PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELLIPTIC CURVE CRYPTOGRAPHY IN HARDWARE FOR SECURE SYSTEMS: A MULTI-USE RECONFIGURABLE SOFT IP

BRUNO FIN FERREIRA

Dissertation presented as a partial requirement to obtain the Master's Degree in Computer Science

Advisor: Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre
March 2014

Dados Internacionais de Catalogação na Publicação (CIP)

F383e Ferreira, Bruno Fin
Elliptic curve cryptography in hardware for secure systems : a multi-use reconfigurable soft ip / Bruno Fin Ferreira. - Porto Alegre, 2014.
74 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Ney Laert Vilar Calazans.

1. Informática. 2. Segurança de Dados. 3. Criptografia (Computação). 4. Segurança em Redes de Computadores. 5. Arquitetura de Computador. I. Calazans, Ney Laert Vilar Calazans. II. Título.

CDD 005.8

Ficha Catalográfica elaborada pelo Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada *"Elliptic Curve Cryptography in Hardware for Secure Systems: A Multi-Use Reconfigurable Soft IP"* apresentada por Bruno Fin Ferreira como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 11/03/2014 pela Comissão Examinadora:

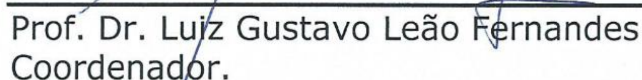
Prof. Dr. Ney Laert Vilar Calazans – Orientador — PPGCC/PUCRS

Prof. Dr. Alexandre de Morais Amory – PPGCC/PUCRS

Prof. Dr. Avelino Francisco Zorzo – PPGCC/PUCRS

Prof. Dr. Diego de Freitas Aranha – UNB

Homologada em 01/08/2014, conforme Ata No. 015 pela Comissão Coordenadora.



Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320–3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

ACKNOWLEDGMENTS

Thanks to my parents, Edite Ines Fin and João José Ferreira Filho, for their support, dedication and patience in educating me and supporting all my decisions. Without their support and motivation, I would not have done this work.

Thanks to my advisor, Prof. Dr. Ney Calazans for the opportunities given to me, for the confidence deposited in me and for all his support in the development of this work.

Thanks to all my friends and everyone who contributed in somehow in the development of this work.

CRIPTOGRAFIA POR CURVAS ELÍPTICAS EM HARDWARE PARA SISTEMAS SEGUROS: UM NÚCLEO IP RECONFIGURÁVEL DE MULTIUSO

RESUMO

Nos últimos anos, a indústria tem desenvolvido e colocado no mercado uma grande quantidade de dispositivos que são capazes de acessar a Internet e outras redes. Isso está tornando mais fácil enviar, receber, compartilhar e guardar todo tipo de informação a qualquer momento de qualquer lugar. Assim, há uma enorme quantidade de informações importantes passando pela Internet, mas há também entidades e/ou indivíduos maliciosos tentando capturar essas informações para roubar ou explorar isso visando obter ganhos financeiros ou estratégicos, ou causar algum dano a outras pessoas. Apesar de existir muitas maneiras de proteger tais informações, a mais relevante é o uso de criptografia. Há muitos algoritmos criptográficos em uso atualmente, tais como DES, 3DES, AES e RSA, que normalmente são implementados em software. Eles atingem baixo desempenho e proveem baixos níveis de segurança para muitas aplicações. Portanto, é necessário criar soluções que disponibilizem maiores níveis de segurança e ao mesmo tempo melhorem o desempenho de criptografar. Este trabalho propõe um sistema de comunicação seguro que pode ser integrado a dispositivos embarcados ou computadores. O sistema de comunicação seguro proposto e desenvolvido neste trabalho é baseado em Criptografia por Curvas Elípticas (ECC), um esquema de criptografia que tem sido estudado e melhorado na última década por muitos pesquisadores, e é indicado como um dos algoritmos de criptografia dos mais seguros. Este trabalho descreve em detalhes a implementação das operações do ECC em hardware, com alvo em prover maior desempenho do que a maioria dos trabalhos disponíveis na literatura. Outro objetivo do trabalho é que mesmo sistemas embarcados críticos possam usar o esquema proposto para criar sistemas de comunicação seguros. Este trabalho utilizou o estado da arte operações de ECC para gerar implementações em hardware. O resultado é um núcleo de propriedade intelectual (IP) flexível para ECC que pode ser sintetizado para FPGAs ou ASICs. A validação deste núcleo incluiu o desenvolvimento de um sistema de comunicação completo que pode criar um enlace de comunicação segura entre dois computadores ou dispositivos similares usando ECC para criptografar todas as informações trocadas. O núcleo IP de ECC dá suporte a qualquer uma das 5 curvas elípticas de Koblitz recomendadas pelo Instituto Nacional de Padrões e Tecnologia (NIST) e aos Padrões para Grupo de Criptografia Eficiente (SECG). Entretanto, o núcleo IP pode também ser facilmente adaptado para dar suporte a outras curvas elípticas. Um sistema de comunicação segura foi desenvolvido, implementado e prototipado em uma placa de desenvolvimento com FPGA Virtex 5 da Xilinx. Além disso, o trabalho demonstra as vantagens e os ganhos de desempenho obtidos quando comparado com implementações em software de sistemas similares.

Palavras Chave: comunicação segura, criptografia, ECC, núcleo IP, implementação em hardware, FPGA.

ELLIPTIC CURVE CRYPTOGRAPHY IN HARDWARE FOR SECURE SYSTEMS: A MULTI-USE RECONFIGURABLE SOFT IP

ABSTRACT

In the last years, the industry has developed and put in the market a plethora of electronic devices that are able to access the Internet and other networks. This is making easier to send, receive, share and store all types of information at any moment, from anywhere. Thus, there is a huge amount of important information crossing the Internet and there are malicious entities and/or individuals trying to capture this information to steal or exploit it in order to obtain financial or strategic gains or to cause damage to other people. There are many ways to protect such information, the most relevant of which is the use of cryptography. There are many cryptographic algorithms in use nowadays, such as DES, 3DES, AES and RSA, which are usually implemented in software. This leads to low performance, and low security levels for several applications. Therefore, it is necessary to create solutions that provide higher security levels and that at the same time improve cryptography performance. This work proposes and presents a secure communication system that can be integrated to embedded devices or computers. The proposed secure communication system developed in this work is based on Elliptic Curve Cryptography (ECC), which is a cryptography scheme that has being studied and improved over the last decade by many researchers and is indicated as one of the most secure among cryptographic algorithms. This work describes in detail the implementation of ECC operations in hardware, trying to provide higher performance than most works available in the literature. Another goal of the work is that even critical embedded systems could use the proposed scheme to build a secure communication system. This work capitalizes on the state of the art in ECC operations and implements these in hardware. The result is a reconfigurable soft IP core for ECC, which can be synthesized for either FPGAs or ASICs. The validation of the soft core comprises the development of a complete communication system that can create a secure communication link between two computers or similar devices using ECC to encrypt all exchanged information. The soft IP core for ECC operations supports any of the five Koblitz curves recommended by the National Institute of Standards and Technology (NIST) and the Standards for Efficient Cryptography Group (SECG). However, the IP core can also be easily adapted to support other elliptic curves. An overall secure communication system was developed, implemented and prototyped in a development board with a Xilinx Virtex 5 FPGA. Furthermore, the work demonstrates the advantages and gains in performance when compared to software implementations of similar systems.

Keywords: secure communication, cryptography, ECC, soft IP core, hardware implementation, FPGA.

LIST OF FIGURES

Figure 1. Hierarchy of operations in ECC. 24

Figure 2. Simplified view of the El Gamal cryptography algorithm...........................24

Figure 3. Functional diagram of the software proposed and implemented in the scope of this work...25

Figure 4. Software architecture operating in server mode.......................................26

Figure 5. Software architecture operating in client mode. 26

Figure 6. Software functional sequence diagram. 27

Figure 7. Test scenario 1, local test without "ecc_connection". 27

Figure 8. Test scenario 2, local test with "ecc_connection" not encrypting. 28

Figure 9. Test scenario 3, local test with "ecc_connection" encrypting. 28

Figure 10. Test scenario 4, testing network communication....................................28

Figure 11. Test scenario 5, testing network communication with "ecc_connection" not encrypting. 28

Figure 12. Test scenario 6, testing network communication with "ecc_connection" encrypting. 29

Figure 13. Overall architecture of the context for the proposed work. 31

Figure 14. Li et al. [LI08] results achieved in device utilization. 33

Figure 15. Li et al. [LI08] speedup achieved in HW vs. SW implementations..........34

Figure 16. Li et al. [LI08] comparison with related works...34

Figure 17. Results obtained by Järvinen [JÄR11]...35

Figure 18. Overall architecture of the proposed communication system. 37

Figure 19. Development platform HTG-V5-PCIE-330 of HiTech Global..................38

Figure 20. Overall component hierarchy in the hardware datapath of the proposed ECC soft IP. 40

Figure 21. Squarer module interface. 41

Figure 22. Steps to square a number over binary finite fields. 41

Figure 23. Multiplier module interface. 42

Figure 24.Datapath of the generic binary finite field multiplier. 42

Figure 25. The finite field divider module interface. 43

Figure 26. Point adder module interface. 47

Figure 27. Modules that compose the point adder. 48

Figure 28. Module interface of negate point. 48

Figure 29. Module interface of the point multiplier. 48

Figure 30. Example of module composition to create the point multiplier. 49
Figure 31. Module interface of the key generator. 49
Figure 32. Composition of modules to implement the (public) key generation. 49
Figure 33. Module interface of the ECC encrypter. 50
Figure 34. Composition of modules to implement the ECC encrypter. 50
Figure 35. Module interface of the ECC decrypter. 51
Figure 36. Composition of modules to implement the ECC decrypter. 51
Figure 37. Simulation of the encrypter module. 52
Figure 38. Simulation of the decrypter module. 52
Figure 39. Total power consumption for ECC ASIC implementations. 55
Figure 40.Hardware architecture of the secure communication system. 59
Figure 41. IPv4 Ethernet frame using the TCP protocol. 61
Figure 42. Detailed information of the IPv4 header. 62
Figure 43. Detailed information of the TCP header. 62
Figure 44. Ethernet frame to configure the frame filter. 62
Figure 45. Detailed datapath of encrypter and decrypter interfaces. 63
Figure 46. Module of the encrypter interface. 63
Figure 47. Splitting the frame data to encrypt. 64
Figure 48.Format of the data block to encrypt. 64
Figure 49. Ethernet frame received to be encrypted. 65
Figure 50. Encrypted Ethernet frame 1. 65
Figure 51. Encrypted Ethernet frame 2. 65
Figure 52. Module of the decrypter interface. 65
Figure 53. Basic design with only Ethernet modules and control. 66
Figure 54. Hardware prototyping environment for the secure communication system. 69

LIST OF TABLES

Table 1. A comparison of the security level of ECC and RSA public-key cryptography schemes according to the key size (in bits). 23

Table 2. Configuration of the used machines. 29

Table 3. Performance test results...29

Table 4. Decomposition of exponents for GF(2^{163})..44

Table 5. Decomposition of exponents for GF(2^{233})..44

Table 6. Decomposition of exponents for GF(2^{283})..45

Table 7. Decomposition of exponents for GF(2^{409})..45

Table 8. Decomposition of exponents for GF(2^{571})..46

Table 9. Initial synthesis results for the ECC encrypter and decrypter, targeting a Xilinx XC5VLX330T FPGA...52

Table 10. RELIC benchmark and modules execution time.....................................53

Table 11. Hardware and software comparison...54

Table 12. Modelsim simulation results: synthesis results for ISE 14.1 XST and Cadence Encounter for 65nm CMOS. 54

Table 13. Comparisons of performance with related works....................................56

Table 14. Synthesis results for a complete ECC soft IP core. 57

Table 15. Average timing for conducted simulations..67

Table 16. Synthesis results for the complete design. 68

Table 17.Estimating throughput of the secure communication link, in Mbits/s. 68

LIST OF ABBREVIATIONS

3DES	*Triple Data Encryption Standard*
AES	*Advanced Encryption Standard*
ASIC	*Application-Specific Integrated Circuit*
CMOS	*Complementary Metal-Oxide-Semiconductor*
DES	*Data Encryption Standard*
DSA	*Digital Signature Algorithm*
ECC	*Elliptic Curve Cryptography*
ECDSA	*Elliptic Curve Digital Signature Algorithm*
ECIES	*Elliptic Curve Integrated Encryption Scheme*
FIFO	*First In First Out*
GF	*Galois Fields*
IP	*Intellectual Property*
LUT	*Look-Up Table*
MAC	*Media Access Controller*
NIST	*National Institute of Standards and Technology*
PHY	*Physical Layer*
RSA	*Rivest Shamir Adleman*
SECG	*Standards for Efficient Cryptography Group*
TCP	*Transmission Control Protocol*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*

CONTENTS

1. INTRODUCTION **21**
1.1. Cryptography **21**
1.1.1. Symmetric and Asymmetric Cryptography 22
1.1.2. Elliptic Curve Cryptography 22
1.2. Software Experiment and its Limitations **24**
1.2.1. Test Scenarios and Results 27
1.3. Project Context, Objectives and Motivation **30**
1.4. Document Structure **31**

2. RELATED WORK **33**
2.1. Bednara et al. [BED02] **33**
2.2. Li et al. [LI08] **33**
2.3. Keller et al. [KEL09] **34**
2.4. Järvinen [JÄR11] **34**
2.5. Masoumi et al. [MAS12] **35**
2.6. Dias et al. [DIA13] **35**
2.7. Loi et al. [LOI13] **36**
2.8. Discussion of Related Work **36**

3. PROPOSED DESIGN **37**
3.1. Hardware Architecture **37**
3.2. Target Hardware **37**

4. SOFT IP CORE FOR ECC **39**
4.1. Hardware Architecture **39**
4.2. Hardware Modules for Operations over Finite Fields **40**
4.2.1. The Squarer 41
4.2.2. The Multiplier 41
4.2.3. The Divider 43
4.3. Hardware Modules for ECC Operations **46**
4.3.1. Point Adder 47
4.3.2. Negating a Point 48
4.3.3. Point Multiplier 48

4.3.4. Key Generator 49
4.3.5. Data Encrypter..50
4.3.6. Data Decrypter 50
4.4. Simulation, Validation and Synthesis 51
4.5. Exploring the Flexibility of the Soft IP Core for ECC...........................54
4.6. Comparing the Results with Related Work 55

5. SECURE COMMUNICATION SYSTEM 59
5.1. Hardware Architecture 59
5.2. Hardware Modules..60
5.2.1. Xilinx MAC Ethernet Wrapper...60
5.2.2. Main Controller 60
5.2.3. Frame Filter 61
5.2.4. Encrypter Interface 63
5.2.5. Decrypter Interface 65
5.3. Simulation, Validation and Synthesis 66
5.4. Hardware Prototyping 68
5.5. Comparing the Results with the Software Experiment 69

6. CONCLUSIONS 71
6.1. Applications and Viability...71
6.2. Future Works 71

REFERENCES...73

1. INTRODUCTION

In the last years, the industry of electronics and software has continually developed numerous electronic devices, which are able to access the Internet through some connection, creating the internet of things. In addition, many applications are taking advantage of this feature, allowing people to be online all the time. Thus, people are sending, receiving, sharing and storing more information through the internet than ever before. However, very few know how to protect their information adequately and they just trust that software applications or hardware devices to be already designed to protect their information. Besides, there are also companies that do not know how to implement security mechanisms to protect their products. Therefore, this scenario enables that malicious entities to capture and steal important information sent through the Internet or other computer networks. One solution to these problems is the use of cryptography to protect information. There exist many cryptographic systems, like the ones explained in Section 1.1. In addition, it is possible to find several applications and devices that transmit relevant data, such as banking transactions or military information. These already use some kind of cryptography. However, according to the desired security level, the use of cryptography can become prohibitive, due to the low performance or high energy cost of most implementations. Ideally, the choice of the cryptography system and its security level is always in accordance to the importance of the data to protect.

Almenares et al. [ALM13] present a research that shows the main problems related to secure wireless communication and the overheads associated to cryptography. They also show that the use of mobile devices to access the Internet are becoming greater than desktop and that mobile devices usually are more constrained in terms of computational power, energy and size. In that paper, the authors considered several cryptography systems implemented in software, such as AES, RSA, DSA and ECDSA, and compared them to determine their relative overhead, performance, energy consumption and efficiency. In the end, authors conclude that in some cases there is a high overhead. This is reducible or passible to optimization through the development of specific hardware for cryptography.

1.1. Cryptography

Cryptography is widely used to guarantee that information will not be exposed to everyone. Since the World War II, cryptography started to gain attention and some machines started to be created to encrypt, decrypt and break important information, like military secrets, which were so important that their knowledge could determine if one side wins or loses the war. Since then, cryptography has been studied in depth and many algorithms and protocols have been proposed. Cryptography comprises some basic definitions, such that the algorithm must be public and security is obtainable only through the used key. Paar et al. [PAA10] explore this concept in more detail. Cryptography algorithms must be public because if the security is based only on the algorithm, the entity that created the algorithm would be able to break any cipher. In this way, it would not be possible to use the same algorithm for other parts and if the algorithm was published then no system would be secure anymore. Therefore, the security of a cryptography system must be based only on the key

used. Thus, everybody can have access to the algorithm and test if it is really secure. However, the key must be kept secret and it also must be long enough to make unviable for someone to try to discover it by exhaustive search. Considering these basic definitions, a cryptography system must be very efficient and secure for its use be viable. The reason behind this is that if a system is very secure but it is also very slow, it can become unviable for many applications. On the other hand, if a system is very fast but it has low security level then it will not be useful for many applications. Therefore, all the cryptography algorithms aim to be very efficient and secure to fulfill the requirements of most applications. Symmetric or asymmetric algorithms implement these functionalities. The differences between these classes of algorithms are the target of the next Section.

1.1.1. Symmetric and Asymmetric Cryptography

The main differences between symmetric and asymmetric cryptography are in the algorithms and in the used keys. Basically, symmetric algorithms use the same algorithm and key for encryption of the plaintext and decryption of the cipher text, while asymmetric algorithms may use an algorithm and key for encryption and other algorithm and key for decryption. Symmetric algorithms were created and proposed first. Examples of these are the Data Encryption Standard (DES), the Triple Data Encryption Standard (3DES) and the Advanced Encryption Standard (AES), accepted as standards by the United States government. They are still widely used in several applications due to their combined requirements of area, performance and security level. It is also already known that some of these algorithms can be easily broken with powerful computers. After these, asymmetric algorithms started to be proposed, such as the Rivest-Shamir-Adleman (RSA) and Elliptic Curves Cryptography (ECC), and these started to be used for key exchange schemes in symmetric algorithms. Generally, comparing symmetric and asymmetric algorithms, the first can usually be implemented in a very efficient way, i.e. symmetric algorithms achieve higher performance than asymmetric algorithms for encryption and decryption of data. On the other hand, asymmetric algorithms can be optimized to fulfill more constraints of area and power when implemented in hardware. Thus, some applications may require an encryption system with performance of symmetric algorithms and with security level of asymmetric algorithms, resulting in a solution that uses both kinds of cryptography, such as key exchange through asymmetric algorithms and encryption with symmetric algorithms.

1.1.2. Elliptic Curve Cryptography

In the 1980s, Miller [MIL86] and Koblitz [KOB87] were the first to propose the use of elliptic curves for cryptography. Elliptic curve cryptography is a public-key scheme just like the widely used RSA algorithm. However, ECC has an advantage over RSA, the fact that for a similar security level it requires smaller keys than RSA. This means that ECC implementations require less memory and computational power to execute. For example, a key size of 163 and 571 bits in ECC has security level equivalent to a 1,024 and 15,360 bits respectively in RSA implementation, as Table 1 shows. Lauter [LAU04] also discusses the main advantages of ECC, mainly for mobile devices, and perform experiments in software showing that ECC is much faster than RSA.

Table 1. A comparison of the security level of ECC and RSA public-key cryptography schemes according to the key size (in bits).

Security Level	Key Sizes of ECC	Key Sizes of RSA
80	163	1024
128	283	3072
192	409	7680
256	571	15360

Due to these advantages, many standards for ECC appeared in the last decade. An example are those proposed by the National Institute of Standards and Technology (NIST) [NIST99], which recommends a set of parameters for several elliptic curves with different sizes. Another example are those suggested by the Standard for Efficient Cryptography Group (SECG) [BRO09], an industrial consortium dedicated to achieve interoperability among cryptography equipment. Martínez et al. also present a survey demonstrating the Elliptic Curve Integrated Encryption Scheme (ECIES), which is defined by many standards, and its advantages when compared to RSA. In short, ECC has been demonstrated to be more efficient than RSA.

In his book, Hankerson et al. [HAN04] present a guide to ECC and detail all the mathematical background, such as finite fields, modular arithmetic and elliptic curve arithmetic that supports the concepts behind ECC. Stated simply, the main ECC operation is the point multiplication, composed by a point addition and a finite field squaring operation. The hierarchy of ECC operations is illustrated in Figure 1. The point addition comprises finite field multiplication and inversion operators, which execute the most time-consuming operations in ECC. There are several studies demonstrating how to implement these operations efficiently in hardware or software and because of this, performance measurements are often based on the time taken to compute a single point multiplication in elliptic curves. This is the basic metric used in this work for comparing ECC implementations. The protocol level is not considered in this work because there are many different ones defined by several standards and this level can be implemented in software without significant effect on the cost of the overall cryptography performance.

ECC can be used over different representations and sizes of finite fields. In binary Galois Fields $GF(2^m)$, the polynomial bases and normal bases are the most used, but for ECC in hardware, polynomial bases are the most indicated, due to their efficiency in this kind of implementation as it is demonstrated by Deschamps et al [DES09]. Accordingly, this work only considers ECC over binary fields $GF(2^m)$, represented in polynomial bases. The implemented elliptic curves and parameters are the Koblitz curves recommended by NIST [NIST99], but the core proposed here can also support the choice of other curves and parameters. Points of an elliptic curve can be represented by affine coordinates, as (x, y), or projective coordinates, such as (x, y, z), and this is important, because some algorithms for scalar point multiplication, which use projective coordinates, are usually more efficient than those which employ affine coordinates. However, using projective coordinates requires a converter between these two kinds of representation to keep the interoperability among different systems and standards.

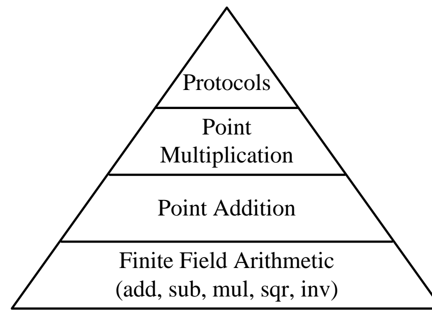


Figure 1. Hierarchy of operations in ECC.

At a high level of abstraction, ECC executes an algorithm composed by the elements and steps of Figure 2. The computation uses a pre-defined elliptic curve, such as one of Koblitz curves, to calculate the listed steps. All these steps are performed over GFs. Elements in uppercase are points of the pre-defined curve and the other elements are integers. According to the literature, the main representations used for ECC are GFs over prime fields, $GF(p^q)$ with p being a prime, and over the special case where p=2, called binary Galois Fields, $GF(2^m)$. The arithmetic operations are different in GF from the traditional arithmetic with integers or real numbers. Integer additions or subtractions in GFs are a same operation, which correspond to just an exclusive-or between binary representations of two integers. Point addition and point multiplication are also modular operations. Implementing these is one of the difficulties in ECC due its time consuming, as numbers are at least 160 bits wide.

- **Q** is a point of the curve, which is the public key;
- **d** is a random number, which is the private key, and $d \in [1, 2^m]$
- **P** is the point generator of the curve;
- **M** is the message to encrypt;
- **k** is a random number, generated for each message;
- **C1** and **C2** are points of the curve, which represent the encrypted message;

1. Key Generation:
   Q = d * P
2. Encryption:
   C1 = k * P
   C2 = M + k * Q
3. Decryption:
   M = C2 – d * C1

Figure 2. Simplified view of the El Gamal cryptography algorithm.

1.2. Software Experiment and its Limitations

During the study of ECC conducted by the author, a software application was developed to encrypt any communication between two computers. The ECC method was chosen because, as showed in Section 1.1, this is one of the currently most secure and efficient methods to have a secure communication. Some software libraries implement all mathematic functions necessary to encrypt any information. One of them is the RELIC [ARA12] that is an efficient library for cryptography, as demonstrated by Pigatto in his MSc dissertation [PIG12], where the author compared this library with other libraries and with other cryptography algorithms.

The developed software in this work was based on that developed by Pigatto [PIG12] that implements an ECC algorithm using the RELIC library and is able to generate public

and private keys, as well as encrypt and decrypt any computer file. This software was modified to enable the cryptography of a communication link between two computers. The aim of this new software was also to enable performance analysis of an encrypted communication with a software-only version of ECC. The code was written in C++, the communication link employs sockets and it is possible to select any TCP port in the communication.

Figure 3 shows the functional diagram of the developed software and illustrates how it works. Consider two computers that could also be mobile devices, connected by some network. The network can be the Internet, a local network or even some ad hoc network, and there are some application “x” running in computer “A” that needs to communicate with another application “y” that is in computer “B” and all the communication between these two applications must be encrypted. Each instance of the software developed, named “ecc_connection”, will respectively create a connection among the computers and with the local application. When the connection is created, the two instances of “ecc_connection” generate their private and public keys and then send to each other their public keys. After this, they are ready to receive application messages, encrypt these and send encrypted messages over the network to the other computer. When, the latter receives the encrypted message, its “ecc_connection” decrypts and delivers the message to the local application. Thus, the encryption and decryption processes are transparent for both applications “x” and “y”, i.e. both applications consider they are running in the same computer, and this transparency is an advantage of this implementation, because no changes are necessary at the application level.

Figure 3. Functional diagram of the software proposed and implemented in the scope of this work.

The “ecc_connection” software has two operation modes, server and client. Figure 4 and Figure 5 respectively illustrate the two modes. Both modes work the same way, the only difference is that one of the instances needs to be a server and the other needs to be a client to enable communication by sockets. This is a limitation of the current implementation that can be improved in a next version, but which does not influence in performance analysis. The software architecture comprises four independent threads and two FIFO buffers in both operation modes. Each instance has two sender threads and two receiver threads. Each pair is used for local communication with application “x” and for external communication with the other computer. Also, the FIFO buffers enable the communication between these threads. So, the “ecc_connection” enables a full duplex communication and there are no dependencies between its threads. The flow of messages is indicated by the arrows in the images and is always the same, but there are one option defined on application startup that indicates if the messages must be encrypted or not. The only difference is that in one case all messages will be encrypted and in the other they will be passed on without cryptography. This was implemented to enable the analysis of the overhead caused by this application without the cryptography process. Thus, it is possible to measure how much the cryptography alone decreases the link communication speed.

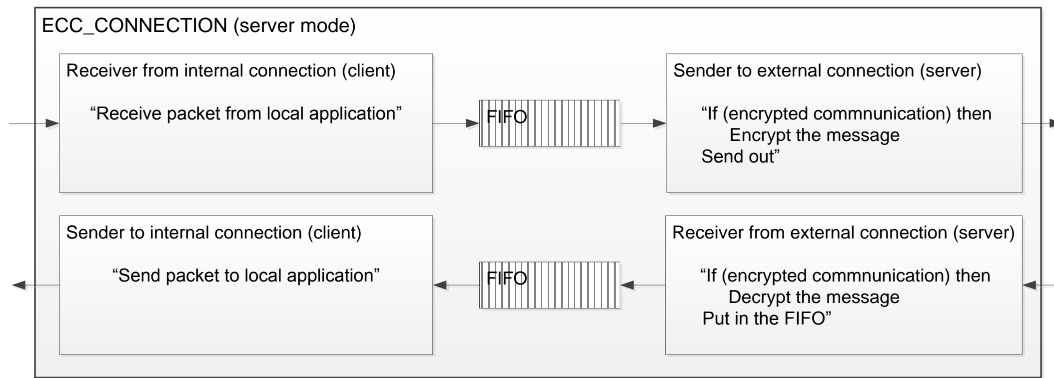


Figure 4. Software architecture operating in server mode.

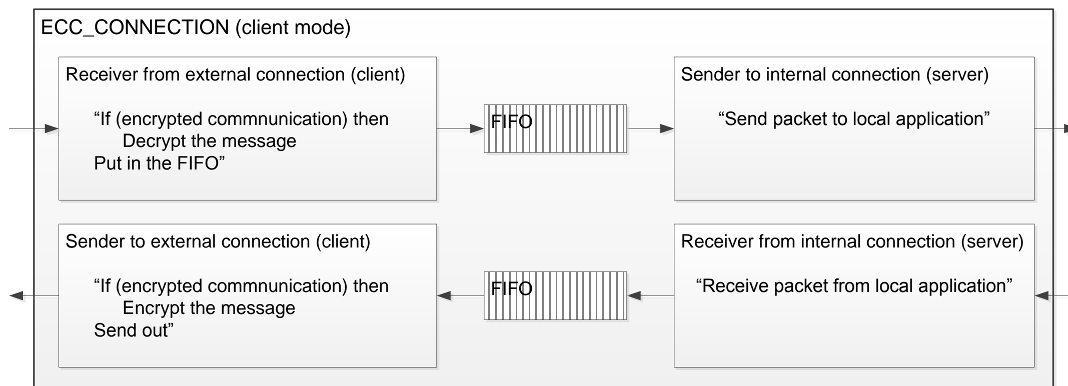


Figure 5. Software architecture operating in client mode.

Figure 6 shows the initial sequence of actions to create the connections between servers and clients processes. Initially, it is necessary to start "ecc_connection" in server mode. Then the client mode is started and requests connection to the server. After this, applications "x" and "y" are enabled to create their connections and use this encrypted link to communicate.

Considering these architectures, the "ecc_connection" can be fully parameterized to enable multiple analysis scenarios. It is possible to define the key sizes for public and private keys, the TCP port that will be used in each communication link, and if the communication will be encrypted or not. Next, Section 1.2.1 discusses some of the initial experiments executed and the associated results.

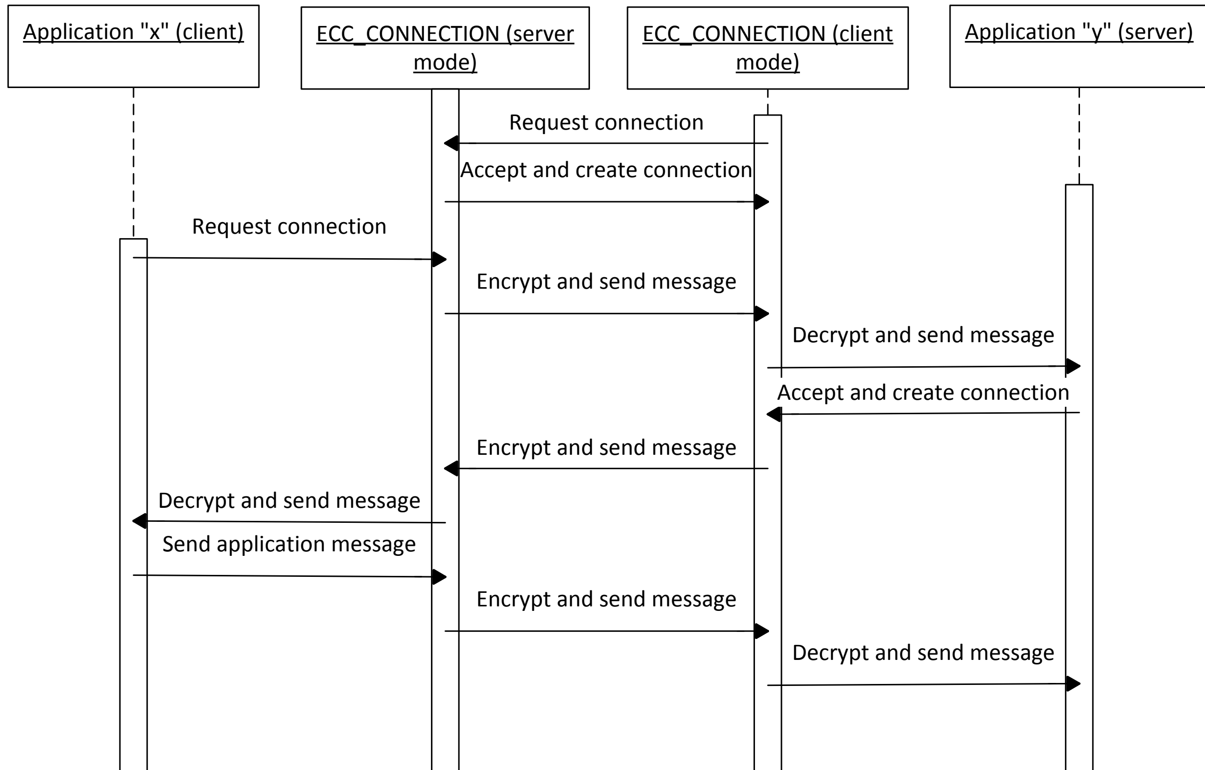


Figure 6. Software functional sequence diagram.

1.2.1. Test Scenarios and Results

The “ecc_connection” software was validated with 6 distinct test scenarios, illustrated from Figure 7 to Figure 12. All these test scenarios were executed in different computers, different networks and with different key sizes, to assist in performance analysis. The software Iperf [DUG12] served as a traffic generator to test the maximum speed of an Ethernet link. Iperf also operates with client and server modes. First, an Iperf instance must be started in server mode, and then another instance of the same software is started in client mode with the server IP address and the specified TCP port. Following this, the client instance connects with the server and starts data traffic, to measure the maximum speed that the link supports. The rest of this Section explains the main objective of each test scenario. Next, it presents and discusses the obtained results.

The test scenario 1, depicted in Figure 7, analyzes what is the maximum communication speed between two applications in the same computer, with a direct connection through some local TCP port.

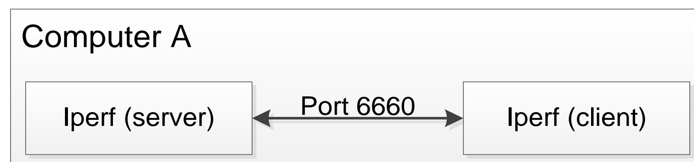


Figure 7. Test scenario 1, local test without "ecc_connection".

Figure 8 shows test scenario 2 that serves to analyze the overhead that the “ecc_connection” software causes compared to the test scenario 1, because it was not encrypting the communication, just forwarding packets.

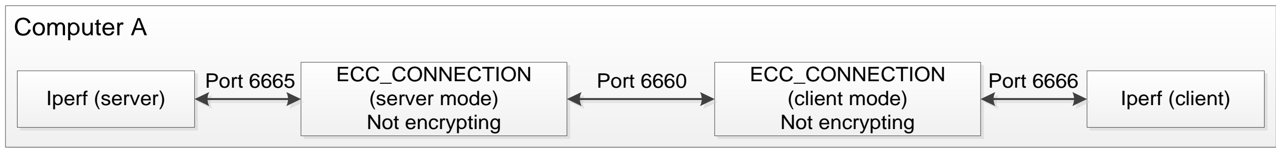


Figure 8. Test scenario 2, local test with "ecc_connection" not encrypting.

Test scenario 3, Figure 9, is the same as test scenario 2 but now “ecc_connection” encrypts the communication. With these three test scenarios, it is possible to analyze the results and know how much the performance degrades in a local communication encrypted by this “ecc_connection”.

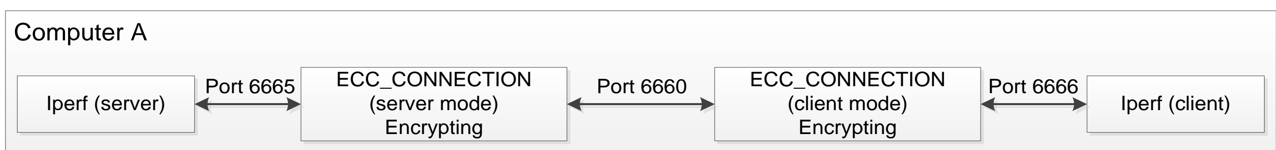


Figure 9. Test scenario 3, local test with "ecc_connection" encrypting.

The next three test scenarios are respectively similar to the previous three, but are performed between two computers and with different networks between them. Test scenario 4, in Figure 10, measures the maximum speed supported by the link between computers A and B. With the next test, it is possible to know how much performance is lost while running the system with the “ecc_connection” software interposed.

Figure 10. Test scenario 4, testing network communication.

Figure 11 depicts the test scenario 5 that performs the same previous test but with the “ecc_connection” forwarding packets. With it, it is possible to compute the overhead caused by this application.

Figure 11. Test scenario 5, testing network communication with "ecc_connection" not encrypting.

The last test scenario, displayed in Figure 12, is the most complete, because it is more like a real application scenario. With this scenario, it is possible to analyze how much losses occur when the communication between two computers is all encrypted by software, compared to the previous scenario results.

Figure 12. Test scenario 6, testing network communication with "ecc_connection" encrypting.

All test scenarios described above were run with different parameters, to assist in deciding what it is the major performance-limiting resource. The different parameters are related to which computers were used, the kind of network and key sizes employed to encrypt data. Table 2 shows the used computer configurations that helped analyze the obtained results. “VAIOs” are two identical laptops, “Computer Host” is another laptop on which the virtual machines are hosted, and GAPHL are two identical high performance desktop workstations from the GAPH laboratory.

Table 2. Configuration of the used machines.

Computer Name	VAIOs	Computer Host of Virtual Machines	Virtual Machines	GAPHL
Processor	Pentium (1 core)	i7 (4 cores)	1 or 2 cores	Xeon (12 cores)
Main Memory	1 GB	6 GB	1 GB	12 GB
Network Card	100 Mbits Full Duplex	1 Gbit Full Duplex	1 Gbit Full Duplex	100 Mbits Full Duplex
Operating System	Ubuntu 10.10 (32 bits)	Windows 7 (64 bits)	Ubuntu 10.10 (32 bits)	Red Hat

Table 3 shows the results for these tests, classified by: employed computers, test scenarios and kind of network. The kind of network is in some cases common to more than one experiment. All tests were performed using these computers and the network infrastructure of the GAPH group laboratory.

Table 3. Performance test results.

<table>
<tr><th colspan="7">Performance Tests - Average Speeds for 180 seconds of execution (Results in Mbits/s)</th></tr>
<tr><th>Computers Used</th><td>Virtual Machines (single core)</td><td>Virtual Machines (dual core)</td><td>GAPHL</td><td colspan="3">VAIOs</td></tr>
<tr><th>Network
Test Scenario</th><td colspan="2">Network Card of the Host Computer</td><td colspan="2">Wired LAN</td><td>Wireless LAN</td><td>Wireless Ad-Hoc</td></tr>
<tr><td>1</td><td>14020,00</td><td>22852,00</td><td>39555,00</td><td colspan="3">3778,00</td></tr>
<tr><td>2</td><td>2211,00</td><td>2862,00</td><td>11753,00</td><td colspan="3">719,00</td></tr>
<tr><td>3 (160-bit key)</td><td>4,18</td><td>7,58</td><td>7,87</td><td colspan="3">1,48</td></tr>
<tr><td>3 (256-bit key)</td><td>2,04</td><td>3,47</td><td>4,00</td><td colspan="3">0,72</td></tr>
<tr><td>4</td><td>1713,00</td><td>2213,00</td><td>94,30</td><td>94,40</td><td>2,87</td><td>13,50</td></tr>
<tr><td>5</td><td>1078,00</td><td>1147,00</td><td>94,30</td><td>94,70</td><td>3,35</td><td>14,10</td></tr>
<tr><td>6 (160-bit key)</td><td>6,82</td><td>6,77</td><td>7,71</td><td>3,12</td><td>1,88</td><td>3,10</td></tr>
<tr><td>6 (256-bit key)</td><td>3,43</td><td>3,40</td><td>3,92</td><td>1,62</td><td>1,62</td><td>1,64</td></tr>
</table>

After analyzing the results obtained in test scenarios 1 and 4, it is possible to notice the maximum speed that each network infrastructure supports with each one of the employed computers. Note that in test scenario 1, the maximum speed is limited by the processing power of the computer, while in test scenario 4 it is limited by the type of network. Test scenarios 2 and 5 allow calculating the overhead when using the “ecc_connection” software without cryptography. Results show that there are two limiting resources, computational power and network type, as the tests demonstrate. However, there is an exception. In the case of virtual machines in both tests, the only limitation is the computational power.

Finally, tests scenarios 3 and 6 enable to measure the maximum speed when the “ecc_connection” software with key sizes of 160 or 256 bits encrypts all data. As expected, the maximum speed is lower with bigger key size. Consider that test scenario 6 is the most similar one to a real scenario and let us compare the overall results. Then, it is possible to see that the main resource limiting the maximum speed is computational power. This is clear, since when data are encrypted, the speed is always lower than the maximum speed for all types of network.

1.3. Project Context, Objectives and Motivation

This work started in the scope of the INCT-SEC project cooperation, which was developing resources for indoor tactical robots and autonomous vehicles, where a communication system that guarantees maximum security and performance between mobile devices or between mobile devices and some base station was envisaged. During the first year of the MSc course, studies and experiments were conducted, as demonstrated in Section 1.2, about secure communication and cryptographic algorithms. As an experiment, software was developed to encrypt a communication between any two computing devices through any network. The tests scenarios demonstrated that the software performance is limited by the processing power of the device that executes it and considering that even when high processing power computers are used the performance was still very low compared to insecure communication. Thus, the main proposal of this work has been to investigate and propose dedicated hardware modules to accelerate the execution of cryptographic functions.

Therefore, this work proposes to develop a module in hardware that does the same as the previously developed software, with the advantage of being a dedicated hardware and so it was expected to achieve a much higher performance than pure software implementations. These hardware modules were developed in VHDL and prototyped on FPGAs. They work as black boxes, where there is an input Ethernet port connected to the mobile device or computer. The output will be another Ethernet port or a wireless transmitter, as illustrated in Figure 13. Thus, all data sent to a specified IP address and TCP port is encrypted creating a secure communication link. Next, applications running in both computers are unaware that the communication is encrypted and no application change is necessary.

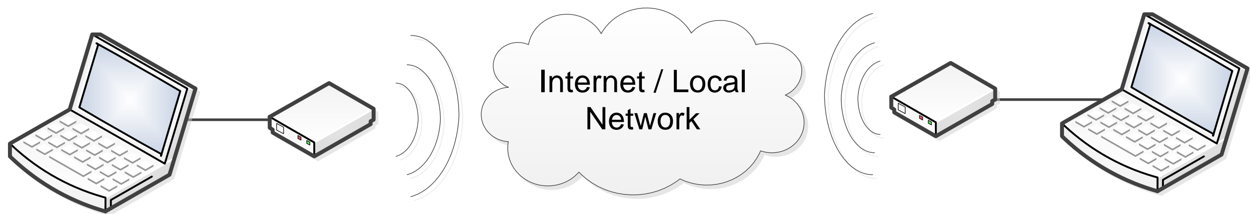


Figure 13. Overall architecture of the context for the proposed work.

1.4. Document Structure

In the next Chapter is presented the state-of-the-art, as the related work of ECC crypto processors, and a discussion relating all works. Chapter 3 presents an overview of the proposed project, showing the overview of the hardware architecture developed and the target hardware used to prototype. Following, Chapter 4 presents the development of the generic soft IP core for ECC, showing its hardware architecture, the development of its hardware modules, as well its results of simulations, validations and synthesis. Chapter 5 details the secure communication system that was developed using the soft core for ECC and in the end is shown all its results, mainly those results of prototyping in the development board. Finally, the last Chapter presents the conclusions, a discussion about applications that could use the soft core and its viability; finally, it points out some future works.

2. RELATED WORK

This Chapter presents the state-of-the-art about crypto processors or coprocessors for ECC. These related works present the main difficulties and challenges to develop an efficient architecture for ECC. Each work has some advantages and disadvantages that vary accordingly to their respective objectives. Thus, the next sections present a brief analysis of each work to show the main points that helped in the decisions and choices to the design project in this work. The last Section (2.8) presents a discussion relating all presented works.

2.1. Bednara et al. [BED02]

Bednara et al. present a coprocessor FPGA based for ECC aiming to explore the tradeoffs between area and performance using several algorithms and constraints. So, their crypto processor was implemented in a generic way to enable the analysis of different representations of elliptic curves over finite fields of characteristic two. The authors present different algorithms for some basic ECC operations, and justify which one is most indicated, according to the required constraints. Authors show some results obtained for an elliptic curve represented over $GF(2^{191})$, using different algorithms with distinct configurations and with mixed coordinates, such as affine/Jacobian and affine/López-Dahab.

Considering the date of this work, they achieved good results with their generic architecture of crypto processor, which was able to fulfill several constraints of area and performance for different elliptic curves. This work also shows that coprocessors for ECC have already been developed and studied for more than ten years.

2.2. Li et al. [LI08]

Li et al. present an FPGA implementation of a point multiplier for ECC in $GF(2^{283})$. The main objective was to demonstrate how fast their implementation in hardware could execute and how much speed-up it would represent when compared to software implementations. So, they used projective coordinates and the faster known algorithms at that time for the operations in finite field $GF(2^{283})$. Consequently, their implementation was not generic. In the end, they present results achieved by their point multiplier implemented in hardware and conclude that their implementation was 31 times faster when compared with software implementations. Figure 14, Figure 15 and Figure 16 show some results the paper obtained when their implementation was synthesized for a Virtex 4 and the comparison with related works and software implementation.

Device utilization summary of ECC

Arithmetic unit	Maximum frequency (MHz)	CLBs	FFs	LUTs
Point addition	283.728	7412	9016	13,826
Point doubling	281.861	5378	6031	10,341
Coordinate converter	183.968	15,009	16,129	27,753
Point multiplication	171.247	30,001	36,142	51,094

Figure 14. Li et al. [LI08] results achieved in device utilization.

Speedup of hardware over software

Operation	FPGA latency, µs	Software latency, µs	Speedup
Point addition	0.6	29	48
Point doubling	0.41	21	51
Coordinate converter	24	58	2.4
Point multiplication	304	9600	31.6

Figure 15. Li et al. [LI08] speedup achieved in HW vs. SW implementations.

Comparisons of latency of point multiplication

Design	Key size	Latency (ms)
Leung et al. [22]	281	14.3
Ernst et al. [9]	270	6.85
Our work	283	0.304

Figure 16. Li et al. [LI08] comparison with related works.

2.3. Keller et al. [KEL09]

Keller et al. present a coprocessor to perform ECC point multiplication in GF(2^m) and GF(*p*). Their main objective was to evaluate and explore several algorithms, design architectures and elliptic curves to determine the tradeoffs between power and performance. They used a development kit board composed by the Spartan 3E FPGA, an external processor, memories RAM and ROM and pins connected to the FPGA that enables the measurement of power consumption. Authors conducted several tests with diverse configurations, considering different coordinates and algorithms for GF(2^{163}) and GF(*p*). In the end, according to their results, they conclude that the implementations using GF(2^m) are more energy efficient than the GF(*p*), due the bit length of each representation, which impacts in the circuit sizes for the operations of addition and multiplication, and if the circuits are bigger it also consumes more energy. As another important point, usually the algorithms for GF(2^m) are easier to implement in hardware than those for GF(*p*), which means that they execute in fewer clock cycles and require less energy. Finally, this paper can be used as terminology guide to ECC definitions, such as the best representation in finite field and algorithms for the operations, according to the desired constraints of power and performance for a future application.

2.4. Järvinen [JÄR11]

Järvinen presents a high-speed coprocessor for ECC, optimized for the Koblitz curve K-163 defined in [NIST99]. The main objective was to implement and demonstrate a processor for ECC with high throughput. Its design architecture comprises a pipeline with three stages that enables the computation of up to three point multiplications at the same time. Operations are performed using projective coordinates. The three stages consist of a preprocessor, a main processor and a post processor. In a simple form, the preprocessor computes some points that are required by the main processor to calculate the point multiplication, so this design also uses some memory blocks to store the points needed in the next stage. The main processor computes the point multiplication, considering the points are represented in projective coordinates, executing a right-to-left algorithm with pre-computed

points. Finally, the last stage performs the point conversion to affine coordinates. After synthesizing the design for a Stratix II FPGA as target, the authors discuss the results. They find an interesting point: when comparing the time to compute a single point multiplication with the related literature, although their computation time is not the smallest one, it does achieve the highest throughput of point multiplications per second (see Figure 17). This is justified by the use of pipelining.

Comparison to other processors available in the literature.

Reference	Device	Curve	Area	Memory	Delay (μs)	Throughput (ops)
This work	Stratix II S180C3	K-163	14280 ALMs	25 M4Ks	11.71	235,550
Ahmadi [15][a]	Virtex-II 4000	K-233	15916 slices	Some BRAMs	7.22	138,504
Dimitrov [13]	Virtex-II 2000-6	K-163	6494 slices	6 BRAMs	39.56	25,278
Dimitrov [14]	Stratix II S180C3	K-163	28328 ALMs	52 M512s, 66 M4Ks	17.15	58,309
Järvinen [19]	Stratix II S180C3	K-163	16930 ALMs	21 M4Ks	16.36	161,290
Järvinen [16][b]	Stratix II S180C3	K-163	26148 ALMs	–	4.91	203,665
Järvinen [17]	Stratix II S180C3	K-163	13472 ALMs	Several M512s and M4Ks	25.81	49,318
Lutz [12][b]	Virtex-E 2000	K-163	10017 LUTs	–	75	13,333
Okada [11] [b,c]	Flex 10K	K-163	n/a	n/a	45,600	22

[a] The values do not include area or delay of the τ-adic conversion, precomputations, or coordinate conversion.
[b] The values do not include area or delay of the τ-adic conversion.
[c] Supports also general curves.

Figure 17. Results obtained by Järvinen [JÄR11].

2.5. Masoumi et al. [MAS12]

Masoumi et al. present another crypto processor optimized for ECC over $GF(2^{163})$, which is very similar to that of Järvinen [JÄR11]. However, in the former authors focus on reducing the critical path. They also to reorganize the basic operations of point multiplication algorithm to increase their parallelism. Furthermore, they use Lopez-Dahab projective coordinates for the scalar multiplication and a specialized algorithm optimized for elliptic curves over $GF(2^{163})$. They synthesized their implementation for Virtex 4 FPGAs and compared the results obtained with other works. Their implementation achieved a good efficiency, which they defined according to the throughput per slice occupied in the FPGA, but their implementation achieved lower throughput if compared with the previous paper [JÄR11]. Thus, it is possible to conclude that it would be nice to combine both techniques presented in these two works, trying to obtain all the optimizations and maybe the design would achieve at the same time higher performance and throughput.

2.6. Dias et al. [DIA13]

Dias et al. present the implementation of a crypto processor for ECC over $GF(2^m)$ using only affine coordinates in polynomial basis. As the authors explain in the paper, they chose to implement this crypto processor using affine coordinates because recent papers demonstrated that affine coordinates provide more security against side channel attacks and simple power attacks, as explained by Fournaris et al. [FOU09]. Also, they used an efficient algorithm to calculate the modular inversion that makes its design to be comparable to others that employ projective coordinates. They synthesized and prototyped their design in a platform composed by two Altera FPGAs running at 250 MHz, where their design performed in average the point multiplication in 0.10 ms, which indicates a good improvement when compared to related work that used projective coordinates. However, their design has a disadvantage on the occupied area (it resulted in a very large circuit, with more than 200K slices of their FPGAs), making unviable its use for mobile applications.

2.7. Loi et al. [LOI13]

Loi et al. present a new design for a crypto processor that supports all the five Koblitz curves defined by the NIST [NIST99] but without the need to reconfiguring the FPGA. This means that the same hardware can be used for achieving different security levels according to the selected elliptic curve. Their design uses projective coordinates and comprises block RAMs that store temporary values, and generic modules that are able to calculate the elliptic curve operations for all the defined Koblitz curves. The design was synthesized for Xilinx Spartan 3 Virtex 4 and Virtex E FPGAs. According to their results, the design synthesized for Virtex 4 presented the best results, performing a single point multiplication in a range of 0.273 ms to 4.335 ms, depending of the selected curve. Another interesting point is the occupied area, about 2431 slices in Virtex 4. So, their design seems to be suitable for mobile applications due to its small size and, at same time, it supports all five NIST Koblitz curves.

2.8. Discussion of Related Work

The comparison of all related works reveals that there are many ways to develop a crypto processor or a coprocessor for ECC, depending on the requirements and constraints to fulfill. Examples of addressed metrics are performance, area and power, which lead to specific hardware. Bednara et al. [BED02] and Keller et al. [KEL09] implemented several different algorithms and hardware architectures to find out the advantages of each one. They showed that for hardware implementations of ECC, it is better to use finite fields of characteristic two, such as GF(2^m), represented in polynomial basis due to the better implementations enabled by their algorithms in hardware. These authors obtained smaller circuits, consuming less energy and achieving higher performance. However, there is a contrasting point between other works, related to the representations of point coordinates. Some papers propose the use of projective coordinates as the best way to achieve higher performance. However, other papers showed that the results achieved when using affine coordinates are currently comparable to the hardware architectures that employ projective coordinates, as presented by Dias et al. [DIA13]. In the end, it seems that some algorithms are optimized for certain representations, thus resulting in better performance. Another important point is that almost all revised hardware designs are implementations for a specific elliptic curve, which make them unviable or very hard to port their designs to other curves, except in the work of Loi et al. [LOI13] that supports the five Koblitz curves defined by NIST.

3. PROPOSED DESIGN

Considering the main objective of this work and analyzing related works, a communication system and a hardware architecture were defined, the expected prototype to develop. As studied and demonstrated in previous Chapters, the main requirements for this communication system are to support elliptic curves cryptography in hardware, and be able to create a secure communication link through any TCP network, such as a local network or the Internet through a wireless or wired connection. The next sections explore and illustrate the expected operation of the system and its main hardware components.

3.1. Hardware Architecture

The hardware architecture comprises two network interfaces and a module responsible for performing elliptic curve operations such as key generation, data encryption and decryption. Figure 18 illustrates this architecture. The main controller creates the secure communication link defined by a configuration that specifies the source and destination address, such as a MAC or IP address, and the TCP port, as implemented in the software experiment detailed in Section 1.2. Only specific data packets must be encrypted and any other packet will be forwarded without any modification. This was the main idea for the proposed hardware architecture that is more detailed in Chapters 4 and 5 that explain how this system was developed.

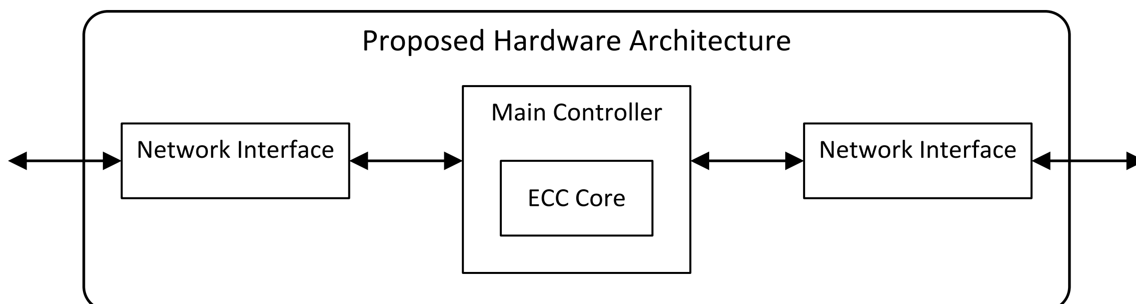


Figure 18. Overall architecture of the proposed communication system.

3.2. Target Hardware

This work aims to demonstrate the viability of creating a secure communication link using ECC implemented in hardware. The hardware chosen as target for prototyping was the HiTech Global development platform HTG-V5-PCIE-330. Figure 19 shows a picture of this platform, which fulfills the main expected hardware requirements: this board has two Ethernet interfaces and the FPGA Virtex 5 LX330T, which is an FPGA capable to prototype large circuits, if necessary.

The development of the project was divided in two parts. In the first one, a soft IP core that implements the ECC operations was developed, as explained in Chapter 4. In the second part, the communication system that uses the previously developed ECC core was built, as detailed in Chapter 5. The expectation is to use two identical development boards to prototype and validate the proposed communication system, providing a secure communication link between two computers.

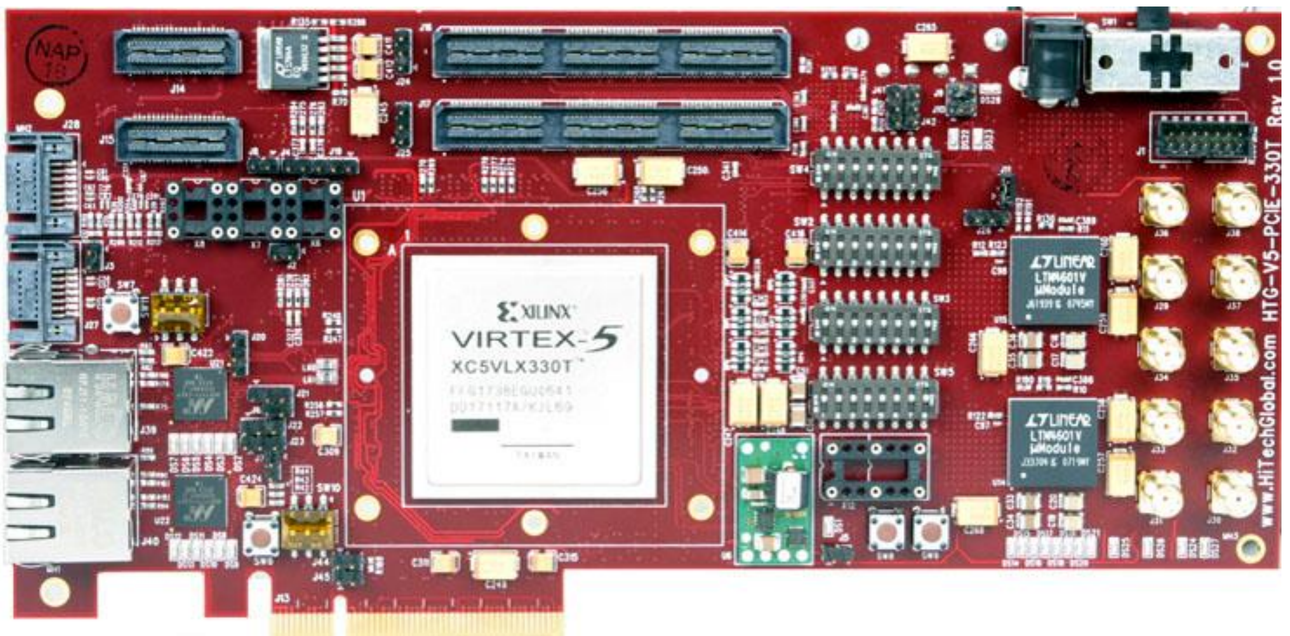


Figure 19. Development platform HTG-V5-PCIE-330 of HiTech Global.

4. SOFT IP CORE FOR ECC

This Chapter discusses the implementation of ECC in hardware, which involved solving several hard problems, due to the complexity of operations over finite fields, and the lack of ready to use solutions to this. The Author used the work of Hankerson et al. [HAN04] as a reference to define the basic hardware architecture, as Section 4.1 describes. The book of Deschamps et al. [DES09] served as a reference to implement the main hardware modules, all of which are explained in Sections 4.2 and 4.3. After this, Sections 4.4 to 4.6 show simulations results, discuss the hardware validation for the basic operations, and some synthesis results considering the target hardware, which will be used to prototype the ECC IP Core from VHDL descriptions.

4.1. Hardware Architecture

The goal of this works is to accelerate the main ECC operations, by developing a generic ECC co-processor. This was developed as a soft IP core to be very flexible, by supporting many different parameter settings and curve choices. The result is a hardware description that can be used in several applications with widely different constraints of area, timing and power. Each finite field operation was developed as a hardware module, including: a finite field squarer, a divider, a multiplier, a point adder and a point multiplier.

ECC can be implemented over different representations of finite fields, such as GF(p) or GF(2^m), which are the most used according to the literature, where each one has its own advantages. However, ECC over GF(2^m) is the most indicated to be implemented in hardware, based in studies presented in the state of the art and in book [HAN04], because these operations are naturally adapted to hardware structures. In this way, the choice was to implement ECC over GF(2^m), using elliptic curves and parameters recommended in the standard SEC2 [BRO09]. This standard contains several different curves but this work uses only curves sect163k1, sect233k1, sect283k1, sect409k1 and sect571k1, which are Koblitz curves, with the respective key sizes of 163, 233, 283, 409 and 571 bits, and these curves are the same used in the software implementation.

The finite field and elliptic curve operators rely on the algorithms presented by Deschamps et al. [DES09], which discuss each operation in detail, providing alternative implementations for every operator and assessing the implementations relative efficiencies. The design proposed here is a generic description able to be used with any of the 5 mentioned Koblitz curves. The basic operations all come from finite field arithmetic, which defines operators for addition, subtraction, squaring, multiplication and division. All these are modular operations, and are in accordance to the specific elliptic curve in use. Also, all operands are numbers with size defined by the curve choice. Point addition is defined as the operation to add two points of an elliptic curve using finite field operations. Point multiplication is the main operation used in ECC and its algorithm depends on the point addition because in its simplest form this is computed by successive point additions.

In arithmetic operations over finite fields, the simplest operation is the addition (which is identical to subtraction) because it is just built using an exclusive-or gate for each bit and it is thus trivially implemented in hardware. The squaring operation is also easy to implement

because its computation can be carried with combinational hardware with minimal cost, and can be easily be computed in a single clock cycle.

Multiplication and inversion are the most complicated operations, because their implementation in hardware is computed by many squaring and multiplications, which can take a large number of clock cycles if little hardware is used, or huge amounts of hardware if they are to take just a few or even a single clock cycle to execute. Much effort is reported on the development of fast multiplication/inversion operators, and this often includes a tradeoff between area/power taken by the hardware and speed (in clock cycles). These operations are the major responsible for defining the performance of ECC, since all other operations are much simpler.

The hardware architecture comprises mainly modules which implement all operations from finite field arithmetic up to point multiplication and at the top level these modules are instantiated to implement the elliptic curve operations, encrypting, decrypting and key generation, which are as explained in Section 1.1.2. Figure 20 illustrates the module datapath hierarchy for the proposed architecture.

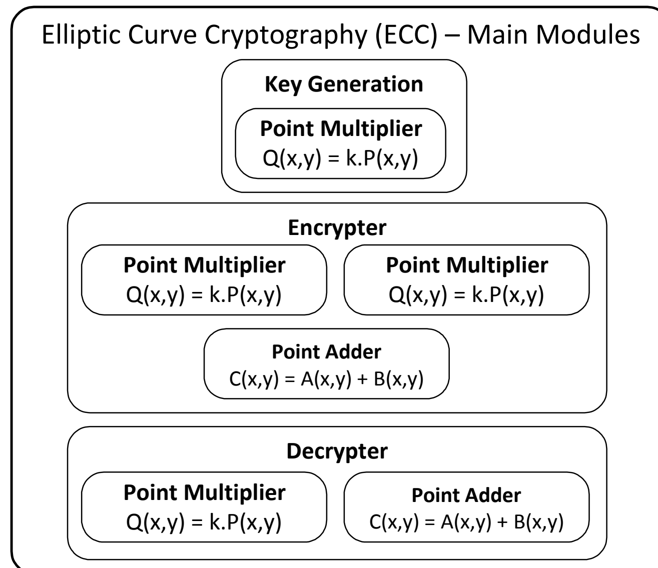


Figure 20. Overall component hierarchy in the hardware datapath of the proposed ECC soft IP.

4.2. Hardware Modules for Operations over Finite Fields

The implementation of each hardware module capitalized on the book of Deschamps et al. [DES09], because it provides VHDL implementations for several operations in finite fields, including GF(2^m), and also compares and explains each one of these operations, indicating the most efficient in each case. Their original VHDL descriptions were adapted to compose the required modules by the defined hardware architecture.

The finite field arithmetic layer comprises three modules: the squarer, the multiplier and the divider. The operations of addition and/or subtraction were not defined as a module, because they are just an exclusive-or operation that is very simply described in VHDL, typically a single line of code. Instead of creating a module for the inversion operation, only the divider was created, because the inversion of a number is only required when it will be multiplied by another, which corresponds to a division.

4.2.1. The Squarer

The squarer module is very simple and its algorithm allows a purely combinational implementation. Figure 21 shows the squarer interface, which has one input and one output. The input “a” is a number of M bits, where M can be 163, 233, 283, 409 or 571 bits depending on which curve will be used. The output “c” is also a number of M bits, because squaring over finite fields is a modular operation. Squaring a number of M bits in polynomial bases consists in inserting ‘0s’ before every bit of input “a” that will result in a 2M-bit number, as illustrated in Figure 22, and then reduce it using an algorithm with the irreducible polynomial of the elliptic curve in use, which is detailed in [DES09]. Squaring is efficiently computable in one clock cycle. Nonetheless, this operation usually takes 2 cycles to compute: one cycle to load the input and one cycle to compute the square and its modular reduction to produce the valid output.

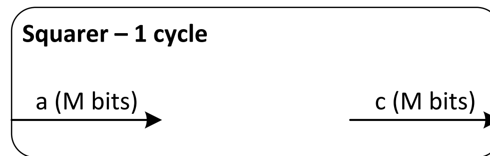


Figure 21. Squarer module interface.

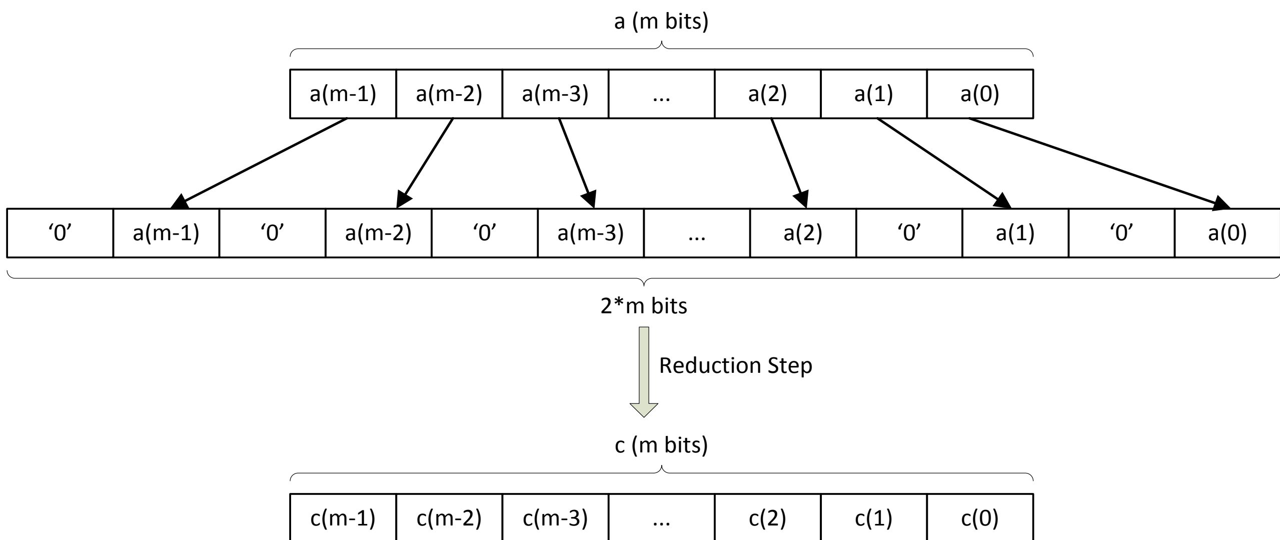


Figure 22. Steps to square a number over binary finite fields.

4.2.2. The Multiplier

Figure 23 shows the external interface of the proposed multiplier module. The module implements an interleaved multiplication algorithm that is a shift-and-add method with an interleaved reduction step, to save area and yet achieve the same efficiency of other algorithms, as presented in [DES09]. This algorithm was improved to perform a multiplication in fewer clock cycles if more area is available, as detailed in [GUA06]. This is one of the original contributions of this design compared to the literature: a configurable multiplier. When choosing the elliptic curve, it is also possible to choose a relative performance for the multiplier. Then, the hardware can be set to compute from one bit per clock cycle to up to ‘d’ bits per clock cycle, where ‘d’ is the result of rounding half the key size. Considering a binary field GF(2^m), the multiplier can perform one operation in ‘m’ clock cycles down to ‘2’ clock

cycles. As an example, using the elliptic curve K-163, which uses GF(2^{163}), if the multiplier is configured to calculate one bit per clock cycle, then one finite field multiplication will takes 163 clock cycles. However, if it is configured to compute 82 bits per clock cycle, then one finite field multiplication will take only 2 clock cycles. Meanwhile, area overhead increases only 4 times, as will later be demonstrated. Thus, this implementation enables to select adequate timing-area tradeoffs for each application.

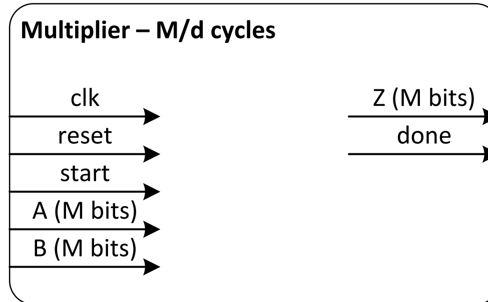


Figure 23. Multiplier module interface.

Figure 24 shows the datapath of the multiplier. It comprises three main blocks that implement the algorithm to multiply two numbers over finite fields.

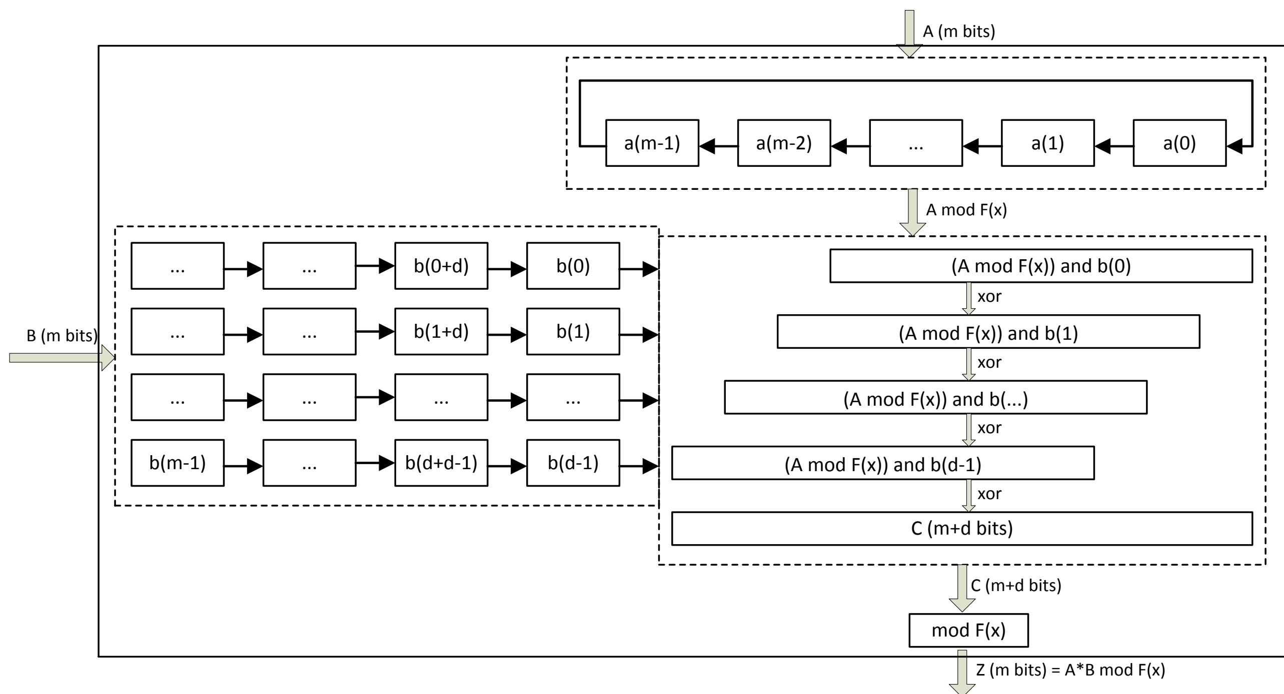


Figure 24.Datapath of the generic binary finite field multiplier.

Numbers A and B are the inputs that will be multiplied and number Z is the result of this modular multiplication, with "Z = A*B mod F(x)". F(x) is the irreducible polynomial of the elliptic curve. In Figure 24, 'd' is the number of bits that will be calculated at each clock cycle. The first block receives number A of length 'm' bits and, at each clock cycle, it rotates A by 'd' bits to the left and performs a modulus with F(x), outputting "A mod F(x)". The second block receives number B and, at each clock cycle, outputs the first 'd' bits of B and shift its 'd' bits to the right. The third block is the multiplier-accumulator. Firstly, it receives the "A mod F(x)" and the first 'd' bits of B. Then, it performs an "and" operation between all bits of

"A mod F(x)" with each 'd' bits of B. Next , it performs an "xor" operation between the previous results of the "and" operation that are also shifted from 0 to 'd-1' bits to the left, resulting in a number with "m+d-1" bits that is stored in accumulator C. Finally, after all bits from A and B were processed, a reduction step is performed. This brings the value stored in C with "m+d-1" bits to a value with "m" bits by calculating the modulus "C mod F(x)" that is the result of "A*B mod F(x)", which represents result Z. The three main blocks operate in parallel and all the operations take "m/d" clock cycles to execute.

The main novelty in this module was the development of the generic description in VHDL that can be configured at design time and used for any elliptic curve; it is easily parameterized to fulfill requirements of performance or area for each application.

4.2.3. The Divider

Figure 25 shows the interface of the divider module or the inversion module. This module has the worst performance of all modules due to the lack of algorithms to perform it efficiently, as explained in [DES09]. One of the best implementations is the binary algorithm, which is implemented through bit vector shifts and exclusive-or steps. This is a sequential algorithm that can take up to 2*m clock cycles, which in this work means 326, or 466, or 566, or 818 or 1142 cycles to compute only one division or inversion.

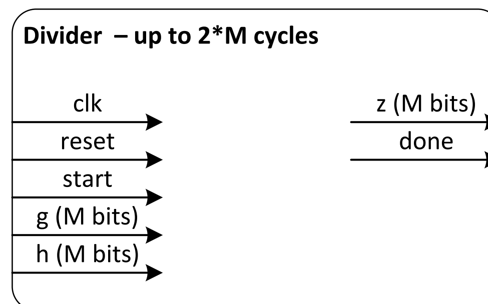


Figure 25. The finite field divider module interface.

Therefore, aiming to improve this operation, some works claim the quality of the alternative method of Itoh-Tsujii to compute the inverse of a number. Itoh and Tsujii [ITO88] proposed a method to calculate the inverse of a number in normal bases that can be performed with (m-1) squaring operations and (m-2) multiplications. This takes much more time than the binary algorithm, due to the required multiplications, which take at least 2 clock cycles each. However, authors such as Guajardo et al. [GUA02] and Bednara et al. [BED02], present the Itoh-Tsujii method adapted to standard bases or polynomial bases. By adapting their methods, it is possible to calculate the inverse of a number in much fewer clock cycles, based on a theorem that the inverse of a number A in a finite field is $A^{-1} = A^{2^{m-1}-1}$ according to their demonstration and experiments that attest its validity. The Author suggests to decompose the exponent for each elliptic curve used here as detailed in the next set of tables. Table 4 shows the decomposition of exponent 162 for the elliptic curve performed over GF(2^{163}), because according to the previously mentioned theorem the inversion of a number in this case is $A^{-1} = A^{2^{163-1}-1} = A^{2^{162}-1}$. The first column indicates the amount of steps needed to perform all the operations to find the inverse of a number. The second column shows the number of squaring operations performed in each step and this is the decomposition of the exponent. The third column details the operations of squaring and multiplications

performed to find the inverse of a number "a". Each step performs only one finite field multiplication and some squaring operations, resulting in this case in 162 squaring and 9 multiplications. Table 5 to Table 8 detail the decomposition of exponents for the other finite fields used by the other elliptic curves considered in this work.

Table 4. Decomposition of exponents for GF(2^{163}).

Step	Exponent Decomposition	Detailed Decomposition
0	1	$B_0 = a^{(2^1-1)} = a^1 = a$
1	1	$B_1 = a^{(2^2-1)} = a^{(2^1-1)} * (a^{(2^1-1)})^{2^1} = B_0 * {B_0}^{2^1} = a^3$
2	2	$B_2 = a^{(2^4-1)} = a^{(2^2-1)} * (a^{(2^2-1)})^{2^2} = B_1 * {B_1}^{2^2} = a^{15}$
3	4	$B_3 = a^{(2^8-1)} = a^{(2^4-1)} * (a^{(2^4-1)})^{2^4} = B_2 * {B_2}^{2^4} = a^{255}$
4	8	$B_4 = a^{(2^{16}-1)} = a^{(2^8-1)} * (a^{(2^8-1)})^{2^8} = B_3 * {B_3}^{2^8} = a^{65.535}$
5	16	$B_5 = a^{(2^{32}-1)} = a^{(2^{16}-1)} * (a^{(2^{16}-1)})^{2^{16}} = B_4 * {B_4}^{2^{16}} = a^{(2^{32}-1)}$
6	32	$B_6 = a^{(2^{64}-1)} = a^{(2^{32}-1)} * (a^{(2^{32}-1)})^{2^{32}} = B_5 * {B_5}^{2^{32}} = a^{(2^{64}-1)}$
7	64	$B_7 = a^{(2^{128}-1)} = a^{(2^{64}-1)} * (a^{(2^{64}-1)})^{2^{64}} = B_6 * {B_6}^{2^{64}} = a^{(2^{128}-1)}$
8	32	$B_8 = a^{(2^{160}-1)} = a^{(2^{32}-1)} * (a^{(2^{128}-1)})^{2^{32}} = B_5 * {B_7}^{2^{32}} = a^{(2^{160}-1)}$
9	2	$B_9 = a^{(2^{162}-1)} = a^{(2^2-1)} * (a^{(2^{160}-1)})^{2^2} = B_1 * {B_8}^{2^2} = a^{(2^{162}-1)}$

Table 5. Decomposition of exponents for GF(2^{233}).

Step	Exponent Decomposition	Detailed Decomposition
0	1	$B_0 = a^{(2^1-1)} = a^1 = a$
1	1	$B_1 = a^{(2^2-1)} = a^{(2^1-1)} * (a^{(2^1-1)})^{2^1} = B_0 * {B_0}^{2^1} = a^3$
2	2	$B_2 = a^{(2^4-1)} = a^{(2^2-1)} * (a^{(2^2-1)})^{2^2} = B_1 * {B_1}^{2^2} = a^{15}$
3	4	$B_3 = a^{(2^8-1)} = a^{(2^4-1)} * (a^{(2^4-1)})^{2^4} = B_2 * {B_2}^{2^4} = a^{255}$
4	8	$B_4 = a^{(2^{16}-1)} = a^{(2^8-1)} * (a^{(2^8-1)})^{2^8} = B_3 * {B_3}^{2^8} = a^{65.535}$
5	16	$B_5 = a^{(2^{32}-1)} = a^{(2^{16}-1)} * (a^{(2^{16}-1)})^{2^{16}} = B_4 * {B_4}^{2^{16}} = a^{(2^{32}-1)}$
6	32	$B_6 = a^{(2^{64}-1)} = a^{(2^{32}-1)} * (a^{(2^{32}-1)})^{2^{32}} = B_5 * {B_5}^{2^{32}} = a^{(2^{64}-1)}$
7	64	$B_7 = a^{(2^{128}-1)} = a^{(2^{64}-1)} * (a^{(2^{64}-1)})^{2^{64}} = B_6 * {B_6}^{2^{64}} = a^{(2^{128}-1)}$
8	64	$B_8 = a^{(2^{192}-1)} = a^{(2^{64}-1)} * (a^{(2^{128}-1)})^{2^{64}} = B_6 * {B_7}^{2^{64}} = a^{(2^{192}-1)}$
9	32	$B_9 = a^{(2^{224}-1)} = a^{(2^{32}-1)} * (a^{(2^{192}-1)})^{2^{32}} = B_5 * {B_8}^{2^{32}} = a^{(2^{224}-1)}$
10	8	$B_{10} = a^{(2^{232}-1)} = a^{(2^8-1)} * (a^{(2^{224}-1)})^{2^8} = B_3 * {B_9}^{2^8} = a^{(2^{232}-1)}$

Table 6. Decomposition of exponents for GF(2^{283}).

Step	Exponent Decom-position	Detailed Decomposition
0	1	$B_0 = a^{(2^1-1)} = a^1 = a$
1	1	$B_1 = a^{(2^2-1)} = a^{(2^1-1)} * (a^{(2^1-1)})^{2^1} = B_0 * {B_0}^{2^1} = a^3$
2	2	$B_2 = a^{(2^4-1)} = a^{(2^2-1)} * (a^{(2^2-1)})^{2^2} = B_1 * {B_1}^{2^2} = a^{15}$
3	4	$B_3 = a^{(2^8-1)} = a^{(2^4-1)} * (a^{(2^4-1)})^{2^4} = B_2 * {B_2}^{2^4} = a^{255}$
4	8	$B_4 = a^{(2^{16}-1)} = a^{(2^8-1)} * (a^{(2^8-1)})^{2^8} = B_3 * {B_3}^{2^8} = a^{65.535}$
5	16	$B_5 = a^{(2^{32}-1)} = a^{(2^{16}-1)} * (a^{(2^{16}-1)})^{2^{16}} = B_4 * {B_4}^{2^{16}} = a^{(2^{32}-1)}$
6	32	$B_6 = a^{(2^{64}-1)} = a^{(2^{32}-1)} * (a^{(2^{32}-1)})^{2^{32}} = B_5 * {B_5}^{2^{32}} = a^{(2^{64}-1)}$
7	64	$B_7 = a^{(2^{128}-1)} = a^{(2^{64}-1)} * (a^{(2^{64}-1)})^{2^{64}} = B_6 * {B_6}^{2^{64}} = a^{(2^{128}-1)}$
8	128	$B_8 = a^{(2^{256}-1)} = a^{(2^{128}-1)} * (a^{(2^{128}-1)})^{2^{128}} = B_7 * {B_7}^{2^{128}} = a^{(2^{256}-1)}$
9	16	$B_9 = a^{(2^{272}-1)} = a^{(2^{16}-1)} * (a^{(2^{256}-1)})^{2^{16}} = B_4 * {B_8}^{2^{16}} = a^{(2^{272}-1)}$
10	8	$B_{10} = a^{(2^{280}-1)} = a^{(2^8-1)} * (a^{(2^{272}-1)})^{2^8} = B_3 * {B_9}^{2^8} = a^{(2^{280}-1)}$
11	2	$B_{11} = a^{(2^{282}-1)} = a^{(2^2-1)} * (a^{(2^{280}-1)})^{2^2} = B_1 * {B_{10}}^{2^2} = a^{(2^{282}-1)}$

Table 7. Decomposition of exponents for GF(2^{409}).

Step	Exponent Decom-position	Detailed Decomposition
0	1	$B_0 = a^{(2^1-1)} = a^1 = a$
1	1	$B_1 = a^{(2^2-1)} = a^{(2^1-1)} * (a^{(2^1-1)})^{2^1} = B_0 * {B_0}^{2^1} = a^3$
2	1	$B_2 = a^{(2^3-1)} = a^{(2^1-1)} * (a^{(2^2-1)})^{2^1} = B_0 * {B_1}^{2^1} = a^7$
3	3	$B_3 = a^{(2^6-1)} = a^{(2^3-1)} * (a^{(2^3-1)})^{2^3} = B_2 * {B_2}^{2^3} = a^{63}$
4	6	$B_4 = a^{(2^{12}-1)} = a^{(2^6-1)} * (a^{(2^6-1)})^{2^6} = B_3 * {B_3}^{2^6} = a^{4095}$
5	12	$B_5 = a^{(2^{24}-1)} = a^{(2^{12}-1)} * (a^{(2^{12}-1)})^{2^{12}} = B_4 * {B_4}^{2^{12}} = a^{(2^{24}-1)}$
6	24	$B_6 = a^{(2^{48}-1)} = a^{(2^{24}-1)} * (a^{(2^{24}-1)})^{2^{24}} = B_5 * {B_5}^{2^{24}} = a^{(2^{48}-1)}$
7	3	$B_7 = a^{(2^{51}-1)} = a^{(2^3-1)} * (a^{(2^{48}-1)})^{2^3} = B_2 * {B_6}^{2^3} = a^{(2^{51}-1)}$
8	51	$B_8 = a^{(2^{102}-1)} = a^{(2^{51}-1)} * (a^{(2^{51}-1)})^{2^{51}} = B_7 * {B_7}^{2^{51}} = a^{(2^{102}-1)}$
9	102	$B_9 = a^{(2^{204}-1)} = a^{(2^{102}-1)} * (a^{(2^{102}-1)})^{2^{102}} = B_8 * {B_8}^{2^{102}} = a^{(2^{204}-1)}$
10	204	$B_{10} = a^{(2^{408}-1)} = a^{(2^{204}-1)} * (a^{(2^{204}-1)})^{2^{204}} = B_9 * {B_9}^{2^{204}} = a^{(2^{408}-1)}$

Table 8. Decomposition of exponents for GF(2^{571}).

Step	Exponent Decom-position	Detailed Decomposition
0	1	$B_0 = a^{(2^1-1)} = a^1 = a$
1	1	$B_1 = a^{(2^2-1)} = a^{(2^1-1)} * (a^{(2^1-1)})^{2^1} = B_0 * {B_0}^{2^1} = a^3$
2	1	$B_2 = a^{(2^3-1)} = a^{(2^1-1)} * (a^{(2^2-1)})^{2^1} = B_0 * {B_1}^{2^1} = a^7$
3	2	$B_3 = a^{(2^5-1)} = a^{(2^2-1)} * (a^{(2^3-1)})^{2^2} = B_1 * {B_2}^{2^2} = a^{31}$
4	5	$B_4 = a^{(2^{10}-1)} = a^{(2^5-1)} * (a^{(2^5-1)})^{2^5} = B_3 * {B_3}^{2^5} = a^{1023}$
5	10	$B_5 = a^{(2^{20}-1)} = a^{(2^{10}-1)} * (a^{(2^{10}-1)})^{2^{10}} = B_4 * {B_4}^{2^{10}} = a^{(2^{20}-1)}$
6	20	$B_6 = a^{(2^{40}-1)} = a^{(2^{20}-1)} * (a^{(2^{20}-1)})^{2^{20}} = B_5 * {B_5}^{2^{20}} = a^{(2^{40}-1)}$
7	40	$B_7 = a^{(2^{80}-1)} = a^{(2^{40}-1)} * (a^{(2^{40}-1)})^{2^{40}} = B_6 * {B_6}^{2^{40}} = a^{(2^{80}-1)}$
8	80	$B_8 = a^{(2^{160}-1)} = a^{(2^{80}-1)} * (a^{(2^{80}-1)})^{2^{80}} = B_7 * {B_7}^{2^{80}} = a^{(2^{160}-1)}$
9	160	$B_9 = a^{(2^{320}-1)} = a^{(2^{160}-1)} * (a^{(2^{160}-1)})^{2^{160}} = B_8 * {B_8}^{2^{160}} = a^{(2^{320}-1)}$
10	160	$B_{10} = a^{(2^{480}-1)} = a^{(2^{160}-1)} * (a^{(2^{320}-1)})^{2^{160}} = B_8 * {B_9}^{2^{160}} = a^{(2^{480}-1)}$
11	80	$B_{11} = a^{(2^{560}-1)} = a^{(2^{80}-1)} * (a^{(2^{480}-1)})^{2^{80}} = B_7 * {B_{10}}^{2^{80}} = a^{(2^{560}-1)}$
12	10	$B_{12} = a^{(2^{570}-1)} = a^{(2^{10}-1)} * (a^{(2^{560}-1)})^{2^{10}} = B_4 * {B_{11}}^{2^{10}} = a^{(2^{570}-1)}$

By decomposing the exponents, it is possible to compute the inverse of a number using only 'm-1' squaring operations and 9 to 12 multiplications when using the Koblitz curves K-163 to K-571, respectively. Therefore, if a finite field multiplication can be performed efficiently and considering that a squaring operation takes just one clock cycle, it is possible to computer the inverse in fewer clock cycles than with the binary algorithm, which takes up to 2*m clock cycles. For example, using again K-163 and a finite field multiplier configured to multiply in 2 clock cycles, computing the inverse would take 162 squaring operations and 9 multiplications, which amounts to 162*1+9*2 clock cycles or 180 clock cycles, much less than the 326 clock cycles required by the binary algorithm.

Thus, this work suggests two alternative architectures to perform division. One implements the binary algorithm detailed in [DES09] that performs a division in '2*m' clock cycles, which is good when area constraints are strict, and the other implements the adapted Itoh-Tsujii method. Considering that the finite field multiplier performance affects the finite field divider module, the latter uses the architecture that is more efficient, depending on the multiplier configuration. If the number of cycles to perform m-1 squaring operations and 9 to 12 multiplications is more than 2*m cycles, the divider uses the first architecture (the binary algorithm), otherwise it uses the second architecture (the adapted Itoh-Tsujii algorithm). Because of this, there are signals "start" and "done" to indicate when the module can start its computation (because inputs are ready) and when it has finished (because the output is valid).

4.3. Hardware Modules for ECC Operations

After designing the three basic modules (squarer, multiplier, and divider), the point adder and point multiplication modules detailed in [DES09] were adapted to perform opera-

tions over any of the Koblitz curves. Next, the basic architecture to implement ECC in hardware was developed including: key generation, encryption and decryption, as Figure 20 shows. To enable parallelizing operations, the key generation comprises a point multiplier; the encrypter contains two point multipliers and one point adder; and the decrypter comprises one point multiplier and one point adder.

4.3.1. Point Adder

Figure 26 shows the module interface of the point adder, which receives “x” and “y” coordinates of two points and outputs “x” and “y” of the resulting point. Point addition in elliptic curves over GF(2^m) follows 5 rules, which in simplified form are stated as:

1. Adding two equal points with (x, y) equal to (0, 0), results in the point (0, 0);

2. Adding any point different of (0, 0) to another point equal to (0, 0), results in a point equal to the first one.

3. Adding two points, with the same x-coordinates and different y-coordinates, results in the point (0, 0).

4. Adding any two generic points results in a new point computed by the equations below, considering $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

5. Adding a point to itself, when its coordinates are not (0, 0), produces the point given by the equations below, considering $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$:

$$\lambda = x_1 + \frac{y_1}{x_1}$$

$$x_3 = \lambda^2 + \lambda + a$$

$$y_3 = x_1^2 + \lambda . x_3 + x_3$$

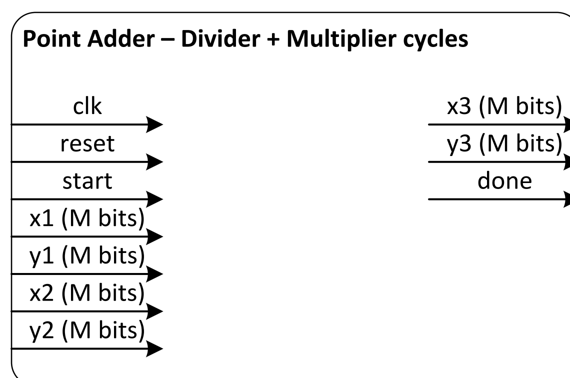


Figure 26. Point adder module interface.

This module follows the template suggested by Deschamps et al. [DES09]. However, in the reference the point addition implements only the fourth and fifth rules. To increase performance, the structure of the module was modified to support the full definition of point

addition, expressed by the 5 rules above, guaranteeing correct results for all possible cases. Considering the most common cases, which are covered by rules 4 and 5, this module can take up to 2*m+m clock cycles, because it uses one division and one multiplication. So, the module performance depends mainly on that of the divider and multiplier modules. Figure 27 illustrates the modules that compose the point adder.

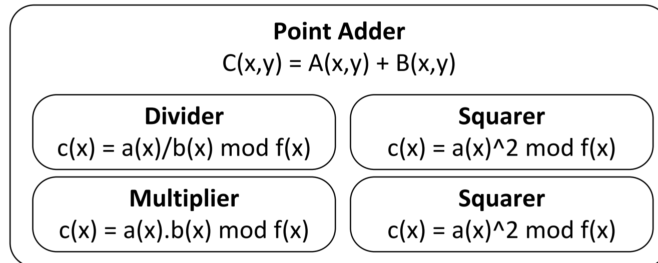


Figure 27. Modules that compose the point adder.

4.3.2. Negating a Point

The negate module, showed in the Figure 28, calculates the negative of a given point and is necessary only when a point subtraction is needed. Point negation is very simple to compute, because given a point (x, y) then –(x, y) = (x, x+y). Thus, this point is used with the point adder module, which will result in the subtraction of two points. In the case of ECC, point subtraction is used only in the decryption process.

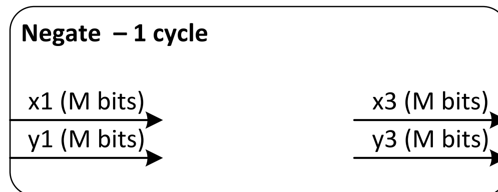


Figure 28. Module interface of negate point.

4.3.3. Point Multiplier

Point multiplication is considered the main operation of ECC because it takes more time to compute. Figure 29 shows the interface of the point multiplier module, which receives the point coordinates and the number to be multiplied; the start and done signals indicate when the module can start to compute and when the operation is done.

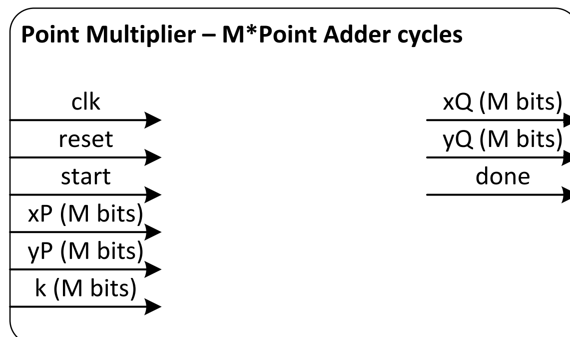


Figure 29. Module interface of the point multiplier.

This occurs because a point multiplication over finite fields is a scalar multiplication, which is basically computed through successive additions. However, using the point multiplication algorithm based on τ-ary representation of k, presented in [DES09], that is an optimized algorithm and it can require up to m point additions and each point addition can takes up to $3*m^2$ clock cycles. Thus, it is extremely important to improve this module by using techniques like pipeline or other parallelization techniques, when the main target of the design is to achieve high throughput. Figure 30 shows an example of a module composition for the point multiplier module. As already mentioned, all modules are configurable to support different sizes, according to the elliptic curve in use. Depending on the size m in bits, the best architecture can be chosen for some modules. For example, the finite field inversion operator offers two different architectures, each based on an algorithm that gives better values depending on the configuration defined for the finite field multiplier.

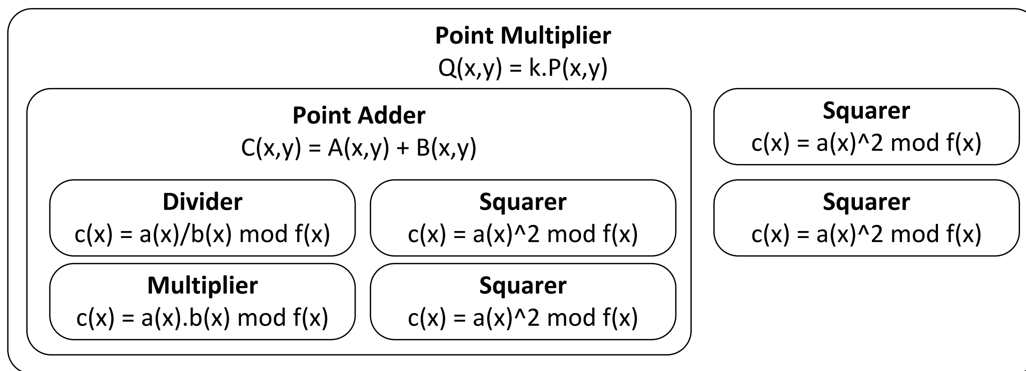


Figure 30. Example of module composition to create the point multiplier.

4.3.4. Key Generator

The key generator module comprises a point multiplier that, as Figure 32 shows, implements public key generation, based on the private key and point generator of the elliptic curve in use. The module interface receives the user-chosen private key and outputs the public key represented by point Q of the elliptic curve. This module requires up to $3*m^2$ clock cycles because it performs only one point multiplication.

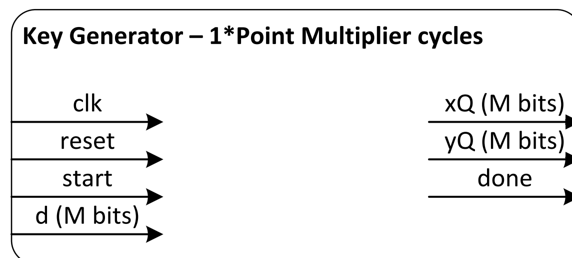


Figure 31. Module interface of the key generator.

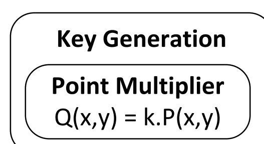


Figure 32. Composition of modules to implement the (public) key generation.

4.3.5. Data Encrypter

The encrypter module consists in two point multipliers and one point adder. Figure 34 illustrates its block diagram. The encryption is performed in up to $3*m^2+3*m$ clock cycles, because it depends of one point addition and two point multiplications but the point multiplications can execute in parallel. Figure 33 shows the module interface, which receives the xQ and yQ (the public key) and "data in”, the data to encrypt. Points C1 and C2 compose the encrypted data, C1 is the point generator of the elliptic curve multiplied by a generated random number and C2 is "data in" added to the public key and multiplied by the same random number, as explained the encryption process (see Figure 2). The random number used to encrypt each datum is extremely important to guarantee the security level of ECC, but the implementation of a real random number generator in hardware is also very difficult. Here an alternative simple method was used to generate this pseudo random number. The first random number is pre-defined, randomly chosen during the hardware implementation, and the next numbers used are the “x” coordinates of the previous point multiplication between the public key with the previous random number. Then, at each encryption process a different number is used that is deterministic but changing the public key, it will result in a different number in each execution. The process for generating the random number is a temporary solution because it is not secure to use it in real applications.

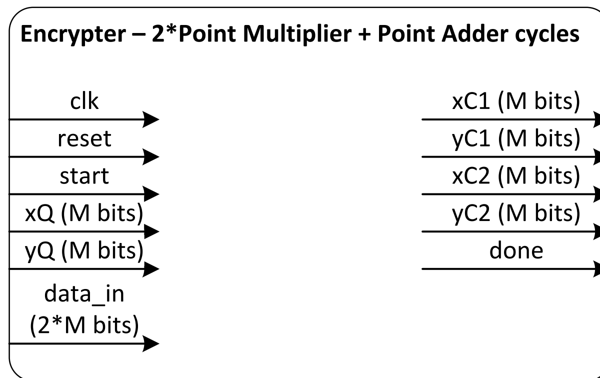


Figure 33. Module interface of the ECC encrypter.

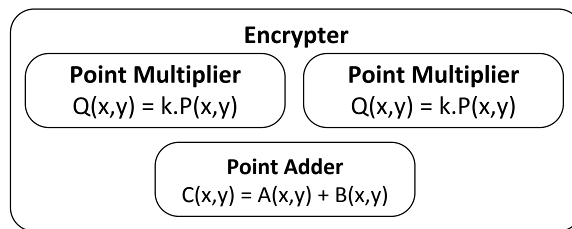


Figure 34. Composition of modules to implement the ECC encrypter.

4.3.6. Data Decrypter

The decrypter module is very similar to the encrypter but requires only a point subtraction and a point multiplication for the decryption operation, (see Figure 36). Thus, this module occupies a smaller area. However, it can also take up to $3*m^2+3*m$ clock cycles to execute. Figure 35 shows the module interface. It receives the number d that is the private key and the encrypted data to be decrypted that is composed by points C1 and C2.

4.4. Simulation, Validation and Synthesis

After all modules presented in the Sections 4.2 and 4.3 were described and implemented in VHDL, they were functionally simulated in Modelsim to verify their correct operation. First, each module of the finite field arithmetic level was simulated alone with several different data in their inputs (typical and cornered cases) and the results were compared with previous results obtained in software when the same operations were performed. Afterwards, point addition and point multiplier modules were simulated and validated with similar steps. Several operations with different data were performed and their results were compared with those of the same operations performed in software. All these comparisons to validate the hardware implementation were possible because software and hardware implementations were developed to be compatible. Thus, in future applications, it will also be possible to use devices with the software implementation and/or the hardware implementation interchangeably.

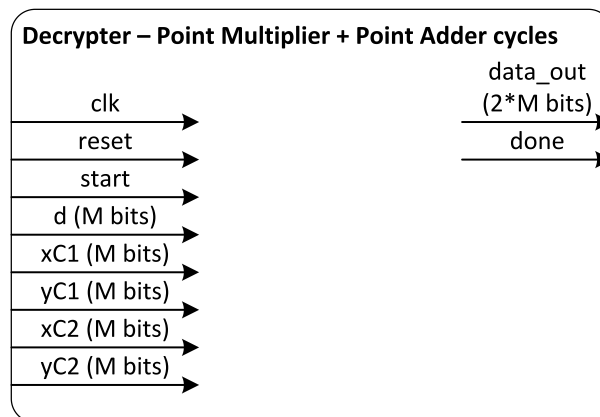


Figure 35. Module interface of the ECC decrypter.

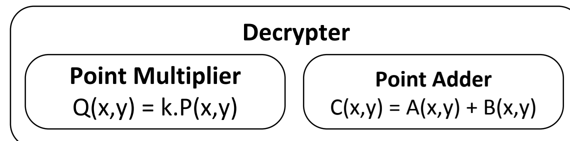


Figure 36. Composition of modules to implement the ECC decrypter.

This Section exemplifies the validation process by depicting a simulation of an encryption and a decryption operation, to demonstrate how the main modules work. First, a testbench was developed that instantiates one encrypter and one decrypter. To simplify the demonstration, the key generation process is not shown, but the private key is pre-defined in the testbench, while the public key is generated in the start of the simulation. The simulation initially encrypts “data in” defined by the test bench and afterwards it decrypts the output of the encrypter, testing the input data of encrypter and the output data of the decrypter for equality. This simulation uses 163-bit keys. The elliptic curve employed was sect163k1 and the “data in” block size was 326 bits. The finite field multiplier was configured to compute 8 bits per clock cycle. Thus, each finite field multiplication takes 21 cycles, each division takes 326 cycles, and each point addition takes up to 347 cycles. Overall, each multiplication takes up to 56561 cycles. The clock frequency was arbitrarily defined at 100 MHz, to facilitate comparisons with works in the literature.

Figure 37 and Figure 38 show the simulation waveforms. It is possible to see the main signals of the encrypter and decrypter modules. Figure 37 shows the elapsed time between the start and end of the encrypter operation, 429010 ns which means 42901 clock cycles at 100 MHz.

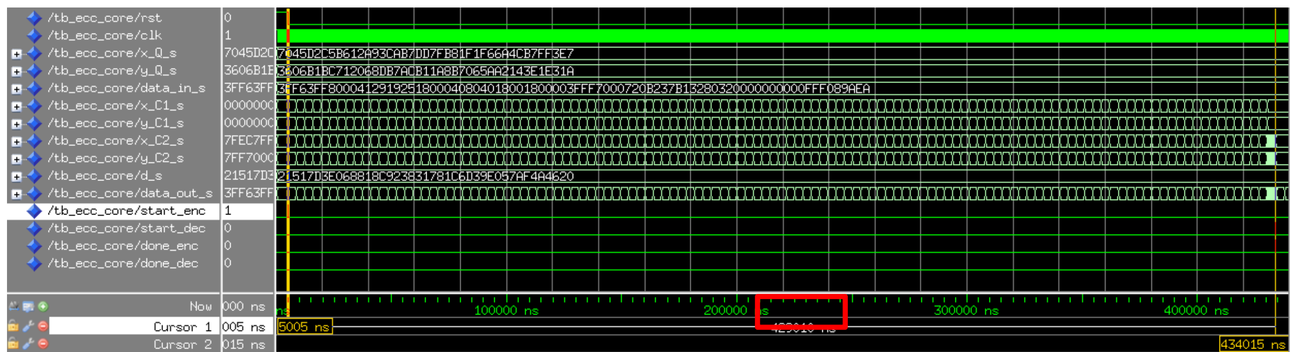


Figure 37. Simulation of the encrypter module.

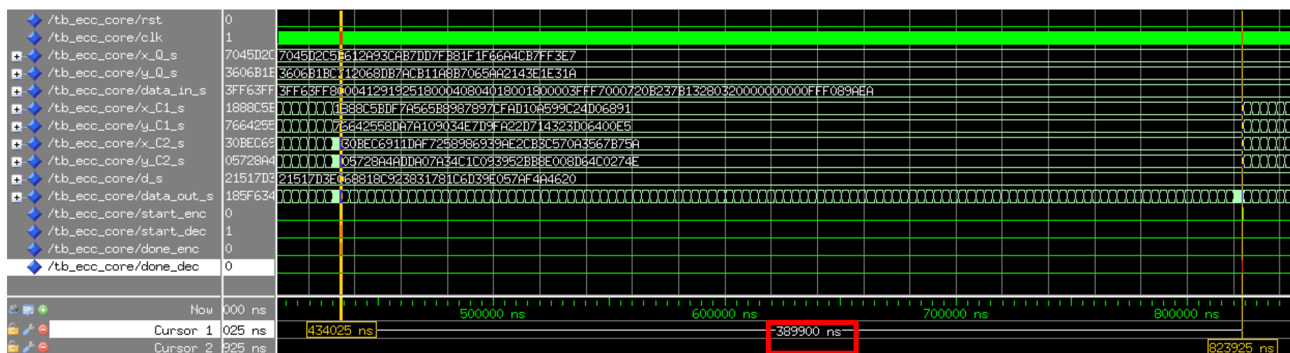


Figure 38. Simulation of the decrypter module.

Figure 38 shows the decryption simulation that was performed after the encryption with the output data of the encrypter module. It is also possible to see here the elapsed time between the start and end of the decryption process, 389900 ns or 38990 clock cycles. Unfortunately, it is not possible to see the “data in” and “data out” in these waveforms to compare their values and see that they are equal. However, when using the Modelsim, it was possible to see and confirm that “data in”, which was encrypted by the encrypter module, was equal to “data out”, which the decrypter module decrypted. Repeating this process both modules were validated with several different values.

The decryption process is usually a little faster than the encryption, as it was possible to notice in this demonstration and is explained in the literature, because the time to perform a point adder operation depends on the input values, varying for different values. The decryption algorithm usually computes the values used to decrypt faster.

Table 9 shows some preliminary results obtained from the synthesis of the developed hardware, the same hardware that was simulated

Table 9. Initial synthesis results for the ECC encrypter and decrypter, targeting a Xilinx XC5VLX330T FPGA.

Logic Utilization	Used	Available	Utilization
Slice Registers	13993	207360	6%
Slice LUTs	25860	207360	12%
Fully Used LUT-FF pairs	12211	27642	44%

The maximum frequency achieved in this first synthesis was 115 MHz. Considering the FPGA utilization, it is clear that this FPGA support bigger circuits and, because this, it was possible to optimize and parallelize more the algorithms to achieve higher performance. By using the timing results obtained in the simulation, it is possible to compare this hardware to the software implementation discussed in Section 1.2. The RELIC library provides a benchmark software, which measures the time of the main library operations. Thus, it is possible to run and compare these times with those obtained in the previous simulation just to analyze how much speedup is achievable. Table 10 shows the benchmark results obtained for all elliptic curves operations available in the library and the operations highlighted with the color red are those used to encrypt and decrypt any data. This benchmark was performed in a notebook with an Intel i7 processor and considering the elliptic curve sect163k1, which was the same used in the previous simulation.

Table 10. RELIC benchmark and modules execution time.

Curve NIST-K163:	
Utilities:	**Time (ns)**
BENCH: ec_null	16
BENCH: ec_new	20
BENCH: ec_free	5
BENCH: ec_is_infty	9
BENCH: ec_set_infty	71
BENCH: ec_copy	72
BENCH: ec_cmp	32
BENCH: ec_rand	997487
BENCH: ec_is_valid	6441
** **Arithmetic:**	
BENCH: ec_add	17849
BENCH: ec_sub	18133
BENCH: ec_dbl	17911
BENCH: ec_neg	74
BENCH: ec_mul	805186
BENCH: ec_mul_gen	1037673
BENCH: ec_mul_pre	8345607
BENCH: ec_mul_fix	970056
BENCH: ec_mul_sim	4624314
BENCH: ec_mul_sim_gen	4623384
BENCH: ec_map	63830
BENCH: ec_pck	14670
BENCH: ec_upk	23172

Considering that the encrypt operation is composed by two multiplications and one addition, it is possible to sum the time of these operations and estimate how much would be the time for one encrypt operation. The same can be done for the decrypt operation, which is composed by one multiplication and one subtraction. Table 11 shows the sum of times for the software operations to encrypt and decrypt data, the time achieved in the previous simulation, and the estimated speed up of the hardware to the software. By analyzing these results, it is clear that a simple implementation in hardware of ECC is already faster than a simple, sequential software implementation. However, this architecture can be improved to increase data throughput much more by parallelizing more operations or creating a pipeline.

Table 11. Hardware and software comparison.

Hardware vs. Software			
Operation	**Time (ns)**		**Speed Up**
	Software	**Hardware**	
Encrypt	1.628.221	429.010	3,80
Decrypt	823.319	389.900	2,11

4.5. Exploring the Flexibility of the Soft IP Core for ECC

As explained in the previous sections, the soft IP core developed is highly parameterizable. To explore its configurability, post-synthesis timing simulations of the point multiplier were conducted in Modelsim, varying several parameters and measuring the average number of clock cycles taken to compute a single point multiplication for the 5 Koblitz curves recommended by NIST. All configurations simulated were also synthesized to obtain detailed area, operating frequency and power consumption statistics. Synthesis results are based on the Xilinx ISE 14.1 XST tool, targeting a Virtex-7 XC7V2000T FPGA with speed grade -2, which is one of the most advanced FPGAs currently available. All configurations were also synthesized to the STMicroelectronics 65nm CMOS technology using Cadence Encounter and the foundry standard cell library in its Low Power, High Threshold version (LPHVt). Table 12 collects this encompassing set of results.

Table 12. Modelsim simulation results: synthesis results for ISE 14.1 XST and Cadence Encounter for 65nm CMOS.

Parameters		**Average Timing (clock cycles)**		**FPGA - XC7V2000T-2FLG1925**				**CMOS - 65 nm**					
M in bits	**W in bits**	**Multiplications**	**Point Multiplications (k*P)**	**Max. Clock Freq. (MHz)**	**Clock Period (ns)**	**Slice (LUTs)**	**Point Multiplication k*P (ns)**	**Max. Clock Freq. in MHz**	**Clock Period (ns)**	**Gates**	**Leakage Power (mW)**	**Dynamic Power (mW)**	**Point Multiplication – k*P (ns)**
163	1	163	48,863	223.326	4.478	5,768	218,797	1,338	0.747	16,652	1,241	29,979	36,520
	21	8	29,463	223.328	4.478	9,976	131,928	1,329	0.752	34,418	2,239	58,896	22,170
	82	2	23,061	222.027	4.504	18,852	103,866	1,342	0.745	72,895	4,174	90,605	17,185
233	1	233	103,449	225.405	4.436	6,737	458,948	1,355	0.738	23,744	1,741	42,086	76,347
	30	8	55,889	225.405	4.436	14,721	247,95	1,342	0.745	58,333	3,642	91,942	41,647
	117	2	45,449	225.405	4.436	32,271	201,633	1,351	0.740	136,846	7,985	140,210	33,642
283	1	283	164,823	192.945	5.183	8,245	854,249	1,283	0.779	29,458	2,215	51,795	128,467
	36	8	85,367	184.733	5.413	19,866	462,111	1,283	0.779	77,858	4,784	116,741	66,538
	142	2	70,469	184.114	5.431	45,510	382,747	1,297	0.771	195,382	11,188	190,865	54,333
409	1	409	314,085	147.358	6.786	11,229	2,131,442	1,126	0.888	36,272	2,567	62,047	278,939
	52	8	142,045	147.358	6.786	33,863	963,945	1,058	0.945	127,355	6,641	152,607	134,259
	205	2	123,829	147.358	6.786	85,447	840,328	1,003	0.997	360,755	18,096	281,859	123,459
571	1	571	654,543	111.506	8.968	16,337	5,870,025	889	1.125	48,691	3,187	71,508	736,269
	72	8	267,205	111.506	8.968	58,164	2,396,329	896	1.116	221,393	11,116	230,604	298,220
	286	2	251,287	111.506	8.968	162,251	2,253,574	844	1.185	673,526	29,623	414,768	297,734

Table 12 shows the average timing to compute one single point multiplication (k*P) for simulations in Modelsim with different parameters. The first column specifies the key size (M) in bits, according to the employed Koblitz curve. The second column defines the finite field multiplier kind, based on how many bits (W) are computed at each clock cycle. The third column is “M/W”, that is the time to compute a finite field multiplication in clock cycles.

The fourth column shows the average time obtained for a point multiplication in each configuration. As expected, the bigger the finite field multiplier, the less clock cycles are necessary to perform each multiplication and inversion. Consequently, point multiplication is also performed in less clock cycles. The Table also shows the FPGA synthesis results, for all multiplier architectural choices, and the synthesis results for the same architectures in the mentioned ASIC technology. The fifth and ninth columns show the maximum predicted operating frequency and the sixth and tenth the associated clock period, in nanoseconds. Considering the measured number of cycles in the fourth column, the eighth and fourteenth columns show the average time in ns for executing a point multiplication. The seventh column shows the amount of used LUTs in the FPGA, from 5.768 to 162.251 LUTs, which correspond to less than 1% and to up to 13% respectively of the target FPGA utilization. The eleventh column contains the needed number of gates for the point multiplier for the ASIC implementation. Finally, the twelfth and thirteenth columns show the estimated leakage and dynamic power, while Figure 39 shows the estimated total power for the ASIC implementations. Table 12 demonstrates that it is possible to configure the parameters according to the application constraints of area, power and performance. Again, it is possible to achieve higher performance using the same design and just setting parameters according to the desired security level, while considering constraints of area and performance. As a conclusion, the ECC soft IP core developed in this work demonstrates to be very flexible, due to the possibility of being configured with different sets of elliptic curves and parameters for the basic operations over finite fields.

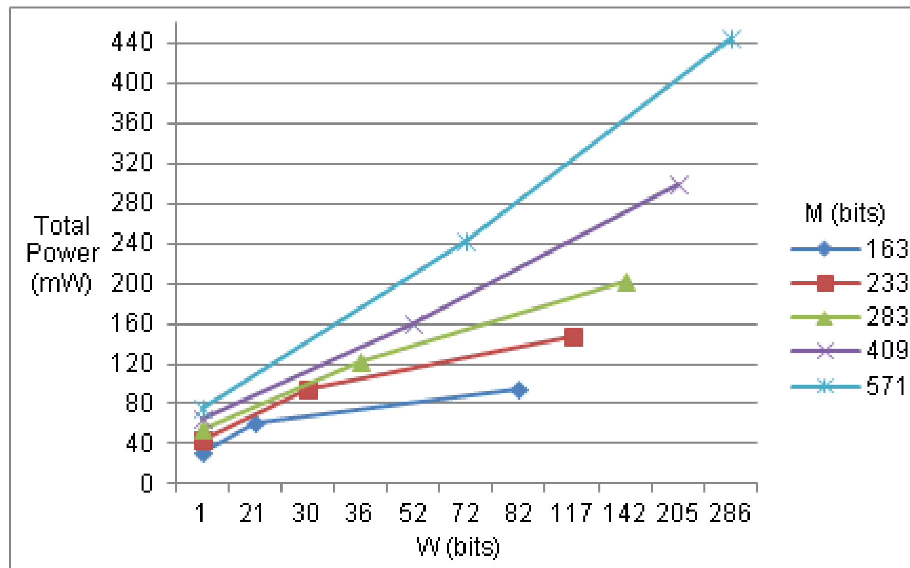


Figure 39. Total power consumption for ECC ASIC implementations.

4.6. Comparing the Results with Related Work

After exercising the soft IP core using several different parameters and elliptic curves, it is possible to compare the obtained results with some related works in the state of the art. Not all related works could be compared, because some of these implemented very specific optimizations and it would not be fair to compare them with the flexible soft IP core proposed here. Thus, only similar works were considered. Table 13 compares the main results obtained here with some related works. Only the higher performances of each related work

and each architecture of this work are compared. This Table also specifies the coordinate representation, the target platform and the maximum achieved frequency, to allow a fair comparison among all works.

First, there is a comparison with the Li et al. [LI08] work that implemented a point multiplier over GF(2^{283}) for the Koblitz curve K-283, using polynomial bases and projective coordinates. Their point multiplier achieved a performance a little better than ours, 0.304 ms, while ours gives 0.382 ms. However, their implementation has almost the same area even using projective coordinates, which usually occupies less area, noting that LUTs for Virtex-7 have 6 inputs while for Virtex-4 have 4 inputs. Second, we compared our results to the work of Dias et al. [DIA13] that implemented a point multiplier for the Koblitz curve K-163, using polynomial bases and affine coordinates. Their work is the most similar to that implemented here. Comparing the results, their implementation achieved practically the same performance than ours, but their point multiplier occupied much more area, even noting that LUTs for Virtex-7 have 6 inputs while for EP2S have 4 inputs, demonstrating that the work implemented here is more optimized and compact. Furthermore, we compared results obtained by Loi and Ko [LOI13] that implemented a crypto processor to support all the five Koblitz curves in the same design without need to resynthesize, using polynomial bases and projective coordinates. It is possible to note their implementation is very compact because it occupies a smaller area, even considering the difference of inputs of the FPGAs. This can be explained by their use of projective coordinates. However, the performance achieved by their crypto processor is worse than the one achieved in our work, which is between two and three times faster than theirs is.

Table 13. Comparisons of performance with related works.

Work	Coordinate System	Koblitz Curves GF(2^m)	Platform	Max. Freq. (MHz)	Area (LUTs or Gates)	Q=k*P (ms)
Li et al. [LI08]	Projective Coordinates	K-283	Virtex-4 XC4VFX140-11	171	51,094	0.304
Dias et al. [DIA13]	Affine Coordinates	K-163	2 Altera FPGAs (EP2S180F1020 C4 and EP2S180F1020C 3)	250	216,288	0.100
Loi and Ko [LOI13]	Projective Coordinates	K-163	Virtex-4 XC4VFX12	155	3,815	0.273
		K-233				0.604
		K-283				0.735
		K-409				1.926
		K-571				4.335
This Work	Affine Coordinates	K-163	Virtex-7	222	18,852	0.103
			65nm CMOS	1,342	72,895	0.017
		K-233	Virtex-7	225	32,271	0.201
			65nm CMOS	1,351	136,846	0.033
		K-283	Virtex-7	184	45,510	0.382
			65nm CMOS	1,297	195,382	0.054
		K-409	Virtex-7	147	85,447	0.840
			65nm CMOS	1,003	360,755	0.123
		K-571	Virtex-7	111	162,251	2.253
			65nm CMOS	844	673,526	0.297

According to the achieved results, Table 13 demonstrates that the soft IP core for ECC developed in this work can achieve higher performance, even when compared to other

works, which are not so flexible and/or use projective coordinates. This occurs mostly because the FPGA and ASIC technologies employed here are clearly more advanced (at least two generations ahead for FPGAs), but the improved architecture design are also expected to contribute to achieve a comparable performance as the simulations results present the average timing in clock cycles. Finally, Table 14 shows some synthesis results for the complete ECC architecture depicted in Figure 20 (including the datapath and associated controller), which is a complete core for ECC in hardware. These results are useful to give an idea of how much area may be necessary to implement a complete system that will use the ECC soft IP core.

Table 14. Synthesis results for a complete ECC soft IP core.

Parameters (bits)		**FPGA XC7V2000T-2**		**CMOS 65 nm**	
M	**W**	**Clock Frequency**	**LUTs**	**Clock Frequency**	**Gates**
163	1	168 MHz	27,297	909 MHz	77,243
	82	158 MHz	103,468	899 MHz	402,587
233	1	199 MHz	39,559	909 MHz	106,141
	117	199 MHz	192,637	906 MHz	756,057
283	1	171 MHz	48,307	858 MHz	137,820
	142	156 MHz	268,095	895 MHz	1,093,746
409	1	147 MHz	64,393	893 MHz	192,421
	205	147 MHz	506,902	869 MHz	2,147,644
571	1	106 MHz	92,459	855 MHz	280,220
	286	95 MHz	964,039	883 MHz	4,062,631

5. SECURE COMMUNICATION SYSTEM

This Chapter presents the hardware prototype developed to implement a secure communication system that uses ECC to encrypt the communication. It details the hardware architecture, and its main modules. Besides, it shows how to use the ECC IP Core presented in the previous Chapter in this system. Simulations and FPGA prototyping served to validate and demonstrate system operation. However, this secure communication system presented here has many points that are not secure because some parts were implemented in an easier way due to the lack of time for its implementations. So, the secure communication system is used as a demonstration of application to use the soft IP core for ECC.

5.1. Hardware Architecture

The hardware architecture was defined based on the requirements that should be met to enable and support the secure communication system. First, this system need to be transparent to the connected devices and to the network, so that the computer connects in the same way if it is connected directly to the network, and the network identifies the computer as normally connected. Next, the communication system should be able to encrypt one link of TCP/IP communication, so that all frames from a specific connection between two computers through a specific TCP port should be encrypted. All other Ethernet frames are just forwarded without any modification. The defined hardware architecture appears in Figure 40.

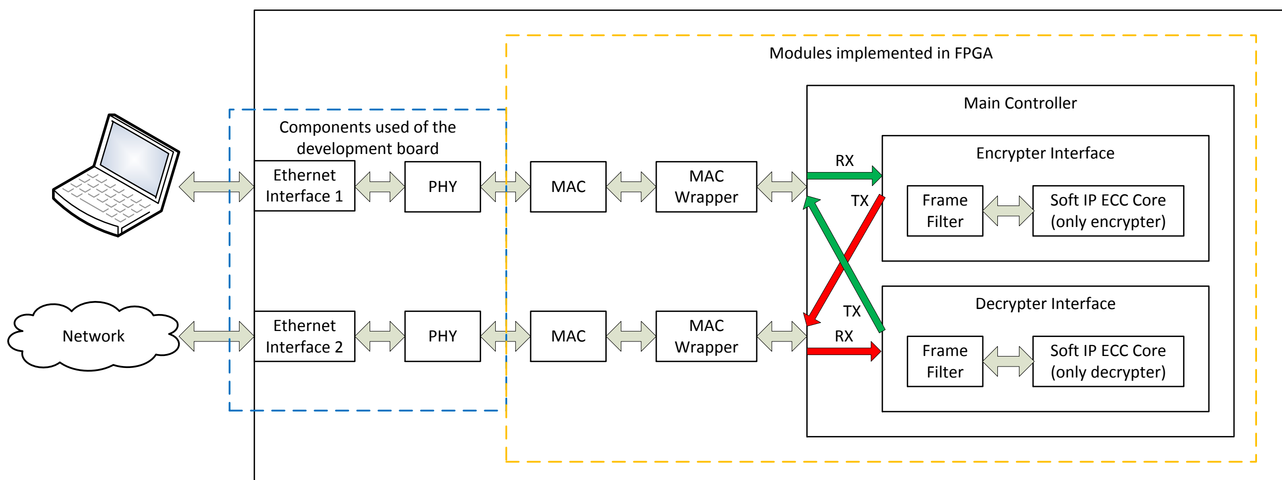


Figure 40.Hardware architecture of the secure communication system.

There are two Ethernet interfaces, one exclusive to a local computer and the other for the network, because there are only one encrypter core and one decrypter core. In this way, the frames of a specific connection coming from the computer interface are encrypted and sent out to the network; the frames of a specific connection coming from the network interface are decrypted and sent out to the computer. In the hardware architecture, some modules that were ready to use in the development board (see Chapter 3). These include the Ethernet interfaces, as well as the PHY and MAC modules. The other modules were developed and implemented in the FPGA, including the MAC wrapper and the main control-

ler with the ECC cores. The MAC wrapper is responsible to establish the connection between the main controller and the external components. It receives and sends all signals that compose an Ethernet interface, and verifies if all the received and sent Ethernet frames are correct. The main controller creates the cross connection between the two interfaces. Frames coming from one interface are sent out to the other interface. Meanwhile, the encrypter and decrypter cores filter the specific frames from a connection to encrypt or decrypt them and all other frames are just forwarded. These are the main modules of the hardware architecture. The next Section details the modules implemented in FPGA, explaining their operation.

5.2. Hardware Modules

The hardware modules detailed here were described in VHDL and some modules contain IP cores provided by Xilinx, such as block RAMs and FIFOs, to simplify the implementation and to optimize the utilization and performance of the FPGA. These modules were specifically designed for the Virtex 5 LXT330 device, as specified in the proposed project detailed in Chapter 3. Using only simple modifications it is possible to use the same design for other FPGAs and different applications.

5.2.1. Xilinx MAC Ethernet Wrapper

To use the Ethernet interfaces and the embedded components of the development board such as PHY and MAC Ethernet modules, Xilinx provides an IP core that acts as a MAC wrapper. Using this wrapper it is possible to configure the Ethernet interfaces, select the operation mode and speed, and receive and transmit the Ethernet frames to/from each interface. In this implementation, interfaces were configured to operate in the mode Media Independent Interface (MII) and at a speed of 100 Mbits/s full duplex, which is the simpler operation mode that can be used for Ethernet interfaces. Then, in conformance to the MII standard, the MAC wrapper operates at 25 MHz, the frequency to provide a communication link of 100 Mbits/s. Therefore, communication between the MAC wrapper and the main controller occurs through a memory block operating in FIFO mode. Thus between each pair of receiver-transmitter interfaces there is a FIFO, which enables the main controller to operate at higher frequencies and not only at 25 MHz as the MAC wrapper. Another advantage of using this wrapper is that it implements all operations needed to send, receive and check frames. Sent and received frames through the FIFOs are composed just by MAC addresses, Ethernet type and payload. The MAC wrapper removes the preamble, start of a frame delimiter and the frame check sequence upon receiving the frame, and these are computed and inserted when the frame is transmitted. Thus, the main controller can create or modify the Ethernet frames as needed.

5.2.2. Main Controller

The main controller just instantiates the encrypter and decrypter interfaces. It also controls the direction of the Ethernet frames, to create the cross connection between the two interfaces. All frames received at the Ethernet Interface 1 are sent to the encrypter interface, and all frames received at the Ethernet Interface 2 are sent to the decrypter interface, and so on. Therefore, a cross connection exist between the Ethernet interfaces.

The development board in use provides a clock frequency of 50 MHz, although the main controller module can operate at higher frequencies. So, the main controller also instantiates a digital clock manager (DCM) provided by Xilinx to create higher clock frequencies. With the goal to support a clock frequency for the main controller signals and another one to the soft IP core for ECC, two independent DCMs are used, and it is possible to configure a specific clock signal to the soft IP core for ECC and another for all other control modules. The next sections detail these implementations.

5.2.3. Frame Filter

The frame filter module is responsible for analyzing all incoming frames and for filtering those frames which are from a specific TCP/IP connection to be encrypted or decrypted. Any other frames are just forwarded to the transmitter with no modifications. There are two instances of this module, one in the encrypter interface and another in the decrypter interface. To explain how this module was implemented, it is important to know what information is present in each Ethernet frame. Ethernet frames are composed by the MAC addresses of destination and source; the Ether type indicates which type of protocol is in use by this frame, such as IPv4, ARP, IPv6 or any other; and the payload can use up to 1500 bytes. In this work, as the secure connection link will be done through a TCP/IP connection, only IPv4 Ethernet frames are considered. Figure 41 illustrates the composition of these Ethernet frames to encrypt. In this case, payload contains the IPv4 header, TCP header and the data frame.

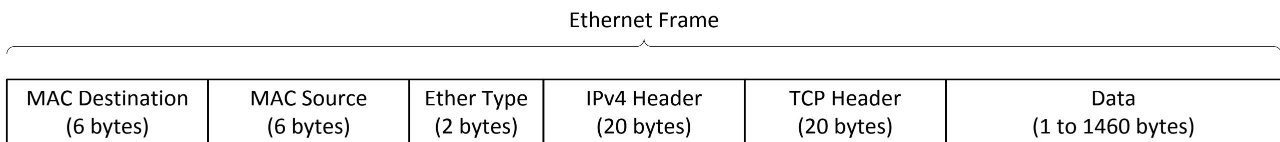


Figure 41. IPv4 Ethernet frame using the TCP protocol.

To help the explanation about the frame filter operation, Figure 42 illustrates the IPv4 header, and Figure 43 details the TCP header. Considering the information provided in this kind of Ethernet frame, the frame filter is configured to filter frames that have a specific MAC address of destination and source, and that also have a specific source or destination port in the TCP header. The only difference between the frame filter used in the encrypter interface from that used in the decrypter interface is that in the first one the frame filter verifies the destination port of the TCP header, while in the other the frame filter verifies the source port of the TCP header. As an example, if a frame coming from the Ethernet interface 1 is an IPv4 Ethernet frame using the TCP protocol and has the MAC addresses and the destination TCP port corresponding to its configured information, the frame filter sends this frame to the encrypter interface. Otherwise, the frame is just forwarded to the transmitter.

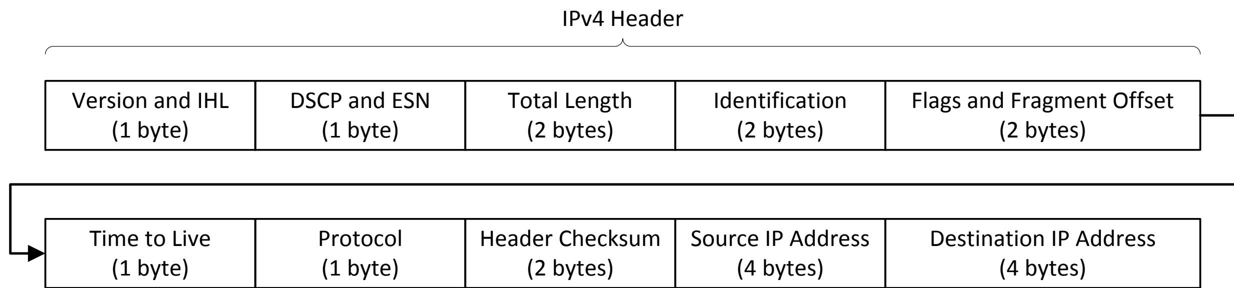


Figure 42. Detailed information of the IPv4 header.

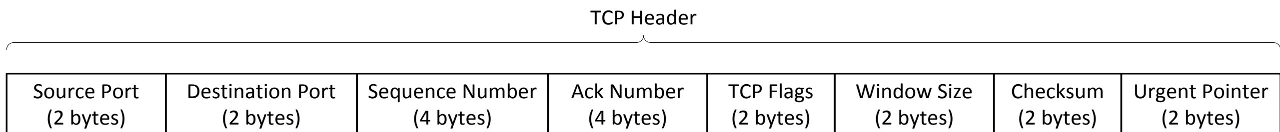


Figure 43. Detailed information of the TCP header.

As already mentioned, the frame filter can be configured during its operation. That is, when the frame filter is initialized, it does not have any MAC addresses or TCP port configured to consider in the filtering process. While the frame filter is not configured, all frames are just forwarded between Ethernet interfaces. The frame filter is configured through an Ethernet frame that must be sent from a computer directly connected to the development board. Therefore, if the frame filter receives an Ethernet frame in the format illustrated in the Figure 44 then it is configured to filter the next frames that have the MAC addresses and TCP port highlighted by the red color in the Figure. This same Ethernet frame must be sent to both interfaces of the board to configure both frame filters.

MAC Destination	MAC Source	Ether Type	Conf. MAC Destination	Conf. MAC Source	Conf. TCP Port
DA:02:03:04:05:06	5A:02:03:04:05:06	0x1234	XX:XX:XX:XX:XX:XX	XX:XX:XX:XX:XX:XX	0xXXXX

Figure 44. Ethernet frame to configure the frame filter.

Accordingly, if the frames are being filtered to be encrypted or decrypted, the frame filter writes these frames to FIFOs that will be read by the encrypter interface or respectively by the decrypter interface. These FIFOs have 32 kilobytes of block memories each. Next, data will be encrypted or respectively decrypted. After frames are encrypted or decrypted, the encrypter interface or respectively decrypter interface will write these to other FIFOs so that the frame filter sends them out to the expected transmitter, as Figure 45 illustrates.

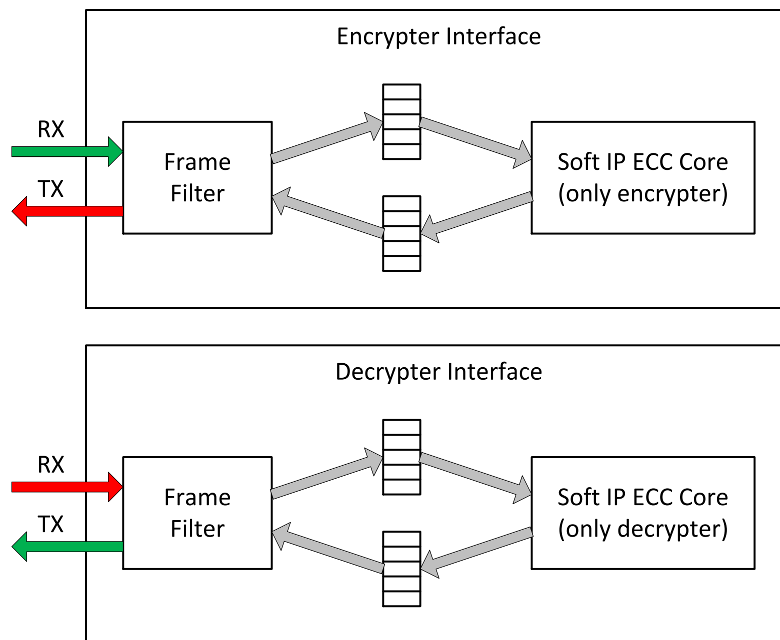


Figure 45. Detailed datapath of encrypter and decrypter interfaces.

5.2.4. Encrypter Interface

The module Encrypter Interface, illustrated in Figure 46, is responsible for receiving all frames coming from the computer and filtering them, to select frames to encrypt. Next, it encrypts these and sends the encrypted frames to the network. It comprises a frame filter and an instance of the soft IP core for ECC containing only the module necessary to encrypt data. The frame filter is the module that receives all frames and selects among these those that must be encrypted before sending. This module was detailed in Section 5.2.3. The soft IP ECC core is the module responsible for encrypting data of all frames and is the same presented in Chapter 4, but some modifications were done to enable it to encrypt Ethernet frames more efficiently, as explained next.

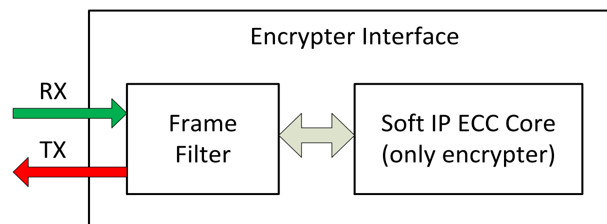


Figure 46. Module of the encrypter interface.

The communication between the frame filter and the soft IP core is through FIFOs which are controlled by the Encrypter Interface. Considering that the soft IP core for ECC can encrypt up to 2*m bits of data each time, remembering that m is the key size in use for ECC, and that the data of the frames can be from 1 to 1460 bytes, these data must be split in blocks of 2*m bits. However, there is another problem. Consider an elliptic curve represented over GF(2^{163}). It will have a private key of 163 bits and a public key with 326 bits. Thus, the soft IP core for ECC will encrypt 326 bits of data each time, and these will result in 652 bits of encrypted data, because it is composed by points C1 and C2 of the elliptic curve. The solution created to encrypt all these data bytes is to break the frame data in

blocks of 2*m/8 bytes, rounding down. Then, in case data blocks would be 326 bits, actually 320 bits are used that result in blocks of 40 bytes, as Figure 47 illustrates.

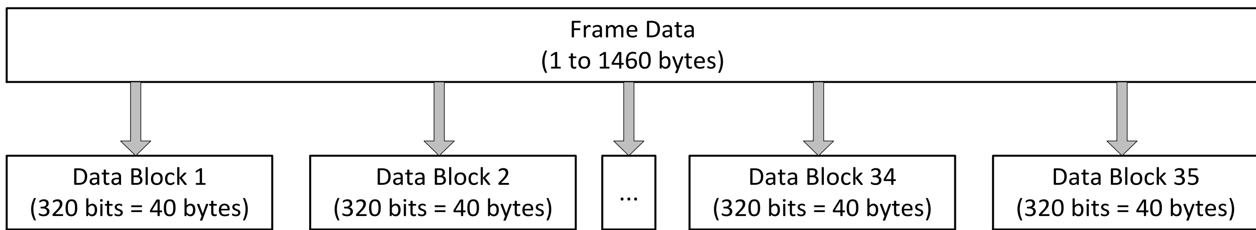


Figure 47. Splitting the frame data to encrypt.

Now, considering that the frame data vary from 1 to 1460 bytes, it could also result in blocks with less than 40 bytes. So, the 6 bits that were discarded in the division of the frame data are used to specify each block size. As the maximum block size of data, in this case, is 40 bytes then it is possible to represent it with 6 bits, as Figure 48 illustrates.

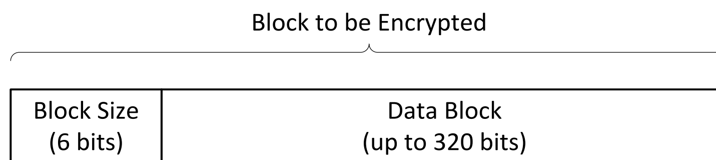


Figure 48.Format of the data block to encrypt.

When all information on the data block is encrypted, it generates 652 bits of encrypted data. This is another problem, because if all frame data is doubled after encrypting, then it would not fit in only one frame. So, revising the ECC algorithms, it is possible to note that points C1 and C2 are needed to decrypt the data, but point C1 is just the multiplication of a random number with the point generator of the elliptic curve. So, if the same random number could be used for all data blocks of the same frame, then it would only be necessary to use 2*m bits to send C1, which would then be the same for all other blocks, and other blocks even when encrypted will use only 2*m bits for each. Thus, the encryption process for Ethernet frames can be depicted in the following steps: (*i*) First, the Ethernet frame is received and its frame data is divided in blocks of 40 bytes. (*ii*) Second, when the first block is encrypted, points C1 and C2 are generated. Point C1 is stored and this point will be the same for all other blocks. (*iii*) Next, point C2 is also stored in the encrypted data frame. (*iv*) Afterwards, all other data blocks are encrypted, one at a time, but then only points C2 are generated. This corresponds to their encryption, which used the same point C1 generated in the encryption of the first block. (*v*) Finally, two Ethernet frames are created, the first one to send the information of point C1 and the second one used to send all the encrypted blocks, which are all the C2 points generated for the data blocks of the original frame.

The next three figures illustrate the structure of the expected frame to encrypt, in Figure 49, and how the generated encrypted frames are composed, in Figure 50 and Figure 51. In this case, it is important to note that the maximum data size of the received frames is limited to 1400 bytes, because splitting 1460 bytes in blocks of 326 bits, or 41 bytes (by rounding up), would result in 36 blocks of 41 bytes, where 1 byte indicates the block size and the other 40 bytes contain encrypted data. Now, 36 blocks of 41 bytes would be bigger than 1460 bytes, the maximum size reserved to frame data. Accordingly, it was limited to 35 blocks of 41 bytes, which results in 1400 bytes of data encrypted to 1435 bytes.

MAC Destination (6 bytes)	MAC Source (6 bytes)	Ether Type (2 bytes)	IPv4 Header (20 bytes)	TCP Header (20 bytes)	Data (up to 1400 bytes)

Figure 49. Ethernet frame received to be encrypted.

MAC Destination (6 bytes)	MAC Source (6 bytes)	Ether Type (2 bytes)	IPv4 Header (20 bytes)	TCP Header (20 bytes)	Point C1 (326 bits)

Figure 50. Encrypted Ethernet frame 1.

MAC Destination (6 bytes)	MAC Source (6 bytes)	Ether Type (2 bytes)	IPv4 Header (20 bytes)	TCP Header (20 bytes)	Encrypted Data (up to 1435 bytes)

Figure 51. Encrypted Ethernet frame 2.

Finally, at the end of the encryption process of all blocks, the Encrypter Interface mounts both Ethernet frames as illustrated in Figure 50 and Figure 51. These represent the encryption of the received frame, which is sent to the frame filter that transmits them to the network.

5.2.5. Decrypter Interface

The Decrypter Interface is very similar to the Encrypter Interface, as Figure 52 illustrates. It is responsible to do the reverse process, receiving the encrypted frames from the network, decrypting them and sending them to the computer. The decrypter interface is composed by another instance of the Frame Filter and the soft IP core for ECC with only the hardware necessary to decrypt data. In this case, the Frame Filter is a little different from the previous module, since it is configured to filter the frames coming from the network that must be decrypted. Accordingly, it verifies if the MAC address and the source TCP port are the expected by its configuration. As explained in Section 5.2.4 the soft IP core for ECC was modified to enable and optimize the encryption of Ethernet frames. It was also modified to enable the efficient frame decryption process.

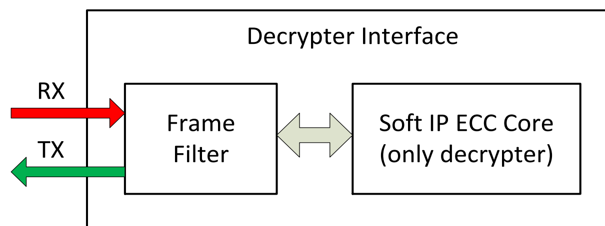


Figure 52. Module of the decrypter interface.

The decryption process is simpler than the encryption one because it does not need to worry about the Ethernet frame limits. Considering that it will receive the two frames that represent one decrypted frame, it cannot pass any limit. The decryption process is basically defined by the following steps: (*i*) First, the two frames with the structure depicted in Figure 50 and Figure 51 are received and stored (point C1 and the first point C2). (*ii*) Second, all C2 points are decrypted using the private key and the same point C1. Considering the same example of the previous Section that used an elliptic curve represented over GF(2^{163}), it generates blocks of 326 bits and according to the first 6 bits, that define the amount of data bytes that the block contains, up to 320 bits of decrypted data are stored. (*iii*) After decrypting all blocks, the Ethernet frame is generated, which is the same frame sent by the sender

device. (*iv*) Finally, the Decrypter Interface sends this frame to the frame filter that will send it out to the computer.

5.3. Simulation, Validation and Synthesis

After all these modules were described and implemented in VHDL, they were simulated and validated through Modelsim simulations. They were synthesized to the target FPGA, as described in Chapter 3. Figure 53 illustrates the produced design.

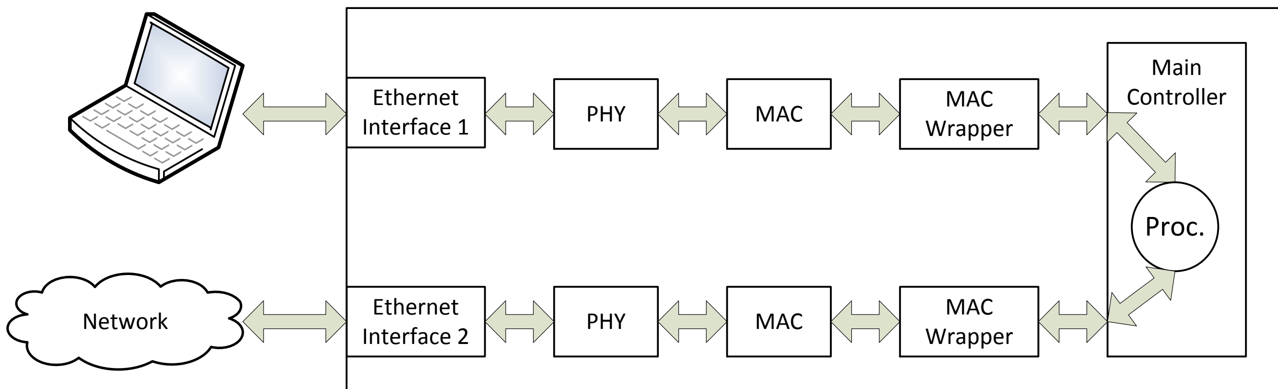


Figure 53. Basic design with only Ethernet modules and control.

It contains only the basic modules that enable receiving and sending all Ethernet frames without any modification. This environment was validated through simulations and prototyped in the development board.

After validating the Ethernet modules and confirming that it does not aggregate any significant overhead to receive and send frames, the design the modules Frame Filter, Encrypter Interface and Decrypter Interface were included in the environment, to enable the creation of a secure communication link, as defined in Section 5.1. To validate all modules, a testbench that injects some Ethernet frames in both interfaces was created. It simulates a communication between a computer and a network. At the Interface 1 random frames were injected that should not be encrypted. In addition, the testbench injects the frame to configure the Frame Filter of the Encrypter Interface and the frames that should be encrypted. Next, we verified that the Frame Filter was working as expected, filtering just the correct frames to encrypt and forwarding all other frames.

The Encrypter Interface was also verified, as it encrypted the expected frames and sent them to the Frame Filter, which sends them out to the transmitter of the network interface. Furthermore, in Interface 2, some random frames were also injected that should not be decrypted, as well as the frame to configure the Frame Filter of the Decrypter Interface and the frames encrypted by the Encrypter Interface. It was verified that the whole design was working properly, since the injected frames, which represent the frames of a secure communication link, were correctly encrypted, transmitted to the network, received in the Decrypter Interface and then decrypted, resulting in the same frames that were injected in Interface 1. During these simulations, the average timing of the encryption and decryption process was measured, indicating the added overhead to create a secure communication link, and this was also tested with several different parameters of the soft IP core for ECC that influence on its performance.

Table 15. Average timing for conducted simulations.

Parameters			Average Timing (clock cycles)					
			Encrypting			Decrypting		
M (bits)	W (bits)	Field Multiplication (clock cycles)	1st Block	Other Blocks	Largest Frame	1st Block	Other Blocks	Largest Frame
163	1	163	52.844	501	72.802	57.808	501	77.766
	33	5	28.379	268	40.415	31.013	268	43.049
	82	2	24.914	235	35.828	27.218	235	38.132
233	1	233	111.231	711	130.367	102.731	711	121.867
	30	8	60.063	383	71.655	55.499	383	67.091
	117	2	48.831	311	58.767	45.131	311	55.067
283	1	283	165.685	861	184.799	166.532	861	185.646
	32	9	88.309	458	99.766	88.753	458	100.210
	142	2	70.837	367	80.565	71.190	367	80.918
409	1	409	330.155	1.239	348.979	338.790	1.239	357.614
	32	13	165.235	619	175.999	169.530	619	180.294
	205	2	130.123	487	139.171	133.494	487	142.542
571	1	571	620.124	1.725	638.250	659.659	1.725	677.785
	32	18	318.564	885	329.130	338.779	885	349.345
	286	2	238.148	661	246.698	253.211	661	261.761

In this Table, the first three columns specify the used parameters, such as “M” that indicates the key size according to the used elliptic curve and “W” that indicates the number of bits that the finite field multiplier computes at each clock cycle. The Field Multiplication indicates how many cycles are needed to compute one finite field multiplication. The other columns indicate the average time, in clock cycles, measured during simulations for several events. The fourth and seventh columns show the average time to encrypt or decrypt the first block respectively. In both cases, the first block takes more time to encrypt or decrypt, because it is only in this part that a point multiplication is performed, as point C1 is generated or used only in this part, and for the other blocks just a point addition is executed or a point subtraction, which takes fewer cycles to perform. The sixth and ninth columns show the total timing to encrypt or decrypt the largest frame possible that has 1400 bytes of data.

Table 16 shows the synthesis results obtained for the same simulated designs. They were synthesized using the Xilinx PlanAhead 14.1 tool, targeting the Xilinx Virtex 5 LX330T FPGA device, the FPGA of the development board detailed in Chapter 3. The fourth column indicates the clock frequency used only in the soft IP core for ECC; all other modules use the 100 MHz clock frequency, as explained in Section 5.2.2, and since this is the same clock for all designs, it is ignored in the Table that compares the performance of designs with different parameters. The sixth column shows the number of slice LUTs used in the FPGA. It is possible to note that the bigger the finite field multiplier, the bigger will the whole design be. However, the average timing to perform an encryption or decryption also decreases, as the seventh and eighth columns show.

Table 16. Synthesis results for the complete design.

Parameters			FPGA - XC5VLX330T FF1738-2				
M (bits)	**W (bits)**	**Field Multiplication (clock cycles)**	**Clock Frequency (MHz)**	**Clock Period (ns)**	**Slice LUTs**	**Encryption of Largest Frame (ns)**	**Decryption of Largest Frame (ns)**
163	1	163	100,000	10,000	37.628	728.020	777.660
	33	5	100,000	10,000	70.366	404.150	430.490
	82	2	100,000	10,000	105.124	358.280	381.320
233	1	233	75,000	13,333	58.655	1.738.227	1.624.893
	30	8	75,000	13,333	105.264	955.400	894.547

To estimate the overhead added by the encryption and decryption processes for a secure communication system, the average throughput was estimated according to the average timing measured in the simulations and the expected clock frequency at which the soft IP core operates. Table 17 shows in the fifth and sixth columns the estimated throughput results for different parameterizations of the soft IP core. Considering the values in nanoseconds of encryption and decryption for the largest frame and the clock frequencies presented in the previous table, the total overhead is the sum of the timing results of encryption and decryption of the largest frame (1400 bytes). This is the additional time necessary to create a secure communication link. The estimated throughput was calculated based on the amount of bytes that are encrypted and decrypted per second, which in these cases is the value “1400 bytes / total overhead” converted to megabits per second, as presented in Table 17.

Table 17.Estimating throughput of the secure communication link, in Mbits/s.

Parameters			Throughput Estimation		Prototyping
M (bits)	**W (bits)**	**Field Multiplication (clock cycles)**	**Total Overhead (ns)**	**Estimated Throughput (Mbits/s)**	**Measured Throughput (Mbits/s)**
163	1	163	1.505.680	7,44	12,00
	33	5	834.640	13,42	19,10
	82	2	739.600	15,14	21,30
233	1	233	3.363.120	3,25	5,15
	30	8	1.849.947	5,92	4,71

After the simulations, these designs were synthesized and prototyped for the development board that is detailed in the next Section, but its results are presented in the last column of the Table 17. The next step is the hardware prototyping to validate the whole system.

5.4. Hardware Prototyping

After validating the designs through simulation, as described in the previous Sections, the same designs were prototyped in hardware, using the development board presented in Chapter 3. A method similar to that employed in the software version (see Section 1.2) helped validate these designs. The Iperf software is used to generate the Ethernet frames and to measure the throughput of the communication between two computers. The whole

communication system interconnects using Ethernet cables, two identical development boards and the laboratory network, as illustrated in Figure 54. The results of this validation process and the measurement of its performance appear in the last column of the Table 17. Almost all results obtained in the prototyping were better than those estimated values probably due to the greater variety of data being encrypted and decrypted. However, in the last result presented in the last row, the measured throughput was lower than the estimated maybe due to some network overload in the instant of the test or some problem in the secure communication system developed that it was not identified.

Figure 54. Hardware prototyping environment for the secure communication system.

Unfortunately, it was not possible to synthesize and prototype the other designs because the FPGA Virtex-5 used does not support all the communication system and the soft IP for ECC developed in this work. So, it is necessary to use bigger FPGAs to enable the prototyping of these bigger designs.

5.5. Comparing the Results with the Software Experiment

By comparing the results of the measured throughput with the results obtained in the software experiment, it is possible to note that the communication system implemented in hardware achieved a greater performance. According to the results of the software experiments, presented in Table 3 that shows the results of tests performed in software. Considering the test scenario 6 using a key of 160 bits and the wired LAN network achieved the throughputs up to 7.71 Mbits/s using two computers. Thus, considering the measured results of the Table 17, the communication system implemented in hardware achieved a speed-up among 1.55 to 2.76 times.

6. CONCLUSIONS

Considering the work developed here and the achieved results, it was demonstrated that using ECC implemented in hardware provides many advantages of performance and robustness when compared to software implementations. Furthermore, the soft IP core for ECC developed here and its results demonstrated that it is possible to create a design that can fulfill different constraints of area, power, performance and security level, depending on the requirements of specific applications. Thus, it is possible to integrate the soft IP core to many embedded devices with different purposes, such as entertainment devices or critical embedded devices, giving support to constraints of area and power for the smaller devices or to constraints of performance and security level for devices used in critical systems. The secure communication system described in Chapter 5 demonstrates the use of this soft IP core for ECC.

6.1. Applications and Viability

According to the results presented in Section 4.5, it is possible to conclude that the soft IP core for ECC can fulfill different sets of constraints due to its flexibility. Thus, almost any kind of applications can use it to provide ECC-based security by just setting the soft IP core configurations according to requirements. Therefore, very small devices, such as smartphones, tablets, smart watches, smart glasses or small robots, which require low area and low power, could use the soft IP core for ECC to provide a secure communication system, if they do not require very high performance. Other devices that require higher performance and high security level, such as critical embedded devices, security devices, computer servers, military computers, tactical robots or autonomous vehicles for example, can also use the same design of soft IP core for ECC by just setting the configurations to fulfill the higher performance and higher security level constraints. Finally, the secure communication system developed in Chapter 5 demonstrated that the soft IP core for ECC works and its use is viable for several applications.

6.2. Future Works

The work presented here was the first one conducted by the Author about cryptography and ECC implemented in hardware. Though it achieved good results, there are many parts that can be optimized and several other hardware architectures could be explored to achieve higher performance or lower area and power. Considering the soft IP core for ECC, there are other algorithms that implement the basic operations over finite fields that could be explored, such as those used for projective coordinates, analyzing their advantages and disadvantages according to specific requirements for different applications. Considering the communication system, many features and configurations can be implemented to enable the configuration of the encryption keys and the key exchange, which are not done in this version, because the keys are just defined in hardware, and it is not currently possible to change them after prototyping, which makes it extremely insecure. So, the soft IP core for ECC can be optimized and different algorithms could be added to increase its flexibility for

more constrained requirements of area, power and performance. In the communication system, a standard protocol could be implemented to increase its security level and some software tool to facilitate its configuration. In this way, there are many points that can be optimized to increase the encryption performance and the security and usability of the communication system.

REFERENCES

[ALM13] Almenares, F.; Arias, P.; Marín, A.; Díaz-Sánchez, D.; Sánchez, R. "*Overhead of Using Secure Wireless Communications in Mobile Computing*". IEEE Transactions on Consumer Electronics, 59(2), 2013, pp.335-342.

[ARA12] Aranha, D. F.; Gouvêa, C. P. L. "*RELIC is an Efficient LIbrary for Cryptography*". 2012. Captured in: http://code.google.com/p/relic-toolkit/, Jul 2012.

[BED02] Bednara, M.; Daldrup, M.; Gathen, J.; Shokrollahi, J.; Teich, J. "*Reconfigurable Implementation of Elliptic Curve Crypto Algorithms*". In: International Parallel and Distributed Processing Symposium (IPDPS´02), 2002. pp. 284-292.

[BRO09] Brown, D. "*Standard for Efficient Cryptography 1*". Certicom Research, May 2009, 144 p.

[DES09] Deschamps, J.; Imanã, J.; Sutter, G. "*Hardware Implementation of Finite-Field Arithmetic*". McGraw-Hill, 2009, 347 p.

[DIA13] Dias, M.; Gouveia, M.; Oliveira, J.; Muñoz, I. "*Bit-Parallel Coprocessor for Standard ECC-GF(2^m) on FPGA*". International Journal of Applied Mathematics, 26(2), 2013, pp. 241-262.

[DUG12] Dugan, J. "*Iperf*". Captured in: http://sourceforge.net/projects/iperf/, Oct 2012.

[FOU09] Fournaris, A. P.; Koufopavlou, O. "*Low Area Elliptic Curve Arithmetic Unit*". In: IEEE International Symposium on Circuits and Systems (ISCAS´09), May 2009, pp. 1397-1400.

[GUA02] Guajardo, J.; Paar, C. "*Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography and Codes*". Designs, Codes and Cryptography, 25(2), Feb 2002, pp. 207-216.

[GUA06] Guajardo, J.; Güneysu, T.; Kumar, S.; Paar, C.; Pelzl, J. "*Efficient Hardware Implementation of Finite Field with Applications to Cryptography*". Acta Applic. Mathematica, 93(1-3), Sep 2006, pp. 75-118.

[HAN04] Hankerson, D.; Menezes, A.; Vanstone, S. "*Guide to Elliptic Curve Cryptography*". Springer, 2004, 318 p.

[ITO88] Itoh, T.; Tsujii, S. "*A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases*". Information and Computation, 78(3), Sep 1988, pp. 171-177.

[JÄR11] Järvinen, K. "*Optimized FPGA-Based Elliptic Curve Cryptography Processor for High-Speed Applications*". Integration the VLSI Journal, 44(4), 2011, pp. 270-279.

[KEL09] Keller, M.; Byrne, A.; Marnane, W. P. "*Elliptic Curve Cryptography on FPGA for Low-Power Applications*". ACM Transactions on Reconfigurable Technology and Systems, 2(1), Mar 2009, 20 p.

[KOB87] Koblitz, N. "*Elliptic Curve Cryptosystems*". Mathematics of Computation, 48(177), Jan 1987, pp. 203-209.

[LAU04] Lauter, K. "*The Advantages of Elliptic Curve Cryptography for Wireless Security*". IEEE Wireless Communications, 11(1), Feb 2004, pp. 62-67.

[LI08] Li, H.; Huang, J.; Sweany, P.; Huang, D. "*FPGA Implementations of Elliptic Curve Cryptography and Tate Paring Over Binary Field*". Journal of Systems and Architecture, 54(12), Dec 2008, pp. 1077-1088.

[LOI13] Loi, K. C. C.; Ko, S. B. "*High Performance Scalable Elliptic Curve Cryptosystem Processor for Koblitz Curves*". Microprocessors and Microsystems, 37(4-5), Jun-Jul 2013, pp. 394-406.

[MAR10] Martínez, V. G.; Encinas, L. H.; Ávila, C. S. "*A Survey of the Elliptic Curve Integrated Encryption Scheme*". Journal of Computer Science and Engineering, 2(2), Aug 2010, pp. 7-13.

[MAS12] Masoumi, M.; Mahdizadeh, H. "*Efficient Hardware Implementation of an Elliptic Curve Cryptography Processor over* $GF(2^{163})$". World Academy of Science, Engineering and Technology, 65, May 2012, pp. 1223-1230.

[MIL86] Miller, V. "*Use of Elliptic Curve in Cryptography*". In: A Conference on the Theory and Application of Cryptographic Techniques (CRYPTO´85), LNCS 218, Aug 1986, pp. 417-426.

[NIST99] National Institute for Standards and Technology, "*Recommended Elliptic Curves for Federal Government Use*", Jul 1999, 43p.

[PAA10] Paar, C.; Pelzl, J. "*Understanding Cryptography: A Textbook for Students and Practitioners*". Springer, 2010, 372 p.

[PIG12] Pigatto, D. F. "*Segurança em sistemas embarcados críticos - utilização de criptografia para comunicação segura*". Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, ICMC-USP, São Carlos, 2012, 84 p.