INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Portable Embedded Systems

Efficient Units for Data Processing and Cryptography

Samuel Freitas Antão

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente:	Doutor José António Beltran Gerald
Orientador:	Doutor Leonel Augusto Pires Seabra de Sousa
Vogais:	Doutor Carlos Nuno da Cruz Ribeiro

Setembro de 2008

Acknowledgments

I would like to express my gratitude to everyone that, somehow, helped me achieving this objective.

To my supervisor, Professor Leonel Sousa for his permanent support and for giving me the opportunity to perform this work over a subject which revealed to be so much interesting.

To Ricardo Chaves, for his valuable suggestions and for thought me so many things. To Nuno Sebastião for his patience to answer all my questions and for his expertise. To José Germano, always available to contribute with his experience. To the people of the SiPS group and to my Instituto Superior Técnico University colleagues for their support and friendship.

For last, but not less important, to Inês and my Family for their motivation and for being by my side.

Thank you, to you all.

Abstract

Most of nowadays applications of electronic devices demand secure communication, which justifies the existence of embedded cryptographic algorithms in these devices. The Elliptic Curve Cryptography public key system has revealed to be more appropriate to embedded systems with limited power and memory resources, when compared with other approaches, such as the widely used Rivest-Shamir-Adleman (RSA) public key system. In this thesis the Elliptic Curve properties are carefully analyzed in order to develop a complete cryptographic processor, which can compute all the necessary arithmetic needed to implement security protocols supported on elliptic curves. Different algorithms to perform the elliptic curve cryptography are analyzed in order to select the most appropriate ones, regarding the introduction of a new efficient method that uses a compact representation of elliptic curve elements. This method, which is designated coordinate collapsing, allows to perform the elliptic curve exponentiation without using the traditional two coordinates to represent a curve element. This improvement allows to reduce the bandwidth requirements by half.

A prototype of the developed processor is also presented. This prototype provides to the host system the complete arithmetic support to elliptic curve cryptographic capabilities when communicating with remote systems. This prototype was thoroughly tested using real protocols which confirms this processor applicability into commercial systems.

Keywords

Elliptic Curve Cryptography, Coordinate Collapsing, Embedded Systems, FPGA, ASIC

Resumo

A maioria das aplicações de dispositivos electrónicos de hoje em dia requerem comunicação segura, o que justifica a existência de algoritmos criptográficos embebidos nestes dispositivos. O sistema de chaves públicas de Criptografia por Curvas Elípticas tem revelado ser mais apropriado para sistemas embebidos com limitados recursos de potência e memória quando comparado, por exemplo, com o sistema geralmente mais usado de chaves públicas Rivest-Shamir-Adleman (RSA). Nesta tese as propriedades das Curvas Elípticas são cuidadosamente analisadas com o objectivo de desenvolver um processador criptográfico completo, capaz de calcular eficientemente a aritmética necessária para implementar protocolos de segurança suportados em curvas elípticas. Vários algoritmos para realizar a aritmética sobre curvas elípticas são analisados, seleccionando-se os mais apropriados tendo em conta a introdução de um método capaz de usar uma representação compacta dos elementos da curva elíptica. Este novo método, designado por colapso de coordenadas, permite a realização eficiente da exponenciação sobre curvas elípticas sem utilizar as tradicionais duas coordenadas para representar um elemento da curva. Esta melhoria permite a redução dos requisitos de largura de banda para metade.

Um protótipo do processador desenvolvido é também apresentado. Este protótipo fornece ao sistema anfitrião um suporte aritmético completo de criptografia sobre curvas elípticas aquando da comunicação com um sistema remoto. Este protótipo foi minuciosamente testado usando protocolos reais, confirmando a aplicabilidade deste processador em sistemas comerciais.

Palavras Chave

Criptografia sobre Curvas Elípticas, Colapso de Coordenadas, Sistemas Embebidos, FPGA, ASIC

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AES	Advanced Encryption Standard
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
CPU	Central Processing Unit
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECPC	Elliptic Curve Processor Control
FIFO	First-In, First-Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPIO	General Purpose Input/Output
I/0	Input/Output
IP	Internet Protocol
LAD	Local Address Data
LMB	Local Memory Bus
LNS	Logarithmic Number System
LUT	Look-Up Table
NAF	Non-Adjacent Forms
OPB	On-Chip Peripheral Bus
OS	Operative System

PC	Personal Computer
PCMCIA	Personal Computer Memory Card International Association
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RNS	Residue Number System
RSA	Rivest, Shamir, and Adleman
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmiter
USB	Universal Serial Bus
WEP	Wireless Encryption Protocol

1 Introduction

Contents

Most electronic devices, in particular embedded ones, such as cell phones, personal computers, smart cards, attempt to provide secure communication and/or authentication for their users. With such security features, these electronic devices allow the users to keep their privacy, e.g. to manage their online bank accounts prevented from attacks, and to unequivocally identify themselves. It is within this scenario that cryptographic algorithms and related data processing are used. Asymmetric key cryptosystems, sometimes called public key cryptosystems, allow to perform these tasks over public unsecure channels. Unlike the symmetric key cryptosystems, public key cryptosystems do not need the existence of pre-defined values to establish a secure communication channel between two entities. The secret key is formed by private information, that is never communicated, and public information that can be communicated without compromising the secret.

Because secure communication is so often used, a significant effort has been spent in the literature in performing the cryptographic procedures as efficient as possible, regarding circuit area, power consumption, throughput, and bandwidth requirements. For this reason, dedicated hardware implementations of cryptographic processors are suitable for applications where this efficiency is the key for the success. The requirements depend on the applications. For example, if a large amount of data, such as a video stream, has to be transferred or if there is a server which needs to establish a large amount of secure connections, the throughput requisite is prioritized. Otherwise, if a small message transfer or a single identification routine is to be performed by a portable device, the power consumption is more important.

The Elliptic Curve Cryptography (ECC) appears as a competitive solution which can balance both power needs and bandwidth reduction by mitigating the communication overhead due to the keys transmission. This thesis discusses the implementations of ECC and proposes a processor based on this cryptosystem. Original properties are exploited to reduce the bandwidth requirements.

1.1 Motivation and Related Work

The Elliptic Curve (EC) cryptosystems were proposed by Koblitz and Miller, in 1985 [1, 2]. However, until now, other public key cryptosystems have been widely used, such as Rivest, Shamir, and Adleman (RSA). The motivation to research a different cryptosystem arose from a new set of constraints, namely the increasing usage of portable devices with reduced power and computing resources, the increasing demand for higher communication security and the fact that the communication bandwidth becomes more valuable. Since EC cryptosystems were initially proposed, the analysis of its properties revealed that the same security level can be achieved for significantly smaller key sizes, allowing for the computation and transmission of less bits, which, in turn, enhances the performance, decreases the computing effort, and the bandwidth requirements.

RSA remains the most known and widely used cryptosystem. However, the use of RSA in portable and autonomous embedded systems is becoming critical, due to its power consumption needs and used key lengths. ECC appears as an interesting alternative since its arithmetic is computationally more efficient and present higher security per key bit [3]. In the last years several ECC implementations were proposed, and designed to be integrated in standard network communication protocols that require public key systems, such as Transport Layer Security (TLS), Secure Sockets Layer (SSL), Secure Internet Protocol (IP), and Wireless Encryption Protocol (WEP) [4]. These related works propose co-processors for network servers, offering time and power efficient alternatives for computing the encryption. Other proposals have targeted these implementations to embedded systems with scarce resources, which can be used, for example, in medical applications [5]. The existent proposed implementations can be classified into three groups: full hardware, full software, and software implementations with dedicated hardware co-processors. In full hardware implementations all the operations, from finite field operations to EC arithmetic, are performed in dedicated hardware structures. Results suggest that the arithmetic operations over binary extension fields (Galois field $GF\left(2^k\right)$) are better suited for hardware implementations, where polynomial basis, normal basis, and optimal normal basis are used to represent the field elements [6]. Nevertheless, there are also a few implementations using prime finite fields (Galois field $GF\left(p\right)$) [7]. Full software solutions are supported by general purpose processors and are mostly developed by using $GF\left(p\right)$. Few software implementations also exist using $GF\left(2^k\right)$ [8]. Software implementations using dedicated co-processors consist of a general purpose processor extended with a dedicated co-processor, used to compute dedicated ECC arithmetic operations [9].

Alternative (projective) representations of the EC elements can also be used, in order to improve the computation performance. This reduces the use of the most expensive operations over the field, namely the inversion operation [10].

Another important issue is the resistance to attacks. A processor for cryptographic applications should not reveal the secret data being processed through its time and power consumption characteristics. This can be achieved using appropriate algorithms which take fixed time and use, during the same time, the same arithmetic units independently of the secrets being processed. The first requisite leads to designs that are more resistant to time attacks, while the second requisite leads to designs with increased power analysis resistance, since using the same units reduces the fluctuations of the power consumption figure. In other words, there should be minimal correlation between the data being processed and processors characteristics, thus all the methods which allow to reduce this correlation improve the resistance of the processor from attacks. More details about time and power attacks can be found in [11].

The motivation of this thesis is the design of a dedicated hardware processor capable of computing all the required cryptographic procedures for the EC cryptography calculation. This thesis also contributes to the design of more competitive EC cryptosystems by studying and exploiting its properties from different and new approaches.

1.2 Objectives

This thesis objectives are the analysis of the mathematical background which support the EC operations in order to identify which approach is better suited for dedicated hardware processor designs. This background is used to design algorithms which allow to compute the EC operations inserting an optimization level, which may be of concept, parallelization and elements representation. The properties of these algorithms are then discussed in order to obtain the basic arithmetic units and an efficient method to interconnect them, obtaining the complete processor. The proposed processor is then tested by using a Field Programmable Gate Array (FPGA) technology as the target, and the results for this complete implementation are obtained and presented for this technology. Other implementations in software and Application Specific Integrated Circuit (ASIC) are also performed and described, allowing a comparative evaluation with the FPGA processor. This thesis also presents the results obtained with the proposed processor into a real system application. These results allow to confirm the feasibility of the proposed processor as a complete system which provides all the ECC support to a host computer.

1.3 Original Contributions

Some original contributions can be identified in this thesis. The first allow to efficiently exploit compact representations of EC points. An EC point has two coordinates, however, these two coordinates can be represented by an element with the size of only one of their coordinates, particularly if the arithmetic of the EC is performed over a cyclic subgroup with prime order. However, this property has not been exploited in the literature for the designing of cryptographic processors, since more attention is left on the cryptographic computation speed up or/and power efficiency, regardless the communication issues. In this thesis a new algorithm is proposed, which allows the full computation of point exponentiation using the compact representation without compromising the algorithm performance regarding the related art. A new unit to be integrated in point addition is also developed allowing this operation to efficiently support the compact representation. These contributions allow the proposed processor to use only half of the bandwidth comparing with an usual EC system, which uses the two coordinates to represent a point. These original contributions led to a communication that was accepted to be presented this year in one of the top conferences in the area of reconfigurable technology and systems [ICFPT08] and in a publication that is now being prepared to submit to a journal.

1.4 Organization

This thesis is organized in six chapters. In Chapter 2 the mathematical notions of group, field and EC group are introduced. The EC is first presented in a generic way and then particularized for finite fields, which are the fields used in this thesis. The choice of the finite field for a hardware implementation is also discussed in this chapter.

Chapter 3 describes the algorithms proposed in the related art to deal with the EC arithmetic. The analysis of these algorithms is performed in order to select the more adequate ones to be integrated in the proposed processor. The compact EC representation (collapsed representation) is also introduced in this chapter, leading to new algorithms used to perform the point addition and exponentiation.

Chapter 4 describes the proposed units used for computing the algorithms presented and selected in Chapter 3. In this chapter the characteristics of these implemented units are analyzed. The characteristics of the complete proposed design are discussed regarding the related art for FPGA technologies. An implementation for an ASIC target is also presented, in order to evaluate the design characteristics for this technology and to compare it with the FPGA implementation.

In Chapter 5 is presented a software implementation to compare with the obtained hardware processor. This software implementation is a mirror of the hardware processor, in terms of implemented algorithms. A prototype using the FPGA processor for a real system application is also presented and the obtained results discussed. The results for these systems are also presented and discussed.

Concluding remarks are presented in Chapter 6 based on the results obtained for the ECC processor proposed in this thesis. Future work is also suggested in this chapter.

1.5 Publication that results from this Thesis

[ICFPT08] S. Antão, R. Chaves, L. Sousa, ”FPGA Elliptic Curve Cryptographic Processor over $GF(2^m)$ with Coordinate Collapsing”, International Conference on Field-Programmable Technology (ICFPT’08), Taiwan, December 2008.

2 Mathematical Support

Contents

In this chapter, the EC arithmetic is formally introduced. First, algebraic groups and fields properties are discussed, followed by the definition of the operations over a special class of fields: the finite fields. To conclude, the high-level EC arithmetic is introduced.

The information introduced in this chapter is the base of an EC cryptosystem, for whose the computed procedures are realized over elements belonging to fields and groups, using different operations defined for these fields and groups. Moreover, the optimization of the computed arithmetic for the cryptosystem arises from specific properties of these entities.

Different fields may suit different applications and different computing support. In this chapter some considerations are introduced in order to justify the project choices for the system proposed in this thesis.

2.1 Groups and Fields

In this section, the properties of groups and fields will be presented. Both groups and fields are algebraic structures defined to describe a set of elements and the relationship between them. The nature of these elements varies, depending on the application. The relationship between these elements is defined in operations, which permit to achieve an element combining other elements. The way this combination is performed depends on the type of group or field. In this sense a field has more stringent properties than a group, because it is assured that two operations (and their inverses) are defined for fields, instead of only one operation for groups. This means that it is possible to combine elements in a field in more different ways than in a group. Furthermore, a field may contain groups supported on its operations.

2.1.1 Groups

Generally, a group is a set of elements and one operation which permit to combine two elements in order to obtain a third one. A group is the fundamental algebraic entity over which multiple other entities are constructed, such as rings or fields.

The group theory origin is related to three different subjects: the algebraic equations solving, number theory and geometry aspects. The first mathematicians which address this theory were the Norwegian Niels Abel and the French Joseph Louis Lagrange and Évariste Galois. This last mathematician name is applied to a special class of fields (finite fields) which are particularly important for cryptographic applications.

Formally, a group $(G, +)$ is a set of elements along with an operation $+$ that obeys the following axioms [12]:

- The operation $+$ has the property of closure: for any $g_1, g_2 \in G$, $(g_1 + g_2) \in G$;
- The operation $+$ is associative: for any $g_1, g_2, g_3 \in G$, $(g_1 + g_2) + g_3 = g_1 + (g_2 + g_3)$;

- The operation $+$ has an identity element: there is an element $g_{id+} \in G$ such that for any $g_1 \in G$, $g_1 + g_{id+} = g_{id+} + g_1 = g_1$;
- Any element g_1 has an inverse towards the operation $+$: there is an element $g_2 \in G$ such that $g_1 + g_2 = g_2 + g_1 = g_{id+}$.

If a group $(G, +)$, besides the previous axioms, obeys the following:

- The operation $+$ is commutative: for any $g_1, g_2 \in G$, $g_1 + g_2 = g_2 + g_1$,

it is called an abelian or commutative group.

The exponentiation over a group $(G, +)$ is defined as the recurrent application of the operation $+$. A group is said to be cyclic if there is an element g_g, called a generator, such that all its powers generate the group. Thus, any element of the group can be written as:

$$g_1 = g_g^i = \underbrace{g_g + g_g + \ldots + g_g}_{i},\ i \in \mathbb{N}.$$

Groups may contain subgroups inside, particularly cyclic subgroups. A cyclic subgroup $(G_c, +)$ has an order p which consists of the smallest integer p such that $g_g^p = g_{id+}$. In practice, p is the number of elements in the subgroup. For p prime, it holds that for all $g_1 \in G_c$, exists an element $g_2 \in G_c$ such that $g_1 = g_2 + g_2 = 2g_2$ [13]. When a group is not cyclic, its order is defined as the number of elements that constitute this group.

2.1.2 Fields

A field, as a group, is an algebraic entity. A field is constructed over a group, thus inherits its properties. To establish a field, a set of elements must allow the definition of another operation, other than the group operation. For this reason the field properties are more stringent. The concept of field was first introduced by the German mathematician Richard Dedekind.

The formal definition of a field $(F, +, \times)$ is a set of elements along with the operations $+$ and $\times$, such that the following properties hold:

- The field elements have the properties of a commutative group towards the operation $+$;
- The operation $\times$ has the property of closure: for any $f_1, f_2 \in F$, $(f_1 f_2) \in F$;
- The operation $\times$ is associative: for any $f_1, f_2, f_3 \in F$, $(f_1 f_2) f_3 = f_1 (f_2 f_3)$;
- The operation $\times$ is distributive towards the operation $+$: for all $f_1, f_2, f_3 \in F$, $f_1(f_2 + f_3) = f_1 f_2 + f_1 f_3$ and $(f_1 + f_2) f_3 = f_1 f_3 + f_2 f_3$;
- The operation $\times$ is commutative: for any $f_1, f_2 \in F$, $f_1 f_2 = f_2 f_1$;
- Any element f_1 has an inverse towards the operation $\times$: there is an element $f_2 \in F$ such that $f_1 f_2 = f_2 f_1 = f_{id\times}$.

- Except for the operation $+$ identity, any field element has an inverse towards the operation $\times$: for any element $f_1 \in F$, with $f_1 \neq f_{id+}$, there is an element $f_2 \in F$ such that $f_1 f_2^{-1} = f_{id\times}$.

Note that the $\times$ operation is usually expressed as the concatenation of the elements which intervene in the operation, i.e., $f_1 \times f_2 \equiv f_1 f_2$.

2.2 Finite Fields

In this section the properties and operations over a particular type of fields are described. These fields have a finite number of elements. Thus, they are called finite fields or Galois fields. From these fields, two of them have a major importance in cryptographic applications, namely prime fields ($GF(p)$) and binary extension fields ($GF(2^k)$).

2.2.1 Prime Fields $GF(p)$

This field is composed by the set $\mathbb{Z}_p$ of integers $\{0, 1, 2, \ldots, p-1\}$ and the operations modulo p, with p a prime. The following depicts an example of the addition, subtraction and multiplication operations over a field with $p = 19$:

$$\begin{aligned}
&(7 + 15) \equiv 22 \equiv 22 - 19 \equiv 3 \; mod \; 19;\\
&(7 - 15) \equiv -8 \equiv -8 + 19 \equiv 11 \; mod \; 19;\\
&(7 \times 15) \equiv 105 \equiv 105 - (19 \times 5) \equiv 10 \; mod \; 19,
\end{aligned}$$

where $a \equiv b \equiv c \; mod \; p$ means that $a \; mod \; p = b \; mod \; p = c \; mod \; p$. Note that $n \times p \equiv 0 \; mod \; p$, with n an integer.

Since p is prime, multiplying an element by every elements of the set, certainly all the residues from 0 to $p-1$ will be obtained. The multiplicative inverse of that field element will be the element that had originated the residue 1. For example $7 \times 11 \equiv 1 + 4 \times 19 \equiv 1 \; mod \; 19$, thus $7^{-1} \times 7 \times 11 \equiv 7^{-1} \times 1 \equiv 11 \equiv 7^{-1} \; mod \; 19$. The most used algorithm to perform field elements inversion is the *Extended Euclidean Algorithm* [12], which can compute an element which has 1 as the greatest common divider with a given element. The computed element is the multiplicative inverse of the given one.

2.2.2 Binary Extension Fields $GF(2^k)$

A binary extension field $GF(2^k)$ is constructed by projecting a binary field $GF(2)$ for k dimensions. These fields have the advantage of not waste the possible representations of a number with a binary vector, as is the case of $GF(p)$.

The operations over this field cannot be performed modulo 2^k, since the set $\mathbb{Z}_{2^k}$ does not create a field. Hence, the operations over this kind of fields are defined differently.

It is important to introduce the polynomial representation of a field element. In this representation a polynomial of order k, with binary coefficients, is used to represent an element. This way, an element A of $GF(2^k)$ with the binary representation $(a_{k-1}a_{k-2},\ldots,a_1,a_0)$ is written as a polynomial of powers of x:

$$A=\sum_{i=0}^{k-1}a_ix^i \tag{2.1}$$

For a binary field GF(2^k) it is possible to get an order k irreducible polynomial with binary coefficients $P\left(x\right)$. An irreducible polynomial is a polynomial which cannot be factorized into polynomials with lower order. There are algorithms to obtain those irreducible polynomials with reasonable computer effort [14]. Nonetheless, in this thesis, there will be used already known polynomials. It is common to use trinomials or pentanomials [15], which are polynomials with three or five non zero coefficients, respectively. Known this polynomial, all the operations are performed modulo this irreducible polynomial.

There are alternative representations of field elements. It is possible to use a canonical base which consists of the first k powers of an element $\alpha\in GF(2^k)$ [16]. Then, it is possible to obtain any of the 2^k elements β of the field using this basis $(1,\alpha,\alpha^2,\ldots,\alpha^{k-1})$:

$$\beta=\sum_{i=0}^{k-1}a_i\alpha^i \tag{2.2}$$

Another type of bases is the normal basis. In this basis there is an element $\alpha\in GF(2^k)$ which generates k linear independent vectors as $(\alpha^{2^0},\alpha^{2^1},\ldots,\alpha^{2^{k-1}})$ [16]. There are specific normal bases called optimal normal bases type I and type II. The type II optimal normal basis is more usual and leads to efficient multipliers. A type II optimal normal basis is constructed like a normal basis but with $\alpha=\gamma+\gamma^{-1}$, where γ has the properties $\gamma^{2k+1}=1$ and $\gamma^n\neq 1$, with $0<n<2k+1$. This kind of basis do not exist for every field $GF(2^k)$, since k must obey to the following conditions [16]:

1. $p=2k+1$ is a prime number;
2. 2 is a primitive root modulo p, which means that all the powers of 2 generate the set $\mathbb{Z}_p$;
3. $p\equiv 7\ mod\ 8$ and k is the smallest integer that verifies $2^k\equiv 1\ mod\ p$, which means that (-1) is a quadratic nonresidue modulo p and 2 generate all the quadratic residues modulo p.

The advantage of such basis is related to the minimization of the dependencies of each product bit from the input operand bits. Another advantage of normal basis is the squaring operation which can be efficiently performed with a cyclic left shift. This property can be observed considering the *Fermat Little Theorem* which states that for any field element $\alpha\in GF(2^k)$, $\alpha=\alpha^{2^k}$. Representing a field element in terms of a normal basis:

$$\beta=\sum_{i=0}^{k-1}a_i\alpha^{2^i}, \tag{2.3}$$

thus its squaring is defined as:

$$\beta^2 = \left(\sum_{i=0}^{k-1} a_i \alpha^{2^i}\right)^2 = \sum_{i=0}^{k-1} a_i \alpha^{2^{i+1}} = a_k \alpha^{2^k} + \sum_{i=0}^{k-2} a_i \alpha^{2^{i+1}} = a_k \alpha + \sum_{i=1}^{k-1} a_{i-1} \alpha^{2^i}. \tag{2.4}$$

Nevertheless, in this thesis the polynomial representation is adopted, because it brings interesting properties for the proposed EC units, as discussed later on.

Using polynomial basis, the operations over $GF(2^k)$ can be summarized as:

1. Addition: bitwise XOR reduction $P(x)$, $(A\ XOR\ B)\ mod\ P(x)$;
2. Multiplication: polynomial multiplication reduction $P(x)$, $(A \times B)\ mod\ P(x)$;
3. Inversion: find the element (A^{-1}) which multiplied by the operand (A) results 1 reduction $P(x)$, $(AA^{-1}) \equiv 1\ mod\ P(x)$.

As explained, $GF(2^k)$ is obtained by expanding $GF(2)$ into k dimensions. In the binary field $GF(2)$ the addition is performed with a XOR between the two input bits. Thus, in $GF(2^k)$, the operation in $GF(2)$ must be expanded for the k dimensions resulting in a bitwise XOR between the k bits that represent the field elements, for example ($k = 5$):

$$(x^4 + x^3 + x + 1) + (x^3 + 1) \equiv x^4 + x\ mod\ \left(x^5 + x^2 + 1\right).$$

The multiplication of two polynomials:

$$A(x) = a_{k-1}x^{k-1} + \ldots + a_1 x + a_0 \text{ and } B(x) = b_{k-1}x^{k-1} + \ldots + b_1 x + b_0,$$

can be written as:

$$A(x)B(x) = \sum_{i=0}^{k-1} a_i x^i \sum_{j=0}^{k-1} b_j x^j = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} a_i b_j x^{i+j} = \sum_{j=0}^{k-1} b_j x_j A(x). \tag{2.5}$$

For example:

$$\begin{aligned}(x^4 + x^3 + x + 1) \times (x^3 + 1) &\equiv x^4 + x^3 + x + 1 + \\ &+\ x^7 + x^6 + x^4 + x^3 \\ &\equiv x^7 + x^6 + x + 1\ mod\ x^5 + x^2 + 1.\end{aligned}$$

Now, it is necessary to reduce the result to a polynomial of order smaller than $k = 5$. One alternative is to calculate the remainder of the division by the irreducible polynomial:

$$x^7 + x^6 + x + 1 = (x^5 + x^2 + 1)(x^2 + x) + (x^4 + x^3 + x^2 + 1),$$

thus, $x^7 + x^6 + x + 1 \equiv x^4 + x^3 + x^2 + 1\ mod\ x^5 + x^2 + 1$.

Another alternative is to compute the multiplication by powers of the polynomial x and to embed the reduction in this operation. It holds that for a polynomial $P(x)$ with order k:

$$x^k \; mod \; P(x) = P(x) - x^k \tag{2.6}$$

Then:

$$A(x) \times x = a_{k-1}x^k + a_{k-2}x^{k-1} + \ldots + a_1x^2 + a_0x \tag{2.7}$$

Using the (2.6) result, in $GF(2^k)$, (2.7) can be written as:

$$A(x) \times x = a_{k-1}\left(P(x) - x^k\right) + a_{k-2}x^{k-1} + \ldots + a_1x^2 + a_0x \tag{2.8}$$

This result shows that if $a_{k-1} = 0$ a multiplication by x corresponds to a one position left shift, while for $a_{k-1} = 1$, beside the shift, the irreducible polynomial that generates the field must be added. To divide by x one can perform the inverse: if $a_0 = 0$ it may be performed as a one position right shift; if $a_0 = 1$, before the shift, the subtraction of the irreducible polynomial without the k order term must be performed. Note that the subtraction and the addition are the same operation over these fields. This allows the computation of the previous multiplication example as:

$$\begin{aligned}(x^4 + x^3 + x + 1) \times (x^3 + 1) \equiv\; & x^4 + x^3 + x + 1 + \\ +\; & (x^4 + x^3 + x + 1)x^3.\end{aligned}$$

Since $(x^4 + x^3 + x + 1)x^3 = ((((x^4 + x^3 + x + 1)x)x)x)$, using (2.8) for three times we obtain that $(x^4 + x^3 + x + 1)x^3 \equiv x^2 + x \; mod \; x^5 + x^2 + 1$. Then the multiplication is performed as:

$$\begin{aligned}(x^4 + x^3 + x + 1) \times (x^3 + 1) \equiv\; & x^4 + x^3 + x + 1 + \\ +\; & x^2 + x \\ \equiv\; & x^4 + x^3 + x^2 + 1 \; mod \; x^5 + x^2 + 1.\end{aligned}$$

To compute the multiplicative inverse it is possible to recall the *Fermat Little Theorem*, which states that for every field element $A \in GF(2^k)$, $A = A^{2^k}$ [17]. Thus, $A^{-1} = A^{2^k-2}$ which can be computed, regarding the property $2^k - 2 = 2 + 2^2 + \ldots + 2^{k-1}$, as $A^{-1} = A^2A^{2^2}\ldots A^{2^{k-1}}$. This means that it is possible to compute a multiplicative inverse using multiplications recurrently.

There is other method to compute the multiplicative inverse, which is the *Extended Euclidean Algorithm* that computes an element (the multiplicative inverse) which has 1 as the greatest common divisor with a given element. This algorithm is presented in Algorithm 2.1.

For example, to invert the polynomial $x^3 + 1$ modulo $x^5 + x^2 + 1$, at the first iteration $Q = x^2$, $D_1 = x^2$ and $D_2 = 1$. Thus, the algorithm stops at the second iteration and the multiplicative inverse x^2 of $x^3 + 1$ is returned.

From the presented properties, one may conclude that the fields $GF(2^k)$ are better suited for hardware implementations than prime fields. These former fields allow to efficiently represent their elements with binary vectors and permit to perform the operations with few logical gates and simple bit manipulations.

Algorithm 2.1 Extended Euclidean Algorithm

```
Require: $P(x)$ (irreducible polynomial) and $A(x)$;
Ensure: $A(x)^{-1}\ mod\ P(x)$;
  $C_1 := 0$; $C_2 := P(x)$; $D_1 := 1$; $D_2 := A(x)$;
  while $D_2 \neq 1$ do
    $Q := C_2/D_2$;
    $E_1 := C_1 - QD_1$; $E_2 := C_2 - QD_2$;
    $C_1 := D_1$; $C_2 := D_2$;
    $D_1 := E_1$; $D_2 := E_2$;
  end while
  return $D_1$;
```

2.3 Elliptic Curve Definition

An EC over a field $\mathbb{F}$, is a smooth curve that can be defined with a *long Weierstrass form* [18]:

$$y^2 + a_1xy + a_2y = x^3 + a_3x^2 + a_4x + a_5,\ a_i \in \mathbb{F} \tag{2.9}$$

The points (x,y) which obey (2.9), along with a point at infinity called $\mathcal{O}$, define a group $E(\mathbb{F})$ where new operations can be defined. Note that (2.9) corresponds to a generic description, over a generic field. However, depending on coefficients a_i and performing the appropriate coordinate changing, it is possible to rewrite (2.9), obtaining EC equations suited for particular fields $\mathbb{F}$. One of these equations is obtained for fields with characteristic different from 2 and 3, where characteristic of a field is the minimum number of times one can add the multiplicative identity to itself and obtain the additive identity:

$$y^2 = x^3 + ax + b,\ a, b \in \mathbb{F}, \tag{2.10}$$

In (2.10), in order to obey a smoothness condition, a and b must have the relation: $-\left(4a^3 + 27b^2\right) \neq 0$ [18]. Considering $\mathbb{F}$ the set of real numbers, the ECs described as in (2.10) have the shape depicted in the Figure 2.1 for different values of the parameters.

The group $E(\mathbb{F})$ is additive and commutative, which means that the addition of its elements is defined as well as the additive inverse. From this point onwards, these elements will be designated as points. Particularly, it is also possible to define a doubling operation. These operations can be described by a geometrical interpretation as depicted in Figure 2.1.

The inverse $(-P)$ of a point (P) is defined as the point $(-P)$ such that $(-P) + P = \mathcal{O}$, with $\mathcal{O}$ the point at infinity which acts as the additive identity. To obtain the inverse of an addition, a line that cross the points to add may be marked. Since $\mathcal{O}$ is vertically at infinity, to obtain $-(P + \mathcal{O})$ a vertical line that crosses P may be marked. The other point in the curve crossed by this line is $(-P)$, as depicted in Figure 2.1(c). Thus, the group inversion operation is analytically computed as:

$$P^{-1} = (x_P, y_P)^{-1} = (x_P, -y_P). \tag{2.11}$$

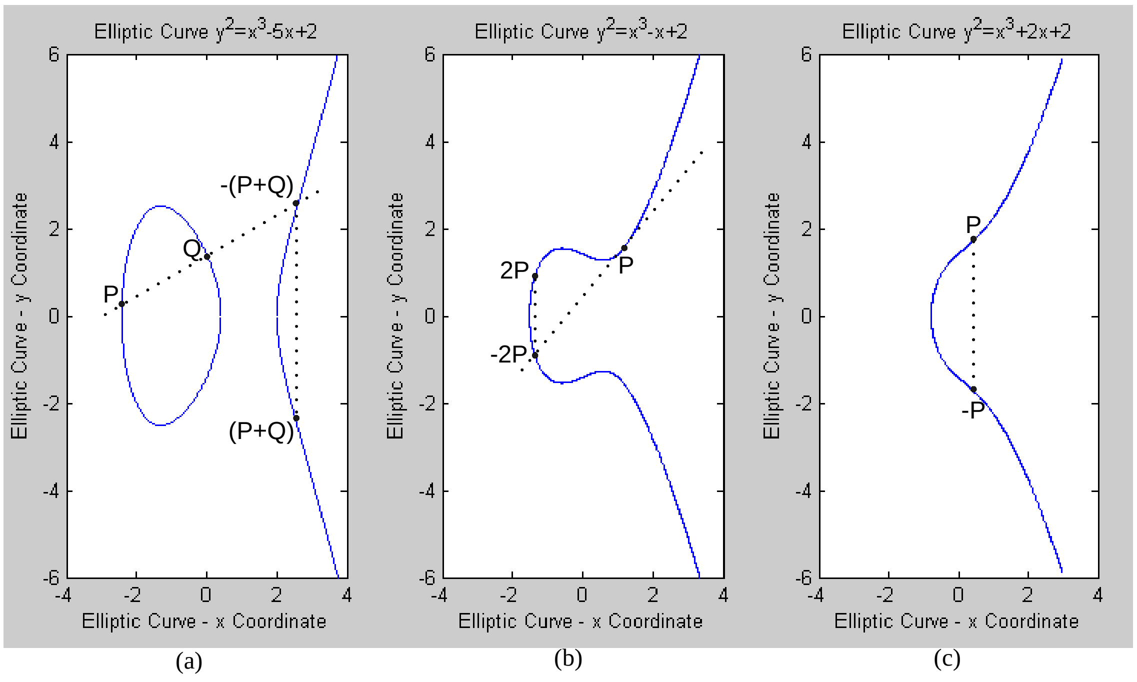

Figure 2.1: Elliptic Curve Operation Geometrical Interpretation over a Real Field

The addition operation is defined as explained for the inversion. For $P, Q \neq \mathcal{O}$, $P \neq Q$ and $P \neq -Q$ the addition result is different from $\mathcal{O}$, thus the marked line crosses P, Q and $-(P+Q)$. Then, $-(P+Q)$ may be inverted. This procedure is depicted in Figure 2.1(a). Note that if $P = -Q$ the marked line will be vertical and the result would be $\mathcal{O}$. Analytically, the computation of the addition of two point $R = P + Q$, with $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ and $R = (x_R, y_R)$, is performed as [12]:

$$\begin{aligned} \lambda &= \frac{y_Q - y_P}{x_Q - x_P}, \\ x_R &= \lambda^2 - x_P - x_Q, \\ y_R &= -y_P + \lambda(x_P - x_R). \end{aligned} \tag{2.12}$$

The doubling operation is depicted in Figure 2.1(b), and differs from the additions in the sense that the marked line is a tangent at the point to be doubled. Note that, to double $P = (x_P, y_P)$, if $y_P = 0$ the tangent is vertical, thus the result would be $\mathcal{O}$. For the other cases the analytical expression for $R = 2P$ is [12]:

$$\begin{aligned} x_R &= \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P, \\ y_R &= \left(\frac{3x_P^2 + a}{2y_P}\right)(x_P - x_R) - y_P. \end{aligned} \tag{2.13}$$

The above operations are combined in order to perform the point exponentiation, which is an operation computationally hard to invert. Thus, it is this operation that holds the security of this cryptosystem. The exponentiation of the point is obtained through the recurrently application of the group operation, which means that $P^n = P + P + \ldots + P$, with P a point and n a scalar. From this point onwards, this operation will be designated as point multiplication (nP). The provided security is determined by the difficulty of obtaining n, where only P and $Q(= nP)$ are known.

The above operations description over the set of real numbers is important to illustrate the concept. However, in real implementations, the field elements representation must be finite, thus the presented operations are adapted to finite fields in order to achieve the point multiplication as in Figure 2.2.

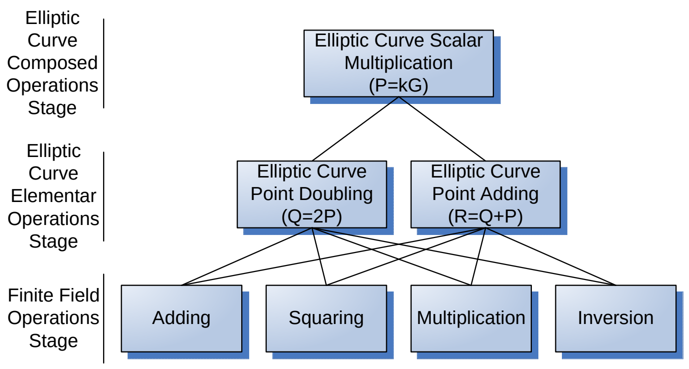
Elliptic Curve Composed Operations Stage
Elliptic Curve Elementar Operations Stage
Finite Field Operations Stage
Elliptic Curve Scalar Multiplication (P=kG)
Elliptic Curve Point Doubling (Q=2P)
Elliptic Curve Point Adding (R=Q+P)
Adding
Squaring
Multiplication
Inversion

Figure 2.2: Elliptic Curve Operations Hierarchy

2.3.1 Elliptic Curves over $GF(p)$

For this kind of fields, the EC description is the same as the one used for the real number field (2.10), except for the curve parameters a, b, which belong to $GF(p)$:

$$y^2 = x^3 + ax + b,\ \ a, b \in GF(p), \tag{2.14}$$

In $GF(p)$, (2.14) does not have solution for every value of x. It only has a solution if $\zeta = x^3 + ax + b$ is a quadratic residue modulo p, which means that it has a square root in $GF(p)$. This property can be tested using the *Legendre Symbol* $\left(\frac{\zeta}{p}\right)$ [18]:

$$\left(\frac{\zeta}{p}\right) \equiv \zeta^{(p-1)/2}\ mod\ p. \tag{2.15}$$

If (2.15) is 0, then p divides ζ and the coordinate y will be 0. If (2.15) is 1 then ζ is a quadratic residue and there are two different coordinates y that correspond to the coordinate x. Otherwise, (2.15) is -1 and ζ is not a quadratic residue, which means that there is no EC point with the correspondent x coordinate.

In these fields the geometrical interpretation vanishes, because of the modular operations. An EC example over $GF(p)$ is presented in Figure 2.3. However, since the EC representation is the same, the addition and doubling analytical expressions in $GF(p)$ are similar to the ones used for the real number field, except for the operations that are performed modulo p.

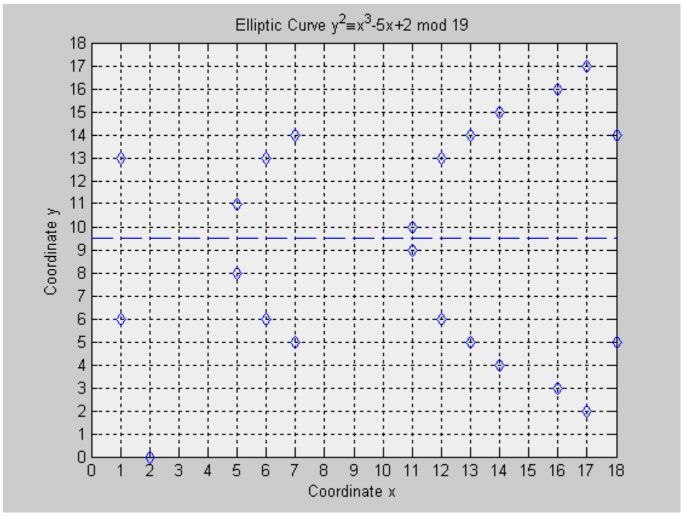

Figure 2.3: Elliptic Curve Example over $GF(p)$

The order of an EC is the number N of points into an EC or, in other words, the number of points that obey (2.14). This number has an upper and bottom bound given the Corollary sometimes called *Hasse's Theorem* [18]. This Corollary establishes that for an EC over $GF(p)$ the number of points N in this EC is delimited by:

$$p + 1 - 2\sqrt{p} \leq N \leq p + 1 + 2\sqrt{p}. \tag{2.16}$$

This result may be important to discuss the security of a cryptosystem based on these curves [18].

2.3.2 Elliptic Curves over $GF(2^k)$

As explained in Section 2.1.2, $GF(2^k)$ has more interesting properties than $GF(p)$ for hardware implementations. Since in this thesis the interest is hardware, it will be given more details about EC in $GF(2^k)$.

A different equation is used to describe the EC over $GF(2^k)$. This equation is derived from (2.9) with the appropriate coordinates change, and generate a so called non-supersingular curve, because the coefficient $a_1 \neq 0$. The obtained expression is:

$$y^2 + xy = x^3 + ax^2 + b, \; a, b \in GF(2^k). \tag{2.17}$$

Choosing a polynomial basis, the parameters a, b are polynomials of powers of x; it is worth to notice that the x coordinate in (2.17), can not get mixed up with the x term in the polynomials.

As in $GF(p)$, in $GF(2^k)$ not all the coordinates x correspond to EC points¡. To find a EC point from (2.17) it is necessary to introduce a field operand, called trace operand. If we have a field GF(2^k) and a polynomial $A(x)$ in this field, the trace of this polynomial is given by [13]:

$$T(A(x)) = \sum_{i=0}^{k-1} A(x)^{2^i}. \tag{2.18}$$

The trace function is known to assume two possible values: 0 or 1. It is also well known that the trace is a linear function, thus can be calculated as an inner product of the form:

$$T(A(x)) = \sum_{i=0}^{k-1} t_i a_i, \tag{2.19}$$

where t_i is obtained through:

$$t_i = T(\alpha_i) \tag{2.20}$$

where α_i represent the various elements of the basis that constructs the field $GF(2^k)$. In the case of a polynomial basis $\alpha_i = x^i$. The computation of the trace of a field element is more efficient for vectors t with fewer coefficients different from zero. A normal basis, for example, generate a vector t with all the coefficients different from zero, while the polynomial bases presents vectors t with few coefficients different from zero [13]. This means that the trace is more efficiently computed over polynomial basis. The trace of a polynomial also has the property $T(A(x)^2) = T(A(x))$.

With the trace operand introduced, it is possible to return to the computation of the EC points. By considering $z = y/x$, (2.17) can be rewritten as:

$$z^2 + z = x + a + bx^{-2} = g\left(x\right) \Leftrightarrow z^2 + z + g\left(x\right) = 0, \tag{2.21}$$

with $g(x) = x + a + bx^{-2}$. Applying the trace operator:

$$T(z^2) + T(z) + T(g(x)) = 0 \Leftrightarrow T(z) + T(z) + T(g(x)) = 0 \Leftrightarrow T(g(x)) = 0.$$

This is the condition that a coordinate x has to obey in order to create an EC point. In general, it is also the condition to a quadratic equation to be solvable. Moreover, if there is a solution z_1, there is another solution $z_2 = z_1 + 1$, which is equivalent to $y_2 = y_1 + x_1$, recalling $z_i = y_i/x_i$.

The following equalities hold for $GF(2^k)$ fields with k odd [19]:

$$\begin{gathered} g\left(x\right) = z^2 + z \\ g\left(x\right)^{2^2} = z^{2^3} + z^{2^2} \\ \vdots \\ g\left(x\right)^{2^{k-1}} = z^{2^k} + z^{2^{k-1}}. \end{gathered}$$

Recalling the property $z^{2^k} = z$, the sum of those equalities is equivalent to:

$$\sum_{i \in E} g\left(x\right)^{2^i} = T\left(z\right) + z, \quad E = \left\{0, 2, 4, \ldots, k-1\right\} \text{ and } T\left(z\right) \in \left\{0, 1\right\}. \tag{2.22}$$

Note that it is also possible to obtain a similar result in the form:

$$\sum_{i \in O} g\left(x\right)^{2^i} = T\left(z\right) + z, \quad O = \left\{1, 3, \ldots, k-2\right\} \text{ and } T\left(z\right) \in \left\{0, 1\right\}. \tag{2.23}$$

It is possible to obtain (2.23) from (2.22), and vice-versa, because it holds that:

$$T\left(g\left(x\right)\right)+\sum_{i\in E}g\left(x\right)^{2^{i}}=T\left(z\right)+z+0,\quad E=\left\{0,2,4,\ldots,k-1\right\}. \tag{2.24}$$

From (2.23) we can calculate the two solutions of (2.21) z_1 and z_2: one of them corresponds to $T(z)=0$, and the other to $T(z)=1$. From these results, it is possible to conclude that to code an EC point it is only necessary the coordinate x and the trace of y/x, because this later value allows to distinguish between the two possible solutions y for the same coordinate x. Summarizing, to compute an EC point the following steps may be taken:

- Calculate $g(x)=x+a+bx^{-2}$;
- Use (2.23) or (2.22), the trace of y/x and $g(x)$, to calculate y/x;
- Perform $y/x\times x=y$ to obtain the y coordinate.

In the case of $GF(2^k)$ the geometrical interpretation also vanishes. This result is justified by the definition of non-linear operations, such as the XOR operation to add field elements. In Figure 2.4 is depicted an EC example over $GF(2^k)$ with $k=5$, using the irreducible polynomial x^5+x^2+1. The used polynomials, including the EC parameters, are in hexadecimal notation. The analytical

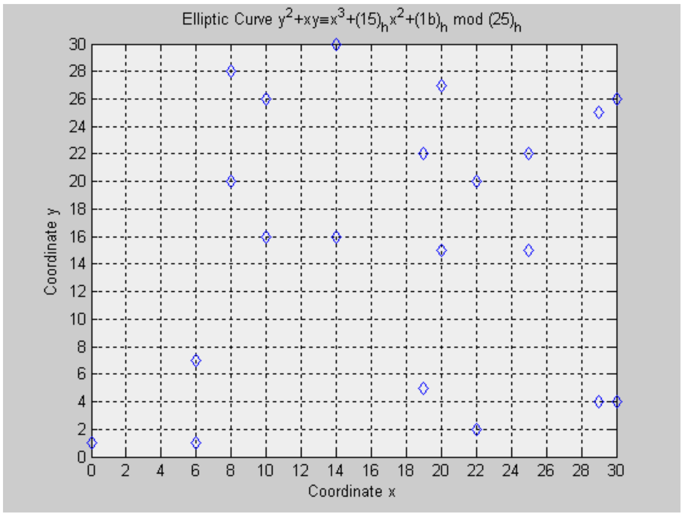

Figure 2.4: Elliptic Curve Example over $GF(2^k)$

expressions for these fields are different from the ones used for $GF(p)$, since the EC equation changed.

Considering two points $P_1=(x_1,y_1)$ and $P_2=(x_2,y_2)$ that obeys (2.17), if $x_1\neq 0$, $x_2\neq 0$ and $x_1\neq x_2$ it is obtained a point $P_3=P_1+P_2=(x_3,y_3)$ that obeys (2.17) with:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = \lambda\left(x_1 + x_3\right) + x_3 + y_1 \tag{2.25}$$
$$\lambda = \frac{(y_1 + y_2)}{(x_1 + x_2)}.$$

The addition inverse is defined for a point $P_1 = (x_1, y_1)$, as a point $P_2 = (x_2, y_2)$ such that $P_1 + P_2 = \mathcal{O}$. The coordinates of the addition inverse can be obtained by:

$$x_2 = x_1$$
$$y_2 = x_1 + y_1 \tag{2.26}$$

To add two points with the same coordinates, the doubling operation equations hold. Consider a point $P_1 = (x_1, y_1)$ that respects equation (2.17). If $P_1 \neq \mathcal{O}$, a point $P_2 = 2P_1 = (x_2, y_2)$ that respects (2.17) can be obtained with:

$$x_2 = \lambda^2 + \lambda + a$$
$$y_2 = x_1^2 + (\lambda + 1)\, x_2 \tag{2.27}$$
$$\lambda = x_1 + \frac{y_1}{x_1}$$

If $P_1 = \mathcal{O}$, then $P_2 = \mathcal{O}$.

To perform point multiplication, a sequence of doubling operations can be applied within conditional addition, which means that there will be only an addition operation for each bit different from zero in the scalar binary representation. The possible ways to perform point multiplication efficiently will be discussed in the Chapter 3.

2.4 Summary

In this section were introduced the fields and groups properties. The various operations over these fields were also introduced, and the most suitable fields to implement efficient cryptosystems on dedicated hardware have been identified. The field $GF(2^k)$ suggests better characteristics to this purpose, since it is supported over operations that can be efficiently implemented at the low level hardware with simple bit manipulations.

There were also introduced various forms of representing field elements. The polynomial basis showed to be efficient to achieve this representation. Furthermore, with this basis, some operands may be more efficiently computed, namely the trace operator, which is important to perform the computation of an EC coordinate, from the other element of a pair of coordinates. Since an optimal normal basis only exists for few fields $GF(2^k)$, a polynomial basis is a better choice if generality is a requirement. In fact, the proposed operations require odd k, but since the standards [15] for ECC use odd k, it is not a demerit factor. Therefore, a polynomial basis is adopted to take advantage of the original work herein proposed.

The EC properties supported over different fields were presented, specially for $GF(2^k)$. The operations performed over an EC were described, namely the point addition and doubling, and how these operations were constructed from the field operations. It was also introduced the complete hierarchy, from the field operation to the EC operation, in order to perform the EC point multiplication (or exponentiation), which is the support of an EC cryptographic system.

In the next chapter algorithms and architectures are discussed and proposed to efficiently perform operations over the finite field $GF(2^k)$ and over the groups supported by ECs.

3 Algorithms

Contents

This chapter presents the related art for the algorithms and architectures used in ECC, since the field arithmetic till the complete EC arithmetic. These algorithms are introduced by using some of the properties of the finite fields, particularly for the field $GF(2^k)$, and EC groups presented in the last chapter.

Different representations of field elements (polynomial and normal basis) are accounted and its advantages are exploited to suggest different algorithms. For each different possibility, the relative performance is analyzed in order to support the design options for the final EC processor suggested in this thesis. Some considerations about point representation are also made to support some of the original contributions in this thesis, namely minimal point representation, which are also introduced in this chapter.

3.1 Field Operations

This section introduces algorithms used to perform the operations over the finite field $GF(2^k)$. The high-level EC arithmetic is achieved by interconnecting these algorithms resulting units and by scheduling the operations in an appropriate way.

3.1.1 Field Multiplication

Two main classes of field multipliers can be identified: one of them includes the multipliers supported over optimal normal basis, which explore the properties of this basis to reduce the necessary dependencies of the output product from the input operands' bits; the other class includes multipliers for polynomial basis, exploring the maximum parallelism to improve performance.

The normal basis multipliers are called Massey-Omura multipliers. These multipliers have the property of applying exactly the same function for any output bit, changing only the input of this function. These multipliers can be efficiently implemented changing the function inputs shifting the input operands. To parallelize this algorithm one only needs to replicate this function. Details about this multiplier can be found in [20]. The complexity of this function depends on the normal basis used, and is minimal for special normal basis. These bases were discussed in Section 2.2.2 and are called optimal normal basis. The most used optimal normal basis is the type II optimal normal basis, because comparing with the type I optimal normal basis, there are more values of k for fields $GF(2^k)$ for which such basis exist [16].

Optimizations for the Massey-Omura multiplier are described in [6, 16]. These optimizations suggest the realization of a basis change to realize the multiplication. Starting from the normal basis, a permutation is made in order to obtain a new temporary base called canonical basis, which consists of a base generated by a field element α as $(\alpha, \ldots, \alpha^k)$. The multiplication can be performed over the basis obtained after the permutation and in the end the inverse permutation is performed. This optimization permits to save area resources [16].

One of the algorithms used to multiply over a polynomial basis is called Montgomery Modular Multiplier, which can be adapted to a finite field $GF(2^k)$ [21]. This kind of multiplier is suited to architectures which compute the operation through the division of the field elements representation for various equal sized words [22], as is the case of software solutions [21]. This algorithm consist on substituting the calculus $A(x)B(x)$ by the calculus $A(x)B(x)R(x)^{-1}$, with $R(x)^{-1}$ a special fixed element in $GF(2^k)$. This will lead to a more efficient implementation which, for 32 bits processors, represents a throughput about 20 times higher [21].

The Karatsuba-Offman multipliers are also presented in the literature to perform $GF(2^k)$ multiplications. This kind of multipliers allows to achieve a time complexity lower than $O(k^2)$ [23]. These multipliers require the division of the input operands ($A(x), B(x)$) into two parts (low and high) so they can be written as $A(x) = A(x)_H x^{k'/2} + A(x)_L$ and $B(x) = B(x)_H x^{k'/2} + B(x)_L$. Thus, the product ($C(x)$) is obtained as:

$$\begin{aligned} C(x) = {} & A(x)_H B(x)_H x^{k'} \\ & + \left[A(x)_H B(x)_H + A(x)_L B(x)_L + (A(x)_H + A(x)_L)(B(x)_H + B(x)_L)\right] x^{k'/2} \\ & + A(x)_L B(x)_L \end{aligned} \tag{3.1}$$

Nonetheless, for binary finite fields, the most efficient solutions do not divide the input polynomials at $k/2$. It is usual to rewrite $k = 2^n + r$, with n, r integers in order not to waste area resources, thus $k' = 2^n$. Thus a polynomial may be rewritten as $A(x) = A(x)_H x^{2^n} + A(x)_L$ [23]. The reason for the presented rewrite of k is that (3.1) can be applied several times, implementing the recursive Algorithm 3.1. The result for each algorithm step is calculated from the partial results obtained with the recursive call of the algorithm that progressively reduce the number of bits of the operands by half. This means that in each algorithm call the input polynomials order will be divided by 2, explaining the rewriting of k with a power of two and other integer r. This last integer can be represented with the power of two immediately above (2^m), but since it might be a small integer, the performance decrease is reduced. After using Algorithm 3.1, to compute the partial results from the two terms resulting from the rewrite of k (2^n and 2^m), these partial results are combined employing (3.1) once again. These multipliers may be very time efficient, but require a separated step to perform the reduction. This reduction step will be discussed later. Another disadvantage is that the rewriting of k imposes restrictions to the scalability of the multiplier for arbitrary fields, which means that this structure may be more or less efficient depending on how close the integer r is from 2^m.

Other method to perform the multiplication $A(x)B(x)$ follows directly from:

$$A(x)B(x) = \sum_{j=0}^{k-1} b_j x^j A(x). \tag{3.2}$$

The multiplication in (3.2) is supported on multiplications by powers of the polynomial x. The multiplication by polynomial x is described in (2.8). This procedure is very efficiently implemented,

Algorithm 3.1 Karatsuba Multiplication Algorithm $KM(A(x), B(x), n)$

```
Require: $A(x)$, $B(x)$ and $n$ ($A(x)$ and $B(x)$ number of bits);
Ensure: $M(x) = A(x)B(x)$;
  if $n = 1$ then
    return $a_0 b_0$
  end if
  {creating 3 pairs of $n/2$ bit sized polynomials}
  $A(x)_L := (a_{n/2-1} \dots a_0)$;
  $B(x)_L := (b_{n/2-1} \dots b_0)$;
  $A(x)_H := (a_{n-1} \dots a_{n/2})$;
  $B(x)_H := (b_{n-1} \dots b_{n/2})$;
  $A(x)_M := A(x)_L + A(x)_H$;
  $B(x)_M := B(x)_L + B(x)_H$;
  {recursive calling of multiplication obtaining partial results with $n$ bits}
  $R(x)_L := KM(A(x)_L, B(x)_L, n/2)$;
  $R(x)_M := KM(A(x)_M, B(x)_M, n/2)$;
  $R(x)_H := KM(A(x)_H, B(x)_H, n/2)$;
  {combining the partial results in order to obtain the final $2n$ bit sized result}
  $M(x) := R(x)_H x^n + (R(x)_H + R(x)_M + R(x)_L) x^{n/2} + R(x)_L$;
  return $M(x)$;
```

since the irreducible polynomials would have at most five non zero coefficients. Thus, a multiplication by the polynomial x can be computed with at most three XOR gates and hardwire shifts. To obtain other powers it is possible to serialize this procedure and since the irreducible polynomial is sparse there will not be a significant degradation of the time performance to compute higher powers. A parallelization of this method is proposed in [24], so (3.2) can be written for a two level parallelization and a field with odd k, as:

$$A(x)B(x) = \left[b_0 A(x) + b_2 x^2 A(x) + \ldots + b_{k-1} x^{k-1} A(x)\right] + \\ + x\left[b_1 A(x) + b_3 x^2 A(x) + \ldots + b_{k-2} x^{k-2} A(x)\right]. \quad (3.3)$$

The adoption of a polynomial basis discards any of the normal basis inspired multipliers discussed in Section 3.1.1, namely Massey-Omura multipliers. It is not of interest to consider multipliers constructed with a fixed word processor to obtain large polynomials multiplications when the target is an EC hardware processor. The reason to adopt polynomial basis is that it brings more interesting properties for computing important operations, namely the trace operator, which supports some of this thesis original work. From Section 3.1.1, one of the possible approaches is to use a Karatsuba-Offman multiplier. This multiplier architecture allows to more efficient area utilization when performing parallel optimizations. Although, it will be used a multiplier as suggested in [24], which, for a 2-level parallelization, is supported in (3.2). This multiplication is supported on the multiplication by powers of polynomial x, which can be efficiently implemented, as will be discussed in Section 4.1.1. Comparing with other multipliers, this implementation has the advantage of embed the reduction in the multiplication by polynomial x unit, avoiding an additional step of reduction. Other known advantages are the fact that can be easily adapted to any irreducible polynomial, affecting only the multiplication by polynomial x unit, and is easily scalable to any field size and any level of parallelization.

3.1.2 Field Squaring

With normal basis the squaring is a cyclic left shift. For polynomial basis, to compute a squaring, the dependencies of each output bit may be pre-computed from the input operand. This may be done regarding that the squaring is a particular case of field multiplication:

$$C(x) = A(x)^2 = \sum_{i=0}^{k-1}\sum_{i=0}^{k-1} a_i a_i x^i x^i = \sum_{i=0}^{k-1} a_i x^{2i}. \tag{3.4}$$

Observing (3.4), when $i < \lceil k/2 \rceil$ the c_i coefficient depends directly from the a_{2i} coefficients. For $i \geq \lceil i/2 \rceil$ the dependencies are calculated through the irreducible polynomial observing the property in (2.6). For these values of i, one shall compute x^{2i} using (2.8) and any of the multiplication algorithms discussed, and the dependencies arise from the result's coefficients which do not equal zero. Since the irreducible polynomial has at most five non zero coefficients, there will be few dependencies and (3.4) can be efficiently computed. An example will be presented to calculate a squaring operation modulo $x^5 + x^2 + 1$.

Table 3.1 lists the binary representation of all the first $2k-1 = 9$ powers of polynomial x modulo the irreducible polynomial $x^5 + x^2 + 1$. The input operand is described as (a_4, a_3, a_2, a_1, a_0). For

Table 3.1: Powers of polynomial x modulo $x^5 + x^2 + 1$

Polynomial	Binary representation
1	00001
x	00010
x^2	00100
x^3	01000
x^4	10000
x^5	00101
x^6	01010
x^7	10100
x^8	01101

example, to obtain the coefficient 3 of the reduced result, one may look the Table 3.1 even entries which have the coefficient 3 different from zero. In this particular case, these entries correspond to x^6, x^8. Thus, $c_3 = a_{6/2} + a_{8/2} = a_3 + a_4$, because any input coefficient a_i, as obtained in (3.4), influence the $2i - th$ power of the polynomial x in the result. The same procedure is repeated for the other coefficients.

The squaring operation is performed very efficiently in hardware solutions and its complexity is assumed to be similar to the addition operation. This method can be used to perform the reduction of an element not only for the squaring. For example, if there is a vector ($a_8, a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0$) to reduce, the coefficient c_2 of the reduction may be computed as $c_2 = a_8 + a_7 + a_5 + a_2$.

3.1.3 Field Inverters

There are two major methods to perform the inversion. One of them is based on the *Fermat Little Theorem* discussed in Section 2.2.2, which results in:

$$A^{-1} = A^{2^k-2} = A^2 A^{2^2} \dots A^{2^{k-1}}. \tag{3.5}$$

There are architectures that implement directly (3.5) [22]. However, there are other architectures that apply an algorithm inspired in (3.5) but with optimizations [25]. These later inverters are called Itoh and Tsujii inverters, and are supported on a rewritten version of (3.5), regarding that the squaring operation may be more efficiently computed than general multiplication:

$$A^{-1} = A^{2^k-2} = \left(A^{2^{k-1}-1}\right)^2 = \begin{cases} \left(A^{2^{(k-1)/2}-1}\right)^2 A & \text{if } k \text{ odd,} \\ \left[\left(A^{2^{(k-1)/2}-1}\right)^{2^{(k-1)/2}+1}\right]^2 A & \text{if } k \text{ even.} \end{cases} \tag{3.6}$$

From (3.6) follows the Algorithm 3.2. This algorithm requires $\lfloor log_2(k-1) \rfloor + h(k-1) + 1$

Algorithm 3.2 Itoh and Tsujii Inversion Algorithm

```
Require: $A(x) \in GF(2^k)$, $P(x)$ (irreducible polynomial);;
Ensure: $A(x)^{-1}$
  $s := log_2(n) - 1$;
  $result := A$;
  while $s \geq 0$ do
    $r := n >> s$; {right shift}
    $q := result$;
    for $i$ from $1$ to $r >> 1$ do
      $q := q^2 \; mod \; P(x)$;
    end for
    $t := result \times q \; mod \; P(x)$;
    if $r$ is odd then
      $t := t^2 \; mod \; P(x)$;
      $result := t \times A \; mod \; P(x)$;
    else
      $result := t$;
    end if
    $s := s - 1$;
  end while
  $result := result^2 \; mod \; P(x)$;
  return $result$;
```

multiplications [25], where $h(\cdot)$ is the binary Hamming weight. This result is better than the one obtained from the direct application of (3.5), which needs $k-2$ multiplications.

The other methods to perform the inversion are based on the *Extended Euclidean Algorithm* presented in Algorithm 2.1. In [17] it is proposed the Algorithm 3.3 which adapts the *Extended Euclidean Algorithm* to invert polynomials over $GF(2^k)$. This last algorithm is a better solution to dedicated implementations, being more time efficient [17]. There are parallel implementations of this algorithm that can be found in [24].

Algorithm 3.3 Brunner Inversion Algorithm:

```
Require: $A(x) \in GF(2^k)$ and $P(x)$ {irreducible polynomial};
Ensure: $A(x)^{-1}$;
  $F := P(x)$;
  $S := F$; {$S$ and $R$ are assumed to have the same degree of $P(x)$}
  $R := B(x)$; {$R$ is initialized with the input polynomial}
  $U := 1$; {maximum degree k-1}
  $V := 0$; {maximum degree k-1}
  $delta := 0$; {$delta$ represents the difference between $S$ and $R$ instant degrees}
  for $i = 1$ to $2k$ do
    if $r_m = 0$ then
      $R := xR$;
      $U := (xU)\ mod\ F$;
      $delta++$; {$R$ degree decreases}
    else
      if $s_m = 1$ then
        $S := S - R$;
        $V := (V - U)\ mod\ F$;
      end if
      $S := xS$; {$S$ degree decrease}
      if $delta = 0$ then
        {degree of $S$ inferior to $R$}
        $(R \leftrightarrow S)$; $(U \leftrightarrow V)$; {polynomials exchange}
        $U := (xU)\ mod\ F$;
        $delta++$;
      else
        $U := (U/x)\ mod\ F$;
        $delta--$;
      end if
    end if
  end for
```

In order to the *Itoh and Tsujii* method to be efficient, very fast multipliers have to be used. There is no possible parallelization of multiplications for this method, as it can be seen in the Algorithm 3.2, where the presented multiplications always have data dependencies from the previous one. The alternative method, the *Brunner Inversion Algorithm* presented in Algorithm 3.3 is well suited for hardware implementations and has a fixed duration of $2k$ iterations. This inverter has also other interesting properties: *i)* it can be parallelized [24], and *ii)* a divider can be implemented over the inversion algorithm differing only on an initialization step. Observing the Algorithm 3.3 to invert $B(x)$, if one wants to obtain $A(x)/B(x)$, the initialization step $U := 1$, may be substituted by $U := A(x)$ [17]. For these reasons, the inverter proposed for the processor presented in this thesis adopts the *Brunner Inversion Algorithm*.

3.2 Elliptic Curve Operations

In this section methods to perform the operation over an EC are discussed, not only the existent but also some alternatives that lead to more efficient implementations. Among these methods it is the utilization of different projective representations that allow to reduce the number of the most expensive field operation (the inversion) employed in point exponentiation. Integer recoding methods for reducing the number of required operations are also discussed. A protocol to communicate private data supported in the ECC is also presented.

3.2.1 Projective Coordinates

As discussed before, the field operations are four: addition, squaring, multiplication and inversion. The most costly of these four operations is by far the inversion operation. For a scalar with m bits, the presented point addition and doubling operations over an EC can be performed with $2\,(m-1)$ inversions. A change of coordinates may be used to avoid this amount of inversions, in order to increase system's performance. These new coordinates are called *projective coordinates*, and the original coordinates are called *affine coordinates*. To define the projective coordinates, consider the integers c, d and an equivalent class in $(GF(2^k))^3$ except for the set $(0,0,0)$ [10]:

$$(X_1,Y_1,Z_1) \sim (X_2,Y_2,Z_2)|\text{if } X_1 = \lambda^c X_2, Y_1 = \lambda^d Y_2, Z_1 = \lambda Z_2.$$

The following class is called a projective point:

$$(X:Y:Z) = (\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in GF(2^k),$$

and the tuple (X,Y,Z) is called a representative point. Considering $Z \neq 0$ and $\lambda Z = 1 \Leftrightarrow \lambda = 1/Z$, it is possible to take one of the representative points of the class $(X:Y:Z)$ as $(X/Z^c, Y/Z^d, 1)$. Therefore, for an appropriate λ, there is an isomorphism between the projective points $(X:Y:Z)$, with $X,Y,Z \in GF(2^k)$ and $Z \neq 0$ and the affine points defined by $x,y \in GF(2^k)$. Working with these projective coordinates, to perform a scalar multiplication the following steps must be taken:

1. Convert affine coordinates to projective coordinates, which may be a straight forward operation.

2. Perform the EC arithmetic in projective coordinates, free of inversions.

3. Convert the result to the original affine coordinates, which involve an inversion operation.

In summary, the great advantage of using projective coordinates is to perform only a final inversion on the last step of the processing corresponding to a point multiplication. This description may suggest that one should always work with projective coordinates, but there are some disadvantages of using this method. One of those is that the projective coordinates are three, versus the original two coordinates that can perform better bandwidth usage. Another disadvantage is that the amount of multiplication operations increases in the projective arithmetic. However, since it is guaranteed that the multiplication algorithm requires far less computing effort, projective coordinates arithmetic remains more efficient.

The used projective coordinates may be standard ($x = X/Z, y = Y/Z$) [23], Jabobian ($x = X/Z^2, y = Y/Z^3$) [26] or Lopez-Dahab ($x = X/Z, y = Y/Z^2$) [6, 25]. From this three representations, the most widely used are the standard and Lopez-Dahab.

3.2.2 Lopez-Dahab Projective Coordinates

Using the relation $x = X/Z, y = Y/Z^2$ the EC equation (2.17) can be rewritten as:

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4. \tag{3.7}$$

With the curve in (3.7), the doubling operation over the EC $Q = 2P$ with $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ is obtained as [6]:

$$\begin{aligned} Z_2 &= Z_1^2 X_1^2; \\ X_2 &= X_1^4 + bZ_1^4; \\ Y_2 &= bZ_1^4 + X_2(aZ_2 + Y_1^2 + bZ_1^4). \end{aligned} \tag{3.8}$$

The doubling operation is performed with four multiplications. The addition operation in these projective coordinates is in fact a mixed coordinates operation because, as it will be discussed later, in the point multiplication algorithm there is an input that is always invariant and can be represented with its affine coordinates as $(X_2, Y_2, 1)$. Thus, the addition of two points $R = P + Q$, with $R = (X_3, Y_3, Z_3)$, $Q = (X_2, Y_2, 1)$ and $P = (X_1, Y_1, Z_1)$, is performed as:

$$\begin{aligned} A &= Y_2 Z_1^2 + Y_1; \\ B &= X_2 X_1 + X_1; \\ C &= Z_1 B; \\ D &= B^2 (C + a Z_1); \\ Z_3 &= C^2; \\ E &= AC; \\ X_3 &= A^2 + D + E; \\ F &= X_3 + X_2 Z_3; \\ G &= X_3 + Y_2 Z_3^2; \\ Y_3 &= EF + Z_3 G. \end{aligned} \tag{3.9}$$

Therefore, the presented point addition takes ten multiplication operations to be perform.

3.2.3 Standard Projective Coordinates

Given the relationship $x = X/Z$ and $y = Y/Z$, the correspondent EC equation to (2.17) using standard projective coordinates becomes:

$$Y^2 Z + XYZ = X^3 + aX^2 Z + bZ^3 \tag{3.10}$$

An efficient way to use the equation (3.10) representation is called *Montgomery Method* and should not be confounded with the field multiplication *Montgomery Algorithm*. The following describes the *Montgomery Method/Algorithm*

Montgomery Method

Considering four points: $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_{add} = (x_{add}, y_{add}) = P_2 + P_1$ and $P_{sub} = (x_{sub}, y_{sub}) = P_2 - P_1$. Then it holds the following [27]:

$$x_{add} = \begin{cases} x_{sub} + \frac{x_1}{x_1 + x_2} + \left(\frac{x_1}{x_1 + x_2}\right)^2 & , P1 \neq P2 \\ x_1^2 + \frac{b}{x_1^2} & , P1 = P2 \end{cases} \tag{3.11}$$

If a standard projective representation is used, thus $x_{add} = X_{add}/Z_{add}$. Then it is possible to define the arithmetic in projective representation to add ($P1 \neq P2$) and double ($P1 = P2$) EC points, using the results in (3.11). Note that this arithmetic does not involve the y coordinate of the affine representation point, thus the resultant y coordinate may be only calculated when performing the final projective do affine coordinates conversion. The double and add operation is defined in projective coordinates as follows.

$$\begin{aligned} X_{double} &= X_1^4 + bZ_1^4, \\ Z_{double} &= X_1^2 Z_1^2 \end{aligned} \tag{3.12}$$

$$Z_{add} = (X_1 Z_2 + X_2 Z_1)^2 \,, \tag{3.13}$$
$$X_{add} = x_{sub} Z_{add} + X_1 Z_2 X_2 Z_1 .$$

Note that in the addition expression, the polynomial x_{sub} is one of the operands. When performing a scalar multiplication over an EC, this polynomial is invariant and corresponds to the original affine x coordinate of the point to be multiplied. The algorithm to compute a point multiplication by a scalar k with n bits using this method is the Algorithm 3.4 [27]. In Algorithm 3.4 the routine

Algorithm 3.4 Montgomery Point Multiplication Algorithm

```
Require: $k := (k_{n-1}k_{n-2}\dots k_0)$ with $k_{n-1} = 1$ and $P(x,y) \in E(GF(2^k))$;
Ensure: $Q := kP$;
  $X_1 := x$; $Z_1 := 1$; $X_2 := x^4 + b$; $Z_2 := x^2$;
  for $i = n-2$ downto $0$ do
    if $k_i = 1$ then
      ($X_1, Z_1$)=Madd($X_1, Z_1, X_2, Z_2$),($X_2, Z_2$)=Mdouble($X_2, Z_2$);
    else
      ($X_2, Z_2$)=Madd($X_2, Z_2, X_1, Z_1$),($X_1, Z_1$)=Mdouble($X_1, Z_1$);
    end if
  end for
  return $Q$ :=Mxy($X_1, Z_1, X_2, Z_2$);
```

`Madd` corresponds to the point addition, the routine `Mdouble` corresponds to the point doubling and the routine `Mxy` corresponds to the conversion to affine coordinates. In this last routine the results $x_R = X_1/Z_1$ and y_R are computed. An expression was presented in [27] to calculate y_R, which can be written in terms of X_1, Z_1, X_2, Z_2 as [10]:

$$y_R = (x + X_1/Z1)(xZ_1Z_2)^{-1}\left[(X_1 + xZ_1)(X_2 + xX_2) + (x^2 + y)(Z_1Z_2)\right] + y, \tag{3.14}$$

with y the algorithm input coordinate. The time this algorithm takes only depends on the scalar binary size, but not on the number of the binary coefficients different from zero, which is an advantage in the sense that it can not be time attacked. Even a differential power attack may be useless to attack this algorithm, since the amount of operations performed are the same for every scalar with the same size, thus the power is the same independently of the secret scalar.

This algorithm and point representation method is adopted in this thesis, because this algorithm has interesting properties to handle with a compact representation of EC points. These properties avoid the y coordinate to be used in most of the algorithm. Only in a final step, in the conversion from projective to affine coordinates, the y coordinate is required. This last step will be exploited in more detail in order to adapt it to the computation of a compact representation, as it will be introduced later.

3.2.4 Scalar Recoding

A scalar k with n bits can be written as [10]:

$$k = k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + \ldots + k_1 2 + k_0 = 2(2(\ldots 2(2k_{n-1} + k_{n-2}) + \ldots) + k_1) + k_0. \tag{3.15}$$

The point multiplication $Q = kP$ over an EC can be supported on the scalar written as in (3.15), using a basic double and add algorithm as Algorithm 3.5.

Algorithm 3.5 Double and Add Point Multiplication Algorithm

```
Require: $k := (k_{n-1}k_{n-2}\dots k_0)$ with $k_{n-1} = 1$ and $P(x,y) \in E(GF(2^k))$;
Ensure: $Q := kP$;
  $P := Q$;
  for $i = n-2$ downto $0$ do
    $Q := 2Q$;
    if $k_i = 1$ then
      $Q := Q + P$;
    end if
  end for
  return $Q$;
```

In the Algorithm 3.5, it is possible to observe that one of the terms of the point addition operation (P) is always the input operand, which means that this algorithm support the mixed coordinate algorithm explained in Section 3.2.2.

For improving the performance of the double and add algorithm, there may be used scalar recodings in order to reduce the Hamming weight of the scalar and to reduce the amount of addition operations. To do this, it will be admitted three possible coefficients to code the scalar (0, 1 and -1) and the Algorithm 3.5 may be adapted to perform a point subtraction, which can be easy since the point addition inverse is obtained with a simple k bits XOR operation (see (2.26)). One of the recoding techniques is supported on the identity [10]:

$$2^{i+j-1} + 2^{i+j-2} + \dots 2^i = 2^{i+j} - 2^i. \tag{3.16}$$

For example, the scalar $(01110)_b = 2^3 + 2^2 + 2^1$ may be recoded using (3.16) as $(1000-10)_b = 2^4 - 2^1$. Using this recoding the scalar size may increase from n to $n+1$.

There are other methods for recoding the scalar, but require pre-computed results and extra memory to store them, namely the Non-Adjacent Forms (NAF) recoding [10].

The using of the recoding has some weaknesses. All the algorithms that exploit the recoding can increase performance by avoiding some operations, thus the power consumption or time performance will be different depending on the scalar used on a particular point multiplication. One consequence is that the system would be more sensitive to differential power and time attacks. In this thesis design no recoding will be used, thus the power or time spent into a point multiplication will be the same independently of the scalar used.

3.2.5 Encrypting and Decrypting Messages

The hardware design may be optimized for the application, in what respects the EC arithmetic. This application may be a digital signature protocol or communication of messages with key exchange. When two entities wish to exchange data, it is usual to use public key cryptography to secretly change a symmetric key, which is then used to cipher or decipher the messages using an appropriate symmetric key algorithm, such as Advanced Encryption Standard (AES). This procedure is due to a better efficiency for symmetric cryptography, which offers a higher throughput with more attractive power and area characteristics.

ECC may be used for message cipher/decipher, namely to encrypt symmetric keys. The most common protocol that uses ECC to send messages is a protocol *El-Gamal* analog [18]. The *El-Gamal* protocol definition assumes the existence of a primitive element α which generates a prime field, as well as a secret a and an element β such that $\beta = \alpha^a$. The elements a and β are the receiver private and public keys, respectively. In this protocol, to transfer secure information a pair (γ_1, γ_2) is created, such that $\gamma_1 = \alpha^k$ and $\gamma_2 = m\beta^k$, where k is the sender private key and m is the information to be sent. The element γ_1 represents the sender public key. In order to obtain the message , the receiver has to compute $m = \gamma_2\gamma_1^{-a}$. It may be observed that $\gamma_1^{-a} = \alpha^{-ak}$ and $\gamma_2 = m\alpha^{ak}$, thus $\gamma_2\gamma_1^{-a} = m\alpha^{ak-ak} = m$.

The EC protocol has some differences from the generic description of the *El-Gamal* protocol. First of all, an EC cryptosystem is defined over a group and not over a prime field, thus the operation used for the exponentiation is different. Second, the operation to perform EC arithmetic is applied over EC points. Thus, it is assumed that, somehow, there is a mapping between message symbols or strings and EC points, and this mapping is bijective. This mapping may be performed by the sender as a trial algorithm [8], reserving a fixed part of the coordinate x to represent the message and the other part to represent a random redundancy such that the whole coordinate x is an EC point. The receiver just discards the random part and keeps the message.

It is assumed that there is a point G that is known by both parties, sender (A) and receiver (B). To perform a communication of a message from A to B, the procedure may be the following:

1. A and B choose a random integer, k_A and k_B, respectively. These integers are the private keys and both A and B only know about their own private keys.

2. A and B generate their public keys, computing $K_A = k_AG$ and $K_B = k_BG$, respectively.

3. A transmits its public key to B and B transmits its public key to A.

4. A maps its message into EC points M and computes (encrypt) an EC point to transmit as $C = M + k_AK_B$

5. B receive the EC point C and computes $C - k_BK_A = M$ (decrypt). This identity is true

because, with $K_A = k_A G$:

$$\begin{aligned} C - k_B K_A &= C - k_B\left(k_A G\right) = \\ &= C - \left(k_A k_B G\right) = \\ &= \left(M + k_A K_B\right) - \left(k_A k_B G\right) = \\ &= \left(M + k_A k_B G\right) - \left(k_A k_B G\right) = \\ &= M + \left(k_A k_B G\right) - \left(k_A k_B G\right) = \\ &= M. \end{aligned}$$

6. B does the reverse mapping of M and obtains the message.

To choose the random private keys it is important to bear in mind the concept of group order. The point G is called a generator and its powers generate a subgroup of EC points. This generator has a known order O_G. Thus, it must be assured that when the above operations are performed the random keys k that multiply G must obey $k < O_G$. This condition guaranties that the public key will not be $\mathcal{O}$ (the group identity) or a lower power of G that would be easily attacked, since multiply G by a scalar k is the same than multiply by $k \; mod \; O_G$. To avoid this situation, the scalar k employed usually has the same size n than O_G, which means that the most significant bits coincide, and the remaining $n-1$ bits assure that $k < O_G$.

These properties will be regarded for the real system application of the proposed design, in order to create and transmit public keys and perform a secure communication.

3.3 Minimal Elliptic Point Representation

In this section is introduced a method suggested in [13] that allows to code all the elliptic point information into a single coordinate, without the extra bit. This method will support some of the original contributions of the structure proposed in Section 4.1.

Attending that the EC equation over $GF(2^k)$ may be written as in (2.21), it is possible to rewrite the doubling point formula to the x coordinate in (2.27) as:

$$\begin{aligned} x_2 &= \left(x_1 + \frac{y_1}{x_1}\right)^2 + x_1 + \frac{y_1}{x_1} + a = \\ &= \left(\left(\frac{y_1}{x_1}\right)^2 + \frac{y_1}{x_1}\right) + x_1^2 + x_1 + a = \\ &= \left(x_1 + a + \frac{b}{x_1^2}\right) + x_1^2 + x_1 + a = x_1^2 + \frac{b}{x_1^2}. \end{aligned} \tag{3.17}$$

As referred in Section 2.3.2, to guarantee that a coordinate x belong to an EC point it must obey:

$$T\left(x_1 + a + \frac{b}{x_1^2}\right) = 0. \tag{3.18}$$

Regarding the linearity of the trace operator, then:

$$T\left(x_1 + \frac{b}{x_1^2}\right) = T\left(a\right). \tag{3.19}$$

Using the result in (3.17) it holds that:

$$T\left(x_2\right) = T\left(a\right). \tag{3.20}$$

In Section 3.2.5 was suggested the utilization of a generator point, which means that, in practice, the arithmetic over the EC will be performed over a subgroup generated by this generator point. In Section 2.1.1 were introduced some of the properties of subgroups, particularly of prime order subgroups, which are usually used in most of the EC protocols [28]. One of these properties for prime sized subgroups is that every point in such subgroup can be written as doubling of other one. Thus, the property in (3.20) is general for a prime order subgroup.

As described in Section 2.3.2, the trace operator can be written as an inner product with a trace vector. This means that only the coefficients in this vector different from zero intervene in the trace operator result. Since this trace vector can be pre-computed and is known at the time of the system setup, it is known which coefficients are these. In this case, one is free to arbitrate one of these coefficients, because the condition in (3.20) will always allow to correct this one coefficient by scanning the other ones.

As an example, in $GF(2^5)$ supported with the irreducible polynomial $x^5 + x^2 + 1$, the trace vector is $(01001)_b$. Considering that the parameter a of the EC is $(10101)_b$, then $T(a) = 1$. Now, considering the EC point $(x, y) = (19, 5) = ((10011)_b, (00101)_b)$ picked from Figure 2.4, one can compute $y/x = (00110)_b$, thus $T(y/x) = 0$. With this information the picked point can be coded as $(1001T(y/x))_b = (10010)_b$.

To decode $(10010)_b$, from the first coefficient one reads $T(y/x) = 0$ and the x coordinate become $(1001\xi)_b$. Then, scanning the trace vector, besides the coefficient 0, only the coefficient 3 is involved in the trace calculation. Since, looking at $(1001\xi)_b$ the third coordinate is 0. Thus, $T(a) = 0 + \xi$, which is the same than $\xi = 0 + T(a) = 1$. The coordinate is now corrected $(10011)_b$. To compute y it may be used (2.23) and $T(y/x)$ to obtain y/x and, finally, a last multiplication by x results in the y coordinate.

These methods will be called from now on *coordinate collapsing* and *coordinate uncollapsing*.

3.4 Efficient Units to Deal with Coordinate Collapsing

This section presents original contributions proposed to improve the computation of EC arithmetic efficiency, starting from a minimal representation of the EC points which leads to a more efficient bandwidth resources utilization. This detail in not often taken into account on the related art implementations of EC cryptosystems.

In Section 2.3.2 it was stated that it is possible to code an EC point with the coordinate x and one more bit which corresponds to $T(y/x)$. This additional bit distinguish the two possible coordinates y which correspond to the coordinate x. It was also discussed how this information can be used to compute the y coordinate. This last point is related to the resolution of a quadratic equation over $GF(2^k)$. In this section it is proposed a method to efficiently use the collapsed representation in point multiplication and a unit to compute y/x from x and the trace of y/x, by using (2.23).

3.4.1 Point Multiplication

The Algorithm 3.4 allows to efficiently multiply EC points using only the coordinate x. This algorithm permits to obtain the product's x coordinate using only the input x coordinate. The routine `Mxy` is the only one that requires the y coordinate. This routine is supported by (3.14). In [10] a schedule to compute `Mxy` is suggested. This schedule is presented in the Algorithm 3.6. The schedule in Algorithm 3.6 requires 9 field multiplication and 1 inversion.

Algorithm 3.6 `Mxy` routine

```
Require: $P = (X_1, Z_1)$, $Q = (X_2, Z_2)$ and $(x, y) \in E(GF(2^k))$ {starting affine coordinate};
Ensure: $(x_R, y_R)$ {result affine coordinates};
 1: $\lambda_1 = Z_1 Z_2$;
 2: $\lambda_2 = Z_1 x$;
 3: $\lambda_3 = \lambda_2 + X_1$;
 4: $\lambda_4 = Z_2 x$;
 5: $\lambda_5 = \lambda_4 + X_1$;
 6: $\lambda_6 = \lambda_4 + X_2$;
 7: $\lambda_7 = \lambda_3 \lambda_6$;
 8: $\lambda_8 = x^2 + y$;
 9: $\lambda_9 = \lambda_1 \lambda_8$;
10: $\lambda_{10} = \lambda_7 + \lambda_9$;
11: $\lambda_{11} = x \lambda_1$;
12: $\lambda_{12} = \lambda_{11}^{-1}$;
13: $\lambda_{13} = \lambda_{12} \lambda_{10}$;
14: $x_R = \lambda_{14} = \lambda_5 \lambda_{12}$;
15: $\lambda_{15} = \lambda_{14} + x$;
16: $\lambda_{16} = \lambda_{15} \lambda_{13}$;
17: $y_R = \lambda_{16} + y$;
18: return $(x_R, y_R)$;
```

In order to efficiently use the compact representation presented in Section 3.3, it is proposed a manipulation of (3.14), in order to `Mxy` calculate the result x_R and $T(y_R/x_R)$ from X_1, Z_1, X_2, Y_2

and $T(y/x)$, thus without having to calculate y or y_R:

$$\begin{aligned} y_R &= (x + X_1/Z1)(xZ_1Z_2)^{-1}\left[(X_1 + xZ_1)(X_2 + xX_2) + (x^2 + y)(Z_1Z_2)\right] + y \Leftrightarrow \\ \Leftrightarrow y_R &= \frac{(X_1 + xZ_1)(X_2 + xZ_2)}{Z_1Z_2} + (x^2 + y) + \frac{x_R(X_1 + xZ_1)(X_2 + xZ_2)}{xZ_1Z_2} + \frac{x_R(x^2 + y)}{x} + y \Leftrightarrow \\ \Leftrightarrow \frac{y_R}{x_R} &= (X_1 + xZ_1)(X_2 + xZ_2)\left(\frac{1}{x_RZ_1Z_2} + \frac{1}{xZ_1Z_2}\right) + \frac{x^2}{x_R} + \frac{y}{x_R} + \frac{x^2}{x} + \frac{y}{x} + \frac{y}{x_R} \Leftrightarrow \\ \Leftrightarrow T\left(\frac{y_R}{x_R}\right) &= T\left(\frac{(X_1 + xZ_1)(X_2 + xZ_2)}{Z_1Z_2}\left(\frac{1}{x_R} + \frac{1}{x}\right) + \frac{x^2}{x_R}\right) + T(x) + T\left(\frac{y}{x}\right). \end{aligned} \tag{3.21}$$

From (3.21) one can conclude that it is possible to compute $T(y_R/x_R)$ of a point multiplication from $T(y/x)$ and terms that depend only on x. Considering $x_R = X_1/Z_1$, is proposed a final manipulation of (3.21):

$$T\left(\frac{y_R}{x_R}\right) = T\left(\frac{y}{x}\right) + T\left(\frac{(xZ_1 + X_1)\,(x(X_1Z_2 + X_2Z_1) + X_1X_2)}{xX_1Z_1Z_2}\right) \tag{3.22}$$

The schedule presented in Algorithm 3.7 is proposed to compute the expression in (3.22). Comparing

Algorithm 3.7 Proposed `Mxy` routine

```
Require: $P = (X_1, Z_1)$, $Q = (X_2, Z_2)$, $x \in GF(2^k)$ and $T(y/x) \in GF(2)$;
Ensure: $x_R$ and $T(y_R/x_R)$; {collapsed representation of the result affine coordinates}
 1: $A_{x2} = xZ_1$;
 2: $A_{x3} = X_2Z_1 + (A_{x1} = X_2Z_2)$;
 3: $A_{x1} = A_{x2}A_{x1}$;
 4: $A_{x3} = A_{x3}x + X_1X_2$;
 5: $X_2 = A_{x2} + X_1$;
 6: $A_{x1} = A_{x1}^{-1}$;
 7: $A_{x3} = X_2A_{x3}$;
 8: $A_{x2} = xZ_2$;
 9: $X_1 = A_{x2}X_1^2$;
10: $x_R = X_1A_{x1}$;
11: $T(y_R/x_R) = T(A_{x3}A_{x1}) + T(y/x)$;
12: return $(x_R, T(y_R/x_R))$;
```

the proposed Algorithm 3.7 with the Algorithm 3.6, one may notice that it needs one more multiplication. However, in the proposed algorithm there is no data dependencies on steps `7)` to `9)` from step `6)`, thus a parallelization may be performed. Moreover, the trace of a multiplication can be obtained without computing the product, but only the bits that define the trace. For this reason the trace of a multiplication is computed more efficiently than the multiplication operation. The construction of an unit that calculate the trace of a multiplication will be introduced later. The number of inversions is the same in both proposed related art solutions. These considerations guarantee that adapting the algorithm `Mxy` to work with a collapsed representation does not increase its complexity.

3.4.2 Point Addition

For point addition was not found an efficient employ of the collapsed representation. In other words, employing the collapsed representation brings no complexity compensation comparing with the basic addition algorithm supported by (2.25) and the y coordinate calculus. For these reasons, to perform addition it is proposed a method which permits to efficiently compute the y coordinate. This method is supported by (2.23). It would also be possible to use (2.22), however, (2.22) has one more term, thus computing (2.23) is more efficient.

To implement (2.23) over $GF(2^k)$ a parallelization is possible rewriting (2.23) as:

$$\sum_{i \in O} \left(g\left(x\right)^{2^i} + g\left(x\right)^{2^{i+2}} \right) = T\left(z\right) + z, \quad O = \{1, 5, 9, 13, \ldots, n-4, n\} \tag{3.23}$$

with $n < k$ and $T\left(z\right) \in \{0, 1\}$.

3.5 Summary

In this chapter the related art algorithms to perform the operations over $GF(2^k)$ were presented, and were discussed some of their characteristics. The different methods to multiplying are discussed, regarding the basis used to support the field. The squaring operation was shown to be efficiently computed on hardware platforms, both in polynomial basis and normal basis. Two kinds of inverters were presented, one supported on recurrent use of the multiplication and other supported on the *Extended Euclidean Algorithm*. The inversion algorithm was shown to be less time efficient than the multiplication, remaining the hardest field operation to compute.

The EC basic algorithms were also introduced. There were discussed different ways of representing the EC group's elements, namely the different projective coordinates, in order to improve performance by reducing the use of field inversions. For these different representations, different algorithms, their properties, and the achieved performance improvement of point multiplication were presented.

Techniques to improve point multiplication regarding a scalar recoding as well as a notion on how to cipher and decipher messages using EC operations, were also presented.

A method was introduced to collapse the representation of an EC point in order to save bandwidth in the EC information transmission. Furthermore, original algorithms to deal with this collapsed representation were also introduced. The utilization of these algorithms suggest better solutions in terms of bandwidth saving and new functionalities.

In the following chapter, the algorithms herein presented will be adopted in order to create a reconfigurable logic platform, based on FPGA technology.

4 Proposed Design and Results

Contents

In this chapter a full cryptographic processor supported over EC arithmetic is presented. The proposed structure was targeted for a FPGA technology. Some of the previous discussed field and groups properties, as well as some algorithms, are considered to efficiently perform the required EC operations. New units resulting from the original algorithms related to the minimization of EC points representation and its use are also introduced. This chapter also presents the implementation results of the proposed EC design in a FPGA technology and the comparison with the related state of the art. In order to fully evaluate the proposed structure, an ASIC technology was also used as an implementation target. Since the main target is the FPGA implementation, more attention will be given to the implementation on this technology.

4.1 Proposed Elliptic Curve Processor Design

This section describes the processor proposed to compute the EC arithmetic, starting with the description of the field operation units and finalizing with the description of the top level EC operations, namely the point multiplication and point addition. The proposed design is focused in a particular $GF(2^k)$ field with $k = 163$, supported by the irreducible polynomial $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$. These field parameters are suggested by NIST [15]. Nevertheless, this design can be easily generalized to any odd k and any irreducible polynomial.

4.1.1 Multiplication and Division by polynomial x

The multiplication and division by polynomial x is important to support the field operations, namely the multiplication and inversion. These operations are supported by (2.8). The multiplication by polynomial x is computed with a hardwired left shift followed by the reduction of the resulting order k coefficient. To perform this reduction the irreducible polynomial can be added with the shifted input if this coefficient is different from zero, otherwise the left shift is sufficient. The resulting unit is depicted in Figure 4.1.

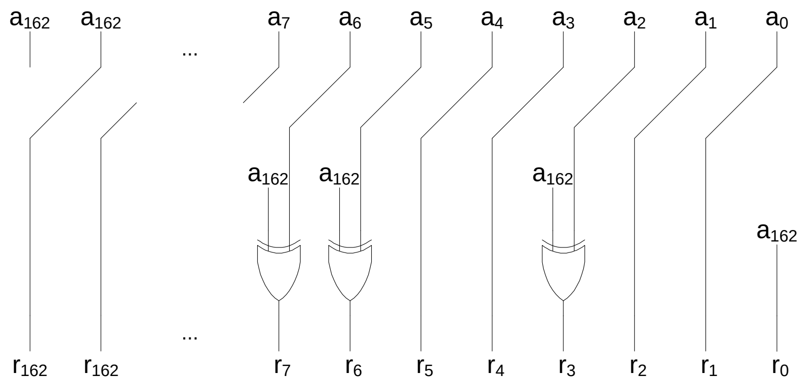

Figure 4.1: Multiplication by polynomial x unit

Since the irreducible polynomial is sparse (usually a pentanomial or trinomial), this unit can be constructed with very few logic gates. For the presented implementation, using a pentanomial, the number of logic gates is 3. To divide by the polynomial x, the inverse can be computed. Observing that the coefficient 0 of the irreducible polynomial must be different from the value 0, if the input coefficient 0 is different from 0 a XOR operation with the irreducible polynomial can be performed (undo a reduction), and a right shift completes the calculation. If the input coefficient 0 is 0, only the right shift is required. The division by polynomial x unit is depicted in Figure 4.2.

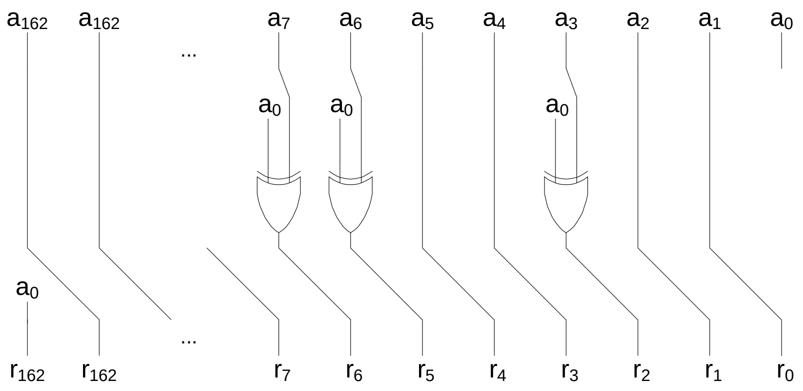

Figure 4.2: Division by polynomial x unit

It is possible to serialize the multiplication and division by polynomial x structures, in order to obtain multiplication and division by any power of the polynomial x. These structures are very area efficient, allowing multiplications and divisions by powers of polynomial x to be designed with minimal area complexity. The hardwire shift operations also limit the number of serial XOR gates in the critical path, thus performance efficient units can be obtained.

4.1.2 Field Multiplication

The proposed multiplication unit, depicted in Figure 4.3, is obtained from the parallelization of the equation (3.2) in four subtasks [24]:

$$\begin{aligned}
Z_0(x) &= b_0A(x) + b_4x^4A(x) + \ldots + b_{160}x^{160}A(x); \\
Z_1(x) &= x\left[b_1A(x) + b_5x^4A(x) + \ldots + b_{161}x^{160}A(x)\right]; \\
Z_2(x) &= x^2\left[b_2A(x) + b_6x^4A(x) + \ldots + b_{162}x^{160}A(x)\right]; \\
Z_3(x) &= x^3\left[b_3A(x) + b_7x^4A(x) + \ldots + b_{159}x^{156}A(x)\right];
\end{aligned} \tag{4.1}$$

with $Z(x) = Z_0(x) + Z_1(x) + Z_2(x) + Z_3(x)$. In order to achieve a good compromise between the available area and time performance, a 4 level parallelization is used. This structure can be further parallelized in order to achieve higher performances. No degradation of the critical path is expected when increasing the parallelization level [24], due to the sparse irreducible polynomials.

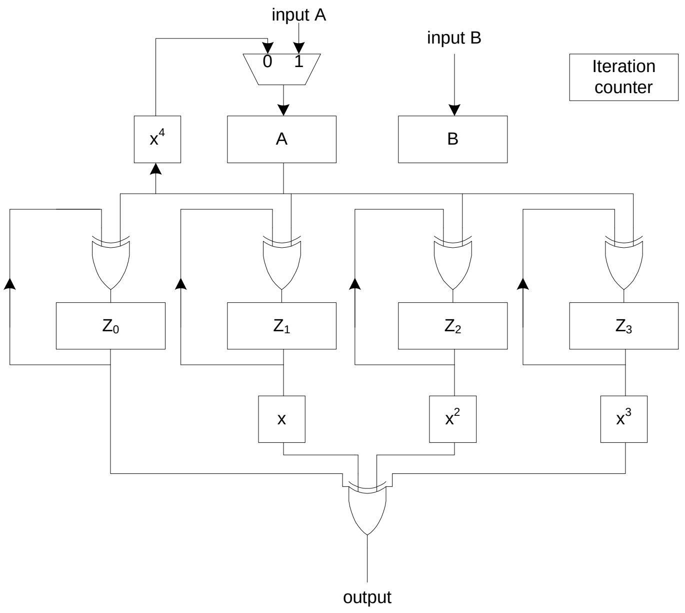

Figure 4.3: Proposed Field Multiplication Unit

The counter controls the iterations needed to complete the multiplication, which in this case is $\lceil k/4 \rceil = 41$. In the last iteration the register Z_3 is not updated, which follows directly from Z_3 in (4.1). In conclusion, considering that multiplying by the polynomial x^i takes $3i$ XOR operations, the proposed multiplier can be constructed with:

- 6 $\times$ 163 bit registers;
- 4 $\times$ 163 bit, 2 input XOR gates;
- 1 $\times$ 163 bit, 4 input XOR gate;
- 30 $\times$ 1 bit, 2 input XOR gates (multiplication by x^i);
- 1 $\times$ 163 bit, 2:1 multiplexer;
- 1 $\times$ 6 bit counter.

4.1.3 Squaring Unit

The squaring unit has a very efficient implementation in hardware platforms and is computed as described in Section 3.1.2. This implementation is performed with a pre-calculated table which translates the dependencies of each output bit from the input operand, regarding the even powers of the polynomial x. The complexity of this operation is greater for irreducible polynomials with higher Hamming weight. Using trinomials or pentanomials (as in the presented implementation) this complexity is small. Using the field size and irreducible polynomial for the implementation herein proposed, it is possible to obtain a $GF(2^{163})$ equivalent table for the example over $GF(2^5)$ presented in Table 3.1. The obtained dependencies can be calculated as presented in the $GF(2^5)$ example. The difference is that the table dependencies are spread over 2×163 rows instead of

2×5 rows as in the example. The number of dependencies per output coefficient are 5 for 2 output coefficients, 4 for 2 output coefficients, 3 for 79 output coefficients, and, finally, 2 dependencies for 80 output coefficients. For example, in the proposed implementation, the even powers of the polynomial x which have the coefficient 10 different from zero are: 10, 166, 170, 322, 324. Thus, the resulting coefficient 10 of the squaring of $A = (a_{163}, \ldots, a_1, a_0)$ is $c_{10} = a_{10/2} + a_{166/2} + a_{170/2} + a_{322/2} + a_{324/2}$ or $c_{10} = a_5 + a_{83} + a_{85} + a_{161} + a_{162}$. In sum, the squaring unit is constituted by:

- 2 $\times$ 5 input XOR gates;
- 2 $\times$ 4 input XOR gates;
- 79 $\times$ 3 input XOR gates;
- 80 $\times$ 3 input XOR gates.

4.1.4 Trace of a Multiplication

Calculating the trace of a multiplication can be performed more efficiently than calculating the multiplication and obtain the trace from the product. This observation is due to the properties of the trace vector calculated as in (2.20). This trace vector, if polynomial basis is being used, may be sparse. In fact, for the particular field properties used in this implementation, this trace vector has only two non-zero coefficients. These coefficients are 0 and 157, which means that if one wants to calculate the trace of a polynomial $A = (a_{163}, \ldots, a_1, a_0)$, a simple XOR operation is enough:

$$T(A) = a_0 + a_{157}. \tag{4.2}$$

The result in (4.2) suggests that to calculate the multiplication trace, only two bits of the product are needed. The multiplication operation may be computed as:

$$A(x)B(x) = \sum_{i=0}^{k-1} a_i x^i \sum_{j=0}^{k-1} b_j x^j = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} a_i b_j x^{i+j}. \tag{4.3}$$

Generating a table of the $2k$ first powers of the polynomial x:

- The powers with non-zero coefficient 0 are: 0, 163, 319, 320, 323;
- The powers with non-zero coefficient 157 are: 157, 313, 314, 317, 320.

Regarding (4.3), the trace of a multiplication can be computed by the addition (XOR) of all the terms $a_i b_j$ such that:

$$(i + j) = \{0, 157, 163, 313, 314, 317, 319, 323\}. \tag{4.4}$$

Note that the terms such that $(i+j) = 320$ will cancel, thus are not considered. Using the previous information, the trace of a multiplication can be computed as:

$$T(A \times B) = a_0b_0 + \sum_{i=0}^{157} a_ib_{(157-i)} + \sum_{i=1}^{162} a_ib_{(163-i)} + \\ + \sum_{i=151}^{162} a_ib_{(313-i)} + \sum_{i=152}^{162} a_ib_{(314-i)} + \sum_{i=155}^{162} a_ib_{(317-i)} + \\ + \sum_{i=157}^{162} a_ib_{(319-i)} + \sum_{i=161}^{162} a_ib_{(323-i)}. \tag{4.5}$$

The computation of the trace of a multiplication requires:

-360 $\times$ 2 input AND gates;

- 1 $\times$ 359 input XOR gate.

This unit is efficient when there are few dependencies of the trace operator on the input coefficients. It would not be the case, thus this unit should be replaced by a multiplier followed by the computation of the trace directly from the product.

4.1.5 Field Division

As introduced in Section 3.1.3, the number of multiplications required by the *Itoh-Tsujii* method to invert a polynomial depends only on the size of the field, and is given by $\lfloor log_2(k-1) \rfloor + h(k-1)+1$ [25]. For the presented implementation the number of multiplications would be $\lfloor log_2(162) \rfloor + h(162) + 1 = 11$. Particularizing for the presented implementation, the adopted *Brunner* inversion method, requires $2k = 326$ iterations to perform an inversion (or division) of a polynomial. Using the multiplier suggested in Section 4.1.2, an *Itoh-Tsujii* inverter would take $11 \times 41 = 451$ iterations, which is more than 326. Thus, the time performance of the *Brunner* inverter, for this particular implementation, is another advantage to add to those introduced in Section 3.1.3.

For the inverter (and divider) the architecture presented in Figure 4.4 is used, which result from the direct application of the Algorithm 3.3, without parallelization.

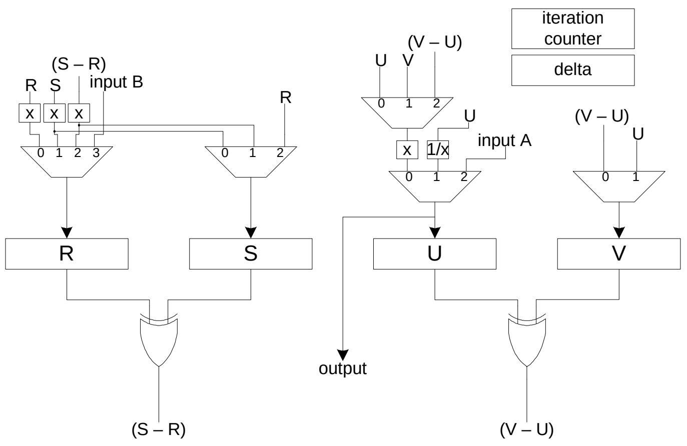

Figure 4.4: Proposed Field Inverter/Divider

Note that the multiplications by the polynomial x, which involve the R and S registers, are not performed $mod\ P(x)$, with $P(x)$ the irreducible polynomial. This operation corresponds to one bit hardwire left shift operation. The iteration counter controls the number of iterations needed to perform an inversion. When the unit starts functioning, an initialization step is performed, where the register U is accordingly loaded or reset, depending on if there will be a division or an inversion, respectively. The control for each iteration is described in Table 4.1, where the bits r_{163} and s_{163} are the most significant bits of the registers R and S, respectively, and the $delta$ counter quantify the difference between the degrees of the polynomials in registers R and S as in Algorithm 3.3.

Table 4.1: Inverter Control

Signals			Registers				Counter
r_{163}	s_{163}	$delta = 0$	R	S	U	V	$delta$
0	$\times$	$\times$	xR	S	xU	V	$delta++$
1	0	0	xS	R	xV	U	$delta++$
1	0	1	R	xS	U/x	V	$delta--$
1	1	0	$x(S-R)$	R	$x(V-U)$	U	$delta++$
1	1	1	R	$x(S-R)$	U/x	$(V-U)$	$delta--$

The proposed division algorithm has the property that the control behavior does not depend on U and V registers, which means that the control does not depend on the dividend. This allows to perform various divisions by the same divisor using the same circuit, replicating only the U and V associated logic and using the same control for all the replicas.

4.1.6 Root Calculation Unit

In order to perform point addition efficiently starting from the collapsed representation of an EC point, an unit to compute y/x from $g(x) = x + a + bx^{-2}$ and $T(y/x)$ is proposed. To obtain the y coordinate one may multiply this unit result by x. This unit computes (3.23), for which needs $\lfloor k/4+1 \rfloor$ or $\lfloor k/4 \rfloor$ iterations, depending on $k\ mod\ 4 = 3$ or $k\ mod\ 4 = 1$, respectively. These conditions define if there is an odd or even number of terms in (2.23), respectively. The proposed unit is depicted in Figure 4.5.

The units w^i compute the $i-th$ power of its input. Initially, the $g(x)$ and $T(y/x)$ registers are loaded with the respective values, and the register $Temp$ is reset. With this level of parallelization, in each iteration the eighth power of the previous result $g(x)$ is computed, which means that in each iteration the previous iteration result is added with two new values. The *control* signal implements a conditional addition operation with the associated term. This is useful if the design is to be implemented for fields $GF(2^k)$ such that $k \equiv 3\ mod\ 4$, which means that, excluding the term $T(y/x)$, there is an odd number of terms in (2.23). The presented counter controls the number of iterations needed. In the last iteration the output is valid and contains the 1 bit XOR operation with $T(y/x)$. After this unit operation to take place, the multiplication of its result (y/x)

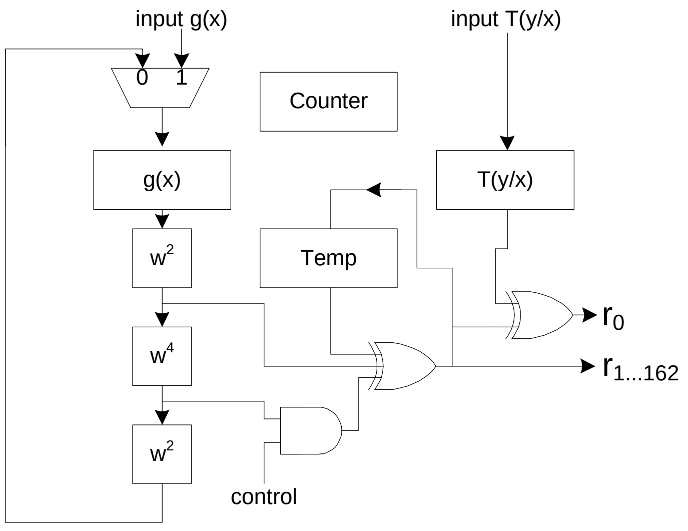

Figure 4.5: Root Calculation Unit

with x computes the y coordinate.

4.1.7 Control Considerations

The Finite State Machine (FSM) used to control the field units, namely multiplication and inversion, are Mealy FSM. This means that the control depends on the unit inputs. This property permits to load the input operands in the same clock cycle when an input *start* command rises, reducing the unit latency.

In particular for the field multiplication unit discussed in Section 4.1.2, there is an extra clock cycle for the output become available, related to the combination of the four parallel results, which is referenced with a FSM state. This means that if the *start* command is always active, the number of clock cycles needed to perform multiplication are the 41 discussed in Section 4.1.2, 1 related with the combination of the parallel results, and another one related to the transition from this last FSM state to an idle state. With the FSM in the idle state, a new multiplication procedure may immediately begin. Summarizing, the multiplication takes 43 clock cycles to perform.

The inversion control of the unit discussed in Section 4.1.5 has different characteristics. In this unit, the output is available at the time of the last iteration and return immediately to an idle state. This means that there will exist one extra latency cycle resulting from the transition for the idle state. For this reason the number of clock cycles needed to perform inversion/division are 327, instead of the 326 discussed in Section 4.1.5.

The root calculation unit suggested in Figure 4.5 has the same characteristics of the field inversion. Thus, regarding that $163 \ mod \ 4 = 3$, the number of clock cycles needed to perform is 42, instead of the $\lfloor 163/4 + 1 \rfloor = 41$ clock cycles introduced in Section 3.4.2.

4.1.8 Complete Elliptic Curve Processor

In this section the complete EC processor is described. Its construction is supported by the units presented in the previous sections. The two operations over the EC to be implemented are the point multiplication and point addition. The first one is the operation used to obtain public keys from private keys, and distributed secrets from one party public key and the other party private key. The last one is the operation used to support the encrypting and decrypting of messages explained in Section 3.2.5.

The point multiplication is the most complex operation, thus the optimization priority is focused in this operation. The Algorithm 3.4 was adopted to compute point multiplication, upgraded with the point collapsing considerations discussed in Section 3.4.1, which resulted into a new routine `Mxy`. Two basic operations are needed to perform this algorithm, which are the projective addition (3.13) and the projective doubling (3.12). The obtained structure is presented in Figure 4.6, using the following units:

- w^2: unit that computes the squaring of a polynomial, as explained in Section 4.1.3;
- *mult i*: unit that computes the field multiplication of two polynomial, as introduced in Section 4.1.2;
- *div*: unit that computes field inversion and division, as explained in Section 4.1.5;
- *Trace of multiplication*: unit that computes the trace of multiplication without calculate the product, as explained in Section 4.1.4;
- *Root Calculation*: unit that calculates y/x from x and $T(y/x)$ using (3.23).

The parameters a, b are the EC parameters used in the analytical description in (2.17). The parameter d is a pre-computed polynomial such that $d^4 = b$. This last parameter can be computed using the *Fermat Little Theorem*, which states that for any field element β, it holds that $\beta = \beta^{2^m}$ [17]. Thus, $d = b^{2^m-4}$, with $2^m - 4 = 2^2 + 2^3 + \ldots + 2^{m-1}$. To perform point multiplication the scalar is assumed to have the same size than the order of the generator point used. The register that loads this scalar, is a left shift register, enabled in each iteration of the point multiplication algorithm. The most significant bit from this register will be denoted as s. The scheduling of the operations for the proposed architecture to follow in order to perform point multiplication is presented in Table 4.2.

Observing the scheduling in Table 4.2 and regarding the algorithm used for point multiplication, the point multiplication initialization step takes one clock cycle.

The point multiplication main cycle takes the time of two field multiplications which is $2\times 43 = 86$ clock cycles. The point multiplication main cycle requires $86 \times \lfloor log(O_G) \rfloor$ clock cycles, for a point generator with order O_G. This cycle contributes with the greater amount of spent time. Since

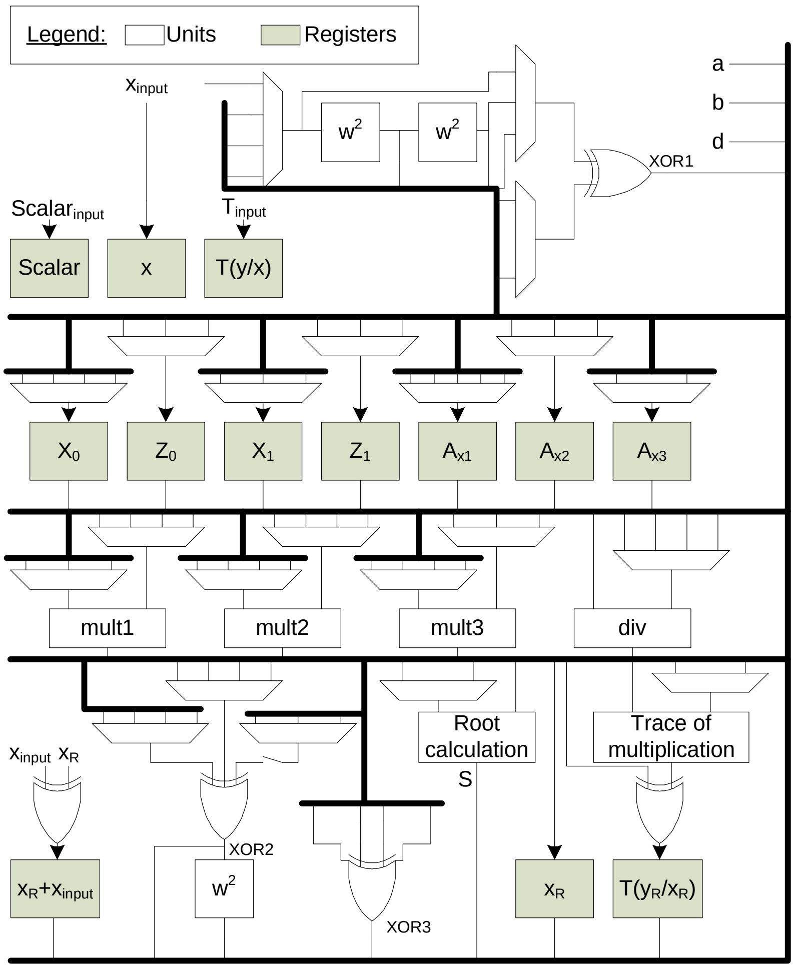

Figure 4.6: Complete Elliptic Curve Processor Schematic

Table 4.2: Architecture point multiplication scheduling

Point multiplication initialization
$X_0 = x$; $Z_0 = 1$; $X_1 = x^4 + b$; $Z1 = x^2$;
Point multiplication main cycle
$Z_{\bar{s}} = ((X_{\bar{s}} = X_s Z_{\bar{s}}) + (A_{x1} = X_{\bar{s}} Z_s))^2$; $A_{x2} = (dZ_{\bar{s}})^4$;
$X_{\bar{s}} = xZ_{\bar{s}} + X_{\bar{s}} A_{x1}$; $Z_s = (X_s Z_s)^2$; $X_s = A_{x2} + X_s^4$;
Obtaining point multiplication result collapsed coordinate
$A_{x2} = xZ_0$; $A_{x3} = X_1 Z_0 + (A_{x1} = X_0 Z_1)$
$A_{x1} = A_{x1}^{-1}$; $A_{x3} = X_1 A_{x3}$; $A_{x2} = xZ_1$; $A_{x1} = A_{x1}^{-1}$ (cont.); $X_0 = A_{x2} X_0^2$; $A_{x1} = A_{x1}^{-1}$ (cont.);
$x_R = X_0 A_{x1}$; $T\left(\frac{y_R}{x_R}\right) = T(A_{x3} A_{x1}) + T\left(\frac{y}{x}\right)$;

it is possible to parallelize this cycle into two steps using three multiplications, the use of three multipliers in the proposed architecture was decided.

Obtaining the collapsed representation of the result takes the time of two multiplications and one inversion, which results in $2 \times 43 + 327 = 413$ clock cycles.

When a point multiplication is completed, its result is stored into the registers x_R and $T(y_R/x_R)$. If a point addition operation is to follow, the input point will be added with this stored point. This stored point may be the secret obtained by multiplying the other party public key with the processor host's private key. Hence, the addition is a decryption or encryption procedure, depending on the origin of the new input: the remote connection or the host, respectively.

The scheduling used to perform addition is presented in Table 4.3. In this Table, the operation *Root_Calc* corresponds to (3.23). In the point addition initialization step, the y coordinate of the stored point is computed. If the stored point is to be added with a set of input points, this step can be performed only once. There is a bit *decrypt* which controls the complementation of the *Root_Calc* operation input trace. This allows to control the two possible resulting values of y/x: (y_R/x_R) or its inverse. Hence, this bit controls if an addition or subtraction is going to be performed. Moreover, this distinguishes an encryption from a decryption operation.

Table 4.3: Architecture point addition scheduling

Point addition initialization
$XOR_2 = x_R + a + \frac{b}{x_R^2}$
$S =$ *Root_Calc* $\left(XOR_2, T\left(\frac{y_R}{x_R}\right) + decrypt\right)$
$Z_0 = x_R S$ (Z_0 become the y coordinate of the last point multiplication result)

Point addition main procedure
$XOR_2 = x + a + \frac{b}{x^2}$
$A_{x1} = \frac{1}{x+x_R}$; $S =$ *Root_Calc* $\left(XOR_2, T\left(\frac{y}{x}\right)\right)$ $A_{x1} = \frac{1}{x+x_R}$ (cont.); $Z_1 = xS$ (Z_1 becomes the y input coordinate) $A_{x1} = \frac{1}{x+x_R}$ (cont.);
$A_{x2} = (A_{x1} = A_{x1}Z_0)^2 + (A_{x1} = A_{x1}Z_0) + x_R + x + a$ (A_{x2} becomes the result's x coordinate) $inv = A_{x2}^{-1}$; $A_{x1} = A_{x1}(A_{x2} + x_R) + A_{x2} + Z_1$
$inv = A_{x2}^{-1}$(cont.); (Output trace)$= T(inv\ A_{x1})$

The main procedure point addition step computes the remaining addition calculation procedure, namely the calculus of the uncollapsed representation of the input point, the computation in (2.25) and the calculus of the resulting point's trace.

The point addition first step requires the same computation time as a division, a root calculation and a multiplication, which results in $327 + 42 + 43 = 412$ clock cycles. The second step requires the time of 3 inversions and 1 multiplication, resulting in $3 \times 327 + 43 = 1024$ clock cycles.

4.2 FPGA Implementation

Given that each technology has different characteristics and resources, the computational structure should be adapted to a specific target. In this case, the proposed processor was targeted for a FPGA, a Xilinx Virtex 4 (model XC4VSX35). Thus, the direction of the implementation effort was to optimize for this specific target, in terms of an area and time performance balanced solution.

FPGAs, in particular the Xilinx Virtex 4, are supported by basic elements such as Configurable Logic Block (CLB). This element contains 4 slices which are composed, besides other peripheral elements, by 2 Look-Up Table (LUT) and 2 Flip-Flops [29]. There are also carry chains and shift chains to efficiently implement ripple carry adders and shift registers, respectively. The LUT elements are responsible for the combinatorial functions construction while the Flip-Flops are responsible for the memory cells (registers). These considerations allow to conclude that the area cost of a multiplexer is superior than for one register. For this reason, the minimization of the number of registers in the proposed structure was not pursued, rather the reduction of the multiplexers inputs. These considerations are comprehensively different for an ASIC technology.

Each LUT has 4 inputs and 1 output. This means that, independently of the complexity of the logic function, the complexity in terms of LUTs depends only on the number of independent variables which intervene in the combinatorial function. For a n input combinatorial function the propagation time will be $\lceil log_4(n) \rceil$ times one LUT propagation time. These characteristics may bring some disadvantages. For example, a n bit field addition is supported by n two input XOR gates. This means that for each result bit, the logic which could be used in 4 input logic functions is being used with a two input logic function. In an ASIC solution this operation may be performed more efficiently by a 2 input XOR standard cell. Nonetheless, for the FPGA technology, this operation, supported over $GF(2^k)$, remains more efficient than an addition using the carry chain, which may be used for $GF(p)$ supported solutions.

As discussed in Section 4.1.3, each output bit of a squaring operation over $GF(2^{163})$ can be obtained as a 5 input logic function for two output bits and 4 or less input logic functions for the other bits. Regarding the LUTs properties, one may conclude that the complexity of the squaring operation is the same as the field addition operation except for two bits regarding that 2 or 4 input functions have the same complexity in FPGA implementations.

The trace of the multiplication operation discussed in Section 4.1.4 can be computed recurring to a large amount basic logic cells. Once again, the FPGA characteristics allow the complexity to depend only on the number of input bits, in this case $2 \times 163 = 326$. This function can be constructed with a tree of LUTs with an appropriate configuration. Another issue that is important to highlight is the reconfigurable advantage of a FPGA. In the case of EC systems several fields and irreducible polynomials, several EC parameters and subgroup generators can be used. The hypothesis of adapting the procedures directly on the hardware accordingly to these situations or protocols used, allows for high performance solutions to be accomplished. This high performance adaptation is more difficult in ASIC solutions. For these solutions, it is possible to implement a structure for a large field prepared to operate with lower fields. The challenge is to obtain a comparable performance of these generic structures with the dedicated solutions for particular smaller field sizes.

The proposed ECC core was successfully implemented and thoroughly tested on a Xilinx Virtex 4 (XC4VSX35) prototyping platform. The implemented design computes the required point multiplication and addition for GF(2^k), supporting the ECC protocols based on this field. Table 4.4 describes the area occupation and performance for the implementation results, after Place&Route, considering a field size of $k = 163$. The area values for the individual units of the proposed ECC design include input and output registers for all data and control signals. These results were obtained from a VHDL description of the design, using Synplify Premier (version 8.6.2) as the synthesis tool and Xilinx ISE (version 9.2.04i) as the Place&Route tool.

Table 4.4: Implementation Results for the Xilinx Virtex 4

Unit	Slices	Freq. [MHz]	Clock cycles	Total time [ns]
squaring w^2	295 (1%)	467	1	2
squaring w^4	352 (2%)	317	1	3
mult$i$	1475 (9%)	248	43	173
div	1169 (7%)	147	327	2,221
root calc.	1066 (6%)	210	42	200
trace of mult.	458 (2%)	216	1	4
Complete ECC unit	10488 (68%)	99	-	-
(mult.)			14,303	144,374
(1st add.)			1,434	14,474
(add.)			1,024	10,336

From Table 4.4 it can be observed that the unit imposing the critical path is the division unit. Since the complete ECC unit is constructed with these units in parallel, it would be expected the complete ECC core to have an identical critical path as the division unit. However, a difference of 67% is observed from the full core and the division unit. This suggests that the interconnection logic, routing and multiplexing logic, has a significant impact in the overall performance of the ECC core. From the obtained figures, for the squaring unit, it can be concluded that this is an area and time efficient unit, occupying only about 2% of the available area. The trace of the multiplication unit is an efficient alternative to direct calculation of the trace of a multiplication instead of calculating the product first. In comparison with the proposed field multiplier performance, with about a third of the area it is possible to speedup the performance by 43 times. Considering the computation time of the field division, field multiplication, and the root calculation units, 412 clock cycles are required to perform a y coordinate calculation. This computation is described in Table 4.3 as the point addition initialization and is only computed once, when a secret is mapped in more than one EC point. It may be noticed that the difference between the first addition and the other ones is of 410 clock cycles instead of 412. This is due to the latency reduction resulting from the usage of a Mealy FSM.

Using the designed unit it is possible to achieve a throughput of $163\ bit/144\ ns \approx 1.13\ Mbit/s$ for the point multiplication and $163\ bit/10,336\ ns \approx 15.77\ Mbit/s$ throughput for point addition. As explained in Section 3.2.5, this last value traduces how fast a message can be encrypted or decrypted, considering that one can use all the collapsed representation bits to transmit information (ideal mapping). This issue is discussed with more detail in Chapter 5. Regarding the related state of the art, the comparison between designs is not straightforward, since different field sizes, different reconfigurable devices, and different design objectives are considered. It is not obvious what will be the influence of expanding or reducing a field size in each implementation. For the proposed structure, this field expansion is expected to degrade the overall performance due to an increased routing complexity. Nevertheless, in order to compare the related state of the art with the work herein proposed, time and area are considered inversely proportional towards the field size, with a correction factor of $163/k$, with k the compared field size. In Table 4.5 the point multiplication characteristics of the related work are presented and, in Table 4.6, the comparison metrics of the results obtained for the proposed structure are stated. Even though the proposed structure has been optimized for the Virtex 4 technology, Virtex-E and Spartan 3 implementations were also realized, in order to properly compare with the related work.

Table 4.5: ECC point multiplication state of the art

Ref.	Device	Field $GF(2^k)$	Slices	Max. Freq $[MHz]$	Total time $[\mu s]$
[25]	XC4085XLA	$GF(2^{191})$	2,634	32	1,610
[26]	XCV1000	$GF(2^{191})$	29,766[1]	36	270
[22]	XCV800	$GF(2^{160})$	1,596[2]	47	3,801
[23]	XCV3200E	$GF(2^{191})$	18,314	9.99	56
Proposed	XCV3200E	$GF(2^{163})$	9,432	49	292
[6]	XC3S1000	$GF(2^{233})$	n.a	80	2280
Proposed	XC3S2000	$GF(2^{163})$	10,379	44	325
Proposed	XC4VSX35	$GF(2^{163})$	10,488	99	144

Table 4.6: ECC point multiplication state of the art comparison metrics

Ref.	Device	Field $GF(2^k)$	required area $\times(\frac{163}{k})$	speedup $\times(\frac{163}{k})$	(Slices$\times$time)$^{-1}$ $\times(\frac{163}{k})$ $[Hz]$	(Slices$\times$time)$^{-1}$ improvement
[25]	XC4085XLA	$GF(2^{191})$	28%	0.21	0.32	0.89
[26]	XCV1000	$GF(2^{191})$	316%	1.27	0.17	0.47
[22]	XCV800	$GF(2^{160})$	17%	0.08	0.16	0.44
[23]	XCV3200E	$GF(2^{191})$	194%	6.11	1.34	3.69
Proposed	XCV3200E	$GF(2^{163})$	100%	1	0.36	1
[6]	XC3S1000	$GF(2^{233})$	-	0.20	-	-
Proposed	XC3S2000	$GF(2^{163})$	100%	1	0.30	1
Proposed	XC4VSX35	$GF(2^{163})$	100%	1	0.66	1

Many of the proposed ECC structures use Lopez-Dahab coordinates [6, 22, 25, 26], and require 10 and 4 multiplications to compute point addition and point doubling, respectively. To perform simultaneously point addition and doubling our design requires 6 multiplications.

[1]The presented Virtex-E implementation ratio 0.616 Slice/LUT was used to convert LUTs to slices.
[2]The Xilinx Virtex datasheet information was used to convert logic gates to slices.

The use of NAF scalar recoding is proposed in [6], in order to reduce the required point additions in the traditional double and add algorithms. Because the proposed design has a speedup of 5 it is expected that only if this design would avoid 4 additions in every 5 additions, it would achieve the same time performance as the design herein proposed, considering identical field multiplication performance. Furthermore, this design does not implement affine point addition or projective to affine point conversion.

To improve the performance of projective to affine conversion, a mixed coordinate representation is proposed in [26]. However, this conversion procedure has far less weight in the complete ECC multiplication procedure. Jacobian coordinates applied on the Montgomery ECC multiplication are also used in [26], resulting in a better performance using the Massey-Omura field multiplier supported over an optimal normal base. Nevertheless, performance comes at the higher expense of circuit area. Moreover, no parallelization of the ECC multiplication algorithm can be exploited. The proposed design suggests a better performance, since parallelization techniques can be used in order to obtain a more area efficient solution, resulting in an improvement of the $(\text{slices}\times\text{time})^{-1}$ by 0.47. In the presented implementation 3 parallel point multipliers are employed.

In [25] an Itoh-Tsujii inverter is implemented using recurrent multiplications. In the proposed design, an inversion takes approximately the same time as 8 multiplications. Since, in [25] more than 8 multiplications are needed to perform the inversion, the proposed design can achieve higher performance. Furthermore, with a dedicated divider architecture it is possible to parallelize the division and multiplication procedures, which is not possible in the approach used in [25]. This design requires $3.2ms$ to perform the generic operation $kP + rQ$ (with k, r integers and Q, P EC points). In the proposed design, the same operation can be computed with two scalar multiplications and one point addition which, for the presented Virtex-E implementation, requires 613μs, thus a speedup of about 5 is achieved when compared with [25]. The inversion procedure used in [22] is based on the *Fermat Little Theorem*, allowing for a more compact design. However, the costs in performance are very high, resulting in $(\text{slices} \times \text{time})^{-1}$ efficiency metric 84% lower.

The design proposed in [23], using two field Karatsuba-Offman multipliers, is able to achieve the computation of point multiplication about 6 times faster than the structure herein proposed. Part of this improvement is obtained from a more aggressive parallelization, which results in an area 94% higher. It is important to highlight that this design does not support the computation of point addition or projective to affine coordinates conversion. Moreover, in [23] embedded memories (BRAM) are used to perform part of the data routing. The 24 BRAMs used are not considered in the area metric. The use of the BRAMs improves the critical path, but only the very first positions of each BRAM are used, thus this solution is not area efficient.

As a concluding remark, it should be noted that none of these designs implements coordinate y collapsing, as is the case of the proposed structure. This coordinate collapse allows for a reduction by half of the required bandwidth.

4.3 ASIC Implementation

An implementation for an ASIC technology is also realized. The goal of this implementation is to evaluate the characteristics of the proposed processor for this type of technology and compare it with the FPGA implementation. To perform this implementation the same VHDL description code was employed, using synthesis and Place&Route tools for ASIC technology. The used synthesis tool was the Design Compiler (version Z-2007.03-SP5) and the First Encounter (version 05.20-s197_1) was the Place&Route tool. The target ASIC technology is supported by the process UMC L180 1P6M MM/RFCMOS (0.18 μm) using Faraday FSA0C_A 0.18 μm Standard Cell Library (version 1.0). The Figure 4.7 shows the layout of the placed and routed processor for this ASIC technology. This layout has a dimension of $1.4 \times 1.4\ mm^2$ and operates at a maximum

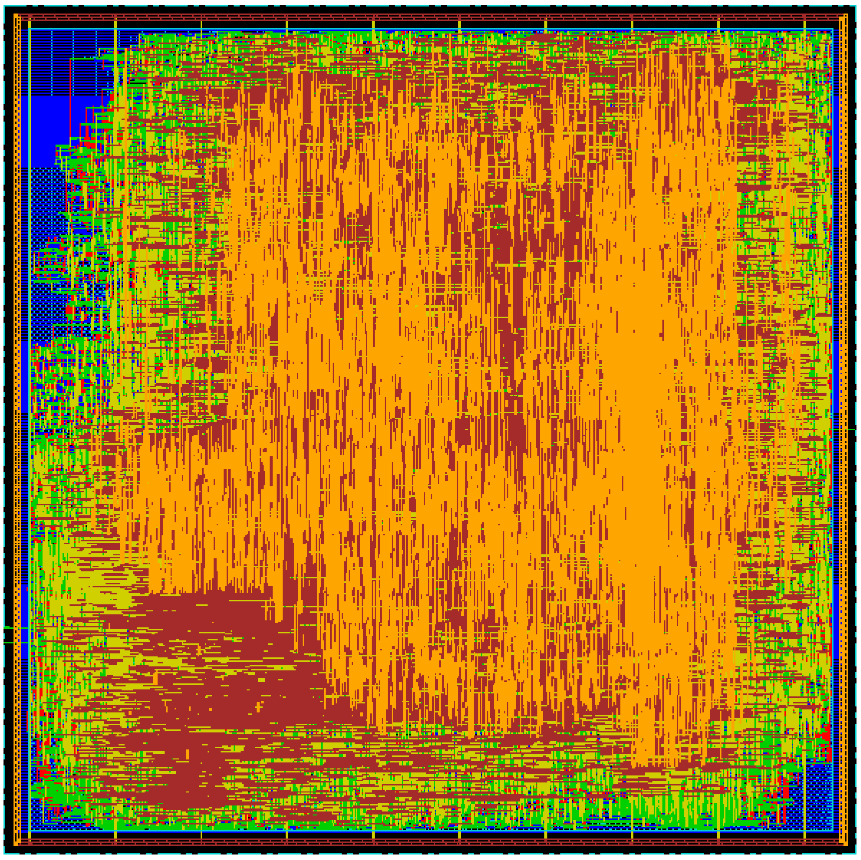

Figure 4.7: ASIC technology elliptic curve processor layout

frequency of 60.7 MHz, which represents a throughput of $693\ Kbit/s$ for point multiplication and $9.67\ Mbit/s$ for point addition (encrypting and decrypting). It is not possible to perform a thorough comparison of these values with the ones obtained for the FPGA implementation. This is due to the fact that the two technologies have completely distinct characteristics, as is the $95\ nm$ technology of the Xilinx Virtex 4 versus the $0.18\ \mu m$ technology of the proposed ASIC implementation or the logic cells of a standard library for the ASIC and the LUTs of the FPGA. For this reason, only some of the characteristics of the ASIC layout are evaluated. One of these characteristics is the routing effort. From Figure 4.7 it can be observed that all the metal layers of the possible 6 that the utilized technology allows are extensively used. This fact suggests that the routing demand of the proposed design is significantly high. To properly evaluate this characteristic two individual implementations of the processor field arithmetic units, namely the field multiplier and divider/inverter, were realized. These individual implementations are presented in Figure 4.8. These implementa-

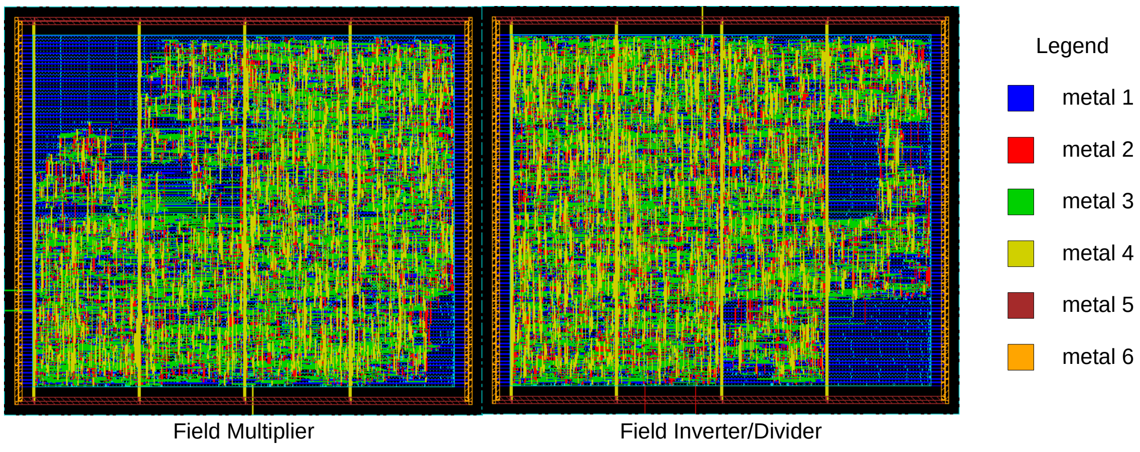

Figure 4.8: ASIC technology field multiplier and inverter/divider layout

tions have the dimension of $0.6 \times 0.5\ mm^2$ and run at the frequency of $205.5\ MHz$ for the multiplier and $211.1\ MHz$ for the inverter/divider. The complete processor was obtained through the interconnection of these units. One method to evaluate the routing effect over the complete processor is to compare the critical path of the basic units (multiplier and inverter) with the critical path of the complete processor constructed over these units. Looking at the operating frequency a decrease in performance for the complete processor of about 70% can be observed. This performance decrease is significant and is caused by the complex routing, increased metal nets, and increased fanout. Analyzing the data path of the basic units it is observable that 10% and 6% of the total delay is caused by the routing nets, for the multiplier and inverter, respectively. Performing the same analysis for the complete processor it can be observed that 31% of the delay is attributed to the routing demands. This result allows to conclude that the routing has a significant impact in the performance of the proposed processor. Reducing the routing requirements of the proposed ECC processor might be an important step in improving the overall performance of the ASIC implementation. The routing costs on the FPGA are mitigated by the optimized routing matrix existent in these devices. For this reason, the compromise between logic and routing delays optimization has to be different for these two technologies. The advantages of the hardware implementation over $GF(2^k)$ arises from the possibility to perform much of the arithmetic, for example the squaring, using hardwired and irregular bits manipulation, demanding an extra effort from the routing tools to obtain high performance designs.

4.4 Summary

This chapter discusses the architectural options in the design of the proposed EC processor. The interconnections between the several field operation units are also analyzed in order to efficiently perform the EC arithmetic operations. The complete architecture of the EC processor and the scheduling of the operations executed to perform the cryptographic routines are also presented. The motivations that lead to the adopted options are presented, namely the point multiplication unit supported by the used Montgomery method. In this chapter, the original contributions related with point collapsed representation, are also introduced and discussed. The units supported by the original point addition method, upgraded with the pre-computed calculation of the y coordinate from the collapsed representation, are also presented.

The implementation results of the proposed EC processor on a Xilinx Virtex 4 FPGA technology are also presented and discussed in this Chapter. These results suggest that it is possible to achieve a throughput of about $1\ Mbit/s$ for point multiplication and $15\ Mbit/s$ for point addition. Implementations on different FPGA technologies were also performed, in order to properly compare the proposed design with the related state of the art. The main advantage of the proposed design, regarding the related state of the art, arises from the careful scheduling of the Montgomery point multiplication algorithm, updated with the collapsed representation considerations, herein introduced. A point addition unit which efficiently manages the collapsed representation was also presented. An ASIC solution is also presented, suggesting a throughput of about $700\ Kbit/s$ for point multiplication and about $9.7\ Mbit/s$ for point addition (message encryption and decryption). The results highlight some characteristics of the proposed structure, namely the significant routing complexity. This suggests that the routing complexity is an important factor to be taken into account in the design of these structures.

In the next chapter the results for a software implementation of the used algorithms and a real system prototype of the proposed EC processor are presented and discussed.

5

System Application

Contents

In order to evaluate the advantage of the proposed EC processor a software implementation using the same algorithms is also presented. Additionally, a prototype has also been designed and used in a real application to validate this EC processor. The main goal of this prototype is not to achieve a very high throughput, rather to demonstrate that the proposed processor can be used in real systems for public key cryptography. This processor provides a complete support for public keys generation and messages encrypting/decrypting. This functionality is very important for servers which need to distribute a significant number of public keys. Another application where such EC processor can have significant importance is point to point secure communication. This processor can be employed in any platform where FPGA technology is available, preventing the host system from EC arithmetic computation. The host only sends and receives the decrypted messages, and provides the necessary random numbers (keys) for this EC processor.

In order to properly analyze the benefits of having this EC dedicated hardware implementation, the performance and hardware resources required by both hardware and software implementation are compared in this chapter. This chapter also describes the processor interface and the application schematic.

5.1 Elliptic Curve Software Implementation

In order to discuss the advantages of the proposed hardware processor, the performance of a software solution is also evaluated. This software implementation is designed based on the same algorithms as the ones used in the proposed processor. Additionally, this software implementation provided the complete library used to facilitate the test of the correctness of the processor.

The direct comparison between software and hardware may not be fair in the sense that all the algorithms, and particularly the chosen field $GF(2^k)$, are optimized for a hardware solution. It would not be possible to exploit the general propose processors arithmetic for this kind of fields, as it would be the case of a $GF(p)$ supported system. For this reason, the comparison is intended to be as a indicative of gain.

Two distinct general purpose processor architectures were used to perform this test: the MicroBlaze soft processor and an Intel® Centrino Duo Central Processing Unit (CPU). The MicroBlaze is a 32 bits processor with reconfigurable components, developed by Xilinx® to be efficiently implemented into FPGAs [30]. For this test, the MicroBlaze was configured with a $16\ Kbytes$ memory size without cache for both instructions and data. This processor was synthesized for the same FPGA used to implement the proposed EC processor. Using Xilinx Platform Studio tools (version 9.1.02i) a frequency of $100\ MHz$ was obtained for the MicroBlaze processor. This frequency differs only in $1\ MHz$ from the frequency obtained for the dedicated hardware solution. The Wildcard 4™ prototyping board was used as the testing platform. This board allows to interact with the FPGA through a PCI bus. The EC implementation using the MicroBlaze processor is depicted in Figure 5.1. More details about MicroBlaze, Wildcard 4 and its components are presented in Appendix A.

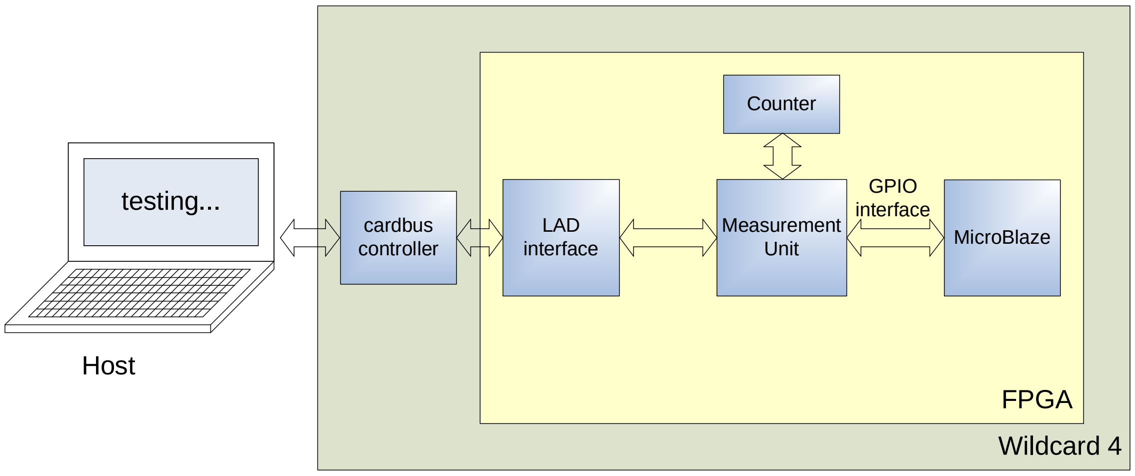

Figure 5.1: MicroBlaze software implementation test layout

The FPGA configuration file (bitstream) was obtained by synthesizing the top level structure with Synplify Premier (version 8.6.2) with the MicroBlaze component as a blackbox. These components were merged and placed in the device using the Xilinx Place&Route tools (version 9.2.04i). The resulting area occupation was of 3218 out of 15360 slices (20%) and the maximum obtained frequency was $103\ MHz$ (above the required $100\ MHz$).

From Figure 5.1, the existence of two blocks *Cardbus Controler* and *Local Address Data (LAD) interface* can be observed. The first one is in the board and the second one is described as VHDL code. In practice, these blocks create the communication interface that allows to easily communicate with the Host by writing/reading to/from registers. The General Purpose Input/Output (GPIO) peripheral is a MicroBlaze port, which provides an easy register read/write communication interface with the software. From the software point of view, these registers are read and written by accessing a memory address.

In order to evaluate the system, a measurement unit was added. This unit, counts the time performance of the operations processed by the MicroBlaze and the communication overheads between the Host and the MicroBlaze. For this count, a counter is used. In order to reduce the counting errors, several measurements were performed to calculate the average.

To estimate the communication time between the Host and the MicroBlaze, 6 positions of 32 bits were transferred from the host to the MicroBlaze and returned in the opposite way without performing any computation. This means that the communication was accounted joining the transmission of an output and the reception of an input set of data. The used data set of 6 positions of 32 bits was chosen since it is the minimum required to transmit 163 bits, the dimension required to code the collapsed representation of one EC point.

The software was written in C language, and compiled for the MicroBlaze using the *mb-gcc* tool (version 3.4.1) with medium optimization effort. In Table 5.1 the time performance for the various operations is presented. For each operation, these results were obtained by sending a set of data to trigger the computation of a large number of iterations of the same operation, and wait for the response of MicroBlaze at the iteration finish. Then, the estimated communication overhead was subtracted from the measured time, in order to obtain the time for a given operation. Finally, the total time was divided by the number of iterations. In Table 5.1 is presented the ratio between the software results and the Hardware (HW) results, considering the $99\ MHz$ working frequency of the full EC processor.

Table 5.1: Software results

Description	MicroBlaze (MB) $[ns]$	PC $[ns]$	Ratio MB/HW	Ratio PC/HW	Ratio MB/PC
Host communication	25,610	-	-	-	-
MicroBlaze communication	25,980	-	-	-	-
Field multiplication	208,170	43,810	479	101	4.75
Field division	786,760	110,360	238	33	7.12
Squaring	274,140	20,990	27,414	2,078	13.06
Proj. coord. doubling	1,307,680	151,170			8.65
Proj. coord. addition	1,214,170	178,040			6.81
Point multiplication	412,946,820	55,874,210	2,858	387	7.39
Proj. to collapsed rep. conversion	3,295,760	546,150			6.03
Obtain y from collapsed rep.	44,988,560	3,575,310			12.58
Point addition	2,067,130	298,570	143	21	6.92

The same operations were tested for a Personal Computer (PC). This machine is supported by an Intel® Centrino Duo CPU model T2400 at $1.83\ GHz$, with $2 \times 512\ Mbytes$ of Random Access Memory (RAM), running a Microsoft Windows XP SP2 (2000 version) Operative System (OS). The same C code was used, compiled using Microsoft Visual C++ .NET with Microsoft Development Environment 2003 (version 7.1.3088) and Microsoft .NET Framework (version 1.1.4322 SP2). No compiler optimization was used. Time accountings were performed recurring to a counter available from the OS for multimedia applications. Once more, a large amount of iterations was used for each operation in order to mitigate fluctuations. The results are also presented in Table 5.1.

Comparing the implementations, it is observed that a point multiplication is 2,858 times faster in the hardware solution when compared with the MicroBlaze implementation, using only 3.26 times more area. Comparing with the PC solution, the hardware multiplication is 387 times faster. For point addition the differences are lower, with the hardware solution being 143 and 21 times faster than the Microblaze and the PC solutions, respectively. Observing the results for the squaring operation, the reason for such a difference in point multiplication and point addition can be outlined. The efficiency of the squaring unit in the hardware solution comes from a non regular table lookup bit selection as explained in Section 4.1.3. This bit selection can not be performed efficiently in software resulting in a very significant difference in terms of performance between the dedicated hardware implementation and the MicroBlaze and PC solutions: $27,414$ and $2,078$, respectively.

5.2 Elliptic Curve Processor Application

This section presents the developed prototype, supported by the proposed EC processor. This prototype provides the complete EC arithmetic to a host system and creates a secure channel with a remote system. It is able not only to encrypt/decrypt messages and send/receive them to/from the remote system, but also to generate a stream of public keys, useful in server applications which need to establish several sessions in short time requirements. In Figure 5.2 is presented the schematic of the complete system. The Elliptic Curve Processor Control (ECPC) block interacts

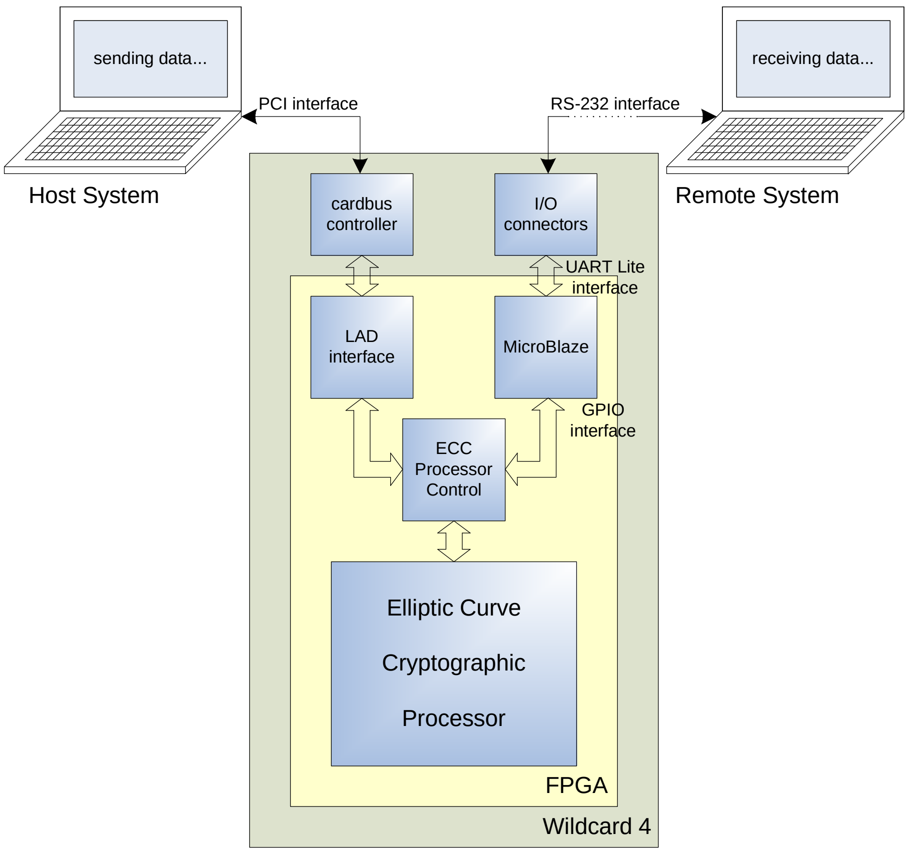

Figure 5.2: Complete System Schematic

with the LAD interface and MicroBlaze GPIO peripheral as registers. The communication between these entities is performed by reading/writing from/to these registers. The details of the blocks used in this application are described in Appendix A.

The cardbus controller together with the LAD interface act as a pair of addressable registers of 32 bits, one to write and the other one to read. The number of register pairs must be a power of 2. To transmit a word of 163 bits (collapsed EC point) 6 registers are needed. Thus, 8 registers must be allocated, which is the smallest power of two that contains 6. Since there are two more registers available, apart from the 6 registers used to transmit data, one is reserved for control commands.

In order to receive new commands from the Wildcard 4, the Host software pulls the register reserved for commands and scans the strobe bit reserved to indicate that the register was updated, indicating the existence of a new command. This scanning consists in determining if this strobe bit changed its state since the last valid command. If there is a new command, it may transmit any specific information or inform that there are new data available. In this last case, the 6 registers reserved for data can be read and the Wildcard notified that the read was successfully accomplished, writing this information to the commands register.

For the ECPC, when a new command is received the command register is updated and a strobe signal activated. When receiving data, the data registers are updated and the respective strobe signal is generated. After that, a command is received in the input control register informing that new data was received. When another set of data is received, the output command register is written with the request to the Host and the strobe bit complemented. In Figure 5.3 are presented the LAD registers and the signals behavior while the reception of data from the Host by the ECPC, and the sending of the reception notification by the ECPC to the Host.

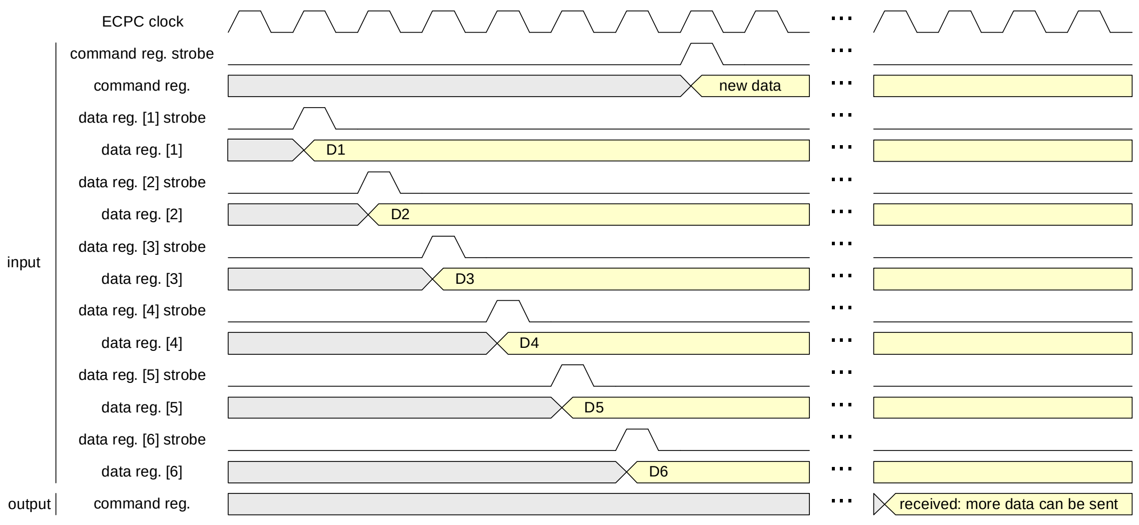

Figure 5.3: LAD registers and signals behavior for ECPC data receiving from Host

The number of clock cycles between the update of the different LAD registers while transmitting the same data set may not be only one clock cycle as depicted in Figure 5.3. It is only granted the order of the updates, but not the time spent between them. Thus, the time information in Figure 5.3 is only indicative.

The communication between the ECPC and the MicroBlaze has the same characteristics as the communication between the LAD and the ECPC, in the sense that it corresponds to read and write the registers content. Nevertheless, there are some differences, namely the inexistence of a strobe signal available from the GPIO and the fact that it is only possible to write/read 32 bits at once to/from the GPIO data register. To solve the inexistence of strobe, strobe bit in the command register is used for both read and write procedures. In order to communicate more than one 32 bits word (collapsed EC points), for each word, a notification of sent/reception is performed through the commands register for every data word.

To send data from the ECPC to the MicroBlaze, a 32 bits word is written to the data GPIO. The command register is updated with the information that there are new data to be sent, and the strobe bit is complemented. The Microblaze scans the strobe bit for valid commands, evaluates the information in the command register, reads the data value from the data register, and writes to the output command register that another word can be sent, complementing the output strobe bit. After scanning the strobe bit for a valid command, the ECPC writes another word to the data register and the procedure is repeated until the 6 words are sent. The diagram for the sending procedure is depicted in Figure 5.4.

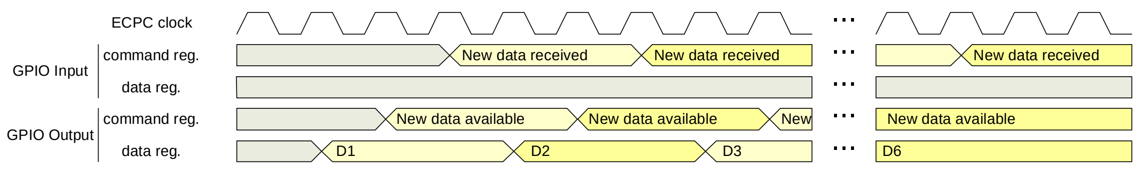

Figure 5.4: GPIO registers behavior for the ECPC to the MicroBlaze data communication.

In order for the ECPC to receive data from the MicroBlaze, it is assumed that, when the MicroBlaze sends data, the ECPC is always waiting these data and that it can be immediately read it. This way, only the input GPIO register is used in this communication, because there is no need for reception acknowledgment commands. The MicroBlaze writes a word to the data registers, updates the commands register, and complements the strobe bit. This procedure is performed six times in order to send all the needed words. When the ECPC receives a valid command, it reads the data and stores it into a register until all the data is transferred. The GPIO registers behavior is presented in Figure 5.5.

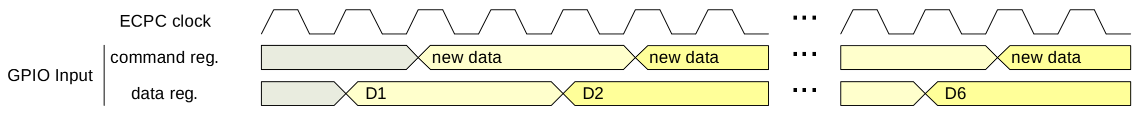

Figure 5.5: GPIO registers behavior for the MicroBlaze to the ECPC data communication.

In order to optimize the used resources, the GPIO commands register was limited to a width of 12 bits. Similarly with the LAD interface, not all 32 bits are needed for the control. The difference is that for the LAD register the 32 bits must be allocated, because it corresponds to one LAD address, while for the GPIO the width corresponding to the GPIO address can be configured and the unnecessary bits removed. The MicroBlaze software registers access is identical to the Host LAD registers access, with the difference that for the MicroBlaze there are two distinct channels to write and read.

The communication between the Microblaze and the Remote System is performed using a serial Universal Asynchronous Receiver/Transmiter (UART) stream over an RS-232 standard connection. The UART is configured with a throughput of 115200 bits per second, one parity bit with odd parity definition, and a frame size of 8 bits.

Using a frame size of 8 bits, it is possible to correspond each frame to one software character. Thus, an EC collapsed representation may be divided into 8 bit consecutive frames. The number of frames is given by $\lceil 163/8 \rceil = 21$. A string termination character and characters to code the necessary commands were also defined.

The logic levels used in the FPGA and in the communication ports of the Remote System PC may differ. In this prototype the FPGA output tension level is at $3.3\ V$ while the communication port in the Remote system is at $5\ V$. For this reason the use of a transceiver is required to adapt the two levels. In this test layout a transceiver that can assure this function for a throughput up to $1\ Mbit/s$ is used, supported by the MAX3222E integrated circuit by Maxim Integrated Products [31]. Furthermore, an adapter from Universal Serial Bus (USB) to RS-232 was used, in order to provide the $5\ V$ supply to the transceiver. The electrical schematic used to connect the transceiver is presented in Appendix B. A photograph of the complete system layout is described in Figure 5.6.

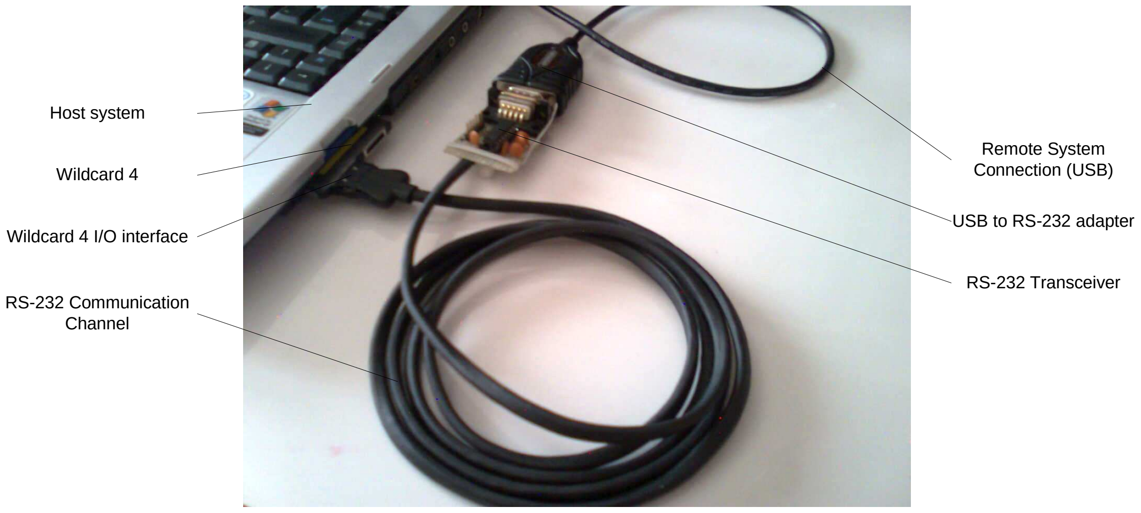

Figure 5.6: Application test layout photograph

5.2.1 Protocols

The proposed application structure supports three protocols: *i)* a public key stream from the Host System to the Remote System (Figure 5.7), *ii)* a secure message stream from the Host to the Remote System (Figure 5.8) and *iii)* a secure message stream from the Remote to the Host System (Figure 5.9). The field parameters, the EC parameters and the subgroup generator are defined statically, which means that they are defined during the systems' setup, and not during the sessions setup. In other words, these parameters are defined in the software code and in the FPGA bitstream, thus there is no need to be communicated. In the case of the FPGA, it means that all the parameters are defined in hardware constants when the FPGA configuration is loaded. The first protocol may fit some server applications which may require regular public

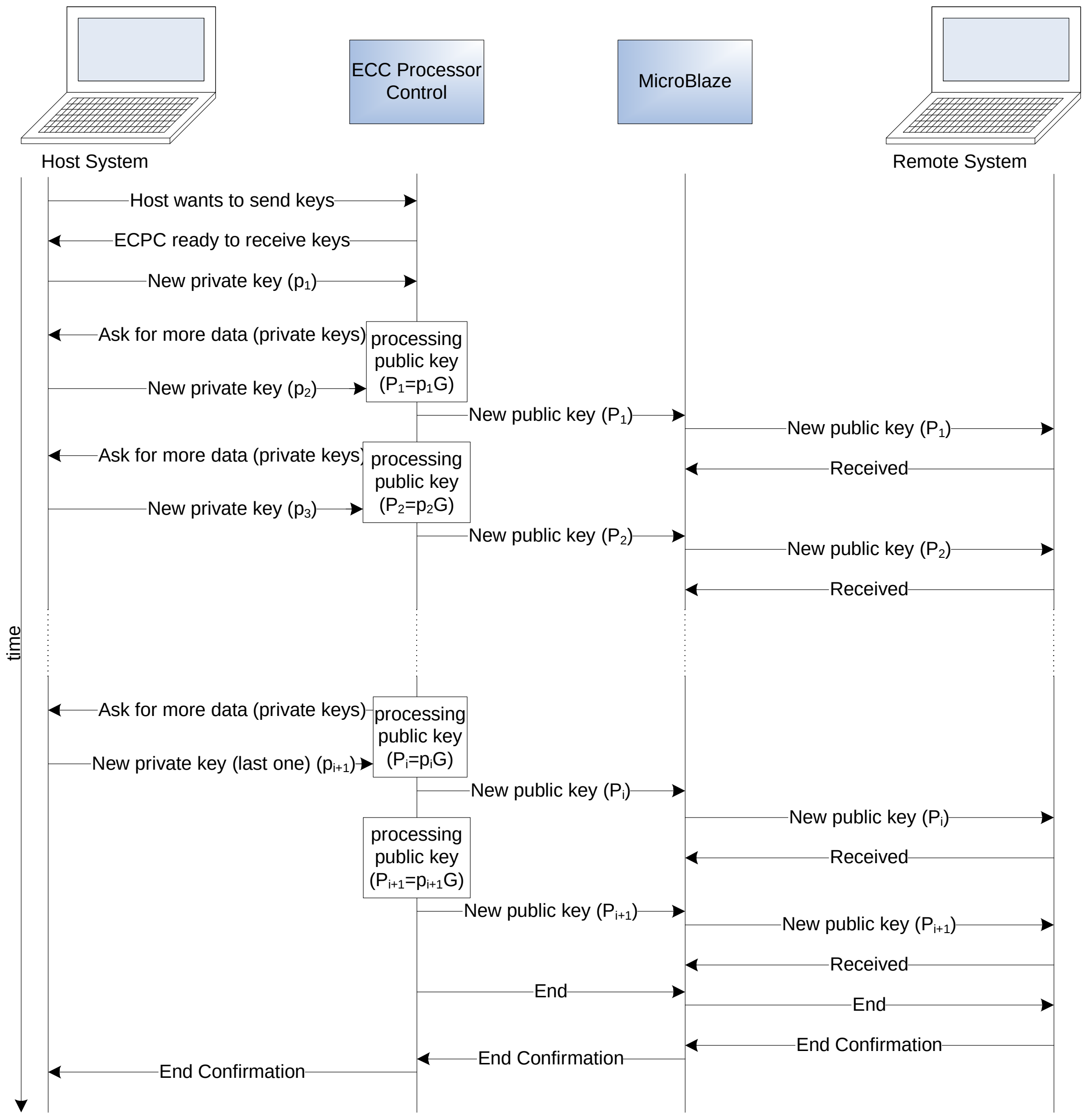

Figure 5.7: Host to Remote public keys stream protocol

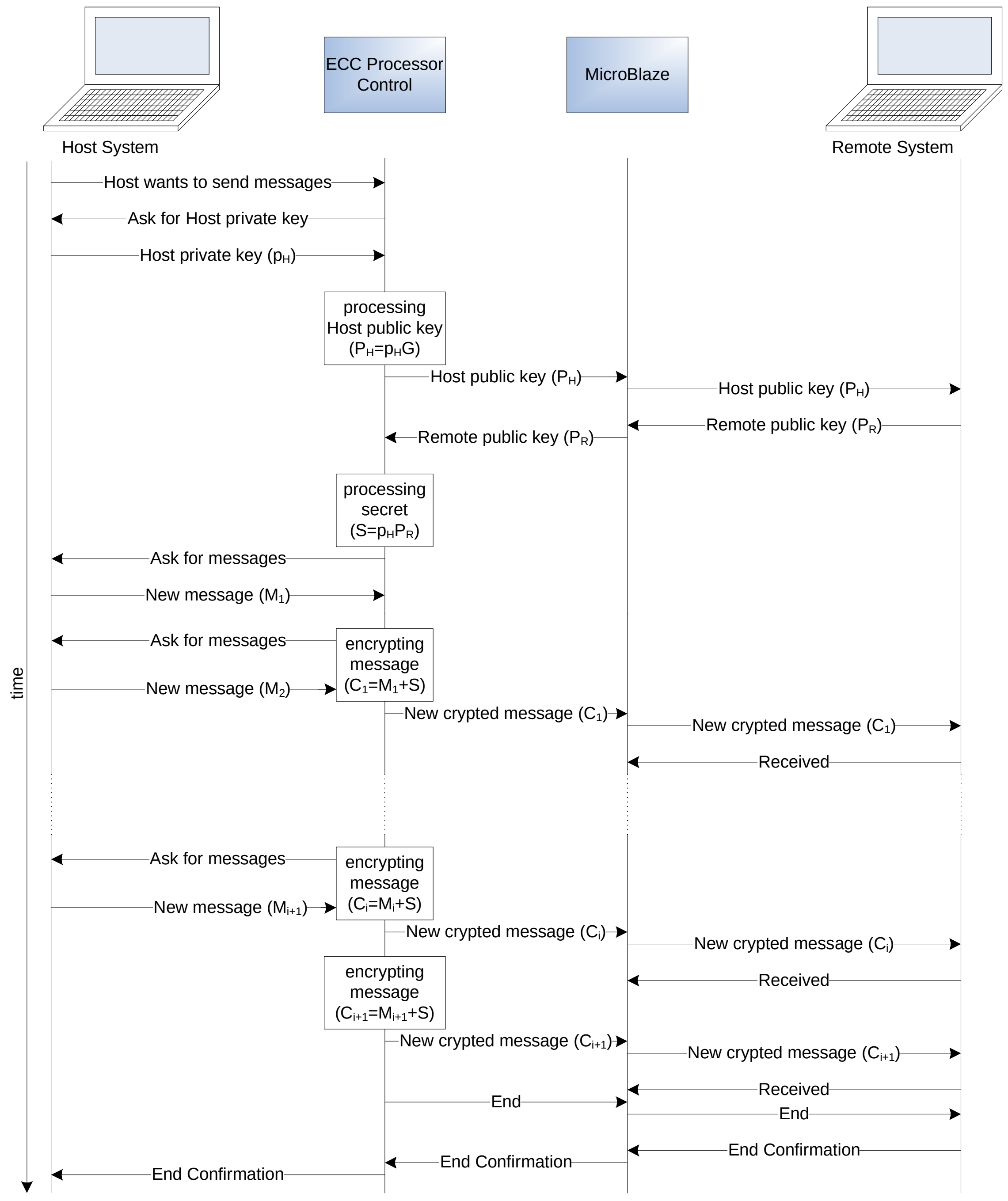

Figure 5.8: Host to Remote message communication protocol

keys distribution. The other protocols suit applications that transmit confidential data between two terminals. The ECPC is capable of storing new input data for the EC processor while this last one is processing. Whit this, when the EC computation finishes another one can be immediately started, reducing the overhead created by the Host or Remote System communication. This is the reason why the ECPC requests for new data while the EC processor is performing the computation. The data transfers between the MicroBlaze and the Remote System requires an

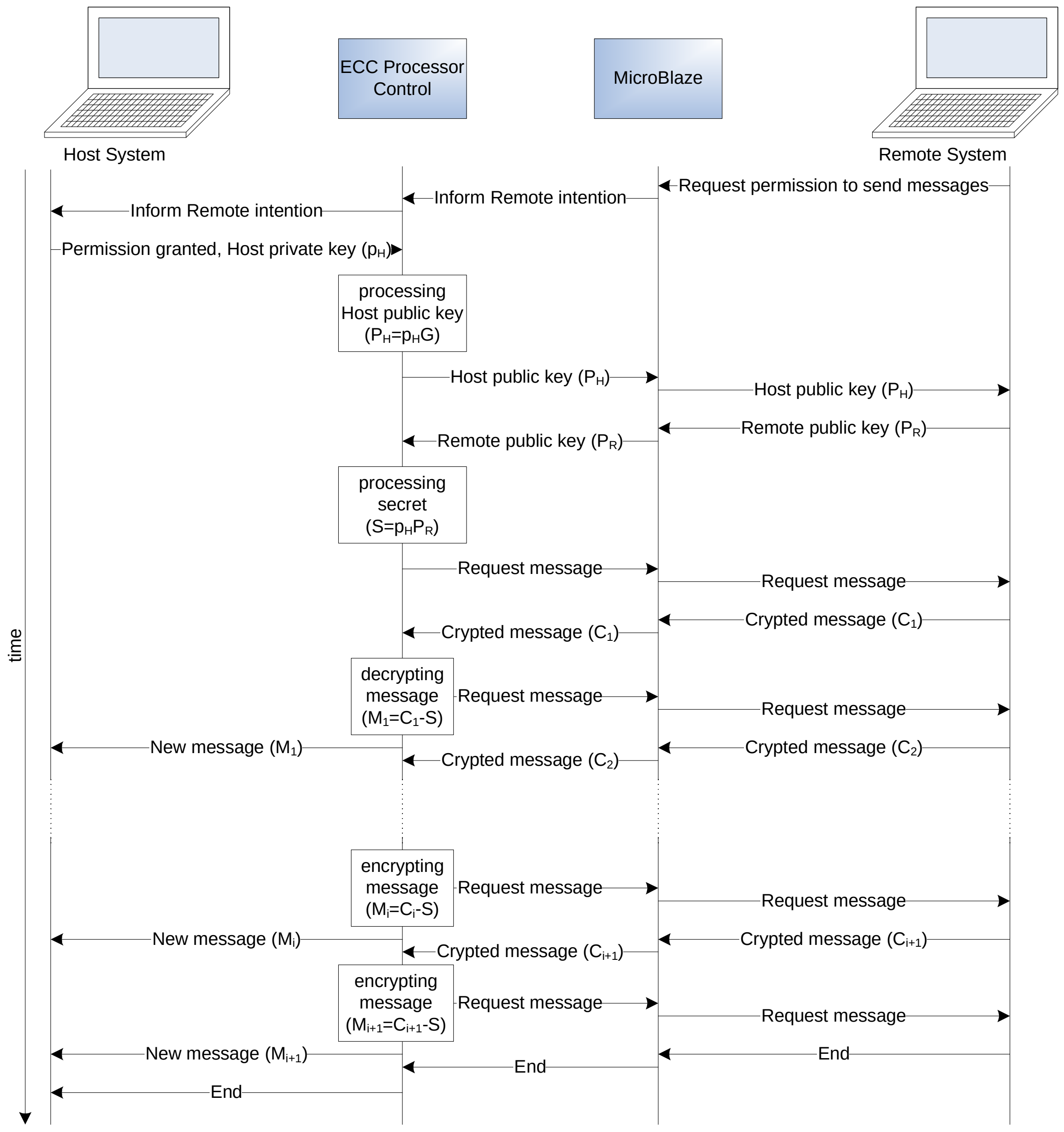

Figure 5.9: Remote to Host message communication protocol

acknowledge of the reception. This may increase the communication overhead but prevents the UART buffers to overrun, avoiding any data loss.

5.2.2 System Implementation and Results

This section describes practical implementation issues of each system component and the experimental test results. This is used to test the concept of the ECC procedures. The RS-232 interface can be easily managed but offers a low throughput. Nevertheless, it is sufficient for the processor functionality test. The Xilinx XC4VSX35 FPGA was chosen as the target reconfigurable technology. The FPGA configuration was obtained by the same method used in the software test presented in Section 5.1. The MicroBlaze characteristics are also the same, except for the working frequency, which in this case is $50\ MHz$. The resulting FPGA occupation is of $13,461$ slices (87% of the available 15,360 slices), with a maximum clock frequency of $53.717\ MHz$. A clock frequency of $50\ MHz$ is used in this prototype.

For this test, the Host and Remote System were the same machine. This machine is supported over an Intel® Centrino Duo CPU model T2400 at $1.83\ GHz$, with $2 \times 512\ Mbytes$ of RAM, running a Microsoft Windows XP SP2 (2000 version) OS. The program code was written in C language, using the Wildard 4 Application Programming Interface (API) for the Host program and the Windows API for the Remote program, in order to configure and access the serial communication port. The programs were developed and compiled using Microsoft Visual C++ .NET with Microsoft Development Environment 2003 (version 7.1.3088) and Microsoft .NET Framework (version 1.1.4322 SP2).

The design was tested with the transmission of 300,000 public keys using the protocol in Figure 5.7 and the transmission of a message mapped into 300,000 EC points. There is not a direct mapping between data and EC points. It is usual to perform a trial search for EC points [8]. In this case, the transmitted message was generated from 300,000 163 bit coordinates. Each coordinate was manipulated conditioning one bit in order for the trace of the coordinate to be equal to the trace of the parameter a of the EC, and for the point collapsed representation to work as described in Section 3.3. A trial for the 5 most significant bits was performed till the coordinate to belong to an EC point, which is equivalent to $T(x+a+b/x^2) = 0$, as explained in Section 2.3.2.

A successful test for the global system was achieved with the correctness of the received data, for both protocols. In the Remote program a precision counter provided by the Windows OS was added, in order to account for the total transfer time. The obtained time results for the key transfer are 1200.11 seconds for $300,000$ keys, and 1201.91 for 300,000 EC coordinates for the message transfer. These results suggest that the system performance is conditioned by the communication overhead, because as obtained in Table 4.4, the point addition is about 14 times faster than point multiplication and the throughput for both message and keys transfer is approximately the same in this application. This result is expected in the sense that the maximum throughput of the EC processor is $163 \times 50\ MHz/14303 \approx 570\ Kbit/s$ for point multiplication and $163 \times 50\ MHz/1024 \approx 8\ Mbit/s$ point addition, while the maximum throughput in the RS-232 serial communication is $115,200\ bit/s$. Furthermore, in the serial communication are used control characters which contribute for the communication overhead.

From Table 5.1, it can be estimated that one coordinate is transferred through the LAD interface in about $2,561/2 \approx 1280$ clock cycles. This corresponds to a throughput of $163 \times 50\ MHz/1280 \approx 6.4\ Mbit/s$, thus the bottleneck may not reside in this interface. One coordinate transfer through the GPIO interface can be performed in about $2,598/2 = 1299$ clock cycles, which corresponds to $1299 \times 20\ ns = 25980\ ns$. The 163 bits belonging to a coordinate can be transmitted over

21 characters. One more character is added to delimit the string. The acknowledgement of the data reception is performed by communicating two characters. This means that to transfer an EC point over the RS-232 connection, 24 characters are transmitted. Moreover, to transmit one frame (character), 8 bits are needed to code the data and two more bits, one for parity and the other one as a stop bit. Considering the bit rate of the RS-232 connection, one coordinate takes $24 \times (10\ bit)/115,200\ bit/s \approx 2.08\ ms$ to be transmitted. From the experimental results, the transmission of one coordinate requires $1201.91/300,000 \approx 4.01\ ms$. This means that there is a communication overhead of about $4.01 - 2.08 = 1.93\ ms$ for the MicroBlaze and/or Remote system to access the communication buffers. Nevertheless, from the proposed implementation it is possible to conclude that, when associated with efficient communication interfaces, the EC processor can achieve a good performance and can be efficiently implemented in real systems.

5.3 Summary

Two software implementations providing the EC processor functionality were developed and tested. One of these solutions ran on the MicroBlaze softprocessor implemented on the same FPGA as the proposed design. The other one ran on an Intel general purpose processor. These solutions confirm that the chosen field properties are better adapted for dedicated hardware solutions, since the hardware solution was shown to be up to 2,858 times faster for point multiplication, and 143 times faster for point addition.

A prototype using the proposed EC processor was also presented. This application was used to demonstrate that the proposed EC processor can be efficiently integrated into communication protocols achieving good performances.

The resources of the prototyping Wildcard 4 card, its characteristics, and how they can be interconnected in order to implement public key protocols are presented. The Xilinx MicroBlaze soft processor configuration was also presented.

The proposed application design test does not need high throughput. The utilized communication options were selected for a simple management and not for a very high performance. The main goal of this prototype is to test the applicability of the EC processor in real systems, and the usability of the proposed processor, which was successfully accomplished.

6 Conclusions

Contents

6.1 Summary and Overall Conclusions

The main objectives of this thesis were successfully accomplished with the designing of a complete ECC processor optimized for a FPGA target, namely the Xilinx Virtex4 FPGA. The proposed ECC processor was thoroughly tested in this target.

Even though a direct comparison with the related work can not be easily made, since different FPGA technologies and field parameters are adopted, implementation results obtained in this thesis suggest that significant performance gains can be achieved with the design proposed in this thesis. In comparison with most of the related state of the art, performance gains in the order of 5 times for point multiplication and addition can be achieved. When compared with the related work with better time results, it is possible to find a hardware structure that performs the point multiplication 6 times faster, but at the expense of 94% more area and several additional BRAMs. Moreover, this related work is only capable of computing point multiplication, while the structure proposed in this thesis is capable of computing the full ECC operations.

The design herein proposed performs coordinate collapsing, allowing for a better usage of the available bandwidth. The proposed coordinate collapsing is supported by a compact representation of EC points which is possible when operating over prime cyclic subgroups. This is an original contribution of this thesis, which adapts this method into an algorithm to compute the exponentiation of a point without computing the y coordinate. Furthermore, it was achieved without compromising the exponentiation algorithm performance regarding to the state of the art. The compact representation, called collapsed representation, represents a point by its x coordinate and the trace of y/x. The point exponentiation is the most expensive operation over EC points, thus its optimization was prioritized and the proposed method introduces a significant improvement in the communication latency. For point addition, an algorithm that avoids the y coordinate computing was not implemented, since it does not result in performance improving. For this reason, an improving was proposed to perform the calculation of the y coordinate using the x coordinate and the trace of y/x.

In order to achieve the necessary algorithms to perform the arithmetic in the proposed processor, the EC properties, including the mathematical entities which support it, were carefully analyzed. From this analysis it was concluded that it is possible to define an EC system over a generic field but the only fields with practical interest are the finite fields (or Galois fields), namely prime fields $GF(p)$ and binary extension fields $GF(2^k)$, since in real implementations the capability of representing field elements is limited to a finite number of elements. From these finite fields, the fields $GF(2^k)$ revealed to be more suitable for hardware implementation. One of the reasons is that the field elements can be represented using all the possible representations the binary vector allows and, more important, the operations over this field can be accomplished using less hardware resources. For example the addition operation is computed by independent bitwise XOR operations, which can be very efficiently implemented in hardware, while an addition over $GF(p)$ would require a carry propagation chain. It was also concluded that, although the EC is supported over a field, the points belonging to this entity constitute a group, which means that only one operation can be defined for an EC. The defined operation is the addition and can be computed with several field operations. This means that the point exponentiation (or point multiplication) is obtained through successive additions. This operation holds the security of the EC cryptosystem, since it is not computationally invertible. This operation is different from other asymmetric key cryptosystems currently being used, such as the RSA cryptosystem, which is supported over field operations, namely prime numbers multiplication. Another analysis issue arises from the field elements is the basis used to represent them. This basis can be canonical, normal or polynomial. Canonical and normal basis are normally used together. The polynomial basis is the one which better suits the objective of this work and its original contributions, namely the calculus of the trace of a field element that facilitates the use of the collapsed representation for an EC point. The trace in this basis has few dependencies, 2 in the case of this thesis implementation example, while in normal basis has k dependencies, with k the field size. This basis also improves the generality of the field size. The trace of an element in polynomial basis can be computed with the minimum of 2 input XOR while for a normal basis this could be calculated with a minimum of k input XOR gates for a field $GF(2^k)$. Another reason to adopt the polynomial basis is that there is always a basis for each k, while optimal normal basis don't have this property. Thus, the polynomial representation is more generic. All the representations require the existence of an irreducible polynomial of degree k.

Several methods to perform field operations, namely field multiplication and inversion/division were researched in order to integrate the processor architecture. For the field multiplication a very efficient multiplier based in multiplications by powers of the polynomial x was selected. Since the used irreducible polynomial is a pentanomial, only three 2 input XOR gates are required to perform a multiplication by a polynomial x. Another advantage of the used multiplication structure is that it can be easily parallelized for any k. An algorithm based on the *Extended Euclidean Algorithm* was used for field inversion. The use of this algorithm suggests a 1.38 times improvement when compared with the algorithm based on the recursive computation of multiplications. The analysis of this operation allows to conclude that the field inversion is the more time consuming operation to be performed over a field. For this reason, methods to mitigate the use of inversion operations while computing the EC exponentiation were studied. These methods are supported by projective representations of the EC points. This kind of representation introduces a third coordinate used to represent an EC point which stores all the information referred to the inversion procedure. It allows a final inversion at the end of the algorithm using this information to be equivalent to the inversion in each point addition. Several projective representations were introduced, but one of them, the standard projective representation, allows to compute point exponentiation x coordinate using only the x coordinate of the input point being more appropriate to the proposed processor.

In order to perform a comparative evaluation of the proposed FPGA processor characteristics, a software and an ASIC technology implementations were performed. The designed FPGA processor supports operations over a generic field $GF(2^k)$, but was implemented for a field with $k = 163$. The proposed arithmetic units do not compromise the generality of the field, because each unit can be easily scalable to any k. The resulting processor requires $10,488$ slices in a total of $15,360$ and computes a point multiplication in $144.3\ \mu s$ and a point addition in $1.024\ \mu s$. Point multiplication allows a maximum throughput of $1\ Mbit/s$ while point addition allows $15\ Mbit/s$. The ASIC implementation shares the same structure as the FPGA implementation, since it was supported by the same VHDL description. In this case, the comparison is rather difficult, since technologies with different characteristics are used. Nevertheless, it is possible to distinguish that the proposed design has very demanding routing requirements, imposing 31% of the critical path. In comparison with the software solution, the proposed FPGA design has a speed up is of $2,858$ for point multiplication, in comparison with the MicroBlaze Xilinx soft-processor, and a speedup of 387 when compared with the general purpose Intel processor. For point addition the speedup is 143 comparing with MicroBlaze, and 21 comparing with the Intel general purpose processor. These results were expected since all the options, namely the choice of the field to support the system, were made regarding hardware implementations and parallelization. The differences between the hardware and software solutions are also explained by the fact that some operations, namely the squaring operation, can be efficiently performed in hardware, with the manipulation of individual bits in a non-regular manner. This manipulation can not be efficiently performed in software.

The design was shown to successfully support a host system granting all the EC operations. An application was implemented using a prototyping board and real protocols were used to test the proposed processor. This prototype allowed us to conclude that, when conjugated with fast communication interfaces, the EC processor can provide very interesting throughputs.

6.2 Future Work

EC cryptosystems is a current and very important research field, not only on efficient implementations but also, for example, in security attacks. This last subject can be used to inspire some of the future work, where the attack possibilities could be analyzed in order to identify weaknesses and adapt the design with more secure structures.

From this thesis other issues related with the architecture suggest possible improvements. The ASIC implementation suggests that the design has complex routing requirements. Thus, other structures can be researched, not only to reduce the number of gates needed to implement the design but also to reduce the routing complexity, improving the global performance. New functionalities can also be researched. One of these functionalities, which is not in this thesis scope, is the mapping between messages and EC points. This thesis introduces a method based on a trial search of the mapping, which assumes that some of the EC coordinate bits are not used to transfer data, but are only used to assure that the representation of the data can lead to an EC point. The probability for this method failing decreases if more redundancy (bits with no data) are used. Since, this method reduces the throughput performance of the system, a more efficient method that guarantees the mapping with minimal redundancy can be researched. It can be also of interest to research a method to avoid the computation of the y coordinates when performing point addition, as it was accomplished for point multiplication. This would permit to reduce the number of divisions used in the system, increasing the performance of the ECC processor.

The utilization of a different basis to represent field elements may also be an interesting research topic. The implementation of these solutions may also be performed and optimized in order to clearly distinguish which advantages exist and how these advantages are translated into an objective performance metric. Hybrid basis methods can be researched in order to exploit the advantages of each representation.

Finally, solutions over $GF(p)$ can be researched because, since it is important to adapt general purpose processors to operate over $GF(2^k)$, it would also be important to adapt dedicated hardware structures to $GF(p)$ allowing all the protocols to be implemented in the different solutions. The field $GF(p)$ is better suited for software solutions, since it is able to use the integer arithmetic units provided by these processors. Although, some devices, such as FPGAs, provide structures to improve the carry propagation which could be exploited. The utilization of $GF(p)$ also introduces the possibility to use different numbering systems, such as the Residue Number System (RNS) or the Logarithmic Number System (LNS).

Bibliography

[1] N. Koblitz, “Elliptic curve cryptosystems,” Mathematics of Computation, vol. 48, no. 177, pp. 203–209, January 1987.

[2] V. Miller, “Uses of elliptic curves in cryptography, Advances in Cryptology, CRYPTO 1985,” Lecture Notes in Computer Science, vol. 218, pp. 417–426, 1986.

[3] K. Leung, K. Ma, W. Wong, and P. Leong, “FPGA implementation of a microcoded elliptic curve cryptographicprocessor,” IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 68–76, 2000.

[4] T. Kerins, E. Popovici, W. Marnane, and P. Fitzpatrick, “Fully Parameterizable Elliptic Curve Cryptography Processor over GF(2^m),” 12th International Conference on Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream, FPL, pp. 750–759, 2002.

[5] J. Park, J. Hwang, and Y. Kim, “FPGA and ASIC Implementation of ECC Processor for Security on Medical Embedded System,” Information Technology and Applications, 2005. ICITA 2005. Third International Conference on, vol. 2, 2005.

[6] Y. Wang, X. Dong, and Z. Tian, “FPGA Based Design of Elliptic Curve Cryptography Coprocessor,” Third International Conference on Natural Computation, ICNC 2007, vol. 5, pp. 185–189, August 2007.

[7] K. Sakiyama, E. De Mulder, B. Preneel, and I. Verbauwhede, “A parallel processing hardware architecture for elliptic curve cryptosystems,” Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pages III–904–III–907, vol. 95, pp. 96–97, 2006.

[8] M. Rosing, Implementing elliptic curve cryptography. Manning Publications Co. Greenwich, CT, USA, 1999.

[9] S. Okada, N. Torii, K. Itoh, and M. Takenaka, “Implementation of elliptic curve cryptographic coprocessor over GF(2^m) on an FPGA,” Cryptographic Hardware and Embedded Systems (CHES), pp. 25–40, 2000.

[10] F. Rodriguez-Henriquez, N. Saqib, A. Perez, and Ç. Koç, Cryptographic Algorithms on Reconfigurable Hardware. New York, NY: Springer, 2007.

[11] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," Advances in Cryptology CRYPTO 99, 19th Annual International Cryptology Conference Proceedings, vol. 1666, pp. 388–397, 1999.

[12] W. Stallings, Cryptography and network security. Upper Saddle River, NJ: Prentice Hall, 2003.

[13] G. Seroussi, "Compact Representation of Elliptic Curve Points over F_{2^n}," HP Laboratories Technical Report, September 1998.

[14] K. Makita, Y. Nogami, and T. Sugimura, "Generating prime degree irreducible polynomials by using irreducible all-one polynomial over F_2," Electronics and Communications in Japan(Part III Fundamental Electronic Science), vol. 88, no. 7, pp. 23–32, 2005.

[15] F. NIST, "186-2, Digital Signature Standard (DSS)," January 2000.

[16] B. Sunar and K. Koç, "An Efficient Optimal Normal Basis Type II Multiplier," IEEE Transactions on Computers, vol. 50, no. 1, pp. 83–87, 2001.

[17] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in GF(2^m)," IEEE Transactions on Computers, vol. 42, no. 8, pp. 1010–1015, August 1993.

[18] E. Oswald, "Introduction to Elliptic Curve Cryptography," Institute for Applied Information Processing and Communication, 2005.

[19] C. Chin-Long, "Formulas for the solutions of quadratic equations over GF(2^m)," IEEE Transactions on Information Theory, vol. 28, no. 5, pp. 792–794, September 1982.

[20] C. Wang, T. Troung, H. Shao, L. Deutsch, J. Omura, and I. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," IEEE Transactions on Computers, vol. c34, no. 8, pp. 709–717, August 1985.

[21] Ç. Koç and T. Acar, "Montgomery Multiplication in $GF(2^k)$," Designs, Codes and Cryptography, vol. 14, no. 1, pp. 57–69, 1998.

[22] N. Mentens, S. Ors, B. Preneel, and J. Vandewalle, "An FPGA Implementation of an Elliptic Curve Processor over GF (2^m)," ACM Great lakes Symposium on VLSI, LSVLSI, Proceedings, pp. 454–457, 2004.

[23] N. Saqib, F. Rodriguez-Henriquez, and A. Diaz-Perez, "A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2^m)," Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, April 2004.

[24] S. Moon, J. Park, and Y. Lee, "Fast VLSI arithmetic algorithms for high-security elliptic curve cryptographic applications," IEEE Transactions on Consumer Electronics, vol. 47, no. 3, pp. 700–708, August 2001.

[25] M. Ernst, B. Henhapl, S. Klupsch, and S. Huss, "FPGA based hardware acceleration for elliptic curve public key cryptosystems," The Journal of Systems & Software, vol. 70, no. 3, pp. 299–313, 2004.

[26] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich, "Reconfigurable implementation of elliptic curve crypto algorithms," International Parallel and Distributed Processing Symposium, IPDPS, Proceedings, pp. 157–164, 2002.

[27] J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curves over GF(2^m) without Precomputation," Cryptographic Hardware and Embedded Systems, First International Workshop, CHES 99, Proceedings, vol. 1717, pp. 316–327, August 1999.

[28] I. Blake, G. Seroussi, N. Smart, and J. Cassels, Advances in Elliptic Curve Cryptography. Cambridge University Press New York, NY, USA, 2005.

[29] Xilinx, Inc., Virtex-4 FPGA User Guide, 2008, http://www.xilinx.com/support/documentation/user_guides/ug070.pdf.

[30] ——, MicroBlaze Processor Reference Guide, 2007, http://www.xilinx.com/support/documentation/sw_manuals/edk92i_mb_ref_guide.pdf.

[31] Maxim Integrated Circuits, Inc., True RS-232 Transceiver MAX3222E Manual, 2007, http://datasheets.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf.

[32] Annapolis Micro Systems, Inc., Wildcard 4 Manual, 2007, http://www.annapmicro.com/manuals/wildcardii_4_refmanual_v3.6.pdf.

A

Hardware Resources Description

Contents

To perform the software test and construct the EC prototype, several components belonging to a Wildcard 4™ Board, MicroBlaze™ Soft Core as well as PC utilities were used. These components and the developed processor interface are described in detail in the following sections.

A.1 Elliptic Curve Processor Interface

The EC processor and its associated signals are depicted in Figure A.1.

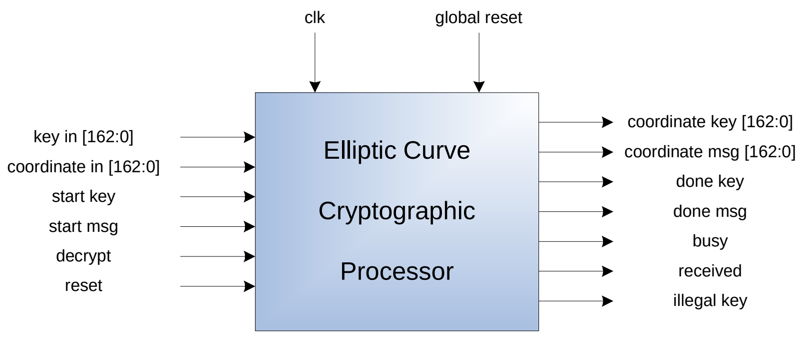

Figure A.1: Elliptic Curve Processor and Associated Signals

The signals presented in Figure A.1 have the following purpose:

- Inputs:
  - *clk*: the processor is a synchronous systems towards this clock signal.
  - *global reset*: it is the initialization reset signal which collocates the system on an idle state.
  - *key in [162:0]*: input scalar involved in point multiplication.
  - *coordinate in [162:0]*: input collapsed coordinate which may represent the point involved in point multiplication or in point addition.
  - *start key*: signal which communicates that the *key in* and *coordinate in* signals are updated and it is possible to begin a new point multiplication.
  - *start msg*: signal which communicates that the *coordinate in* signal is updated and it is possible to begin a new point addition with the processor inside stored point.
  - *decrypt*: if this signal is activated, when starting a point addition procedure it will be performed $A - B$, with A the input coordinate and B the stored coordinate. Otherwise, it will be performed $A + B$. Considering A a message and B a secret, it will correspond to message decrypting and encrypting, respectively.
  - *reset*: when performing point addition, the processor can efficiently add several input points A_i to the stored point B, performing the point B initialization operations only

once. When the set of points to be added ends, this reset signal may be activated in order to the processor returns to an idle state.

- Outputs:
  - *coordinate key [162:0]*: resulting collapsed coordinate from point multiplication. It may represent a public key or a secret, accordingly with the input operands.
  - *coordinate msg [162:0]*: resulting collapsed coordinate from point addition. It may represent encrypted or decrypted messages, accordingly with the input operands and the *decrypt* signal.
  - *done key*: when active informs that there is a new point multiplication result available in the output *coordinate key*.
  - *done msg*: when active informs that there is a new point addition result available in the output *coordinate msg*.
  - *busy*: when active informs that the processor is processing.
  - *received*: when active informs that the input *coordinate in* signal was read and a new point addition began, thus the *coordinate in* signal can be updated with another input coordinate. This signal is useful to decrease the latency in the communication of new points since it can be asked other input while a point addition is being performed.
  - *illegal key*: this signal is active when point multiplication result is illegal to represent keys, namely it is null.

A.2 Wildcard 4^{TM} Board

The Wildcard 4 is a processing Personal Computer Memory Card International Association (PCMCIA) card which contains a Xilinx Virtex 4 FPGA, model XC4VSX35, that can be used as a PC extension. The schematic layout of this card is presented in Figure A.2 [32].

This prototyping card comes with drivers for Windows 2000, complete VHDL models to easily interact with the FPGA surrounding components, and an API to use the card functionality in a software application.

Not all card components are necessary. Only the following is used:

- *Cardbus* and *Cardbus Controller*: it is through these components that the communication to the host system is performed. The FPGA programming is also performed through these components. The *Cardbus Controller* also provides a LAD bus which allows to easily interchange data between the host and the FPGA.
- *Clock Generator*: This component is a programmable frequency clock provider. This component allows to feed the FPGA with a clock signal at the frequency defined by the user.

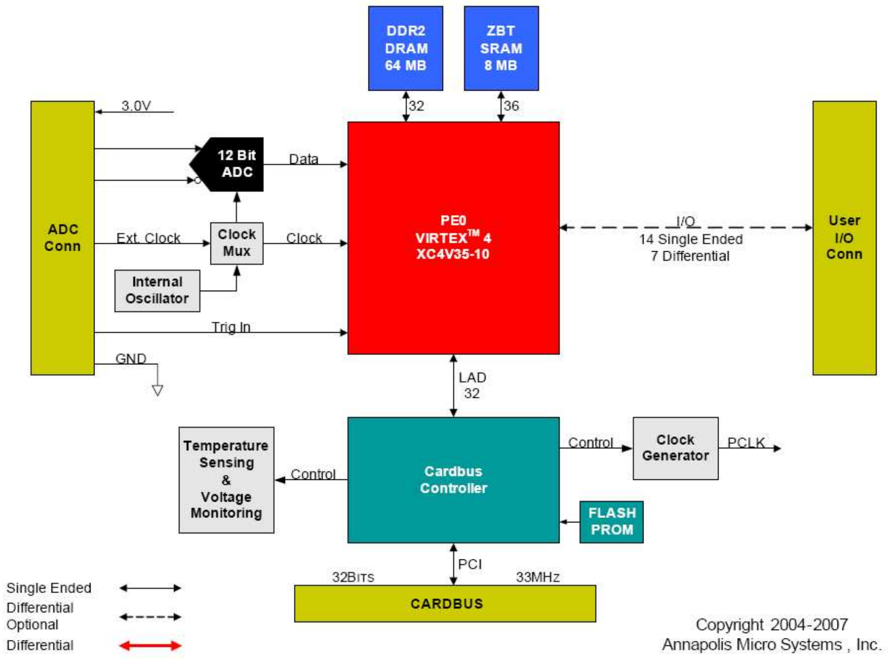

Figure A.2: Wildcard 4 Organization

This component programming is performed directly by the host system through the *Cardbus Controller*.

- *I/O connector*: This corresponds to the general Input/Output (I/0) pins available in the card. It will be useful to perform the communication with a remote terminal.

A.3 MicroBlaze^{TM} Soft Core

The MicroBlaze is a Reduced Instruction Set Computer (RISC) reconfigurable processor optimized to be implemented in Xilinx FPGA [30]. This processor allows the configuration of the memory size, the existence or not of memory cache, the existence or not of floating point unit, and the extension with several peripherals.

The possibility of extending the processor with several available peripherals is one of the advantages of using this processor, reducing the design effort. These peripherals can be easily configured and accessed in software. The address space contains two buses: the Local Memory Bus (LMB) and the On-Chip Peripheral Bus (OPB). The first one connects both instruction and data memory and the second one connects to the extension peripherals. Peripheral access is easily done with memory read and write instructions.

For the proposed application, three peripherals are used: *i)* two GPIO peripherals and *ii)* a UART Lite to support the RS-232 communication standard.

GPIO Peripheral

Outside the processor, this peripheral provides an output as a register, clock synchronous, and it is possible to write to this peripheral as a register too. From the software point of view, a simple write/read routine is sufficient to update the output register or to use the value in the input register. Although, an address can only be configured to write or to read. It is possible to configure the peripheral on-the-fly by software, but it brings an extra overhead if several writes and reads are performed. It is possible to mitigate these problems using other functionality of this peripheral, namely the possibility of activating two channels. With this, one channel can be configured to read and the other one to write, using the same peripheral.

The bit size of the registers used in this peripheral can be an integer up to 32. This peripheral provides other interfaces outside the processor, namely I/O buses and tri-state buses. These interfaces were not required.

UART Lite Peripheral

This peripheral provides a simple exterior communication using a serial stream that can support standards, such as RS-232. It has a First-In, First-Out (FIFO) queue, were the data to be sent through the serial stream is written and the received data is stored until a software read.

Outside the processor, this peripheral provides one TX (Transmit) bit and one RX (Receive) bit. The transmission can be configured through the following parameters:

- Number of bits per frame from 5 up to 8;
- Utilization or not of a parity bit;
- The parity type: odd or even;
- Bit stream throughput from 110 up to 921600 bits per second.

A.4 Host System

The host system is composed by a PC extended with a Wildcard 4, and connected through a PCI bus. The system is designed so that all EC arithmetic operations are performed by the EC processor implemented in the FPGA.

This system only provides the necessary data for the EC processor, namely private keys and plain messages.

A.5 Remote System

The remote system is composed by a PC connected to the MicroBlaze through a RS-232 connection. This system is able to compute all the functionality of the conjunction of the host system, Wildcard 4 and the MicroBlaze inside it, thus it can interact with them with the necessary data. For this task, the libraries developed in Section 5.1 are used. This system provides public keys and encrypted data, in order to establish a secure channel between the MicroBlaze and itself.

B

Transceiver Electrical Schematic

The proposed prototype contains a transceiver used to convert the FPGA tension level ($3.3\ V$) to the RS-232 communication port level ($5\ V$). This transceiver is supported by the integrated circuit MAX3222E from Maxim Integrated Circuits, Inc. The used electrical schematic follows a typical operating circuit suggested in the MAX3222E data sheet, and is represented in Figure B.1. This circuit is supplied with $5\ V$ by an USB to RS-232 adapter.

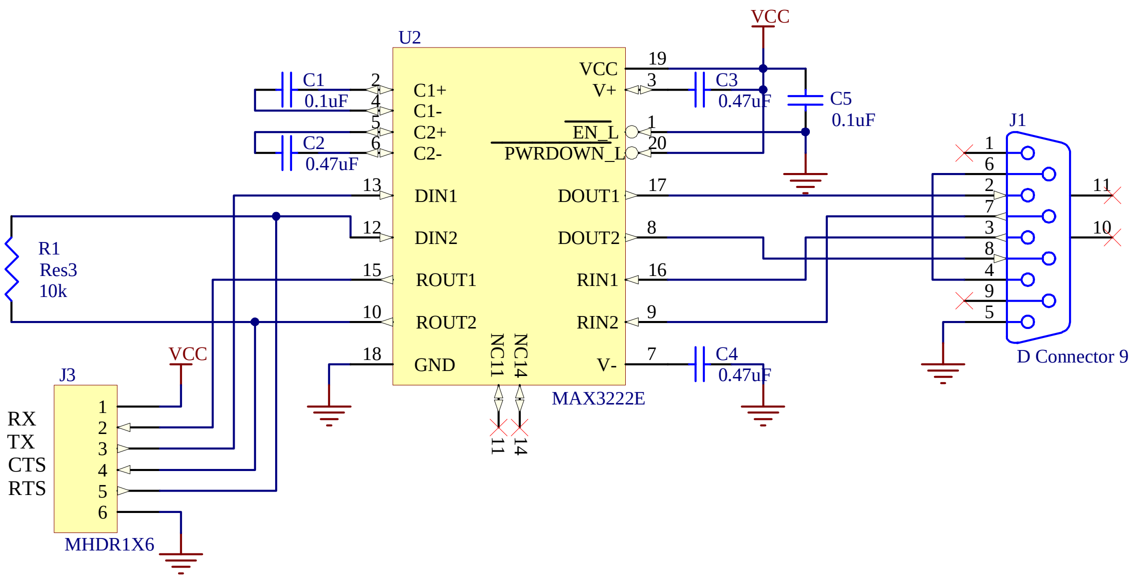

Figure B.1: Electrical Schematic of the Transceiver