UNIVERSITE BLIDA1

Faculté des Sciences
Département d'Informatique

THESE DE DOCTORAT

Systèmes Informatiques

Architectures Logicielles/Matérielles embarquées pour le chiffrement/déchiffrement hybride ECC-AES

Par

Bellemou Ahmed Mohamed

Devant le jury composé de :

Abderrezak **GUESSOUM**	Professeur	U. Blida 1	Président
Mohamed **OULD KHAOUA**	Professeur	U. Blida 1	Examinateur
Djamel **BENNOUAR**	Professeur	U. Bouira	Examinateur
Nadjia **BENBLIDIA**	Professeur	U. Blida 1	Directrice de thèse
Mohamed **ANANE**	MCB	ESI	Co Directeur de thèse
Yacine **CHALLAL**	Professeur	ESI	Invité

Blida, 2021

BLIDA1 University

SCIENCES FACULTY
Department of Computer Science

DOCTORAL THESIS

Computer systems

Embedded Software/Hardware architectures for hybrid encryption/decryption based on ECC-AES

By

Bellemou Ahmed Mohamed

Doctoral Committee:

Abderrezak **GUESSOUM**	Professor	U. Blida 1	Chair
Mohamed **OULD KHAOUA**	Professor	U. Blida 1	Examinator
Djamel **BENNOUAR**	Professor	U. Bouira	Examinator
Nadjia **BENBLIDIA**	Professor	U. Blida 1	Thesis supervisor
Mohamed **ANANE**	MCB	ESI	Co Thesis supervisor
Yacine **CHALLAL**	Professor	ESI	Guest

Blida, 2021

ABSTRACT

Security management for embedded systems is a critical research field, especially when taking into account the performance variation over different embedded devices. In this thesis, we present efficient SW/HW architectures of hybrid ECC-AES encryption/decryption for FPGA-based embedded cryptosystem. Indeed, the main aim is to achieve the best tradeoff between flexibility, security level, timing execution and area consumption.

In the first contribution, we present MicroBlaze-based parallel architectures of Elliptic Curve Scalar Multiplication (ECSM) computation for Elliptic Curve Cryptosystems (ECC) on Xilinx Virtex-5 FPGA. ECSM is performed by Montgomery Power Ladder (MPL) algorithm in Chudnovsky projective system, which allows the parallelism exploitation with several degrees. At low abstraction level, the critical operation Montgomery Modular Multiplication (MMM) is implemented within a hardware Accelerator MMM (AccMMM) core based on the modified high radix r($r=2^{32}$) MMM algorithm. The efficiency of our parallel implementations is achieved by combining both mixed SW/HW approach and Multi Processor System on Programmable Chip (MPSoPC) design. The Virtex-5 parallel implementations of 256-bit and 521-bit ECSM computations run at 100MHZ frequency to perform single ECSM are between 204.5 ms and 14.72 ms. The proposed architectures consume between 2739 and 6533 slices, 22 and 72 RAMs and between 16 and 48 DSP48E cores.

The second contribution consists of the proposition of high-performance client/server coordinators on low-cost SoC-FPGA devices for secure IoT data collection. Security is ensured by using the Transport Layer Security (TLS) protocol based on ECC/AES schemes. The hardware architecture of the proposed coordinators is based on SW/HW co-design approach, implementing ECSM within hardware accelerator core. Meanwhile, the control of the overall TLS handshake is performed by ARM Cortex-A9 microprocessor. The integration of ARM processor enables to exploit embedded Linux features for high system flexibility. The proposed ECC accelerator requires limited area, with only 3395 LUTs on the Zynq device and performs high-speed 233-bit ECSM in 413 µs, with a 50 MHz clock. The generation of a 384-bit TLS handshake secret key between client and server coordinators requires 67.5 ms on a low cost Zynq 7Z007S device.

RESUME

La gestion de la sécurité pour les systèmes embarqués est un domaine de recherche critique avec la variation des performances des différents dispositifs embarqués. Dans cette thèse, nous présentons des architectures logicielles/matérielles embarquées sur circuit FPGA pour le chiffrement/déchiffrement hybride ECC-AES. Notre objectif est d'obtenir le meilleur compromis entre flexibilité, niveau de sécurité, temps d'exécution et surface occupée.

Dans la première contribution, nous présentons des architectures parallèles de type MPSoC sur circuit FPGA de Xilinx basées sur l'intégration de plusieurs processeurs embarqués MicroBlaze pour le chiffrement asymétrique ECC. L'exécution de l'opération de base de l'ECC, en l'occurrence, la multiplication scalaire est basée sur la combinaison de l'algorithme Montgomery Power Ladder (MPL) et le système de représentation de points projectif de Chudnovesky. Cette combinaison permet l'exploitation du parallélisme à plusieurs degrés. Au niveau bas d'abstraction, la Multiplication Modulaire de Montgomery (MMM) est considérée comme l'opération critique. Elle est implémentée en matérielle (AccMMM) autour d'un ou plusieurs processeurs Microblaze en se basant sur l'exécution de l'algorithme MMM dans une base élevée, (r=2^{32}). Les implémentations proposées consomment entre 2739 et 6533 slices, entre 22 et 72 RAMs et entre 16 et 48 DSP48E. Nos implémentations opèrent avec une fréquence de 100 MHZ et exécute la multiplication scalaire avec des délais qui vont de 204 ms à 14,72 ms.

La deuxième contribution consiste en la proposition d'une implémentation efficace du protocole TLSv1.2 à base du processeur ARM sur des circuits FPGA récent de type Zynq à faible coût, dédiée aux applications IoT. Parmi les suites de chiffrement prises en charge par le protocole TLS, nous avons sélectionné la suite ECC_AES_HMAC pour la génération des clés secrètes de 384 bits. L'idée principale est d'implémenter la multiplication scalaire dans un accélérateur matérielle évolutif autour d'un microprocesseur ARM Cortex-A9, tandis que le contrôle de toutes les négociations de prise de contact TLSv1.2 est assuré par le processeur. Ce dernier s'exécute dans un système d'exploitation Linux embarqué avec une fréquence de 50MHZ pour effectuer une négociation TLS complète en 67,5 ms. L'accélérateur ECC proposé ne nécessite que 3395 LUTs tandis que l'architecture proposée occupe 8503 LUT.

ملخص

تعد إدارة أمن الأنظمة المضمنة مجال بحث مهمًا مع تفاوت أداء الأجهزة المدمجة .الهدف من هذه الأطروحة، يتمثل في زرع نظام تشفير هجين ECC/AES في شريحة مكونه من مبرمج FPGA. الهدف الرئيسي هو تحقيق مفاضلة بين المرونة ومستوى الأمان و توقيت التنفيذ واستهلاك المنطقة. أعمال البحث التي أنجزت في إطار هذه الأطروحة تستند على مساهمتين رئيسيتين :

المساهمة الأولى تتمثل في زرع العملية الأساسية لنظام ECC و المتمثلة في ضرب عدد في نقطه (ECSM) على FPGA و ذلك بإستعمال معالجات للمعلومات MicroBlaze ل Xilinx من أجل التنفيذ المتوازي للعملية المستهدفة. لتحسين زمان حساب هاته العملية، استعملنا كتلة IP بجوار المعالجات. دور IP هو حساب عملية الضرب الترديدي. نتائج التشفير بمفتاح ذو مقاس 256-bits و 521-bits، بينت ان زمن حساب ECSM تتراوح بين 115 ms و 14.72 ms و تحوز على مساحة داخل FPGA تتراوح بين 6533 و 2739 slices.

المساهمة الثانية تتمثل في اقتراح منسق عالي الأداء على أجهزة FPGA منخفضة التكلفة لتجميع بيانات IoT بشكل آمن. يتم ضمان الأمان باستخدام بروتوكول TLS وذلك بإستعمال شريحة Zynq تحتوي على معالج للمعلومات ARM CortexA9. لتحسين زمان مصافحة TLS ، استعملنا كتلة IP بجوار المعالج دورها تشفير ECC. يتطلب إنشاء مفتاح سر مصافحة TLS بمقاس 384-bits ب 67.5ms على جهاز Zynq-7Z007S منخفض التكلفة. المساحة داخل FPGA مقدرة ب 8503 LUTs.

ACKNOWLEDGEMENTS

Foremost, I would like to thank God Almighty for giving me the strength, ability and opportunity to undertake this research study and to complete it satisfactorily.

The research work for this thesis was carried out in the LRDSI Laboratory, Department of computer science, Blida 1 university. First, I would like to express my sincere gratitude to my supervisor Prof. Benblidia Nadjia, who shared a lot of her experience with me. I appreciate her patience, motivation, enthusiasm, immense knowledge and the freedom she gave me in pursuing various ideas.

Besides, I wish to express my deepest gratitude to my co-supervisor Dr Anane Mohamed for his invaluable guidance, proposed ideas, valuable discussions and constructive suggestions without which it could not have been possible to succeed in my research. His vision and motivation have deeply inspired me to carry out the research and to present the thesis as clearly as possible. It was great privilege and honor to work under his guidance.

I would like to thank the Prof. Guessoum Abderrezak, Prof. Ould Khaoua Mohamed, Prof. Bennouar Djamel and Prof. Challal Yacine for their time spent on reading my thesis and their valuable comments.

I am grateful to Dr. Issad Mohamed for his friendship, encouragement, advice, and help whenever needed. I am indebted to the AC2 team members and the Center for Development of Advanced Technologies for offering me a fabulous environment to work.

The Ministry of Higher Education and Scientific Research offered me an opportunity to visit the University of Granada, Spain, for one-year scholarship. It is hard to exaggerate the importance of the visit for my thesis work, and I am therefore most grateful for this wonderful opportunity that I had. Especially, I would like to thank Prof. Luis Parrilla and Prof. Antonio García with whom I had the pleasure of working during my stay in Granada.

I would like to thank my parents, my wife, and my sister for the love and encouragement I received. Without their heartfelt support this thesis would not have been possible. I would like to dedicate this thesis to them.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

IoT	Internet of Things
LAN	Local Area Networks
WAN	Wide Area Networks
SSL	Secure Sockets Layer
TLS	Transport Layer Security
IoTS	Internet of Things Server coordinator
IoTC	Internet of Things Client coordinator
FPGA	Field Programmable Gate Array
ASIC	Specific Integrated Circuits
GPP	General Purpose Processors
MPSoC	Multi-Processor System on Programmable Circuits
AES	Advanced Encryption Standard
DES	Data Encryption Standard
PKC	Public-Key Cryptosystems
NIST	National Institute of Standards and Technology
RSA	Rivest, Shamir and Adleman
ECC	Elliptic Curve Cryptosystems
DHKE	Diffie-Hellman Key Exchange
DSA	Digital Signature Algorithm
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
ECDSA	Elliptic Curve Digital Signature Algorithm
HMAC	Keyed-hash message authentication
ECSM	Elliptic Curve Scalar Multiplication
ECPA	Elliptic Curve Point Addition
ECPD	Elliptic Curve Point Doubling
MM	Montgomery Power Ladder
MA	Modular Multiplication
MI	Modular Inversion
Mexp	Modular Exponentiation
MMM	Montgomery Modular Multiplication
AccMMM	Montgomery Modular Multiplication accelerator core
NSP	Network Security Processors
DSP	Digital Signal Processor
CLB	Configurable Logic Blocks
LUT	Look-Up-Table

CONTENTS

INTRODUCTION

Thesis motivation

The increase deployment of large embedded and smart devices in our daily life, be it in fields such as internet communication, e-commerce, internet banking, storage and retrieval of sensitive data from cloud servers, mobile commerce, wireless sensors networks and many others, open the subject of intensive research for the information security concerns. In fact, large amounts of information are transferred through heterogeneous networks, ranging from Local Area Networks (LAN) to Wide Area Networks (WAN). Fortunately, cryptography [1, 2] plays a vital role to guaranty only authenticated access to information. It provides network secure information transfer over insecure channels by combining heterogeneous cryptographic protocols like public-key protocols (e.g., RSA, ECDH, ECDSA) [3-7], private-key schemes (e.g., 3DES, AES) [8, 9] and secure hash functions (e.g., SHA-1, SHA-2, SHA-3) [10-12]. Moreover, in the case of end-to-end network secure systems that require transferring data through internet, Secure Sockets Layer (SSL) [13] or Transport Layer Security (TLS) [14] are the preferred hybrid cryptographic solutions.

Several interesting alternative platforms for embedded cryptosystems such as Field Programmable Gate Array (FPGA), Application-Specific Integrated Circuits (ASIC) and Multicore microcontroller platforms are reported in the literature. The selection between these platforms depends on the targeted applications as well as the considered performances. Microcontrollers are the best platforms when high-efficiency in terms of power/energy consumption and execution time is promoted since these are made in ASIC. With the integration of panoply of components on a single programmable chip such as processor cores and programmable logic devices, FPGA provides lesser design time, re-configurability and high-performance to implement Programmable Systems on Chip (PSoC). It allows exploring new architectures and flexible designs. Therefore, FPGAs are considered as suitable solutions for applications that promote the compromise between performance, flexibility and re-configurability. Hence, we decided to go forward for FPGA platform.

Various FPGA-based cryptosystem implementations have been proposed in the literature [15]. The key issues of such works are defined by the security level, execution time, occupied area and flexibility. The flexibility constraint provides the possibility of an easier design modification due to system update functionality, while leaving the hardware architecture fixed. Another major factor that should be considered when designing a cryptosystem is the security of the hardware platform. The security analysis shows that the embedded cryptosystems can leak sensitive information during the execution of a computation, no matter how cryptographic patterns are mathematically hard and proven secure. Hence, efficient implementation of embedded cryptosystems needs the study of the vulnerability from various attacks on hardware platforms such as side channel attack [3].

Depending on the considered goals, three categories of implementation approaches are reported: Hardware implementation (HW), Software implementation (SW) and Co-design implementation (SW/HW). The first approach is used for high speed execution. The second approach is the best option when flexibility is strongly desired. It is based generally on embedded processors. The third approach offers the trade-off between flexibility, area and speed. Furthermore, some researchers propose to exploit Multi-Processor System on Programmable Chips (MPSoPC) approach [16, 17] for parallel execution in order to enhance the timing execution. In fact, MPSoPC is suitable solution for parallel implementations of applications that require intensive and critical computations. The performance of the overall system depends on the tasks partitioning of the targeted operation between the integrated processors.

Thesis aim and contribution

The main objective of this thesis is to propose novel architectures for efficient implementation of hybrid ECC-AES embedded cryptosystems on FPGA circuits. The major aim is to achieve the best trade-off between flexibility, security, area consumption and timing execution. Thus, the contributions of this thesis are mainly comprised of:

1. The design of new parallel implementations of efficient Elliptic Curve Cryptosystems (ECC) for embedded applications.

2. The implementation of TLSv1.2 protocol for efficient Client/Server IoT applications.

Efficient implementations of Elliptic Curve Scalar Multiplication

Public-Key Cryptosystems (PKC) are computationally intensive, due to the complex operations required by its algorithms, and may not be of generalized use for embedded devices due to hardware limitations. The implementation of PKC as embedded cryptosystems should guaranty high security and throughput with minimum hardware resources utilization. Therefore, suitable PKC associated protocols are essential to design efficient embedded cryptosystems. In the mid-1980s, Miller and Koblitz found the ability to use Elliptic Curves (EC) in PKC by building a cryptosystem based on Elliptic Curve Discrete Logarithm Problem (ECDLP) [1]. ECC provides better security with smaller key size compared to Rivest, Shamir and Adleman (RSA) cryptosystem [18]. This property changed ECC to a preferable choice for embedded cryptosystems due to the limit of resources and strict power requirements. ECC schemes have shown that their complexities are reduced to the computations of Elliptic Curve Scalar Multiplication (ECSM) [3]. This latter is considered as the most expensive operation in ECC protocols.

Efficient implementation of ECSM has received recently special attention. Good surveys in this area are described in Refs [19-22]. For fast prototyping, the purely-SW [21, 23] implementation is suitable approach by offering lowest cost and high degree of flexibility. This approach can effectively adapt to the modifications and updates in ECC standards which are related to the EC forms, the finite field and the security-level size. However, these implementations are usually impractical for applications that require fast-timing responsiveness and processing. Under these restrictions, HW and SW/HW implementations are the more suited approaches [24-30]. The high-speed could be achieved by using dedicated hardware as coprocessor or accelerator blocks to perform the hall ECSM computation or certain finite field operations. These approaches provide high-performance and offers high reliability. However, they come with higher cost and area consumption due to the required additional hardware blocks.

In this work, we present parallel architectures of ECSM computation for embedded ECC on FPGA circuit. ECSM is performed using Montgomery Power Ladder (MPL) algorithm [31] in projective coordinates systems. MPL algorithm is often suggested to withstand side channel attacks. The projective system allows the performance enhancement of ECSM execution not only by avoiding the execution of the complex finite field inversion operation but also by exploring the inherent parallelism within ECSM computations. In order to achieve the best trade-off between flexibility, timing execution and area occupation, the main idea is to employ SW/HW approach in MPSoPC. The performance of the overall system depends on SW/HW partitioning as well as the tasks partitioning of the targeted operation between the integrated processors.

Our strategy in this work is based on two folds. The first is the analysis of the operations within ECSM computations that could be executed in parallel. The second consists of the design and the implementation of the critical finite field operation which is Modular Multiplication (MM) within a scalable HW accelerator core based on Montgomery Modular Multiplication (MMM) algorithm [32, 33]. Thus, five SW/HW ECSM implementations based on MicroBlaze processor [34] are proposed. MicroBlaze processor is reconfigurable soft-core processor which could be integrated independently of the FPGA circuit family. Then all proposed designs can be implemented as MPSoC in all FPGA circuit families [35].

The objective behind the first fold is to reduce the critical path of ECSM execution. A lot of implementations show that the parallelization within ECSM computation provides a significant speed up of the ECSM performance [24-26, 29, 30]. Our study shows that the parallelism within the ECSM computation process could be explored in several degrees of parallelization. This parameter corresponds to the number of the combined Micoblaze processors in a single architecture. Our challenge is to investigate the optimum number of processors that allows achieving the best trade-off between area and speed. As well as the workload spreading of the targeted operation over the integrated processors, with keeping the difference in idle time of each MicroBlaze as low as possible. As a result, we proposed to exploit the parallelism with two, three, four and six degrees. Thus, our first contribution is the proposed tasks partitioning of ECSM process on the integrated MicroBlaze processors.

The second contribution is the implementation of MMM algorithm within a scalable HW accelerator core (AccMMM). The objective behind this fold is the design of dedicated HW accelerator, where its data path is independent of the input data length. The adopted SW/HW co-design leads to enhance the overall cryptosystem performance. To adapt the execution of MMM to the data bus of the integrated MicroBlaze processors, we proposed to use the modified high radix $r(r=2^{32})$ MMM algorithm [36]. In fact, when long modulus is considered, the original MMM algorithm requires not only a long carry propagation paths and multipliers but also long registers to store the operands and the intermediate results. Therefore, we exploit SelectRAM blocks in the internal architecture of our AccMMM core to store the data instead of long registers. Moreover, Xilinx's DSP48E [37] cores are exploited to accelerate the execution of the required 32×32-bits multiplications. The scalability of AccMMM core is represented by the independency of its hardware architecture with the FPGA circuit as well as the security-level size.

Efficient implementation of TLS protocol

The growth in the penetration of the Internet of Things (IoT) [38] in our daily life such as smart homes, smart enterprises, smart hospitals or smart cities, increases the concerns about the security management for large number of interconnected IoT devices. In fact, IoT paradigm implies different agents, such as sensors, cameras, actuators or microchips, which collect and transfer information through the Internet. As it is difficult to regulate the performance of all IoT devices, the security management becomes much more difficult than for a single device [39]. Due to the low performance hardware resources of a large number of IoT agents [13], the targeted cryptographic algorithms are not suitable to be implemented on every IoT device [40]. Hence, we propose to design high-performance client/server coordinators on low-cost SoC-FPGA devices for secure IoT data collection. We focus on securing data transferred from/to IoT coordinators by means of the TLS protocol, since SSL is considered insecure [41]. Efficient implementations of SSL/TLS protocols as embedded cryptosystems can be problematic, since the target devices are usually very limited in terms of power, resources and timing. Several TLS/SSL embedded cryptosystem implementations have been proposed in the literature [42-47]. OpenSSL [48, 49] is the most deployed library for TLS/SSL applications through software implementations of basic cryptographic functions. For only-software TLS/SSL implementations [47, 50], servers can be overloaded with heavy cryptographic operations, which results in long response times. To alleviate this bottleneck, dedicated hardware coprocessors [42-47, 51] have been proposed, as Network Security Processors (NSP), as a solution to free these severs from cryptographic operations for flexible management. Nevertheless, although effective efforts have been made [40] for the acceleration of encryption methods, NSPs can provoke an overhead of hardware resources utilization [42, 46] to achieve high-performance, due to the required intensive computations within cryptographic algorithms. This constraint paves the way for a HW/SW co-design implementation approach to provide a trade-off between security, area and speed. This approach is based on implementing the computing-intensive cryptosystems in hardware [52, 53], while the control of TLS/SSL protocols is performed in software using microprocessors. In this context, FPGA devices are suitable platforms, as they provide reconfigurability, flexibility and high performance. This is of special interest for the new FPGA generations; such as Zynq from Xilinx or Stratix 10 SoC from Intel, which are equipped with advanced components in a single chip including ARM microprocessors [54], Advanced eXtensible Interface (AXI) [55] buses, embedded memory or DSPs, and completely match the System on Chip paradigm.

In this work, we present a carefully designed SW/HW implementation of the client/server TLSv1.2 protocol, which is implemented on low-cost FPGAs/SoCs suitable for IoT applications. The use of modern FPGA-based SoCs enables the achievement of an optimal trade-off between security, flexibility, area, and speed. The third contribution of this thesis is the proposed SW/HW partitioning for efficient TLSv1.2 negotiations. Among the supported TLS cipher-suites, we have selected Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) [5], Elliptic Curve Digital Signature Algorithm (ECDSA) [6], Advanced Encryption Standard (AES-128), Secure Keyed-hash message authentication (HMAC) [12] and Secure Hash Algorithm (SHA256) for our implementation. These algorithms are all combined to generate 384-bit TLS secret shared keys. Thus, the main idea is to implement ECSM within a scalable hardware coprocessor accelerator and to integrate it around an ARM microprocessor. Meanwhile, the control of ECDHE and ECDSA protocols, the execution of AES-128 algorithm, HMAC and SHA256 functions are ensured by the ARM microprocessor. The internal architecture of ECC accelerator is based on time-area optimized finite field units and the use of dual-port block RAMs as registers. In addition, the I/Os of this ECC accelerator are 32-bit wide, which allow an easier integration with 32-bit microprocessors (e.g., ARM, PowerPC and MicroBlaze) via 32-bit buses (e.g., AXI and PLB). As results, ECC accelerator provides low area requirements while maintaining high performance.

Thesis organization

The rest of this thesis is organized as follows:

Chapter 2 summarizes the state-of-the-art related to the modern cryptography. It investigates about the theoretical background and the most popular security protocols. It illustrates also the mathematical background of finite fields and ECC.

Chapter 3 starts by presenting the challenges of secure embedded systems. Then, discussing the selection criteria between the hardware embedded platforms. Finally, the chapter gives an overview of FPGA internal architecture, the design flow and discusses the FPGA-based MPSoC solutions.

Chapter 4 presents the description of the proposed MicroBlaze-based parallel architectures for ECSM computation on FPGA circuit. The basic operation of ECC and the considered algorithmic optimizations for ECSM execution are shown first. Then, the proposed parallelism analysis within ECSM computation is illustrated. Next, it describes the FPGA implementation of MMM algorithm and the proposed ECSM embedded systems. Finally, the experimental results are discussed and compared with some recent works.

Chapter 5 presents the description of the proposed Zynq-based implementation of TLS negotiations for IoT Client/Server applications. Thus, it begins by presenting TLSv1.2 handshake protocol and the considered ECC cryptosystems. Then, the description of the internal architecture of our ECC accelerator and the hall embedded system are presented. Finally, the performance evaluation on a Xilinx Zynq device and comparisons with other works in the literature are illustrated.

Chapter 6 is devoted for the overall conclusion of this thesis and the possible future research directions.

CHAPTER 1
CRYPTOGRAPHY FOR EMBEDDED SYSTEMS

1.1 Introduction

Cryptography [1, 2] is the science of mathematics used to secure information transmission by encrypting data. Encryption of data is to transform a plaintext to an unreadable form that can be transmitted over an insecure channel. Decryption of data is transforming back the resulting ciphertext to intelligible form. Data encryption and decryption processes always include a key which is only known by the communicating parties. Thus, only the persons who have the key could decrypt the message. However, cryptanalysis is the art of revealing messages hidden by means of cryptography. It means that incorrect application of cryptography can result in disclosure of information to third-party users. In fact, cryptography ensures not only the confidentiality of information but also it provides other services such as authentication, integrity and non-repudiation. Authentication allows the recipient to confirm the sender's identity. Integrity ensures that the message was not modified during the transmission. Non-repudiation prevents the sender to deny not sending the message.

Cryptography consists of two main categories, namely, symmetric systems and Public Key Crypto-systems (PKC). Each system has its strengths and weaknesses. The main advantage of PKC systems is that they do not require any secret key exchange between the users at the contrary of private-key cryptography that requires the secret key to be transmitted from the sender to the receiver, either manually or through some communication channel. PKC also offers techniques for digital signature which provide authentication and non-repudiation services. However, symmetric systems are more efficient and faster encryption than PKC since public-key algorithms involve complex mathematics and are relatively slower. Therefore, we focus in the chapter on PKC to understand their complexity.

This chapter presents the state of the art of the presented work in this thesis. First, basic concepts of modern cryptography are introduced. Then, the background and mathematical tools of Elliptic Curve cryptosystems as PKC are reported to understand the underlying of the thesis research field. Finally, design methodologies and implementation strategies for efficient ECC cryptosystem designs on FPGA circuits are discussed.

1.2 Symmetric cryptography

Symmetric or private-key cryptosystem [2] is a class of algorithms that use a single secret key (k) for encryption and decryption functions as it is shown in Figure 1.1.

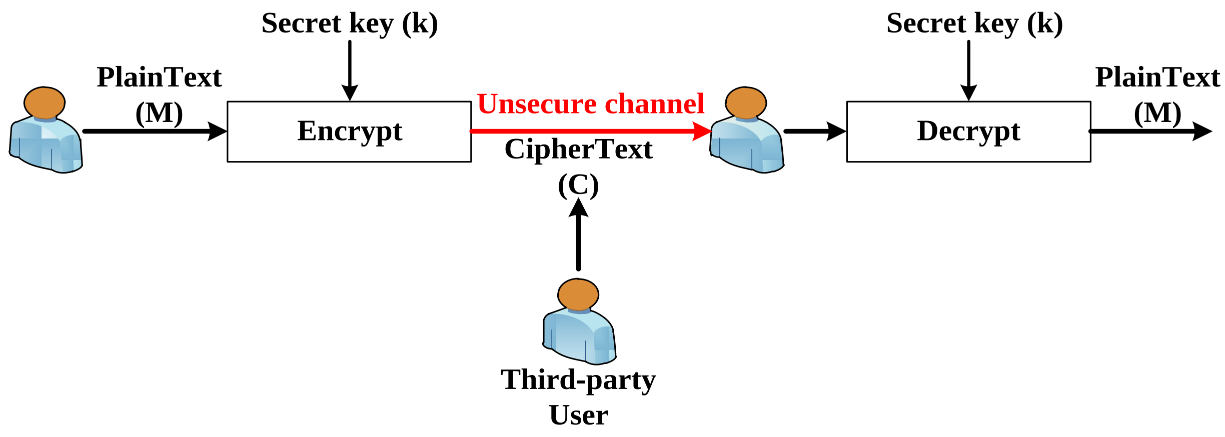


Figure 1. 1 Private-key Cryptosystem

This type of systems is used to provide confidentiality. It could be defined by the tuple (M, C, K, E_k, D_k), where:

- M represents the finite set of possible plaintexts.
- C represents the finite set of possible ciphertexts.
- K represents the finite set of possible keys.
- For each $k \in K$ there is an encryption rule E_k: M → C and a corresponding decryption rule D_k: C → M, where E_k and D_k are functions such that $D_k(E_k(m)) = m / m \in$ M

Symmetric cryptosystems are categorized into stream ciphers and block ciphers algorithms based on how data is manipulated. Stream cipher algorithms handle serially with very small chunks of input data, typically a byte, and produce the corresponding small chunks of output data. While block cipher algorithms operate on fixed-length blocks of input data and produce the corresponding output blocks. Practically, most block ciphers have a block length of 128 bits like Advanced Encryption Standard (AES) [9], or 64 bits like Data Encryption Standard (DES) and Triple DES (3DES) algorithms [8]. In the literature, AES is considered as the most popular and secure symmetric system used currently.

All Private-key Encryption/Decryption algorithms are characterized by its high-performance implementation as embedded cryptosystems due to a relative mathematic simplicity. However, this kind of systems suffers from key distribution and key management problems. The first problem is due to use of the same secret key that has to be securely shared among all the communicant parties. It means that there has to be an exchange of key information over secure channels for the first time. The second problem reveals because a communication in a group of n parties would require $n(n-1)/2$ keys.

1.3 Public-Key Cryptography

Asymmetric or Public-Key Cryptosystem (PKC) [3, 5, 6] is a class of algorithms in which encryption and decryption are performed using two different keys (k_e, k_d) as it is shown in Figure 1.2. The encryption key (k_e) is distributed in public while the decryption key (k_d) is kept private and secret.

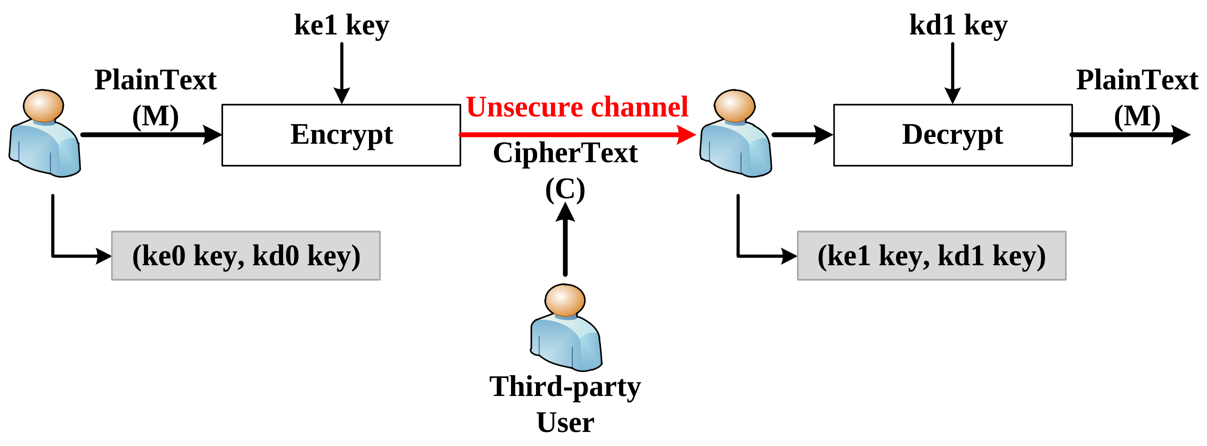


Figure 1. 2 Asymmetric Encryption/Decryption global scheme

This type of systems is introduced initially in 1970 by Whitfield Diffie and Martin Hellman in order to solve the key distribution problem of symmetric-key cryptography. Diffie-Hellman Key Exchange (DHKE) algorithm was the first PKC implementation [1]. It is mostly used to establish a secure communication channel for secret key exchange. They suggested generating public key K_e and private key K_d using hard computationally one-way mathematical problem like modular exponentiation. In such way, it is easy to derive K_e from K_d but it would be infeasible to find K_d from K_e. The private key is used to do symmetric encryption between the two systems.

In 1978, Ronald L. Rivest, Adi Shamir, and Leonard Adleman inspired from DHKE and published a new PKC in their paper “A method for obtaining digital signatures and public-key cryptosystems” [7]. RSA is the name of their proposal which is composed from the first letter of the inventor’s names. Since that time, RSA has been widely used in many applications and communication networks. It can be used for encryption as well as authentication. In RSA, the key pair is generated according to the security parameter which refers to the bit length of the keys. RSA-512, RSA-1024 and RSA-2048 refer to RSA algorithm with 512-bit, 1028-bit and 2048-bit key sizes, respectively.

After that, many interesting PKC protocols and schemes are constructed not only for secret key exchange and encryption/decryption but also for digital signature. In digital world, the property of proving the identity of the message sender is very important to withstand some attacks. The general idea is that the sender generates a digital signature for the message using his own private key. The receiver could validate the signature using the sender’s public key. In this way, the authenticity of the message could be checked. RSA and Digital Signature Algorithm (DSA) are widely used for a digital signature generation [7]. Figure 1.3 demonstrates the general idea of the digital signature process.

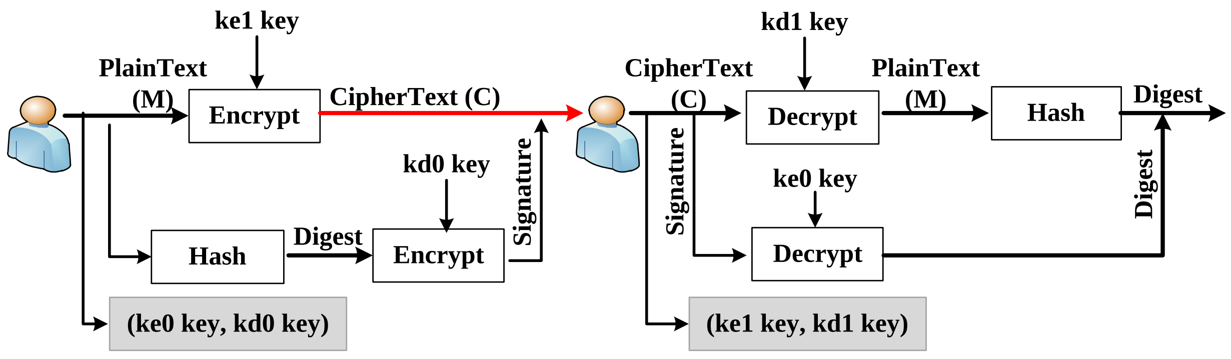


Figure 1. 3 Digital Signature process

In practice, the signature is not applied on the message itself but rather on its digest using hash functions. These functions generate a unique digest for a given message, which is short and a fixed-length string. In this way, the integrity, as well as the authenticity of the message could be validated. The hash functions are one-way functions that it is almost impossible to retrieve the input message from the output digest. SHA-1, SHA-2, and SHA-3 are popular algorithms of hash functions.

Although PKC provides high-security based on the hardness of DLP which is considered impossible to solve, several attacks have been proposed for breaking PKC [15]. These attacks target the mathematical system, weaknesses in PKC parameters and the hardware implementation platforms. In 1991, the RSA factoring challenge was launched by RSA Laboratories to motivate researches for successful factorizations of products of two primes [56]. The challenge was solved in 2005 for 640-bit length. Then, RSA-768 was factored in using the General NFS algorithm. Nowadays, the smallest key length recommended for RSA-keys is 1024 bits. The exponential increase of RSA key-sizes could be a bottleneck for embedded systems due to the limit of resources and strict power requirements.

In the mid-1980s, Miller and Koblitz found the ability to use Elliptic Curves in public-key Cryptography (ECC) by building a cryptosystem based on the Elliptic Curve Discrete Logarithm Problem (ECDLP) [3]. ECC provides better security with a smaller key size compared to RSA cryptosystem. This property changed ECC to a preferable choice for embedded cryptosystems. Table 1.1 presents a comparative analysis between RSA and ECC.

Table 1. 1 Recommended key-sizes for RSA and ECC [3]

Key sizes (bits)	
RSA	ECC
1024	160 – 223
2048	224 – 255
3072	256 – 383
7680	384 – 511
15360	521

Through this comparison, we observe that the minimum key size required for a secured cryptosystem of ECC is 160 bits. This latter has equivalent security level with 1024-bit RSA. Until today, the maximum ECC bit-length is 521 which is equivalent to 15360 RSA key-size. The advantage of ECC over RSA is obvious since with a shorter length for key it can provide the same level of security. When it comes to embedded systems, the use of small key-size presents high-performances with low-area consumption for PKC systems.

1.4 Elliptic Curve Cryptography

1.4.1 Elliptic Curve over a prime finite field

Finite field is the fundamental of cryptography, coding theory, and many other areas of mathematics and computer science [16]. Finite field also called a Galois field is a field that has a finite number of elements. The number of elements in the field is called the order of the field.

The order of a finite field is always the power of a prime p i.e. $q = p^m$, where m is any positive integer and q is the order of field. The prime p is called the characteristic of a field. If the order q of the field is equal to p then the field is called a prime field. However, if it is power of two ($q = 2^m$) then the field is named binary field.

In ECC, most curves are defined over prime fields (Fp) [4] and binary fields ($GF(2^m)$) [57]. The prime fields are more preferable for software implementations while binary fields are suitable for hardware implementations [53]. Hence, elliptic curves defined over prime field Fp are considered in this work.

Let Ep an EC defined over prime field Fp, where p is a large prime integer of w-bit. Ep(Fp) is a set of points on Ep represented by the affine coordinates (x,y), solving the simplified Weierstrass equation shown by expression (1.1).

$$E_p(F_p) = \{(x, y) : \ y^2 = x^3 + a\,x + b / \ (a, b) \ \in \ F_p\} \, U \ O \qquad (1.1)$$

Where a, b, x and y ∈ $[0, p-1]$ and $4a^3 + 27b^2 \neq 0$. O is the infinity point on Ep. The prime field Fp consists of all integers {0, 1, 2,....., p-1}, where arithmetic operations are performed on integers modulo p.

1.4.2 Elliptic Curve Scalar Multiplication

ECSM is the fundamental and the main operation of most ECC schemes. Is also known as point multiplication is given as follows:

$$Q = g \times P \qquad (1.2)$$

ECSM is defined as the multiplication of a scalar g by a point P defined on E. The scalar g is called the discrete logarithm of the point Q to the base P, denoted $g = log_P Q$. It is almost impossible to find the scalar g from the points P and Q. ECSM is computed by a series of EC Point Addition (ECPA) and EC Point Doubling (ECPD) operations as is shown by expression 1.3. ECPA is the addition of two distinct points P and Q represented by the affine coordinates $(x1, y1)$ and $(x2, y2)$, respectively. While ECPD is the addition of a point P with itself.

$$g \times P = P + P + \cdots + P \ (g \ times \ ECPA) \qquad (1.3)$$

The geometric interpretations of the ECPA operation and ECPD are shown in Figure (1.4.a) and (1.4.b), respectively. For ECPA, First, draw a line through P and Q; this line intersects the elliptic curve at a third point R'. Then, the reflection of this point about the x-axis is the resulting point $R(x3, y3)$ of ECPA. For ECPD, first, draw the tangent line to the elliptic curve at P. This line intersects the elliptic curve at a second point. Then R is the reflection of this point about the x-axis, which is the resulting point of ECPD.

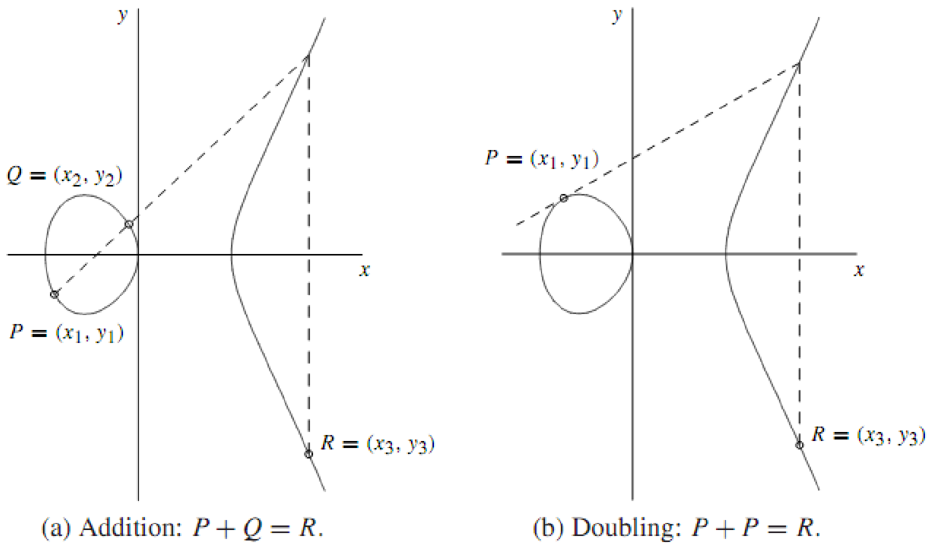


Figure 1. 4 Points Addition of Point Doubling [58]

The mathematical interpretation of ECPA and ECPD resulting point coordinates are given by the expressions 1.4 and 1.5, respectively.

$$R = P + Q = \begin{cases} x_3 = \lambda^2 - (x_1 + x_2) \\ y_3 = \lambda \times \left(x_{1-b} - x_{1-b(i)}\right) - y_{1-b(i)} \\ \lambda = (y_2 - y_2)\,/(x_2 - x_1) \end{cases} \tag{1.4}$$

$$R = 2 \times P = \begin{cases} x_3 = \lambda^2 - 2x_1 \\ y_3 = \lambda \times (x_1 - x_3) - y_1 \\ \lambda = (3x_1^2 + a)/2y_1 \end{cases} \tag{1.5}$$

1.4.3 Elliptic Curve Cryptosystems

The design of secure ECC schemes consists of three parts, namely, the curve parameters domain, the key generation, and the EC-based cryptographic protocols. In the following, each part is developed.

1.4.3.1 Elliptic Curve domain parameter

The domain parameter $D = (p, L, a, b, P, n, h)$ describes an elliptic curve defined over prime finite field Fp [3]. It is composed by:

- The finite field order p.
- The security level L which represents the bit-length of Fp elements.
- Two coefficients $a, b \in Fp$ that define the equation of the elliptic curve E over Fp.
- A point generator P defied with two field coordinates xP and yP in Fp.
- The order n of the curve E.
- The cofactor $h = \#E(Fq)/n$.

The security of ECC schemes is based on the hardness of ECDLP [18]. In fact, there is no mathematical proof that the ECDLP is unbreakable. The most naïve algorithm for solving the ECDLP is the exhaustive search whereby one computes the sequence of ECPA such 2P, 3P, 4P,... until Q is founded. However, several mathematical attacks are proposed in the literature such as: Pohlig-Hellman, Pollard’s rho, isomorphism algorithms, and prime-field-anomalous curves. These are the best general-purpose attacks known on the ECDLP [3]. The domain parameter D should be carefully chosen for maximum ECDLP resistance for all knows attacks such:

- Select the scalar g sufficiently large $g \geq 2^{80}$.
- Choose curve so that its order is sufficiently large ($n > 2^{160}$) or divisible by a large prime number.
- It is recommended that the cofactor $h \in \{1,2,3,4\}$.

In the literature, secure domain parameters generation and verification algorithms are proposed so that all ECDLP security constraints are considered. On the other hand, several standard curve domains over prime and binary fields are recommended with different key-length by many accredited standards bodies as National Institute for Standards and Technology (NIST) [59]. For prime field Fp, the recommended lengths of p are in the set {112, 128, 160, 192, 224, 256, 384, 521}.

1.4.3.2 Elliptic Curve Key generation

The EC-based key generation algorithm results in the (d,Q) key pair, where the private key d is an integer of L-bits and the public key Q is a point on E. The key pair (d,Q) is calculated as follows [3]:

1. Choose an integer d from [1, n − 1].
2. Calculate Q = d × G, where P is the generator point defined on E.

In the literature, several schemes are reported to apply the ECs for information security. They are often used for key agreement, encryption/decryption and digital signature. In the following, we present the ECDHE and ECDSA algorithms.

1.4.3.3 Elliptic Curve Diffie Hellman

ECDH stands for EC-based Diffie-Hellman key agreement protocol [5]. It ensures the establishment of a secret shared key between two parties A and B through an insecure channel. This key could be used by symmetric cryptosystem for data encryption. In the literature, two versions of ECDH are reported, namely, ECDH static and ECDH Ephemeral (ECDHE) [5]. The difference is that the first version always uses the same key pair, while the second generates new key pair for each connection. The shared secret key is obtained by applying the following steps:

1. A chooses integer n1 from [1, n − 1] and computes Q1 = n1 × G.
2. In parallel, B chooses integer n2 from [1, n − 1] and computes Q2 = n2 × G.
3. A and B exchange Q1 and Q2. Then compute Q = n1 × Q2 and Q = n2 × Q1, respectively.
4. A and B could extract the shared secret key from the coordinates of the shared point Q.

We note that ECDHE requires the execution of four ECSMs. However, the computation of Q1 and Q2 are performed in parallel, as well as the computation of $Q = n1 \times Q2$ and $Q = n2 \times Q1$.

1.4.3.4 Elliptic Curve Digital Signature Algorithm

ECDSA is an EC-based DSA algorithm proposed in 1992 by Scott Vanstone et al [6]. It is used for data integrity to avoid message tampering during transfer between server and client by signing the message. This protocol consists of two algorithms, namely signature generation and signature verification. In our work, the signature generation procedure is performed by the server to sign a message using his private key. Meanwhile, the verification is executed by the client to check if the received message is appropriate to the server or a third party using the public key of the client. Pseudocode descriptions of the two algorithms are presented in Algorithm 1.1 and Algorithm 1.2, respectively, while their detailed justification and description can be found in [60] . The resulting signature of the message m is represented by (r, s).

Algorithm 1. 1 Elliptic Curve Digital Signature Generation [3]

```
Inputs: private key $d$, message $m$, $D = (q, L, a, b, P, n, h)$
Outputs: Signature $(r, s)$
1. Generate random integer $k \in [1, n - 1]$
2. Compute $e$ = Hash($m$)
3. Compute $R = k \times G$
4. Set $r = x_R \bmod n$. If $r = 0$ return to step1
5. Compute $s = (k^{-1} \times (e + d \times r)) \bmod n$
6. The signature for $m$ is then $(r, s)$
```

The complexity of algorithm 1.1 is linked to the execution of: secure random generation of k, secure hash function for e computation and single ECSM for computing the coordinates of the point R, Modular Inversion, two Modular Multiplications and Modular Addition to obtain s.

Algorithm 1. 2 Elliptic Curve Digital Signature Verification [3]

```
Inputs: message $m$, signature $(r, s)$, $D = (q, L, a, b, P, n, h)$, public key $Q$
Outputs: accept or reject signature
1. verify $r, s \in [1, n - 1]$
2. compute $e$ = Hash($m$)
3. compute $v = s^{-1} \bmod n$
4. compute $u1 = e \times v \bmod n$
5. compute $u2 = r \times v \bmod n$
6. compute $X = u1 \times G + u2 \times Q$
7. if $X = O \rightarrow$ reject signature
8. else if $X_x \bmod n = r \rightarrow$ accept signature
```

According to Algorithm 1.2, the verification of the signature requires the execution of the secure hash function, the computation of a MI and two MMs for v, u1 and u2 calculations. To compute the coordinates of the point X, two ECSMs and a single ECPA are required.

1.5 Challenges of embedded cryptography

Designing efficient cryptographic implementations as embedded cryptosystems requires maintaining the following objectives [15]:

- High-Speed: An implementation should be fast enough to ensure that executing cryptographic algorithms does not slow down all the system significantly. Achieving high-speed is often a difficult task due to the intensive computations. Thus, algorithmic and architectural optimizations have attained a considerable amount of interest.

- Low-Resources: Many environments set tight constraints to resources available for the implementations of cryptographic algorithms. If resources are generally low, only a small portion of those few resources is devoted to cryptography which complicates implementing high-security cryptographic algorithms. The resources may include power consumption and available logics and memories, etc. The importance of this requirement is increasing because of the emerging use of cryptographic algorithms in various low-cost applications, such as smart cards or mobile hand-held devices.

- Security: The security analysis shows that the embedded cryptosystems can leak sensitive information during the execution of computation, no matter how cryptographic schemes are mathematically hard and proven secure. Hence, efficient implementation of embedded cryptosystems needs the study of the vulnerability from various attacks on hardware platforms such as side-channel attacks. These attacks are based on the interpretation of behavioral observation of the physical platforms such as power consumption and electromagnetic emanation.

- Flexibility: The flexibility of the embedded system is required, due to the rapid changes in cryptographic algorithms and standards. This constraint provides the possibility of an easier system functionality modification and updating, while leaving the hardware architecture fixed.

- Cost: The implementation of cryptographic algorithms for embedded systems should not affecting on the total cost of a product significantly as well as the performance of the application. Hence, the compromise between high-performance and low-cost embedded cryptosystem sets strict constraints for designers to provide efficient low-price consumer products.

1.6 Efficient embedded ECC implementation

As ECC is a multi-layer system, its overall performance could be significantly improved by optimization at different abstraction levels. The literature review shows that the design methodology for efficient implementation of ECC consists of two main procedural steps [19]. The first step in the process consists of selecting the most suitable ECC system parameters and algorithms. The second step represents the adoption of the best implementation approach that matches with the considered parameters and algorithms to respond to the aims of the targeted system.

1.6.1 ECC system parameters and algorithms

ESCM is the most time-consuming operation in ECC protocols. Thus, the performance enhancement of this operation will have a significant impact on the overall performance of ECC system. The optimization techniques for efficient ECSM execution are based on the choice of ECSM algorithms, coordinate systems, finite field arithmetic operation methods and parallelism exploration.

The scalar multiplication is performed by repeated ECPA and ECPD operations. The fundamental optimization technique for the performance improvement of ECSM execution is by reducing the number of iterations of ECPA and ECPD. Hence, various algorithms for point multiplication ($g\times P$) computation are proposed in the literature [61]. The most popular point multiplication methods are Double-and-Add binary algorithm, Non-Adjacent Form (NAF) algorithm, Windowing methods and Montgomery Power Ladder binary algorithm. The Double-and-Add binary method with its two variants algorithms; left-to-right and right-to-left; is the simplest for ECSM computation. It represents the scalar g in binary format and always performs ECPD operation, whereas the ECPA operation is executed only for non-zero scalar bit value g_i. The security analysis shows that the implementation of this method on embedded systems suffers from various side channel attacks in the context where the scalar g is secret [25]. The value of the bits of the scalar g could be easily revealed by tracing the timing and power consumption of the device. Montgomery Power Ladder algorithm (MPL) is reported in the literature as one of the binary methods that ensure the resistance against these kinds of attacks [28, 29] by performing ECPD and ECPA in parallel independently of the current scalar bit value g_i. This algorithm is considered in our proposed implementations.

While designing high-speed ECC architecture, one of the main concerns is the choice of appropriate point coordinates representation system. ECPA and ECPD could be performed using affine and projective systems. In affine system, both ECPA and ECPD operations involve modular inversion which is the most expensive operation in terms of execution time and hardware resource requirements [24]. Usually, modular inversion is performed by hardware dedicated inverters based on the extended Euclidean algorithm and Fermat's little theorem [62]. However, this approach could be an inefficient choice for low area implementations. The best solution is to free ECPA and ECPD operations from modular inversion at the cost of an increasing number of modular multiplications by using the projective system [63, 64]. In fact, most efficient ECC designs promote the use of a projective system since the complexity of a high-speed modular inverter is much higher than a fast multiplier.

Although powerful computational resources are available on FPGA, the implementation of the critical finite field Modular Multiplication operation required in projective system still costly and complex for hardware implementation. The complexity of this operation comes from the need to perform reduction over large prime modulo [19]. In fact, an optimized modular multiplier leads to improve the overall performance of ECC. Thus, different reduction techniques are reported. One of these techniques is by choosing the order p of the curve of a special structure called pseudo-Mersenne primes [27]. This kind of structure could provide higher speed reduction. Therefore, NIST recommended five specialized curves (p_{192}, p_{224}, p_{256}, p_{384}, p_{512}) for different levels of security [59] based on pseudo-Mersenne primes. Consequently, a large number of high-performance hardware modular multipliers for ECC over NIST curves are proposed in the literature [27, 29, 30]. The major lack in this approach is that these implementations are not able to provide the flexibility for performing ECC over general primes. However, the high-performance reduction over general primes could be achieved by introducing low complexity modular multiplication algorithms. Barrett and Montgomery [65] modular reductions are the most methods used for high performance modular multipliers. The two methods are simple and suitable for hardware implementations. Montgomery Modular Multiplication (MMM) [32, 66], which is considered in this thesis, improves the reduction over general prime modulo by replacing the required division operation with cheaper shift and addition operations.

To obtain maximum performance, parallelization is an optimization technique used to speed up the ECSM execution by reducing the critical path delay. This technique requires investigation about operations independence within ECSM computations for parallel execution. The parallelization possibilities for the computation of ECSM depends on the adopted ECC parameters and algorithms. It could be exploited in ECSM algorithm, between the required finite field arithmetic operations and within modular multiplication algorithms. This approach requires the integration of multiple processing elements such as embedded processors and hardware accelerators. Thus, the second challenge is to investigate also about the optimum number of processing elements that allows improving the execution time and throughput of the ECC system, while keeping an eye on the resource requirements.

The adopted ECC system parameters and algorithms will have an impact on the selection of the implementation approach design. In the following, some of the related works in the field are reported to understand the efficient ECC design issues, the adopted optimization techniques and the metrics used to evaluate the efficiency of ECC designed systems.

1.6.2 ECC implementation approaches

FPGAs have been a popular hardware embedded platform for efficient implementation of cryptographic systems. Various high-performance ECC systems on FPGA circuits are proposed in the literature [19-30]. The general architecture of an ECC processor is governed by the considered ECC parameters and algorithms as well as by the targeted circuit and the objectives of the targeted application. These objectives include security, flexibility, computation time, resource requirements, power consumption, throughput, etc. Depending on the considered goals, ECC designs could be classified into three main categories: Software ECC cryptosystem, Hardware ECC cryptosystem and Hardware/Software co-design ECC cryptosystem. Each approach is proposed for a particular use, having its advantages and limitations.

The implementation of ECC computations completely in software using an embedded processor offers the lowest cost and a high degree of flexibility [21, 23]. This approach is suitable for fast prototyping because it is easy to adapt to the changes in ECC parameters and algorithms. However, it provides usually low computation time and high-power consumption solutions since embedded processors are not specialized to perform finite field arithmetic operations over large prime numbers. To execute operations on operands of large sizes, the processor will have to decompose the operands until they are sufficiently small so that the processor’s ALU can operate on them. This technique increases the number of clock cycles required to process the data, which thus increases the overall computation time.

A high-speed constraint is extremely required for real-time embedded information security systems that need fast responsiveness and processing. The implementation of the entire ECC scheme on dedicated hardware cores is a technique that provides extraordinary high-speed performance and low power consumption results [24-30]. These performances are achieved by increasing significantly the hardware resources which is not suitable for resource-constrained small FPGA devices. Moreover, these hardware processors lack the flexibility of software. Therefore, the design of area-time optimized flexible ECC hardware crypto processors is a challenging research field.

FPGA offers the possibility to implement both embedded processors and hardware accelerator blocks in the same chip in order to balance between the speed performance and the flexibility. Hardware/Software ECC implementation approach [24-30] benefits from this option by integrating dedicated hardware acceleration blocks as coprocessors around embedded processors such that the hardware module is controlled by the host processor. The hardware accelerators are used to increase the performance of a specific computations in ECC system. The main idea is to free the embedded processor from the intensive computations in ECC. In such a way, the crypto-design can achieve high performance but with an extra area requirement due to the need for additional hardware resources. The challenge of this approach is to investigate the best partitioning between the software and the hardware to provide the best performance trade-off in terms of timing computation and area consumption.

1.6.3 Related works

This section reviews the literature of available efficient ECC cryptosystems. It outlines some related works in the field to present the state-of-the art of design methodology and the techniques for efficient ECSM implementation.

K.C.C. Loi and S. Ko [13] present the implementation of a high-performance scalable elliptic curve cryptography processor (ECP) that supports all five NIST prime field curves. The block diagram of the proposed ECP consists of two parallel prime field arithmetic blocks. One computes integer multiplication based on Comba algorithm, and the other computes addition/subtraction/reduction. The point multiplication was performed using the double-and-add algorithm and projective system. The proposed ECP is also able to implement the prime field inversion algorithm efficiently using the same arithmetic units. The main contribution of this work was the novel hardware architecture of the finite field arithmetic units that take advantage of the DSP48E available on Xilinx FPGAs to improve the efficiency of the ECSM operation. The proposed design has been implemented on Xilinx Virtex-5 and Virtex-4 FPGA circuits. The performance evaluation of the proposed ECP and the comparison with some works was done using the efficiency metric (1/ latency × number of slices). Thus, the authors believe that their processor is the fastest and smallest ECP that supports all prime fields, being able to compute ECSM between 1,709 and 28,04 ms using only 7 DSP and 1980 slices on a Virtex5 XC5LX101T FPGA. It completes single ECSM computation between 3,361 and 38,73 ms while only 8 DSP and 7020 slices on a Virtex4 XC4VFX100 FPGA. However, their ECP is not generic but specific to only NIST curves.

B. Baldwin et al. [26] reports a survey of various ECSM algorithms based on efficient co-Z arithmetic for general prime curves. These algorithms were implemented in HW, SW and SW/HW architectures for three field sizes, namely, 192, 256 and 521 bits. The main aim of this work was the performance evaluation and comparison of the studied algorithms on FPGA circuit in terms of security, speed, memory, power and energy consumption. The point multiplication was performed using Montgomery ladder and Joye’s double-add algorithms, while, point addition and point doubling were performed by various projective systems. First, all the considered algorithms were implemented completely in software using Xilinx MicroBlaze softcore embedded processor. This approach leads to a very slow computation time for ECSM since that Microblaze is not optimized to perform large finite field multiplication. For 521-bit case, ECSM computation times are as long as 22 seconds. Therefore, the authors propose to implement a dedicated Montgomery multiplier around Microblaze through a Fast Simplex Link (FSL) to increase performance on any ECC algorithm that uses 192-bit, 256-bit or 521-bit Montgomery multiplications. The multiplier runs at 100 MHz in the case of the 192-bit implementation and 75 MHz for both the 256-bit and 521-bit implementations. The authors concluded that the hardware multiplier reduces the computation time by on average 89–94 % for the algorithms by using 334-904 extra area slices required for the multipliers. This type of design is suitable when flexibility and low area are more promoted than high-speed performance. To obtain high-speed, the authors propose to implement ECSM algorithms in a dedicated hardware croptoprocessor. The proposed ECP consists of control circuitry, BlockRAM for storage of results and some parallel finite field arithmetic units, namely field multipliers, adders and subtractors. The adder/subtracter unit is generated using single clock carry propagate adders.

It performs an addition or subtraction in four clock cycles, comprising two clocks for the additions and two clocks for move operations from and to the RAM. The modular multiplier performs modular multiplication based on Montgomery multiplication algorithm. It is designed using carry propagate adders. The major contribution of the proposed ECP is the possibility to perform ECSM for any particular algorithm, with only minor changes in the control unit. Moreover, it can be configured to run one to four parallel multipliers depending on the operation dependency of the considered algorithms. Therefore, a list-based scheduling (LBS) technique was adopted to maintain the list of operations whose predecessors have already been scheduled. In this way, they can optimize the area-speed tradeoff and pick the best fit amount for each particular algorithm. The evaluation of the proposed ECP shows the speed difference is roughly a factor of ×450 compared to the software approach but with scarifying the area requirement constraint. The overall performance metrics used to differentiate the studied algorithms are: area-time (AT) and area-energy (AR) products. AT product was calculated to get a representation of any speed decrease against an increase in size, while, AR product was calculated to give a representation of the power increase against the increase in area requirements. Finally, the authors claim that each algorithm performs differently depending on how the targeted platform handles the modular multiplication and the parallelism scheduling. They concluded that no one of the studied algorithm performs best on every platform. To achieve the best performance from each algorithm, the platform must be optimized to suit that algorithm.

H.Marzouqi et al. [27, 30] propose a programmable and configurable application-specific-instruction-set ECC processor (ASIP) that supports the recommended NIST curve P_{256}. The proposed ASIP performs ECSM based on the left-to-right binary method in affine coordinates system. Thus, the processor consists of three Redundant Signed Digits (RSD)-based high-throughput arithmetic units: modular adder/subtracter, modular multiplier and modular inverter. RSD representation is a carry free arithmetic where integers are represented by the difference of two other integers. The divider is designed through a new efficient binary GCD architecture based on simple logical operations using the extended Euclidean algorithm. The multiplier is implemented based on pipelining Karatsuba–Ofman architecture. Karatsuba and Ofman method allows reducing the complexity of regular multiplication by dividing the operands into smaller and equal segments. The proposed processor was implemented in Xilinx Virtex-5 XC5VLX110T FPGA. It operates at 160 MHZ to perform a single point multiplication for P256 in 2.26 ms. Their processor requires 34612 LUTs. The authors believe that their processor is the first FPGA implementation of RSD-based ECC processor and that the proposed divider is the fastest to be performed on FPGA device. They claim that the exportability feature of the processor in different FPGA devices from different vendors, is ensured since none of the macros or embedded blocks within the FPGA fabric is utilized in the internal architecture.

1.7 Conclusion

In this chapter, the basic cryptographic systems are introduced first. The implementation of PKC as an embedded cryptosystem is more complex than symmetric systems. Therefore, we focused next on the efficient implementation of PKC schemes, especially ECC. Thus, we presented the mathematical tools required to understand the underlying ECC systems. It has been shown that the performance of ECC depends on the performance of the point multiplication, which is the fundamental operation is most ECC schemes. The performance of ECSM as embedded cryptosystem can be evaluated by flexibility, area, speed and power. Depending on the considered performance, various optimization techniques and implementation approaches are developed in the literature. To demonstrate the design methodology efficient ECC implementation, some related works are presented in this chapter. The first step in the design methodology process is the choice of the most suitable parameters and algorithms at different levels in ECSM hierarchy. This latter consists of three main layers that are ECSM algorithms, point coordinates system representation and finite field arithmetic operations. The adopted algorithms determine the implementation approach and the general architecture of the targeted ECC design. The literature review presents three implementation approaches. For low-cost and fast prototyping application, the implementation of ECSM completely in software using an embedded general-purpose processor is the preferable solution. However, it provides low computation time and high-power consumption. For some high-timing constrained applications, the implementation of ECSM in ECC hardware processor provides high-speed designs. In this case, the speed is achieved by using large area. FPGA offers the possibility to implement both embedded processors and hardware accelerator blocks in the same chip to balance between the speed performance and the flexibility. This option changed FPGA to most common suitable platform for embedded cryptography. The main aim of this thesis is to design an efficient ECC architecture that provides the best trade-off between software flexibility and hardware speed with keeping an eye on the area consumption. Thus, we decided to go forward for FPGA platform. In the next chapter, an overview of FPGA internal architecture, the design flow cycle and FPGA-based MPSoC solutions, are discussed.

CHAPTER 2
FPGA PLATFORMS FOR EMBEDDED SYSTEMS

2.1 Introduction

An embedded system is defined as a computing platform that performs a limited set of functions using a combination of hardware and software under real-time constraints [67]. Development of embedded systems and their employment in large applications has transformed the computing paradigm from high-performance desktop machines to small embedded devices. These devices are characterized by the limited resources in terms of processing capability, energy supply and limited area and memory. As mentioned in the previous chapter, PKC protocols are computationally intensive, and deploying them on resource-constrained embedded platforms without degrading their performance is a challenging task. In the last decades, efficient PKC implementations over reconfigurable FPGA devices have been explored to provide the flexibility of general-purpose processors and the high-performance of dedicated hardware.

In this chapter, we demonstrate the advantages of FPGA circuits and their suitability for cryptography as an embedded cryptosystem. Thus, we present first the main objectives of the most efficient cryptographic implementations as embedded cryptosystems. Then, a performance comparison between the most common hardware platforms for cryptography are illustrated. Next, the structure of FPGAs and FPGA-based design flow cycle are presented. Finally, the difficulties and the techniques of writing parallel programs on FPGA circuit are reported. This last section is required since we are targeting MPSoC approach on the first contribution of this thesis.

2.2 Embedded systems platforms

Several interesting alternatives for embedded cryptosystems such as General Purpose Processors (GPP), Application-Specific Integrated Circuits (ASIC) and Field Programmable Gate Array (FPGA) circuits are presented in the literature [15]. The selection between these hardware platforms depends on the targeted applications as well as the considered performances.

GPP is the best solution when high flexibility is promoted. It executes a program that contains a specific set of instructions in serial mode. This program is developed in software using high-level language such as C or assembly. This kind of implementations is referred as the Software implementation approach (SW). This approach offers the portability of the application on large GPP devices since the same code is usually easy to use in different processors. The performance of a program depends on programmer's skill, the instruction set provided by the processor, and the efficiency of the processor. Traditionally, the implementation of public-key algorithms presents low-performance and high-power consumption since that GPP are not designed to perform large finite field arithmetic operations.

As the name implies, ASICs are integrated circuits designed for specific applications. The logic function of ASICs is specified based on the Hardware implementation approach (HW) using hardware description languages such as Verilog or VHDL. Its digital circuitry is made up of permanently connected gates and flip-flops in silicon. Thus, the resulted design c-an be highly optimized in terms of speed, area, and power consumption. However, once the application is taped-out into silicon, it performs the same function all its operating life and could not be changed to anything else. Moreover, the development of ASIC designs is very expensive and large time consuming, thus, they are not recommended for designs prototyping when a small quantity of production is targeted. But in large production, the cost per volume becomes very less.

With the integration of a panoply of components on a single Programmable System on Chip (PSoC) such as multi-processor cores and programmable logic devices, FPGAs as suitable platforms for cryptographic modules, secure network routers, military devices and other systems requiring high security, software flexibility, hardware speed and re-configurability. FPGAs fill the gap and combine the advantages of GPP and ASICs by exploring new parallel architectures based on SW/HW co-design and Multi-Processor System-on-Chip (MPSoC) approaches. In other words, FPGAs are suitable for cryptography not only for their high level of parallelism and flexibility but also for the ability to accelerate the computation of any operation using dedicated embedded hardwired units. FPGA provides significant cost advantages compared to ASIC by offering the same level of performance in most cases. Another advantage of the FPGA is its ability to be dynamically reconfigured.

In our work, our main aim is to achieve the best trade-off between flexibility, security, area consumption and timing execution. Hence, we decided to go forward for FPGA platform.

2.3 Field Programmable Gate Array (FPGA) circuits

In 1984, Xilinx introduced the first FPGA, and then Actel popularized the term in 1988 [68]. In fact, FPGA innovation was the elimination of the AND-array that provided the programmability in the previous programmable Chips such as EPROM-programmed Programmable Array Logic (PAL) and Complex Programmable Logic Device (CPLD). Instead, configuration memory cells were distributed around the array to control functionality and wiring. Since that time, FPGA have progressed through several distinct phases of development following quantitative effects of Moore's Law. Figure 2.1 shows the Xilinx's published scaling process of the development of FPGA technology in terms of capacity, performance, cost and energy. FPGAs has grown in capacity by more than a factor of 10000 and in performance by a factor of 100. Cost and energy per operation have both decreased by more than a factor of 1000.

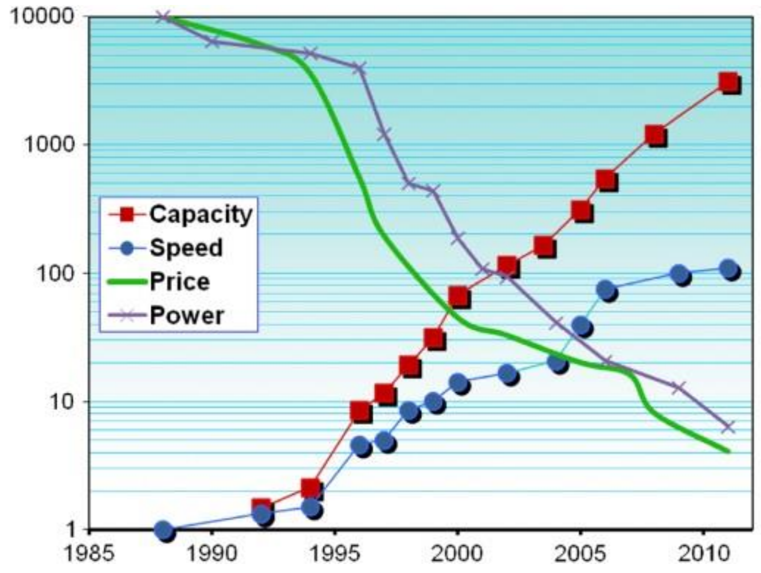


Figure 2. 1 Xilinx's published scaling process of the development of FPGA technology in terms of capacity, performance, cost and energy [68]

With the elimination of the AND-array, FPGA architects had the freedom to build any logic block and any interconnect pattern. The architecture of the FPGA consists of an array of programmable logic blocks interconnects with field-programmable switches. The performance of the FPGA depends on where the logic is placed and how it is routed in the FPGA. The FPGA vendors concluded that increasing the number of logics was insufficient to compete against ASIC technology. Therefore, they focused on FPGA efficiency and produced families of lower-capacity, lower-performance and low-cost FPGAs (Virtex and Spartan from Xilinx, Cyclone from Altera and EC/ECP from Lattice). Furthermore, they produced soft logic libraries (IP) for important functions such as softcore microprocessors (Xilinx MicroBlaze and Altera Nios), memory controllers and communications protocol stacks. Moreover, they proposed to add dedicated logic blocks within the FPGAs like memories, DSP multipliers, flexible I/O and source-synchronous transceivers. As result, FPGA was not simply a collection of Look-Up-Tables (LUTs), flip-flops, I/O and programmable routing but it included multipliers, RAM blocks, multiple hardware microprocessors, interconnect buses (PLB and AXI), clock management and gigahertz-rate source-synchronous transceivers. To ease the burden of designing FPGA architectures, tools have been provided also such as Embedded Design Kit (EDK) and Vivado by Xilinx and Embedded System Design Kit (ESDK) by Altera.

2.3.1 FPGA internal architecture

An FPGA is a type of programmable integrated Chips [69]. Modern FPGA devices consist of up to two million logic cells that can be configured to implement a variety of applications. Figure 2.2 illustrates the basic structure of an FPGA. It consists of three major components, namely, Configurable Logic Blocks (CLB), Programmable switches and Input Output Block (IOB) pads. CLB is the basic logic element that provides basic computation and storage elements used in digital systems. Programmable switches a establish connection between CLBs. physically available IOB ports get data in and out of the FPGA.

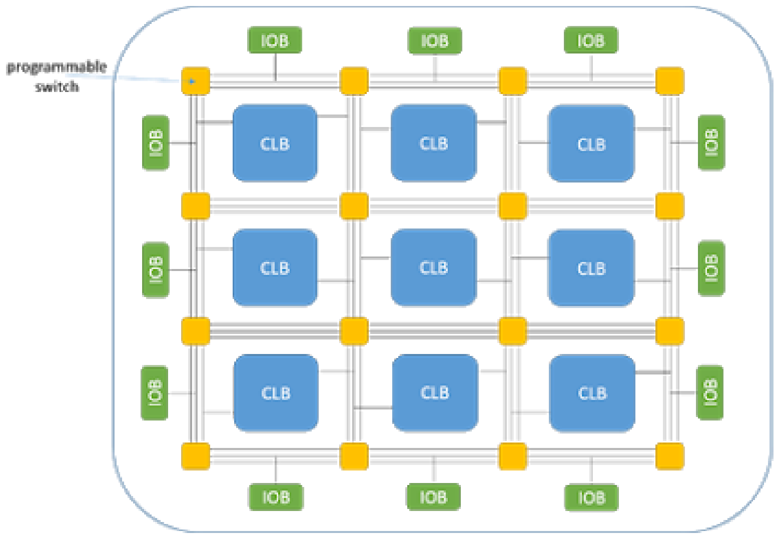


Figure 2. 2 Basic structure of FPGA

CLBs are the main logic resources for implementing sequential as well as combinatorial circuits. Its internal architecture is given in figure 2.3. Each CLB element contains a pair of slices and it is connected to a switch matrix for access to the general routing matrix. Slice consists of four 6-input Look-Up-Tables (LUTs) which perform logic operations, eight Flip-Flop (FF) registers to store the result of the LUT, Wide multiplexers and arithmetic carry logic.

The LUT is the basic block of FPGA and is capable of implementing any logic function of N-Boolean variables. Essentially, this element is a truth table in which different combinations of the inputs implement different functions to yield output values. The limit on the size of the truth table is N, where N represents the number of inputs to the LUT.

The flip-flop is the basic storage unit within the FPGA fabric. This element is always paired with a LUT to assist in logic pipelining and data storage. The basic structure of a flip-flop includes a data input, clock input, clock enable, reset, and data output. During normal operation, any value at the data input port is latched and passed to the output on every pulse of the clock. The purpose of the clock enable pin is to allow the flip-flop to hold a specific value for more than one clock pulse. New data inputs are only latched and passed to the data output port when both clock and clock enable are equal to one.

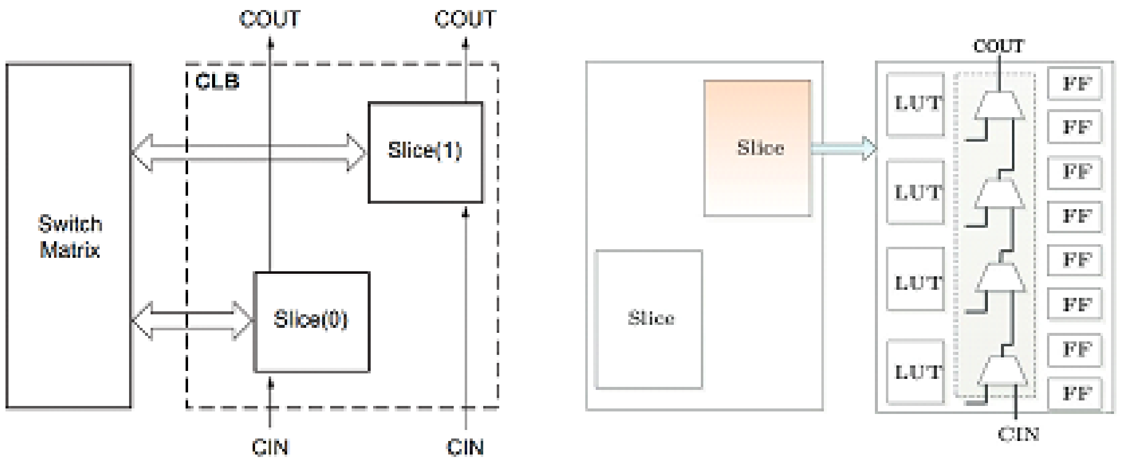


Figure 2. 3 CLB internal architecture [70]

As illustrated by Figure 2.4, FPGA basic architecture is incorporated with dedicated hardware elements and data storage blocks to increase the computational density and efficiency of the device. The contemporary FPGA Architecture is composed by: Embedded memories for distributed data storage, Phase-locked loops (PLLs) for driving the FPGA fabric at different clock rates, High-speed serial transceivers, Off-chip memory controllers and Multiply-accumulate blocks.

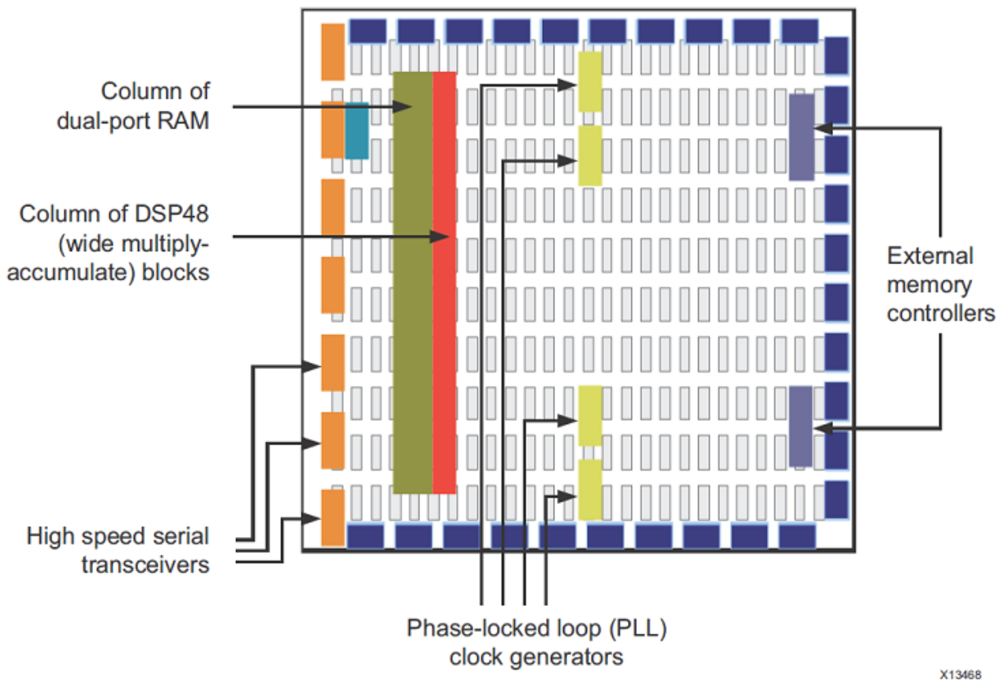

Column of dual-port RAM
Column of DSP48 (wide multiply-accumulate) blocks
High speed serial transceivers
External memory controllers
Phase-locked loop (PLL) clock generators
X13468

Figure 2. 4 The contemporary FPGA internal Architecture [69]

2.3.2 FPGA Design Flow

The FPGA designers must go through a series of steps from initial ideas to the final hardware circuit. Figure 2.5 demonstrates the design flow of FPGA-based implementations. The most common FPGA-based design flow used nowadays is based on the following phases [72]:

1. Formal Specification: The desired system functionalities and requirements are expressed in a formal model and then the description is validated by simulation or verification techniques. The result is a functional specification, without any implementation details.

2. Exploration: The resulting formal model is used to analyze the best SW/HW partitioning which depends on the goals of the targeted application. The result of this phase is the definition of functions that will be implemented in software and which ones will be performed by dedicated hardware components.

3. Hardware and software synthesis: An implementation is created on FPGA for each component of the system architecture defined in the previous step. The result is the

software code for the software components and the RT-level implementation for the hardware parts. An interface between the HW and the SW must be established.

4. Verification and Evaluation: In this step, verification of the correct behavior of software and hardware components as well as the targeted system is performed. Then evaluation of the final embedded system performance is performed in terms of speed, area, costs, real-time constraints, power consumption, reliability, availability and safety. This phase allows validating the proposal that should satisfy the requirements.

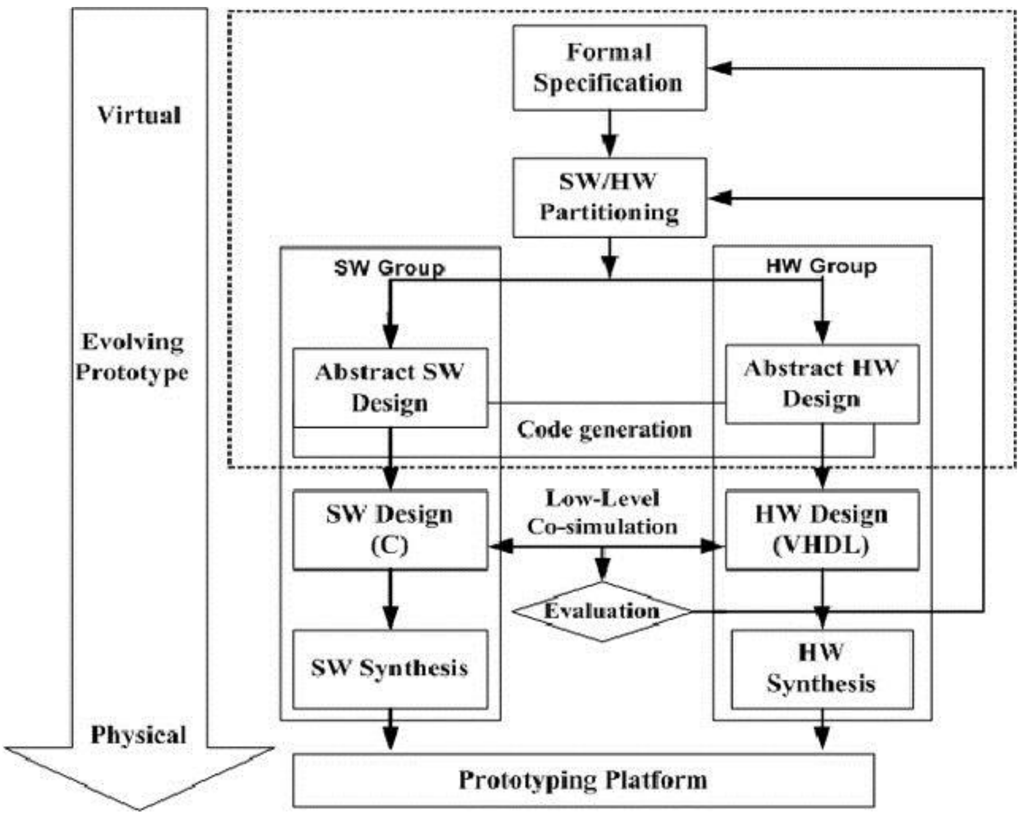


Figure 2. 5 FPGA design Flow [71]

2.3.3 FPGA-based MPSoC

For decades, MPSoC approach was adopted by FPGA platforms as a suitable solution for providing a high level of parallelism [67]. MPSoC is an embedded system that integrates multiple embedded processors to run multitasking software programs in parallel mode. It refers to the hardware components and the software programs. The MPSoC approach is a promising trend in embedded applications for responding the requirements of recent embedded systems: real-time, flexibility, speed, area and low-power. MPSoC performance is determined by the capacity of the used hardware components and the partitioning of the targeted software program into tasks for parallel execution.

2.3.3.1 Hardware architecture for MPSoC

MPSoC hardware architecture is composed mainly of three elements: the embedded processors, the interconnect network that connects the processors and the memory hierarchy. It could be categorized according to the following [73]:

1. The kind of integrated processors: MPSoCs are classified by two approaches, namely, homogeneous and heterogeneous. The homogeneous approach consists of using identical processors while the heterogeneous approach consists of combining softcore and hardcore processors in the same architecture.

2. The behavior of the integrated processors: MPSoC could be also categorized into three implementation approaches: Master/Slave system, Net architecture system and Pipelined system. In the Master/slave approach, one or more processors act as the master processor, controlling the behavior of the other slave processors. These latter have the ability to communicate only with their masters, contrary to Net approach where there is no hierarchical organization between the processors, thus, each processor can communicate with other processors and exchange data with them. In Pipeline architecture, each processor acts as a pipeline stage for the other processors to execute a function in pipeline mode.

3. The topology of physical communication between the integrated processors: the communication can be done within 3 physical topologies: point-to-point, shared bus and net-on-chip. In the first topology, each processor is connected directly to another processor through point to point fast link bus. This approach is not efficient when the MPSoC system is large. The second approach is based on a shared communication bus used by all the processors. The third approach ensures the communication between the processors by installing small routers to manage the communication between a set of processors. The weakness of the shared bus and net-on-chip schemes is lies in performance.

Depending on the considered FPGA-based MPSoC architecture, two data/instruction exchanging techniques are proposed: shared memory and distributed memory. In the first technique, the communication between the processors is done by writing to and reading from a shared memory so that any changes happened will be visible to the other processors. While, the second technique is based on separated memories where each processor manages its own local memory. The instruction/data exchanging in distributed memory is ensured directly between two point-to-point interconnected processors or explicitly via a shared bus or interconnection network.

The shared memory system is commonly used in FPGAs because of memory limitation constraints. The shared memory system is classified to uniform memory access (UMA), non-uniform memory access (NUMA) and Cache-Only Memory Architecture (COMA). In the UMA system, all the processors have equal access permission to read from and write from any location of a memory location. In the NUMA system, each processor has a portion of the shared memory. However, it is almost impossible to prevent any processor from any memory location since that the shared memory has a single address space. The Cash-Only Memory Architecture (COMA) has the same NUMA architecture, except that here the shared memory contains only cashes and no memory hierarchy is included. Instead, a memory directory is involved to help in remote cache access. The main problem of the shared memory technique is the control of the shared memory access and the synchronization of data when multiple processors try to manipulate the shared data at the same time. Therefore, shared memory is used with synchronization mechanisms such as semaphores, barriers and locks to avoid the overlap between processors. These mechanisms provide an easy access management to the shared memory by determining if the memory resource is available. Thus, the processor must wait for other processors to unlock the critical section. The most important advantage of distributed memory systems is abandoning the need for large global memory as well as the elimination of the need for synchronization. Thus, distributed memory systems lead to higher performance and also the flexibility to integrate a large number of processors and in addition to being easily scalable.

2.3.3.2 Embedded processors

In FPGA-based MPSoC systems, the most used embedded processors are either softcore such as Xilinx Microbaze and Alter Nios processors or hardcore like Xilinx PowerPc and ARM-based processors. A soft-core processor is a microprocessor described in an HDL language, which can be synthesized in programmable hardware, such as FPGAs. These processors implemented in FPGAs can be easily configured to the needs of the target application. A hardcore processor is a processor that's physically implemented as a structure in the silicon. It is highly optimized to provide high-performance. In the following MicroBlaze softcore and ARM Cortex-9 hardcore processors, used in our work are presented.

A. MicroBlaze:

MicroBlaze is Xilinx 32-bit soft-core processor based on Reduced Instruction Set Computer (RISC) architecture [34]. Its internal architecture is highly configurable as shown in figure 2.6. The configurable components include cache size, pipeline depth (3-stage on 5-stage), Memory Management Unit (MMU), bus interfaces and arithmetic units. MicroBlaze supports different interconnect systems. The primary used system is the PLB bus, which is a traditional system-memory mapped transaction bus with master/slave capability. MicroBlaze uses a dedicated LMB interconnect for communicating to local memory. It contains 16 FSL (Fast Simplex Link) bus which is a dedicated point-to-point unidirectional FIFO connection.

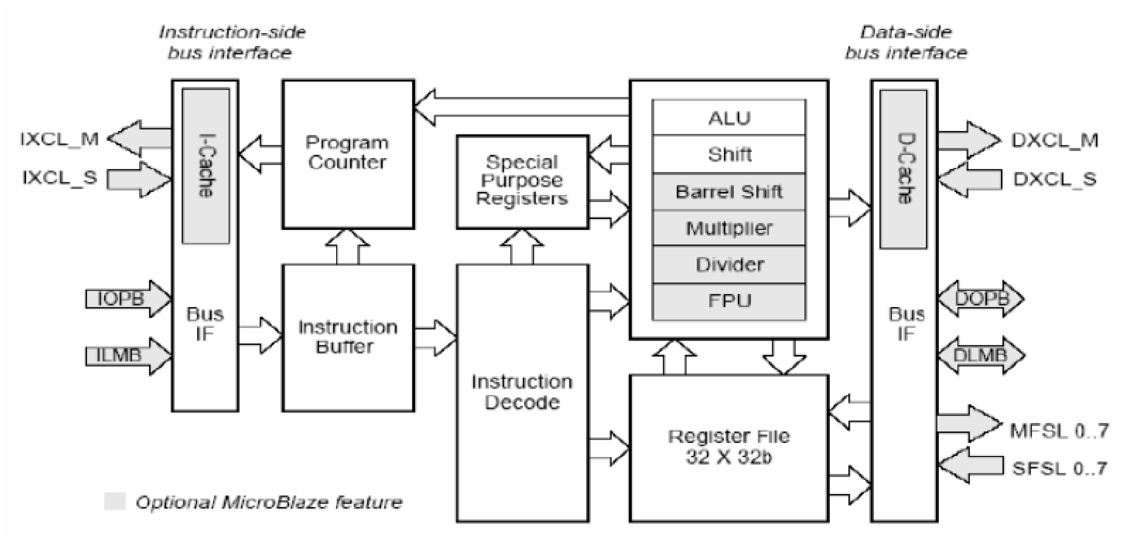


Figure 2. 6 Internal architecture of Xilinx MicroBlaze softcore processor [34]

B. ARM Cortex-A9

ARM Cortex-A9 [54] is a 32-bit hardcore processor licensed by ARM Holdings implementing the ARMv7-A architecture. Figure 2.7 presents the internal architecture of ARM Cortex-A9.

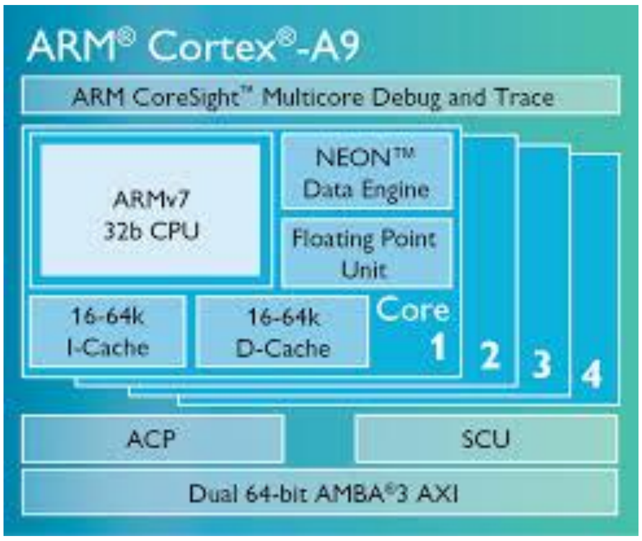


Figure 2. 7 Internal architecture of ARM Cortex-A9 hardcore processor [54]

The Cortex-A9 processor is a multicore processor providing from one to four cache-coherent cores. It is a popular choice for high-performance, low-power cost-sensitive applications. Cortex-A9 processors are proven to offer highly effective outcomes in embedded systems. Additionally, they are able to implement multi-core designs that further scale the performance increase. As Cortex-A9 is based on ARMv7-A architecture, the following benefits can be obtained [54]:

1. Dynamic length pipeline (8-11 stages).
2. Highly configurable Level-1 (L1) caches.
3. NEON technology can be implemented.
4. Scalable multi-core configuration with up to 4 coherent cores.

2.3.3.3 Difficulties of Writing Parallel Programs for MPSoC

The development of parallel programs on MPSoC architectures is not an easy task and is more difficult than sequential programs. The challenge consists not only of an efficient partitioning of a program over multiple processors, where each processor executes a set of tasks, and taking into account the coordination between them. The difficulties of writing parallel programs for MPSoC are summarized as follows [74]:

- Finding the best level of parallelism: The number of processing units is limited not only by the limited area but also by the level of parallelism of the application. Thus, the first problem to which the developer is confronted to identify the parallel code sections in the application.

- Debugging and Profiling issues: For a sequential application, the developer has only one application to debug / profile, but for a parallel application running on multiple processors, the developer faces several threads. Analyzing the complex interactions, the concurrent processes, and the relationships between program processes is a challenging task.

- Characteristics of MPSoC architecture: Embedded software development is challenging because of the hardware complexity of MPSoC. It requires parallel programming for homogeneous or heterogeneous multiprocessors. It also must take into account diverse communication architectures and design constraints, such as hardware cost, power, and timeliness. Therefore, the developer must understand the various complete characteristics of MPSoC hardware.

2.4 Conclusion

As presented in the previous chapter, cryptography could be embedded on several interesting hardware platforms. In this chapter, a comparison between these platforms is done first in order to understand the underlying of selection criteria between them. FPGAs are the best reconfigurable platforms that fill the gap and balance between the GPP flexibility and ASICs high-speed performance. The structure of FPGA and the design flow cycle for FPGA-based embedded systems are presented. The FPGA structure allows providing high level of parallelism by the integration possibility of multiple embedded processors and hardware accelerators in a single chip based on MPSoC approach. The development of MPSoC-based ECC parallel architecture depends not only on the efficient ECSM partitioning over multiple tasks but also on the kind of the integrated processors, the behavior of the integrated processors and the topology of physical communication between the integrated processors. Thus, the techniques used for designing efficient MPSoC-based parallel architectures on FPGA circuits are illustrated at the end of this chapter. The reported techniques are considered in our first contribution in this thesis which consists of the proposition of MPSoC-based parallel architectures for efficient ECSM computation on FPGA circuits. The proposed architectures are described in the next chapter.

CHAPTER 3
MICROBLAZE-BASED MULTIPROCESSOR EMBEDDED CRYPTOSYSTEM ON XILINX VIRTEX-5 FPGA FOR ELLIPTIC CURVE SCALAR MULTIPLICATION OVER Fp

3.1 Introduction

Elliptic curve point multiplication is the most time-consuming operation in ECC schemes. The performance improvement of this operation will have a significant impact on the overall performance of the ECC system. The performance is usually analyzed in terms of flexibility, security, execution time, and area consumption. However, it is difficult to provide all these constraints at the same time. Thus, we present several architectures of ECSM computation for efficient ECC designs on FPGA circuits. Our main aim is to investigate the best trade-off between flexibility, security, execution time, and occupied area.

This chapter presents novel SW/HW parallel architectures of ECSM computation for efficient ECC cryptosystems on FPGA circuits. It begins by reporting the considered design methodology in our proposals. Next, the adopted algorithms of ECSM computation in our designs that provide high security, low computation complexity, and high parallelism possibility are presented. Then, the parallelism investigation for the independent operations within ECSM computations and its impact on ECSM critical path delay is illustrated. Finally, the proposed architectures with the experimental results and the performance comparison with some recent works are discussed.

3.2 Design methodology

Depending on the considered goals, the proposed designs involves several algorithmic and architectural optimizations:

1. To yield high-security, we propose to perform ECSM in our designs based on MPL algorithm. In fact, the security of embedded applications relies not only on the hardness of ECC mathematical systems but also on their secure implementation on the hardware platforms. These platforms can leak sensitive information during the computation execution, by observing their behavior, such as power consumption and

electromagnetic emanation. Hence, efficient implementation of embedded cryptosystems needs the study of the vulnerability from various physical attacks on hardware platforms. As it is not possible to fully protect a device from all attacks, side-channel attack resistance is considered in all the proposed designs at the algorithmic level. Using point multiplication algorithms that employ a regular sequence of ECPA and ECPD, independently of the scalar is a method of protecting against SPA attacks. Therefore, we propose to implement ECSM based on the binary regular MPL algorithm. Each iteration of the MPL algorithm performs ECPA followed by ECPD, so that the side-channel information is viewed as a regular alternating series of points doublings and additions.

2. To yield high-flexibility, we propose to implement ECSM in software by using Xilinx's MicroBlaze softcore processor. Implementing ECSM in software offers flexible and easy design adaptation to the changes in curve parameters, field sizes, and ECSM algorithms. Moreover, the use of MicroBlaze processors increases the integration possibility of our designs on large FPGA circuits since MicroBlaze can be implemented on any of the Xilinx FPGA families. In such a way, all the proposed designs support any arbitrary prime curves and offer an easier design modification capability. These features are desired due to the rapid changes in cryptographic algorithms and standards.

3. To yield time-area balance, various optimization techniques are adopted in our designs at algorithmic and architectural levels. In fact, the implementation of ECSM completely in software using MicroBlaze processor offers a high degree of flexibility with less area consumption. However, it provides a very slow execution time due to the computation complexity of ECSM operation. To improve the performance of ECSM execution in our designs, we propose to:

   - Reduce the computation complexity of ECPA and ECPD by using the projective system. Projective coordinate system is a better choice than affine coordinates to provide the best trade-off between speed and area. It reduces the computation complexity of ECSM by avoiding the execution of modular inversion in return using extra modular multiplications. The modular inversion is very costly operation in terms of hardware resources. Thus, the projective system is used in the proposed architectures to achieve high-speed ECSM computation with low-area requirements for efficient ECC designs.

- Speed-up the execution time of ECSM by implementing MM within a scalable HW accelerator hardware accelerator coprocessor around MicroBlaze processor. The computations of ECSM in the projective systems are based on large operand finite field operations of which modular multiplication is the most critical operation. Thus, a high-speed MM hardware accelerator coprocessor leads to the execution time improvement of the overall ECSM system. Therefore, we propose to implement MM within an area-optimized HW accelerator core based on Montgomery Modular Multiplication (MMM) algorithm. MMM is an efficient method to perform fast MM reduction when an arbitrary large prime modulo is targeted, which is the case in our work. The proposed hardware accelerator is integrated around MicroBlaze processor as a coprocessor. MicroBlaze is based on 32-bit RISC architecture. To adapt the execution of MMM to the data bus of the MicroBlaze processor and to benefit from the DSP48E hardware 32-bit multipliers available on FPGAs, we proposed to use the modified high radix $r(r=2^{32})$ MMM algorithm. High-radix based multipliers are faster than the bit-level implementations because of their lower iteration. The major advantage of our SW/HW partitioning is the enhancement of ECSM computation time by using few extra resources with maintaining high-flexibility in all the proposed designs since the point operations are managed in software by MicroBlaze.

- Reduce the critical path delay by performing ECSM in a parallel way. To obtain maximum performance, we investigate the independent operations in ECSM computation that could be executed in parallel. Our investigation demonstrates that the combination of MPL algorithm and the projective system shows a good scope of parallelism in ECSM hierarchy level with several parallelization degrees. To exploit the inherent parallelism in ECSM computations, we propose to integrate multiple MicroBlaze processors with multiple MMM accelerator cores in a single architecture.

Based on the adopted optimization techniques, five SW/HW architectures are proposed in this work. All the proposed architectures perform ECSM based on MPL algorithm and projective. Their internal architectures consist of the integration of MicroBlaze processor to maintain high-flexibility. The first architecture is a single MicroBlaze-based SW/HW implementation of sequential ECSM computation. It is proposed mainly as a comparison model that achieves good execution time with minimum hardware resources. The four other architectures are multiple MicroBlaze-based SW/HW implementation of parallel ECSM computation. Each of the proposed parallel designs integrates more processors than the previous one. The idea behind the four architectures is to investigate the optimum number of MicrBlaze processors to achieve the best trade-off between execution time and area consumption.

3.3 Adopted ECSM algorithms

Figure 3.1 represents a typical ECSM implementation hierarchy. The first level corresponds to the adopted algorithm for ECSM ($g \times P$), which is Montgomery Power Ladder. The second level corresponds to the point representation system used to perform ECPA and ECPD operations. In this work, projective coordinate system is used. The third level is composed of the required finite field arithmetic operations. In the following, the adopted algorithms in the three levels are detailed.

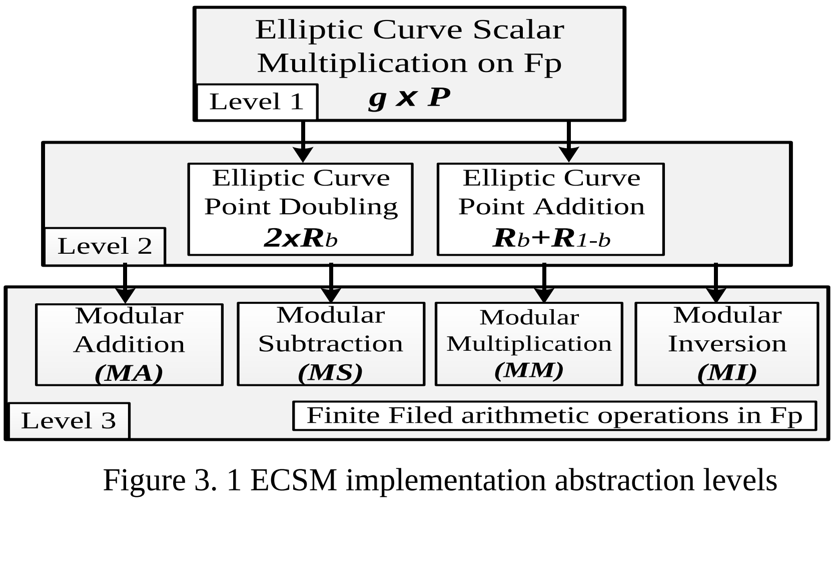

Elliptic Curve Scalar Multiplication on Fp
Level 1
g x P
Elliptic Curve Point Doubling
2xRb
Elliptic Curve Point Addition
Rb+R1-b
Level 2
Modular Addition (MA)
Modular Subtraction (MS)
Modular Multiplication (MM)
Modular Inversion (MI)
Level 3
Finite Filed arithmetic operations in Fp

Figure 3. 1 ECSM implementation abstraction levels

3.3.1 Montgomery Power Ladder Algorithm

ECSM is the main operation of most ECC schemes. It is of the form $C = g \times P$ where (C,P) are two points and g is an integer of w-bit. From the literature, ECSM can be performed by a variety of algorithms according to the representation of the scalar g [75]. MPL binary method [31] for ECSM is simple and efficient method. It is based on the binary representation of the scalar $g = \sum_{i=0}^{w-1} g_i \times 2^i$, such that:

$$C = [2 \times (\dots(2 \times (2 \times g_{w-1} + g_{w-2}) + g_{w-3}) + \cdots) + g_0] \times P \tag{3.1}$$

The security analysis shows that the implementation of MPL algorithm on embedded systems ensures the resistance against various side-channel attacks [25] by performing ECPD and ECPA in parallel independently of the current scalar bit value g_{i} [28, 29]. However, if the hardware resources are limited, this algorithm can be executed in sequential mode. The MPL binary method is given in algorithm 3.1. In this algorithm, two intermediate points R_0 and R_1 are required for the computation of $C = g \times P$. When the current scalar bit value $b = 1$, $R_{0(i)} = 2 \times R_{0(i+1)}$ and $R_{1(i)} = R_{0(i+1)} + R_{1(i+1)}$ are performed. On the other hand, $R_{1(i)} = 2 \times R_{1(i+1)}$ and $R_{0(i)} = R_{0(i+1)} + R_{1(i+1)}$ are carried out when $b = 0$. The performance execution of ECSM depends on ECPA and ECPD computations. Thus, an efficient implementation of ECSM requires the optimization of both ECPA and ECPD execution performances

Algorithm 3. 1 Montgomery power ladder for ECSM [31]

```
Inputs : $g = \sum_{i=0}^{w-1} g_i \times 2^i$ , $P(x_P, y_P)$
Variables : $R_{0(w-1)}(x_{R0}, y_{R0}) \in E_p$, $R_{1(w-1)}(x_{R1}, y_{R1}) \in E_p$.
Output : $C = g \times P$
Begin
1. $R_{0(w-1)} = P$
2. $R_{1(w-1)} = 2 \times P$                      //ECPD
3. For i from w-2 down to 0 do
4.     $b = g_i$
5.     $R_{b(i)} = 2 \times R_{b(i+1)}$                // ECPD
6.     $R_{1-b(i)} = R_{b(i+1)} + R_{1-b(i+1)}$          // ECPA
   End For
7. $C = R_{0(0)}$
8. Return $C$
End
```

3.3.2 Projective coordinates system

ECPA and ECPD operations could be carried out in an affine coordinates system. This system suffers from the latency of the Modular Inversion (MI) problem. This operation is the most costly and complex operation for hardware implement [24]. Hence, the affine coordinate system is an inefficient choice for low area implementations. The best method to avoid MI operation consists of using a projective coordinates system [63, 64]. In such system, the execution of ECPA and ECPD requires the computation of Modular Addition (MA), Modular Subtraction (MS) and Modular Multiplication (MM).

In the projective coordinates system, a point is defined by (X, Y, Z) instead of (x, y). The computation of $C = g \times P$ needs to convert first the coordinates (x_P, y_P) of the point P to $(X_P,\ Y_P,\ Z_P)$ as follows:

$$(X_P,\ Y_P,\ Z_P) = (x_P, y_P, 1) \qquad (3.2)$$

Then performing MPL algorithm with the new coordinates of P, where ECPA and ECPD are performed in projective coordinates. The last step applies MI operation to convert back the coordinates of the resulting point $C(X_C, Y_C, Z_C)$, to affine coordinates $C(x_C, y_C)$, such that:

$$(x_C, y_C) = (X_C \times {Z_C}^{-c}, Y_C \times {Z_C}^{-d}) \qquad (3.3)$$

c and d represent the parameters of the projective coordinates system. In the literature, according to c and d, three variants are proposed as forms of projective system, namely, standard projective system, Jacobian projective system and Chudnovsky projective system. The comparison between these forms is given in Table 3.1. Since the complexities of MA/S are low compared to the MM complexity, the comparison is reported in terms of number of MM.

Table 3. 1 Summary of projective coordinate systems complexities

Systems (c,d)	ECPA	ECPD	ECPA+ECPD
Standard (1,1)	$14 \times MM$	$12 \times MM$	$26 \times MM$
Jacobian (2,3)	$16 \times MM$	$10 \times MM$	$26 \times MM$
Chudnovsky(2,3)	$14 \times MM$	$11 \times MM$	$25 \times MM$

Through Table 3.1, we note that the Chudnovsky system requires $25 \times MM$ at each iteration, whereas, the other systems require $26 \times MM$. Hence, the Chudnovsky system has the best performance for sequential ECSM execution. Furthermore, if the parallel execution of ECPA and ECPD is targeted, Chunovsky and Standard systems offer the same performance. The complexity in terms of the MMs number is about $14 \times MM$. This latter is required by ECPA computation. From these comparisons, the Chudnovsky system is our choice for the implementation of ECSM as an MPSoPC cryptosystem.

3.3.3 Finite field operations

The study of the required arithmetic operations is crucial, before the implementation of ECSM. In the required finite field arithmetic operations for ECSM computation, the data are unsigned integers of w-bit, defined in the set $\{0,1,\ldots,p-1\}$. However, MicroBlaze is not designed to perform large finite field arithmetic operations. In order to adapt the execution of these operations to internal resources of the integrated processors, the effective approach is to use radix-r ($r = 2^k$) sequential version algorithms. The value of the parameter r is chosen according to the data path size of MicroBlaze which is about 32-bit [34]. Hence, the value of k is of 32. In our work, the inputs, outputs and intermediate variables of all the arithmetic operations are decomposed first on e digits of 32-bit size. In other words, the data X is coded in radix-2^{32} and stored in an array of size $e = w/32$, using expression (3.4). Then the computations are performed digit-by-digit in a serial mode.

$$X = \sum_{i=0}^{e-1} X[i] \times 2^{i\times 32} \text{ with: } X[i] = \sum_{j=0}^{31} X_j \times 2^j \tag{3.4}$$

3.3.3.1 Modular Addition and Subtraction

The modular addition adds two operands A and B, then reduces the sum $(A + B)$ over modulo p. Its expression is given by : $Z = (A + B) \bmod p$. To obtain Z, MA requires first the computation of two intermediate variables $Z1$ and $Z2$, such that, $Z1 = A + B$ and $Z2 = Z1 - p$. Z is selected between $Z1$ and $Z2$ according to the sign of $Z2$. Algorithm 3.2 presents radix-2^{32} Modular Addition. In this algorithm, the computations of $Z1$ and $Z2$ are carried out digit-by-digit in the main loop (i). The sign of $Z2$ is obtained from the last value of $C2_e$.

Algorithm 3. 2 Radix-2^{32} Modular Addition

```
Inputs : $A = \sum_{i=0}^{e} A[i] \times 2^{i\times 32}, B = \sum_{i=0}^{e} B[i] \times 2^{i\times 32},$
         $p = \sum_{i=0}^{e} p[i] \times 2^{i\times 32},\ A[e] = B[e] = p[e] = 0,$
Output : $Z = \sum_{i=0}^{e} Z[i] \times 2^{i\times 32}$
Begin
1.  $C1_{-1} = 0;\ C2_{-1} = 1;$
2.  For $i$ from $0$ to $e$
3.      $(C1_i, Z1[i]) = A[i] + B[i] + C1_{i-1}$
4.      $(C2_i, Z2[i]) = Z1[i] + \overline{p[i]} + C2_{i-1}$
    End For
5.  If $(C2_e = 0)$ then $Z = Z2$
6.      Else $Z = Z1$
7.  Return $Z;$
End.
```

The modular subtraction is defined as: $Z = (A - B)\ mod\ p$. It subtracts B from A, and reduces the result $(A - B)$ over modulo p. Algorithm 3. 3 shows radix-2^{32} Modular Subtraction. In this algorithm, two intermediates variables $Z1$ and $Z2$ are used, where $Z1 = A - B$ and $Z2 = Z1 + p$. The result Z is selected between $Z1$ and $Z2$ based on the sign of $Z1$, obtained according to the last value of $C1_e$

Algorithm 3. 3 Radix-2^{32} Modular Subtraction

```
Inputs : $A = \sum_{i=0}^{e} A[i] \times 2^{i\times 32}, B = \sum_{i=0}^{e} B[i] \times 2^{i\times 32},$
         $p = \sum_{i=0}^{e} p[i] \times 2^{i\times 32},\ A[e] = B[e] = p[e] = 0,$
Output : $Z = \sum_{i=0}^{e} Z[i] \times 2^{i\times 32}$
Begin
1.  $C1_{-1} = 0;\ C2_{-1} = 1;$
2.  For $i$ from $0$ to $e$
3.      $(C1_i, Z1[i]) = A[i] + \overline{B[i]} + C1_{i-1}$
4.      $(C2_i, Z2[i]) = Z1[i] + p[i] + C2_{i-1}$
    End For
5.  If $(C1_e = 0)$ then $Z = Z1$
6.      Else $Z = Z2$
7.  Return $Z;$
End.
```

3.3.3.2 Montgomery Modular Multiplication

MM is the critical finite field in the projective system. It defined by $S = (A \times B)\ mod\ p$. The reduction of the result of the product $A \times B$ over the modulus p requires the division by p. Montgomery Modular Multiplication (MMM) [32, 66] is an efficient method to perform fast MM reduction when arbitrary ECs over a large prime field ($\mathrm{F_p}$) are targeted. It performs the reduction by a series of additions and right shifts instead of using division. The MMM based on High radix $r(r = 2^k)$ is defined by expression (3.5). $R \geq 2^{w \times k}$ is the Montgomery constant, where $\gcd(r, p) = 1$. This condition is often satisfied, since p is prime integer.

$$S = Mont(A, B) = (A \times B \times R^{-1})\ mod\ p \qquad (3.5)$$

Note that the result S is obtained with an additional factor R^{-1}. Hence, the computation of ECSM based on the MMM algorithm requires to convert first the parameter a and the coordinates ($\mathrm{X_P}$, $\mathrm{Y_P}$, $\mathrm{Z_P}$) of the point P in Montgomery domain, such that:

$$Mont(a, R^2) = a \times R\ mod\ p \qquad (3.6)$$

$$\left(Mont(X_P, R^2), Mont(\ Y_P, R^2), Mont(\ Z_P, R^2)\right) = (X_P \times R, Y_P \times R, Z_P \times R) \qquad (3.7)$$

Then executing the MPL algorithm with the new parameters. The last step is to convert back the coordinates of the resulting point ($\mathrm{x_c} \times R$, $\mathrm{y_c} \times R$) from Montgomery representation to classical representation, as follows:

$$\left(Mont(x_c \times R, 1),\ \ Mont(\ y_c \times R, 1)\right) = (x_c, y_c) \qquad (3.8)$$

Algorithm 3.4 represents the scalable version of the original MMM algorithm, where the entire basics arithmetic operations are executed in radix-2^{32} [33]. This algorithm is based on iterative calculation of the intermediate results S_{i+1} which are defined by expression (3.9). It uses two intermediate variables M_i and L_i to obtain S_{i+1}.

$$S_{i+1} = (S_i + (A[i] \times B) + (q_i \times p))/2^{32} \qquad (3.9)$$

Where: $q_i = (S_i + A[i] \times B[0]) \times p'\ mod\ 2^{32}$.

Algorithm 3. 4 Radix-2^{32} Montgomery Modular Multiplication [36]

```
Inputs: $A = \sum_{i=0}^{e} A[i] \times 2^{i\times32}, B = \sum_{i=0}^{e} B[i] \times 2^{i\times32}, p = \sum_{i=0}^{e} p[i] \times 2^{i\times32}$.
Variables: $H1_i = \sum_{j=0}^{e} H1[j]_i \times 2^{j\times32}, H2_i = \sum_{j=0}^{e} H2[j]_i \times 2^{j\times32}$,
        $M_i = \sum_{j=0}^{e} M[j]_i \times 2^{j\times32}, L_i = \sum_{j=0}^{e} L[j]_i \times 2^{j\times32}$,
        $C1_i = \sum_{j=0}^{e} C1[j]_i \times 2^{j\times32}, C2_i = \sum_{j=0}^{e} C2[j]_i \times 2^{j\times32}$,
        $c1_j = c2_j = c3_j = c4_j$
Pre-computed: $p' = -p[0]^{-1} \bmod 2^{32}$.
Output: $S_{e+1} = \sum_{j=0}^{e} S[j]_{e+1} \times 2^{j\times32} = (A \times B \times R^{-1}) \bmod p$.
Begin
1.  $S_0 = \sum_{j=0}^{e} S[j]_0 \times 2^{j\times32} = 0$
2.  For $i$ from $0$ to $e$
3.    $C1[-1]_i = 0 \quad C2[-1]_i = 0$;
4.    $c1_{-1} = c2_{-1} = c3_{-1} = c4_{-1} = 0$;
5.    $H_i = S[0]_i + A[i] \times B[0]$
6.    $q_i = H_i \times p' \bmod 2^{32}$
7.    For $j$ from $0$ to $e$
8.      $(C1[j]_i, H1[j]_i) = A[i] \times B[j]$
9.      $(c2_j, c1_j, M[j]_i) = H1[j]_i + S[j]_i + C1[j-1]_i + c1_{j-1} + c2_{j-1}$
10.     $(C2[j]_i, H2[j]_i) = q_i \times p[j]$
11.     $(c4_j, c3_j, L[j]_i) = M[j]_i + H2[j]_i + C2[j-1]_i + c3_{j-1} + c4_{j-1}$
12.     $S[j-1]_{i+1} = L[j]_i$
      End For
13.   $S[e]_{i+1} = c1_e + c2_e + c3_e + c4_e + C1[e]_i + C2[e]_i$
    End For
11. Return $S_{e+1}$
End.
```

In this algorithm, two nested loops (i) and (j) are introduced. The outer loop (i) scans the digits $A[i]$ and computes q_i in line 6. The reduction $(H_i \times p') \bmod 2^{32}$, for q_i calculation, requires to determinate the least significant digit of $(H_i \times p')$ result. The inner loop (j) reads the digits $B[j]$ and $p[j]$ to compute the digits of the variables M_i and L_i. The computation of M_i requires the multiplication $A[i] \times B[j]$ of line 8 and the additions of line 9. The most and the least significant digits of the multiplication result are stored in the variables $C1[j]_i$ and $H1[j]_i$, respectively. The computation of L_i is carried out by performing first the multiplication $q_i \times p[j]$ of line 10 then the additions of line 11. The most and the least significant digits of line 10 result are stored in the variables $C2[j]_i$ and $H2[j]_i$, respectively. The carries c1j, c2j, c3j, c4j and the digits $C1[j]_i$, $C2[j]_i$ are stored in order to be used in the computations of the next iteration (j+1).

Through expression (3.9), the computation of S_{i+1} requires the division over 2^{32}. This operation is performed in line 12 using the right shift operation. The final result S_{e+1} is obtained for $i = j = e$.

3.3.3.3 Modular Inversion

In the combination of the MPL algorithm with the projective system, MI operation does not affect the ECSM performance, since it is executed one time for converting back the resulting point coordinates from Chudnovsky to the affine system. Hence, we propose to perform MI using modular exponentiation (Mexp) according to Fermat's little theorem as it is shown in expression (3.10) [3]. The easiest way to perform Mexp is the left-to-right binary method [36].

$$A^{-1} = A^{p-2} \, mod \, (p) \tag{3.10}$$

3.4 Parallelism exploration in ECSM hierarchy

In this section, the parallelization possibilities for the computation of ECSM using the MPL algorithm in the Chudnovsky projective system are studied. The exploration of the parallelism within ECSM abstractions levels leads to improve the performance of ECSM execution. However, in the case of limited-resources FPGA circuits, ECSM could be carried out sequentially using a single MicroBlaze processor and hardware accelerator multiplier. The critical path delay Δ_{ECSM} of single ECSM in this approach can be evaluated by expression (3.11), where Δ_{ECPA} and Δ_{ECPD} are respectively the execution delays of ECPA and ECPD executions in the Chudnovsky system.

$$\Delta_{ECSM} = (w-1) \times \Delta_{ECPA} + w \times \Delta_{ECPD} \tag{3.11}$$

Δ_{ECPD} and Δ_{ECPA} depend on the formulas of ECPD and ECPA coordinates computations, respectively. In the Chudnovsky system, a point is represented by (X, Y, Z, Z^2, Z^3). The coordinates of ECPD resulting point are defined by the equation system (3.12).

$$R_{b(i)} = 2 \times R_{b(i+1)} = \begin{cases} X_{b(i)} = M^2 - 2 \times S \\ Y_{b(i)} = M \times (S - X_{b(i)}) - T \\ Z_{b(i)} = 2 \times Y_{b(i+1)} \times Z_{b(i+1)} \\ Z^2_{b(i)} = Z_{b(i)} \times Z_{b(i)} \\ Z^3_{b(i)} = Z^2_{b(i)} \times Z_{b(i)} \end{cases} \tag{3.12}$$

Where: $$\begin{cases} \mathrm{S} = 4 \times \mathrm{X}_{b(i+1)} \times Y^2_{b(i+1)} \\ \mathrm{M} = 3 \times \mathrm{X}^2_{b(i+1)} + \mathrm{a} \times (\mathrm{Z}^2_{b(i+1)})^2 \\ T = 8 \times \mathrm{Y}^4_{b(i+1)} \end{cases}$$

The coordinates of ECPA resulting point are defined by the equation system (3.13).

$$R_{1-b(i)} = R_{b(i+1)} + R_{1-b(i+1)} = \begin{cases} X_{1-b(i)} = S^2 - U^3 - 2 \times V \\ Y_{1-b(i)} = S \times \left(V - X_{1-b(i)}\right) - G \\ Z_{1-b(i)} = U \times Z_{b(i+1)} \times Z_{1-b(i+1)} \\ Z^2_{1-b(i)} = Z_{1-b(i)} \times Z_{1-b(i)} \\ Z^3_{1-b(i)} = Z^2_{1-b(i)} \times Z_{1-b(i)} \end{cases} \tag{3.13}$$

Where: $$\begin{cases} U = U2 - U1,\ \mathrm{U1} = \mathrm{X}_{b(i+1)} \times \mathrm{Z}^2_{1-b(i+1)}\ ,\ \mathrm{U2} = \mathrm{X}_{1-b(i+1)} \times \mathrm{Z}^2_{b(i+1)} \\ S = S2 - S1,\ S1 = \mathrm{Y}_{b(i+1)} \times \mathrm{Z}^3_{1-b(i+1)},\ S2 = \mathrm{Y}_{1-b(i+1)} \times \mathrm{Z}^3_{\mathrm{b}(i+1)} \\ V = \mathrm{U1} \times \mathrm{U}^2\ \ ,\ \ G = \mathrm{S1} \times \mathrm{U}^3 \end{cases}$$

Through these formulas, Δ_{ECPD} and Δ_{ECPA} could be evaluated respectively by expressions (3.14) and (3.15). Note that the multiplications by the constants 2, 3, 4 and 8 required for the computation of variables $X_{1-b(i)}, M, S, T$ and $Z_{b(i)}$ are performed by a sequence of MA.

$$\Delta_{ECPD} = 11 \times MM + 10 \times MA/S. \tag{3.14}$$

$$\Delta_{ECPA} = 14 \times MM + 7 \times MA/S. \tag{3.15}$$

By substituting the critical path delays Δ_{ECPD} and Δ_{ECPA} to the expression 3.11, the delay Δ_{ECSM} of ECSM computation in sequential mode could be evaluated by the following expression:

$$\Delta_{ECSM} = (w - 1) \times (25 \times MM + 17 \times MA/S) + (11 \times \mathrm{MM} + 10 \times MA/S) \tag{3.16}$$

Parallelization is an optimization technique used to speed up the computations of ECSM by reducing its critical path delay Δ_{ECSM}. In the following, the impact of the parallelization possibilities on the critical path delay of ECSM is illustrated.

3.4.1 Parallelism exploration with two degrees

The intuitive choice to exploit the parallelism is to compute ECPA and ECPD simultaneously at each iteration of the MPL algorithm, using two parallel embedded processors. In other words, parallelism is introduced in the second level of the ECSM hierarchy (figure 3.1). This approach corresponds to the ECSM execution with two degrees of parallelization. The corresponding critical path delay $\Delta_{ECSM/2}$ of this approach is evaluated by expression (3.17).

$$\Delta_{ECSM/2} = (w-1) \times \text{MAX}(\Delta_{ECPA}, \Delta_{ECPD}) + \Delta_{ECPD} \qquad (3.17)$$

By substituting the critical path delays Δ_{ECPD} and Δ_{ECPA} to the expression 4.19, the critical path $\Delta_{ECSM/2}$ could be estimated by expression (3.18).

$$\Delta_{ECSM/2} = (w-1) \times (14 \times MM + 7 \times MA/S) + (11 \times \text{MM} + 10 \times MA/S) \qquad (3.18)$$

Compared to Δ_{ECSM}, the execution of ECSM with two degrees of parallelization using two parallel processors allows to reduce the ECSM critical path delay by two times and saves $11 \times MM$+$7 \times MA/S$ at each iteration of the MPL algorithm. In this approach, the processor responsible for ECPD computation stays inactive for $3 \times MM$ until the second processor completes the ECPA computation, at each iteration. This idle time represents the difference between Δ_{ECPA} and Δ_{ECPD}. However, this idle time could be exploited by ECPD processor for reading the scalar value bit $g_{(i+1)}$ of the next iteration of MPL algorithm.

3.4.2 Parallelism exploration with four degrees

The parallelism could be introduced not only in the second level but also in the third level of the ECSM hierarchy (figure 3.1), between the required finite field operations of ECPD and ECPA formulas. From expressions 3.12 and 3.13, we note that ECPD and ECPA could be performed by two processors for each point operation. This approach requires the integration of four embedded processors to compute ECSM with four degrees of parallelization. Our study is based on partitioning the computation of ECPD and ECPA on two parallel tasks (Tsk_i) for each point operation. A task regroups a set of finite field operations.

From equation system (3.12) of ECPD coordinates, we note that the computations of the results ($X_{b(i)}$, $Y_{b(i)}$) and the variable S are independent of the results ($Z_{b(i)}, Z^2_{b(i)}, Z^3_{b(i)}$) and the variable M, respectively. Hence, we propose to split ECPD process into two tasks (Tks_0, Tks_1) as it is given in table 3.2.

Table 3. 2 ECPD partitioning into two tasks

Steps	ECPD tasks execution	
	$\boldsymbol{Tks_0}$	$\boldsymbol{Tks_1}$
1	$(\mathrm{Z}^2_{b(i+1)})^2$	$3\mathrm{X}^2_{b(i+1)}$
2	$\mathrm{a} \times (\mathrm{Z}^2_{b(i+1)})^2$	$Y^2_{b(i+1)}$
3	$M = 3X^2_{b(i+1)} + a \times (Z^2_{b(i+1)})^2$	$4 \times Y^2_{b(i+1)}$
4	M^2	$\mathrm{S} = \mathrm{X}_{b(i+1)} \times (4 \times Y^2_{b(i+1)})$
5	$T = 2 \times 4\mathrm{Y}^2_{b(i+1)} \times \mathrm{Y}^2_{b(i+1)}$	$Z_{b(i)} = 2 \times Y_{b(i+1)} \times Z_{b(i+1)}$
6	$X_{b(i)} = M^2 - 2S$	$Z^2_{b(i)} = Z_{b(i)} \times Z_{b(i)}$
7	$Y_{b(i)} = M \times (S - X_{b(i)}) - T$	$Z^3_{b(i)} = Z^2_{b(i)} \times Z_{b(i)}$

The execution of ECPD based on two processors allows reducing its critical path by saving $5 \times MM + 5 \times MA/S$ for single computation. The delay $\Delta_{ECPD/2}$ of ECPD using two processors based on the proposed tasks partitioning could be evaluated as follows, where Δ_{Tks_i} is the critical path delay of Tsk_i.

$$\Delta_{ECPD/2} = MAX(\Delta_{Tks_0}, \Delta_{Tks_1}) = 6 \times \Delta_{MM} + 5 \times \Delta_{MA/S}. \qquad (3.19)$$

On the other hand, from equation system (3.13) of ECPA coordinates, the computations of S and the results ($X_{1-b(i)}, Y_{1-b(i)}$) are independent of the computations of the variable U and the results ($Z_{1-b(i)}, Z^2_{1-b(i)}, Z^3_{1-b(i)}$), respectively. Also, the computations of the variables V and U are independent. Therefore, we propose to split ECPA process into two tasks (Tks_2, Tks_3) as it is shown in table 3.3

Table 3. 3 ECPA partitioning into two tasks

Steps	ECPA tasks execution	
	Tsk_2	Tsk_3
1	$S1 = \mathrm{Y}_{b(i+1)} \times \mathrm{Z}^3_{1-b(i+1)}$	$\mathrm{U1} = \mathrm{X}_{b(i+1)} \times \mathrm{Z}^2_{1-b(i+1)}$
2	$S2 = \mathrm{Y}_{1-b(i+1)} \times \mathrm{Z}^3_{b(i+1)}$	$\mathrm{U2} = \mathrm{X}_{1-b(i+1)} \times \mathrm{Z}^2_{b(i+1)}$
3	$S = S2 - S1$	$U = U2 - U1$
4	S^2	U^2
5	$V = U1 \times U^2$	U^3
6	$2 \times V$	$S^2 - U^3$

7	$G = \mathrm{S1} \times \mathrm{U}^3$	$Z_{b(i+1)} \times Z_{1-b(i+1)}$
8	$X_{1-b(i)} = (S^2 - U^3) - 2V$	$Z_{1-b(i)} = U \times (Z_{b(i+1)} \times Z_{1-b(i+1)})$
9	$Y_{1-b(i)} = S \times (V - X_2) - G$	$Z^2_{1-b(i)} = Z_{1-b(i)} \times Z_{1-b(i)}$
10		$Z^3_{1-b(i)} = Z^2_{1-b(i)} \times Z_{1-b(i)}$

The execution of ECPA using two processors allows reducing its critical path by saving $6 \times MM + 5 \times MA/S$ for single computation. The critical path delay $\Delta_{ECPA/2}$ of ECPA based on the proposed tasks partitioning could be evaluated by:

$$\Delta_{ECPA/2} = MAX\left(\Delta_{Tks_2}, \Delta_{Tks_3}\right) = 8 \times \Delta_{MM} + 2 \times \Delta_{MA/S}. \quad (3.20)$$

By substituting the critical path delays $\Delta_{ECPD/2}$ and $\Delta_{ECPA/2}$ to the expression 3.17, the critical path delay $\Delta_{ECSM/4}$ of ECSM computation with four degrees of parallelization could be estimated by expression (3.21).

$$\Delta_{ECSM/4} = (w-1) \times (8 \times \Delta_{MM} + 2 \times \Delta_{MA/S}) + (6 \times \Delta_{MM} + 5 \times \Delta_{MA/S}). \quad (3.21)$$

Compared to Δ_{ECSM}, the execution of ECSM with four degrees of parallelization using four parallel processors reduces the ECSM critical path by more than three times and saves $17 \times MM + 15 \times MA/S$ at each iteration of MPL algorithm.

3.4.3 Parallelism exploration with six degrees

From tables 3.2 and 3.3, we can observe that there are independent operations in each task that could be executed in parallel. Thus, we propose to add Tsk_4 to table 3.2 and Tsk_5 to table 3.3 in order to perform the required finite field operations of ECPD and ECPA into three tasks for each point operation, based on the following notes:

- The parallel computation of the multiplications $3X^2_{b(i+1)}$ and $Y^2_{b(i+1)}$ saves one MM in Tsk_1. Hence, we propose to move the first multiplication to Tsk_4.

- The results $(Z_{b(i)}, Z^2_{b(i)}, Z^3_{b(i)})$ obtained from the step 5 to 7 of Tsk_1, could be carried out simultaneously with the other operations. Therefore, these results can be transferred to Tsk_4.
- The saved delay of step 4 in Tsk_1 could be used to compute the variable T, performed initially at step 5 of Tsk_0.
- The parallel computation of the variables $U1$ and $U2$ saves one Δ_{MM} in Tsk_3. Hence, we propose to move $U1$ computation to Tsk_5.
- The execution of the operations defined from steps 7 to 10 in Tsk_3 are performed independently of the other operations. Thus, these operations could be transferred also to Tsk_5.
- The saved step 6 in Tsk_3 could be exploited for computing the variable G, obtained initially at step 7 in Tsk_2.

In this way, the parallelism is explored within ECPD and ECPA computations by using three embedded processors for each operation. This approach requires the integration of six embedded processors to compute ECSM with six degrees of parallelization. Tables 3.4 and 3.5 present the proposed partitioning of ECPD and ECPA into three tasks, respectively.

Table 3. 4 ECPD partitioning into three tasks

Steps	ECPD tasks execution		
	Tsk_0	Tsk_1	Tsk_4
1	$(Z^2_{b(i+1)})^2$	$Y^2_{b(i+1)}$	$3X^2_{b(i+1)}$
2	$a \times (Z^2_{b(i+1)})^2$	$S = 4 \times X_{b(i+1)} \times Y^2_{b(i+1)})$	$Y_{b(i+1)} \times Z_{b(i+1)}$
3	$M = 3X^2_{b(i+1)} + a \times (Z^2_{b(i+1)})^2$	$2 \times S$	$Z_{b(i)} = 2 \times (Y_{b(i+1)} \times Z_{b(i+1)})$
4	M^2	$T = 8 \times Y^4_{b(i+1)}$	$Z^2_{b(i)} = Z_{b(i)} \times Z_{b(i)}$
5	$X_{b(i)} = M^2 - 2S$		$Z^3_{b(i)} = Z^2_{b(i)} \times Z_{b(i)}$
6	$Y_{b(i)} = M \times (S - X_{b(i)}) - T$		

Table 3. 5 ECPA partitioning into three tasks

Steps	ECPA tasks execution		
	Tsk_2	Tsk_3	Tsk_5
1	$S1 = \mathrm{Y}_{b(i+1)} \times \mathrm{Z}^3_{1-b(i+1)}$	$\mathrm{U2} = \mathrm{X}_{1-b(i+1)} \times \mathrm{Z}^2_{b(i+1)}$	$\mathrm{U1} = \mathrm{X}_{b(i+1)} \times \mathrm{Z}^2_{1-b(i+1)}$
2	$S2 = \mathrm{Y}_1 \times \mathrm{Z}^3_{b(i+1)}$	$U = U2 - U1$	$Z_{b(i+1)} \times Z_{1-b(i+1)}$
3	$S = S2 - S1$	U^2	$Z_{1-b(i)} = U \times (Z_{b(i+1)} \times Z_{1-b(i+1)})$
4	S^2	U^3	$Z^2_{1-b(i)} = Z_{1-b(i)} \times Z_{1-b(i)}$
5	$V = U1 \times U^2$	$G = \mathrm{S1} \times \mathrm{U}^3$	$Z^3_{1-b(i)} = Z^2_{1-b(i)} \times Z_{1-b(i)}$
6	$2 \times V$	$S^2 - U^3$	
7	$X_{1-b(i)} = (S^2 - U^3) - 2V$		
8	$Y_{1-b(i)} = S \times (V - X_{1-b(i)}) - G$		

The critical paths $\Delta_{ECPD/3}$ and $\Delta_{ECPA/3}$ of ECPD and ECPA computations, respectively, could be evaluated by the following expressions.

$$\Delta_{ECPD/3} = MAX(\Delta_{Tks_0}, \Delta_{Tks_1}, \Delta_{Tks_4}) = 4 \times \Delta_{MM} + 4 \times \Delta_{MA/S}. \quad (3.22)$$

$$\Delta_{ECPA/3} = MAX(\Delta_{Tks_2}, \Delta_{Tks_3}, \Delta_{Tks_5}) = 5 \times \Delta_{MM} + 5 \times \Delta_{MA/S}. \quad (3.23)$$

By substituting the critical path delays $\Delta_{ECPD/3}$ and $\Delta_{ECPA/3}$ in the expression 3.17, the critical path delay $\Delta_{ECSM/6}$ of ECSM computation with six degrees of parallelization could be estimated by expression (3.24).

$$\Delta_{ECSM/6} = (w-1) \times (5 \times \Delta_{MM} + 5 \times \Delta_{MA/S}) + (4 \times \Delta_{MM} + 4 \times \Delta_{MA/S}) \quad (3.24)$$

Compared to Δ_{ECSM}, the execution of ECSM with six degrees of parallelization using six parallel processors reduces the ECSM critical path by more than five times and saves $20 \times MM$+$12 \times MA/S$ at each iteration of the MPL algorithm.

3.4.4 Parallelism exploration with three degrees

The integration of four and six embedded processors in single architecture can provoke an overhead of hardware resources utilization for low-cost FPGA circuits. Therefore, we propose to perform ECSM with three degrees of parallelization using three processors. In the proposed approach, the independent finite field operations required for ECPD and ECPA are performed in parallel based on tables 3.4 and 3.5. While, the coordinates of ECPD and ECPA resulting points of MPL algorithm are obtained sequentially. The critical path delay $\Delta_{ECSM/3}$ of this approach could be estimated by substituting the critical path delays $\Delta_{ECPD/3}$ and $\Delta_{ECPA/3}$ to the expression 3.11. This latter represents the critical path delay estimation of sequential MPL algorithm execution. $\Delta_{ECSM/3}$ is given as follows:

$$\Delta_{ECSM/3} = (w-1) \times (9 \times \Delta_{MM} + 9 \times \Delta_{MA/S}) + (4 \times \Delta_{MM} + 4 \times \Delta_{MA/S}) \qquad \textit{(3.25)}$$

Compared to Δ_{ECSM}, this approach allows to reduce the ECSM critical path by three times and saves $14 \times MM + 8 \times MA/S$ at each iteration of the MPL algorithm.

In this approach, ECPA is not performed until the computation of coordinate $Y_{b(i)}$ of ECPD is done. However, the processors responsible for performing Tsk(1) and Tsk(4) of table 3.4 are free in steps 5 and 6 . Thus, we propose to exploit these idle times for performing a set of ECPA finite field operations, based on the following:

- The idle delay of Tsk(1) in table 3.4 could be used for the computation of the variables S1 and U2 obtained initially from step 1 of Tsk(2) and Tsk(3) in table 3.5, respectively.

- The variables U1 obtained initially from the step 1 of Tsk(5) in table 3.5 could be carried out in the idle time of Tsk(4) in table 3.4.

- The computation of the variable V obtained initially from the step 5 of Tsk(2) in table 3.5. could be moved to Tsk(4) in table 3.4 after computing Z1-b(i).

Table 3.6 presents the proposed interlacing approach for the execution of ECPD followed by ECPA into three tasks.

Table 3. 6 ECPD and ECPA interlacing

Steps	ECPD followed by ECPA computations coordinates steps			
	Tsk(0)	Tsk(1)	Tsk(4)	
1	$(Z^2_{b(i+1)})^2$	$Y^2_{b(i+1)}$	$3X^2_{b(i+1)}$	
2	$a \times (Z^2_{b(i+1)})^2$	$S = 4 \times X_{b(i+1)} \times Y^2_{b(i+1)})$	$Y_{b(i+1)} \times Z_{b(i+1)}$	
3	$M = 3X^2_{b(i+1)} + a \times (Z^2_{b(i+1)})^2$	$2 \times S$	$Z_{b(i)} = 2 \times (Y_{b(i+1)} \times Z_{b(i+1)})$	ECPD
4	M^2	$T = 8 \times Y^4_{b(i+1)}$	$Z^2_{b(i)} = Z_{b(i)} \times Z_{b(i)}$	
5	$X_{b(i)} = M^2 - 2S$	$S1 = Y_{b(i+1)} \times Z^3_{1-b(i+1)}$	$Z^3_{b(i)} = Z^2_{b(i)} \times Z_{b(i)}$	
6	$Y_{b(i)} = M \times (S - X_{b(i)}) - T$	$U2 = X_{1-b(i+1)} \times Z^2_{b(i+1)}$	$U1 = X_{b(i+1)} \times Z^2_{1-b(i+1)}$	
7	$S2 = Y_1 \times Z^3_{b(i+1)}$	$U = U2 - U1$	$Z_{b(i+1)} \times Z_{1-b(i+1)}$	
8	$S = S2 - S1$	U^2	$Z_{1-b(i)} = U \times (Z_{b(i+1)} \times Z_{1-b(i+1)})$	
9	S^2	U^3	$V = U1 \times U^2$	ECPA
10	$2 \times V$	$S^2 - U^3$	$Z^2_{1-b(i)} = Z_{1-b(i)} \times Z_{1-b(i)}$	
11	$X_{1-b(i)} = (S^2 - U^3) - 2V$	$G = S1 \times U^3$	$Z^3_{1-b(i)} = Z^2_{1-b(i)} \times Z_{1-b(i)}$	
12	$Y_{1-b(i)} = S \times \left(V - X_{1-b(i)}\right) - G$			

The critical path delay $\Delta_{ECSM/3i}$ of ECSM based on table 3.6 could be estimated by the expression (3.26).

$$\Delta_{ECSM/3i} = MAX\left(\Delta_{Tks_0}, \Delta_{Tks_1}, \Delta_{Tks_4}\right)$$

$$= (w-1) \times (7 \times \Delta_{MM} + 9 \times \Delta_{MA/S}) + (4 \times \Delta_{MM} + 4 \times \Delta_{MA/S}) \quad (3.26)$$

Compared to $\Delta_{ECSM/3}$, interlacing the computations of ECPA and ECPD allows saving $2 \times MM$. Compared to Δ_{ECSM}, this approach allows to reduce the ECSM critical path by 3.5 times and saves $14 \times MM + 8 \times MA/S$ at each iteration of MPL algorithm.

3.5 FPGA implementations

In this section, we present the proposed parallel architectures of ECSM computation for embedded ECC on an FPGA circuit. Our main aim is to achieve the best trade-off between security, software flexibility, hardware high-speed and low area requirements. Figure 3.2 shows the considered optimizations in the ECSM hierarchy for the proposed architectures.

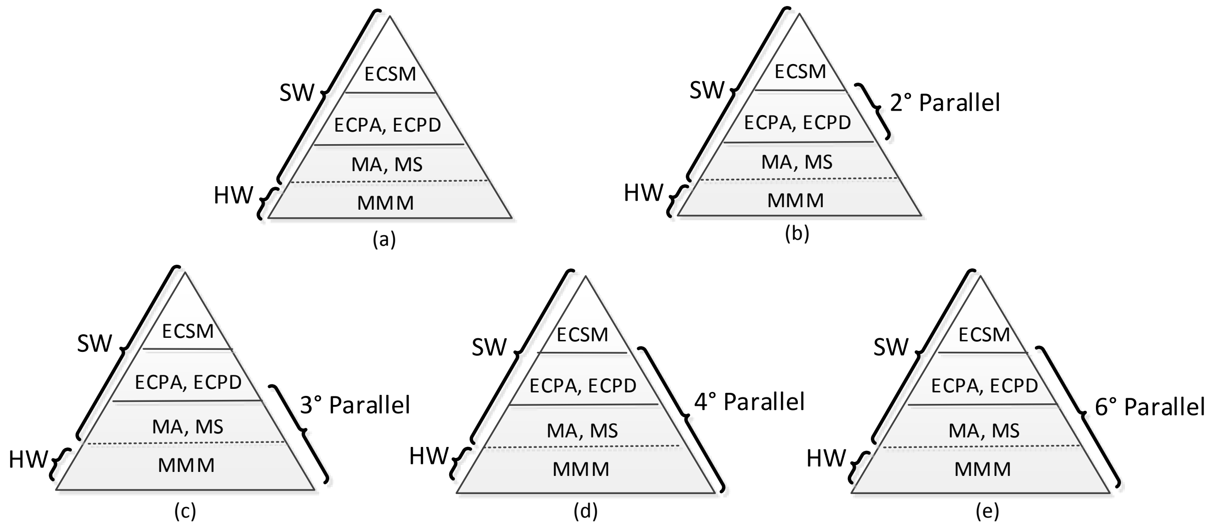


Figure 3. 2 ECSM hierarchy modeling for the proposed architectures

The Integration of Xilinx's MicroBlaze processor on FPGA increases the flexibility of our embedded cryptosystem. However, it does not provide high timing performance, because ECSM requires intensive finite field arithmetic computations. Since MMM is the critical finite field operation in ECSM execution, our SW/HW partitioning is based on the implementation of this operation within a Montgomery Modular Multiplier unit (MMMu). This dedicated Multiplier is integrated around each MicroBlaze processor to perform ECSM computation. The proposed SW/HW partitioning allows not only the execution time improvement but also the overall embedded system flexibility since ECPA, ACPD and ECSM are implemented in SW. Thus, the first approach proposed in this work (figure.3.2(a)) is a Single MicroBlaze-based SW/HW implementation (1MbSW/HW). It corresponds to the execution of ECSM in sequential mode. It is proposed mainly as a comparison model that requires minimum hardware resources. 1MbSW/HW could be suitable for low-area FPGA circuits.

The previous section illustrates that the parallelization technique allows reducing the critical path delay of ECSM computation. Therefore, we propose to exploit the inherent parallelism by integrating multiple Microblaze processors and multipliers in the same architecture. Thus, parallel SW/HW implementation approaches based on the MPSoC approach are proposed [76, 77]. Each implementation supports more optimizations compared to the previous one. They differ from the number of processors used.

The second approach (figure.3.2(b)) is Dual MicroBlaze-based SW/HW implementation (2MbSW/HW). It is based on two parallel Micoblaze processors: MB_0 and MB_1. It corresponds to the ECSM execution with two degrees of parallelism, where, the parallelism is introduced in the computations of ECPA and ECPD.

The third approach (figure.3.2(c)) is triple MicroBlaze-based SW/HW implementation (3MbSW/HW): MB0, MB1 and MB4. It presents the implementation of the ECSM algorithm with three degrees of parallelism based on table 3.6. Each processor $\mathrm{MB_i}$ ensures the control of the tasks $\mathrm{Tsk_i}$ reported in table 3.6.

The fourth approach (figure.3.2(c)) is Four MicroBlaze-based SW/HW implementation (4MbSW/HW): $\mathrm{MB_0}, \mathrm{MB_1}, \mathrm{MB_2}$ and $\mathrm{MB_3}$. It presents the implementation of the ECSM algorithm with four degrees of parallelization. Each processor $\mathrm{MB_i}$ is responsible for the tasks $\mathrm{Tsk_i}$ reported in tables 3.2 and 3.3.

The fifth approach (figure.3.2(d)) is six MicroBlaze-based SW/HW implementation (6MbSW/HW): $\mathrm{MB_0}, \mathrm{MB_1}, \mathrm{MB_2}$ $\mathrm{MB_3}, \mathrm{MB_4}$ and $\mathrm{MB_5}$. It corresponds to the implementation of ECSM with six degrees of parallelization. Each processor $\mathrm{MB_i}$ is dedicated to control the tasks $\mathrm{Tsk_i}$ reported in tables 3.4 and 3.5.

In the following, the internal hardware architectures of MMMu is presented first. Then the proposed architectures for the FPGA implementation of ECSM are detailed.

3.5.1 Montgomery Modular Multiplier unit

The hardware architecture of our MMMu is shown in figure 3.3.

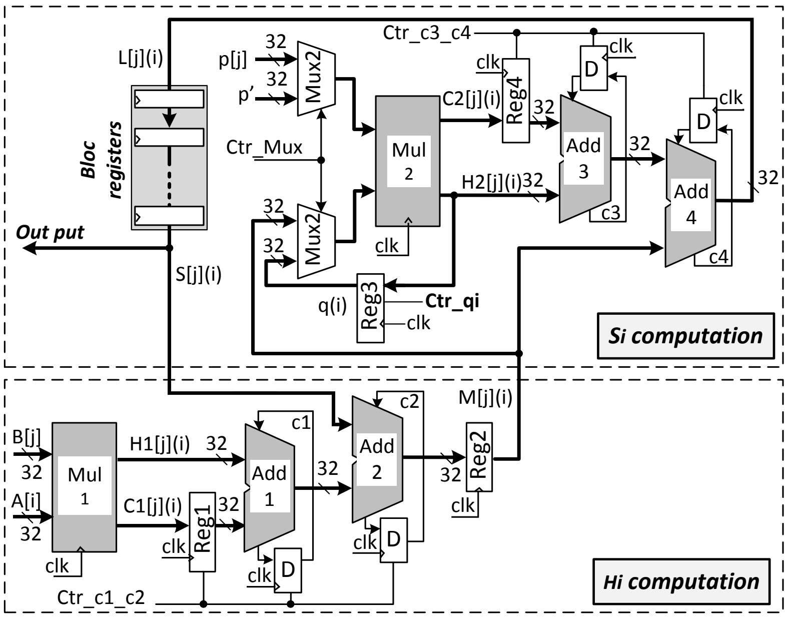


Figure 3. 3 Internal hardware architecture of MMMu

The internal architecture is designed in a single stage in order to minimize the hardware resources utilization. It contains two 32×32 bits multipliers (Mul1 and Mul2), four Carry

Propagate Adder (Add1, Add2, Add3 and Add4), four registers (Reg1, Reg2, Reg3, Reg4), four D Flip-flops, two multiplexers (Mux1, Mux2) and one block registers. The execution of the proposed MMMu requires the storage of the operands A, B and p in memories. The intermediate results digits $S[j]_i$ of MMM algorithm are stored in the block register as a queue. MMMu receives four control signals, namely, Ctr_Mux, Ctr_q_i , Ctr_c1_2 and Ctr_c3_4.

MMMu performs each iteration (i) of the MMM algorithm in three steps. In the first step, the process begins by the execution of lines 5 and 6 for computing the digit q_i. The obtained value is stored in Reg3 which is controlled by Ctr_q_i. Reg3 holds the value of q_i constant during the execution of the iterations (j) of the MMM algorithm. The second and the third steps consist of performing the operations of lines (8, 9) and (10, 11), respectively. They allow the computations of the digits $H[j]_i$ and $S[j]_i$. In order to increase the timing performance of the MMMu, we propose to compute $M[j]_i$ and $L[j]_i$ in pipeline processing. Indeed, the processes for computing both digits are interleaved by one clock cycle. The digits $M[j]_i$ are carried out by the multiplier Mul1 and the adders add1 and add2. The digits $L[j]_i$ are obtained by the multiplier Mul2 and the adders Add3 and Add4. At each iteration (j), the generated carries $(c1, c2, c3, c4)$ and the most significant digits $(C1[j]_i, C2[j]_i)$ of the multiplications results are delayed by one clock cycle and added at the next iteration (j+1). They are initialized at the beginning of each iteration (i) by the signals Ctr_c1_2 and Ctr $\_c3\_4$. The multiplier Mul2 is shared between the execution of the multiplications of lines 6 and 10 of MMM algorithm. In other words, Mul2 allows the computation of q_i and $H2[j]_i$. The selection of the operands at its inputs is ensured by the multiplexers Mux1 and Mux2 which is controlled by the signal Ctr_Mux as is shown in table 3.7

Table 3. 7 Selection of operands at the inputs of Mul2

Ctr_Mux	Selected operands		Executed operation
	Mux1 output	Mux2 output	
1	p'	$H[0]_i$	$q_i = (H[0]_i \ \times\ p'\ )\ mod\ 2^{32}$
0	$p[i]$	q_i	$(C2[j]_i, H2[j]_i) = q_i \times p[j]$

3.5.2 ECSM implementations

3.5.2.1 Single MicroBlaze-based SW/HW implementation (1MbSW/HW)

In this approach, the ECSM process is implemented without any parallelism. All ECSM abstraction levels (figure.3.2(a)) are managed in a sequential scheme by a MicroBlaze processor. Figure 3.4 presents the hardware architecture of the corresponding embedded system.

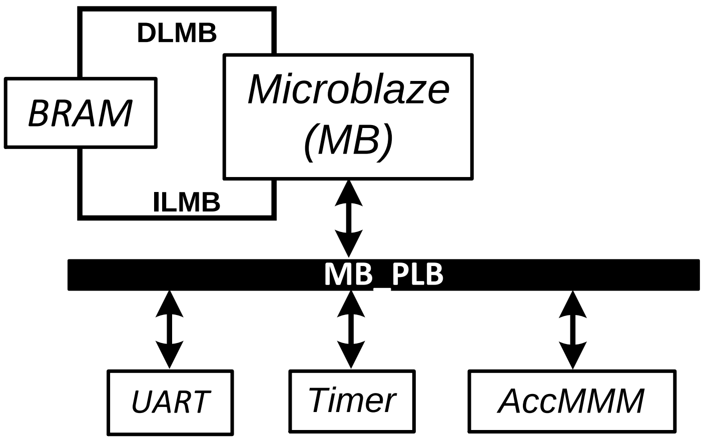


Figure 3. 4 Hardware architecture of 1MbSW/HW approach

This architecture contains the following components: MicroBlaze processor (MB), AccMMM core, BRAM memory, Local Memory Buses (ILMB, DLMB), Universal Asynchronous Receiver Transmitter (UART) to exchange the inputs and the outputs data with the serial port of the used development board. A timer to evaluate the temporal performances.

The internal architecture of the AccMMM core is shown in figure 3.5. It is composed of Xilinx’s Intellectual Property InterFace (IPIF) and User_Logic blocks. IPIF ensures data/instruction exchanging between MicroBlaze and the multiplier. User_Logic is the part that describes the logic of the circuit. The communication between the two blocks is achieved by a standard back-end interface IP InterConnect (IPIC) [78].

The IPIF is configured with three registers: Ins_reg, DataIn_reg and DataOut_reg. MicroBlaze controls our AccMMM core using a set of instruction codes through Ins_reg instruction register. DataIn_reg is used to transfer data from MicroBlaze to the internal architecture of the AccMMM core. The digits of MMM result are transferred to MicroBlaze through DataOut_reg.

The User_Logic is composed of three units, Memory Unit (MU), Control Unit (CU) and our MMMU. MU is a local memory used to store the modulus p, the operands A and B and the digits of the MMM result. CU retrieves the instructions from Ins_reg and manages MU and MMMu.

The data/instruction exchanging between MicroBlaze and AccMMM core is carried out through PLB bus. After system initialization, MicroBlaze sends the digits of the modulus p, the size of operands w and Montgomery's constants to AccMMM core. At each MMM execution, MicroBlaze sends $2 \times (\mathrm{e}+1)$ digits of the operands A and B. Then, the MMMu performs MMM algorithm and sends $(\mathrm{e}+1)$ digits of the result to the processor.

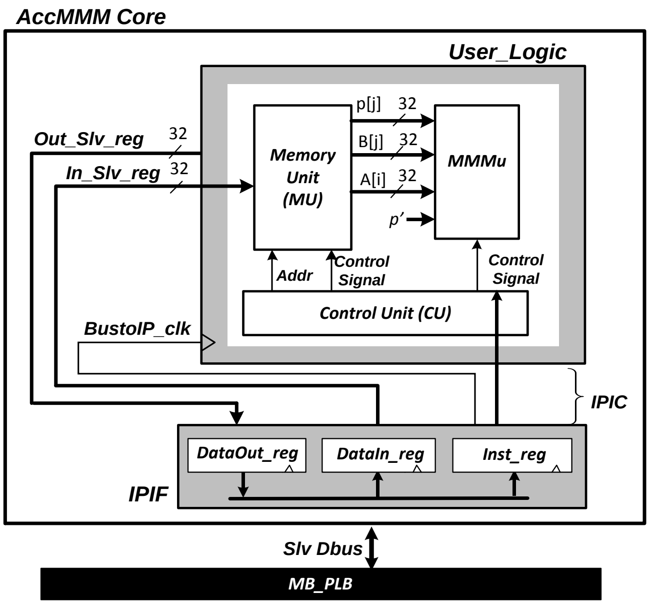


Figure 3. 5 Hardware architecture of our AccMMM core

3.5.2.2 Dual MicroBlaze-based SW/HW implementation (2MbSW/HW)

In order to benefit from the inherent parallelism between ECPD and ECPA of MPL algorithm, we proposed to integrate two MicroBlaze processors in single architecture as MPSoPC. As result, MPL algorithm is executed in parallel mode with two degrees of parallelism. Figure 3.6 shows the proposed Master-Slave architecture [79]. It is composed of two MicroBlaze processors: MB_0 (Master) and MB_1 (slave), where the ECPD and the ECPA are performed in parallel by MB_0 and MB_1, respectively. The processors are connected through FLS Bus (FSL_0_1, FSL_1_0).

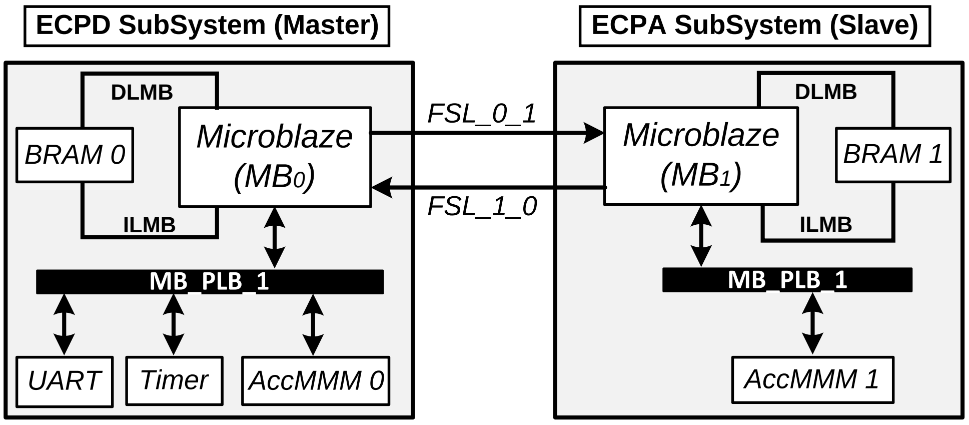


Figure 3. 6 Hardware architecture of 2MbSW/HW approach.

The control of the ECSM computations between both processors is ensured using some instructions. Their codes and their descriptions are presented in table 3.8.

Table 3. 8 Instructions Codes

Code	Instruction	Description
0x00000001	Sync_Request	Mb_0 to Mb_1 synchronization request
0x00000010	Sync_Response	Mb_1 to Mb_0 synchronization response
0x00000011	End_Loop	End of the loop execution of MPL algorithm
0x00000100	Chv_to_Aff	Chudnovsky to Affine conversion

Four steps are required to perform ECSM algorithm in this approach, namely, processors synchronization, affine to Chudnovsky conversion, loop execution and Chudnovsky to affine conversion. Figure 3.7 presents the sequence diagram of tasks partitioning between the two processors for performing ECSM with two degrees of parallelism.

Initially, the system needs to synchronize between the processors. For that, MB_0 sends first Sync_Request instruction to MB_1 and waits the reply with the Sync_Response instruction. Next, MB_0 executes:

- The conversion of the point P from Affine to Chudnovsky coordinates system according to expression (3.4).
- The initializations defined by the lines 1 and 2 of MPL algorithm.

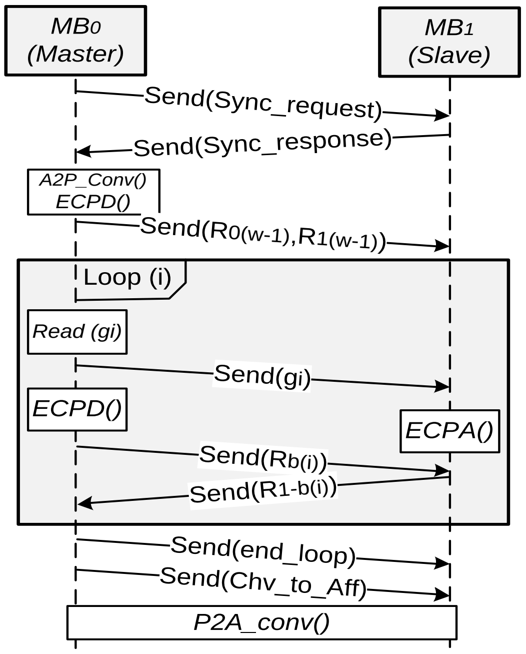


Figure 3. 7 Sequence diagram for parallel ECSM implementation with two processors

Then MB_0 sends $R0_{(w-1)}$ and $R1_{(w-1)}$ to MB_1. With a perfect coordination, the processors share between them the resulting points $R_{1-b(i)}$ and $R_{b(i)}$ of the performed operation at each iteration (i). The process of the main loop (i) of MPL algorithm starts first by shifting the scalar g and reading the g_i value. This operation is executed by MB_0. When g_i is done, MB_0 sends its values to MB_1. The master processor executes ECPD() and sends the point $R_{b(i)}$ to MB_1. In parallel, the slave processor executes ECPA() and sends the point $R_{1-b(i)}$ to MB_0. Since ECPA is more complex than ECPD, MB_0waits until MB_1 finishes its execution then launches the next iteration execution $(i-1)$. When the loop (i) ends, MB_0 sends to MB_1 the End_Loop instruction. Finally, MB_0 controls the conversion of the resulting point $R0_{(0)}$ to affine coordinates system according to the expressions (3.5). The required Mexp for the conversion is executed in parallel mode by both processors using Chv_to_Aff instruction.

3.5.2.3 Triple MicroBlaze-based SW/HW implementation (3MbSW/HW)

This approach corresponds to the implementation of ECSM algorithm with three degrees of parallelism, where the parallelism is introduced in the third abstraction level of ECSM hierarchy (see figure 3.1). The hardware architecture of this approach is presented in figure 3.8.

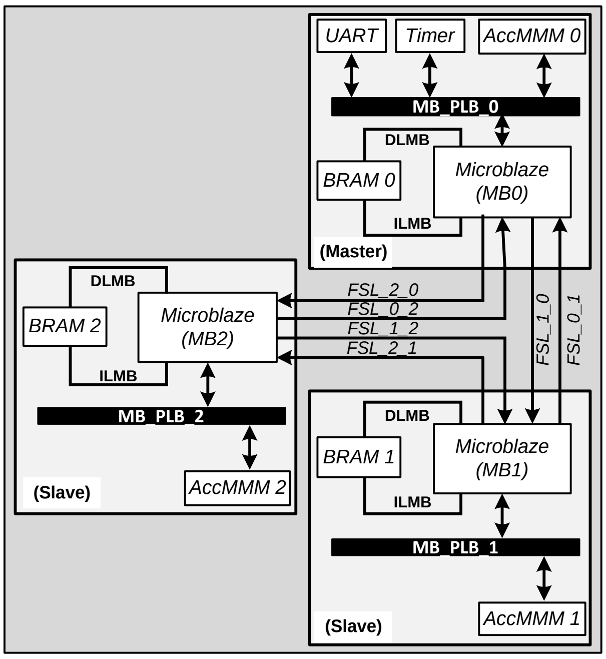


Figure 3. 8 Hardware architecture of 3MbSW/HW approach

This design represents a full mesh network topology [79] with three MicroBlaze processors: MB_0, MB_1, and MB_2. In this kind of topologies, each processor is connected directly to the other processors. The proposed tasks scheduling between the processors is based on our study reported in table 3.6. Each processor MB_i is dedicated to the execution of the Tsk(i). In order to accelerate the MMM execution, single AccMMM core is integrated around each processor (AccMMM0, AccMMM1, AccMMM2). This approach requires the same steps as the previous approach for ECSM execution. The difference lies in the following points:

- The ECPD and ECPA resulting points of MPL algorithm are obtained sequentially, at each iteration.
- The three integrated processors intervene for performing the required finite field operations of ECPD and ECPA in parallel.

The synchronization between the processors is required at the system initialization. Therefore, the master processor MB_0 sends Sync_Request instruction to the slaves (MB_1, MB_2) and waits their replies with Sync_Response instruction. The conversion from affine to Chudnovsky is ensured by MB_0. The required ECPD for initialization step of MPL algorithm is carried out by the three processors. The process of the main loop (i) requires the synchronization between the processors at the beginning of each iteration. After reading g_i value by MB_0, the three processors coordinate between them for performing ECPD followed by ECPA to obtain the points $R_{b(i)}$ and $R_{1-b(i)}$. The conversion of the resulting point $R0_{(0)}$ to affine coordinates system is performed by MB_0. The required Mexp for this conversion is executed in parallel mode by MB_0 and MB_1.

3.5.2.4 Four MicroBlaze-based SW/HW implementation (4MbSW/HW)

This approach corresponds to the MPSoPC implementation of ECSM algorithm with four degrees of parallelization. The hardware architecture of this approach is presented in figure 3.9. The proposed design is based on the integration of four MicroBlaze processors: MB_0, MB_1, MB_2 and MB_3. The proposed tasks scheduling between the processors is based on our study reported in tables 3.2 and 3.4. Each processor MB_i is dedicated to the execution of the tasks Tsk_i. In order to accelerate the MMM execution, AccMMM core is integrated around each processor. This architecture is composed by two Master/Slave SubSystems: ECPD and ECPA SubSystems. The first is represented by the processors MB_0 and MB_1. It ensures the ECPD coordinates computations based on table 3.2. The second regroups MB_2 and MB_3. It is dedicated to ECPA coordinates computations using table 3.3. MB_0 and MB_2 are the master processors in ECPD and ECPA SubSystems, respectively. In the entire system, ECPD SubSystem is considered as the master.

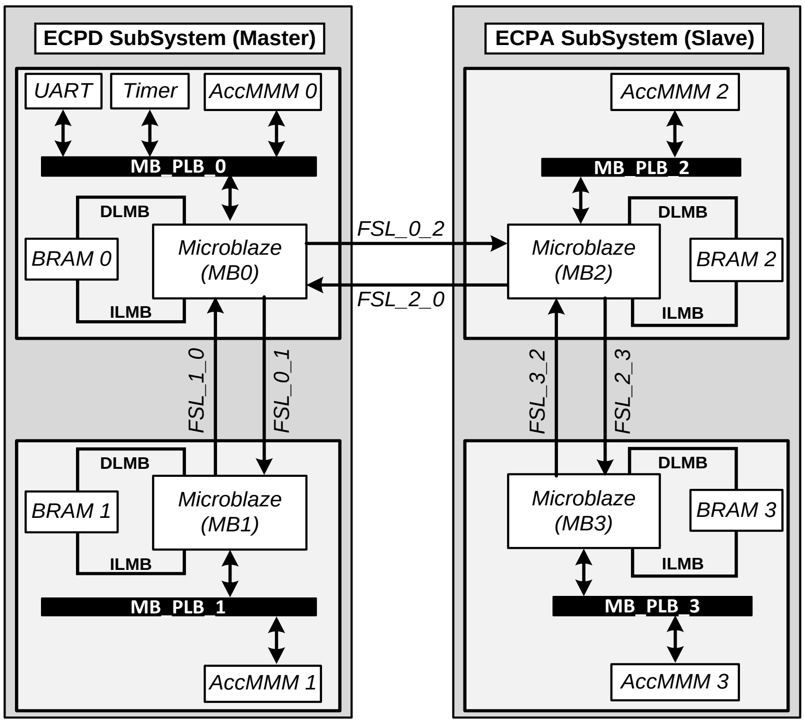


Figure 3. 9 Hardware architecture of 4MbSW/HW approach

Figure 3.10 presents the sequence diagram of tasks partitioning between the four processors for performing ECSM.

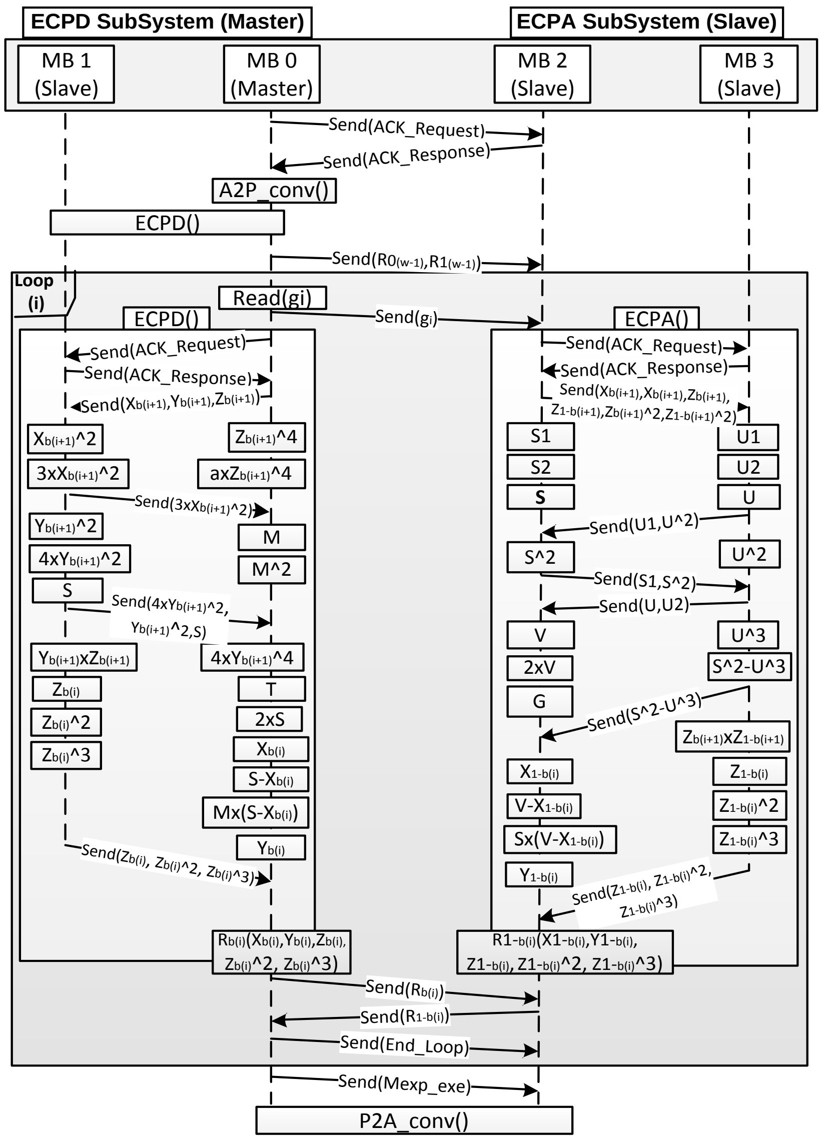


Figure 3. 10 Sequence diagram for parallel ECSM implementation with four processors

In this approach, the process of ECSM can be considered as identical to the second approach. The functions executed by MB_0 and MB_1 of 2MbSW/HW are partitioned and performed in the present approach by ECPD and ECPA SubSystems, respectively. The synchronization between both Subsystems is based on using the instructions defined in table 3.8. At the top level, the data and the instruction transfer is carried out between the masters of each Subsystem. Initially, the system begins by the synchronization between ECPD and ECPA Subsystems. Indeed, MB_0 sends Syn_Request instruction to MB_2 and waits the instruction response Sync_response of MB_2. The conversion from affine to Chudnovsky systems is carried out by $\mathrm{MB_0}$. The ECPD of line 2 of MPL algorithm is performed by ECPD Subsystem. The points $\mathrm{R_{0(w-1)}} = \mathrm{P}$ and $\mathrm{R_{1(w-1)}} = 2 \times \mathrm{P}$ are transmitted to $\mathrm{MB_2}$. At each iteration (i), $\mathrm{MB_0}$ starts by shifting the scalar g, reading the $\mathrm{g_i}$ value and sending its value to $\mathrm{MB_2}$ (Task 3). The operations of lines 5 and 6 are executed in parallel by ECPD and ECPA Subsystems, respectively. At the end of the loop (i), $\mathrm{MB_0}$ sends end_loop instruction to $\mathrm{MB_2}$. Finally, $\mathrm{MB_0}$ controls the conversion of the resulting point $\mathrm{R0_{(0)}}$ to affine coordinates system. The required Mexp execution is carried out in parallel scheme by $\mathrm{MB_0}$ and $\mathrm{MB_2}$ using the instruction Chv_to_Aff.

The execution details of ECPD and ECPA points computations, according to the steps of tables 3.2 and 3.3, respectively, are described as follows:

1. The system needs the synchronization between the processors of each Subsystem before computing the coordinates of both points. Therefore, the masters MB_0 and MB_2 send the Sync_Request instruction to the slaves MB_1 and MB_3, respectively. Then, they wait the instruction response Sync_Response.

2. In order to execute the ECPD, MB_0 sends first the coordinates $(X_{b_{(i+1)}}, Y_{b_{(i+1)}}, Z_{b_{(i+1)}})$ of the point $R_{b(i+1)}$ to MB_1. MB_0 and MB_1 execute in parallel the steps 1 and 2 of Tsk_0 and Tsk_1, respectively. MB_1 sends the result of $3 \times X^2_{b_{(i+1)}}$ to MB_0. Then both processors perform the steps 3 and 4 to get the values of M^2 and S. MB_1 sends the results of $Y^2_{b_{(i+1)}}$, $4 \times Y^2_{b_{(i+1)}}$ and S to MB_0. The next operations consist of the execution of the steps 5, 6 and 7 of Tsk_0 and Tsk_1. The coordinates $(X_{b_{(i)}}, Y_{b_{(i)}})$ of the resulting point $R_{b(i)}$ are computed by MB_0. While, the values of the cordinates $(Z_{b_{(i)}}, Z^2_{b_{(i)}}, Z^3_{b_{(i)}})$ are obtained from MB_1.

3. For performing ECPA, MB_2 sends first to MB_3 the coordinates $(X_{b_{(i+1)}}, Z_{b_{(i+1)}}, Z^2_{b_{(i+1)}})$ and $(X_{1-b_{(i+1)}}, Z_{1-b_{(i+1)}}, Z^2_{1-b_{(i+1)}})$ of the points $R_{b(i+1)}$ and $R_{1-b(i+1)}$, respectively. MB_2 executes the steps 1 to 4 of Tsk_2 and sends the results of $S1$ and S^2 to MB_3. In parallel, MB_3 performs the steps 1 to 4 of Tsk_3 and sends the results of $U1$ and U^2 to MB_2. Finally, MB_2 computes the coordinates $(X_{1-b_{(i)}}, Y_{1-b_{(i)}})$ of the resulting point $R_{1-b(i)}$ by performing the steps 5 to 9 of Tsk_2.

Simultaneously, MB_3 executes the steps 5 to 10 of Tsk_3 to obtain the coordinates $(Z_{1-b_{(i)}}, Z^2_{1-b_{(i)}}, Z^3_{1-b_{(i)}})$ of the resulting point $R_{1-b(i)}$.

3.5.3.5 Six MicroBlaze-based SW/HW implementation (6MbSW/HW)

This approach presents the MPSoPC implementation of ECSM algorithm with six degrees of parallelism. It this approach, six processors MB_0, MB_1, MB_2, MB_3, $MB_4$and MB_5 are integrated. The proposed tasks scheduling between the processors for computing ECPD and ECPA is based on our study reported in tables 3.4 and 3.5, respectively. Each processor MB_i ensures the execution of the corresponding task Tsk_i. The MMM computations are performed using six AccMMM cores which are integrated around the processors. The hardware architecture of this approach is shown in figure 3.11. This architecture is composed by two Master/Slave Subsystems: ECPD and ECPA Subsystems. Each Subsystem consists of three MicroBlaze processors. It uses a completely meshed network topology. MB_0 and MB_2 act as masters in ECPD and ECPA SubSystems, respectively. At the system top level, ECPD Subsystem is considered as the master Subsystem.

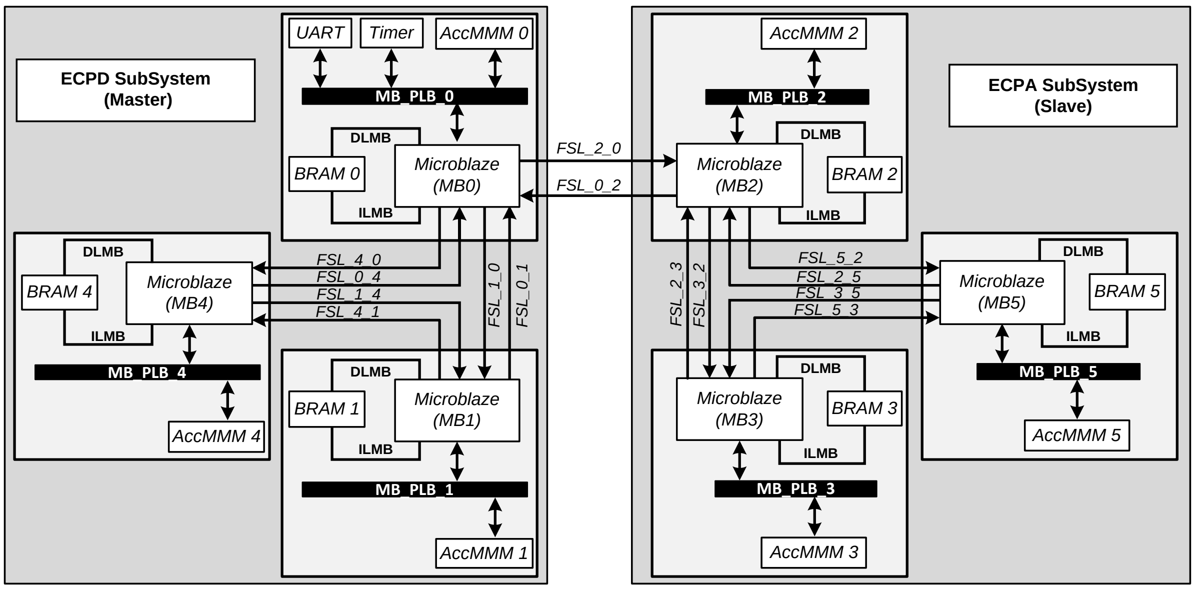


Figure 3. 11 Hardware architecture of 6MbSW/HW approach

In this approach, the steps for performing ECSM algorithm are the same as the previous approach. The difference is in the execution process of each ECPD and ECPA. Since the synchronization between the processors of each Subsystem is required for both points coordinates computations, the masters MB_0 and MB_2 send the Sync_Request instruction to the slaves (MB_1, MB_4) and (MB_3, MB_5), respectively. Then, they wait the instruction response Sync_Response of the slaves. Figure 3.12 presents the sequence diagram of tasks partitioning between the six processors for performing ECSM algorithm.

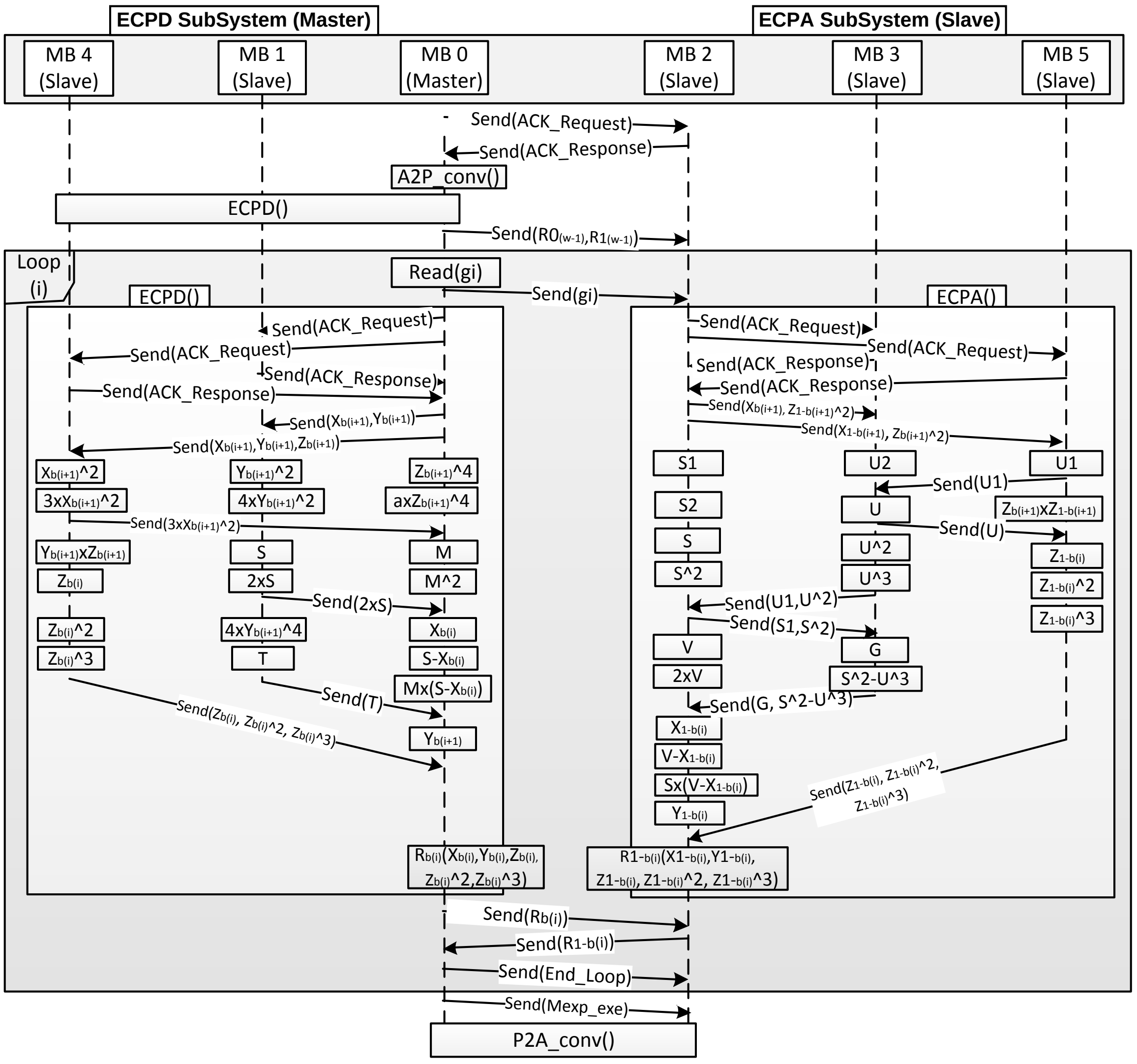


Figure 3. 12 Sequence diagram for parallel ECSM implementation with six processors

For the execution of ECPD, MB_0 sends first the coordinates $(X_{b_{(i+1)}}, Y_{b_{(i+1)}})$ and $(X_{b_{(i+1)}}, Y_{b_{(i+1)}}, Z_{b_{(i+1)}})$ of the point $R_{b(i+1)}$ to MB_1 and MB_4, respectively. The three processors perform in parallel the steps 1 and 2 of ECPD coordinates computations. MB_4 sends the result of $3 \times X^2_{b_{(i+1)}}$ to MB_0 and executes the steps 3 to 5 of Tsk_4. Simultaneously, MB_0 and MB_1 perform the steps 3 to 6 and 3 to 4 of ECPD coordinates computations, respectively. Once MB_0

computes the coordinates $(X_{b_{(i)}}, Y_{b_{(i)}})$ of the resulting point $R_{b(i)}$, it recovers the coordinates $(Z_{b_{(i)}}, Z^2_{b_{(i)}}, Z^3_{b_{(i)}})$ of the resulting point $R_{b(i)}$ carried out by MB_4.

For performing ECPA, MB_2 sends the coordinates $(X_{0_{(i+1)}}, Z^2_{1_{(i+1)}})$ and $(X_{1_{(i+1)}}, Z^2_{0_{(i+1)}})$ to MB_3 and MB_5, respectively. MB_2 and MB_3 performs the steps 1 to 4 of Tsk_2 and Tsk_3, respectively. In parallel, MB_5 executes the step 1 of Tsk_5 to obtain $U1$ and sends the obtained value to MB_3. Then it performs the steps 2 to 5 of Tsk_5. The next operations consist of the execution of the steps 5 to 8 of Tsk_2 and the steps 5 to 6 of Tsk_3 by MB_2 and MB_3, respectively. The master processor computes the coordinates $(X_{1-b_{(i)}}, Y_{1-b_{(i)}})$ of the resulting point $R_{1-b(i)}$. Then it gets the coordinates $(Z_{1-b_{(i)}}, {Z_{1-b_{(i)}}}^2, {Z_{1-b_{(i)}}}^3)$ of the resulting point $R_{1-b(i)}$ from MB_5.

3.6 Implementations results and discussion

The hardware architectures of the proposed embedded systems for ECSM computation were implemented in Xilinx Virtex-5 XC5VLX50T Genesys development board [80] using XPS 13.2 (Xilinx Platform Studio) environment. Our AccMMM core was coded in VHDL language, using Xilinx ISE design suite 13.2. The DSP48E cores and the RAM blocks of AccMMM core were generated by Core Generator tool of ISE. To guarantee the correct behavior of the integration of our AccMMM core in the proposed embedded systems, functional simulations and comparisons with mathematical models were made respectively using ModelSim SE 64 10.0c and Eclipse IDE for Java tools.

Our main goal is to propose an efficient implementation of ECSM computation that achieves the best trade-off between flexibility, security, speed and area. In the following, we present the performance of the proposed ECSM implementations in terms of area occupation and timing execution. The results are given for two different security-levels with bit-lengths of 256-bit and 521-bit.

3.6.1 Occupied area

Table 3.9 shows the required hardware resources occupied by our embedded systems on the targeted FPGA circuit. These results are reported in terms of occupied slices, of selected block RAMs and of DSP48E cores [37]. The collected results show that each design requires more hardware resources compared to the previous one. The growth in area was expected since that this parameter depends on the number of the integrated MicroBlaze processors and AccMMM cores. Compared to 1MbSW/HW, the occupied area increases approximately by two, three, four and six times in 2MbSW/HW, 3MbSW/HW, 4MbSW/HW and 6MbSW/HW approaches, respectively.

Table 3. 9 Hardware resources requirements of our implementations

Approach	Security-levels (bits)	Occupied Slices	Block RAMs	DSP48E Cores
1MbSW/HW	256	1474	11	8
	521	1548	12	8
2MbSW/HW	256	2739	22	16
	521	2895	24	16
3MbSW/HW	256	3727	33	24
	521	3949	36	24
4MbSW/HW	256	4737	44	32
	521	4819	48	32
6MbSW/HW	256	6259	66	48
	521	6533	72	48

We have implemented the basic system without AccMMM core in order to evaluate the cost of the MMM accelerator. This latter is presented in table 3.10.

Table 3. 10 Hardware resources requirements AccMMM core

Data-path (bits)	Occupied Slices	Block RAMs	DSP48E Cores
256	434	7	8
521	508	8	8

3.6.2 Timing execution

The temporal performances of the AccMMM core to perform 256-bit and 521-bit MMM using are shown in table 3.11.

Table 3. 11 Temporal performances of AccMMM core

Data-path (bits)	F (MHZ)	Δ_{MMM} (μs)
256	100	5.4
521	100	10.7

Table 3.12 presents the execution time (t_{ECSM}) to complete a single ECSM computation. It includes:

- The conversion of the coordinates $(\mathrm{x_P}, \mathrm{y_P})$ of the point P to $(\mathrm{X_P}, \mathrm{Y_P}, \mathrm{Z_P})$.
- The execution of ECSM using MPL algorithm in Chudnovsky system.
- The conversion of the coordinates $(\mathrm{X_C}, \mathrm{Y_C}, \mathrm{Z_C})$ of the resulting point C to affine system $(\mathrm{x_C}, \mathrm{y_C})$.

t_{ECSM} is computed by the multiplication of the necessary Clock Cycles Count (CCC) with the clock period $t_{clk} = 1/f_{max}$. f_{max} is the maximum frequency of our embedded systems. CCC is collected by enabling and disabling the Timer at the beginning and at the end of ECSM process, respectively. In the proposed SW/ HW implementation approaches, the timing report showed that f_{max} that can be achieved is about 100 Mhz. This frequency is determined by the critical path of the MMMu integrated within the AccMMM core. In order to analyze the impact of the parallelization on the performance of ECSM execution, a comparison between 1MbSW/HW and the parallel implementation approaches is performed using Speedup metric [81]. This latter is calculated by expression (3.27). $t_{ECSM\_1MbSW/HW}$ is the execution time of ECSM computation in 1MbSW/HW. $t_{ECSM\_kMbSW/HW}$ represents the execution time of ECSM computation in the proposed parallel approaches.

$$SpeedUp = t_{ECSM\_nb\_MbSW/HW} / t_{ECSM\_1MbSW/HW} \tag{3.27}$$

Table 3. 12 Temporal performances of the proposed implementations

Approach	Security-level (bits)	Frequency (MHZ)	Execution Time (ms)	SpeedUp
1MbSW/HW	256	100	48.72	-
	521	100	204.50	-
2MbSW/HW	256	100	26.87	1.81
	521	100	115.00	1.78
3MbSW/HW	256	100	19.98	2.43
	521	100	81.42	2.51
4MbSW/HW	256	100	16.29	2.99
	521	100	71.57	2.86
6MbSW/HW	256	100	14.72	3.30
	521	100	65.11	3.14

Through these results, we note that the use of two parallel processors speedup the execution time by 1.81 and 1.78 times for the security levels of 256-bit and 521-bit, respectively.

Moreover, in the second approach, the speedup metric increases to 2.43 and 2.51 for 256-bit and 521-bit ECSM computation, respectively. Furthermore, the integration of four parallel MicroBlaze processors increases the speedup to 2.99 and 2.86 for 256-bit and 521-bit, respectively. Finally, the speedup achieved by six parallel processors is about 3.3 and 3.14 for the two security levels, respectively.

3.6.3 Performance comparisons with some recent works

To evaluate the merit of this work, we compare our designs with relevant works in the field. For a fair comparison, we tried to collect the works that use the same FPGA family as the one used in our work. Remind that our main aim is not the optimization of the execution time or the occupied area, but to achieve the best tradeoff between flexibility, speed, area and security.

Tables 3.13 and 3.14 show the performance comparison of our implementations with some recent FPGA-based ECSM designs for 256-bit and 521-bit, respectively. The comparisons include the execution time, the required hardware resources and the design efficiency. This latter is calculated by expression (3.28).

$$efficiency = \frac{Datapath}{Occupied\ area \times t_{ECSM}} \qquad (3.28)$$

Among the listed works, the implementations [27, 29, 30] target the recommended NIST prime curves. The reason behind this recommendation is the capability of fast modular reduction. These works provide high timing performance implementations. However, the proposed designs support only NIST curves which are not suitable for ECC applications that do not follow NIST recommendations. Compared to these works, the major advantage of our designs is the flexibility of the cryptosystem which support any arbitrary prime curves.

The proposed ECPs in [25, 27, 29, 30] present the best execution time performance but with high number of slices. In terms of efficiency, the ECPs from [29, 30] demonstrate the best performance, where, single slice treats more than 13 bits per second for 256-bit NIST curve. These results could be justified by the reason that ECSM is implemented completely in hardware. In fact, these ECPs are suitable for embedded systems on high-cost FPGA circuits that focus on the high speed without considering the area constraint, at the contrary of our designs that promote the trade-off between flexibility, speed and area. Thus, it is clearly evident that our implementations outperform all listed works for low area requirements systems. They achieve a good timing performance with less slices.

Table 3. 13 256-bit ECSM Performance comparisons with recent works

Ref	Approach	Frq (MHZ)	Area (slices)	T_{ECSM} (ms)	Efficiency (bit/s/slice)	Devices
1MbSW/HW	SW/HW	100	1474	48.72	3.56	Virtex-5
2MbSW/HW	SW/HW	100	2739	26.87	3.47	Virtex-5
3MbSW/HW	SW/HW	100	3727	19.98	3.43	Virtex-5
4MbSW/HW	SW/HW	100	4737	16.30	3.31	Virtex-5
6MbSW/HW	SW/HW	100	6259	14.72	2.77	Virtex-5
[26]	SW/HW	75	1800	166	0.38	Virtex-5
[28]	SW/HW	39.36	2237	200	0.57	Virtex-5
[25]	HW	54	11953	6.26	3.41	Virtex-4
[27]	HW	66.3	10200	6.67	3.76	Virtex-5
[30]	HW	160	8653	2.26	13.1	Virtex-5
[29]	HW	100	19540	0.98	13.36	Virtex-6

B. Baldwin, R. Goundar, R. Hamilton, M. Mark and P. William present Hw, SW and SW/HW implementations of various co-Z algorithms for ECSM on Virtex-5 [26]. The point multiplication was performed using Montgomery ladder and the Joye's double-add algorithms, while, point addition and point doubling was performed by various projective systems. The studied algorithms are implemented for three general prime field sizes, namely, 192, 256 and 521 bits. Their SW/HW architecture is based on implementing single dedicated multiplier around MicroBlaze processor through a Fast Simplex Link (FSL). The internal architecture of this multiplier is designed using carry propagate adders and follows the process described in Montgomery multiplication algorithm. The multiplier consumes between 334-904 slices. It runs at 100 MHz in the case of the 192-bit implementation and 75 MHz for both the 256-bit and 521-bit implementations. This architecture does not involve any parallelism in ECSM computation. It is designed for low-cost FPGA circuits for the application that promote the flexibility and low-area consumption. For equitable comparison, the performance of this approach was compared with our first implementation. It is clearly evident that 1MbSW/HW achieves fast time execution with less area consumption in both 256-bit and 521-bit field sizes.

J, Balasch, B. Gierlichs, K. Ja and I. Verbauwhede propose SW/HW implementations on Xilinx Virtex-5 for prime 256-bit ECSM [28]. The main aim of this work is the comprehensive analysis of many implementation options with respect to implementation cost and attack resistance on a single common platform. Thus, four SW/HW implementation approaches based on 8051 embedded microcontroller and hardware coprocessor accelerators are proposed. Each design supports more functionalities in hardware than the previous one. The first coprocessor supports only field multiplication, while the second supports all the required field arithmetic (addition, subtraction and multiplication). The authors noted that the bottleneck of these approaches was the amount of data/instruction exchanging between the processor and the coprocessors. Therefore, the third approach was proposed to solve this limitation by computing ECPA and ECPD on the coprocessor. they proposed to implement in the third approach the point arithmetic. In the fourth approach, ECSM is performed completely in the coprocessor. Moreover, they proposed in the fourth approach to implement all scalar multiplication algorithms in hardware and remove the need for software control during scalar multiplication. In their designs, the point multiplication is performed by three algorithms: left-to-right, Double-and-add-always and Montgomery Power Ladder. The point arithmetic is executed in projective coordinate system. The proposed designs consume between 2010 and 2525 slices. The first approach takes about 400ms to complete ECSM computation, while the second approach the operation requires about 120ms. The authors noted that the implementations three and four require substantially less time to process point operations and a scalar multiplication. However, they did not mention their execution times. It obviously clear that our implementations provide better delays using fewer slices.

S. Ghosh, D. Mukhopadhya and D. Roychowdhury [25] present efficient ECC hardware processor architecture. The proposed processor is composed by multiple hardware programmable GF(p) arithmetic unit (PGAU) that perform high speed finite field addition, subtraction, multiplication, inversion, and division operations. The PGAU is programmable in the sense that it supports all primes smaller than the given lengths (192, 224, 256 bits). The multiplication is based on interleaved multiplication algorithm, while the division is performed using the binary inversion algorithm. ECSM is performed by MPL algorithm and affine coordinates system. To exploit the inherent parallelism in MPL algorithm, the authors propose to integrate two PGAU in the presented ECC processor as parallel accelerators. The first one is used to compute ECPA and the second for ECPD. The proposed parallel architecture was implemented on Virtex-2, Spartan-3 and Virtex-4. The performance of Virtex-4 implementation complete parallel 256-bit ECSM execution in 6.26ms but with using 11953 slices. The required number of slices is huge and only high-cost FPGA circuits can support it. Thus, the proposed processor lacks the exportability on low-cost FPGA circuits at the contrary of our designs that offers the flexibility and the exportability on large FPGA circuits. In terms of efficiency, our designs provide the same performance but with more software coding to promote the flexibility. Finally, our designs could perform ECSM over any field size which no the case in their design.

H.Marzouqi, M. Al-Qutayri, K. Salah, D. Schinianakis and S. Stouraitis propose an exportable application-specific-instruction-set ECC processors based on redundant signed digit representation [27, 30]. The proposed ECC processors employ Karatsuba–Ofman method to achieve high throughput multiplication. They employ also an efficient modular adders and high throughput modular dividers. The proposed processors support the recommended P192 and P256 NIST curves. In fact, the processor from [30] is an optimized version of the work [27], where extensive pipelining techniques for Karatsuba–Ofman method is introduced to speed up the multiplication. Both processors were implemented in Xilinx Virtex-5 FPGA. The main advantage of these processors is their exportability on various FPGA devices from different vendors since that none of the macros or embedded blocks of specific FPGAs fabric are utilized in the internal architecture. The optimized version operates at 160 MHZ to perform a single point multiplication for NIST P256 curve in 2.26 ms and requires 8653 slices. In terms of efficiency and timing execution, the performance of this processor outperforms our designs but with sacrificing the flexibility and the area. Moreover, the processor cannot be employed in applications that targeted general prime curve because it supports only NIST curves. It lacks also the flexibility constraint since ECSM in implemented completely in hardware.

Table 3. 14 521-bit ECSM Performance comparisons with recent works

Ref	Approach	F (MHZ)	Area (slices)	T_{ECSM} (ms)	Efficiency (bit/s/slice)	Devices
1MbSW/HW	SW/HW	100	1548	204	1.64	Virtex-5
2MbSW/HW	SW/HW	100	2895	115	1.56	Virtex-5
3MbSW/HW	SW/HW	100	3949	81.4	1.85	Virtex-5
4MbSW/HW	SW/HW	100	4819	71.6	1.51	Virtex-5
6MbSW/HW	SW/HW	100	6553	65.1	1.22	Virtex-5
[26]	SW/HW	75	2205	681	0.34	Virtex-5
[28]	HW	100	19540	3.9	3.35	Virtex-6

Authors of the work [29] present high speed hardware ECC processor that supports five NIST recommended curves. ECSM is performed by MPL algorithm and projective system. The internal architecture of the proposed processor is based on novel arrangement of modular arithmetic computations and associated data transfers into parallel atomic blocks to resist side-channel attacks during scalar multiplications. The atomic blocs are designed to perform ECPA and ECPD in parallel mode. A single atomic block consists of two modular multiplications and four modular additions/subtractions. The implementation of the hardware processor on Virtex-6 FPGA runs at 100 MHz and requires between 0.30 ms (192-bit ECC) and 3.91 ms (521-bit ECC) to perform a typical scalar multiplication on NIST curves. It uses 11.2K slices (32.9K LUTs), 289 DSP48E blocks, and 128 RAMB36 blocks. The proposed processor performs ECSM significantly faster than our designs but with using huge number of RAMs and DSP48E Cores. This constraint limits the integration possibility of the proposed processor only to the boards with very high resources FPGAs, whereas our designs could be highly exploited in large FPGA circuits.

3.7 Conclusion

In this chapter, efficient MicroBlaze-based parallel architectures of ECSM computation for embedded Elliptic Curve Cryptosystem on Xilinx FPGA is presented. In the proposals, ECSM is performed by using MPL algorithm and projective coordinates system. This combination allows the performance improvement of ECSM execution not only by reducing the computation complexity of ECPA and ECPD but also by offering high parallelization possibilities in several hierarchy systems. First of all, we proposed to implement ECSM without any parallelism in order to design minimum hardware resources ECC system dedicated for small FPGAs. This approach corresponds to 1MbSW/HW approach, where the critical finite field multiplication in hardware accelerator unit (MMMu) is based on a high-radix MMM algorithm around a single MicroBlaze processor. The efficiency of the MMMu is the exploitation of the DSP48E cores available on Xilinx FPGAs. To exploit the inherent parallelism in ECSM computations, four parallel architectures based on the integration of multiple MicroBlaze processor are proposed. Each design supports more optimizations compared to the previous one. They differ from the number of processors used. The task partitioning between the integrated processors is based on our study on the impact of parallelization on ECSM computation.

The proposed architectures were implemented on Virtex-5 FPGA circuit. The first approach computes 256-bit sequential ECSM in 48.72 with only 1474 slices, 11 RAMs, and 8 DSP. While the parallel approaches perform 256-bit parallel ECSM between 26.87 and 14.72 ms but with extra area. They require between 2739 and 6533 slices, 22 and 72 RAMs, and between 16 and 48 DSP48E cores. The efficiency of our designs is roughly convergent and stable. Therefore, we believe that all the proposed implementations could be highly exploited as efficient ECC designs for embedded secure systems that require the trade-off between flexibility, execution time and area consumption. In the next chapter, we will investigate about the possibility exploitation of the proposed architectures on secure IoT applications based on TLS protocol.

CHAPTER 4
ZYNQ-BASED IMPLEMENTATION OF TLS

4.1 Introduction

Security management for IoT applications is a critical research field, with the performance variation over the very different IoT devices. As it is difficult to regulate PKC performance for all IoT devices, we propose to design low-cost IoT coordinators/gateways for secure IoT data collection. The main role of the coordinators is to collect data from IoT agents and send them to cloud servers. This information has to be securely sent through the internet. Thus, we propose to secure the communication between the coordinators by using the TLS handshake protocol. This latter is based on symmetric protocols, hash functions and PKC schemes to maintain data protection and privacy. In the proposed designs, low-cost and high-performance should be considered since the coordinators could be employed with a large number of IoT systems that use many IoT agents.

Recent low-cost FPGA devices are becoming a very useful platform for implementing efficient TLS protocols with the integration of ARM hardware processor. This later enables the achievement of an optimal trade-off between flexibility, area, and speed. Thus, we propose a carefully designed SW/HW implementation approach of high-performance TLS execution for efficient IoT coordinators on low-cost recent FPGA circuits. The proposed SW/HW partitioning is based on performing AES symmetric algorithm and SHA-2 hash function in SW by ARM processor. While an area-optimized hardware ECC coprocessor is used to speed up the execution of the required ECC schemes in TLS protocol. The used coprocessor allows alleviating the processor from the heavy ECSM operation in ECC protocols.

In this chapter, we present the proposed high-performance coordinators/gateways on low-cost SoC-FPGA devices for secure IoT data collection [82]. First, we present the global scenario of the targeted IoT secure application with TLS protocol. Then, we describe the internal architectures of the used ECC hardware accelerator core and the proposed Zynq-based IoT coordinators. Finally, we demonstrate the experimental results and the performance comparison with some recent works.

4.2 Global Scheme of the targeted IoT application

Figure 4.1 presents the Global Scheme of the targeted IoT application. The IoT Client coordinators (IoTC1, IoTC2) collect data from IoT agents (A1, A2, A3, A4, A5, A6) and send it to the server through the Internet. The IoT Server coordinator (IoTS) acts as an interface between the IoTCs and the server's memory, where these data will be stored. The secure data transfer between IoTCs and the IoTS is ensured by the TLSv1.2 protocol, in order to protect the information from unauthorized users. In fact, TLSv1.2 allows generating a shared secret key between the IoTS and each IoTC (Key1, Key2) that could be used to encrypt/decrypt data based on private-key algorithms.

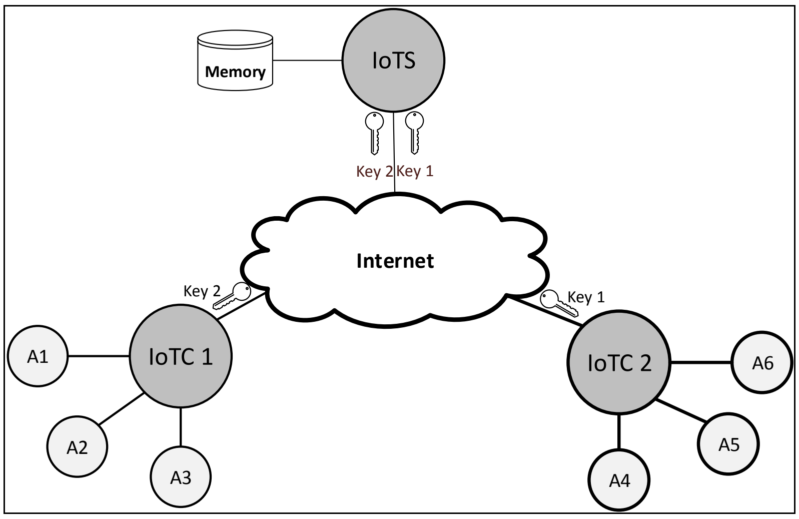


Figure 4. 1 Global Scheme of the targeted IoT application.

To achieve our main aim, we present a carefully designed SW/HW implementation of the client/server TLSv1.2 protocol for IoTCs and IoTSs, which is implemented on low-cost FPGAs/SoCs suitable for IoT applications. The main idea is to implement the core operation of ECC, which is ECSM, within a scalable hardware coprocessor accelerator and to integrate it around an ARM microprocessor. Meanwhile, the control of ECDHE and ECDSA protocols, the execution of the AES-128 algorithm, HMAC and SHA256 functions are ensured by the ARM microprocessor.

TLSv1.2 protocol allows generating a shared private key between server and client for each session based on cipher suite agreed during the TLS handshake. A demonstration of the TLS handshake between client and server is shown in Figure 1.7. The negotiations are based on sending and receiving records, which are blocks of data. Initially, TLS1.2 begins with ClientHello() (step 1), in which the client provides the cipher suite of the supported cryptographic algorithms and compression methods. It also provides random client data

(Randclient) to be used later in the handshake. Then, the server replies with ServerHello() (step 2) by providing random server data (Randserver) and the list of the selected cryptographic and compression methods to be used during the TLS process.

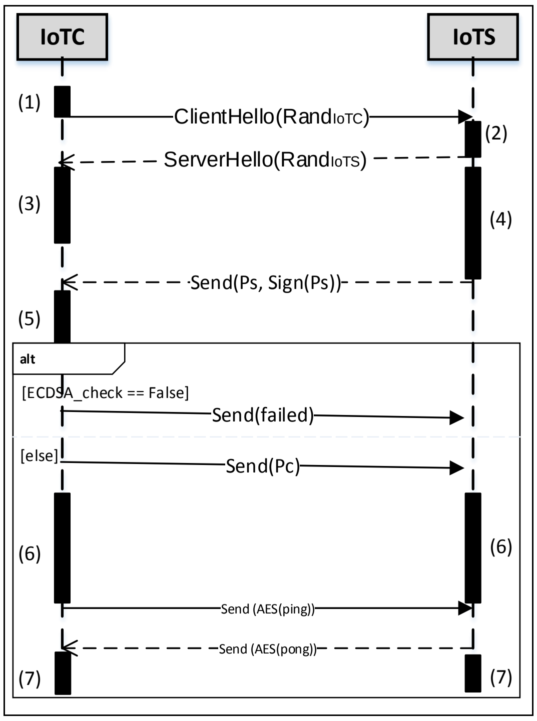


Figure 4. 2 Transport Layer Security (TLS) Handshake demonstration

Once the Hello step is done, the server and the client calculate in parallel a pair of private/public ephemeral keypairs (steps 3 and 4) using an EC-based keypair generation algorithm and send to the other party the public key. The server uses ECDSA to sign its ephemeral public key (Ps) in Step 4 and sends the signature to the client. On the other side, the client verifies the received signature using the ECDSA verification algorithm (Step 5). If the verification is successful, the client sends its public key (Pc). Then, a 384-bit shared secret key will be generated (Step 6) by the combination of ECDHE and HMAC-SHA256. The first algorithm provides a 256-bit PreMasterSecret key, while, the second generates a 384-bit MasterSecret key. From the latter, two 128-bit (client_write_key, server_write_key) secret keys are extracted. Finally, in order to check if the handshake was not tampered with (Step 7), the client and server encrypt "ping" and "pong" using the AES algorithm by server write key and client write key, respectively. Then, they exchange the encrypted messages, and each part decrypts the received message using the appropriate key to retrieve "ping" and "pong" messages. Otherwise, the TLS handshake process has been corrupted.

4.3 ECC Accelerator Design

As we saw previously, several fast and regular ECSM algorithms are reported in the literature [61]. In this work, the ECSM is performed based on the MPL algorithm over the projective coordinate system, making field operations explicit [4].

Algorithm 4. 1 Montgomery ladder over projective coordinates, making field operations explicit.

```
Inputs: $k=k_{m-1}k_{m-2}...k_1k_0$, $P(x,y)$, domain parameters (m, $a$, $b$, $G$, $n$, $h$)
Outputs: $k \times P = (x_3, y_3)$
1. $X_1 = x$, $Z_1 = 1$, $X_2 = x^4 + b$, $Z_2 = x^2$
2. $P_1 = P$, $P_2 = 2P$                \\ ECPD
3. for $i = m - 2$ downto 0 do
4.  If ($k_I$ == 0) then
5.    $T = Z_2$ ; $Z_2 = (X_1 \times T + X_2 Z_1)^2$ ; $X_2 = x Z_2 + X_1 X_2 Z_1 T$     \\ ECPA
6.    $T = X_1$ ; $X_1 = T^4 + b Z_1^4$ ;  $Z_1 = T^2 Z_1^2$           \\ ECPD
7.  else
8.    $T = Z_1$ ; $Z_1 =(X_1Z_2 + X_2 T)^2$ ; $X_1 = xZ_1 + X_1X_2Z_2T$       \\ ECPA
9.    $T = X_2$ ; $X_2 = T^4 + bZ_2^4$; $Z_2 = T^2Z_2^2$           \\ ECPD
10.  end if
11. end for
12. $x_3 = X_1Z_1^{(-1)}$
13. $y_3 = (x + x_3) [(X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1Z_2)](xZ_1Z_2)^{(-1)} + y$
```

This algorithm uses the binary representation of the scalar k as it is shown in Algorithm 4.1. The computation of ECSM based on this algorithm requires three steps: initialization (lines 1 and 2), main loop (lines 3 to 11) and calculation of the resulting point coordinates (lines 12 and 13). The first step performs two field squarings (line 1) and a single ECPD (line 2). The second step performs, at each iteration, ECPA (lines 5 and 8) followed by ECPD (lines 6 and 9). These operations are executed in the projective system by performing a set of field additions, field multiplications and field squarings. In the final step, two field inversions are required (lines 12 and 13) to obtain the coordinates (x_3, y_3) of the resulting point.

The internal architecture of the proposed ECC accelerator for ECSM computation based on Algorithm 4.1 is presented in Figure 4.3.

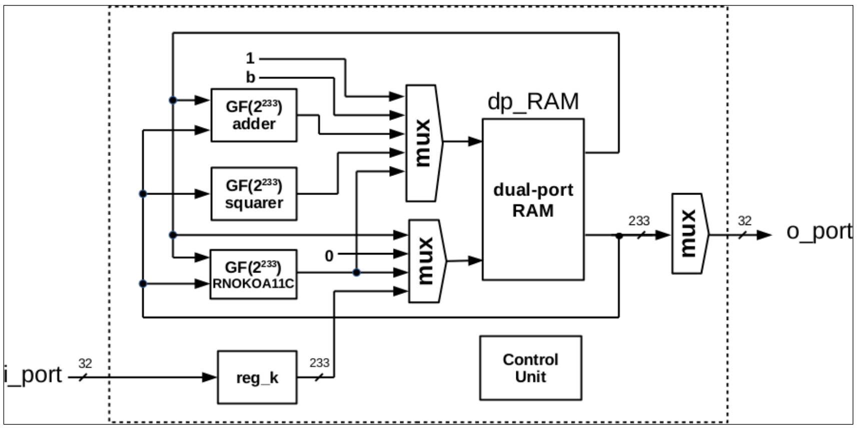


Figure 4. 3 Hardware architecture of ECC (Elliptic Curve Cryptosystems) accelerator

The proposed architecture consists of three finite field arithmetic units over GF(2^{233}) (an adder, a squarer and a multiplier, called RNOKOA11C), dual-port RAM (dp_RAM), a 233-bit register (reg_k), a control unit, and three multiplexers (mux). The field units ensure the computations of field addition, field squaring and field multiplication, respectively. The dp_RAM block is used for storing the coordinates (x, y) of the point P, the intermediate results of Algorithm 4.1 and the coordinates (x_3, y_3) of the resulting point. Meanwhile, the reg_k register is used to store the scalar value k to manage the main loop. It is also useful as temporary storage of (x,y) values before transferring them to the RAM. The control unit is responsible for the coordination between the integrated components of the internal architecture for performing ECSM. The proposed architecture provides an excellent trade-off between area and performance, based on the following aspects:

1. Exploiting the block RAMs available in FPGA devices within the internal architecture of the ECC accelerator instead of using registers, thus saving Look-Up Tables (LUT) resources at the expense of introducing some extra clock cycles.

2. Integrating the I/O interface into the ECC processing unit, taking advantage of the displacement reg_k.

3. Avoiding the use of a dedicated field divider/inverter, by means of using the Itoh-Tsujii algorithm (ITA) [3], thus requiring only the multiplier and the squaring units. In this case, our ECC accelerator needs 353 clock cycles for performing 231 squarings and 10 multiplications required for GF(2^{233}) field inversion execution. This

performance overhead is assumable, taking into account that only two field inversions are required.

It is worth mentioning, that the input (i_port) and the output (o_port) ports of the proposed ECC accelerator are 32-bit wide. It means that this ECC accelerator can be easily integrated around various 32-bit microcontrollers through 32-bit buses.

4.3.1 Field Multiplier Unit

As shown in Algorithm 4.1, the field multiplier is the one having the most noticeable effect on the performance of the scalar-point multiplying, thus requiring a careful design. The RNOKOA11C multiplier unit is implemented based on an improvement of the Karatsuba–Ofman Algorithm (KOA) [83], named Non-Overlapping KOA (NOKOA) multiplier [84]. The NOKOA multiplier allows performing field multiplication in only one clock cycle, thus enabling high-performance ECC accelerators. However, area requirements are excessive for its implementation on low-cost devices [52]. Two modifications of NOKOA, requiring 3 and 9 clock cycles for completing a field multiplication, are presented in [52]. These modifications, named NOKOA3C and NOKOA9C, respectively, require less area but are not suitable for use in our ECC scalar-point multiplication unit, due to the lack of output registers. In fact, the use of RAM blocks instead of registers makes it necessary to register the result provided by the multiplier. Figure 4.4 shows the proposed architecture of the RNOKOA11C multiplier, which meets the requirements imposed by the use of RAM as registers. It presents a recursive structure, thus consisting of a lower-level NOKOA multiplier, a control unit, two multiplexers, two XOR networks, and the RT, RE, RO and MO registers. This new multiplier requires 11 clock cycles for performing a field multiplication.

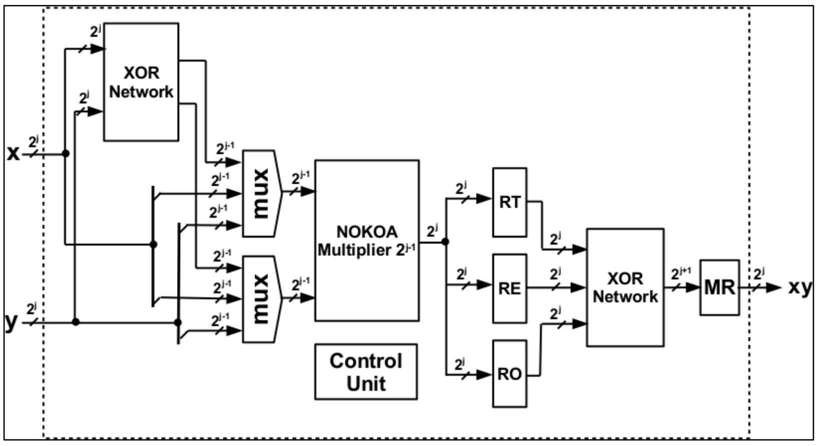


Figure 4. 4 Internal architecture of RNOKOA11C field multiplier unit

Table 4.1 shows synthesis results comparing NOKOA9C [52] to RNOKOA11C multipliers over GF(2^{233}) finite field. These results have been obtained using Xilinx ISE 14.4 over Virtex-5 devices (xv5vlx110-3f1760). As it is shown, the number of LUTs is almost the same, because the additional register required by RNOKOA11C is included in the LUTs occupied by the XOR network. Small differences in the number of LUTs and delay are due to optimizations performed by the software tool. Regarding the number of clock cycles, RNOKOA11C requires 11 clock cycles instead of the 9 clock cycles required by NOKOA9C, but it fits the requirements for being the multiplier unit of our ECC accelerator, which has been named MP_ECC_B-233_RNOKOA11C.

Table 4. 1 Synthesis results for NOKOA9C and RNOKOA11C over GF(2^{233}) on Virtex- 5 devices.

Design	**# LUTS**	**# Max. Freq. (MHz)**	**# Cycles**	**Total Time @50MHz**	**Total Time @Max. Freq.**
NOKOA9C	2366	214	9	0.18 µs	42 ns
RNOKOA11C	2344	205	11	0.22 µs	54 ns

4.3.2 Implementation of MP ECC_B-233_RNOKOA11C

In order to check the suitability of MP_ECC_B-233_RNOKOA11C for medium-performance applications, such as IoT coordinators/gateways, it has been implemented in a MiniZed board [85] with a Zynq 7Z007S device from Xilinx. This low-cost device includes a single-core ARM Cortex-A9 microprocessor and 14400 LUTs of programmable logic for software/hardware co-design. The software tool used for this implementation has been Vivado 2018.2 from Xilinx. Also, for comparison purposes, it has been implemented on Virtex 5 devices using Xilinx ISE 14.4. Implementation results are presented in Table 4.2, where MP_ECC_B-233_NOKOA11C is compared to other ECC scalar-point multipliers with similar area. From this table, it is evident that this new design requires less than half the area of other implementations, while providing similar performance figures. Thus, it is perfectly suitable for the target application. It should also be noted that our design includes a 32-bit I/O interface, while the other alternatives do not include such feature.

Table 4. 2 MP_ECC_B-233_RNOKOA11C implementation results and comparison to other implementations.

Design	#LUTS	#Cycles	Time @50MHz	Device
RNOKOA11C	3203	20637	413 µs	Virtex-5
RNOKOA11C	3395	20637	413 µs	Zynq
[52](NOKOA9C)	6223	14013	315 µs	Zynq
[86]	13396	5890	117 µs	Virtex-4
[87]	7895	5924	118 µs	Virtex-7
[88]	13244	8193	163 µs	Virtex-5

4.4 FPGA Implementation of TLS Cryptosystem

Among the considered TLS cipher-suites, HMAC, SHA256 and AES are characterized by their high-performance implementation due to relative mathematic simplicity. ECDHE and ECDSA are characterized by their high security but are considered the most time/area consuming as they involve complex operations over large prime numbers. To achieve the best trade-off between flexibility, area and speed, a SW/HW co-design implementation approach is presented in this work. The proposed partitioning is based on the implementation of ECSM within a compact ECC hardware accelerator for faster execution. The dedicated core is integrated around an embedded ARM microprocessor. The rest of the required operations for TLS negotiation are managed in SW by the processor. Figure 4.5 presents the hardware architecture of the proposed embedded system. The hardware architecture was implemented on the Xilinx Zynq-7Z007S SoC device in the Avnet Minized Dev board [85] for both IoTS and IoTC coordinators. As commented in the previous section, this low-cost device consists of a single-core ARM Cortex-A9 microprocessor, able to run at up to 666.666 MHZ, along with 100 block RAMs and 14400 slice LUTs for software/hardware co-design. The MiniZed board also includes a Murata “Type 1DX” LBEE5KL1DX wireless module for wireless communications.

The proposed architecture contains a single Cortex-A9 ARM microprocessor (PS), the MP_ECC_B-233_RNOKOA11C accelerator, an AXI interconnect bus and a Wireless_mgr controller. The latter is used for the WiFi connection of the IoTS and IoTC designs with gateways that provide internet access. The AXI bus allows 32-bit data/instruction exchanges between the ARM microprocessor and the ECC accelerator. It runs with a 50 MHz clock. The ARM processor ensures not only the control of the ECC accelerator but also of all TLS processes. The roles assigned to the processor are defined as follows:

- Generation of 256-bit random numbers.
- Execution of AES, HMAC and SHA256 functions.
- Computation of finite field inversions, multiplications and additions required for ECDSA.
- Control of the MP_ECC_B-233_RNOKOA11C accelerator.
- Control of ECDHE and ECDSA algorithms.
- Control of internet communication between the IoTS and the IoTCs.

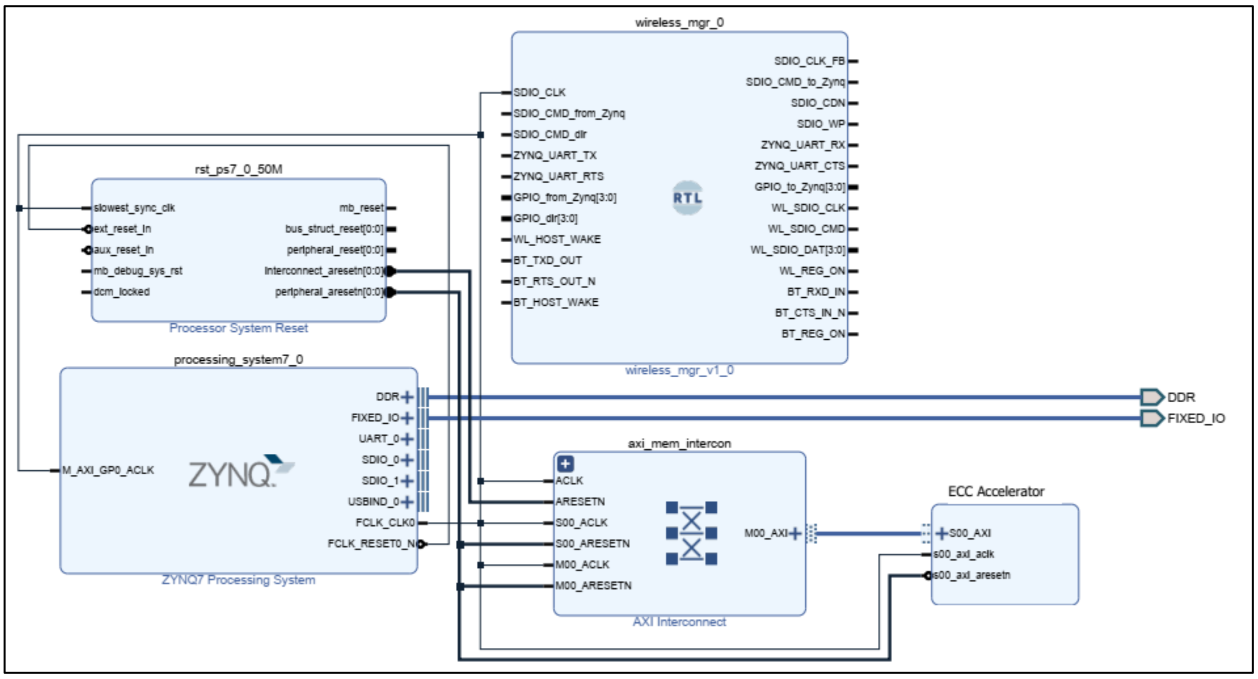


Figure 4. 5 Hardware architecture of IoTS and IoTC designs

4.4.1 ECC Accelerator Integration around ARM Processor

For connecting the ECC accelerator with the ARM processor through the AXI bus, IPIF is used for 32-bit data/instruction exchanging, as is shown in Figure 4.6. The IPIF is configured with four 32-bit registers: InsIn, DataIn, InsOut and DataOut. The processor uses a set of instruction codes through the InsIn register to manage the ECC core. The second register is used to transfer the digits of the input point coordinates and the scalar from the ARM to the req_k register. The control unit makes use of the third register to notify the processor that the coordinates of the resulting point from the ECSM computations are ready. The last register ensures the transfer of the resulting point coordinates to the processor.

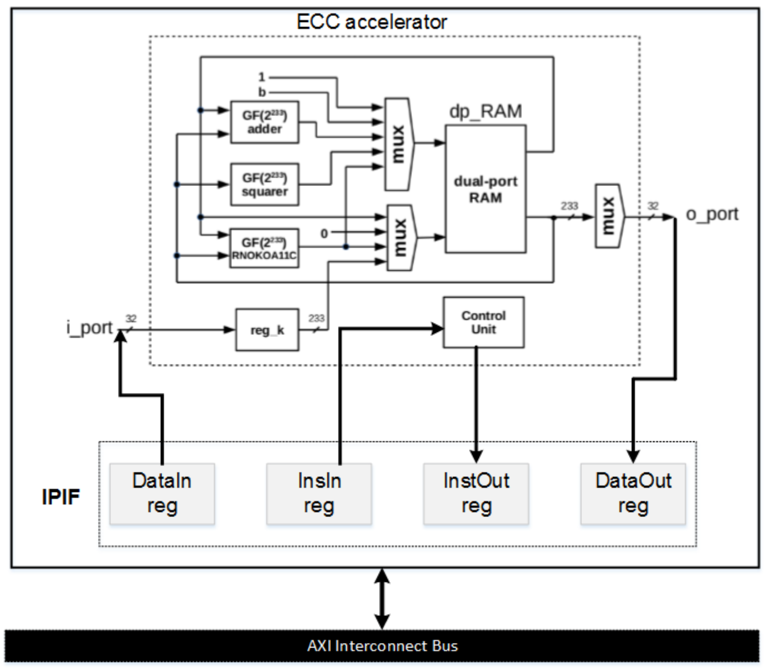


Figure 4. 6 Integration of ECC accelerator with AXI bus

To perform ECSM, three steps are required for each execution, namely, ECC core reset, the transmission of the inputs, and retrieving of the resulting point coordinates. Before starting the ECSM computation, the ARM processor resets the ECC accelerator by sending the 0x000000001 instruction. After that, the control unit stores sixteen 8-bit digits of the ECSM input point coordinates followed by eight 8-bit digits of the scalar transmitted from the processor to the dp_RAM. It must be noted that the processor transmits the 0x000000002 instruction after each digit to prepare the control unit to receive the next digit. Once twenty-four 8-bit digits of the inputs are loaded, the control unit manages the field units to perform ECSM computations. During this time, the InsOut register value is 0x000000000. When the ECC accelerator completes the execution, the control unit changes the InsOut register value to 0x000000003 in order to notify the processor that the ECSM execution is done, then sends sixteen 8-bit digits of the resulting point coordinates. The processor uses the 0x000000004 instruction after receiving each digit to order the control unit to send the next digit.

Table 4.3 summarizes the hardware resources occupied by the ECC accelerator and the proposed architecture for IoTS and IoTC coordinators on the Zynq-7Z007S device. The results are shown in terms of slice LUTs and selected RAM blocks.

Table 4. 3 Hardware resources requirements of the proposed architectures

Design	# LUTS	RAMs
ECC accelerator	3395	7
IoTS	8503	9
IoTC	8503	9

From Table 4.3, it must be noted that the difference in hardware resources between IoT designs and the ECC accelerator is 5108 LUTs and 2 RAMs. This is due to the AXI interconnect bus and the wireless_mgr controller. The proposed ECC accelerator requires only 24% of the total available LUTs in the targeted device. Meanwhile, the overall design occupies 60% of them. Moreover, the proposed architecture requires only 9 block RAMs.

4.4.2 Software Development

The proposed IoTS and IoTC coordinators run on Embedded Linux by loading the Linux boot image for Zynq (BOOT.bin) and the Linux system image (image.ub) files to the QSPI flash and the eMMC memory, respectively, both available on the board. These files are generated by means of Xilinx Petalinux 2018.2 tool based on the hardware description file (bitstream.bit) of the proposed hardware architecture. The idea behind the use of embedded Linux is that the OS allows flexible use of the WiFi module for internet communication between the IoTS and IoTCs using TCP/IP client/server sockets. Figure 4.7 summarizes the software development required to implement the TLS1.2 protocol.

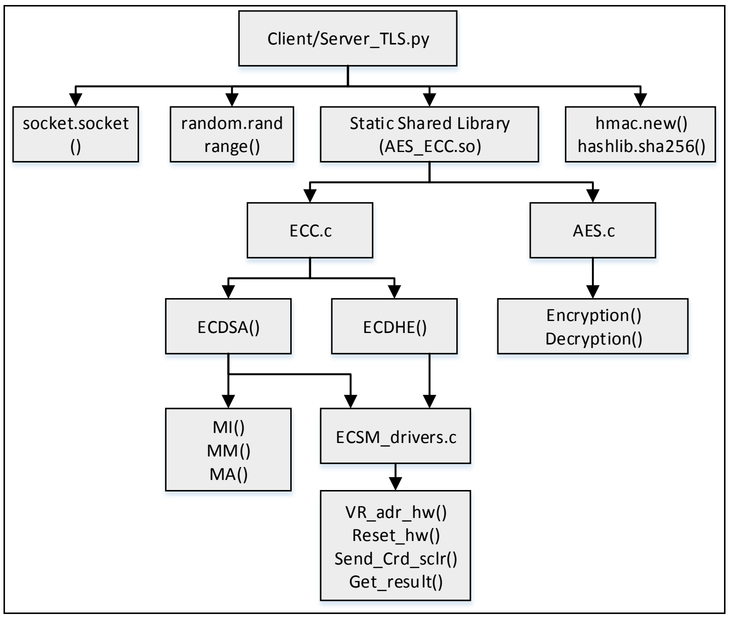

Client/Server_TLS.py
socket.socket()
random.randrange()
Static Shared Library (AES_ECC.so)
hmac.new() hashlib.sha256()
ECC.c
AES.c
ECDSA()
ECDHE()
Encryption() Decryption()
MI() MM() MA()
ECSM_drivers.c
VR_adr_hw() Reset_hw() Send_Crd_sclr() Get_result()

Figure 4. 7 TLS software functions

To implement TLS1.2 protocol between the IoTS and IoTC coordinators, Server_TLS.py and Client_TLS.py python codes have been developed for each design, respectively. Python has been used to exploit socket, random, hashlib and hmac libraries for TCP/IP socket communication, random generation, SHA256 and HMAC executions, respectively. Since Python is interpreted code, which makes its execution slower, we propose to implement the AES and ECC algorithms in C for faster executions. Then, we generate the static shared library (AES_ECC.so) from the resulting C code to be imported and used in Client/Server_TLS.py files. The C code and the static shared library are generated using the Xilinx Software Development Kit (XDSK) tool. The shared library consists of two C function files, namely AES.c and ECC.c. The first file defines AES encryption() and decryption() functions. The second file describes ECDSA() and ECDHE() functions for performing the considered ECC protocols. The two functions are based on the ECC_driver.c file and finite field functions required in the ECDSA algorithm. It must be noted that the inputs and the outputs of the AES and ECC functions are based on radix-2^8 and radix-2^{32} representations, respectively. Radix-28 is used since the AES algorithm performs 8-bit operations, while, radix-2^{32} is considered for ECC algorithms not only because the ARM is a 32-bit microprocessor but also for the AXI 32-bit bus where data/instruction are transferred digit-by-digit in serial mode. The representation of large numbers in radix 2^8 and radix 2^{32} is performed in Python based on the ctypes.c_int library. The ECDSA() function requires the computation of MA, MM and MI. In the ECDSA protocol, MA, MM and MI computations over 256-bit operands are required. These computations are ensured by the MA(), MM() and MI() functions. In fact, MM is performed based on the Montgomery radix-2^{32} Modular Multiplication algorithm. On the other hand, MI is executed by Mexp according to Fermat’s little theorem.

The ECC_driver.c file contains C drivers to control the ECC accelerator. It is composed of four functions: reset_hw(), send_crd_sclr(), Get_result() and VR_adr_hw(). The first three functions allow to reset the ECC accelerator, send the inputs of Algorithm 5.1, and retrieve the resulting point coordinates, respectively. As our designs run on embedded Linux, the ARM processor needs at system initialization to generate a virtual address (ECC_vr_adr) for the ECC accelerator and map it to its physical address (ECC_BASE_ADDR). This step is ensured by the VR_adr_hw() function, where the following instructions are executed:

1. int fd = open("/dev/mem",O_RDWR);
2. int pg_size = sysconf(_SC_PAGESIZE);

3. int pg_adr_ECC = ECC_BASE_ADDR & (pg_size-1);
4. int pg_offset_ECC = ECC_BASE_ADDR - pg_adr_ECC;
5. ECC_vr_adr = mmap(NULL, pg_size, PROT_READ|PROT_WRITE, MAP_SHARED, fd, (ECC_BASE_ADDR & (pg_size-1)));

Once the virtual address is generated, the addresses of the four registers used for data/instruction exchanging can be calculated as follows:

- InsIn_adr = *((unsigned *)(ECC_vr_adr+pg_offset_ECC))
- DataIn_adr = *((unsigned *)(ECC_vr_adr+pg_offset_ECC+4))
- InsOut_adr = *((unsigned *)(ECC_vr_adr+pg_offset_ECC+8))
- DataOut_adr = *((unsigned *)(ECC_vr_adr+pg_offset_ECC+12))

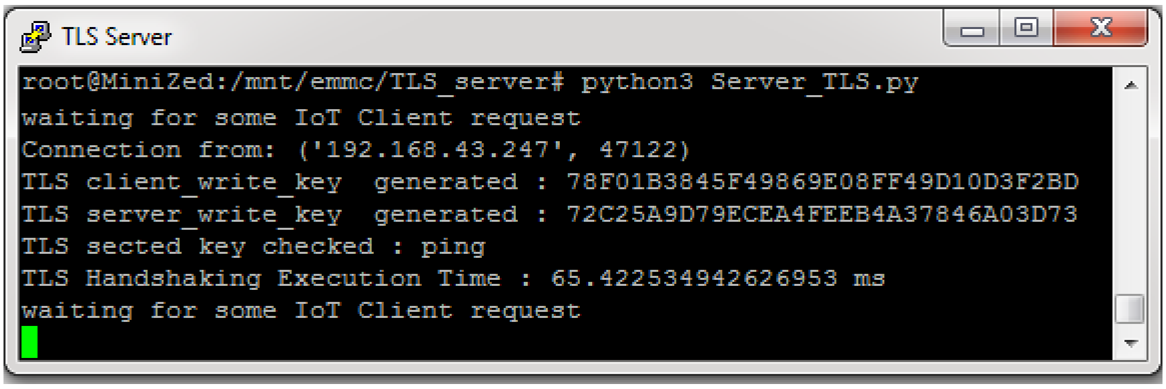
Table 4.4. presents the execution time of the developed crypto functions for the TLS1.2 protocol, as well as the time of all of the TLS1.2 process. The reported performances include the following execution times:

- Random generation.
- SHA256 and HMAC functions.
- Data representation from large numbers to radix-r for AES and ECC computations.
- Client/Server data exchanging via sockets.

Table 4. 4 Execution time of the involved crypto functions for TLS execution

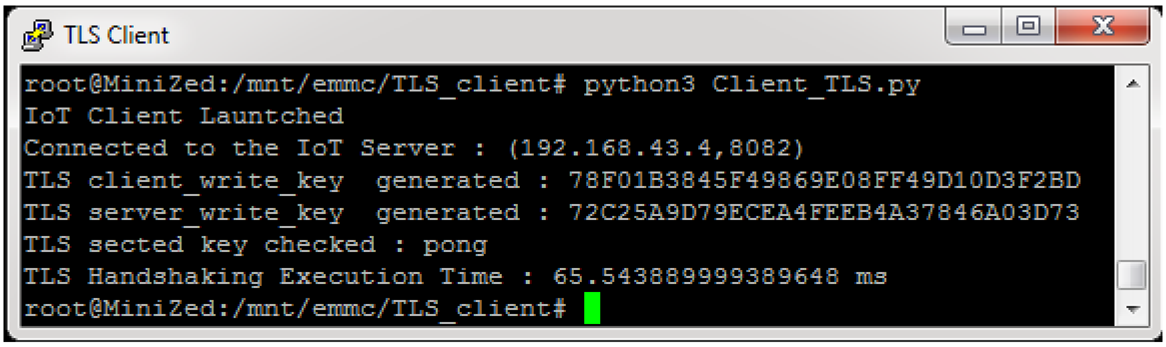
Protocol	**Bit-Length**	**Function**	**Execution Time**
AES	128 bits	AES_encryption()	56 µs
		AES_decryption()	101 µs
ECC	233 bits	ECSM()	413 µs
		ECDHE()	1.7 ms
		ECDSA_gen()	3.5 ms
		ECDSA_check()	4.1 ms
TLS	384 bits	TLS1.2()	67.5 ms

The proposed design performs a single 233-bit ECSM using the ECC accelerator in 400 µs. Moreover, the IoTS and IoTC perform the ECDHE procedure in 1.7 ms. This time depends on the size of n1 and n2, which are required in the ECDHE procedure. In our case, the size of both n1 and n2 is 233 bits. For the ECDSA protocol, the IoTS generates the signature in 3.5 ms, while the IoTC checks the received signature in 4.1 ms. These two times are linked to the bit-size of the k generated in line 1 of ECDS generation algorithm and the intermediate results (u1, u2) of ECDS verification algorithm. Finally, the generation of the 384-bit secret key between the IoTS and the IoTC based on the TLS1.2 protocol is achieved in 67.5 ms. Figure 4.8 shows a screenshot of one measurement of TLS1.2() execution in the server side (Figure 4.8a), and the client side (Figure 4.8b).



```
root@MiniZed:/mnt/emmc/TLS_server# python3 Server_TLS.py
waiting for some IoT Client request
Connection from: ('192.168.43.247', 47122)
TLS client_write_key  generated : 78F01B3845F49869E08FF49D10D3F2BD
TLS server_write_key  generated : 72C25A9D79ECEA4FEEB4A37846A03D73
TLS sected key checked : ping
TLS Handshaking Execution Time : 65.422534942626953 ms
waiting for some IoT Client request
```

(a)



```
root@MiniZed:/mnt/emmc/TLS_client# python3 Client_TLS.py
IoT Client Launtched
Connected to the IoT Server : (192.168.43.4,8082)
TLS client_write_key  generated : 78F01B3845F49869E08FF49D10D3F2BD
TLS server_write_key  generated : 72C25A9D79ECEA4FEEB4A37846A03D73
TLS sected key checked : pong
TLS Handshaking Execution Time : 65.543889999389648 ms
root@MiniZed:/mnt/emmc/TLS_client#
```

(b)

Figure 4. 8 Screenshot of TLS1.2() execution time in the (a) server side, and the (b) client side.

4.4.3 Comparison with Some Recent Works

In order to compare our proposal with other works, an ad hoc experimental setup has been prepared, which consists of two Minized boards hosting Zynq devices and communicated using WiFI. Both boards' devices include the MP_ECC_B-233_RNOKOA11C accelerator for cryptographic operations, while one of the Minized boards acts as a server and the other one acts as a client for the TLS handshaking. Figure 4.9 shows a picture of this experimental setup.

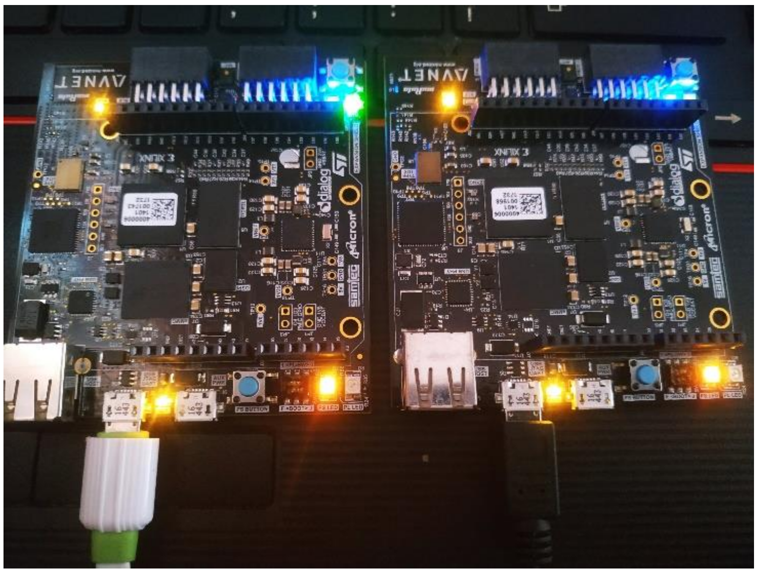


Figure 4. 9 Experimental setup for TLS handshaking.

Table 4.5 shows the performance comparison of our design and some FPGA-based TLS/SSL implementations. The comparisons are made in terms of occupied slice LUTs, selected RAM blocks and execution time for single TLS/SSL handshake negotiations.

Table 4. 5 TLSv1.2 implementation performance comparisons to recent works

Design	**Approach**	**Freq MHz**	**# LUTs**	**RAMs**	**Execution Time**	**Device**
Our Contribution	SW/HW	666	8503	9	67.5 ms	Zynq-7Z007S
[46]	HW	150	90644	216	0.62 ms	Spartan-3
[42]	HW	75	39052	75	11.3 ms	Virtex-5
[44]	SW/HW	125	27559	-	-	Zynq-7z020
[47]	SW/HW	2100	-	-	59241 kB/s (*)	Huawei-Taushia
[51]	HW	-	52005	225	220 ms	Virtex 7

(*) the authors report the performance only in terms of throughput.

Wang, H., G. Bai, and H. Chen present a Network Security Processor (NSP) implementation on a Spartan-3 FPGA device of the IPSec/SSL protocols [46]. The results show that their processor provides high timing performance by achieving 1600 full SSL handshakes per second with a 150 MHz clock. However, it requires 10 times more slice LUTs and 24 times more RAMs than our design.

An FPGA-based NSP of the TLSv1.2 protocol on a Virtex-5 device was proposed in [42]. The NSP was implemented with a secure true random number generator and ECC coprocessor. Compared to our design, the proposed processor is 6 times faster. However, it requires 5 times more slice LUTs and 9 times more RAMs.

A pipelined architecture of an NSP for the SSL/TLS protocols is implemented on a Zynq-7z020-clg484 device is presented in [44]. The proposed NSP presents high area requirements with 3 times more slice LUTs than our design. The authors did not present the timing performance of the TLS/SSL handshake.

We note that these implementations [42, 44, 46] present high-speed processors but with high-area requirements. Hence, these designs are not recommended for low-area FPGA devices, as opposed to the contrary of our design, which can be efficiently used on such devices.

A SW/HW implementation of an Energy-Efficient Crypto Accelerator (EECA) for an HTTPS server on an 8-Core HUAWEI Taishan server and an ARM Cortex-A57 CPU was proposed in [47]. The evaluation of the Web server was reported in terms of throughput and energy consumption for different data sizes, ranging from 1 KB to 2 MB. The obtained throughputs vary from 59241 KB/s to 1001 KB/s with a 2.1 GHz clock. The high-performance of this HTTPS server is obtained by using very expensive hardware platforms, once more opposed to our implementation targeting low-cost FPGA devices.

R.P. Genssler, O. Knodel, and R.G. Spallek present the implementation of the TLSv1.3 protocol for end-to-end secure connection between an Intel i5 client trusted workplace and a Virtex-7 FPGA cloud node (SecFPGA) [51]. The proposed design takes about 220 ms to perform the TLSv1.3 handshake and to deploy a 4 MB file. It requires 52005 LUTs and 225 RAMs. Thus, our design shows better time execution while requiring less area.

4.5 Conclusion

In this chapter, high-performance client/server coordinators on low-cost SoC-FPGA devices for secure IoT data collection are presented. For the purposed to secure the data transfer between the IoT agents (sensors, cameras, actuators, microchips, etc…) and the central server, we proposed to design low-cost embedded IoT clients (IoTCs) and IoT server (IoTS). The IoTCs collect data from IoT agents and send it to the server through the Internet. The IoT server acts as an interface between the IoTCs and the server's memory, where these data will be stored. The secure data transfer between IoTCs and the IoTS was ensured by the TLSv1.2 protocol based on ECC schemes. The global scenario of the targeted IoT application and the TLS Handshake protocol are presented first. From the literature, very few hardware implementations that fully support the acceleration of the TLS protocols to avoid the overhead of hardware resources utilization. The most related works focus on accelerating specific cryptography algorithms supported by the TLS. To achieve the best trade-off between flexibility, area and speed, a SW/HW co-design implementation approach is presented in this chapter, where ECSM is implemented within a scalable hardware coprocessor accelerator around ARM microprocessor. Meanwhile, the control of ECDHE and ECDSA protocols, the execution of the AES-128 algorithm, HMAC and SHA256 functions are ensured by the ARM microprocessor. The proposed partitioning allows achieving the best trade-off between flexibility, area and speed. The internal architecture of the proposed ECC accelerator and the coordinators are described then. The proposed coordinators were implemented on Zynq-7Z007S circuit. They occupied 8503 LUTs and perform full handshake negotiations in 67.5 ms. From the performance comparisons of our results and other works in the literature, it can be concluded that our designs achieve the best trade-off between security, area and speed for the target application. They require less area while providing reduced timing execution. Thus, we believe that the proposed implementation approach is suitable for small IoT embedded Client/Server secure coordinators implemented on low-cost devices.

CONCLUSION

This thesis presents novel architectures for efficient implementation of hybrid ECC-AES embedded cryptosystems on FPGA circuits. Our main aim is to achieve the best trade-off between flexibility, security, timing execution, and area consumption, with special attention to area requirements for enabling low-cost implementations while maintaining good performance figures.

The manuscript described the work progress during the thesis in chronological order. The objective of chapter 2 and chapter 3 was to summarize the state-of-the-art related to the modern cryptography and embedded systems which are essential for further understanding of the thesis.

Chapter 4 was oriented to the investigation about efficient MicroBlaze-based parallel architectures of ECSM computation for embedded Elliptic Curve Cryptosystem on Xilinx FPGA. Thus, five flexible implementations based on the combination of projective coordinate system with MPL algorithm have been proposed. To enhance the execution time, the proposed implementations are based on SW/HW co-design approach, where the critical operation MMM has been implemented in HW around MicroBlaze processors. The efficiency of the Accelerator MMM core is the exploitation of the DSP48E cores available on Xilinx FPGAs. We have shown that the parallelism could be exploited in ECSM abstraction levels with different degrees. Therefore, the second, the third, the fourth and the fifth implementations represent parallel architectures based on MPSoPC approach, where, two, three, four and six degrees of parallelism are considered, respectively. The degree of parallelism corresponds to the number of the integrated MicroBlaze processors in single architecture. Our parallel designs combine SW flexibility, HW speed, system security and MPSoPC features. To the best of knowledge, our parallel architectures are the first of its kind as embedded cryptosystem on FPGA. The Xilinx virtex-5 implementations of the proposed parallel architectures consume between 2739 and 6533 slices, 22 and 72 RAMs and between 16 and 48 DSP48E cores. Note that our cryptosystem supports arbitrary EC forms defined over large prime field (F_p) with different security-level sizes, without modifying the hardware. Depending on the considered security-level sizes, namely, 256-bit and 521-bit, our parallel implementations run at 100 MHZ frequency and take between 204 and 14.72 ms to perform single ECSM. From the performance comparisons of our results with recent works, we believe that the proposed implementations could be highly exploited for the implementation of different ECC protocols as FPGA embedded cryptosystems independently of the FPGA circuit family. This feature is ensured by the fact that MicroBlaze and our AccMMM core could be integrated in large Xilinx's FPGA circuits.

In Chapter 5, FPGA-based Client/Server designs, implemented on a Zynq FPGA device, of the TLSv1.2 protocol for IoT applications is presented. To improve the execution time, a SW/HW co-design implementation approach is proposed. Thus, the critical ECSM is implemented in HW around an ARM Cortex A9 microprocessor, while, the control of the TLSv1.2 handshake negotiations is ensured by the processor, which runs on embedded Linux OS for Zynq. The proposed 32-bit I/O ECC accelerator requires only 3395 slice LUTs, thus allowing not only flexible integration around various 32-bit microprocessors but also an easier implementation on low-cost FPGA devices. The proposed architecture occupies 8503 LUTs and performs full handshake negotiations between IoTS and IoTC designs in 67.5 ms. From the performance comparisons of our results and other works in the literature, it can be concluded that our design achieves the best trade-off between security, area and speed for the target application. It requires less area while providing reduced timing execution. Therefore, the proposed implementation approach is suitable for small IoT embedded Client/Server secure coordinators implemented on low-cost devices.

The focus of the first work was the efficient implementation of ECSM operation over the prime field. To make this work usable in practice, the proposed architectures should be exploited within heterogeneous MPSoC architectures for designing symmetric/asymmetric hybrid embedded cryptosystems on FPGA circuits based on ECC protocols. The proposed designs should be incorporated also with the second work to extend the supported TLS cipher-suites for prime curves.

In other hand, several new forms of EC curves have been proposed such as Edwards, twisted Edwards, Montgomery, etc. The computations of ECSM over these curves are faster and more secure against side-channel attacks compared to the Weierstrass forms. Thus, a possible future work could be an efficient implementation of ECSM over these new forms of EC curves on FPGA circuits.

REFERENCE

1. C. Paar and J. Pelzl, Understanding cryptography: a textbook for students and practitioners. 2009: Springer Science & Business Media.
2. A.J. Menezes, P.C.V. Oorschot and S.A. Vanstone, Handbook of applied cryptography. 1996: CRC press.
3. D. Hankerson, A.J. Menezes and S. Vanstone, Guide to Elliptic Curve Cryptography. 2003: Springer-Verlag. 332.
4. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen and F. Vercauteren, Handbook of Elliptic and Hyperelliptic Curve Cryptography, Second Edition. 2012: Chapman & Hall, CRC. 1024.
5. S. Blake-Wilson, B. Moeller, V. Gupta, C. Hawk and N.B.O Wheeler, Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS). 2006, RFC 4492.
6. D. Johnson, A. Menezes, and S. Vanstone, The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal of Information Security, 2001. 1(1): p. 36-63.
7. R.L Rivest., A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978. 21(2): p. 120-126.
8. NIST, Data Encryption Standard (DES) (FIPS–46-3), National Institute of Stan-dards and Technology. 1999.
9. NIST, Advanced Encryption Standard (AES) (FIPS–197), National Institute of Standards and Technology. 2001.
10. NIST, Secure Hash Standard (SHS) (FIPS 180-4), National Institute of Standards and Technology. 2015.
11. NIST, Secure Hash Standard (SHS) (FIPS 202), National Institute of Standards and Technology. 2015.
12. H. Krawczyk, M. Bellare, and R. Canetti, HMAC: Keyed-Hashing for Message Authentication. 1997: RFC Editor.
13. H. Tao, M.Z.A. Bhuiyan, A.N. Abdalla, M.M. Hassan, J.M. Zain and T. Hayajneh, Secured Data Collection With Hardware-Based Ciphers for IoT-Based Healthcare. IEEE Internet of Things Journal, 2019. 6(1): p. 410-420.
14. T. Dierks and E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2. 2008, RFC 5246.
15. K. Järvinen, Studies on high-speed hardware implementation of cryptographic algorithms. 2008, phd thesis, Helsinki University of Technology, Espoo, Finland.

16. A. Jerraya and W. Wolf, Chapter 1-The What, Why, and How of MPSoCs, in Multiprocessor Systems-on-Chips. 2005, Morgan Kaufmann: San Francisco, Elsevier.

17. L. Torres, P. Benoit, G. Sassatelli, M. Robert, F. Clermidy and D. Puschini, et al., An introduction to multi-core system on chip–trends and challenges, in Multiprocessor System-on-Chip. 2011, Springer. p. 1-21.

18. H. Kamarulhaili and L.K. Jie, Elliptic Curve Cryptography and Point Counting Algorithms. Cryptography and Security in Computing, 2012: p. 91.

19. H. Marzouqi, M. Al-Qutayri and K. Salah, Review of elliptic curve cryptography processor designs. Microprocessors and Microsystems, 2015. 39(2): p. 97-112.

20. B. Halak, S.S. Waizi and A. Islam, A survey of hardware implementations of elliptic curve cryptographic systems. IACR Cryptol. ePrint Arch. 2016.

21. M. Rashid, M. Imran, and A.R. Jafri. Comparative analysis of flexible cryptographic implementations. in 2016 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). 2016. IEEE.

22. N. Alimi, Y. Lahbib, M.Machhout, T. Mohsen and T. Rached. On elliptic curve cryptography implementations and evaluation. in 2016 2nd International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). 2016. IEEE.

23. S. Agarwal, S. Saha, R. Paul and A. Chakrabarti, Performance Evaluation of ECC in Single and Multi Processor Architectures on FPGA Based Embedded System. arXiv preprint arXiv:1401.3421, 2014.

24. K. Ananyi, H. Alrimeih and D. Rakhmatov, Flexible Hardware Processor for Elliptic Curve Cryptography Over NIST Prime Fields. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2009. 17(8): p. 1099-1112.

25. S.Ghosh, D. Mukhopadhyay, and D. Roychowdhury, Petrel: Power and Timing Attack Resistant Elliptic Curve Scalar Multiplier Based on Programmable Arithmetic Unit. IEEE Transactions on Circuits and Systems, 2011. 58(8): p. 1798-1812.

26. B. Baldwin, R.R. Goundar, M. Hamilton and W.P. Marnane, Co-Z ECC scalar multiplications for hardware, software and hardware–software co-design on embedded systems. Journal of Cryptographic Engineering, 2012. 2(4): p. 221-240.

27. H.Marzouqi, M. Al-Qutayri, and K. Salah. An FPGA implementation of NIST 256 prime field ECC processor. in 2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS). 2013.

28. J. Balasch, B. Gierlichs, K. Ja, and I. Verbauwhede,. Hardware/software co-design flavors of elliptic curve scalar multiplication. in 2014 IEEE International Symposium on Electromagnetic Compatibility (EMC). 2014.

29. H.Alrimeih and D. Rakhmatov, Fast and Flexible Hardware Support for ECC Over Multiple Standard Prime Fields. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2014. 22(12): p. 2661-2674.

30. H. Marzouqi, M. Al-Qutayri, K. Salah, D. Schinianakis and T. Stouraitis, A High-Speed FPGA Implementation of an RSD-Based ECC Processor. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016. 24(1): p. 151-164.

31. M. Joye and S.M. Yen. The Montgomery powering ladder. in International workshop on cryptographic hardware and embedded systems. 2002. Springer.

32. P.L Montgomery, Modular multiplication without trial division. Mathematics of computation, 1985. 44(170): p. 519-521.

33. C.K.Koc, T. Acar and B.S. Kaliski, Analyzing and Comparing Montgomery Multiplication Algorithms. IEEE Micro, 1996. vol 16(3): p. 26-33.

34. Xilinx, I., Microblaze processor reference guide. reference manual, 2006. 23.

35. B. Senouci, F. Rousseau and F. Petrot. Multi-CPU/FPGA platform based heterogeneous multiprocessor prototyping: New challenges for embedded software designers. in 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping. 2008. IEEE.

36. M. Issad, B. Boudraa, M. Anane and N. Anane,, Software/Hardware Co-Design of Modular Exponentiation for Efficient RSA Cryptosystem. Journal of Circuits, Systems and Computers, 2014. vol 23(3).

37. UG193, X., Virtex-5 FPGA XtremeDSP Design Considerations User Guide. 2009, Jan.

38. S. Wang, Y. Hou, F. Gao and X. Ji, et al. A novel IoT access architecture for vehicle monitoring system. in 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT). 2016.

39. J. Dofe, J. Frey, and Q. Yu. Hardware security assurance in emerging IoT applications. in 2016 IEEE International Symposium on Circuits and Systems (ISCAS). 2016.

40. A. Al-Omary, O. Ali, H. M. AlSabbagh and H. Al-Rizzo, Survey of Hardware-based Security support for IoT/CPS Systems. KnE Engineering, 2018. 3(7).

41. B. Moeller, D. Duong and K.Ko towicz, This POODLE bites: exploiting the SSL 3.0 fallback. 2014, Security Advisory (Google).

42. M. Hamilton and W.P. Marnane, Implementation of a secure TLS coprocessor on an FPGA. Microprocessors and Microsystems, 2016. 40: p. 167-180.

43. M. Khalil-Hani, V.P. Nambiar, and M.N. Marsono. Hardware Acceleration of OpenSSL Cryptographic Functions for High-Performance Internet Security. in 2010 International Conference on Intelligent Systems, Modelling and Simulation. 2010.

44. R. Paul, A. Chakrabarti, and R. Ghosh, Multi core SSL/TLS security processor architecture and its FPGA prototype design with automated preferential algorithm. Microprocessors and Microsystems, 2016. 40: p. 124-136.

45. R. Paul, and S. Shukla, Partitioned security processor architecture on FPGA platform. IET Computers & Digital Techniques, 2018. 12(5): p. 216-226.

46. H. Wang, G. Bai, and H. Chen, A Gbps IPSec SSL Security Processor Design and Implementation in an FPGA Prototyping Platform. J. Signal Process. Syst., 2010. 58(3): p. 311-324.

47. C. Xiao, L. Zhang, W. Liu, N. Bergmann and Y. Xie, Energy-efficient crypto acceleration with HW/SW co-design for HTTPS. Future Generation Computer Systems, 2019. 96: p. 336-347.

48. D. B. Roy, S. Agrawal, C. Reberio and D. Mukhopadhyay. Accelerating OpenSSL's ECC with low cost reconfigurable hardware. in 2016 International Symposium on Integrated Circuits (ISIC). 2016.

49. J. Viega , P. Chandra, and M. Messier, Network Security with Openssl. 2002: O'Reilly &amp; Associates, Inc. 384.

50. L. Wu, C. Weaver, and T. Austin. CryptoManiac: a fast flexible architecture for secure communication. in Proceedings 28th Annual International Symposium on Computer Architecture. 2001.

51. Paul R. Genssler, Oliver Knodel, and R.G. Spallek, Securing Virtualized FPGAs for an Untrusted Cloud, in ESCS'18. 2018: Las Vegas, Nevada, USA.

52. L. Parrilla, et al., Elliptic Curve Cryptography hardware accelerator for high-performance secure servers. The Journal of Supercomputing, 2019. 75(3): p. 1107-1122.

53. L. Parrilla, et al., Unified Compact ECC-AES Co-Processor with Group-Key Support for IoT Devices in Wireless Sensor Networks. Sensors, 2018. 18(1): p. 251.

54. Cortex, A., A9 MPCore: Technical Reference Manual Revision: r4p1. ARM information Center, 2012.

55. Xilinx, A., Reference Guide, UG761 (v13. 1). URL http://www. xilinx. com/support/documentation/ip documentation/ug761 axi reference guide. pdf, 2011.

56. J. Nechvatal, et al., Report on the development of the Advanced Encryption Standard (AES). Journal of Research of the National Institute of Standards and Technology, 2001. 106(3): p. 511.

57. W.E. Burr, Selecting the advanced encryption standard. IEEE Security & Privacy, 2003. 1(2): p. 43-52.

58. E.W Weisstein, RSA Number. 2003.

59. N. Koblitz, Elliptic curve cryptosystems, in Math Comput 48 (177). 1987. p. 109–203.

60. C. Rebeiro, ARCHITECTURE EXPLORATIONS FOR ELLIPTIC CURVE CRYPTOGRAPHY ON FPGAS. 2008.

61. Nist, Recommended Elliptic Curves for Federal Government Use. 1999.

62. A.Miyaji, T. Ono, and H. Cohen, Efficient elliptic curve exponentiation, in Information and Communications Security: First International Conference, ICIS '97 Beijing, China, November 11–14, 1997 Proceedings, Y. Han, T. Okamoto, and S. Qing, Editors. 1997, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 282-290.

63. IEEE, IEEE Standard Specifications for Public-Key Cryptography. IEEE Std 1363-2000, 2000: p. 1-228.

64. B.D Masmoudi, Performance and complexity optimization in heterogeneous multiprocessors system on chip. Thesis, 2015.

65. Trimberger, S.M.S., Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology: This Paper Reflects on How Moore's Law Has Driven the Design of FPGAs Through Three Epochs: the Age of Invention, the Age of

Expansion, and the Age of Accumulation. IEEE Solid-State Circuits Magazine, 2018. 10(2): p. 16-29.

66. Xilinx, I., Introduction to fpga design with vivado high-level synthesis. 2013.

67. Zhang, J., et al., Techniques for Design and Implementation of an FPGA-Specific Physical Unclonable Function. Journal of Computer Science and Technology, 2016. 31: p. 124-136.

68. F. Gioulekas, M. Birbas, N. Voros, G. Kouklaras and A. Birbas,, Heterogeneous system level co-simulation for the design of telecommunication systems. Journal of Systems Architecture, 2005. 51: p. 688-705.

69. Serrano, J., Introduction to FPGA design. 2008, CAS - CERN Accelerator School: Digital Signal Processing, pp.231-247, DOI: 10.5170/CERN-2008-003.231.

70. Yousif, O.F., et al. FPGA based embedded homogenous and hetrogenous multi-processor SoC design: A review. in 2014 IEEE Conference on Open Systems (ICOS). 2014. IEEE.

71. M.A. Cohen, Nouveaux outils de profilage de MP-SoC basés sur des techniques de fouille de données. 2014, , theseUniversité Joseph Fourier.

72. M. Rivain, Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves. IACR Cryptology ePrint Archive, 2011. 2011: p. 338.

73. H.Cohen, A. Miyaji, and T. Ono, Efficient Elliptic Curve Exponentiation Using Mixed Coordinates, in Advances in Cryptology — ASIACRYPT'98: International Conference on the Theory and Application of Cryptology and Information Security Beijing, China, October 18–22, 1998 Proceedings, K. Ohta and D. Pei, Editors. 1998, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 51-65.

74. C.D. Walter, Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli. in Cryptographers', RSA Conference. 2002. Springer.

75. A.M. Bellemou, et al., MicroBlaze-Based Multiprocessor embedded cryptosystem on FPGA for Elliptic Curve Scalar Multiplication over Fp. Journal of Circuits, Systems, and Computers, 2018. 28(03).

76. A.M. Bellemou, et al., Microblaze-based parallel implementations of elliptic curve scalar multiplication over Fp on FPGA. International Journal of Internet Technology and Secured Transactions, 2020. 10(1-2): p. 171-195.

77. Xilinx, I., PLB IPIF. Xilinx Document DS448 (v2. 02a), Xilinx Inc, 2005.

78. M. Karim, A MicroBlaze-based Multiprocessor System on Chip for real-time cardiac monitoring. in 2014 International Conference on Multimedia Computing and Systems (ICMCS). 2014. IEEE.

79. Xilinx, Genesys Board, Reference Manual, Revision. 2012.

80. C.L. Sotiropoulou and S. Nikolaidis. Design space exploration for FPGA-based multiprocessing systems. in 2010 17th IEEE International Conference on Electronics, Circuits and Systems. 2010. IEEE.

81. A.M. Bellemou, et al., Efficient Implementation on Low-Cost SoC-FPGAs of TLSv1. 2 Protocol with ECC_AES Support for Secure IoT Coordinators. Electronics, 2019. 8(11): p. 1238.

82. M.Rivain, Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves. International Association for Cryptologic Research (IACR), 2011.

83. A. Karatsuba, MathThe complex ity of computations, Proc Steklov I nst. 1995. p. 169–183.

84. H. Fan, J. Sun, M. Gu and K. Lam , Overlap-free Karatsuba-Ofman polynomial multiplication algorithms. IET Information Security, 2010. 4(1): p. 8-14.

85. Minized, Minized board datasheet.

86. B. Ansari and M.A. Hasan, High-Performance Architecture of Elliptic Curve Scalar Multiplication. IEEE Transactions on Computers, 2008. 57(11): p. 1443-1453.

87. Z. Khan and M. Benaissa, Throughput/Area-efficient ECC Processor Using Montgomery Point Multiplication on FPGA. IEEE Transactions on Circuits and Systems II: Express Briefs, 2015. 62(11): p. 1078-1082.

88. G.D. Sutter, J. Deschamps, and J.L. Imana, Efficient Elliptic Curve Point Multiplication Using Digit-Serial Binary Field Operations. IEEE Transactions on Industrial Electronics, 2013. 60(1): p. 217-225.