finish the multiplica-

tion operations and one additional clock cycle is needed between two consecutive multiplication operations to minimize the propagation delay in the circuit.

In order to implement the best design, selecting the optimal digit size D, which determines the clock count of the $\mathbb{F}_q$ multiplier, is crucial for the critical path delay and design area. For example, the clock count of a $\mathbb{F}_q$ multiplier with $D = 16$ is approximately 10% lower than another $\mathbb{F}_q$ multiplier with $D = 15$. On the other hand, the area and critical path delay of $\mathbb{F}_q$ multiplier with $D = 16$ is higher than $\mathbb{F}_q$ multiplier with $D = 15$. In this work, we made some experiments for D values to find the optimum digit size. The results are reported in Chapter 4.

3.1.3.2 Karatsuba Multipliers

The Karatsuba algorithm is a fast multiplication algorithm, which is originally proposed to perform the multiplication operation for two large numbers efficiently. It can also be used for polynomial multiplications as explained below.

Let $a(x)$ and $b(x) \in \mathbb{F}_q$, where $q = 2^n$. We can also express $a(x)$ and $b(x)$ as follows

$$a(x) = a_1 x^{n/2} + a_0$$

$$b(x) = b_1 x^{n/2} + b_0.$$

In this expression, a_i and b_i for $i \in \{0, 1\}$ are polynomials of degree $n/2 - 1$. To simplify the following discussion, n is always assumed to be an even number. In case that n is an odd number, n is set to $n + 1$ and the remaining operations are performed in the same manner.

In general, we can define the product of $a(x)$ and $b(x)$ as follows

$$a(x)b(x) = a_1 b_1 x^n + (a_1 b_0 + a_0 b_1)x^{n/2} + a_0 b_0. \tag{7}$$

As can be seen, all possible products of a_i and b_j for $i, j \in \{0, 1\}$ are computed (i.e., four multiplications of half-degree polynomials). By using the Karatsuba algorithm, we may express $(a_1 b_0 + a_0 b_1)$ term as $[(a_1 + a_0)(b_1 + b_0) + a_1 b_1 + a_0 b_0]$ as we work in

a field of characteristic 2. Then the polynomial multiplication can be performed as

$$a(x)b(x) = a_1b_1x^n + [(a_1 + a_0)(b_1 + b_0) + a_1b_1 + a_0b_0]x^{n/2} + a_0b_0.$$

Note that, $-$ and $+$ operations are same in a binary field. In this method, the operation can be performed by using only three half-sized polynomial multiplications than the schoolbook multiplication algorithm given in Eq. (7). It is true that several extra polynomial addition operations are needed in the Karatsuba method. However, polynomial additions are much simpler than the polynomial multiplications. Thanks to the Karatsuba algorithm, the overall complexity of the $\mathbb{F}_q$ multiplication circuit decreases dramatically.

As be explained above, we can implement the polynomial multiplication circuit by using three degree-$n/2$ multiplication operations, instead of one degree-n multiplication operation. In addition, we can implement the Karatsuba algorithm recursively. For instance, each of the multiplications $(a_1 + a_0)(b_1 + b_0)$, a_1b_1 and a_0b_0, whose results are needed to compute $a(x) \cdot b(x)$, can further be simplified by a second application of the Karatsuba method. In this case, we only need to perform seven less quarter-sized polynomial multiplications than the schoolbook algorithm which needs 16.

The recursion can be applied repetitively until the inputs are 2 bits. However, the recursion loses its advantage after a certain level. At that point, classical multiplication is used instead of Karatsuba. The recursion level is determined by experimentation on the target device, which is FPGA in this work.

In our design, we employ one half-sized polynomial multiplier, which performs corresponding operations in consecutive clock cycles as illustrated in Figure 2. The half-sized polynomial multiplier takes two degree-63 polynomials and generates their product as a degree-126 polynomial in one clock cycle. In other words, it is a combinational multiplier, which takes two 64-bits inputs and gives a 127-bit output.

In the first clock cycle, the operands, a_0 and b_0 are written in registers T_1 and T_2, which are illustrated in Figure 2. The product of $a_0 \cdot b_0$ is written to T_3 in the

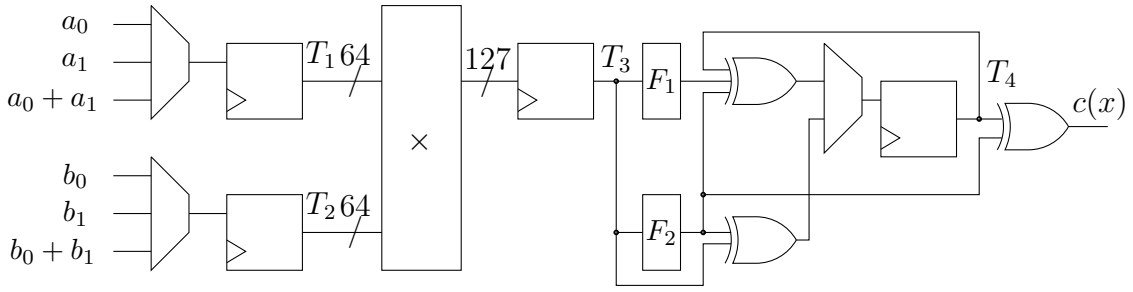

Figure 2: Karatsuba Based Multiplier

second clock cycle. T_1 and T_2 store a_1 and b_1 in the same clock cycle. The two blocks, F_1 and F_2, perform the following operations, respectively

$$F_1 = T_3 x^{128} \bmod f(x)$$
$$F_2 = T_3 x^{64} \bmod f(x).$$

F_1 and F_2, which perform shifts and reductions with the irreducible polynomial $f(x)$ operations, are also implemented as fully combinational. In the third clock cycle of the computations, the following value is written to the register T_4 in Figure 2

$$T_4 \leftarrow a_0 b_0 + (a_0 b_0 x^{64} \bmod f(x)).$$

The operation $a_1 \cdot b_1$ is performed and stored in the register T_3, concurrently. In the fourth clock cycle, the following value is obtained and written in the rightmost register

$$T_4 \leftarrow T_4 + (a_1 b_1 x^{64} \bmod f(x)) + (a_1 b_1 x^{128} \bmod f(x)) =$$
$$a_0 b_0 + (a_0 b_0 x^{64} \bmod f(x)) + (a_1 b_1 x^{64} \bmod f(x)) + (a_1 b_1 x^{128} \bmod f(x)).$$

In the same clock cycle, the multiplication of $(a_1 + a_0)$ and $(b_1 + b_0)$ is completed and saved in the register T_3. Finally, in the fifth clock cycle, the final result is obtained at the output of the circuit

$$c(x) = T_4 + ((a_1 + a_0)(b_1 + b_0) x^{64} \bmod f(x))$$
$$= a(x) b(x) \bmod f(x).$$

As can be understood from the explanation, the latency of the $\mathbb{F}_{2^{127}}$ multiplier circuit is 5 clock cycles and the throughput is 4 clock cycles per multiplication .

3.2 Arithmetic in Quadratic Extension Field $\mathbb{F}_{q^2}$

In this section, we explain the arithmetic operations in the quadratic extension field $\mathbb{F}_{q^2}$ and the hardware architectures to perform them.

3.2.1 Squaring in Quadratic Extension Field $\mathbb{F}_{q^2}$

Let $a(u)$ and $b(u)$ be two elements in $\mathbb{F}_{q^2}$ with irreducible polynomial $g(u) = u^2 + u + 1$ and $b(u)$ be the square of $a(u)$. We can compute the square of $a(u)$, namely $a(u)^2 \bmod g(u)$, as follows

$$\begin{aligned} b(u) = a(u)^2 &= (a_1 u + a_0)^2 \bmod g(u) \\ &= a_1^2 u^2 + 2a_1 a_0 u + a_0^2 \bmod g(u) \\ &= a_1^2 u^2 + a_0^2 \bmod g(u) \\ &= a_1^2 u + (a_1^2 + a_0^2). \end{aligned}$$

The squaring operation in $\mathbb{F}_{q^2}$ can be implemented by using two squaring operations, in order to calculate a_1^2 and a_0^2, and one addition operation, for $a_1^2 + a_0^2$, in $\mathbb{F}_q$. The block diagram for operation flow is illustrated in Figure 3. ($S_{\mathbb{F}_q}$ represents squaring operation and $+_{\mathbb{F}_q}$ represents addition operation in $\mathbb{F}_q$). As mentioned in Section 3.1.1 and 3.1.2, the complexity of squaring and addition operations in $\mathbb{F}_q$ are extremely low in terms of critical path delay and area requirement. Therefore, the squaring in $\mathbb{F}_{q^2}$ be implemented as fully combinational. Consequently, the effect of squaring operation in $\mathbb{F}_{q^2}$ on the critical path of the complete design becomes limited and no extra clock cycle is needed to preform this operation.

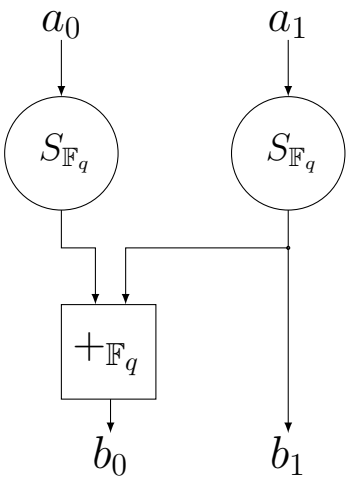

Figure 3: Squaring operation in $\mathbb{F}_{q^2}$

3.2.2 Multiplication in Quadratic Extension Field $\mathbb{F}_{q^2}$

The multiplication operation in $\mathbb{F}_{q^2}$ is more expensive than the squaring operation. Let $a(u)$, $b(u)$ and $c(u)$ be three polynomials of degree one over $\mathbb{F}_q^2$ and the multiplication can be performed with the irreducible polynomial $g(u) = u^2 + u + 1$ as follows

$$\begin{aligned} c(u) = a(u)b(u) \bmod g(u) &= (a_1u + a_0)(b_1u + b_0) \bmod g(u) \\ &= a_1b_1u^2 + (a_1b_0 + a_0b_1)u + a_0b_0 \bmod g(u) \\ &= (a_1b_1 + a_1b_0 + a_0b_1)u + a_0b_0 + a_1b_1. \end{aligned}$$

As shown above, we need to perform four multiplication operations (i.e. $a_1 \cdot b_0$, $a_0 \cdot b_1$, $a_0 \cdot b_0$ and $a_1 \cdot b_1$) in $\mathbb{F}_q$ to calculate $a(u) \cdot b(u)$. On the other hand, $a_1b_0 + a_0b_1$ can be computed by using Karatsuba technique as

$$a_1b_0 + a_0b_1 = (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0.$$

Therefore the following formula can be profitably used to compute multiplication in $\mathbb{F}_q$

$$\begin{aligned} c(u) = a(u)b(u) &= (a_1b_1 + a_1b_0 + a_0b_1)u + a_0b_0 + a_1b_1 \\ &= [a_1b_1 + (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0]u + a_0b_0 + a_1b_1 \\ &= [(a_1 + a_0)(b_1 + b_0) + a_0b_0]u + a_0b_0 + a_1b_1. \end{aligned}$$

Note that, $-a_0b_0$ is equal to $+a_0b_0$ in a field with characteristic 2. As shown above, three multiplication and four addition operations in $\mathbb{F}_q$ is used to implement a $\mathbb{F}_{q^2}$ multiplier. The architecture of the proposed $\mathbb{F}_{q^2}$ multiplier is illustrated in Figure 4.

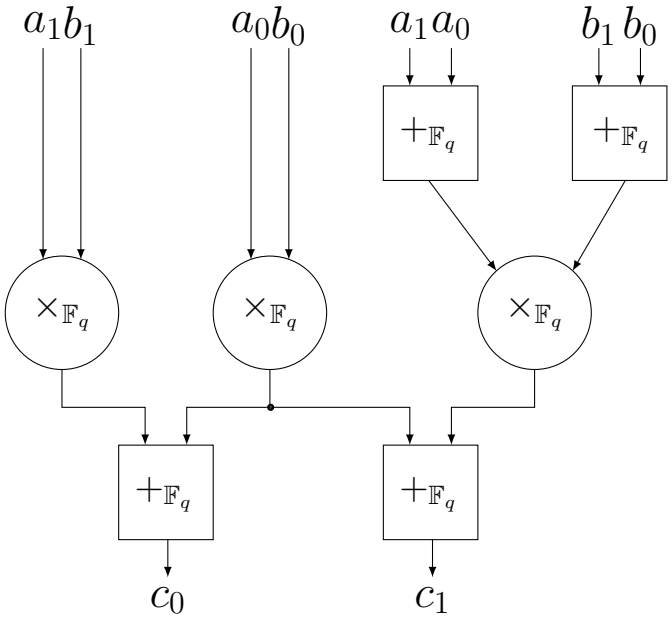

Figure 4: Multiplication operation in $\mathbb{F}_{q^2}$

3.3 Architectures for Elliptic Curve Arithmetic

In order to perform elliptic curve point multiplication, we employ the left-to-right Montgomery ladder algorithm in our design. The Montgomery ladder algorithm, which is shown as Algorithm 1, is a constant time algorithm. The algorithm complete the calculation after ℓ iteration, where ℓ is $\lceil \log_2(r) \rceil$ and r presents the order of elliptic curve.

In each iteration, the algorithm performs one point addition and one point doubling operations. The operations do not use the result of others. In other words, they are not dependent on each other. This property of the algorithm allows to perform the point addition and point doubling operations concurrently. In the proposed design, we use two separate modules for point addition and point doubling operations by using the concurrency property of the Montgomery ladder algorithm.

Let R_0, R_1 and R_2 be three points on the elliptic curve. For point addition operation, we use the following formulae

$$Z_2 = (X_0 Z_1 + X_1 Z_0)^2$$
$$X_2 = xZ_2 + (X_0 Z_1)(X_1 Z_0).$$

which are introduced by López and Dahab in [23]. As can be seen from the equation above, the point addition operation requires four multiplication operations in $\mathbb{F}_{q^2}$, namely X_0Z_1, X_1Z_0, xZ_2 and $(X_0Z_1)(X_1Z_0)$. However, as the computation of X_2 requires Z_2, Z_2 must be computed before the computation of X_2 starts. In this case, not all of the four multiplication operations can be performed in parallel.

In the proposed design, we use two $\mathbb{F}_{q^2}$ multipliers. One of them computes the multiplications X_0Z_1 and $(X_0Z_1) \cdot (X_1Z_0)$ while the other performs X_1Z_0 and xZ_2. In this case, let t represent the clock cycles count, which is required to perform one multiplication in $\mathbb{F}_{q^2}$. Then a point addition operation takes $2t$ clock cycles considering the units for addition and squaring in $\mathbb{F}_{q^2}$ are fully combinational circuits. The architecture of point addition operation is shown in Figure 5. The upper side of dotted line illustrates the first t clock cycles while the down side illustrates the second t clock cycles of the point addition operation.

For point doubling, we use the formulae

$$X_2 = {X_0}^4 + b{Z_0}^4$$
$$Z_2 = {X_0}^2 {Z_0}^2$$

which are taken from [23]. The point doubling operation needs two multiplication

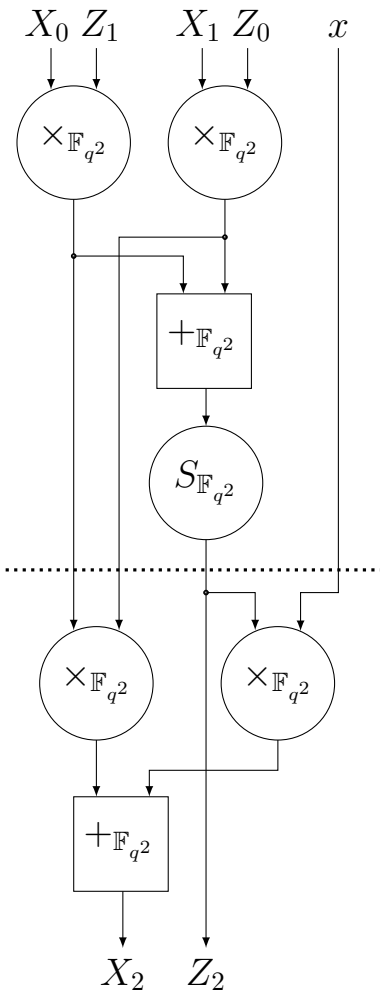

Figure 5: Point Addition in $\mathbb{F}_{q^2}$

operations, namely $b \cdot {Z_0}^4$ and ${X_0}^2 \cdot {Z_0}^2$, in $\mathbb{F}_{q^2}$. Actually, the point doubling operation can be performed in t clock cycles by using two $\mathbb{F}_{q^2}$ multipliers concurrently. However, the point addition operation takes $2t$ clock cycles. The point doubling module must wait for t clock cycles to keep concurrency, if it completes its operations in t clock cycles. In other words, the point doubling module can start a new calculation after each $2t$ clock cycles. Therefore, we use only one $\mathbb{F}_{q^2}$ multiplier instead of two in order to reduce the area of point doubling module. The architecture of point doubling operation is illustrated in Figure 6.

As mentioned in Section 3.2, the $\mathbb{F}_{q^2}$ multipliers are more dominant than addition and squaring modules in terms of area requirement and critical path delay. The addition and squaring modules are implemented as fully combinational whereas the multiplier can operates sequentially. Consequently, the effect of the adder and

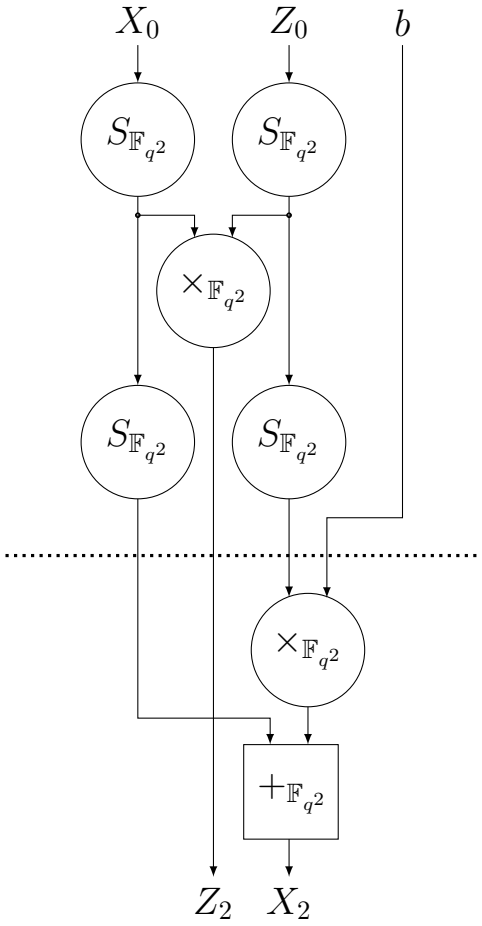

Figure 6: Point Doubling in $\mathbb{F}_{q^2}$

squaring units on the performance of scalar multiplication module is negligible. Therefore, the point addition and the point doubling modules take $2t$ clock cycles, where $\mathbb{F}_{q^2}$ multiplier takes t clock cycles. As the Montgomery ladder algorithm iterates ℓ times, one elliptic curve point multiplication is completed in $(2 \cdot t \cdot \ell)$ clock cycles

4 Implementation Details and Results

In this chapter, we provide our implementation details and results. In addition, we report the results of other similar implementations in the literature and compare them with our design.

4.1 Hardware

In this work, we used XILINX KC705 Evaluation Kit to test our implementation on real hardware. The evaluation kit contains a XILINX KINTEX-7 XC7K325T-2FFG900C FPGA, featuring 50950 slices, which are the basic building blocks of FPGA. In each slice, there are four 6-input LUTs (lookup tables), each of which implements any combinational function of six variables, and eight flip-flops as storage units. These resources suffice to implement our design. We synthesized our design by using XILINX Vivado Design Suite 2014.4, which gives the best synthesis result comparing with the other versions of XILINX Vivado Design Suite.

4.2 $\mathbb{F}_{q^2}$ Multiplier

As explained in Section 3.2.2, $\mathbb{F}_{q^2}$ multiplier consists of three $\mathbb{F}_q$ multipliers and combinational circuits to perform four $\mathbb{F}_{q^2}$ additions. We adopt two different approaches, namely digit-based and the Karatsuba-based algorithms as explained previously, to implement $\mathbb{F}_q$ Multipliers, where $q = 2^n$ and $n = 127$.

For digit-based multiplier, we experimented with four different digit sizes, namely $D \in \{22, 26, 32, 43\}$. These are the minimum digit sizes to complete one multiplication in $\mathbb{F}_q$ in 8, 7, 6 and 5 clock cycles, respectively.

For the Karatsuba-based multiplier, the first three levels of recursion are implemented. In 1-level Karatsuba, only one fully combinational 64-bit multiplier is used. The recursion is executed for twice and stops as 32 bit for 2–level Karatsuba. In this design, 3 fully combinational 32–bit multipliers work consequently. Similarly, 3-level Karatsuba uses nine fully combinational 16–bit multipliers. All Karatsuba multipliers completes multiplication operations in four clock cycles. The implementation results are provided in Table 1.

Design	**LUTs**	**Max Freq (MHz)**	**clock cycles**	**Delay (μS)**
22–bit Digit–Based	4745	478,01	8	0,017
26–bit Digit–Based	5342	434,22	7	0,016
32–bit Digit–Based	6514	448,63	6	0,013
43–bit Digit–Based	9237	397,93	5	0,013
1–level Karatsuba	6118	456,00	4	0,009
2–level Karatsuba	5753	384,17	4	0,010
3–level Karatsuba	9579	356,25	4	0,011

Table 1: $\mathbb{F}_{q^2}$ multiplier implementation results

In hardware implementations, we can generally expect that the area and the frequency of a circuit are inversely proportional. In our experiments, this expectation comes true for digit-based multipliers. The 22-bit digit–based multiplier requires the minimum area and is able to work with the maximum clock frequency. In contrast, the 43–bit digit–based multiplier requires the maximum area and can work with the minimum frequency.

On the other hand, a higher level Karatsuba-based multiplier is expected to require smaller area than a lower level of Karatsuba multiplier. For instance, a 3–level Karatsuba multiplier is expected to occupy less area than a 2–level Karatsuba multiplier. However, in our implementations the 3–level Karatsuba-based multiplier turns out to require higher amount of area resources than the 2–level Karatsuba multiplier. The first explanation for this that it can be attributed to amount and types of optimization applied by the software design tools. XOR operations can be easier than AND operation for optimization tools. A classical fully combinational polynomial multiplier can be imagined as a row of AND operations, followed by an XOR tree, which is illustrated for a 5–bit multiplier in Figure 7. Note that, the operation is performed for $a(x) \cdot b(x)$, where $a(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$ and $b(x) = b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$. The 1–level Karatsuba Multiplier employs one 64–bit combinational multiplier. In this case, the depth of a 64–bit classical multiplier is higher than the depth of a 32–bit classical multiplier. Similarly, a 32–bit multiplier has more depth than a 16–bit multiplier. As can be inferred, 3–level Karatsuba multiplier does not have enough depth for Vivado to optimise the area and critical path delay. As a result, we receive the best implemetation results for 2–level Karatsuba approach.

Secondly, that 3–level Karatsuba-based multiplier is worse than the 1–level and 2–level multipliers can be attributed to the additional routing delay inside the FPGA. As in the first reason, Vivado tool is not capable to optimise the 3–level karatsuba multiplier. Therefore, the 3–level karatsuba multiplier occupies more LUTs, which results in higher routing delays and hence higher critical path delay.

4.3 Point Addition and Point Doubling

In our design the higher level architecture of Point Addition and Point Doubling modules, explained in Section 3.3, are the same for different types of $\mathbb{F}_{q^2}$ multipliers explained in the preceding section. The implementation results Point Addition and Point Doubling modules are provided in Table 2 and Table 3, respectively.

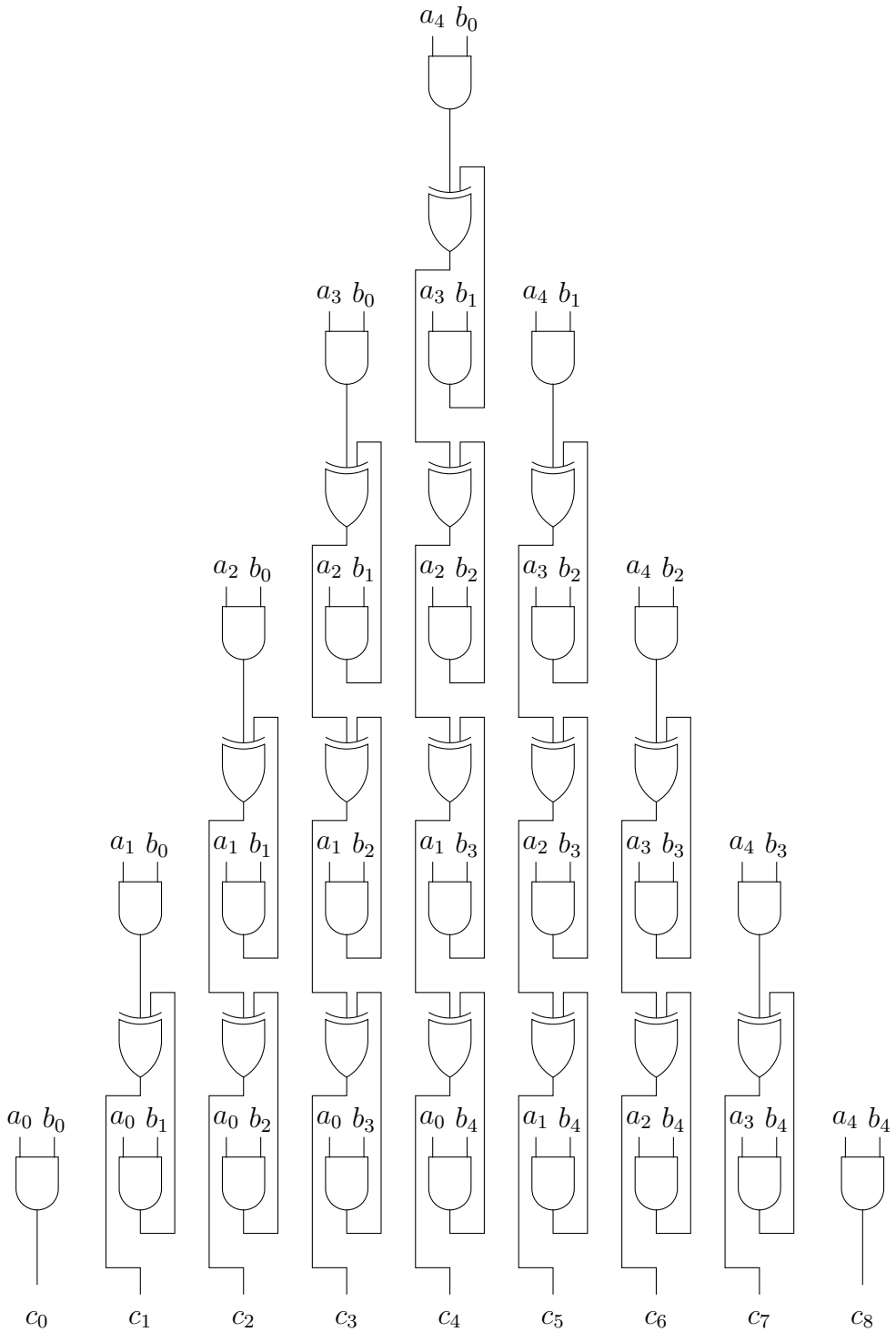

Figure 7: Tree structure of a 5–bit Fully Combinational Multiplier

Base Multiplier Type	**LUTs**	**Max Freq (MHz)**	**clock cycles**	**Delay (μS)**
22–bit Digit–Serial	12648	293,341	16	0,055
26–bit Digit–Serial	13731	286,779	14	0,049
32–bit Digit–Serial	16023	268,889	12	0,045
43–bit Digit–Serial	21234	254,259	10	0,039
1–level Karatsuba	15459	327,869	8	0,024
2–level Karatsuba	14706	322,997	8	0,025
3–level Karatsuba	22003	298,954	8	0,027

Table 2: Point Addition implementation results

Base Multiplier Type	**LUTs**	**Max Freq (MHz)**	**clock cycles**	**Delay (μS)**
22–bit Digit–Serial	7357	309,693	16	0,052
26–bit Digit–Serial	8447	279,720	14	0,050
32–bit Digit–Serial	9266	292,056	12	0,041
43–bit Digit–Serial	12062	275,862	10	0,036
1–level Karatsuba	8158	436,872	8	0,018
2–level Karatsuba	7827	381,971	8	0,021
3–level Karatsuba	11548	350,754	8	0,023

Table 3: Point Doubling implementation results

As can be anticipated (compare the architectures in Figure 5 and Figure 6), point addition circuit requires more area, which is about the double of the area of the point doubling circuit. $\mathbb{F}_{q^2}$ multiplier dominates the circuit in terms of area as mentioned in Chapter 3 and point addition and doubling circuits employ two $\mathbb{F}_{q^2}$ multipliers and one $\mathbb{F}_{q^2}$ multiplier, respectively.

Although they consist of one $\mathbb{F}_{q^2}$ multiplier and some combinational elements in their critical path, the point addition module is more complicated than the point doubling unit, therefore results in a slightly higher critical path delay. While this is expected and intuitive, the difference between two units for 1–level Karatsuba multiplier is much higher than those for 2 and 3–level Karatsuba multipliers. There are two 64-bit multipliers in the point addition module while there is only one in the point doubling module. It turns out that the optimization tool cannot handle two large 64-bit multipliers, while optimizing a single one is easier. The optimization process is more effective for smaller multiplier modules in higher level Karatsuba multipliers.

4.4 Scalar Multiplication

We report the result of scalar point multiplication circuits, which are built by using different $\mathbb{F}_{q^2}$ Multipliers, in Table 4. Note that, the results are reported for 127-bit scalar k, as explained in Algorithm 1.

Base Multiplier Type	**LUTs**	**Max Freq (MHz)**	**clock cycles**	**Delay (μS)**
22–bit Digit–Based	23074	197,550	2017	10,21
26–bit Digit–Based	25002	199,422	1765	8,85
32–bit Digit–Based	28422	187,793	1513	8,05
43–bit Digit–Based	36411	170,765	1261	7,38
1–level Karatsuba	27517	241,429	1009	4,17
2–level Karatsuba	25777	253,229	1009	3,98
3–level Karatsuba	37007	233,100	1009	4,32

Table 4: Scalar Point Multiplication Implementation results

As mentioned in Section 3.3, two independent modules, point addition and point doubling, are working concurrently. Therefore, the area of scalar multiplication module is supposed to be about the total area of point addition and point doubling circuits. Similarly, the critical path delay of scalar multiplier circuit is expected to be close to the delays of point addition and point doubling modules. However, scalar multiplication circuits require $1000 - 4000$ additional LUTs for all different base multiplier types. Moreover, the maximum frequencies of scalar point multiplication modules are $60 - 80$ MHz lower than point addition and point doubling. For example, point addition and point doubling modules requires 14706 and 7827 LUTs for 2–level Karatsuba, respectively. Thus, the area requirement of scalar point multiplication should be about $22500 - 23000$ LUTs. However, scalar point multiplier requires 25777 LUTs, which is 10% higher than expected. In addition, the maximum frequency values for point addition and point doubling modules are 323 MHz and 382 MHz, respectively. Therefore, the maximum frequency of scalar point multiplier is expected to be about 323 MHz. However, the maximum frequency of scalar multiplication is 253 MHz, which is about %20 lower than expected. Additional routing delays and other unexpected factors of FPGA design cause these dramatic

differences.

In order to perform a scalar point multiplication with 254-bit scalar k, we use two different approaches, namely 1–Core Design and 2–Cores Design by taking the advantage of the endomorphism associated with the GLS curves.

4.4.1 1–Core design

In 1–Core design, we use one scalar multiplication module and do not use the endomorphism associated to the GLS curves. Therefore, the Montgomery ladder algorithm iterates 253 times to compute $Q = kP$. As a result, the area requirement of scalar point multiplication is the same as the figures reported in Table 4. On the other hand, since the delays in Table 4 are given for a 127-bit scalar integer, the delay in 1–Core design is twice those listed in Table 4. For instance, 26–bit digit–based multiplier architecture and 2–level Karatsuba based multiplier architectures require 25,002 and 25,777 LUTs and execute in 3,530 and 2,018 clock cycles to finish one scalar multiplication, respectively. The implementation results of 1–Core design for different types of multipliers are reported in Table 5.

Base Multiplier Type	**LUTs**	**Max Freq (MHz)**	**clock cycles**	**Delay (μS)**
22–bit Digit–Based	23074	197,550	4034	20,42
26–bit Digit–Based	25002	199,422	3530	17,70
32–bit Digit–Based	28422	187,793	3026	16,10
43–bit Digit–Based	36411	170,765	2522	14,76
1–level Karatsuba	27517	241,429	2018	8,34
2–level Karatsuba	25777	253,229	2018	7,96
3–level Karatsuba	37007	233,100	2018	8,64

Table 5: 1-Core design implementation results

4.4.2 2–Core design

Thanks to the endomorphism associated with the GLS curves, two scalar point multiplication module can run in parallel and the computation of $Q = kP$ takes shorter amount of time than 1-core design. As explained in Section 2, we can

perform a scalar point multiplication by taking advantage of the endomorphism as follows

$$\begin{aligned} Q &= kP \\ &= k_1 P + k_2 \cdot \delta P \\ &= k_1 P + k_2 \psi(P), \end{aligned} \tag{8}$$

where the bit lengths of scalars k_1 and k_2 are approximately 126–bits. The endomorphism $\psi(P)$ can be computed at a negligible cost as we have

$$\psi(P) \leftarrow ((x_0 + x_1) + x_1 u, (x_0 + y_0 + y_1) + (x_0 + x_1 + y_1)u).$$

Also, decomposition of the scalar k into the two sub-scalars k_1 and k_2 is possible by solving a closest vector problem in a lattice [11]. However, in our implementation we use two randomly chosen sub-scalarsk_1 and k_2 and report implementation results accordingly [22].

As a result, two identical scalar point multiplication modules are employed in the proposed design to compute $k_1 P$ and $k_2 \psi(P)$ in parallel. Each module executes 126 iterations of Algorithm 1. Two points are returned by the two modules as a result of the computations $k_1 P$ and $k_2 \psi(P)$. Finally, they are added to compute the final result, Q.

In this scenario, two scalar point multiplication modules work concurrently and additional operations are performed in combinational circuits. Therefore, the computation of Q requires the same number of clock cycles as one scalar point multiplication module with 126–bit scalar, which are reported in Table 4 for different multipliers. On the other hand, the area requirement increases by a factor of two. For instance, 26–bit digit–based and 2–level Karatsuba based multiplier architectures require 50,004 and 51,554 LUTs and execute in 1,765 and 1,009 clock cycles to finish one scalar multiplication, respectively. The implementation results of 2–Cores design for different base field multipliers are reported in Table 6.

Base Multiplier Type	LUTs	Max Freq (MHz)	clock cycles	Delay (μS)
22–bit Digit–Based	46148	197,550	2017	10,21
26–bit Digit–Based	50004	199,422	1765	8,85
32–bit Digit–Based	56844	187,793	1513	8,05
43–bit Digit–Based	72822	170,765	1261	7,38
1–level Karatsuba	55034	241,429	1009	4,17
2–level Karatsuba	51554	253,229	1009	3,98
3–level Karatsuba	74014	233,100	1009	4,32

Table 6: 2-Core design Implementation results

4.5 Comparison

In the literature, there are a number of works, which report different hardware accelerators. Making a fair comparison of the proposed architectures and those in the literature is not straightforward. The first reason is the variety of target devices. In the literature, hardware accelerators are implemented on different target devices; i.e., various FPGA devices and ASIC technology, which have completely different types and amount of resources. Moreover, two FPGA devices, which are produced by different companies (e.g., Xilinx vs. Altera) or members of different device families (e.g., Virtex vs. Kintex Families by Xilinx), can be manufactured by using different semiconductor technologies or different architectures. Consequently, same design can have different critical path delays or result in different resource usage for different target FPGA devices.

The second reason is that the designs in literature are implemented for different security levels or different elliptic curves. For example, two scalar point multiplication circuits, which have different security levels, will have different critical path delays and area requirements for the same target device.

As a result, making a comparison is difficult, but not impossible. For instance, if two circuits, which have different security levels, require similar amount of areas and have similar critical path delays on the same target device, a realistic comparison can be made. For another example, two circuits, which have same security level on the same target device, can be compared in terms of area requirement, critical path

delay or any other metric. (e.g. critical path delay $\times$ area requirement)

As mentioned at the beginning of this section, there are a number of works, which report different hardware accelerators. in Table 7, we report some of them, which are similar with our work in terms of the security level, target device or latency.

Design	**Field**	**Platform**	**Area (# of LUTs)**	**Freq (MHz)**	**clock cycles**	**Latency (μS)**	**Area-Time per Bit**
[19]	$\mathbb{F}_{2^{163}}$	Virtex 7	27,105	223	780	3.49	580
[35]	$\mathbb{F}_{2^{283}}$	ASIC 130nm	10,204 (GE)	16	1,566,000	97,875	–
[16]	$\mathbb{F}_{2^{163}}$	Stratix II	16,766	185	2167	11.71	1,204
[17]*	$\mathbb{F}_{2^{163}}$	Stratix II	26,148	188	921	4.91	788
[17]*	$\mathbb{F}_{2^{233}}$	Stratix II	38,056	181	1,465	8.09	1,321
[17]*	$\mathbb{F}_{2^{283}}$	Stratix II	39,862	180	2,170	12.08	1,702
[33]	$\mathbb{F}_{2^{233}}$	Virtex 5	18,097	156	1,919	12.3	955
[21]	$\mathbb{F}_{2^{283}}$	ASIC 65nm	19,058 (GE)	–	512,555	–	–
[1]	$\mathbb{F}_{2^{233}}$	Virtex 4	28,145	205	6,789	33.1	3,998
[41]	$\mathbb{F}_{p_{256}}$	ASIC 130nm	208,000 (GE)	214	45,154	211	–
[40]	$\mathbb{F}_{p_{255}}$	ASIC 90nm	33,000 (GE)	48	1,110,000	23,125	–
[29]	$\mathbb{F}_{2^{254}}$	Haswell 1-core	–	3.0GHz	60,000	20	–
[30]	$\mathbb{F}_{2^{254}}$	Haswell 2-cores	–	2.4GHz	52,000	21.66	–
[30]	$\mathbb{F}_{2^{254}}$	Haswell 4-cores	–	2.4GHz	34,800	14.50	–
Our work DS 1–Core	$\mathbb{F}_{2^{254}}$	Kintex 7	25,002	199 MHz	3,530	17.70	1,742
Our work DS 2–Core	$\mathbb{F}_{2^{254}}$	Kintex 7	50,004	199 MHz	1,765	≈ 8.85	1,742
Our work K 1–Core	$\mathbb{F}_{2^{254}}$	Kintex 7	25,777	253 MHz	2,018	7.97	808
Our work K 2–Core	$\mathbb{F}_{2^{254}}$	Kintex 7	51,554	253 MHz	1,009	≈ 3.98	808

* This work did not implement the scalar τ-NAF conversion

Table 7: Comparative table

Generally speaking, latency and area requirement are two inversely correlated parameters in FPGA implementations. Therefore, "latency $\times$ area requirement" metric is commonly used to compare two implementations, which perform the same computational task. For instance, our 1–Core and 2–Core designs, which employ 22–bit digit–based multiplier, perform the same computational task. The 1–core design requires 23,074 LUTs and completes the computation in 20,42 μs, while 2–core design requires 10,21 μs to finish the multiplication and the area requirement is 46,168. As a result, both of the designs have the same "latency $\times$ area" values. Therefore, we can simply claim that these two designs provide the same performance from the point of "latency $\times$ area" metric.

In cryptographic designs, an implementation, which provides a higher security level, requires generally more area and has higher latency (or longer critical path). In other words, we can generally claim that area requirement and/or latency of a cryptographic implementation on FPGA are directly proportional to a given security level. For these reasons, we define one more metric, namely "Area-Time per Bit" in Table 7, in order to compare the FPGA implementations more fairly. We compute this metric as follows

$$Area - Time\ per\ Bit = \#\ of\ LUTs \times latency \times n^{-1}, \tag{9}$$

where n is binary field extension and latency is in μs. We can use this metric only to compare the FPGA designs. Naturally, "Area-Time per Bit" cannot be computed for ASIC and high-end microprocessors. In terms of this metric, 26–bit digit–based and 2–level Karatsuba–based multipliers give the best results among all of our digit–based and Karatsuba–based designs. In Table 8, the results of our designs are reported for different base field multipliers. Note that, the values of the metric for the same base field multiplier are equivalent for 1–core and 2–core designs.

Historically, binary curves defined over the field $\mathbb{F}_{2^{163}}$, which offers approximately 80 bits of security, have been selected for a number of scalar point multiplication

Base Multiplier Type	**Area-Time per Bit**
22–bit Digit–Based	1855
26–bit Digit–Based	1742
32–bit Digit–Based	1802
43–bit Digit–Based	2116
1–level Karatsuba	904
2–level Karatsuba	808
3–level Karatsuba	1259

Table 8: Area-Time per Bit Implementation results

accelerators. In addition, some of these designs, which are reported in the literature, are not secure against side-channel attacks. Some of the architectures reported in Table 7 compute the point multiplication on Koblitz curves, which are seen by many hardware designers as the fastest choice for elliptic curve cryptography. On the other hand, the scalar τ-NAF conversion, which is a relatively costly operation, is required for Koblitz Curves and the cost of this operation is not included in the computational and area costs of some the scalar multiplication designs, as in the case of the architectures reported in [17].

As can be seen, our work is highly competitive compared with all the other designs, which are reported in Table 7. In terms of latency, only the design, which is reported in [19], has 13% lower latency than our 2–Cores 2–Level Karatsuba-based scalar multiplier. The latency of design, which is reported in [17], has the second lowest latency, which is %23 higher than our work. These are only designs, which gives better results than our work in terms of "Area-Time per Bit" metric. However, these two designs offer a lower security level than our work. They uses the field $\mathbb{F}_{2^{163}}$. In addition, the scalar τ-NAF conversion was not implemented in [17]. Consequently, our 2-Core design option achieves the fastest constant-time elliptic curve scalar point multiplication computation at the 128-bit security level for any hardware or software platform.

5 Conclusion

In this thesis, we proposed a fast and efficient hardware accelerator for elliptic curve point multiplication over binary extension field $\mathbb{F}_{q^2}$, where $q = 2^{127}$, designed and optimized for XILINX Kintex 7 family FPGA devices. We experimented with different algorithms and architectures for multiplication in $\mathbb{F}_q$ and reported the best architectures for the target device. Firstly, we proved that the scalar point multipliers based on Karatsuba algorithms are superior to digit-based multipliers. In addition, the scalar point multiplier has relatively simple and fast logic realization on the target FGPA by taking the advantage of carry-free nature of arithmetic in binary extension fields. Furthermore, we found out that the routing part of the design dominates the critical path delay of the circuit.

We utilized the GLS elliptic curves that allow us to exploit parallelism at two different levels. At the first level, we perform elliptic curve addition and doubling operations concurrently by utilizing the Montgomery Ladder algorithm. And at the second level we reduced a 253-bit scalar point multiplication into two independent 126-bit scalar point multiplication that can be computed in parallel taking the advantage of endomorphism endowed to GLS curves. In addition, the formulae for the point arithmetic of the GLS elliptic curves enable to hide computational cost of addition and squaring operations in $\mathbb{F}_{q^2}$. As a result of these design choices and extensive optimizations, to the best of our knowledge, our design achieves the best

execution time for elliptic curve scalar point multiplication operation at the security level of 128 bits. Moreover, our design provides a first line of defence against side-channel attacks as we utilize the constant-time Montgomery Ladder algorithm. The design was tested on the XILINX KINTEX 7 FPGA using the reported maximum clock frequency of 250 MHz.

And yet, the following issues can be worked to improve the design. First of all, different design choices for the base multiplier in $\mathbb{F}_q$ can be further experimented. Especially, Karatsuba-based multipliers, which potentially offer multiple design options, can give better results. Moreover, the routing delays dominate the critical path of the design. Improvement on the routing performance of the place-and-route tool with small changes in the pipeline structure can crucially improve the performance of the design. Finally, pre-computation techniques can also be used to accelerate the Montgomery Ladder algorithm.

Appendix: Parameters used for the Galbraith-Lin-Scott elliptic curve

Let the Galbraith-Lin-Scott elliptic curve is defined over $\mathbb{F}_{q^2}$, where $q = 2^n$, with $n = 127$, as follows

$$\tilde{E}/\mathbb{F}_{q^2} : y^2 + xy = x^3 + a'x^2 + b,$$

with $a' \in \mathbb{F}_{q^2}$ such that $Tr(a') = 1$, and $b \neq 0$. Then, $\#\tilde{E}(\mathbb{F}_{q^2}) = h \cdot r$, where $h = 2$ and r is 253-bit prime number.

In this thesis, the parameters, which are shown in below, were used to define the binary GLS elliptic curve $\tilde{E}(\mathbb{F}_{q^2})$.

- $a' = u$
- $b \in \mathbb{F}_q$ is a degree 126 binary polynomial that can be represented in hexadecimal format as,
  $b = 0x540451444104015441015405405151 01$
- The 253-bit prime order r of the main subgroup of $\tilde{E}(\mathbb{F}_{q^2})$ is,

$$r = 0x1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF$$
$$FFDAC40D1195270779877DABA2A44750A5;$$

- The affine base point $P = (x_p, y_p)$ of order r is,

$$\begin{aligned} x_p &= 0x4a21a3666cf9caebd812fa19df9a3380 \\ &\quad + 0x358d7917d6e9b5a7550b1b083bc299f3 \cdot u \\ y_p &= 0x6690cb7b914b7c4018e7475d9c2b1c13 \\ &\quad + 0x2ad4e15a695fd54011ba179d5f4b44fc \cdot u \end{aligned}$$

References

[1] Reza Azarderakhsh and Arash Reyhani-Masoleh. Parallel and high-speed computations of elliptic curve cryptography using hybrid-double multipliers. *IEEE Trans. Parallel Distrib. Syst.*, 26(6):1668–1677, 2015.

[2] Daniel J. Bernstein, Chitchanok Chuengsatiansup, David Kohel, and Tanja Lange. Twisted hessian curves. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015*, volume 9230 of *Lecture Notes in Computer Science*, pages 269–294. Springer, 2015.

[3] Jean-Luc Beuchat, Takanori Miyoshi, Yoshihito Oyama, and Eiji Okamoto. Multiplication over F_{p^m} on FPGA: A survey. In Pedro C. Diniz, Eduardo Marques, Koen Bertels, Marcio Merino Fernandes, and João M. P. Cardoso, editors, *Reconfigurable Computing: Architectures, Tools and Applications, Third International Workshop, ARC 2007, Mangaratiba, Brazil, March 27-29, 2007.*, volume 4419 of *Lecture Notes in Computer Science*, pages 214–225. Springer, 2007.

[4] Jesús-Javier Chi and Thomaz Oliveira. Attacking a binary GLS elliptic curve with magma. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015*, volume 9230 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2015.

[5] Craig Costello and Patrick Longa. Four($\mathbb{Q}$): Four-Dimensional Decompositions on a ($\mathbb{Q}$)-curve over the Mersenne Prime. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 214–235. Springer, 2015.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[7] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, jul 1985.

[8] Armando Faz-Hernández, Patrick Longa, and Ana H. Sánchez. Efficient and secure algorithms for glv-based scalar multiplication and their implementation on GLV-GLS curves (extended version). *J. Cryptographic Engineering*, 5(1):31–52, 2015.

[9] Steven D. Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Des. Codes Cryptography*, 78(1):51–72, 2016.

[10] Steven D. Galbraith, Xibin Lin, and Michael Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 518–535. Springer, 2009.

[11] P. Gaudry, F. Hess, and N. P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15:19–46, March 2002.

[12] D. Hankerson, K. Karabina, and A. Menezes. Analyzing the Galbraith-Lin-Scott Point Multiplication Method for Elliptic Curves over Binary Fields. *Computers, IEEE Transactions on*, 58(10):1411 – 1420, October 2009.

[13] F. Hess. Generalising the GHS Attack on the Elliptic Curve Discrete Logarithm Problem. *LMS Journal of Computation and Mathematics*, 7:167–192, June 2004.

[14] Yun-Ju Huang, Christophe Petit, Naoyuki Shinohara, and Tsuyoshi Takagi. Improvement of faugère et al.'s method to solve ecdlp. In Kazuo Sakiyama and Masayuki Terada, editors, *Advances in Information and Computer Security - IWSEC 2013*, volume 8231 of *Lecture Notes in Computer Science*, pages 115–132. Springer, 2013.

[15] Koh ichi Nagao. Complexity of ecdlp under the first fall degree assumption. Cryptology ePrint Archive, Report 2015/984, 2015. `http://eprint.iacr.org/`.

[16] Kimmo Järvinen. Optimized FPGA-based elliptic curve cryptography processor for high-speed applications. *Integration*, 44(4):270–279, 2011.

[17] Kimmo U. Järvinen and Jorma Skyttä. Fast point multiplication on koblitz curves: Parallelization method and implementations. *Microprocessors and Microsystems - Embedded Hardware Design*, 33(2):106–116, 2009.

[18] Koray Karabina. Point decomposition problem in binary elliptic curves. Cryptology ePrint Archive, Report 2015/319, 2015. `http://eprint.iacr.org/`.

[19] Zia Uddin Ahamed Khan and Mohammed Benaissa. High speed ECC implementation on FPGA over gf(2^{m}). In *25th International Conference on Field Programmable Logic and Applications, FPL 2015, London, United Kingdom, September 2-4, 2015*, pages 1–6, 2015.

[20] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[21] Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari Kermani. Low-resource and fast binary edwards curves cryptography. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology - INDOCRYPT 2015*, volume 9462 of *Lecture Notes in Computer Science*, pages 347–369. Springer, 2015.

[22] Tanja Lange and Igor Shparlinski. Collisions in fast generation of ideal classes and points on hyperelliptic and elliptic curves. *Appl. Algebra Eng. Commun. Comput.*, 15(5):329–337, 2005.

[23] Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over GF(2^{m}) without precomputation. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International*

Workshop, CHES'99, volume 1717 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 1999.

[24] Alfred Menezes and Minghua Qu. Analysis of the weil descent attack of gaudry, hess and smart. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 2001.

[25] Alfred Menezes and Scott A. Vanstone. Elliptic curve cryptosystems and their implementations. *J. Cryptology*, 6(4):209–224, 1993.

[26] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[27] Peter L Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[28] C. NIST. The digital signature standard. *Commun. ACM*, 35:36–40, jul 1992.

[29] Thomaz Oliveira, Diego F. Aranha, Julio López Hernandez, and Francisco Rodríguez-Henríquez. Fast point multiplication algorithms for binary elliptic curves with and without precomputation. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014*, volume 8781 of *Lecture Notes in Computer Science*, pages 324–344. Springer, 2014.

[30] Thomaz Oliveira, Julio López, Diego F. Aranha, and Francisco Rodríguez-Henríquez. Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Cryptographic Engineering*, 4(1):3–17, 2014.

[31] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Eurocrypt*, 1999.

[32] Christophe Petit and Jean-Jacques Quisquater. On polynomial systems arising from a weil descent. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 451–466. Springer, 2012.

[33] Chester Rebeiro, Sujoy Sinha Roy, and Debdeep Mukhopadhyay. Pushing the limits of high-speed GF(2 m) elliptic curve scalar multiplication on fpgas. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 494–511. Springer, 2012.

[34] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[35] Sujoy Sinha Roy, Kimmo Järvinen, and Ingrid Verbauwhede. Lightweight coprocessor for koblitz curves: 283-bit ECC including scalar conversion with only 4300 gates. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 102–122. Springer, 2015.

[36] J. Salowey. Confirming Consensus on removing RSA key Transport from TLS 1.3. Transport Layer Security working group of the IETF Mailing List, May 2014.

[37] Igor Semaev. New algorithm for the discrete logarithm problem on elliptic curves. Cryptology ePrint Archive, Report 2015/310, 2015. `http://eprint.iacr.org/`.

[38] Michael Shantz and Edlyn Teske. Solving the elliptic curve discrete logarithm problem using semaev polynomials, weil descent and gröbner basis methods - an experimental study. In Marc Fischlin and Stefan Katzenbeisser, editors, *Number Theory and Cryptography - Papers in Honor of Johannes Buchmann on*

the Occasion of His 60th Birthday, volume 8260 of *Lecture Notes in Computer Science*, pages 94–107. Springer, 2013.

[39] J. Silverman. The arithmetic of elliptic curves. graduate texts in mathematics series. *Springer*, 2010.

[40] Thomas Unterluggauer and Erich Wenger. Efficient pairings and ECC for embedded systems. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2014.

[41] Zhenwei Zhao and Guoqiang Bai. Ultra high-speed SM2 ASIC implementation. In *13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2014*, pages 182–188. IEEE Computer Society, 2014.