Efficient Hardware Architecture for Scalar Multiplications on Elliptic Curves over Prime Field

Khalid Javeed

BEng, MEng

A Disertation submitted in fulfilment of the requirements for the award of Doctor of Philosophy (Ph.D.)

DUBLIN CITY UNIVERSITY

SCHOOL OF ELECTRONIC ENGINEERING

Supervisors: Dr. Xiaojun Wang and Dr. Mike Scott

September 2016

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ph.D is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: ——————————————————————-

Candidate ID No: ——————————————————————-

Date:——————————————————————-

Acknowledgement

I would like to express my sincere gratitude to my supervisor Dr. Xiaojun Wang for his continuous support in technical and non-technical matters related to my Ph.D studies, research work and thesis writing. I am also very thankful to Dr. Mike Scott for helping in understanding elliptic curve cryptography.

I thank my all fellow lab mates for their help and support. Due to their company, my stay at Dublin City University was comfortable and enjoyable.

Last but not the least, I would like to pay my humble but full of emotion gratitude to my parents without their prayers and assistance this would not have been possible. I would also like to thank my entire family for providing me courage which I required most of the time during this work. Lastly, I am very thankful to my loving wife, my lovely daughter Meerab and son Abdul Hadi for their love, support and patience.

Contents

List of Acronyms

AES Advanced Encryption Standards

Add Addition

ASIC Application Specific Integrated Circuit

ATB Area-Time product per bit

AU Arithmetic Unit

BE Booth Encoding

CLB Configurable Logic Block

DA Double-and-Add

DAA Double-and-always-add

DES Digital Encryption Standards

DLP Discrete Logarithm Problem

Div Division

DPA Differential Power Analysis

EC Elliptic curve

ECC Elliptic curve Cryptography

ECDH Elliptic Curve Diffie-Hellman

ECDSA Elliptic Curve Digital Signature Algorithm

ECDLP Elliptic Curve Discrete Logarithm Problem

FPGA Field Programmable Gate Array

Freq Frequency

IM Interleaved Modular Multiplication

Inv Inversion

LUT Look-Up-Table

MM Modular Multiplication

MMM Montgomery Modular Multiplication

MPL Montgomery Powering Ladder

MR Montgomery Reduction

Mul Multiplication

NAF Non-Adjacent Form

NIST National Institute of Standards and Technology

PA Point Addition

PAU Parallel Arithmetic Unit

PKC Public Key Cryptography

PD Point Doubling

PKI Public Key Infrastructure

RSA Rivest Shamir Adleman

R2IM Radix-2 Interleaved Modular Multiplication

R2PIM Radix-2 Parallel Interleaved Modular Multiplication

R4BIM Radix-4 Booth Encoded Interleaved Modular Multiplication

R8BIM Radix-8 Booth Encoded Interleaved Modular Multiplication

R4PIM Radix-4 Parallel Interleaved Modular Multiplication

R4BPIM Radix-4 Booth Encoded Parallel Interleaved Modular Multiplication

R8PIM Radix-8 Parallel Interleaved Modular Multiplication

R8BIM Radix-8 Booth Encoded Parallel Interleaved Modular Multiplication

SPA Simple Power Analysis

Sub Subtraction

TPAR Timing and Power Attacks Resistance

List of Figures

List of Tables

List of Algorithms

Efficient Hardware Architecture for Scalar Multiplications on Elliptic Curves over Prime Field

Khalid Javeed

Abstract

Suitable cryptographic protocols are required to meet the growing demands for data security in many different systems, ranging from large servers to small hand-held devices. Many constraints such as computation time, silicon area, power consumption, and security level must be considered by the designers of hardware accelerators of the cryptogrpahic protocols.

Elliptic curve cryptography (ECC) proposed by Koblitz and Miller, has been widely accepted. It is now considered as one of the best Public-Key Cryptography (PKC) algorithms and provides higher security strength per bit than RSA, with considerably smaller key sizes. For example, a 256-bit ECC can provide the same security strength as 3072-bit RSA. Due to its much smaller key sizes, ECC based crypto-systems are better in terms of bandwidth utilization, power consumption, and implementation cost as compared to the traditional RSA based crypto-systems. However, PKC algorithms, especially ECC are relatively expensive as compared to their symmetric-key counterparts in terms of computation time. It is an open area of research to reduce their computation cost, so that they could be used for secure communication in commercial internet based applications. Efficient implementation of elliptic curve cryptography over several new platforms have been explored in the last few decades.

This work presents efficient design strategies to perform elliptic curve scalar multiplication, the fundamental operation in all ECC based crypto-systems. Finite field arithmetic is the bottleneck in the computation of the EC scalar multiplication operation. Especially, finite field multiplication is the most time-critical operation in projective coordinates, a technique which eliminates modular inversion/division from elliptic curve group operations.

Two efficient design strategies to perform finite field multiplication are presented. The first design strategy proposes modifications to the interleaved modular multiplication algorithm using radix-4, radix-8 and Booth encoding techniques to reduce the required number of clock cycles to perform a finite field multiplication. However, higher-radix techniques incur longer critical path delay so performance is limited.

Subsequently, parallel optimization techniques are incorporated in the modified interleaved modular multiplication algorithms which enable concurrent execution of the critical operations. So the higher-radix parallel modular multipliers are optimized in terms of required number of clock cycles and critical path delays. It is observed that using Booth encoding in the parallel modular multipliers can reduce resource requirements with a slight degradation in the speed performance.

Based on the presented finite field multipliers, low latency flexible architectures to perform elliptic curve point multiplication over general prime field GF(p) is developed. On a system level, standard double-and-add and double-and-always-add techniques

are adopted. The implementation results show that the presented elliptic curve scalar multiplier architectures in this work are good trade-offs between performance and flexibility. The presented designs support general prime field so these can be used in many ECC applications.

Publications

This work is based on the following contributions.

- Khalid Javeed and Xiaojun Wang "FPGA Based High Speed SPA Resistant Elliptic Curve Scalar Multiplier Architecture", International Journal of Reconfigurable Computing, Volume 2016 (2016), Article ID 6371403, 10 pages, http://dx.doi.org/10.1155/2016/6371403
- Khalid Javeed and Xiaojun Wang "Design and performance analysis of Modular Multipliers on FPGA Platform." International Conference on Cloud Computing and Security (ICCCS 2016), Nanjing, China.
- Khalid Javeed, Xiaojun Wang "Speed and Area Optimized Higher Radix Modular Multipliers "Cryptology ePrint Archive, Report 2016/053, 2016.
- Khalid Javeed, Xiaojun Wang, and Mike Scott "Serial and Parallel Modular Multipliers over FPGA Platform " in IEEE international Conference on Field Programmable Logic and Applications (FPL 2015), London, United Kingdom.
- Khalid Javeed and Xiaojun Wang "Radix-4 and radix-8 Booth encoded interleaved modular multipliers over general prime field" in IEEE international Conference on Field Programmable Logic and Applications (FPL 2014), Munich, Germany.
- Khalid Javeed and Xiaojun Wang "Efficient Montgomery Multiplier for ECC and Pairing based Cryptography " in IEEE International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP 2014), Manchester, United Kingdom.

Chapter 1

Introduction

1.1 Thesis Motivation

Cryptography is the study and design of methods to protect secret information over an insecure channel against adversaries. Cryptographic protocols are imperative to protect files and other information due to the rapid growth of security requirements on the Internet being used as a channel for communication and business in today's society. Billions of people are using the Internet as a tool for communication, e-commerce, internet banking, storage and retrieval of sensitive data from cloud servers, wireless sensors networks, mobile commerce, and many others.

Successful deployment of a data communication network depends largely on the network's ability to counter against different unwanted attackers (users), that is, how secure the network system is in the presence of many fraudulent users. Therefore, Suitable cryptographic schemes are essential to meet the growing demands for data security in many different systems, ranging from large servers to small hand-held devices. Many constraints such as computation time, area consumption, flexibility, and security must be considered by network system designers.

Different systems have different computing powers, resource limitations, and security requirements. For example, a server needs to complete a large number of tasks

in a short duration of time, whereas, a more compact design is required to meet security demands in hand-held devices such as smartphones and smartcards because of their resource limitations.

On the other hand cryptanalysis, a reverse operation of cryptography, is the study of methods to break cryptographic systems either by solving the underlying mathematical problem or by exploiting the algorithmic and implementation weaknesses of the crypto-system . With the rapid advancement in technology, cryptanalysis has also flourished. Many new efficient cryptanalysis algorithms and procedures have been figured out to attack cryptographic systems either to reveal sensitive data, to alter sensitive data, or to hack a system to perform a task for which it is not designed for. Thus, security of a system can be compromised at any time. Therefore, the underlying implementation platform must be flexible to adopt new algorithms and security parameters regularly to avoid any security breach.

Dedicated hardware architectures are essential to meet the speed requirements of many real-time applications. Dedicated hardware processors have many advantages over the general purpose processor (GPP). For example, implementation of a crypto scheme on dedicated hardware yields higher performance and lower power consumption results compared to an implementation of the crypto scheme on a GPP. Therefore, in many applications the most computationally intensive tasks are performed on dedicated hardware to boost the overall performance of the systems. However, an implementation on GPP is more flexible than dedicated hardware. Field programmable gate array (FPGA) is a hardware platform that can offer the performance of a dedicated hardware as well as the flexibility of a GPP.

FPGA is a hardware platform which provides the flexibility for users to replace a current design with a new one in-house. FPGA has established itself as a suitable platform for implementation of security algorithms. This is due to its short design cycle time, low cost and re-usability which make it a more attractive choice compared to application specific integrated circuits (ASICs). It should be pointed out FPGA offers flexibility at the cost of lower performance and higher power consumption compared to ASIC. Therefore for applications demanding a balance of performance and flexibility, FPGA implementation is recommended. ASIC implementation is preferable in applications where high performance is the only major requirement.

1.2 Thesis Aim

Many security protocols are designed using Elliptic Curve Cryptography (ECC), RSA and Pairing-based Cryptography. All these are popular types of public key cryptography (PKC) also known as asymmetric cryptography which is discussed in detail in chapter 2.

The complex and elegant mathematics of elliptic curves have attracted many researchers which led to the proposal of ECC by Miller [1] and Koblitz [2] in 1985. It provides relatively high security strength per bit resulting in a reduced bit length compared to RSA [3]. The reduced bit length means elliptic curve cryptosystems require smaller key sizes for a certain security level as compared to the traditional cryptosystems like RSA. For example, to achieve a 128-bit Advanced Encryption Standard (AES) security level, the US National Institute of Standards and Technology (NIST) recommends ECC key sizes of 256 bits. To achieve the same security level with RSA would require key sizes of 3072 bits, which is almost twelve times more than the corresponding ECC key sizes.

As a consequence, this significant reduction in key sizes has led to several new power and memory efficient implementations of ECC schemes in a variety of resource constrained environments such as wireless sensor nodes, smartphones, smartcards and many other hand-held devices and the Internet of Things (IOT).

All elliptic curve (EC) cryptographic schemes depend on a scalar multiplication operation, denoted as $Q = dP$, where a point P on a suitably chosen elliptic curve is multiplied with a scalar d to obtain another point Q on the same curve. In this scenario points P, Q are public parameters while the scalar d is a secret used to enable a secure communication. To find the scalar d knowing points P and Q is believed to be an intractable problem and widely known as Elliptic Curve Discrete Logarithm Problem (ECDLP), which is the basis of all ECC based schemes.

The overall performance of any ECC scheme depends on the efficient computation of the elliptic curve scalar multiplication operation, which is the most computationally intensive operation. Implementation of EC scalar multiplication on a general purpose processor can not meet performance demands of many time critical real time applications. Hence, there is a need for high speed, flexible, and reconfigurable hardware

accelerators to reduce the computation time of EC scalar multiplication. The implementation of the EC scalar multiplication must be cost effective both in terms of time and space requirements.

The main objective of this research work is to design efficient hardware architectures to compute the EC scalar multiplication operation. The scalar multiplication dP is achieved through a series of EC group operations i.e., EC point addition and EC point doubling. These group operations further rely on finite field arithmetic primitives, i.e., addition, subtraction, multiplication, and inversion/division. Among these filed operations, multiplication and inversion/division are very critical components and their efficient implementation can significantly speed up EC scalar multiplication. One common optimization technique is to eliminate inversion/division operations from EC group operations at the cost of extra modular multiplication operations. Hence, an optimized modular multiplier is very critical in a high performance design of EC scalar multiplier. Therefore, this research work focuses on the design of flexible and low latency modular multipliers over general prime field. There are several scalar multiplication algorithms and many different elliptic curves offering different tradeoffs between computational performance and level of security, therefore flexibility is considered in the proposed designs, which is required in many applications.

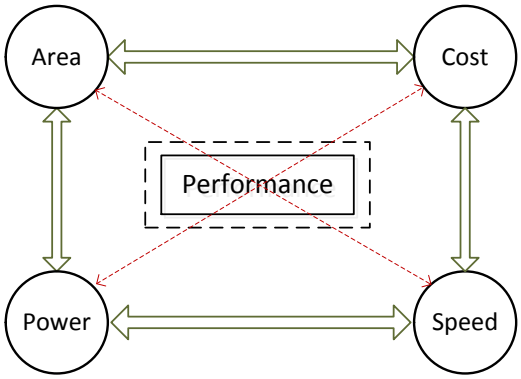


Figure 1.1: Performance evaluation metrics

This research work first explores several hardware design techniques to optimize the finite field arithmetic primitives especially a modular multiplication operation. Subsequently, based on these optimized finite field arithmetic primitives and by exploiting the possible parallelism in EC group operations, the work focuses on the design of high performance hardware architectures to perform EC scalar multiplication operation.

In cryptanalysis, the ECDLP can be bypassed by exploiting several algorithmic and implementation weaknesses termed as side channel attacks (SCA). For example, if one can have somehow gain access to a cryptographic device, then he may be able to reveal the secret scalar d by monitoring timing and power consumption profiles of the device. Simple and most common SCAs are based on the timing and simple power analysis. Therefore, this research work also adopted the most common techniques to resist the timing and simple power analysis attacks.

Figure 1.1 demonstrates that several performance evaluation metrics are interrelated, thus enhancing one of these can affect the others. For example, increasing performance by improving speed (reducing computation time) may increase area, power consumption, and cost requirements, thus, it is very difficult to achieve all design goals at the same time. Therefore, it is important to evaluate different designs optimized to achieve different performance metrics. In this work, we are more focused on flexible and high performance (in terms of computation time) designs without significant increase in area as compared to other contemporary EC scalar multiplier designs.

1.3 Thesis Contributions

The contribution of this research work is mainly comprised of efficient hardware architectures for finite field arithmetic primitives including addition, subtraction, multiplication, inversion, and division. Based on these optimized finite field arithmetic primitives, high performance hardware architectures for elliptic curve scalar multiplication over a general prime field are presented. The presented hardware architectures for modular addition, modular subtraction, and modular inversion/division operations are considered as minor contributions while the major contributions of the work are:

- Radix-4 and Radix-8 Booth Encoded Interleaved Modular Multipliers
    - The bit serial interleaved multiplication algorithm is modified using radix-4, radix-8 and Booth encoding techniques. The modified radix-4 and radix-8 interleaved multipliers can reduce the number of clock cycles required for one modular multiplication by 50% and 66%, respectively as compared to the bit serial interleaved multipliers while maintaining a competitive critical path delay. Through efficient use of optimized carry chains available in

FPGAs and through exploiting the parallelism among operations, the proposed radix-4 and radix-8 Booth encoded multipliers can compute a 256-bit modular multiplication in 1.48μs and 1.24μs respectively, which are 26.6% and 39% improvement over the corresponding bit serial interleaved multiplier. A thorough comparison of the radix-4 and radix-8 Booth encoded interleaved multipliers with the bit serial interleaved multipliers shows that the proposed radix-4 and radix-8 interleaved multipliers are optimized for a high throughput rate.

- Parallel Radix-4 and Radix-8 Interleaved Modular Multipliers

  - This part of the work presents radix-4 and radix-8 parallel interleaved modular multipliers with their efficient hardware architectures. The introduced parallelism helps to execute the critical operations concurrently while radix-4 and radix-8 techniques are incorporated to reduce the iteration count which determines the required number of clock cycles. It is also observed that incorporating Booth encoding logic in the parallel interleaved multipliers can reduce area cost with a slight degradation in the maximum achievable frequencies. The proposed radix-4 and radix-8 parallel interleaved multipliers are implemented in Verilog HDL and synthesized targeting virtex-6 FPGA platform using Xilinx ISE 14.1 Design suite. The radix-4 parallel interleaved multiplier computes a 256-bit modular multiplication in 0.78 μs, occupies 1985 slices, at 166 MHz in a cycle count of $\lfloor n/2 \rfloor + 5$. The radix-8 parallel interleaved multiplier performs the same bit length operation in 0.69 μs, occupies 3622 slices, achieves 123.43 MHz frequency in a cycle count of $\lfloor n/3 \rfloor + 4$. The implementation results further reveal that incorporating Booth encoding logic in the radix-4 and radix-8 parallel interleaved multipliers can save 18% FPGA slices without any significant performance degradation.

- High Performance Elliptic Curve Scalar Multiplier Architectures

  This part of the thesis presents efficient hardware architectures to compute EC scalar multiplication operation in affine and standard projective coordinates. On the top level the double-and-add (DA) method and the double-and-always-add (DAA) method for EC scalar multiplication are used. In affine coordinates low

level field operations required to perform EC group operations are modular addition, subtraction, multiplication and inversion/division. In the case of projective coordinates, to compute the EC group operations only modular addition, subtraction and multiplication operations are required. Strategies to perform these low level field operations are presented in Chapters 3 and 4. The design of EC scalar multiplier architectures making use of the low level field operations are described below.

- Using the double-and-add (DA) method one can not perform EC group operations in parallel, as there is very limited scope of parallelism in the low level field operations in affine coordinates. Therefore an arithmetic unit in this case incorporates a single modular adder/subtractor, multiplier and divider units.

  On the other hand using the double-and-always-add (DAA) method, EC group operations can be performed concurrently, so dual instances of the arithmetic unit are used in the design of a high speed elliptic curve scalar multiplier architecture in affine coordinates. The proposed architecture for elliptic curve scalar multiplier in affine coordinates is synthesized targeting Virtex-6 FPGA platform for various different field sizes. In the case of a single arithmetic unit, it computes a 256-bit elliptic curve scalar multiplication in 2.51 *ms* in 330K clock cycles and consumes 4807 Virtex-6 FPGA slices. Whereas in the case of two arithmetic units it takes 1.75 *ms*, 229.37K clock cycles to compute the same bit length operation and consumes 9213 Virtex-6 FPGA slices. The presented EC scalar multiplier architecture using two arithmetic units also provides a resistance to timing and simple power analysis attacks.

- Using standard projective coordinates one can eliminate field inversion/-division operation in the computation of EC group operations at the cost of more field multiplications. This part presents a high performance hardware architecture to compute a EC scalar multiplication operation using projective coordinates. It shows that using projective coordinates there are many possibilities of parallelism in the underlying field operations, therefore the performance of the presented EC scalar multiplier architecture in projective coordinates is shown by employing a number of parallel multiplier units.

On the system level again the same algorithms (standard DA and DAA) are adopted as in the case of affine coordinates. On Virtex-6 FPGA platform using four parallel multipliers, a 256-bit EC scalar multiplication operation is completed in 1.46 *ms* and consume 11.65K slices. The results show that the proposed EC scalar multiplier designs offer significant improvements in the computation time with significant reduction in the required number of clock cycles as compared to the other reported designs. Therefore, the presented EC scalar multipliers are useful for many ECC based schemes.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

1. **Chapter 2** gives a brief introduction to the mathematical background of finite field and elliptic curve cryptography. It also lists different algorithms and some of the common optimization techniques to compute EC scalar multiplication operations using layered hierarchical implementations. Finally, the chapter also gives a basic introduction to FPGA architecture.

2. **Chapter 3** first presents hardware architectures for modular addition, subtraction and inversion/division operations. Then, the radix-4 and radix-8 Booth encoded interleaved modular multipliers are presented with their hardware architectures.

3. **Chapter 4** details further optimization of the modular multiplier architectures presented in Chapter 3. Then, it presents a detailed performance evaluation of the proposed interleaved modular multipliers compared to related contemporary designs on the basis of area, speed, throughput and flexibility.

4. **Chapter 5** presents high performance elliptic curve scalar multiplier architectures by utilizing the hardware architectures of finite field arithmetic primitives proposed in Chapters 3 and 4. The performance of the proposed elliptic curve scalar multipliers are also compared with contemporary designs in the literature.

5. **Chapter 6** is devoted to possible future research directions and it also presents an overall conclusion of the thesis work.

Chapter 2

Background

This chapter briefly introduces the background and mathematical tools that are of prime importance in elliptic curve scalar multiplication. First, some basic concepts of different cryptographic schemes with their recommended key sizes are introduced. Then, an introduction to finite field and elliptic curve arithmetic over prime field is presented. Subsequently, implementation strategies of elliptic curve crypto schemes at different levels of implementation hierarchy are discussed. Finally, an introduction to FPGA is given in the last section of this chapter.

All cryptographic encryption/decryption methods, irrespective of their applications can be categorized into symmetric or asymmetric key algorithms. Symmetric key algorithms are sometimes called Private-Key cryptography, whereas asymmetric key algorithms are widely referred to Public-Key cryptography (PKC).

2.1 Symmetric-Key Cryptography

Symmetric or Private-Key algorithms are a class of algorithms which use a single key for encryption and decryption purposes, therefore the key used for these tasks should be kept secret and must be communicated securely among the participants prior to any communication. Encryption/Decryption tasks using symmetric key algorithms are fairly simple as shown in Figure 2.1, where Alice sends her message (plaintext)

after doing encryption with a key, which must be available to Bob as well. Bob after receiving "ciphertext" (encrypted message from Alice) decrypts it with the same key and recovers the original plain text message. Security of these systems depends on

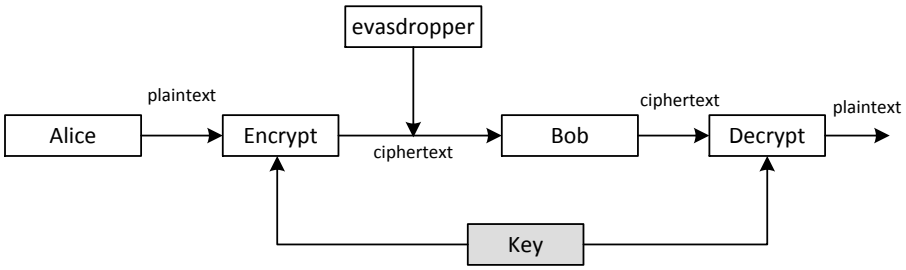


Figure 2.1: Symmetric-Key encryption/decryption

how securely the key is managed and transmitted among users in a communication network. The private key algorithms are efficient and easy to implement but there are certain drawbacks as well. The first problem is that each party must have this secret key before any secure communication between them, in other words the secret key must be securely shared among all parties involved in the communication. The second problem refers to the key management issues, because a communication in a group of n parties would require $n(n-1)/2$ keys, so these keys should be kept secure and must be changed regularly to avoid any security breaches.

Symmetric-Key cryptographic algorithms are further classified into block and stream ciphers. Block cipher algorithms operate on blocks of input data and produce the corresponding output blocks, where as in stream cipher very small chunks (can be a single bit) of input data are fed into the algorithm to obtain the corresponding small chunks of output data. Digital Encryption Standard (DES) [4] and Advanced Encryption Standard (AES) [5], [6] are the well known block ciphers schemes, whereas RC4 [7] is an example of stream ciphers.

2.2 Public-Key Cryptography

Whitfield Diffie and Martin Hellman introduced Public-Key cryptography (PKC) in 1970 [8]. After its inception PKC has solved many problems and enabled the creation of many new interesting protocols considered to be impossible with Symmetric-Key cryptography [9]. PKC algorithms use a pair of keys (public key, private key) for encryption and decryption tasks. A user, Bob, generates his key pair ($p_b$key, $s_b$key), $p_b$key

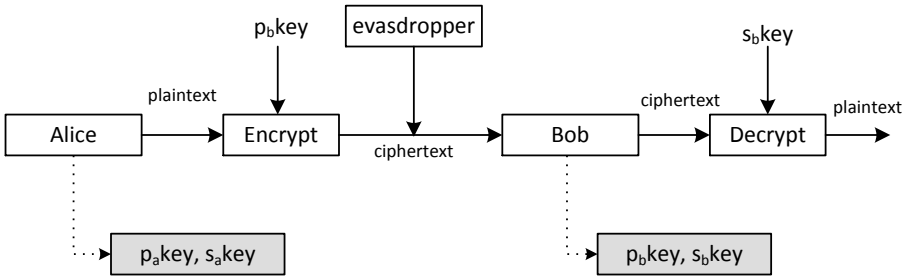


Figure 2.2: Public-Key encryption/decryption

is his public key while $s_b$key is his private key. He publishes the $p_b$key and securely stores his $s_b$key. If Alice wants to send a message to Bob, she needs to encrypt the message with Bob's public key, i.e, $p_b$key. Bob on receiving the ciphertext decrypts it with his private key ($s_b$key) to recover the message as shown in Figure 2.2.

The pair of keys are related in such a way that from the knowledge of one to infer the other is a mathematically intractable problem. Ideally to generate a public key from a private key is based on a one way function. The one way function, as its name suggests, is easy to compute in one direction and is completely infeasible to reverse the operation. Different PKC schemes can be constructed based on different one way functions.

Ronald Rivest, Adi Shamir, and Len Adleman in 1978, proposed a very popular crypto-system which is widely known as RSA [3]. Since its appearance, RSA has been adopted and used widely in many applications and communication networks. Theoretically, the security of RSA crypto-system is based on the mathematical problem (one way function) of integer factorization. RSA has been a dominant Public key system for many years, but with the rapid increasing of the number of resource constrained devices connected to the Internet, a more compact public-key system is required. In 1985 Victor Miller [1] and Neal Koblitz [2] proposed elliptic curve cryptography (ECC) which requires much smaller key sizes as compared to RSA and is discussed in the next section.

Cryptography is the fundamental tool to secure sensitive data. However, efficient implementations of cryptographic algorithms are required to meet speed requirements in high-speed networks. The high processing rate enables cryptographic algorithms to fully utilize the available network bandwidth. The implementation must also be flexible and upgradeable in the field to the rapid changes in algorithms and standards. Therefore, FPGA as an underlying implementation platform provides software-like

Table 2.1: NIST Gudielines for Key Sizes 2012 [10, 11]

Date	Minimum Strength	Symmetric Algorithms (AES)	RSA	ECC	ECC : AES	RSA: ECC
2010	80	2-key triple-DES	1024	160	2:1	6.4:1
2011-230	112	3-key triple-DES	2048	224	2:1	9.14:1
>2030	128	AES-128	3072	256	2:1	12:1
>>2030	192	AES-192	7680	384	2:1	20:1
>>>2030	256	AES-256	15360	512	2:1	30:1

flexibility and hardware-like performance. FPGA based security protocols can be deployed in many critical embedded systems such as wireless networks, electronic banking, electronic commerce, government online service and Virtual Private Networks (VPNs).

Mostly PK algorithms such as RSA and ECC are deployed in hybrid schemes, where they are used to design different protocols e.g. key exchange, digital signature, etc., while normal encryption/decryption tasks are achieved using symmetric key algorithms such as AES and DES due to their simplicity. However, PK algorithms can be used for encryption/decryption. This work sees ECC based cryptographic schemes being deployed in a hybrid scenario.

2.3 Cryptographic Key Sizes

To ensure cryptographic schemes are secure against different attacks, different recommendations have been made and updated with time to overcome known weaknesses of the cryptographic systems [12]. In symmetric key cryptography the key sizes directly determines the level of security. Nowadays AES is considered to be a benchmark among symmetric schemes, while RSA is considered as a benchmark in asymmetric schemes.

Table 2.1 and Table 2.2 demonstrate two recommendations by the US National Institute of Standards and Technology (NIST) and ECRYPT II, respectively [13]. The NIST [10] recommendations in Table 2.1 suggest that 112-bit symmetric key sizes are enough for up-to 2030 after that 128-bit symmetric key sizes are recommended. As in symmetric key algorithms key sizes directly determine the level of security, therefore

Table 2.2: ECRYPT II Recommended key sizes 2012 [14]

Date	Symmetric Algorithms	RSA	ECC	Hash	ECC : AES	RSA: ECC
Protection upto 2015	80	1024	160	160	2:1	6.4:1
Short-term Protection (2015-2020)	96	1176	192	192	2:1	6.13:1
Medium-term protection (2015-2030)	112	2432	224	124	2:1	10.9:1
Long-term protection (2015-2040)	128	3248	256	256	2:1	12.69:1
Foreseeable future	256	15424	512	512	2:1	30.1:1

AES-128 would be required to provide a minimum security after 2030. Similar conclusions can be drawn from ECRYPT II recommendations given in Table 2.2, where short-term, medium-term, and long-term key sizes recommendations are listed.

As mentioned before for asymmetric schemes RSA is considered as a benchmark. In both Tables 2.1 and 2.2, it is recommended that to achieve 128-bit AES security level, RSA needs to have more than three thousand bits, more precisely 3072 [10] (NIST recommendation) and 3248 [14] (ECRYPT II recommendation). It is also recommended that to achieve the same 128-bit AES security level, the required key sizes in ECC is only 256-bit which is 12 times smaller than RSA key sizes. This difference in required key sizes is even bigger at higher security level, 20 times smaller ECC key sizes than the corresponding RSA at 192-bit AES security level. For 256-bit AES security level, ECC key sizes are 30 times smaller than RSA.

It is worth noticing that the ECC key sizes are only twice the symmetric key sizes while these are much smaller than the traditional asymmetric schemes such as RSA. Smaller key sizes translate into lower implementation cost, higher performance, lower power consumption, lower bandwidth requirements, and many other benefits. Therefore, ECC will play a very important role in secure communications in resource constrained devices in the near feature.

2.4 Finite Field

Finite field is the fundamental of cryptography, coding theory, and many other areas of mathematics and computer science. This section describes some basic definitions and then arithmetic operations in a finite field are discussed [15].

2.4.1 Groups

A Group concept is extensively used in modern cryptography. A Group G consists of set of elements and an operator $*$. When the operator is applied on the elements of G, it satisfies the following properties:

- The group is closed with respect to operator $*$, i.e., $\forall a, b \in G, a * b = c \in G$.
- **Associative law :** $a * (b * c) = (a * b) * c, \forall (a, b, c) \in G$.
- **Identity law :** $a * 1 = 1 * a = a, \forall a \in G$.
- **Inverse law :** $a * a^{-1} = a^{-1} * a = 1, \forall a \in G$.
- **Commutative law :** $a * b = b * a, \forall a \in G$

2.4.2 Rings

A ring R is an algebraic structure, in which elements can be added and multiplied while satisfying the following properties:

- **Commutativity :** $\forall a, b \in R$, $a + b = b + a \in R$
- **Associativity :** $\forall a, b, c \in R$, $(a+b)+c = a+(b+c) \in R$; $(a \times b) \times c = a \times (b \times c) \in R$
- **Distributivity :** $\forall a, b, c \in R$, $a \times (b + c) = (a \times b) + (a \times c)$
- **Additive identity :** An element 0 in R such that $a + 0 = a \ \forall a \in R$.
- **Multiplicative identity :** An element 1 in R such that $a \times 1 = a \ \forall a \in R$.
- **Additive inverse :** An element a_1 in R such that $a + a_1 = 0 \ \forall a \in R$.

Examples of rings are integer numbers, the rational numbers, the complex numbers and the real numbers. A number in a ring is said to have a multiplicative inverse if there is a unique element $b \in R$ such that $a \times b = b \times a = 1$. Then the element b is a multiplicative inverse of a.

2.4.3 Finite Fields

A field is a commutative ring that has multiplicative inverse for all non-zero elements. A field is a set equipped with arithmetic operations such as addition, subtraction, multiplication and division, while satisfying commutative, associative and distributive properties.

A finite field also called a Galois field is a field which 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 prime p then the field is called a prime field. A more formal definition of finite field and its properties are given below.

A finite field consists of a set F together with two operations i.e, addition (denoted by $+$) and multiplication (denoted by $\times$), such that it satisfies the following arithmetic properties:

1. $\forall a, b \in F$, $a + b \in F$ **and** $a \times b \in F$
2. $\forall a, b \in F$, $a + b = b + a$ **and** $a \times b = b \times a$
3. $\forall a, b, c \in F$, $a \times (b + c) = (a \times b) + (a \times c)$
4. $\forall a, b, c \in F$, $(a + b) + c = a + (b + c)$ **and** $(a \times b) \times c = a \times (b \times c)$
5. $\exists 0, 1 \in F$, $(a + 0) = (0 + a) = a$, $(a \times 1) = (1 \times a) = a$. Then, 0, 1 are additive and multiplicative identities of the group respectively.
6. $\forall a \in F, \exists (-a) \in F$ such that $(a + -a) = (-a + a) = 0$
7. $\forall a \in F, \exists a^{-1} \in F$ such that $a \times a^{-1} = a^{-1} \times a = 1$. Then, a^{-1} is called a multiplicative inverse of a.

As an algebraic structure every field is a commutative ring with an additional property of a multiplicative inverse for non zero elements, however, every ring may not be a field. The smallest set of finite fields are defined on characteristics 2 and 3 and are denoted as $\mathbb{F}_2$ ($GF(2)$) and $\mathbb{F}_3$ ($GF(3)$). In this work, finite fields defined over a large prime characteristic p are used and described as $GF(p)$ or $\mathbb{F}_p$. The number of elements in a finite field is called its order.

This research work is focused on elliptic curve cryptography over prime fields. In this case $\mathbb{F}_p$ or $GF(p)$ consists of all integers $\{0, 1, 2, ..., p-1\}$, where arithmetic operations are performed on integers modulo p.

2.4.4 Prime Field Arithmetic

This section describes the arithmetic operations over prime field $\mathbb{F}_p$. There are different strategies to compute these operations efficiently [16], [15]. Efficient techniques to compute finite field arithmetic operations are described in Chapter 3. However, a general description of these operations is described here as follows:

- $\mathbb{F}_p$ **Addition:** Given $a, b \in \mathbb{F}_p$, compute $(a+b)$ and $(a+b-p)$. Output $=$ $(a+b-p)$ if $(a+b) \geqslant p$, else output$=$ $(a+b)$.

- $\mathbb{F}_p$ **Subtraction:** Given $a, b \in \mathbb{F}_p$, compute $(a-b)$ and $a-b+p$. Output $=$ $(a-b+p)$ if $(a-b) < 0$, else output$=$ $(a-b)$.

- $\mathbb{F}_p$ **Multiplication:** Given $a, b \in \mathbb{F}_p$, compute z=$(a \times b)$ mod p, where z is the remainder of dividing $(a \times b)$ by p.

- $\mathbb{F}_p$ **Inversion:** For a given non zero element $a \in \mathbb{F}_p$, a multiplicative inverse exists, if and only if a and p are relatively prime i.e, $gcd(a, p) = 1$, then compute $z = a^{-1}$ mod p, where z is a unique integer in $\mathbb{F}_p$ such that $(a \times z)$ mod $p = 1$

- $\mathbb{F}_p$ **Squaring:** For a given $a \in \mathbb{F}_p$, compute z=$(a^2$ mod $p)$, which is $\mathbb{F}_p$ multiplication of an operand to itself.

2.5 Introduction to Elliptic Curves

This work considers an elliptic curve $\mathbb{E}$, defined over prime field $GF(p)$, where p is a large prime characteristic number, then $\mathbb{E}$ is defined as a set of points (x, y), with elements in $GF(p)$ and the curve equation in short Weierstrass form [15, 16] is represented as

$$\mathbb{E} : y^2 = x^3 + ax + b \tag{2.1}$$

Where, a, b, x, and $y \in GF(p)$ and $4a^3 + 27b^2 \neq 0$ (modulo p). The set of all points (x, y) that satisfy (2.1), plus the point at ∞ (infinity) make an abelian group. The number of points on the curve is called the $order$ of the curve. EC point addition and EC point doubling operations over such groups are used to construct many elliptic curve crypto-systems.

2.5.1 Elliptic Curve Scalar Multiplication

The main operation in all EC cryptographic schemes is the multiplication of a point on an elliptic curve with a scalar (an integer). It is also known as point multiplication and is given as

$$Q = dP \tag{2.2}$$

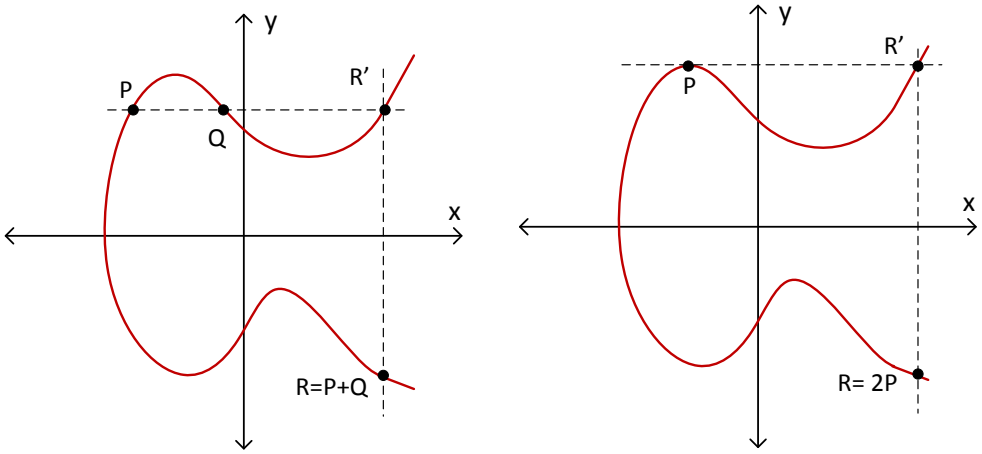
Where d is a scalar value, P, Q are points on a same elliptic curve. The operation

$$dP = \underbrace{P + P + P + ... + P}_{d \text{ times}}$$

can be achieved by $d - 1$ repeated point additions. All ECC based protocols need to compute this dP, hence it is the central operation in all ECC schemes. It is a one way function where a forward computation i.e, dP is easy, but to calculate d from the given Q and P is computationally hard. It is called the elliptic curve discrete logarithm problem (ECDLP). Thus, mathematically, the security of elliptic curve cryptosystems depends on the hardness of ECDLP which is defined as

For a given elliptic curve E defined over $\mathbb{F}_p$, a point $P \in E(\mathbb{F}_p)$ of order r, and a second point $Q \in E(\mathbb{F}_p)$, ECDLP is to determine the integer $d \in [0, r-1]$ such that $Q = dP$

ECDLP is the heart of elliptic curve cryptography. The security of any cryptosystem defined over elliptic curves depends on the hardness of the ECDLP problem. It is



(a) Point addition (b) Point doubling

Figure 2.3: EC group operations

believed to be stronger and harder than the other problems such as the integer factorization problem, which is the foundation of RSA cryptosystems. As a consequence, it is expected that the key sizes of a cryptosystem defined over ECC, using a suitable chosen elliptic curve and underlying field for a given security level are significantly smaller than those cryptosystem defined over RSA as demonstrated in Tables 2.1 and 2.2.

Given a large d, it is not feasible to compute dP through repeated EC point addition (PA) operation. Therefore, another special group operation of adding a point to itself is defined and called EC point doubling (PD). There are many EC scalar multiplication algorithms discussed in [16] [15], EC PA and PD are the two main basic operations in all of these algorithms.

2.5.2 Elliptic Curve Group Operations

EC scalar multiplication is computed by a series of EC PD and PA operations. The EC PA operation is an addition of two distinct EC points P with coordinates (x_1, y_1) and Q with coordinates (x_2, y_2).

The geometrical interpretation of the EC PA operation on EC is shown in Figure 2.3(a), where a line is drawn passing through the two given points P and Q. The line intersects the curve at a third point R'. The point R' when reflected along the x-axis results in a point R, which is the resultant point of the EC PA operation. Let (x_3, y_3) be

the coordinates of R, then the mathematical interpretation of Figure 2.3(a) is given as

$$x_3 = \lambda_{PA}^2 - x_1 - x_2 \tag{2.3}$$

$$y_3 = \lambda_{PA}(x_1 - x_3) - y_1 \tag{2.4}$$

$$\lambda_{PA} = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{2.5}$$

Similarly, in the EC PD operation, a tangent line to the curve is drawn at the given point P. The tangent line intersects the curve at point R'. The reflection of point R' along the x-axis is the resultant point of the EC PD operation, i.e., $R = 2P$, as shown in Figure 2.3(b). A mathematical translation of this procedure is given as follows:

$$x_3 = \lambda_{PD}^2 - 2x_1 \tag{2.6}$$

$$y_3 = \lambda_{PD}(x_1 - x_3) - y_1 \tag{2.7}$$

$$\lambda_{PD} = \frac{(3x_1^2 + a)}{2y_1} \tag{2.8}$$

Note that, the only difference in the computation of EC PD and PA operations are their respective λ values as given in equations (2.5) and (2.8) . It is also worth mentioning that EC points with two coordinates (x, y) is called affine coordinates representation.

2.5.3 Order of an Elliptic Curve

All the points in F_p that satisfy the equation (2.1) plus the point at infinity ∞ forms the elliptic curve group and is denoted as $\mathrm{E}(F_p)$. Each group is comprised of a finite number of elements. The total number of points on the curve, including the point ∞, is called the order of the curve. The order of the curve is usually denoted as $\#E(F_p)$. The upper and lower bounds of the order of the curve can be approximated by Hasse's theorem described as follows:

Let $\#E(F_p)$ be the number of points in $E(F_p)$, then, it is

$$p + 1 - 2\sqrt{p} \leq \#E(F_p) \leq p + 1 + 2\sqrt{p}$$

Table 2.3: Implementation Hierarchy of ECC Based Crypto Schemes

Leyers	Operations
4	EC curve crypto schemes (key exchange, digital signature, etc)
3	EC scalar multiplication
2	EC group operations (PA, PD)
1	Finite Field arithmetic primitives $GF(p)$ addition, subtraction, multiplication, inversion/division

The interval $[p+1-2\sqrt{p} \leq \#E(F_p) \leq p+1+2\sqrt{p}]$ is known as the Hasse interval [17, 18] . Since $2\sqrt{p}$ is very small relative to p, therefore $\#E(F_p) \approx p$. However, the Schoof algorithm [19] is an efficient way to find the exact number of points on an elliptic curve. Similarly, the order of a point is described as follows.

For any point P on elliptic curve E over F_p, there is a small positive integer r such that $rP = \infty$, then r is called the order of point P. The order of any point always exists and divides the order of the curve $\#E(F_p)$.

2.5.4 EC Crypto Schemes Implementation Hierarchy

Typical implementation hierarchy of EC based cryptographic schemes is shown in Table 2.3. It is divided into four layers. Top layer consists of ECC based cryptographic protocols such as key exchange [20], EC digital signature algorithm (ECDSA) [21], secure shell (SSH) [22], transport layer security (TLS) [23], Bitcoin [24], etc. An interested reader is referred to [25]. The next layer is the EC scalar multiplication operation which is comprised of two EC group operations: EC PD and PA operations. Further down, these EC group operations consist of finite field arithmetic operations including modular addition, subtraction, multiplication and division. These finite field primitives are the fundamental arithmetic operations, therefore they have a strong impact on the overall crypto-system performance. The next section describes a simple key exchange protocol based on ECC to illustrate the operations in elliptic curve cryptography.

2.5.5 Diffie-Hellman Key Exchange

Diffie-Hellman key exchange method provides an ability to transfer keys securely over an insecure channel without compromising security of the encryption process. In a

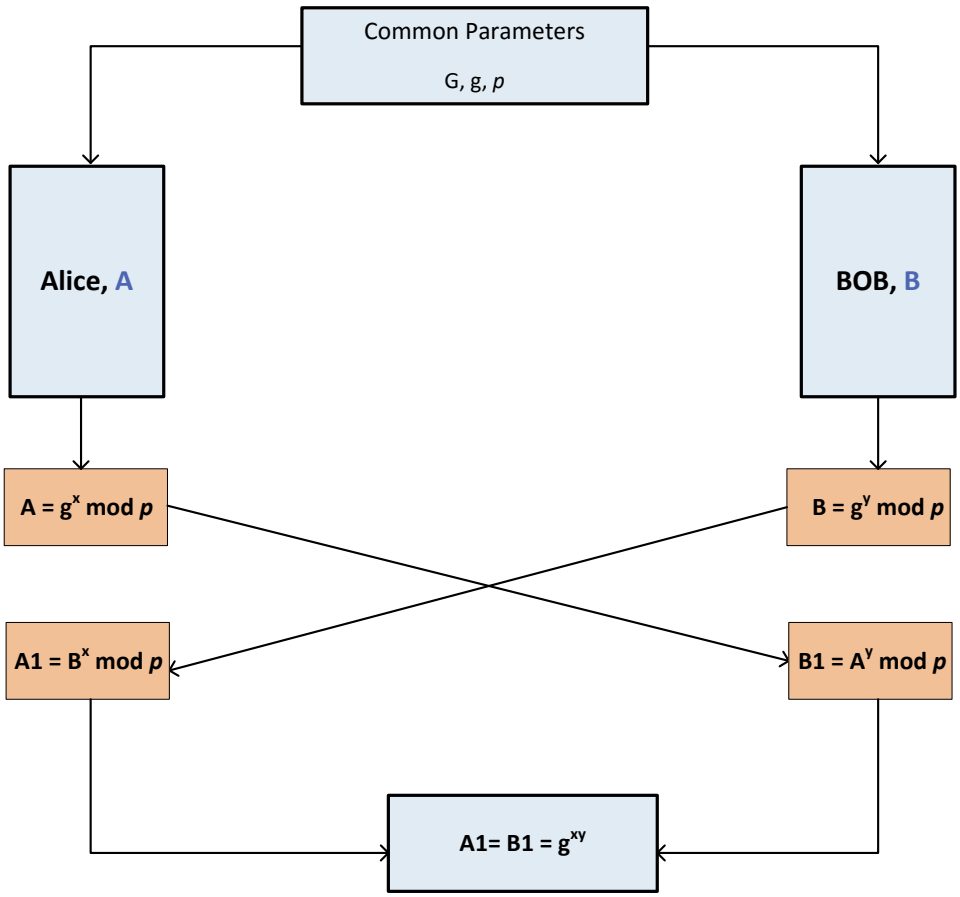


Figure 2.4: Diffie-Hellman key exchange scheme

finite field, the Diffie-Hellman key exchange methods involves several steps show in Figure 2.4 and described as follows.

- First Alice and Bob has to agree on common parameters (G, g, p), where G is group with generator g and a prime p.
- Alice computes $A = g^x \bmod p$, for a random chosen $x \in [1, p-1]$ and sends A to Bob.
- Bob computes $B = g^y \bmod p$, for a random chosen $y \in [1, p-1]$ and sends B to Alice.
- Alice computes $A1 = B^x \bmod p$.
- Bob computes $B1 = A^y \bmod p$.

Since $A1$ and $B1$ are equal i.e., $g^{xy} \bmod p$, Alice and Bob successfully shared a secret which they can use as an encryption key in the further subsequent communication. An eavesdropper only have A, B, g, p, to find x while knowing g and $g^x \bmod p$ he has to

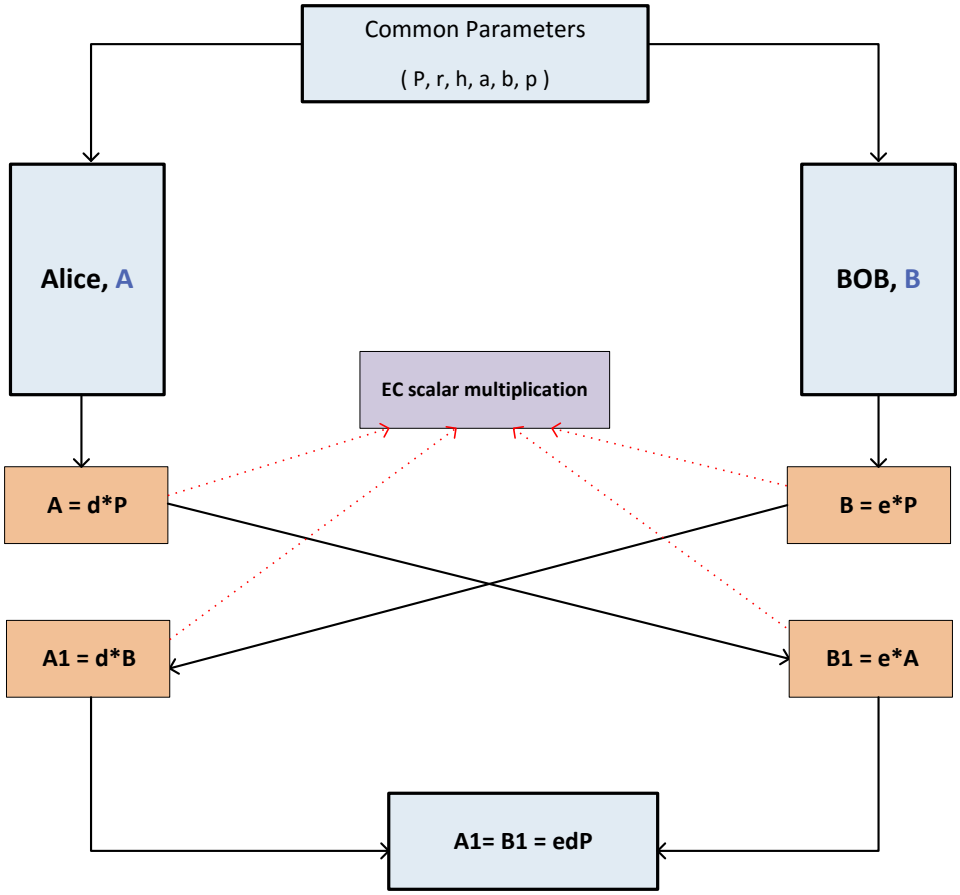


Figure 2.5: EC based Diffie-Hellman key exchange scheme

solve the discrete logarithm problem (DLP), which is not feasible to solve for enough large values of p in a polynomial time.

An elliptic curve version of the Diffie-Hellman key exchange method is known as Elliptic Curve Diffie-Hellman (ECDH) key exchange. It is shown in Figure 2.5 and the steps involved in the ECDH is given as follows.

- First Alice and Bob has to agree on common parameters (P, r, h, a, b, p), where point P is a group generator of order r, h is a cofactor, a and b are elliptic curve constants and p is a large prime.
- Alice computes $A = dP$, for a randomly chosen $d \in [1, r-1]$ and sends A to Bob.
- Bob computes $B = eP$, for a randomly chosen $e \in [1, r-1]$ and sends B to Alice.
- Alice computes $A1 = dB$.
- Bob computes $B1 = eA$.

Since $A1$ and $B1$ are the same i.e., $dB = d(eP) = dA = e(dP)$, both parties agreed on the common secret using the ECDH method. Note that the operations dP, eP, dB, eA are commonly known as an EC scalar multiplication or EC point multiplication. Therefore, it is the fundamental operation in all the protocols based on ECC.

EC version of the DH key exchange protocol is further extended and standardized in [26, 27]. An interested reader is referred to [25] for ECC protocols, deployment, and security.

2.5.6 Standard Projective Coordinates

EC points represented in affine coordinates (x, y) require a modular inversion operation to compute both EC PD and EC PA operations, see equations 2.5 and 2.8. It is the most expensive operation in terms of computation time and resource requirements. In order to speed up the EC group operations, one common optimization is to represent points on an EC in such a way so that inversion free EC PD and PA operations can be computed. Different projective coordinate systems have been explored. These projective coordinate systems have the advantage of eliminating modular inversion from the group operations at the cost of increased number of modular multiplication operations. Typically at the end, one or two inversions are required to re-map from projective to affine coordinates. An overall implementation flow of the EC scalar multiplication operation in projective coordinates is shown in Figure 2.6.

One such coordinate system is called standard projective coordinates. In the standard projective space setting, a point is represented using three coordinates (X, Y, Z). An affine point $P(x, y)$ corresponds to the point $P(XZ^{-1}, YZ^{-1}, Z)$, where $Z \neq 0$ in the standard projective coordinates.

Conversions from affine-to-projective and projective-to-affine spaces are required, however these occur only once during the scalar multiplication operation. An input

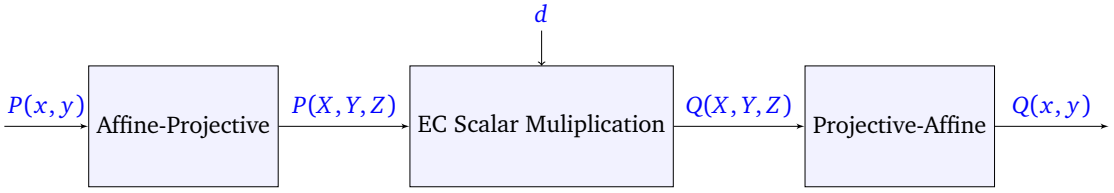


Figure 2.6: EC scalar multiplication in projective coordinates

point for EC scalar multiplication operation is in affine coordinates (x, y) and its projective representation is (XZ^{-1}, YZ^{-1}, Z), therefore affine-to-projective conversion is required which is achieved by setting $Z = 1$ to avoid the conversion cost. Hence, affine-to-projective transformation becomes a trivial process given as

$$(x, y) \longmapsto (X, Y, 1) \tag{2.9}$$

More precisely, given point in affine space (x, y), its standard projective space representation is derived by setting the Z coordinate equal to one, then the other X, Y coordinates are given as

$$(x, y) \longmapsto (X, Y, 1), \quad X = x, \quad Y = y, \quad Z = 1$$

At the end of the EC scalar multiplication operation, the projective-to-affine conversion is required which is achieved as follows:

$$x = XZ^{-1}, \qquad y = YZ^{-1} \tag{2.10}$$

This conversion costs two multiplications and a single inversion.

2.5.7 Jacobian Projective Coordinates

The other commonly used coordinates system is Jacobian projective coordinates, where an affine point $P(x, y)$ is represented as $P(XZ^{-2}, XZ^{-3}, Z)$. Similarly, affine-to-Jacobian transformation is trivial by setting the Z coordinate equal to one.

$$(x, y) \longmapsto (X, Y, 1) \tag{2.11}$$

At the end of scalar multiplication, conversion back to affine space is done as

$$x = XZ^{-2}, \qquad y = YZ^{-3} \tag{2.12}$$

The cost of this conversion is four multiplications and one inversion. A more detailed analysis of EC point operations in standard projective coordinates is presented in Chapter 5 with complete EC scalar multiplier architectures. Further details of the Jacobian

projective coordinates can be found in [28], [29], [16].

2.6 Side Channel Attacks

Algebraic attacks are not the only solution to deduce sensitive information of the cryptosystem. There are many methods to retrieve sensitive information from the physical implementation of a cryptographic device by monitoring some side channel information which are called side channel attacks (SCA) [30].

Theoretically, the security of elliptic curve cryptographic systems relies on the hardness of the ECDLP problem. However, ECDLP can be bypassed by exploiting several algorithmic and implementation weaknesses. For example, if somehow an adversary gets access to a cryptographic device, then the adversary may be able to reveal the secret by observing timing and power consumption information. Timing and simple power analysis (SPA) are the most common and simple side-channel attacks [31]. There are even more sophisticated attacks based on fault injection or differential power analysis [32]. Fan et al. in [33, 34] surveyed most of the side-channel attacks and their countermeasures.

Power analysis side channel attacks are grouped into simple power analysis (SPA) and differential power analysis (DPA) [32]. SPA monitors a single instance of power consumption of a device and tries to deduce the secret information. On the other hand DPA gathers power data of several instances of the device and then statistically analyses the data to reveal the secret information.

To employ any side channel attack, an attacker needs a physical access to a cryptographic device; therefore countermeasures against these attacks are very important in cryptosystems implemented on smart cards. However, this work targets the FPGA as an implementation platform therefore it only considers algorithmic level countermeasures against timing and simple power analysis attacks.

2.7 Related Work

This section reviews the literature of available hardware accelerators for point multiplication on elliptic curves. It outlines some of the proposed designs to establish a

basic understanding of the state-of-the art research in this domain.

2.7.1 Hardware Architectures for EC Scalar Multiplication

As implementation of point multiplication on elliptic curves can be decomposed into several layers, therefore, the overall performance and efficiency could be significantly improved by optimization at different layers, independently. The fundamental or base layer of an EC cryptosystem implementation is the finite field arithmetic operations. There are different design approaches to optimize these field operations which is discussed in chapter 3. Optimized field operations can boost the overall performance of EC point multiplication which is based on EC PA and PD operations which are in turn based on these field operations.

Crypto-systems based on ECC are designed using either elliptic curves defined over the binary extension field $GF(2^m)$ or curves defined over a prime field $GF(p)$. The nature of operations in these fields are quite different from each other. In a binary extension field, elements are described using polynomials and reduction is done using an irreducible polynomial. On the other hand, elements in a prime field are integers and arithmetic operations are done using integer operations modulo a prime p. Therefore, binary field arithmetic imposes completely different design challenges as compared to that in prime field [35]. Typically, field operations over $GF(2^m)$ are very much hardware friendly due to carry free arithmetic. Therefore, design challenges, implementation cost, and performance of ECC processors over binary and prime fields are not comparable, their comparison is misleading and even not possible because of their different underlying field representations. However, due to recent advancements in methods for attacking discrete logarithms, there are some concerns regarding binary curves security. Modern cryptographers tend to avoid binary curves and would like to use prime curves for long-term security. Performance comparison of ECC hardware implementation using binary and prime fields is presented in [36].

This work focuses on point multiplication on elliptic curves over prime field, therefore, the main point of discussion throughout this work is hardware implementations and analysis of ECC over prime field. For references, some of the ECC processor designs over $GF(2^m)$ are reported in [37, 38, 39, 40, 41, 42, 43]. Review of high speed ECC processors over $GF(2^m)$ is reported in [44], [45].

Table 2.4: NIST Recomended Primes

$\|p\|$ size	Numerical value	AES equivalent security level
p_{192}	$2^{192}-2^{64}-1$	96
p_{224}	$2^{224}-2^{96}+1$	112
p_{256}	$2^{256}-2^{224}+2^{192}+2^{96}-1$	128
p_{384}	$2^{384}-2^{128}+2^{96}+2^{32}-1$	192
p_{521}	$2^{521}-1$	256

Several prime field hardware accelerators for elliptic curve scalar multiplication have been proposed during the last fifteen years. These designs can be classified into two categories: designs over standard and designs over general prime fields.

2.7.1.1 EC Scalar Multipliers over Standard Prime Fields

Modular multiplication is the most time critical component in the construction of elliptic curve point multiplication in projective coordinates. One of the optimization techniques is by choosing a prime modulus p of special structure (very close to a power of 2) called pseudo-Mersenne primes, which can reduce the computational complexity of the reduction stage in a modular multiplication operation. In this regard NIST recommends five prime fields [16] for different levels of security given in Table 2.4.

The prime P_{224} supports a security level of AES-112 bits and P_{256} supports a security level upto AES-128 bits which is more than enough in current scenario. Modular multiplication based on each one of the NIST recommended primes is usually fast. However, as the prime modulus is of a special form which results in a fast and rigid dedicated hardware architecture. This dedicated hardware is not flexible to work for any other prime values except for that which it is designed for. A straight forward implementation of any NIST recommended prime is also not scalable, which means a design for P_{224} is not able to work for the P_{192}. Designs reported in [46], [47], [48], [49], [50], [51] are based on NIST recommended elliptic curves over prime fields. Virtex-4 implementation of [46] completes a 256-bit EC scalar multiplication in 6.1 *ms* at 43 MHz clock frequency, occupies 20.1K slices and 32 DSPs blocks (16 × 16 embedded multipliers). Alrimeih et al. in [47] extended the same design to increase performance and side channel attacks resistivity. Its implementation on Virtex-6 computes a 192-bit EC scalar multiplication in 0.3 *ms* and 3.91 *ms* for 512-bit multiplication. It occupies 33K LUTs, 289 DSPs blocks (18 × 18 multipliers),

and 128 RAMB36 (36K random access memory). In [48], two high speed ECC processors are proposed: one design is for NIST prime p_{224} while the other is for the NIST prime p_{256}. The designs are implemented on Virtex-4 FPGA and utilized embedded 16×16-bit multipliers in DSP blocks and built-in RAM. A similar design optimized for NIST prime p_{256} is reported in [52]. Moreover, it is worth noticing that designs in [48], [52] are only optimized for a single NIST recommended prime, whereas [47] is the only available design that supports all NIST primes.

The designs in [50], [49] are efficient residue number system (RNS) implementations of elliptic curve point multiplication over prime field. The designs are structured on a new finite field multipliers, which is designed using Montgomery and RNS techniques. Arithmetic in RNS domain enables carry free computation and hence results in lower computation time. However, arithmetic using RNS domain requires binary-to-residual and residual-to-bianry transformations besides the original operation. The designs exploited NIST primes special structure (not flexible). Moreover, [50] is very vulnerable to timing and simple power analysis attacks while [49] provides resistivity to the mentioned attacks. Bernstein et al. in [53] presents that there are many concerns regarding NIST chosen primes and ECs, therefore it is worth to target general prime field for ECC applications to provide the user more flexibility and security.

2.7.1.2 EC Scalar Multipliers over General Prime Field

Several general prime field ECC processors are also available in the literature. These designs can be classified according to their adopted coordinates systems and reduction techniques. Another optimization technique is to use different number systems to compute a modular multiplication operation as fast as possible for example, the Montgomery number system [54] and the Residue number systems (RNS) [16]. These are useful where the conversion cost before and after an operation does not dominate the overall cost of the operation.

Designs reported in [51, 55, 56, 57] are built using EC points representation in affine coordinates. Ghosh et al. in [56], proposed an elliptic curve scalar multiplier architecture over general prime field resilient to timing and power analysis attacks. The design performed point addition and doubling operations using affine coordinates (x, y), which also requires modular inversion/division operations in addition to the field addition, subtraction, and multiplication. Its arithmetic unit consisted of

two modular addition, two modular subtraction, two modular multiplication, and two modular division units. The Virtex-4 FPGA implementation of the design computes a 256-bit EC scalar multiplication in 7.7 *ms*, cycle count of 330K, runs at a maximum clock frequency of 43 MHz, and occupies 20.1K slices. The design is also resilient against timing and SPA attacks.

[55] proposed a compact programmable arithmetic unit (PAU) to perform finite field arithmetic operations. Then, EC scalar multiplier architecture is presented based on dual instances of the PAU. EC points are represented in affine coordinates, and Montgomery powering ladder method [58, 59] for EC scalar multiplication is adopted to perform point doubling and addition operations in parallel. Its implementation on Virtex-II pro completes a 256-bit EC scalar multiplication in 9.38 *ms*, achieves a maximum frequency of 36 MHz, cycle count of 338K, and consumes 12K slices. The design also provides protection against timing, SPA, and DPA attacks.

The designs reported in [60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72] are based on projective coordinates and most of the designs have adopted the Montgomery modular multiplication technique to perform the modular multiplication operation. A very recent survey on hardware analysis of ECC processors on binary and prime fields are reported in [73].

Moreover, there are many elliptic curve representations offering different trade-offs between computational performance and security [16]. Constructions of new elliptic curves is also an active area of research; their structures, parameters and security is discussed in [74]. Another active research domain is to formulate new coordinate systems with fewer field multiplications to compute EC group operations [28], [29].

Furthermore, the underlying platform also plays an important role in the performance of point multiplication on elliptic curves. The same design implemented on a CMOS customized library would be faster than the corresponding FPGA implementation. However, the FPGA is reconfigurable which means that an existing design can be replaced with a new one on the same FPGA. The other important factor to choose an FPGA as the underlying implementation platform is its lower design cost. In this thesis FPGA is used for implementing EC scalar multiplication and therefore a brief introduction to FPGAs is provided below.

2.8 FPGA Architecture

Field programmable gate arrays (FPGAs) are semiconductor devices that offer in-house programmability to users. The design concept of FPGA is directly opposite to the application specific integrated circuits (ASICs), which are built for particular applications. FPGA based design is more flexible as compared to ASIC design. However

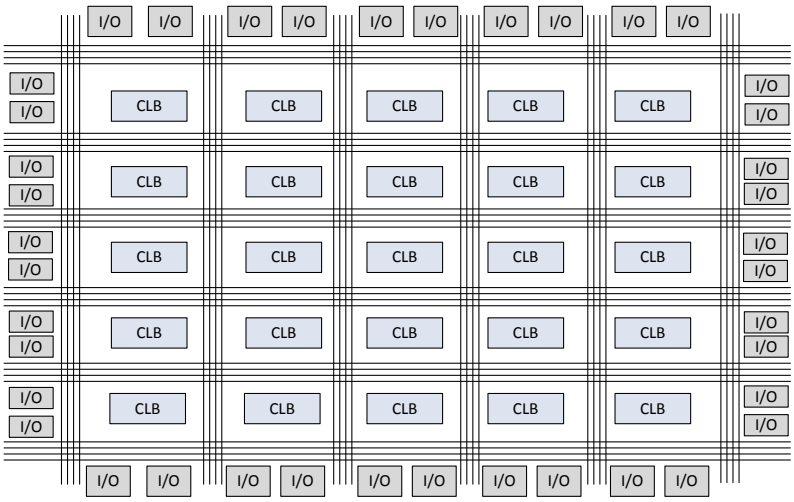


Figure 2.7: A Generic FPGA Architecture [75]

ASIC design yields higher performance and lower power consumption. Therefore, for applications where flexibility and cost are more important than performance (speed), FPGA as an implementation platform is more suitable than ASIC.

A generic FPGA architecture is shown in Figure 2.7. It is a programmable matrix of configurable logic blocks (CLBs). These CLBs can be connected to one another by available horizontal and vertical wires, which behave as programmable interconnects. Input/output (I/O) ports of different capacity and speed are located around the edges to handle I/O signals. Through CLBs, FPGAs can be programmed for different desired functionalities or applications. The reconfigurability of FPGA makes it the most suitable implementation platform for security algorithms, which may need to be updated from time-to-time to avoid many security breaches.

CLB is the fundamental component of a FPGA to implement combinational and sequential circuits. Its internal architecture varies for different vendors and families. Xilinx [76] and Altera [77] are the two well known FPGA vendors that enjoy large market shares. Virtex-6 is considered as a suitable family of Xilinx FPGA devices to achieve high level of performance and functionality of any design. It is selected as the

Table 2.5: Virtex-6 FPGA CLB Internal Resources [78]

Slices	LUTs	Flip-Flop	Carry Chains	DRAM	Shift Registers
2	8	16	2	256 bits	128 bits

targeting device to evaluate the performance of the EC scalar multipliers in this work.

Virtex-6 FPGA [79] is build on a high performance logic fabric of a 40*n*m CMOS technology. Each CLB in a Virtex-6 FPGA consists of two slices organized in column with dedicated carry chain. Each slice consists of the following elements

- Four function generators or look-up-tables (LUTs).
- Eight storage elements.
- Wide-function multiplexers.

Table 2.5 demonstrates logic resources in one Virtex-6 FPGA CLB. Each of the function generators are implemented as six input look-up-table (LUTs). These LUTs can be used to implement any arbitrary six-input boolean functions. In addition to these configurable blocks, modern FPGAs are also equipped with dedicated blocks to perform arithmetic operations. For example, in Virtex-6, DSP48E1 slices are also available to perform signal processing specific tasks. These DSP blocks can be configured to performed a variety of other arithmetic functions. Each DSP48E1 slice is comprised of a 25×18-bit multiplier, an accumulator and an adder. On the other hand Virtex-4 FPGA DSP48E slice consists of a 16×16-bit multiplier and an accumulator. These available small multipliers can be utilized to build large integer multipliers to perform large operand multiplications.

However, a design utilizing these in-built multipliers may not be portable to other FPGA families because of different internal architecture of DSP slices. It may also be less flexible as compared to a design that is build using CLBs only. Therefore, DSP blocks are not used in this research work to add more flexibility and portability in the presented designs.

In this work all the designs are coded in Verilog HDL. Xilinx ISE 14.1 Design Suite is used for synthesis, mapping, placement and routing purposes while Xilinx ISE simulator (ISim) is used for behavioral simulation and initial design verification.

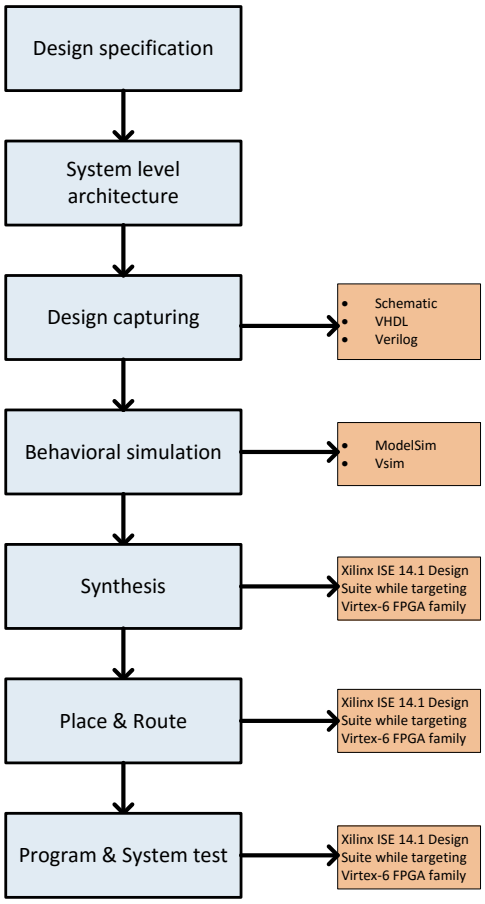


Figure 2.8: Design steps of FPGA implementation

2.8.1 FPGA Implementation Design Flow

Figure 2.8 shows the steps usually followed to implement any design on an FPGA platform. Initially to design any system, a specification is required called design specification. The design specifications are generally presented as a document describing a set of functionalities that the final solution will have to provide and a set of constraint that it must satisfy. In this context, the system level architecture is the initial process of deriving a potential and realizable solution from the design specifications and requirements.

This work first implements a system level architecture of the proposed modular multipliers and EC scalar multipliers in software using C# as an implementation language.

After system level design verification, potential hardware architectures are modelled and designed at Register Transfer Level (RTL) using a Hardware Description Language (Verilog HDL). Then, RTL design verification consists of acquiring a reasonable confidence that a circuit will function correctly, under the assumption that no

manufacturing fault is present. To validate functionality, the RTL design is run and tested on Xilinx ISIM and Modelsim simulators. This step is called the behavioral simulation. After simulation, the proposed architectures are synthesized using Xilinx ISE Design Suite 14.1 targeting Virtex-6 FPGA to check whether designs are suitable for the selected device or not. The report generated by the synthesis tool (ISE Design Suite 14.1) summarizes the initial design implementation results on a selected FPGA device. The report shows area consumption (LUTs, slices, registers), maximum clock frequency (critical path delay), etc. All design and logic errors of the proposed architectures are removed in the synthesis phase. Post place and Route is done for routing hardware thus optimizing hardware (gates), power and latency. After successful validation, the RTL design is ready for implementation on the FPGA.

It is not easy to provide fair performance comparisons of a design implemented on different platforms. However, the analysis presented in [80] shows that an FPGA implementation is approximately 35 times larger and between 3.4 to 4.6 times slower on average compared to an ASIC implementation. In [81] a performance comparison of different cryptographic algorithms implemented on different platforms (FPGA, ASIC, General Purpose Processor) has been presented. The performance comparison shows that ASIC and FPGA implementations are always faster than the software implementations. It is reported that FPGA implementation of a cryptographic algorithm is two times faster than its software implementation.

Implementation results are sensitive to the chosen platform. Comparing ASIC and FPGA, an FPGA LUT has higher delay as compared to an ASIC gate. Also when measured in kilo-gates per square micrometre, ASIC gate density is very high as compared to an FPGA [82]. Results also depend on the synthesis tool used for ASIC and FPGA implementations. FPGA and ASIC tools might support different synthesis directives and options related to a design optimization.

Different FPGA families have different slice architectures. For example, Xilinx Virtex-6 has four LUTs and eight registers. Xilinx Virtex-4 FPGA slice has two LUTs and two registers. Due to this reason, slice logic utilization for the same design will be different for different FPGA families. Also since each family of FPGAs has unique architecture, it will also greatly affect the measured performance [83].

2.9 Conclusion

This chapter introduces the background and mathematical tools that are of prime importance in the design of elliptic curve scalar multiplier. Basic concepts of different cryptographic schemes with their recommended key sizes are introduced first. Then, finite fields and elliptic curve arithmetic over prime field are presented. Next, different implementation strategies of EC scalar multiplication at different levels of its implementation hierarchy are discussed. Finally, FPGA structure is briefly introduced. The discussion of hardware acceleration of finite field arithmetic operations is presented in the next chapter.

Chapter 3

Hardware Architectures for Finite Field Arithmetic

Crypto-systems based on public-key cryptography (PKC) [1], [2], [3], [8] are structured using finite field arithmetic primitives such as modular addition, subtraction, multiplication, and inversion [45]. Among these primitives, modular multiplication and inversion/division are the most computational intensive operations. In fact modular inversion/division are the most tedious and expensive operations as compared to modular multiplication operation. Therefore, alternative ways have been investigated and designed to perform inversion/division free EC group operations. These methods are known as projective coordinates as described in Chapter 2. Therefore using projective coordinates is the most critical field operation is a modular multiplication [16], [15].

This chapter first describes algorithms and design strategies to perform modular addition, modular subtraction, and modular inversion/division operations. Then, it presents two novel modular multiplier architectures based on radix-4, radix-8, Booth encoding and interleaved multiplication techniques. Radix-4, radix-8 and Booth encoding techniques are used to optimize the interleaved modular multiplication algorithm. The optimized radix-4 and radix-8 versions of interleaved modular multiplication algorithms result in 50% and 66% reduction in total number of clock cycles

required to perform a modular multiplication operation. The proposed multipliers do not require any operand and result conversion as required in Montgomery method discussed in the next section. Performance of the presented multiplier architectures is discussed and analysed for different field sizes.

3.1 Background and Related Work

Finite filed arithmetic operations are the fundamental components to construct any EC crypto-systems. Among these components field multiplication, field inversion and field division are more critical than field addition and subtraction due to their inherent computational difficulties. In fact, field inversion/division operations are more expensive in terms of computation time and resource requirements as compared to field multiplication both on hardware and software platforms. Projective coordinates systems enable inversion/division free EC group operations. Thus, the most critical operation in EC group operation in projective coordinates is finite field multiplication. Several techniques have been proposed to speed-up finite field multiplication operation discussed as follows.

The classical method to perform a finite field multiplication of operands a and b over a prime modulus p is defined in equations (3.1) and (3.2).

$$R = a \times b \tag{3.1}$$

$$c = R \bmod p \tag{3.2}$$

It is a two-step process: integer multiplication and reduction modulo p. The reduction modulo p step typically requires a trail division operation which is a very computational intensive operation, therefore many strategies have been proposed to lower the computational intensity of the reduction step. Generally these strategies can be divided into in three main categories [35, 84]: designs over standard primes [85], designs based on Montgomery multiplication method [54] and designs over interleaved multiplication method [86, 87].

In order to lower the computational intensity of the reduction step NIST recommended five specialized primes p of size $(p_{192}, p_{224}, p_{256}, p_{384}, p_{512})$ as given in Table 2.4. These primes have a special structure that are very close to a power of 2 i.e.,

$2^a \pm 2^b \pm 2^c \pm 2^d \pm 1$, and are called pseudo-Mersenne primes. Modular multiplication operation over this type of prime can result in higher performance and lower computational cost. However, a design optimized for a particular modulus value results in a very dedicated architecture, which can not be used for any other prime values, hence the architecture lacks flexibility. A pipelined modular multiplier design reported in [88] can support five NIST recommended primes. Its datapath is comprised of 8 pipeline stages with a latency of 80ns for primes of size 192, 224, 256-bits and 200 ns for 384, 256-bits. It consumes 8340 slices and 259 dedicated DSPs blocks on Virtex-6 FPGA platform, which may not fit into smaller FPGAs, but is suitable for high speed applications. Designs reported in [48], [52] also exploited special structure of NIST primes, p_{224} and p_{256}. These implementations are devoted to p_{224},p_{256} and are not able to provide the flexibility to accommodate other primes, which is one of the main focuses of this thesis.

Montgomery multiplication method converts the required division operation into cheaper shift and addition operations. However, to make use of the Montgomery method operands must be transformed from normal to Montgomery representation to perform operation in the Montgomery domain, and the result must be transformed back to the normal domain to yield the final result of a modular multiplication operation. The method is suitable where the conversion overhead is negligible as compared to the main operation cost, for example in exponentiation algorithms. Montgomery multiplication based designs are reported in [89] and [90], in which [90] is based on radix-4 and [89] incorporates radix-2^{16} techniques. The designs reported in [57], [60], [63], [69], [71] are based on radix-2 implementation. In [91] several possible implementation strategies of Montgomery multiplication are discussed on the basis of performance and implementation cost. Amnor et al in [92] report that radix-2 implementation of interleaved modular multiplication has better area-delay product.

Other interesting hardware implementations of Montgomery modular multiplication are [89], [93], [94] and [95]. Among these [93] presents interleaved modular multiplier based on Montgomery and Barrett reduction techniques, [94] presents time and area efficient modular multiplier. The design in [95] is based on redundant radix-2^{16} while [89] is based on radix-256.

The designs reported in [96], [97] used built-in FPGA Digital signal Processing (DSP) blocks to design 256-bit modular multiplier architecture based on Montgomery

method.

Interleaved multiplication method was proposed by Blakley [86, 87] in 1983. The method is based on iterative addition and reduction of partial products. Partial products accumulation and intermediate results reduction are integrated in a way to eliminate the final division. The idea is to reduce intermediate results below the modulus value in each iteration so that the final division can be avoided. The algorithm starts traversing a multiplier from most-significant-bit (MSB) to least-significant-bit (LSB). Several modifications and hardware architectures have been reported [55], [56], [92], [93], [98], [99], [100], [101], [102]. In [93] a faster interleaved modular multiplier based on Montgomery and Barrett reduction techniques is reported. Its 130-nm ASIC implementation runs at a maximum frequency of 320 MHz and computes one 256-bit modular multiplication in 0.05 us.

Ghosh et al. in [100] reports a radix-2 parallel interleaved modular multiplier. Its Virtex-II Pro FPGA implementation consumes 3475 slices with a latency of 3.2 us and takes n clock cycles to perform an n-bit modular multiplication. The same multiplier is utilized in [99] in construction of a dual core pairing processor. A robust $GF(p)$ parallel arithmetic unit for public key cryptography is reported in [103]. The parallel arithmetic unit can perform modular addition, subtraction, multiplication and inversion/division operations. The arithmetic unit adopted interleaved modular multiplication to perform modular multiplication and extended Euclidean algorithm (EEA) to perform inversion/division operations.

Similarly in [55] a compact programmable arithmetic unit is based on the same algorithms (Interleave multiplication and EEA). The required number of adders is reduced by exploiting hardware sharing techniques, however the unit is not able to execute field operations in parallel and is not suitable for high performance applications. The design in [104] is based on pre-computation, carry save addition and sign estimation techniques. However, it requires carry propagation adder at the final stage.

Montgomery and interleaved multiplication methods are widely used in the design of finite field multiplier. Performance comparison of these methods are discussed and analysed in [105]. The proposed higher radix modular multipliers in this thesis is based on interleaved multiplication method, works directly on numbers in two's complement formats and thus do not require any conversion. Performance of these

Algorithm 1: Modular addition

Input: $a = \sum_{i=0}^{n-1} a_i \cdot 2^i$, $b = \sum_{i=0}^{n-1} b_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $a + b \mod p$

```
1 S ← a + b;
2 if S ≥ p then
3 |   S ← S − p;
4 end
5 return S;
```

modular multipliers is evaluated against bit level implementation of interleaved multiplication method.

The rest of this chapter explains the adopting techniques in this work to perform finite field arithmetic primitives.

3.2 Modular Addition/Subtraction

Modular addition $(a + b) \mod p$ and modular subtraction $(a - b) \mod p$ primitives involve two n-bit adders cascaded in series [35], [84]. In addition to these adders, multiplexing logic is also required at different levels to control the datapath. The critical component is the adder logic due to long carry propagation delay. FPGA in-built fast carry chains (FCC) can be used to speed up the addition operation. This work uses high speed adder based on FCC and carry select approach [99]. Modular addition operation is described in algorithm 1 while modular subtraction is described in algorithm 2.

3.2.1 Modular Addition

Hardware realization of modular addition $(a + b) \mod p$ of two operands a and b is shown in Figure 3.1. The first n-bit adder performs addition of two input operands i.e., $S_1 = a + b$. Then, the prime p is subtracted from the result S_1 by taking two's complement of p i.e., $S_2 = S_1 + (\sim p) + 1$ in the second n-bit adder. These partial results are multiplexed and assigned to the final result S. During the modular addition operation the input signal c_{in} is set to zero.

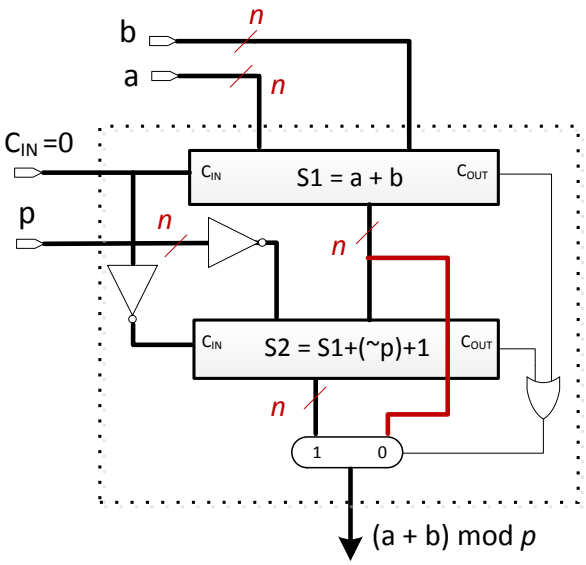


Figure 3.1: Modular addition architecture

Algorithm 2: Modular subtraction

Input: $a = \sum_{i=0}^{n-1} a_i \cdot 2^i$, $b = \sum_{i=0}^{n-1} b_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $a - b \mod p$

1 $S \leftarrow a - b$;
2 **if** $S < 0$ **then**
3 $\quad S \leftarrow S + p$;
4 **end**
5 **return** S;

3.2.2 Modular Subtraction

The architecture in Figure 3.2 performs modular subtraction $(a - b) \bmod p$ of two operands a and b, when the input signal c_{in} is set to one i.e., $(c_{in} = 1)$. The first adder logic performs subtraction $(a - b)$ as $(S_1 = a + (\sim b) + 1)$, where $(\sim b + 1)$ is the two's complement of b (because $c_{in} = 1$). Then, the result is added with a prime p by the second adder and the final result is selected by the carry out signal of first adder which is indication of an underflow.

Modular addition/subtraction operations can be performed by a same architecture as shown in Figure 3.3. The select signal (sel) determine the operation to be performed by the architecture. For example if sel is equal to zero then it outputs $s = (a + b) \bmod p$ otherwise it performs $s = (a - b) \bmod p$ operation. This architecture takes a single clock cycle to produce the results of modular addition or subtraction operation.

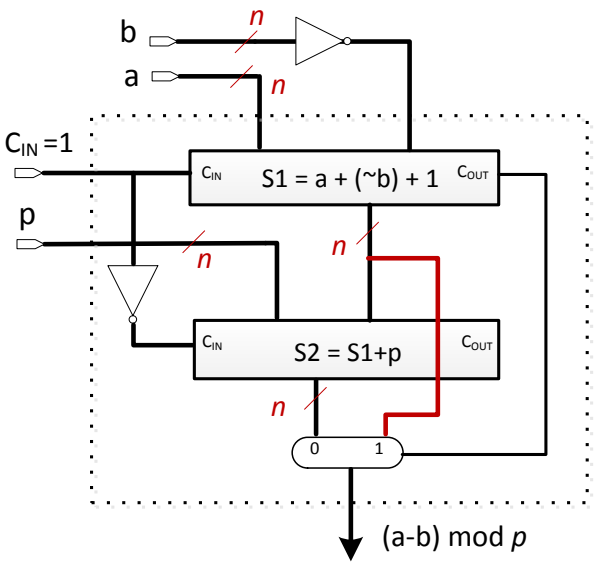


Figure 3.2: Modular subtraction architecture

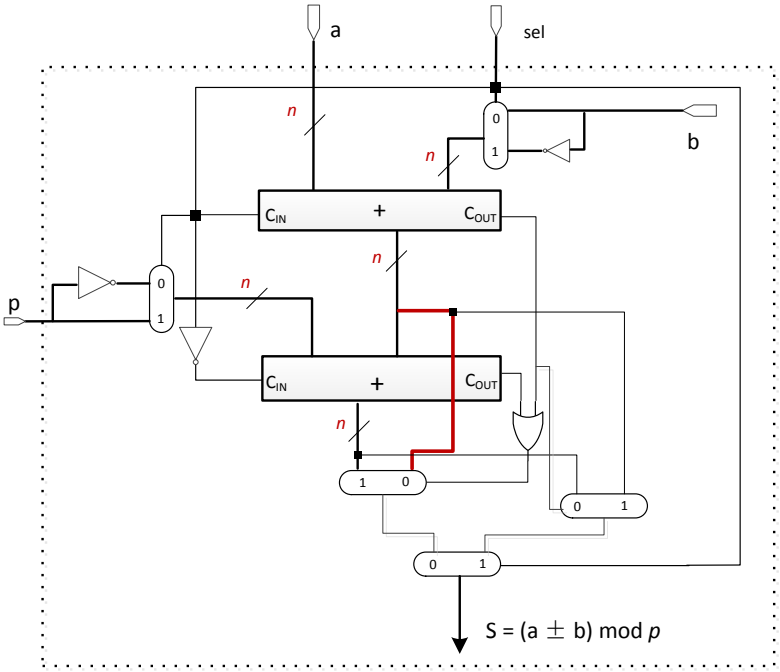


Figure 3.3: Modular addition/subtraction architecture

3.3 Modular Inversion/Division

Modular inversion of a mod p exists if and only if a and p are relatively prime, i.e. when the greatest common divisor of a and p is equal to one i.e. $gcd(a,p) = 1$. Then the modular inversion of a is given in equation (3.3)

$$b = a^{-1} \bmod p \tag{3.3}$$

[**?**]In ECC, usually modular inversion is performed by two methods: Fermat's little theorem and extended Euclidean algorithm [16]. Fermat's little theorem dictates that a^{p-1} **mod** $p = 1$ and therefore dividing both sides by a turns into a^{p-2} **mod** $p = a^{-1}$. By adopting this method an inverse can be calculated by modular exponentiation which requires a large number of modular multiplication operations. Modular division using this method is usually performed by a modular inversion followed by one modular multiplication operation.

Another method to calculate modular inverse is by knowing the greatest common divisor of two integers expressed as a linear combination of two. Since a and p are relatively prime then the following expression may be solved for integers b and t:

$$ab + pt = 1 \bmod p$$

This linear equation implies that:

$$ab \equiv 1 \bmod p$$

Thus, b is the inverse of a mod p and the values of b and t are derived by an algorithm known as the extended Euclidean algortihm (EEA). Kaliski in [106] proposes a variant of EEA which is able to perform Montgomery inverse. This algorithm is useful when the operands are represented in the Montgomery domain.

The binary version of EEA [16] is widely used due to its simpler shift (division by 2) and subtraction operations and is given in algorithm 3. The algorithm is implemented to compute modular inversion/division operations following the guidelines and architectural flow reported in [107], [103]. The algorithm consists of one outer and two inner loops. In the inner loops similar operations are performed on different intermediate signals. At start variables u, v, a_1 and a_2 are loaded with an operand a, a modulus p, one and zero respectively. It is worth mentioning that to compute a modular division (c/a mod p), the variable a_1 must be loaded with c instead of one. The algorithm can be divided into three smaller parts which are given as follow.

1. First inner loop (FIL)
2. Second Inner loop (SIL)
3. Outer loop (OL)

Algorithm 3: Modular Inversion/Division

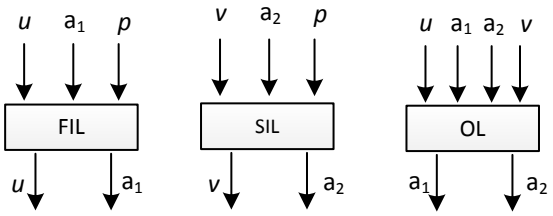
Input: $a = \sum_{i=0}^{n-1} a_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $b = a^{-1} \bmod p$

1 $u \leftarrow a, v \leftarrow p, a_1 \leftarrow 1, a_2 \leftarrow 0$
2 **while** ($u \neq 1$ ***and*** $v \neq 1$) **do**
3 | **while** (u ***is even***) **do**
| | `// First inner loop //`
4 | | $u \leftarrow u/2$
5 | | **if** a_1 **is even then** $a_1 \leftarrow a_1/2$ **else** $a_1 \leftarrow (a_1 + p)/2$
6 | **end**
7 | **while** (v ***is even***) **do**
| | `// Second inner loop //`
8 | | $v \leftarrow v/2$
9 | | **if** a_2 **is even then** $a_2 \leftarrow a_2/2$ **else** $a_2 \leftarrow (a_2 + p)/2$
10 | **end**
| `// Outer loop //`
11 | **if** $u \geq v$ **then** $u \leftarrow u - v, a_1 \leftarrow a_1 - a_2 \bmod p$
12 | **else** $v \leftarrow v - u, a_2 \leftarrow a_2 - a_1 \bmod p$
13 **end**
`// Final step //`
14 **if** $u = 1$ **then**
15 | $b \leftarrow a_1$
16 **end**
17 **else**
18 | $b \leftarrow a_2$
19 **end**

These three loops are shown in Figure 3.4, where outputs v, a_2, u and a_1 are new values at each iteration of the algorithm. FIL step is comprised of steps 4 and 5, SIL is comprised of steps 8 and 9, and OL consists of steps 11 and 12.

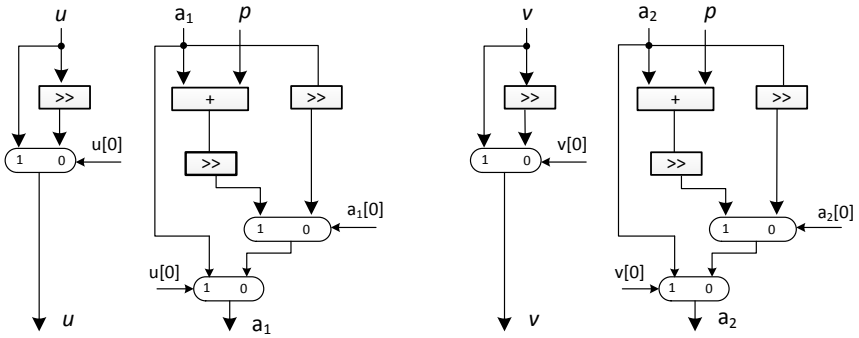
Internal architectures of FIL and SIL are shown in Figure 3.5. The architectures perform exactly identical operations on their respective inputs concurrently. Even or odd signal is determined by the least significant bit indexed with [0]. The value is even if the least significant bit is zero and odd otherwise.

An internal architecture of the OL unit is shown in Figure 3.6. It is comprised of two magnitude (-) and two modular subtractors (-) mod p. The condition $u \geq v$ is checked by borrow out signal of $v - u$ operation. The total number of iterations in the algorithm is $2n$ where n is bit length of modulus p therefore the presented architecture computes n-bit modular inversion/division operations in $2n$ clock cycles. For example, inversion/division in a 256-bit field size are performed in 512 clock cycles.



(a) FIL (b) SIL (c) OL

Figure 3.4: Overall steps in EEA algorithm



(a) FIL (b) SIL

Figure 3.5: FIL and SIL internal architecture

3.3.1 Implementation Results

The cost of division and inversion using the binary version of EEA is the same, which is exactly $2n$ clock cycles. The implementation of modular inversion/division using the EEA technique on the Virtex-6 FPGA platform is given in Table 3.1.

For a 256-bit field size implementation, it takes 3.52 *us*, consumes 5363 LUTs and achieves 149 MHz maximum clock frequency. A drawback of the EEA technique is its implementation cost as compared to Fermat's little theorem. Inversion using Fermat's little theorem can be accomplished using modular multiplication therefore there is no

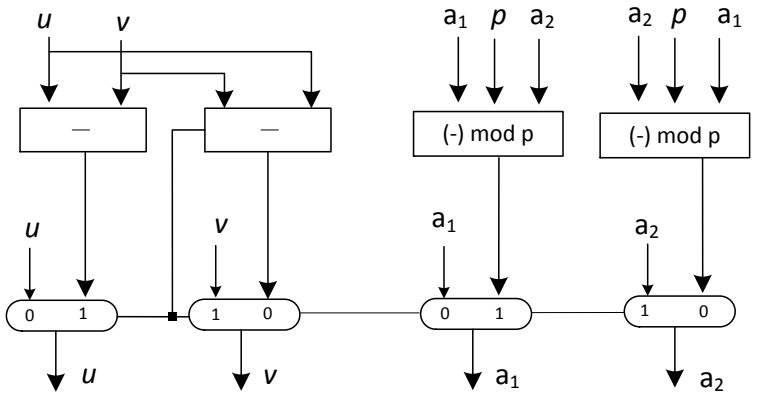


Figure 3.6: OL internal architecture

Table 3.1: Modular inversion/division implementation on Virtex-6

Field size (bits)	Area (LUTs)	Freq (MHz)	clock cycles	Time (us)
192-bits	3931	173	384	2.2
224-bits	4550	159	448	2.83
256-bits	5363	149	512	3.52

need for dedicated hardware for inversion. However, EEA is mostly adopted where performance is more critical than implementation cost. The main focus of this work is on the performance (speed) so EEA is used.

3.4 Modular Multiplication

An interesting algorithm to perform modular multiplication with interleaved reduction is reported in [86], [87] and is known as Interleaved modular multiplication (IM) method. Two main advantages of the IM method are: Unlike the Montgomery method it does not require any operands and result conversions between conventional and Montgomery domain and still it is able to eliminate the final division step. It reduces the intermediate results below the modulus in each iteration to eliminate the final division step. The complete method is given in algorithm 4.

It is based on an iterative addition of partial products $(x \cdot y_i)$ to an accumulator z. In each iteration the contents of accumulator z are single-bit left-shifted and reduced modulo p i.e., $(2z \bmod p)$. In the same iteration the partial product $(x \cdot y_i)$ is conditionally added to the accumulator z depending on the i^{th} bit of a multiplier y as explained in the algorithm.

The procedure adopted in algorithm 4 is known as double and add algorithm evident from steps 4 and 6. It starts from the most significant bit (MSB) of multiplier y_{n-1} and conditionally adds multiplicand x to the accumulator z depending on y_i. At each iteration intermediate results and partial products are reduced by a modulus p to keep them in the range $(0, p-1)$. The main operations are

1. One bit left-shift of z modulo p, this operation is denoted as modular doubling mentioned in step 4.

Algorithm 4: Basic Serial radix-2 Interleaved Multiplication (R2IM)

Input: $x = \sum_{i=o}^{n-1} x_i \cdot 2^i, y = \sum_{i=o}^{n-1} y_i \cdot 2^i, p = \sum_{i=o}^{n-1} p_i \cdot 2^i$
Output: $z = x \times y \mod p$

```
1 z ← 0;
2 for i from n−1 downto 0 ;          // n is a bit-length of p
3 do
4 |   z ← 2z mod p ;                 // modular doubling
5 |   if y_i = 1 then
6 |   |   z ← (z + x) mod p ;        // modular addition
7 |   end
8 end
9 return z
```

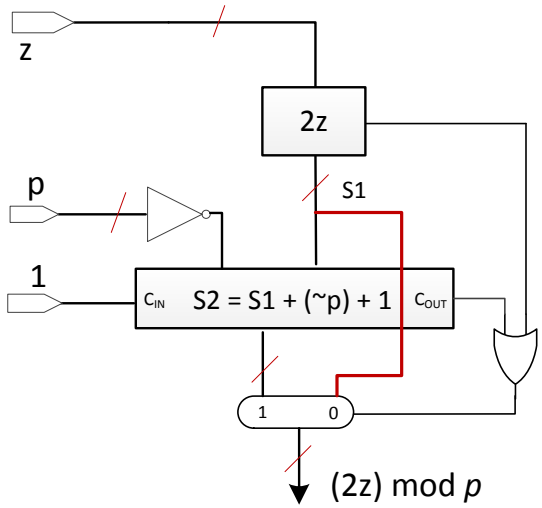


Figure 3.7: Modular doubling architecture

2. Modular addition of multiplicand x to z as described in step 6.

These two steps constitute the overall data path of the radix-2 implementation of the IM algorithm (R2IM). In an integer multiplication, doubling ($2z$) is simply accomplished by one bit left-shift operation which is merely a rewiring in hardware (free of cost), but in case of modular multiplication a reduction by a modulus p is also needed, therefore it is either shift ($2z$) or shift-and-reduce ($2z - p$) operations i.e., a modulus p is subtracted from the result of $2z$ if it is greater than or equal to p. Thus in a finite field its hardware realization consists of one n-bit adder and one $2:1$ multiplexer as shown in Figure 3.7. The modular addition step can be realized in hardware by two n-bit adders and one $2:1$ multiplexer as shown in Figure 3.1.

Let t_{add} denote the critical path delay of an n-bit adder and t_{mux} denote the critical path delay of a $2:1$ multiplexer. Then the critical path delay of R2IM multiplier

architecture (T_{R2IM}) is given in equation 3.4.

$$T_{R2IM} = 3t_{add} + 2t_{mux} \tag{3.4}$$

It is evident from the equation that hardware realization of the R2IM algorithm has a critical path delay of three n-bit adders and two $2:1$ multiplexers. The total number of iterations in the R2IM algorithm is n, if each of the iterations is executed in a single clock cycle then the total number of clock cycles required to execute the R2IM algorithm is exactly n, where n represents the bit length of modulus p.

3.5 Radix-4 BE Interleaved Multiplication

Several modifications have been proposed to optimize the IM method. Ghosh et al. in [100] proposed an architecture that exploited Montgomery powering ladder technique [59] to execute modular doubling and modular addition operations in parallel (R2PIM). However, internal operations are performed bit wise so it also takes n clock cycles to execute an n-bit modular multiplication operation. Two other novel modifications of IM algorithm and their optimized hardware architectures are presented in [98].

There are three basic building blocks of any multiplier design, they are.

- Partial products generation
- Partial products reduction
- Partial products accumulation

There are n partial products in the R2IM and R2PIM multipliers, thus without any partial products reduction techniques their generation and accumulation take n clock cycles, therefore these modular multiplier designs takes n clock cycles to perform an n-bit modular multiplication operation for n-bit operands.

There are several techniques to reduce the total number of partial products and one such technique is known as Booth encoding (BE) [108], [109], [110], [111]. Combining BE with higher radix methods can reduce the total number of partial products, which ultimately reduces the total number of iterations in the R2IM algorithm.

Table 3.2: Radix-4 Booth encoding

Y_i	Y_{i-1}	Y_{i-2}	Encoded value
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	−2
1	0	1	−1
1	1	0	−1
1	1	1	0

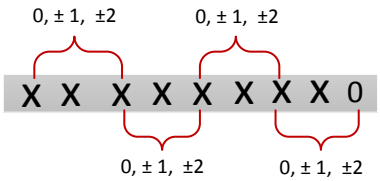


Figure 3.8: Radix-4 Booth encoding

In a simple radix-4 method two bits of a multiplier are processed at a time either moving from MSB towards LSB or vice versa. The two-bit pair can possibly be 00_2, 01_2, 10_2, and 11_2, in integer form these are $\{0, 1, 2, 3\}$. Therefore, using radix-4 the possible partial products can be 0, x, $2x$ mod p, and $3x$ mod p.

By combining radix-4 and BE techniques, the possible partial products can be 0, x, $-x$, $2x$ mod p, and $-2x$ mod p. BE technique shown in Figure 3.8 works on groups of three bits with an overlapping bit from the previous group and encodes these groups into one of the possible values in Table 3.2 i.e., $\{0, \pm 1, \pm 2\}$. BE is a sign digit representation where each group or block is encoded as a signed number in two's complement format. Sign extension of the multiplier y is also required in the case where there are less than 3 bits in the left most group. As in ECC, the multiplication operation is computed over positive numbers. So adding zeros to the left of the MSB of a multiplier is enough to complete the left most group. As the effective number of bits processed in each iteration in Radix-4 BE is two, the number of zeros to be added to the left of the MSB of an n-bit multiplier y is determined as follows:

- If n modulo $2 = 0$, then append two zeros to the left of the MSB of multiplier y
- if n modulo $2 = 1$, then append a single zero to the left of the MSB of multiplier y

As the recommended ECC key sizes are all of even number of bits, so two zeros are

Algorithm 5: Radix-4 BE Interleaved Multiplication (R4BIM)

Input: $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $y = \sum_{i=0}^{n-1} y_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $(z = x \times y) \bmod p$

1 $z \leftarrow 0$
2 $R \leftarrow 2x \bmod p$ // Pre-computed value //
3 $N \leftarrow n+2$ // append two 0s to left of MSB of y //
4 $N \leftarrow N+1$ // append a single 0 to right of LSB of y //
5 **for** $(i = N; i \geq 2; i = i-2)$ **do**
6 $\quad z \leftarrow 4z \bmod p$
7 $\quad$ **switch** $(y_{(i:i-2)})$ **do**
8 $\quad\quad$ **when** $\{000 \mid 111\} \Longrightarrow z \leftarrow z$
9 $\quad\quad$ **when** $\{001 \mid 010 \mid 101 \mid 110\} \Longrightarrow z \leftarrow z \pm x \bmod p$
10 $\quad\quad$ **else** $\Longrightarrow z \leftarrow z \pm R \bmod p$
11 $\quad$ **endsw**
12 **end**
13 **return** z

inserted to the left of the MSB of a multiplier. One extra zero needs to be added to the right of the LSB of multiplier y, as the overlapping bit for the first block. Therefore, an extra iteration is required due to the adding of two zeros to the left of the MSB of multiplier y.

Modification to the R2IM algorithm based on radix-4 and BE techniques (R4BIM) is given in algorithm 5. The proposed R4BIM algorithm reduces the total number of iterations from n to $\lfloor n/2 \rfloor + 1$, therefore the number of partial products is halved, which results in a considerable amount of reduction in multiplication time and the required number of clock cycles. Algorithm 5 also involves two major operations given as follows:

- Two-bit left shift mod p operation i.e., $4z \bmod p$ as specified in step 6.
- Modular addition or subtraction (add/sub) operation as specified in steps 9 and 10.

Due to processing two bits of multiplier y at a time excluding the overlapping bit the R4BIM algorithm becomes double-double add or subtract algorithm. In each iteration step 6 and either of step 8, step 9 or step 10 are executed. The steps 6 and either of step 9 or 10 constitute an overall data path of the R4BIM algorithm.

In every iteration of the algorithm, the accumulator z is two-bit left shifted and then reduced modulo p i.e., $4z \bmod p$. Then, partial products $b_1(0, \pm x, \pm 2x)$ are

modular added or subtracted from the accumulator z depending on the three respective multiplier bits. Modular addition of z and b_1 ($z + b_1 \bmod p$) is required in the case of partial products $b_1 = \{x, 2x\}$, while if partial products $b_1 = \{-x, -2x\}$, then modular subtraction operation is needed i.e., $(z - b_1) \bmod p$. The two bit left shift mod p operation is performed as two single bit left shift mod p operations which is described in the next section. It is worth mentioning that step 2 of the algorithm is performed by a pre-computation process.

3.5.1 Hardware Architecture

Hardware architecture to execute the R4BIM algorithm is shown in Figure 3.9. The architecture can be divided into macro and micro blocks. Two modular doubling ($2z$ mod p) and a single modular Add/Sub blocks are the macro blocks while the BE and M are considered as a micro blocks. In addition to these blocks the architecture also contains one shift register SR, two n-bit registers z, R and a control unit which is not shown in Figure 3.9. The modular doubling blocks are identical, each of them is responsible for performing a $2z$ mod p operation. The internal architecture of the modular doubling block is shown in Figure 3.7, where it is discussed that hardware realization of $2z$ mod p operation requires one n-bit adder and one 2:1 multiplexer. The second modular doubling block in Figure 3.9 performs a single bit left shift mod p operation on the output of the first modular doubling block and produces the result of the $4z$ mod p operation. As this operation is executed as two sequential $2z$ mod p operations, so in total it is carried out by two n-bit adders and two multiplexers. Note that left-shift operation ($\ll$) does not cost anything in hardware because it is achieved by just rewiring.

The macro Add/Sub mod p block performs modular addition or subtraction operation depending on its input carry in signal (cin). Its internal architecture has also been discussed in section 3.2 (see Figure (3.3)). Its critical path delay is comprised of two n-bit adders and three 2:1 multiplexers. Partial products b_1 can have five possible values i.e., $\{0, \pm 1, \pm 2\}x$. In the implementation these are divided into two parts: $b_1 = \{0, +1, +2\}x$ and $b_1 = \{-1, -2\}x$. These two parts are distinguished by the output of block BE (cin), which indicates the Add/Sub block either to perform a modular addition or subtraction operation. The block BE is based on radix-4 Booth encoding logic, where y_i, y_{i-1} represent the two current bits and y_{i-2} represents the overlapping

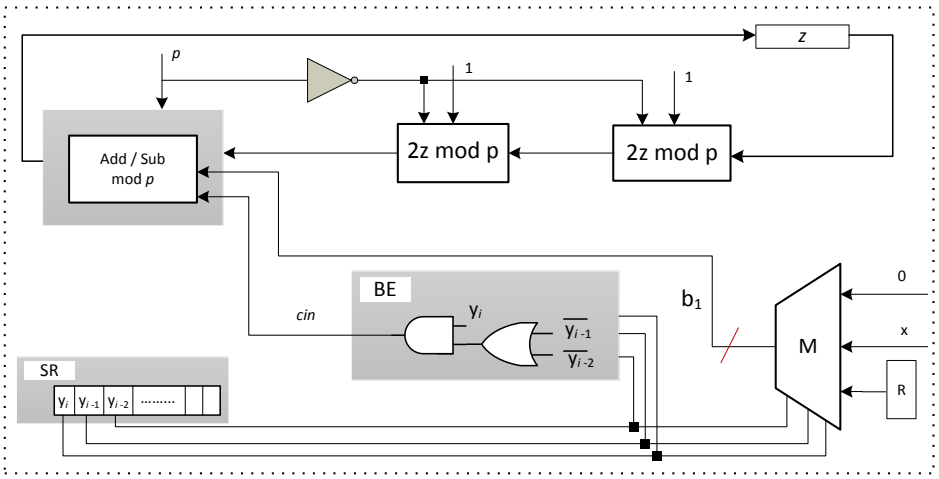


Figure 3.9: R4BIM multiplier architecture

bit of the previous block of a 3 bits block of multiplier y. The overall execution flow of algorithm 5 on the architecture in Figure 3.9 is described below.

- In the first clock cycle register z is loaded with multiplicand x, registers SR, z are loaded with the multiplier y and 0 receptively. In the next clock cycle, partial product $2x$ mod p is pre-computed in the first modular doubling block, which is then stored in register R. Thus, the pre-computation of $2x$ mod p is completed in two clock cycles.

- Then, the register z is two-bit left shifted and are reduced modulo p in the two modular doubling blocks. Partial products $(0, \pm x, \pm 2x)$ are then modular added or subtracted from the accumulator z in the Add/Sub block. Modular addition or subtraction is controlled by the BE block. All these operations are performed in a single clock cycle.

- The SR register left shifts two bits of the multiplier y in each iteration.

- Note that the micro blocks M, BE, and SR perform their respective operations in parallel with the macro blocks.

The total number of iterations of the algorithm is exactly $\lfloor n/2 \rfloor + 1$. The given hardware architecture executes each iteration in a single clock cycle, therefore it takes $\lfloor n/2 \rfloor + 3$ clock cycles to perform an n-bit modular multiplication operation. Note that an extra two clock cycles are consumed for the pre-computation of the $2x$ mod p operation. It is also worth mentioning that step 5 of the R4BIM algorithm is controlled by a counter

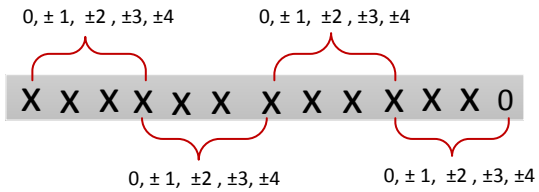


Figure 3.10: Radix-8 Booth encoding

which is decremented twice after each iteration. This counter logic is a part of the control unit, which is not shown in Figure 3.9.

The critical path delay of the R4BIM is comprised of the two modular doubling and a single modular add/sub blocks which is given in equation 3.5.

$$T_{\text{R4BIM}} = 4t_{add} + 5t_{mux} \tag{3.5}$$

3.6 Radix-8 BE Interleaved Multiplication

The R4BIM algorithm reduces the number of loop iterations from n to $\lfloor n/2 \rfloor + 1$. The iteration count can be further reduced by adopting radix-8 instead of radix-4 method. The modified IM algorithm based on radix-8 and BE techniques (R8BIM) is given in algorithm 6. The radix-8 BE technique is given in Table 3.3 and Figure 3.10, where each quadruple of multiplier y is encoded as $\{0, \pm 1, \pm 2, \pm 3, \pm 4\}$ with a one bit overlap from the previous group. Sign extension of multiplier is also required in the case where there is less than four bits in the left most group. As in ECC positive numbers are handled so adding zeros to the left of the MSB of multiplier y, it is enough to complete the left most group. As the effective number of bits processed in radix-8 BE is three, therefore the number of zeros to be added to the left of the MSB of an n-bit multiplier y is determined as follows.

- If n modulo $3 = 0$, add three zeros.
- If n modulo $3 = 1$, add two zeros.
- If n modulo $3 = 2$, add a single zero.

Algorithm 6: Radix-8 BE Interleaved Multiplication (R8BIM)

Input: $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $y = \sum_{i=0}^{n-1} y_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $z = x \times y \bmod p$

1 $z \leftarrow 0$, $R_1 \leftarrow 2x \bmod p$, $R_2 \leftarrow 3x \bmod p$, $R_3 \leftarrow 4x \bmod p$,

$$N = \begin{cases} n+3, & \text{if } n \bmod 3 = 0 \text{ append three 0 to the left of MSB of } y \\ n+2, & \text{if } n \bmod 3 = 1 \text{ append two 0 to the left of MSB of } y \\ n+1, & \text{if } n \bmod 3 = 2 \text{ append single 0 to the left of MSB of } y \end{cases}$$

2 $N \leftarrow N+1$, append single 0 to the right of LSB of y
3 **for** $(i = N;\ i \geq 3;\ i = i-3)$ **do**
4 $\quad z := 8z \bmod p$
5 $\quad$ **switch** $(y_{(i:i-3)})$ **do**
6 $\quad\quad$ **when** $\{0000 \mid 1111\} \Longrightarrow z \leftarrow z$
7 $\quad\quad$ **when** $\{0001 \mid 0010 \mid 1101 \mid 1110\} \Longrightarrow z \leftarrow z \pm x \bmod p$
8 $\quad\quad$ **when** $\{0011 \mid 0100 \mid 1011 \mid 1100\} \Longrightarrow z \leftarrow z \pm R_1 \bmod p$
9 $\quad\quad$ **when** $\{0101 \mid 0110 \mid 1001 \mid 1010\} \Longrightarrow z \leftarrow z \pm R_2 \bmod p$
10 $\quad\quad$ **else** $\Longrightarrow z \leftarrow z \pm R_3 \bmod p$
11 $\quad$ **endsw**
12 **end**
13 **return** z

Finally, one extra zero needs to be added to the left of the LSB of a multiplier y acting as the overlapping bit for the first block. The generation of $\pm 3, \pm 4$ is the major difference to the R4BIM algorithm.

The R8BIM algorithm takes $\lfloor n/3 \rfloor + 1$ iterations to perform an n-bit modular multiplication operation. It is comprised of several steps which are explained as follows.

- In step 1, $2x \bmod p$, $3x \bmod p$ and $4x \bmod p$ values are computed only once. This is done by a pre-computation process which is explained in the next section.
- Step 4 is performed iteratively throughout the loop iterations. It is a three-bit left shift modulo p value of an accumulator z i.e., $8z \bmod p$.
- In steps 7-10, respective partial products are modular added or subtracted from the accumulator z i.e., $z = z \pm \{ x, R_1, R_2, R_3 \}$. Selection of the respective partial product is based on the current four bits of multiplier y being processed according to Table 3.3.

Therefore in each iteration of the algorithm first the accumulator z is three bits left shifted and reduced modulo p. Then in the same iteration the respective partial prod-

Table 3.3: Radix-8 Booth encoding

y_i	y_{i-1}	y_{i-2}	y_{i-3}	Encoded value
0	0	0	0	0
0	0	0	1	+1
0	0	1	0	+1
0	0	1	1	+2
0	1	0	0	+2
0	1	0	1	+3
0	1	1	0	+3
0	1	1	1	+4
1	0	0	0	−4
1	0	0	1	−3
1	0	1	0	−3
1	0	1	1	−2
1	1	0	0	−2
1	1	0	1	−1
1	1	1	0	−1
1	1	1	1	0

uct is modular added or subtracted from the accumulator. The total number of partial products using radix-8 and BE techniques is reduced from n to $\lfloor n/3 \rfloor + 1$.

3.6.1 Hardware Architecure

A hardware architecture to execute the R8BIM algorithm is shown in Figure 3.11. The R8BIM architecture is composed of four major blocks: three modular doubling and one modular Add/Sub. These four major blocks are cascaded in series which means that each block output is the input to the next block. In addition to these major blocks, the R8BIM architecture also contains a look-up-table based multiplexer M, a BE block, a three-bit shift register SR, and four n-bit data registers R_1, R_2, R_3 and z. Execution of the R8BIM algorithm on the presented architecture can be divided into two phases: A and B. These are detailed as follows:

3.6.1.1 Phase A

As in the step 1 of the R8BIM method a pre-computation of $2x$ mod p, $3x$ mod p and $4x$ mod p is needed for a multiplicand x. The pre-computation of these required values is completed in Phase A and is explained below.

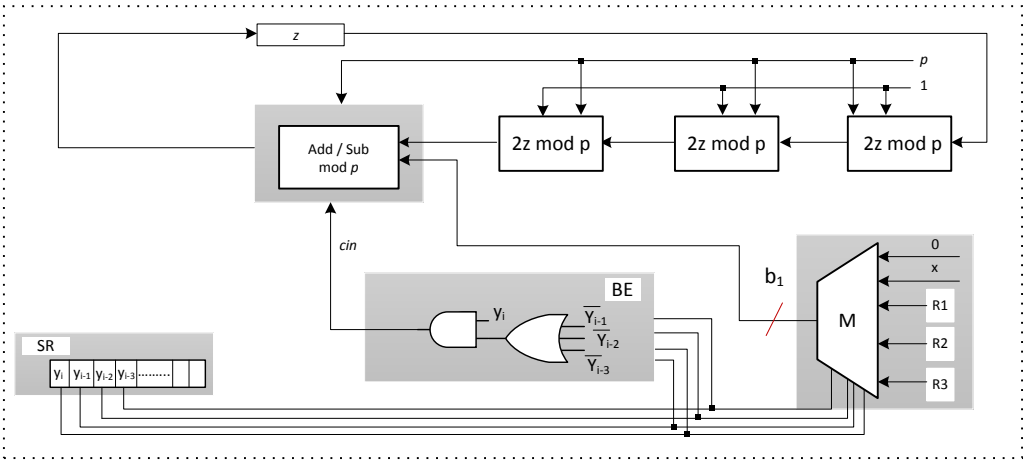


Figure 3.11: R8BIM multiplier architecture

- In the first clock cycle register z is loaded with the multiplicand x. In the next clock cycle, the first and second modular doubling units output $2x$ mod p, $4x$ mod p values receptively. In the same clock cycle these values are stored in registers R_1, R_3 respectively.

- In the next clock cycle $3x$ mod p value is computed in Add/Sub block by setting its inputs to x, $2x$ mod p, and cin to zero. Note that zero is set for the output of block BE which indicates add/sub block to execute a modular addition operation on its respective inputs. Then the result of Add/Sub block is stored in register R_2 in the next clock cycle.

After four clock cycles registers R_{1-3} are loaded with their respective values and the pre-computation process is completed.

3.6.1.2 Phase B

In phase B, the following steps of the algorithm are performed on their respective hardware units in iterative fashion, which are explained as below.

- Loop iteration is controlled by a counter which is decremented by three after each iteration.

- Step 4, $8z$ mod p is performed by three identical modular doubling units cascaded in series. The operation is divided into three single bit left shift mod p

operations executed in a serial fashion given in equation 3.6.

$$8z \bmod p = 2(2(2z \bmod p) \bmod p)) \bmod p \tag{3.6}$$

Register z is initially loaded with zero in phase A. In each iteration of phase B the accumulator is three bits left shifted and reduced modulo p. Then a respective partial product is modular added to or subtracted from the accumulator.

- Steps 5-11 are executed by the M block and Add/Sub block. Block M is a look-up-table based multiplexer that selects the appropriate partial products to be modular added or to subtracted from the accumulator in the Add/Sub block.
- The BE block accepts four consecutive multiplier bits and generates a single bit control signal (cin) that controls the Add/Sub block to perform either modular addition or subtraction operation. If it is equal to zero then modular addition is performed otherwise subtraction is executed.

The controller is based on a counter. It controls the loop execution and also generates appropriate signals to execute phase A and B. Each iteration of the loop is executed in a single clock cycle. As the total number of iterations in BE radix-8 IM algorithm is exactly $\lfloor n/3 \rfloor + 1$. Therefore, the proposed architecture performs an n-bit modular multiplication operation in $\lfloor n/3 \rfloor$+5, in which the extra four clock cycles are consumed in the pre-computation process (phase A) of the algorithm.

The critical path delay of BE radix-8 IM is comprised of three modular doubling and one modular add/sub blocks which is given in equation (3.7).

$$T_{\text{R8BIM}} = 5t_{add} + 6t_{mux} \tag{3.7}$$

By comparing equations (3.5) and (3.7), it is evident that in the critical path of R8BIM multiplier, one extra n-bit adder and one 2:1 multiplexer is required as compared to R4BIM multiplier (see equation (3.5)), however it reduces the number of clock cycles from ($\lfloor n/2 \rfloor + 3$) to ($\lfloor n/3 \rfloor + 5$), which is almost 33% lower than the required number of clock cycles of the R4BIM multiplier . A performance analysis on the basis of computation time, area requirements, operating frequencies and throughput is discussed in the next section.

Table 3.4: Area comparison of IM multipliers implementation on Virtex-6

Design	Field size (p)	Slices	Look-up-tables (LUTs)	Slice registers
R2IM	160-bits	631	1733	531
	192-bits	757	2049	627
	224-bits	993	2401	723
	256-bits	1012	2900	777
R4BIM	160-bits	1186	2911	556
	192-bits	1272	3511	652
	224-bits	1447	4053	748
	256-bits	1550	4606	845
R8BIM	160-bits	1320	3234	562
	192-bits	1442	4119	659
	224-bits	1547	4549	755
	256-bits	1710	5149	851

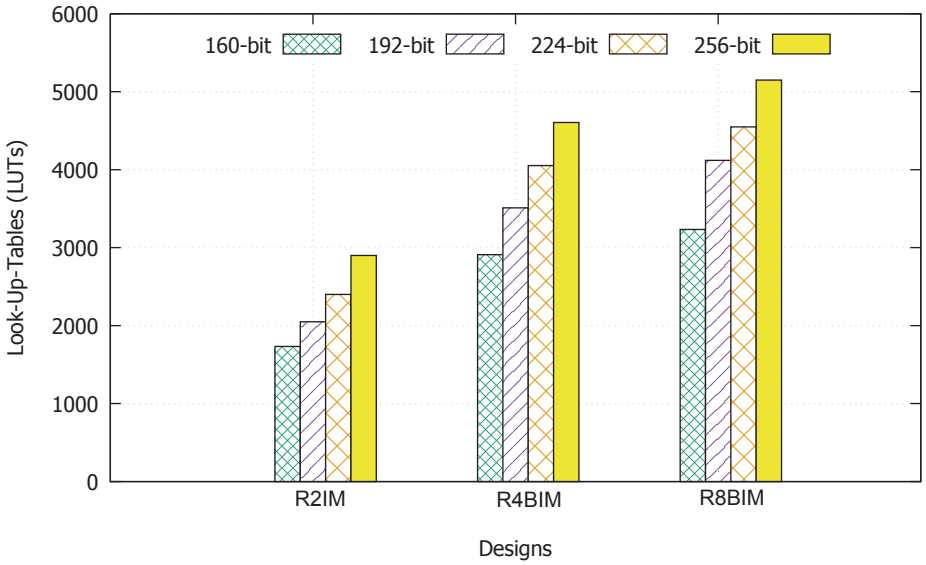


Figure 3.12: Area comparisons of IM multipliers

3.7 Implementation and Results

The R4BIM and R8BIM multipliers are coded in Verilog HDL and are synthesized targeting Virtex-6 FPGA. The Xilinx ISE 14.1 design suite is used for synthesis, mapping, placement, and routing. For behavioral simulation, ModelSim and Xilinx Isim simulators are used. The proposed modular multipliers are also implemented in C# to validate the design functionality.

These architectures have inherent programmability features i.e., the modulus value p can be changed without reconfiguring the FPGA. Table 3.4 lists the area consumption on Virtex-6 platform for 160, 192, 224 and 256-bit field sizes.

A 256-bit implementation of the R4BIM multiplier occupies 1550 Virtex-6 slices (4606 LUTs, 845 slice registers), whereas the R8BIM multiplier on the same platform for the same field size occupies 1710 slices (5149 LUTs, 851 slice registers). Note that these indicate that the R8BIM multiplier consumes 9.4% more FPGA slices than the R4BIM multiplier.

Implementation results for the R2IM technique have been reported in [55], [56], [94], [104]. However, for these implementations different FPGA platforms have been used, thus, a direct comparison with the proposed designs is not very effective. For a fair comparison, this work also implemented R2IM algorithm on Virtex-6 FPGA platform as well. Its implementation results are also given in Table 3.4. For a 256-bit implementation, it occupies 1012 slices (2900 LUTs, 777 slice registers). Its slice consumption is almost 35%, 41% lower than the corresponding R4BIM and R8BIM multipliers, respectively. It is evident from Table 3.4 and Figure 3.12 that the presented higher-radix IM designs consume more FPGA slices, LUTs and slice registers as compared to bit level implementation (radix-2). However, higher-radix designs result in reduced multiplication time as discussed below.

A comparison on the basis of multiplication time, throughput (TP) and area×time per bit (ATB) value is demonstrated in Table 3.5. ATB values in the table is calculated on the basis of equation (3.8).

$$ATB = \frac{\text{no. of occupied slices } \times \text{ total time}}{\text{no. of bits}} \tag{3.8}$$

For a 224-bit implementation the R4BIM architecture computes one modular multiplication operation in 1.29μs at a maximum clock frequency of 89.1 MHz. For a 256-bit implementation it takes 1.48μs at a maximum clock frequency of 88.5 MHz. Similarly, a 224-bit R8BIM architecture takes 1.08μs to compute one modular multiplication operation at a maximum frequency of 73.2 MHz, which is almost 17% faster than the R4BIM design.

A 224-bit implementation of the R2IM multiplier takes 1.74 μs to compute a modular multiplication operation and achieves 129 MHz maximum frequency. The proposed R4BIM and R8BIM multiplier are 35% and 61% faster than the corresponding R2IM multiplier. Figure 3.13 depicts computation time of the presented designs against different field sizes.

Table 3.5: Performance of IM multipliers on Virtex-6 for different field sizes

Design	Field size (p)	Freq (MHz)	Time (μs)	TP (Mbps)	ATB
R2IM	160-bits	136.8	1.18	135.5	4.65
	192-bits	131.6	1.46	131.5	5.77
	224-bits	129	1.74	128.7	7.71
	256-bits	125	2.03	126	8.02
R4BIM	160-bits	94	0.88	181	6.52
	192-bits	91.6	1.08	178	7.15
	224-bits	89.1	1.29	173.6	8.33
	256-bits	88.5	1.48	172.9	8.96
R8BIM	160-bits	77.1	0.752	213	6.2
	192-bits	75.7	0.91	211	6.8
	224-bits	73.2	1.08	206	7.5
	256-bits	72	1.25	205	8.35

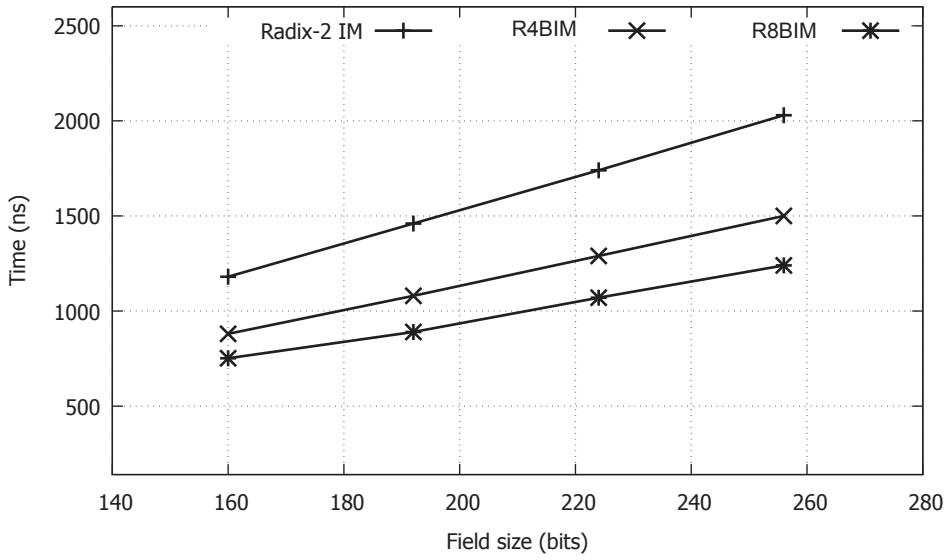


Figure 3.13: Computation time of different IM multipliers

For a 256-bit implementation, the R4BIM and R8BIM multipliers have throughput of 172.9 and 205 Mbps (Mega bits per second), which is 37% and 63% higher than the throughput of R2IM multiplier (126 Mbps) respectively.

The number of clock cycles required to perform a multiplication may be considered as a platform independent parameter. The R4BIM and R8BIM architectures reduce the required number of clock cycles to perform one multiplication by almost 50% and 66% respectively. However, inside the presented designs the main operations are executed in serial fashion which results in the longer critical path delays. Thus, despite the lower clock cycle count the ATB values are higher than the R2IM multiplier as evident from the last column of Table 3.5. Therefore, the presented R4BIM and R8BIM designs

are suitable for high performance applications at the cost of more logic resources.

3.8 Conclusion

Basic finite field arithmetic operations are the building blocks in the design of elliptic curve scalar multiplier architecture. In this regard, this chapter first presents algorithms and corresponding architectures for modular addition/subtraction, inversion/-division operations. Then, two modular multiplier designs based on radix-4, radix-8, Booth encoding, and interleaved multiplication techniques are presented. The architectures and implementation results are compared and discussed.

The presented R4BIM and R8BIM multipliers provide 27% and 39% improvement in a computation time over a corresponding bit level radix-2 IM multiplier. However, internal critical operations of the presented designs are performed in serial fashion which limits their performance. In the next chapter these designs are optimized in terms of critical path delay.

Chapter 4

High Performance Parallel Modular Multipliers

Typical higher-radix based multipliers produce faster results because of their lower iteration count as compared to a bit-level radix-2 implementation. However, these techniques deteriorate the critical path delays, which limit the maximum achievable clock frequencies and speed performance. To obtain maximum performance several optimization techniques can be explored to reduce the critical path delay in higher-radix multiplier designs. Parallelization is an optimization technique that reduces the computation time by reducing the critical path delay.

This Chapter shows that there is a good scope of parallelism in the designs of radix-4, radix-8 Booth encoded interleaved multipliers presented in Chapter 3. This chapter first investigates independent operations in the designs and then presents parallel high performance hardware architectures that facilitate the parallel execution of these independent operations. Then, the chapter also presents a comprehensive performance analysis of different parallel and serial higher-radix interleaved multiplier designs.

4.1 Introduction

As discussed in Chapter 3, modular multiplication is the fundamental and computationally intensive operation in many Public-Key cryptographic processors. Hence, it has a substantial impact on the overall performance of the associated cryptosystems. Therefore, its optimization is one of the common strategies to boost the overall performance of the cryptosystem.

Performance of a modular multiplier can be assessed using some quantitative metrics. These performance measuring metrics include computation time, resource requirements, power consumption, throughput, etc. As the objective of this research is to reduce the overall computation time for scalar multiplication on elliptic curves, emphasis is on increasing the performance of modular multiplier in terms of computation time and throughput, while keeping an eye on the resource requirements.

As discussed in chapter 3, modular multiplication over a prime field is either performed by iterative interleaved additions and reduction or absolute reduction following an integer multiplication. The absolute reduction step involves division by a large prime p which is another costly operation. Therefore, this work focuses on the technique based on repeated interleaved addition and reduction modulo p. Another well known method for modular multiplication is Montgomery method [54]. This method requires operand conversion from normal binary form to Montgomery domain and the result conversion from Montgomery domain back to normal binary form, which needs extra computations besides the actual modular multiplication operation. The method adopted here works directly on numbers in two's complement form and does not need any operands and result conversions. Higher-radix IM methods presented in chapter 3 exhibit longer critical path delays as compared to the radix-2 or bit level implementation of the IM method as indicated in equations (3.4), (3.5) and (3.7).

Critical path of a R2IM multiplier is comprised of three n-bit adders and two multiplexers, whereas the critical path of a R4BIM multiplier consists of four n-bit adders and five multiplexers, and in the case of R8BIM it is five n-bit adders and six multiplexers. The longer critical paths decrease maximum achievable frequencies. Therefore, optimization techniques to shorten the critical path is of utmost importance in higher-radix based IM multiplier designs. An IM multiplier design with an optimized critical path is reported in [103], in which the critical path is reduced to two n-bit adders and

two 2:1 multiplexers as given in equation (4.1).

$$T_{R2PIM} = 2t_{add} + 2t_{mux} \tag{4.1}$$

Where T_{R2PIM} denotes the critical path delay of the radix-2 parallel interleaved multiplier (R2PIM). This bit-level parallel design also requires n clock cycles to perform an n-bit modular multiplication operation.

This chapter presents parallel modular multipliers with their efficient architectures based on higher-radix and BE techniques. The parallelism idea is adapted from the Montgomery powering ladder approach. Due to the introduced parallelism, the designs are able to execute the main operations concurrently. Higher-radix can reduce the required number of clock cycles for a multiplication operation. In this regard radix-4 and radix-8 discussed in chapter 3 have been adopted to reduce the iteration count. It is also observed that incorporating BE logic in the radix-4 and radix-8 parallel multipliers helps to reduce the area cost with a slight degradation in the maximum achievable clock frequencies. Therefore, the number of required clock cycles are reduced by using higher-radix techniques and the critical paths are reduced by introduced parallelism to execute the critical operations concurrently.

Novel higher-radix parallel modular multiplication algorithms and the corresponding hardware architectures are presented in the following sections. Performance comparison of these higher-radix parallel multipliers and higher-radix serial multipliers presented in Chapter 3 are also compared in detail on the basis of computation time, operating frequency, area consumption and throughput.

4.2 Motivation

There are two ways to speed up a modular multiplication, reducing the required number of clock cycles or decreasing the critical path delay which in turn increases the operating frequency. These aspects of any design are interrelated in such a way that typically it is not possible to optimize both at the same time. Optimization in terms of reducing the number of clock cycles using higher-radix techniques deteriorates the critical path delay as shown in the R4BIM and R8BIM multipliers in the previous chapter.

The critical operations in these designs are executed in serial fashion and hence, these designs have long critical paths thus are not able to achieve higher frequencies. These designs are referred as serial higher-radix IM multipliers. The main operations in R4BIM algorithm are given below:

```
1 z ← 0;
2 for i = N; i ≥ 2; i = i − 2 do
3 |   z ← 4z modulo p;
4 |   z ← z ± pp modulo p        // pp denotes partial products //
5 end
```

It processes a two-bit of a multiplier in each iteration, therefore the number of iterations are reduced to half as compared to a radix-2 implementation. However the critical path is comprised of steps 3 and 4, which consists of four n-bit adders and five 2:1 multiplexers i.e., $4add + 5mux$. Similarly, in case of the R8BIM multiplier design the step 4 remains the same while step 3 is replaced by a three-bit left shift modulo p operation, which is realized using $3add + 3mux$. Hence, the overall critical path of the R8BIM multiplier consists of $5add + 6mux$. Introducing parallelism allows the execution of steps 3 and 4 of the algorithm concurrently as explained in the next section.

4.2.1 Montgomery Powering Ladder

Algorithm 7: The Montgomery Powering Ladder for exponentiation [59]

Input: $x, y \leftarrow [y_{n-1}, y_{n-2}, ..., y_0]$
Output: x^y

```
1 R0 ← 1,   R1 ← x
2 for i = n − 1 downto 0 do
3 |   if (yi = 0) then
4 |   |   R1 ← R0 · R1;   R0 ← (R0)^2
5 |   end
6 |   else
7 |   |   R0 ← R0 · R1;   R1 ← (R1)^2
8 |   end
9 end
10 return R0
```

Parallelization of the main operations in higher-radix IM designs is inspired by the Montgomery powering ladder (ML) technique which is given in algorithm 7. The ML

Algorithm 8: Radix-4 Parallel IM Multiplication (R4PIM)

Input: $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $y = \sum_{i=0}^{n-1} y_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $z = x \times y \bmod p$

1 $z \leftarrow x, R_1 \leftarrow x$
2 $R_2 \leftarrow 2x \bmod p$, $R_3 \leftarrow 3x \bmod p$ // Pre-computed values //
3 $z \leftarrow 0$
4 **for** $(i = 0; i \leq N-2; i \leftarrow i+2)$ **do**
5 **switch** $(y_{(i+1:i)})$ **do**
6 **when** $00 \Longrightarrow v \leftarrow 0$
7 **when** $01 \Longrightarrow v \leftarrow R_1$
8 **when** $10 \Longrightarrow v \leftarrow R_2$
9 **when** $11 \Longrightarrow v \leftarrow R_3$
10 **endsw**
 // Following operations are executed in parallel //
11 $R_1 \leftarrow 4R_1$ modulo p
12 $R_2 \leftarrow 4R_2$ modulo p
13 $R_3 \leftarrow 4R_3$ modulo p
14 $R \leftarrow z + v$ modulo p
15 **end**
16 **return** z

method was initially proposed to speed-up the square and multiply technique of an exponentiation. The ML method eliminates conditional branch evaluation and enables parallel execution of a multiplication and squaring operations as shown in steps 4 and 7 of the algorithm. Both operations are performed at every iteration of the algorithm irrespective of the exponent bit y_i.

At the end of each iteration, internal variables R_0, R_1 are assigned the results of multiplication or squaring operations depending on the current exponent bit y_i. Therefore, data dependencies between these operations are completely eliminated.

4.3 Radix-4 Parallel Interleaved Multiplier (R4PIM)

Radix-4 Parallel interleaved multiplication (R4PIM) method is given in algorithm 8. This method, instead of two-bit left shift mod p of the accumulator contents, shifts the possible partial products in each iteration starting from LSB to MSB of a multiplier. The algorithm is comprised of two phases: A and B. In phase A pre-computation of possible partial products are computed which are $2x \bmod p$ and $3x \bmod p$. Notice that partial products 0, x are available and do not require any pre-computation. In phase B, several operations in the algorithm are performed independently and iteratively. In

each iteration all the partial products are two-bit left shifted and are reduced modulo p which are shown in the steps 11, 12, 13. In step 14, the modular addition of the accumulator and respective partial product is performed. Note that there is no data dependency in steps 11, 12, 13 and 14. Therefore, they can be performed concurrently on their respective hardware units. It is also worth noticing that steps 11, 12 and 13 are exactly identical i.e., $4R_{1-3} \bmod p$ operation whereas step 14 is a modular addition operation. As these operations can be performed in parallel therefore, any of the steps 11, 12, 13 and 14 constitute an overall critical path of the R4PIM multiplier which is discussed in the next section.

4.3.1 Hardware Architecture

A hardware architecture to execute algorithm 8 is shown in Figure 4.1. The architecture is comprised of four processing elements (PE_{1-4}) operating in parallel and four n-bit data registers R_{1-4} and some multiplexers. The internal architectures of (PE_{1-3}) units consist of two modular doubling blocks cascaded in series as presented in chapter 3 (Figure 3.9) while PE_4 unit is a modular adder described in section 3.2. PE_1 has two outputs i.e. $2x$ modulo p and $4x$ modulo p so it is slightly different than PE_2 and PE_3 which have only one $4x$ modulo p output.

The R4PIM algorithm also needs some pre-computed values therefore the whole process in the algorithm is divided into two phases: phase A and phase B. Phase A deals with the pre-computation process while phase B carries out the iterative execution of the algorithm. The functionality of these phases and their execution on the given architecture in Figure 4.1 are detailed below:

4.3.2 Phase A

For radix-4 technique where two bits of a multiplier are processed at a time, possible partial products are $\{0, x, 2x, 3x\}$, where x is the multiplicand and $\{2x, 3x\}$ needs to be pre-computed before start of the multiplication process. Phase A deals with the computation of $\{2x, 3x\}$ modulo p operations. To compute these operations registers z, R_1 are loaded with a multiplicand x, then $\mathbf{PE}_1$ performs $2x$ modulo p, which is either $2x$ or $2x-p$ and are available after a single clock cycle at output u_1 of the unit as shown in Figure 4.2. Then, it is stored in register R_2 by setting the respective select signal. In

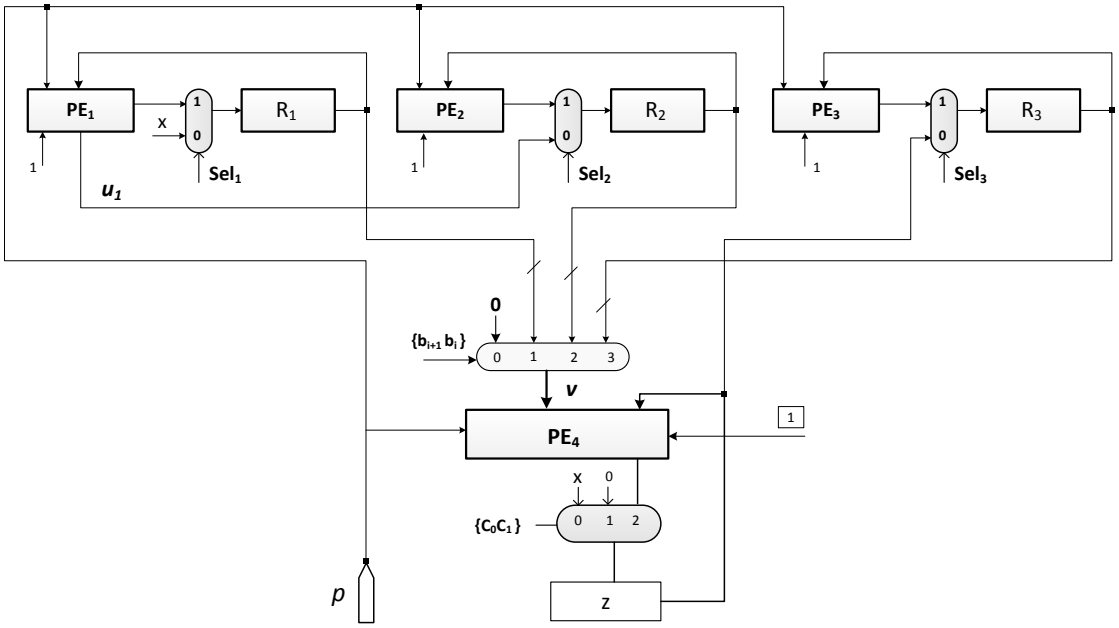


Figure 4.1: R4PIM multiplier hardware architecture

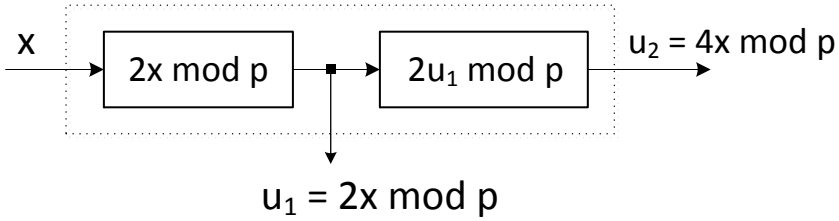


Figure 4.2: Internal architecture of first processing element

first two clock cycles computation of $2x$ modulo p operation is completed and stored in register R_2. Then, $3x$ modulo p operation is performed as $(R_2 + x)$ mod p, which is computed in $\mathbf{PE}_4$ unit in a single clock cycle by selecting appropriate operands (R_2 and x) and the result is stored in register z. In the next clock cycle, register R_3 is updated by register z, which is the required result of $3x$ mod p operation. Therefore, four clock cycles are consumed in the computation of $2x$ and $3x$ mod p operations. $\mathbf{PE}_4$ architecture (modular adder) consist of two n-bit adders cascaded in series with some data-multiplexing circuitry. The first adder performs operand addition $s_1 = (x + y)$, then modulus p is subtracted from the result in the second adder i.e., $s_2 = (s_1 - p)$, and finally outputs either s_1 or s_2. This phase of the algorithm is completed in just four clock cycles. After these four cycles, now registers R_1, R_2 and R_3 hold operand x, $2x$ mod p, and $3x$ mod p, respectively.

4.3.3 Phase B

In phase B of the algorithm several operations are executed in parallel on the hardware architecture as given in Table 4.1. In each iteration four operations specified as $R_1 = 4R_1 \bmod p$, $R_2 = 4R_2 \bmod p$, $R_3 = 4R_3 \bmod p$, and $z+v \bmod p$ are executed in parallel on $\mathbf{PE}_1$, $\mathbf{PE}_2$, $\mathbf{PE}_3$ and $\mathbf{PE}_4$ units, respectively.

In this execution phase select signals sel_{1-3} are set to one so that registers R_1, R_2 and R_3 can not be updated with operands x, u_1 and z, respectively. $\mathbf{PE}_1$, $\mathbf{PE}_2$, and $\mathbf{PE}_3$ units perform two-bit left shift mod p operations i.e., $4R_1$, $4R_2$, and $4R_3$ in parallel . Internal architectures of two-bit left shift mod p is presented in chapter 3 Figure 3.9 where it is executed as two single bit left shift mod p operations and each single bit left mod p operation consists of a single n-bit adder and a multiplexer. Therefore, the critical paths of $\mathbf{PE}_1$, $\mathbf{PE}_2$, and $\mathbf{PE}_3$ are identical, and is comprised of $2add + 2mux$. Operation $(z + v)$ of the algorithm is performed by $\mathbf{PE}_4$, which has a critical path of $2add + mux$, as all these four operations are executed in parallel, therefore the critical path of the radix-4 parallel modular multiplier is given in equation (4.2).

$$T_{\text{R4PIM}} = 2t_{add} + 4t_{mux} \tag{4.2}$$

Note that in Figure 4.1 the critical path is between any of registers R_1, R_2, R_3 and z, where two data multiplexers (4:1 and 2:1)[1] are in the path in addition to $\mathbf{PE}_4$ unit. There are exactly $\frac{n}{2}$ iterations. In every iteration, all steps in phase B of the algorithm are executed in a single clock cycle, therefore, this phase is completed in $\frac{n}{2}$ clock cycles and overall the algorithm takes $\frac{n}{2} + 4$ clock cycles to perform an n-bit modular multiplication operation. For example a 256-bit modular multiplication is performed in 132 clock cycles. An overall execution of a modular multiplication operation on the proposed R4PIM multiplier is given in Table 4.1.

[1] 4:1 multiplexer has a critical path delay of two 2:1 multiplexers

Table 4.1: Operation sequence of modular multiplication on R4PIM multiplier

#cycle	$\mathbf{PE}_1$	$\mathbf{PE}_2$	$\mathbf{PE}_3$	$\mathbf{PE}_4$
1	$R_1 = x$	-	-	$z = x$
2	-	$R_2 = (2x \bmod p)$	-	-
3	-	-	-	$z = (z + R_2) \bmod p$
4	-	-	$R_3 = (3x \bmod p)$	$z = 0$
5	$R_1 = 4R_1 \bmod p$	$R_2 = 4R_2 \bmod p$	$R_3 = 4R_3 \bmod p$	$z = (z + v) \bmod p$
6	$R_1 = 4R_1 \bmod p$	$R_2 = 4R_2 \bmod p$	$R_3 = 4R_3 \bmod p$	$z = (z + v) \bmod p$
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
$\lfloor \frac{n}{2} \rfloor + 4$	-	-	-	z= final value

Algorithm 9: Radix-4 BE Parallel IM Multiplication (R4BPIM)

Input: $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $y = \sum_{i=0}^{n-1} y_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $z = x \times y \bmod p$

1 $z \leftarrow 0, \quad R_1 \leftarrow x$
2 $R \leftarrow 2x \bmod p$ // Pre-computed value //
3 $N \leftarrow n+2$ // append two 0's to left of MSB of y //
4 $N \leftarrow N+1$ // append a single 0 to right of LSB of y //
5 **for** $(i = 0; i \leq N-2; i \leftarrow i+2)$ **do**
6 **switch** $(y_{(i+2:i)})$ **do**
7 **when** $000 \mid 111 \Longrightarrow v \leftarrow 0$
8 **when** $001 \mid 010 \mid 101 \mid 110 \Longrightarrow v \leftarrow R_1$
9 **else** $\Longrightarrow v \leftarrow R_2$
10 **endsw**
 // Following operations are executed in parallel //
11 $R_1 \leftarrow 4R_1$ modulo p
12 $R_2 \leftarrow 4R_2$ modulo p
13 $z \leftarrow z \pm v$ modulo p
14 **end**
15 **return** z

4.4 Radix-4 Booth Encoded Parallel Interleaved Multiplier (R4BPIM)

In the design of R4PIM multiplier four processing units **(PE_{1-4})** are operated in parallel. As each of $\mathbf{PE}_{1-3}$ is comprised of two n-bit adders and two 2:1 data multiplexers. $\mathbf{PE}_4$ is a modular adder and its hardware realization consists of two n-bit adders and a single 2:1 multiplexer. Therefore, there are in total of eight n-bit adders. By adopting BE logic one processing unit can be saved which ultimately saves two n-bit adders and three 2:1 multiplexers as explained below.

In case of radix-4 Booth encoding, the possible partial products are $0, \pm x$ and $\pm 2x$ using two's complement representation subtraction can be implemented using addition. Therefore, only two partial products $+x, +2x$ are required. Hence, one processing unit is saved as compared to the R4PIM multiplier design. The processing unit PE_4 (modular addition unit) needs to be replaced with a modular addition/subtraction unit. Therefore the critical path of radix-4 booth encoded parallel IM (R4BPIM) multiplier is slightly longer than the R4PIM multiplier.

R4BPIM method is given in algorithm 9. Zeros need to be appended in the case where there is not enough bits in the MSB block of a multiplier as discussed in Chapter

3. The R4BPIM algorithm scans triplets of a multiplier y from LSB to MSB and instead of shifting the accumulator it shifts partial products in each iteration. It is comprised of several independent steps such as 11, 12 and 13.

In steps 11, 12 two-bit left shift modulo p operation is performed on the partial products as discussed in the previous section while step 13 is a modular addition or subtraction determined on the output cin signal of BE logic. The BE logic circuit generates the cin signal on the basis of three respective multiplier bits. The R4BPIM method given in algorithm 9 also works in two phases, phase A and phase B similar to the R4PIM algorithm. Now in phase A only $2x$ mod p value is required to be pre-computed. These phases are discussed in the following section.

4.4.1 Hardware Architecture

Hardware architectures of the R4BPIM multiplier is shown in Figure 4.3.The presented architecture consists of three processing elements $\mathbf{PE}_1$, $\mathbf{PE}_2$ and $\mathbf{PE}_3$. In addition to these main elements there is a **BE** block, three n-bit data registers R_1, R_2, z. The internal architectures of $\mathbf{PE}_1$ and $\mathbf{PE}_2$ are exactly the same as the $\mathbf{PE}_{1-3}$ units in the R4PIM multiplier. The $\mathbf{PE}_3$ unit is a modular addition/subtraction unit given in Figure 3.3. The execution process of the R4BPIM algorithm on the given architecture in Figure 4.3 is explained below.

4.4.2 Phase A

Again the phase A deals with the computation of $2x$ modulo p operation. The registers R_1 is loaded with the multiplicand x, then in the next clock cycle $\mathbf{PE}_1$ performs $2x$ modulo p which is available at output u_1 of the unit. Then, in the same clock cycle it is stored in register R_2 by setting the sel_2 signal equal to zero. Therefore step 2 of the algorithm is completed in two clock cycles.

4.4.3 Phase B

In phase B of the algorithm several operations are executed in parallel on the hardware architecture as demonstrated in Table 4.2. In each iteration three operations $R_1 = 4R_1$ mod p, $R_2 = 4R_2$ mod p and $z = z \pm v$ mod p are executed in parallel on $\mathbf{PE}_1$, $\mathbf{PE}_2$

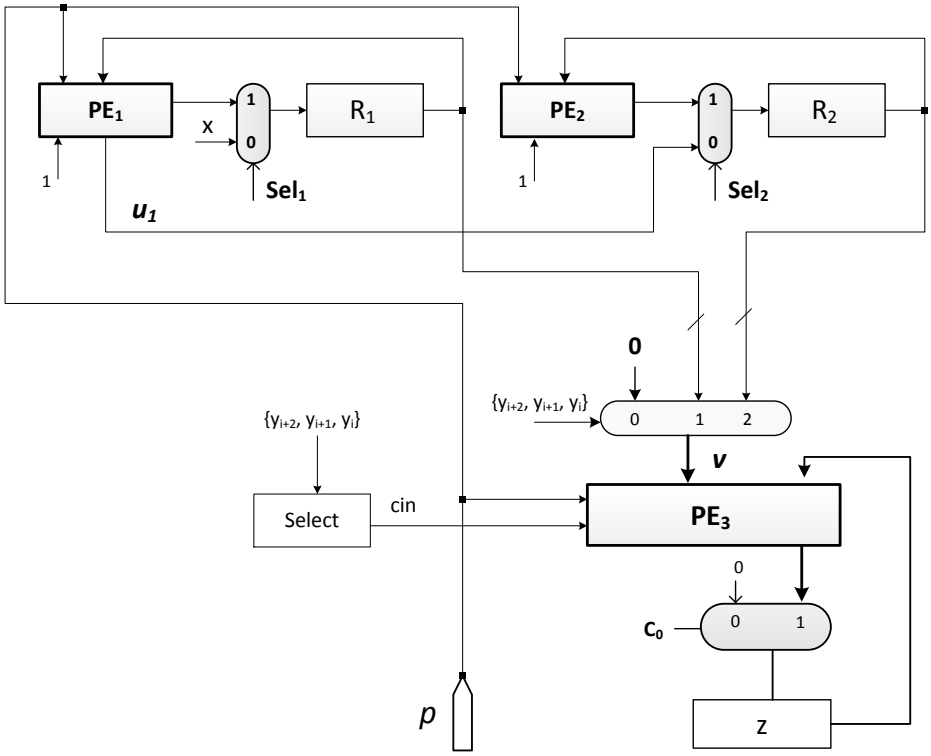


Figure 4.3: R4BPIM multiplier hardware architecture

and $\mathbf{PE}_3$ units, respectively. In this execution phase select signals sel_1 and sel_2 are set to one, which indicates that registers R_1 and R_2 can not be updated with x and u_1 respectively. $\mathbf{PE}_1$ and $\mathbf{PE}_2$ perform two-bit left shift mod p operations i.e., $4R_1$ mod p and $4R_2$ mod p in parallel. Each of these individual operations are executed as two single bit left shift mod p operations where each single bit left shift mod p operation consists of a single n-bit adder and a multiplexer. Therefore, the critical paths of $\mathbf{PE}_1$ and $\mathbf{PE}_2$ are identical which is comprised of $2add + 2mux$.

The operation of $z = z + v$ or $z = z - v$ of the algorithm is performed by $\mathbf{PE}_3$ unit, where its critical path consists of $2add + 3mux$ (see Figure 3.3), as these three operations are executed in parallel, therefore the critical path of the R4BPIM multiplier is given in equation (4.3).

$$T_{\text{R4BPIM}} = 2t_{add} + 7t_{mux} \tag{4.3}$$

Note that in Figure 4.3 critical path is either between registers R_1 and z or R_2 and z, where there is a single (8:1) and a single (2:1) data multiplexers [2] are in the path in addition to $\mathbf{PE}_3$ unit. Their are exactly $\lfloor \frac{n}{2} \rfloor + 1$ iterations and in every iteration all steps in phase B of the algorithm are executed in a single clock cycle, therefore, this phase is completed in $\lfloor \frac{n}{2} \rfloor + 1$ clock cycles and overall the algorithm takes $\lfloor \frac{n}{2} \rfloor + 3$ clock cycles

[2] 8:1 multiplexer has a critical path delay of three 2:1 multiplexers

Table 4.2: Operation sequence of modular multiplication on R4BPIM multiplier

#cycle	$\mathbf{PE}_1$	$\mathbf{PE}_2$	$\mathbf{PE}_3$
1	$R_1 = x$		-
2	-	$R_2 = 2x \bmod p$	-
3	$R_1 = 4R_1$	$R_2 = 4R_2$	$z = (z \pm v)$
3	$R_1 = 4R_1$	$R_2 = 4R_2$	$z = (z \pm v)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\lfloor \frac{n}{2} \rfloor + 3$	-	-	$z =$ final result

to perform an n-bit modular multiplication operation. The pre-computation process does not incur any additional combinational blocks and it only costs two clock cycles overhead.

4.5 Radix-8 Booth Encoded Parallel Interleaved Multiplier (R8BPIM)

The iteration count in R4BPIM multiplier can be reduced from $\lfloor \frac{n}{2} \rfloor + 1$ to $\lfloor \frac{n}{3} \rfloor + 1$ using radix-8 and BE techniques as explained in Chapter 3. A radix-8 BE parallel IM (R8BPIM) multiplier technique is given in algorithm 10. The radix-8 BE technique is shown in Figure 3.10, where it scans a quadruplet of a multiplier y with a single bit overlap between adjacent quadruplets. Possible partial products in this case are $\{0, \pm 1, \pm 2 \pm 3, \pm 4\}x$. Radix-8 BE technique is discussed in detail in Chapter 3, see Section 3.6.

The R8BIM algorithm is comprised of five main steps i.e. 13, 14, 15, 16, 17. The step 17 is a modular add/sub operation. The other steps (13-16) are three-bit left-shift modulo p operation. There is no data dependency, therefore, all these operations can be executed in parallel. In the case of R8BPIM, the iteration count is reduced to $\lfloor n/3 \rfloor + 1$, however it requires more design space due to the increased number of processing units required to execute more operations in parallel which is discussed in the next section.

Algorithm 10: Radix-8 BE Parallel IM Multiplication (R8BPIM)

Input: $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $y = \sum_{i=0}^{n-1} y_i \cdot 2^i$, $p = \sum_{i=0}^{n-1} p_i \cdot 2^i$
Output: $z = x \times y \bmod p$

$z \leftarrow x, R_1 \leftarrow x$
$R_2 \leftarrow 2x \bmod p, R_3 \leftarrow 3x \bmod p, R_4 \leftarrow 4x \bmod p$ // pre-computed values //
$N = \begin{cases} n+3, & \text{if } n \bmod 3 = 0, \text{ append three 0 to the left of MSB of } y \\ n+2, & \text{if } n \bmod 3 = 1, \text{ append two 0 to the left of MSB of } y \\ n+1, & \text{if } n \bmod 3 = 2, \text{ append single 0 to the left of MSB of } y \end{cases}$
$N \leftarrow N + 1$ // append a single 0 to right of LSB of y //
for $(i = 0; i \leq N-3; i = i+3)$ **do**
  switch $(y_{(i+3:i)})$ **do**
    when $0000 \mid 1111 \Longrightarrow v \leftarrow 0$
    when $0001 \mid 0010 \mid 1101 \mid 1110 \Longrightarrow v \leftarrow R_1$
    when $0011 \mid 0100 \mid 1011 \mid 1100 \Longrightarrow v \leftarrow R_2$
    when $0101 \mid 0110 \mid 1001 \mid 1010 \Longrightarrow v \leftarrow R_3$
    else $\Longrightarrow v \leftarrow R_4$
  endsw
  // Following operations are executed in parallel //
  $R_1 \leftarrow 8R_1 \bmod p$
  $R_2 \leftarrow 8R_2 \bmod p$
  $R_3 \leftarrow 8R_3 \bmod p$
  $R_4 \leftarrow 8R_4 \bmod p$
  $z \leftarrow z \pm v \bmod p$
end
return z

4.5.1 Hardware Architecture

The R8BPIM architecture in Figure 4.4 is comprised of four identical three-bit left shift mod p processing units PE_{1-4} and the modular add/sub unit named as PE_5. In addition to these it also contains some data registers R_{1-4}, z and a BE logic block.

Here phase A of the algorithm is completed in four clock cycles. In clock cycle one registers R_1, z are loaded with multiplicand x. Then in the next clock cycle, PE_1 computes $2x$ mod p and $4x$ mod p values which are then stored in registers R_2, R_4, respectively. In the third clock cycle $3x$ mod p value is computed in PE_5 for inputs z, R_2 and the result is stored in register z. In clock cycle four this value is loaded to register R_4 and the pre-computation process is completed. Note that this is a very similar procedure adopted in the pre-computation process of R4PIM multiplier shown in Figure 4.1. Therefore it is also completed in four clock cycles.

In phase B, PE_{1-4} units performs three-bit left shift mod p operation i.e., $8x$ mod p. The internal architectures of these units are identical as explained in chapter 3

(see section 3.6.1), where it has been shown that $8x$ mod p operation can be computed by three n-bit adders and three multiplexers cascaded in series. The BE block as shown in Figure 4.4 now operates on four multiplier bits i.e., $y_{i+3}, y_{i+2}, y_{i+1}, y_i$ and generates a control signal c_{in} for the PE_5 unit. As the total number of iterations in the algorithm is $\lfloor n/3 \rfloor + 1$, therefore the proposed architecture computes an n-bit modular multiplication operation in $\lfloor n/3 \rfloor + 5$ clock cycles.

Operation scheduling of the R8BPIM algorithm on the architecture in Figure 4.4 is given in Table 4.3. It is observed that now the critical path is shifted to PE_{1-4} units because each of these unit is comprised of three n-bit adders and three multiplexers, therefore the critical path of the BE radix-8 IMML multiplier is given in equation (4.4).

$$T_{\text{R8BPIM}} = 3t_{add} + 4t_{mux} \tag{4.4}$$

Table 4.3: Operation sequence of modular multiplication on R8BPIM architecture

#cycle	$\mathbf{PE}_1$	$\mathbf{PE}_2$	$\mathbf{PE}_3$	$\mathbf{PE}_4$	$\mathbf{PE}_5$
1	$R_1 = x$	-	-	-	$z = x$
2	-	$R_2 = 2x \bmod p$	-	$R_4 = 4x \bmod p$	
3	-	-	-		$z = z + R_2 \bmod p$
4		-	$R_3 = 3x \bmod p$		$z = 0$
5	$R_1 = 4R_1 \bmod p$	$R_2 = 4R_2 \bmod p$	$R_3 = 4R_3 \bmod p$	$R_4 = 4R_4 \bmod p$	$z = z \pm v \bmod p$
6	$R_1 = 4R_1 \bmod p$	$R_2 = 4R_2 \bmod p$	$R_3 = 4R_3 \bmod p$	$R_4 = 4R_4 \bmod p$	$z = z \pm v \bmod p$
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
$\lfloor \frac{n}{3} \rfloor + 5$	-	-	-	-	$z =$ final result

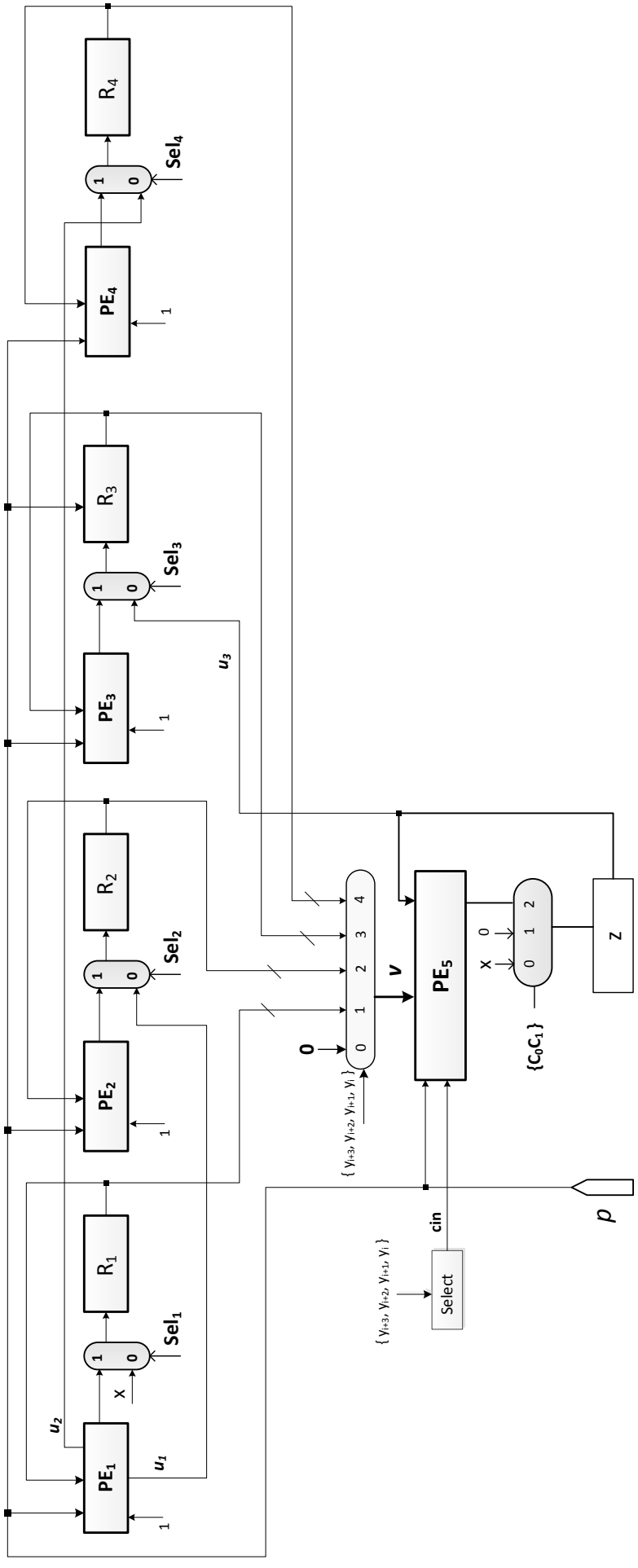


Figure 4.4: R8BPIM multiplier hardware architecture

Table 4.4: Resource requirements analysis of IM multipliers

Design	Resource requirements
R2IM [86]	$3A_{add} + 2A_{mux} + A_{reg}$
R2PIM [100]	$3A_{add} + 5A_{mux} + 2A_{reg}$
R4BIM	$4A_{add} + 4A_{mux} + 2A_{reg}$
R4PIM	$8A_{add} + 14A_{mux} + 5A_{reg}$
R4BPIM	$6A_{add} + 17A_{mux} + 4A_{reg}$
R8BIM	$5A_{add} + 5A_{mux} + A_{reg}$
R8PIM	$23A_{add} + 31A_{mux} + 9A_{reg}$
R8BPIM	$14A_{add} + 28A_{mux} + 6A_{reg}$

4.6 Platform Independent Performance Analysis

This section presents performance analysis of different IM multiplier designs. The same design on different implementation platforms produces varying results, hence it is not conclusive to compare designs implemented on different platforms. This section demonstrates a platform independent analysis of the IM multiplier designs. The designs are analysed on the basis of their space complexity (resource requirements), critical path delay and latency.

4.6.1 Resource Requirements

Table 4.4 and Figure 4.5 demonstrates resource requirements of the IM multiplier designs, where A_{add}, A_{mux} and A_{reg} represent the area of an n-bit adder, an n-bit multiplexer and an n-bit register respectively. The radix-2 implementation of the IM algorithm (R2IM) requires $3A_{add} + 2A_{mux}$ and a single n-bit register. Similarly, R2PIM design reported in [100] has an area complexity of $3A_{add} + 5A_{mux}$ and two n-bit registers. Note that it is obvious from the table that the presented designs are more complex and therefore have higher area complexities as compared to the Radix-2 designs especially higher-radix PIM multipliers. This is because of using more resources to execute operations in parallel. However, among these PIM designs, it is shown that BE logic helps to reduce the area complexity compared to non BE designs. For example, in the Table 4.4, it is shown that R4BPIM requires $6A_{add}$, $17A_{mux}$ and four n-bit registers. The same design without BE logic i.e., R4PIM has an area complexity of $8A_{add}$, $14A_{mux}$ and five n-bit registers.

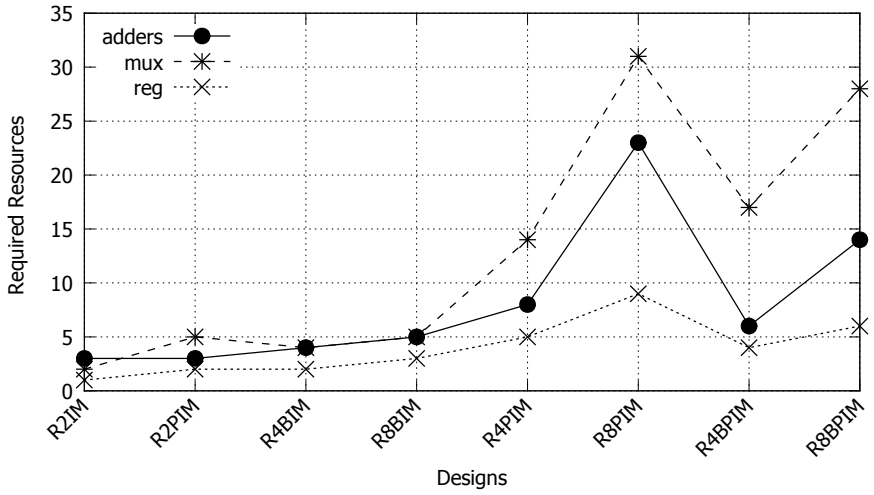


Figure 4.5: Resource requirements of IM multipliers

Table 4.5: Latency analysis of IM multipliers

Design	Critical Path (T_{clk})	# clock cycles	Latency
R2IM [86]	$3t_{add} + 2t_{mux}$	$n+1$	$(n+1) \times T_{clk}$
R2PIM [100]	$2t_{add} + 2t_{mux}$	$(n+1)$	$(n+1) \times T_{clk}$
R4BIM	$4t_{add} + 5t_{mux}$	$(\lfloor n/2 \rfloor + 3)$	$(\lfloor n/2 \rfloor + 3) \times T_{clk}$
R4PIM	$2t_{add} + 4t_{mux}$	$(\lfloor n/2 \rfloor + 5)$	$(\lfloor n/2 \rfloor + 5) \times T_{clk}$
R4BPIM	$2t_{add} + 7t_{mux}$	$(\lfloor n/2 \rfloor + 3)$	$(\lfloor n/2 \rfloor + 3) \times T_{clk}$
R8BIM	$5t_{add} + 6t_{mux}$	$(\lfloor n/3 \rfloor + 5)$	$(\lfloor n/3 \rfloor + 5) \times T_{clk}$
R8PIM	$3t_{add} + 4t_{mux}$	$(\lfloor n/3 \rfloor + 7)$	$(\lfloor n/3 \rfloor + 7) \times T_{clk}$
R8BPIM	$3t_{add} + 4t_{mux}$	$(\lfloor n/3 \rfloor + 5)$	$(\lfloor n/3 \rfloor + 5) \times T_{clk}$

‡ Total clock cycles ($\#clk$), clock period (t_{clk}), adder (add), multiplexer (mux)

A more clear picture can be observed by comparing area complexities of R8PIM[3] and R8BPIM designs. R8BPIM requires fewer adders and registers as compared to R8PIM as demonstrated in Table 4.4.

4.6.2 Critical Path and Latency

Table 4.5 lists critical path delay and latency of the IM multipliers. In the table t_{add} and t_{mux} represent time delay of a n-bit adder and a n-bit (2:1) multiplexer. Moreover, a critical path delay which determines the minimum clock period and is denoted as T_{clk}. The design reported in [86] is a serial radix-2 implementation with a critical path delay of $3t_{add} + 2t_{mux}$ and it takes $n+1$ clock cycles to perform a modular multiplication operation.

The design in [100] is based on parallel radix-2 approach with a T_{clk} delay of

[3] R8PIM is extension of R4PIM design and it is not presented in this work

$2t_{add} + 2t_{mux}$, which is a saving of one t_{add} in the minimum clock period T_{clk} as compared to the serial radix-2 approach and it also takes $n+1$ clock cycles.

The R4BIM and R8BIM multipliers take $\lfloor \frac{n}{2} \rfloor + 3$, $\lfloor \frac{n}{3} \rfloor + 5$ clock cycles respectively, which is almost 50% and 66% reduction in the number of clock cycles as compared to the designs reported in [86] and [100]. However these designs exhibit longer critical paths as shown in Table 4.5. The parallel versions of these multipliers (R4PIM, R4BPIM, R8PIM, R8BPIM) have critical path delays comparable to the IM bit-level implementation. It is worth mentioning that the parallel higher-radix IM multipliers consumes almost the same number of clock cycles to compute a modular multiplication operation as serial higher-radix multipliers (R4BIM, R8BIM). However, their critical path delay T_{clk} delay is half of the serial higher-radix T_{clk} delay. Therefore, these designs can provide almost 50% speed-up to a modular multiplication operation as compared to the design in [98] and [100].

4.7 Implementation Results

This sections presents implementation results and performance evaluation of higher-radix Booth encoded parallel interleaved multipliers. The multipliers are coded in Verilog HDL and Xilinx ISE 14.2 Design Suite is used for synthesis, mapping, placement and routing purposes targeting Virtex-6 FPGA device XCV6LX550. The Xilinx ISIM simulator is used for behavioral simulation of the designs. A software implementation of the proposed multipliers is done in C#. The outputs from the simulator tool and the software are compared to verify the correctness of the designs by applying different test inputs.

The proposed multipliers in this chapter are also implemented in Addition and subtraction are performed making use of in-built fast carry chains of the device using carry select approach [100]. Performance of the designs are compared in two stages. In the first stage the designs are analysed and compared on the basis of computation time, FPGA occupied resources, maximum attainable frequency, and the number of required clock cycles. In the second stage, performance of these parallel designs are compared against other designs reported in the literature on the basis of area-delay product, throughput, and (throughput / slice area).

Table 4.6: Area results of Virtex-6 FPGA implementation of Parallel IM multipliers

Design	Platform	Field siz (p)	Area (slices)	LUTs[b]	Slice registers
R4PIM	Virtex 6	256	1985	6300	2187
		224	1745	5367	1883
		192	1519	4641	1625
		160	1268	3780	1366
R4BPIM	Virtex 6	256	1631	4935	1382
		224	1496	4427	1221
		192	1395	3846	1057
		160	1042	3184	910
R8PIM	Virtex 6	256	4428	13880	2756
		224	4014	12737	2436
		192	3631	10520	2116
		160	3191	8821	1795
R8BPIM	Virtex 6	256	3622	10284	1952
		224	3326	9115	1727
		192	2745	7728	1502
		160	2306	6334	1276

[a]Look-up-tables

4.7.1 Area Results

FPGA area consumption of the designs are listed in Table 4.6 for four different field sizes p (160, 192, 224, 256). The R4PIM design consumes 1985 FPGA slices (including 6300 LUTs and 2187 slice registers) while computing a 256-bit modular multiplication operation. The R8PIM design for the same bit length consumes 4428 slices (including 13880 LUTs, 2756 slice registers), which is almost 2.23 times more than R4PIM multiplier design.

Similarly, the R4BPIM design consumes 1631 slices (including 4935 LUTs and 1382 slice registers) which is almost 18% fewer FPGA slices as compared to the R4PIM design. On the other hand, R8BPIM design occupies 3622 FPGA slices (including 10284 LUTs and 1952 slice registers) which is again almost 18% saving in slice area as compared to the R8PIM design. A graphical view of the area consumption of the designs is shown in Figure 4.6. Therefore, it is evident from the table that BE logic in the higher-radix parallel IM multipliers helps to lower their area cost.

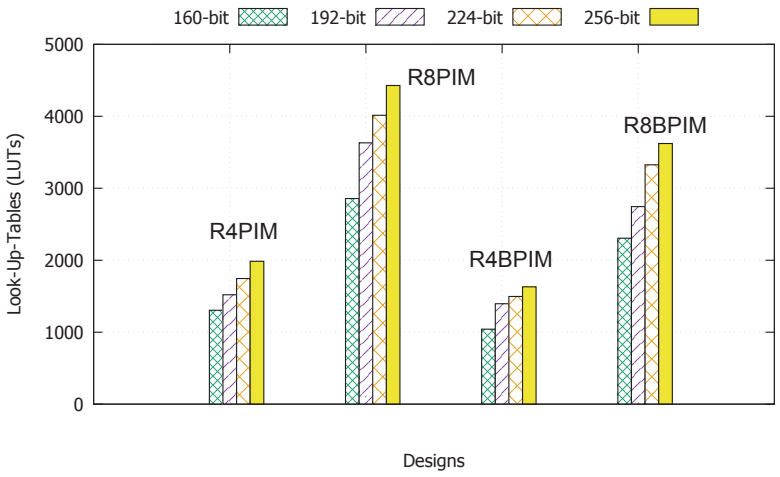


Figure 4.6: Area comparison of parallel IM multipliers

4.7.2 Execution Time Results

Execution time of a modular multiplication operation by the higher-radix parallel IM multipliers are listed in Table 4.7. The R4PIM design computes a 256-bit modular multiplication operation in 0.8 *us* while running at a maximum frequency of 166 MHz. The R8PIM design takes 0.74 *us* to compute the same bit length operation at a maximum frequency of 124.4 MHz.

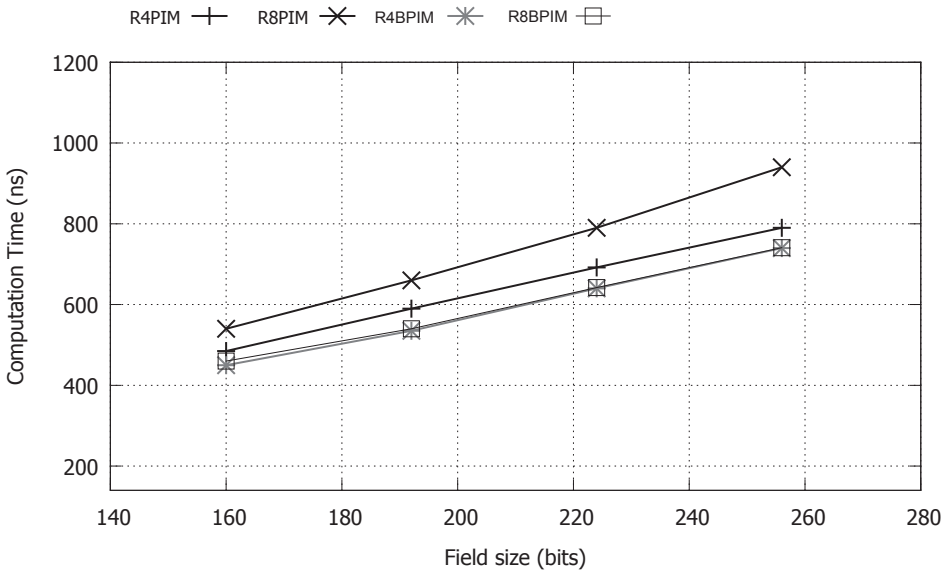


Figure 4.7: Time comparison of higher-radix parallel IM multipliers

The R4BPIM design takes 0.94 *us* to compute a 256-bit modular multiplication while running at 137.9 MHz. It is 17.5% slower than the R4PIM design. A similar conclusion can be drawn by comparing the R8PIM and the R8BPIM designs in the table. Therefore, it shows that BE logic in higher-radix parallel IM multipliers can decrease the design space complexity, with slight degradation in the speed performance

Table 4.7: Timing results of Higher-radix Parallel IM multipliers on Virtex-6 FPGA

Design	Field size (p)	Frequency (MHz)	#Clock cycle	Time (us)
R4PIM	256	166	133	0.8
	224	167.6	117	0.7
	192	168.7	101	0.6
	160	173	85	0.5
R4BPIM	256	137.87	131	0.94
	224	142.7	115	0.8
	192	145.7	99	0.68
	160	147	83	0.56
R8PIM	256	124.4	92	0.74
	224	127.6	81	0.63
	192	132.6	71	0.54
	160	136	60	0.44
R8BPIM	256	123.43	90	0.73
	224	125.7	79	0.63
	192	127	69	0.54
	160	128.4	58	0.45

as compared to the non BE higher-radix parallel IM multipliers. A comparison of time taken by the higher-radix parallel IM multipliers to compute a modular multiplication operation of different field sizes is shown in Figure 4.7.

4.8 Performance Evaluation and Analysis

Table 4.8 presents performance evaluation of the different IM based modular multipliers. Note that R4BIM and R8BIM designs are presented in Chapter 3 while R2IM and R2PIM are reported in [103] and [99,100]. The table shows occupied FPGA slices for different designs against their performance in terms of maximum frequency and computation time and it also lists synthesis results of the designs against four different field sizes (160, 192, 224, 256).

The implementation results listed in Table 4.8 for R2IM and R2PIM designs are the implementation results on Virtex-6 platform. The available implementation results in the literature for these designs are on different FPGA platforms, therefore, these have been carefully implemented on the same Virtex-6 platform along with the other proposed modular multipliers in this work.

From the synthesized results in Table 4.8 it is clear that the presented multipliers are better in terms of computation time. For example for 256-bit field size, the R4PIM

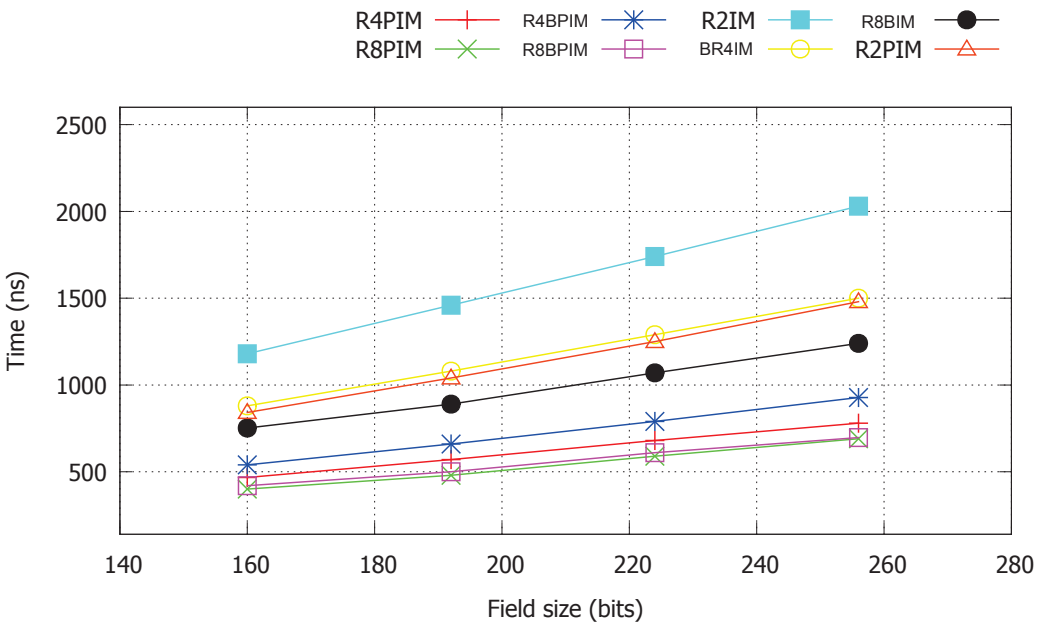


Figure 4.8: Time comparison of different IM multipliers

is upto 2.6, 1.89 times faster than R2IM and R2PIM designs, respectively. However it consumes 1.96, 1.66 times more FPGA slices as compared to R2IM and R2PIM designs, respectively. Similar conclusion can be drawn for the other designs. It is also worth noticing that the serial IM multipliers (R4BIM, R8BIM) are significantly slower than the Parallel IM multipliers. This is because the parallel IM multipliers (R4PIM, R8PIM) introduced parallelism inspired by the Montgomery powering ladder technique to execute the internal operations in parallel, while the R4BIM, R8BIM multipliers execute internal operations in a serial fashion.

Comparison of time required to perform a modular multiplication by different multipliers is shown in Figure 4.8 and hardware resource utilization is shown in Figure 4.9.

Table 4.8: Virtex-6 FPGA implementation results of different IM multipliers

Design	Field size (p) size	Area (slices)	LUTs	Slice registers	Freq (MHz)	Time (us)
R2IM	160	631	1733	531	136.8	1.18
	192	757	2049	627	131.6	1.46
	224	993	2401	723	129	1.74
	256	1012	2900	777	125	2.03
R2PIM	160	712	2002	691	191	0.84
	192	910	2401	819	184.6	1.04
	224	995	2787	947	179.1	1.25
	256	1190	3207	1075	174	1.48
R4BIM	160	1186	2911	556	91.6	0.88
	192	1272	3511	652	89	1.08
	224	1447	4053	748	87.3	1.29
	256	1550	4606	845	85.5	1.5
R4PIM	160	1268	3780	1366	173	0.5
	192	1519	4641	1625	168.7	0.6
	224	1745	5367	1883	167.6	0.7
	256	1985	6300	2187	166	0.8
R4BPIM	160	1042	3184	910	147	0.56
	192	1395	3846	1057	145.7	0.68
	224	1496	4427	1221	142.7	0.8
	256	1631	4935	1382	137.87	0.94
R8BIM	160	1320	3234	562	77	0.752
	192	1442	4119	659	75.7	0.89
	224	1424	4549	755	73.2	1.07
	256	1820	5149	851	72	1.24
R8PIM	160	3191	8821	1795	136	0.44
	192	3631	10520	2116	132.6	0.54
	224	4014	12737	2436	127.6	0.63
	256	4428	13880	2756	124.4	0.74
R8BPIM	160	2306	6334	1276	128.4	0.45
	192	2745	7728	1502	127	0.54
	224	3326	9115	1727	125.7	0.63
	256	3622	10284	1952	123.43	0.73

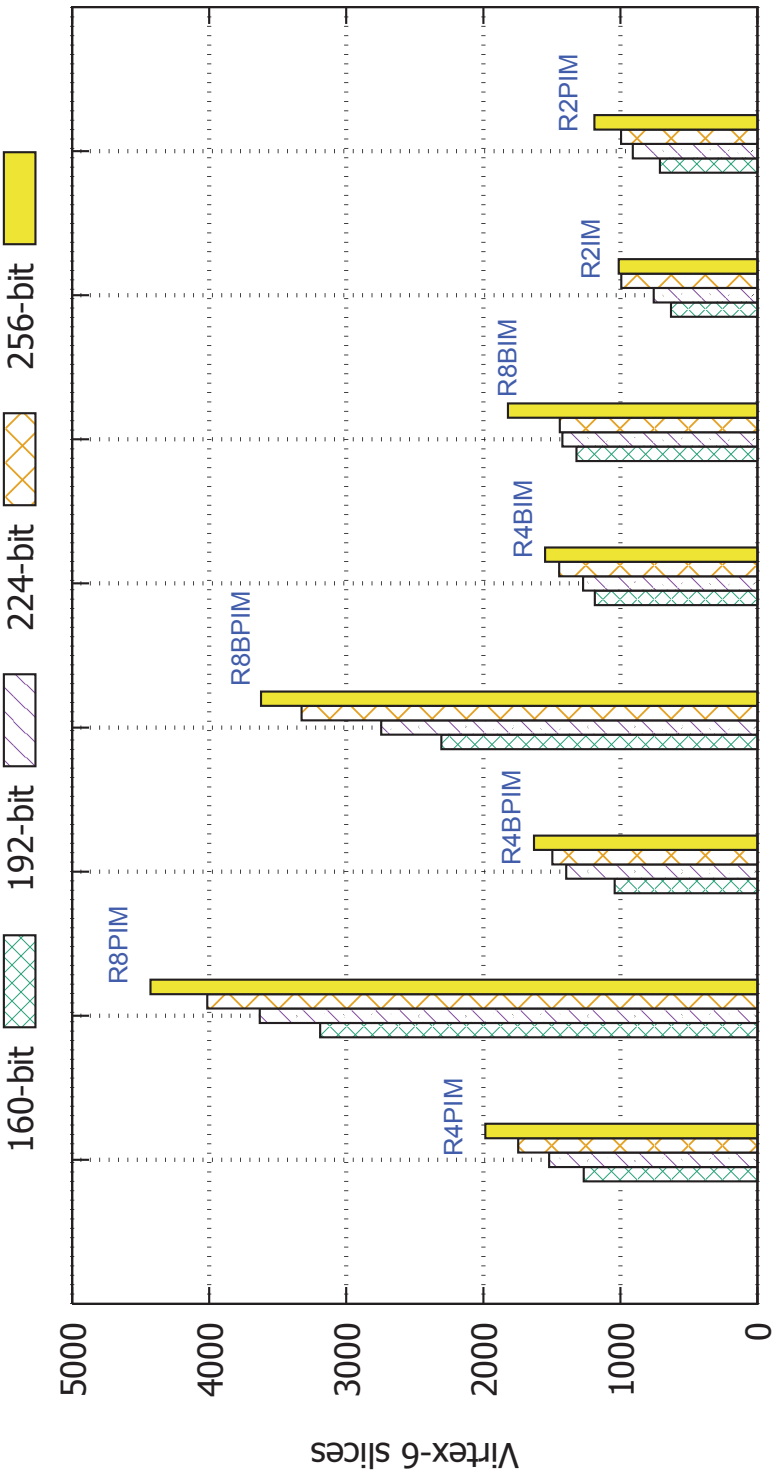


Figure 4.9: Area comparison of different IM multipliers

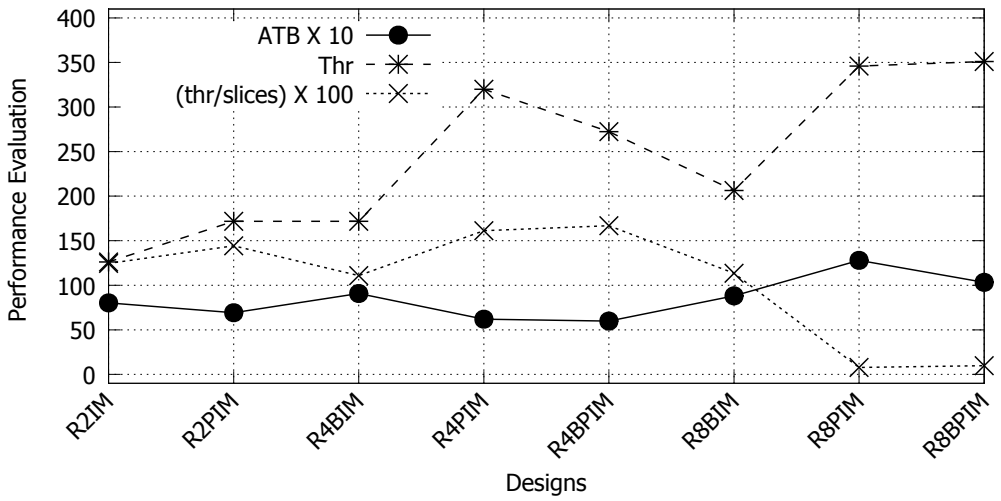


Figure 4.10: Performance evaluation of IM multipliers

4.9 Throughput and Area-Delay Product

The performance of different IM multiplier designs are evaluated basd on throughput (thr), area-delay product per bit (ATB), and (thr/slices) given in Table 4.9 and Figure 4.10. In the table α factor is a measure of throughput/ slice area, higher α factor indicates that a design is better optimized for throughput and area, while lower ATB value indicates that a design is better optimized for computation time and area. Therefore, a design having a low ATB value and high α factor is optimized for a good trade-off between hardware resources, computation time, and throughput.

R4PIM and R4BPIM designs have low ATB and high α than the other designs listed in Table 4.9. Moreover, the R4PIM design has a slightly higher ATB value than that of the R4BPIM design with an almost same α factor. Therefore, these designs are suitable for those applications where performance and resource consumption are of equal importance. On the other hand, R8PIM and R8BPIM have high ATB values with low α factor, which indicates that these designs have higher throughput rate as compared to the other listed designs. Therefore, these designs are suitable for very high performance applications.

R4BIM and R8BIM are not using parallelism, hence, they execute the main operations of IM algorithm in a serial fashion. Although these designs consume almost the same amount of clock cycles to perform a modular multiplication operation, however due to the serial nature of the designs they have longer critical path delays as compared to the R4PIM, R4BPIM, R8PIM and R8BPIM.

Table 4.9: Throughput and area-delay product of different IM multipliers

Design	Field siz (p)	[1]ATB	[2]Thr. (bps)	α =(Thr / slice area)
R2IM [86]	224-bits	7.71	128.7 M	0.1296
	256-bits	8.02	126.1 M	0.1245
R2PIM [100]	224-bits	5.55	179.2 M	0.1801
	256-bits	6.92	171.8 M	0.1443
R4BIM	224-bits	8.33	173.6 M	0.1199
	256-bits	9.08	171.8 M	0.1108
R4PIM	224-bits	5.45	320 M	0.1833
	256-bits	6.2	320 M	0.1612
R4BPIM	224-bits	5.34	280 M	0.1871
	256-bits	5.98	272.3 M	0.1669
R8BIM	224-bits	6.80	209.3 M	0.1469
	256-bits	8.81	206.4 M	0.1134
R8PIM	224-bits	11.28	356 M	0.0886
	256-bits	12.79	346 M	0.0781
R8BPIM	224-bits	9.35	356 M	0.1070
	256-bits	10.32	351 M	0.0969

[1] Area-delay product per bit (ATB)
[2] Throughput (Thr) bits per second (bps)

R4BPIM design has the lowest area-delay product. R8BPIM has the same throughput but much lower area-delay product as compared to the R8PIM. These comparisons indicate that introducing BE and parallelism in the IM algorithm result in speed and area optimized modular multipliers. Figure 4.11 shows comparison of the IM multipliers, where radix-2 Montgomery multiplier design (R2MM) in [105] computes a 256-bit modular multiplication operation in 1.68 *us* and consumes 947 slices. It is clear that R4PIM and R4BPIM are better optimized for speed and resources. R4PIM design requires more adders and fewer multiplexers as compared to the R4BPIM design which ultimately results in more FPGA slice consumption. This is because of the fact that a multiplexer is a simple circuit as compared to an adder and requires few logic resources.

R4PIM design is more suitable for high speed applications, therefore it is utilized in the design of EC scalar multiplier presented in the next chapter.

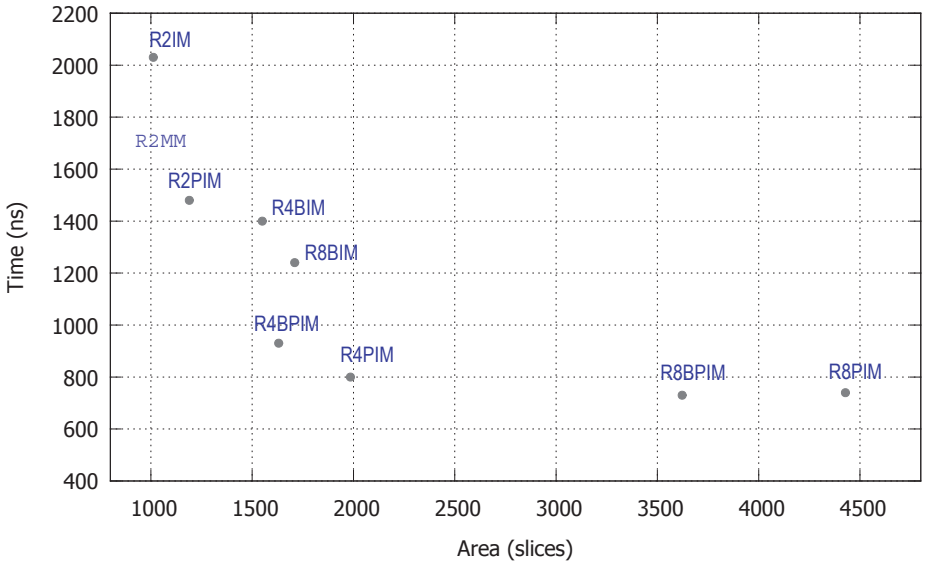


Figure 4.11: Comparison of IM multipliers

4.10 Conclusion

Higher-radix based multipliers are faster because of their lower iteration count as compared to the bit-level implementation. However, these techniques deteriorate the critical path delays, which limit their maximum achievable clock frequencies and desired performance. To obtain a maximum performance optimization techniques can be explored to reduce the critical path delay in higher-radix multiplier designs. Parallelization is one such optimization technique that reduces the computation time by reducing the critical path delay.

This chapter shows that there is a good scope of parallelism in the design of interleaved multipliers presented in Chapter 3. It first identifies independent operations in the designs and then presents parallel high performance hardware architectures that facilitate the parallel execution of these independent operations. The chapter also presents a comprehensive performance analysis of the parallel and serial higher-radix interleaved multipliers.

Chapter 5

EC Scalar Multiplier Architectures

Public-key cryptography (PKC) has solved many problems that were previously considered impractical such as key exchange, digital signatures, etc [25]. Most of the PKC protocols are based on two efficient schemes: RSA [3] and elliptic curve cryptography (ECC) [1], [2]. Recommendations by different standards [112] indicate that 256-bit ECC implementation is capable of providing an equivalent security in comparison to 3072-bit RSA. This gap of the required number of bits in ECC and RSA is expected to increase further in future due to higher security demands. Therefore, due to the much smaller key sizes for same level of security, the ECC based crypto-systems are better in terms of bandwidth utilization, power consumption, and implementation cost as compared to the traditional RSA based crypto-systems.

ECC is particularly useful in resource constrained devices because ECC requires lower implementation and transmission cost and thus lower power consumption [113]. ECC will find applications in the Internet of thing, where more and more resource constrained devices will be connected to the Internet.

This chapter presents efficient EC scalar multiplier architectures using affine and projective coordinates. On the system level, double-and-add and always-double-add algorithms are adopted. The presented EC scalar multiplier architectures are designed using the radix-4 parallel interleaved multiplier presented in Chapter 4.

5.1 Introduction And Related Work

Elliptic curve scalar multiplication is a fundamental and computationally intensive operation in nearly all ECC based crypto-systems. It is the multiplication of a scalar (integer) value to a point on an elliptic curve. Mathematically it is denoted as, $Q = dP$, where a point P and a scalar d are multiplied together to generate another point Q on the curve. In this scenario, points P and Q are public parameters, while scalar d is a secret value that is used in the process of secure encryption.

Mathematically, finding the secret d, while knowing the public parameters P and Q is known as the elliptic curve discrete logarithm problem (ECDLP). The hardness of the ECDLP is the basis of the security of all ECC crypto-systems. However, ECDLP can be bypassed by exploiting several algorithmic and implementation weaknesses termed as side channel attacks (SCA) [114]. For example, if an adversary somehow gains access to a cryptographic device, then the adversary may be able to figure out d by monitoring timing and power consumption profiles of the device. The most simple and common SCAs are based on timing and simple power analysis [31]. There are also more sophisticated attacks based on fault injection, differential power analysis [32], and many others [33], [34]. This work focuses on the efficient implementation of EC scalar multiplication that provides resistance to only timing and simple power analysis attacks. These attacks are simple and more common in practice, strategies to fight against these type of attacks need to be incorporated in any cryptosystem.

Their are different EC representations such as short Weierstrass, Edwards [115], Twisted Edwards [116], Montgomery [117, 118], etc. Currently most security protocols use EC in short Weierstrass representation. Although EC point multiplication on the other mentioned curves are faster than on the short Weierstrass representation, these are not standardised yet. Therefore, the focus of this research is on the efficient implementation of EC scalar multiplication using short Weierstrass representation.

A number of hardware architectures have been reported to efficiently compute the EC scalar multiplication operation [46, 47, 51, 55, 56, 57, 61, 62, 63, 64, 65, 67, 68, 69, 71, 103, 119]. Among these, [46], [47], [119] are based on elliptic curves (ECs) and prime fields recommended by the US National Institute of Standards and Technology (NIST) [85], while all other designs support any general prime field $GF(p)$. A comprehensive overview of EC scalar multiplier hardware architectures can be found

in [44], [73]. Typically, NIST based designs are superior in terms of performance, however they are less flexible compared to design over general $GF(p)$. All these designs developed EC scalar multiplier architecture using standard EC Weierstrass, $\mathbb{E}_{\mathbf{W}}$, representation. EC scalar multiplier architecture in [43] is developed over binary Edwards curves, which imposes completely different design challenges, whereas [70] is a hardware implementation over twisted Edwards curves [116]. EC group operations in $\mathbb{E}_{\mathbf{W}}$ representation using affine (x, y) coordinates have limited parallelism scope at low level finite field arithmetic operations, whereas in projective coordinates several possible parallelism strategies can be devised based on the available finite field primitives.

5.2 Elliptic curve scalar multiplication

Algorithm 11: Double-and-add (DA) method for EC point multiplication [16]

Input: An integer $d = \sum_{i=0}^{n-1} d_i \cdot 2^i$ and a point P on elliptic curve
Output: dP

$Q \leftarrow 0$
for $(i = n-1; i \geq 0; i = i-1)$ **do**
  $Q \leftarrow 2Q$ // EC Point doubling //
  if $(d_i = 1)$ **then**
    $Q \leftarrow Q + P$ // EC Point addition //
  end
end
return Q

Algorithm 12: Double-and-always-add (DAA) for EC point multiplication [16, 56]

Input: An integer $d = \sum_{i=0}^{n-1} d_i \cdot 2^i$ and a point P on elliptic curve
Output: $d = dP$

$Q_0 \leftarrow P, \quad Q_1 \leftarrow 0$
for $(i = 0; i \leq n-1; i = i+1)$ **do**
  $Q_2 \leftarrow Q_0 + Q_1$ // EC Point addition //
  $Q_0 \leftarrow 2Q_0$ // EC Point doubling //
  $Q_1 \leftarrow Q_{(1+d_i)}$
end
return Q_1

As EC crypto-systems are mostly based on the EC scalar multiplication operation, therefore several methods have been proposed to compute this operation [16]. All

of these methods compute the EC scalar multiplication operation as a sequence of EC point addition (PA) and EC point doubling (PD) operations. Algorithm 11 presents a left-to-right binary method for EC point multiplication. The algorithm encodes the scalar (d) in binary format and always performs EC PD operation as shown in step 3, whereas the EC PA operation is executed if the respective scalar bit is one. The total number of iterations in the algorithm is exactly equal to the number of required bits to represent the scalar d. This technique is often known as standard double-and-add (DA) method for EC scalar multiplication.

Note that in the DA method EC PA is dependent of the respective scalar bit i.e., d_i. In other words the number of EC PA operations depends on the Hamming weight of the scalar d and on an average of any binary number, half of the bits are non-zero. Therefore, the computational complexity of EC point multiplication using DA method is $n \times PD + n/2 \times PA$, where n is the number of bits of d in binary representation.

It is also worth mentioning that the EC PA and PD operations in the DA method can not be executed in parallel. The computational complexities of PA and PD operations are different which is discussed in the next section. Therefore, an attacker can easily distinguish between these two operations by tracing timing and power consumption of the device and ultimately can reveal the bits being processed for the scalar d. Therefore, this method is vulnerable to most of the side-channel-attacks including the very simple timing and simple power analysis attacks [31], [32], [120], [121], [122], [123], [124], [125].

Algorithm 12 shows another method for EC point multiplication, known as double-and-always-add (DAA) [16, 56]. The DAA also works on the binary representation of the scalar d as well. Note that the PA operation in algorithm 12 is not dependant on the bit pattern of d, so these operations can be performed in parallel. As the PD and PA operations can be executed concurrently, therefore the DAA method gives an extra feature of protection against timing and simple power analysis (SPA) attacks [31]. However, it is not adequate to fight against more sophisticated attacks such as differential power analysis [32], differential fault analysis [126, 127] and electromagnetic radiation based attacks [128, 129]. Resistance against these attacks are not the focus of this work.

5.2.1 EC Point Operations Using Affine Coordinates

This work considers an elliptic curve $\mathbb{E}$, defined over prime field $GF(p)$, where p is a large prime characteristic number. Field elements are represented as integers in the range $[0 \rightarrow p-1]$. An elliptic curve $\mathbb{E}$ over $GF(p)$ in short Weierstrass form is represented as

$$\mathbb{E} : y^2 = x^3 + ax + b \tag{5.1}$$

Where, a, b, x, and $y \in GF(p)$ and $4a^3 + 27b^2 \neq 0$ (modulo p). The set of all points (x, y) that satisfy (5.1), plus the point at ∞ make an abelian group. EC point addition and EC point doubling operations over such groups are used to construct many elliptic curve cryptosystems. The EC point addition and EC point doubling operations in affine coordinates can be represented as follows: let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are two points on the elliptic curve. The group operation is the point addition, $P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2)$ which is defined by the group law and is given as

$$x_3 = \lambda^2 - x_1 - x_2 \tag{5.2}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{5.3}$$

where

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \text{ if } P_1 \neq P_2 \\ \frac{3x_1^2 + a}{2y_1} \text{ if } P_1 = P_2 \end{cases} \tag{5.4}$$

If, $P_1 = P_2$ then a special case of adding a point to itself is called the EC point doubling operation. In affine coordinates the EC point addition requires one division (D), two multiplication (M) and six addition or subtraction (A) operations, whereas the PD operation can be performed by using one (D) , three (M) and eight (A) operations.

Tables 5.1 depicts the number of field operations (FOP) for EC PD and PA operations. Therefore, using the DA method for EC scalar multiplication a single iteration of the algorithm requires 14A+5M+2D underlying field operations.

Table 5.1: EC point operations using affine coordinates [16, 28, 29]

Point addition, **(PA)**	Point doubling, **(PD)**	No of field operations **(FOP)**
$x_3 = \lambda^2 - (x_1 + x_2)$	$x_3 = \lambda^2 - (x_1 + x_1)$	**PA** = 6A+2M+1D
$Y_3 = \lambda(x_1 - x_3) - y_1$	$y_3 = \lambda(x_1 - x_3) - y_1$	
$\lambda = (y_2 - y_1)/(x_2 - x_1)$	$\lambda = (3x_1^2 + a)/2y_1$	**PD** = 8A+3M+1D

Point addition (**PA**), Point doubling (**PD**), Field operation (**FOP**), modular addition/subtraction (A), modular division (D)

5.3 EC Scalar Multiplier Architecture in Affine Coordinates

This section presents an architecture to compute the EC scalar multiplication operation using affine coordinates. It consists of three modular arithmetic units, a modular adder/subtractor (A/S), a modular multiplier (Mul) and a modular divider (Div). A/S unit performs either modular addition or subtraction operation at a time in a single clock cycle, while the Div unit takes $2n$ clock cycles to compute an n-bit modular division operation as discussed in Chapter 3. The Mul unit is based on the radix-4 parallel interleaved multiplier (R4PIM) presented in Chapter 4. Implementation results making use of other multipliers presented in Chapter 4 are given in appendix A.

In Chapter 4 it is discussed that the R4PIM multiplier completes an n-bit modular multiplication operation in $\lfloor n/2 \rfloor + 5$ clock cycle. The EC scalar multiplier performance results are shown for different field sizes. The architecture in Figure 5.1 also consists of instruction memory, register file and a control unit. The instruction memory is loaded with micro-coded instructions, the register file is responsible for storing intermediate and final results while the control unit generates the necessary control signals to execute the required operations. The register file consists of two separate sets of registers, dedicated purpose registers (DPR) and general purpose registers(GPR). The scheduling of finite field operations to perform EC PD and PA operations in affine coordinates are given in Table 5.2 and 5.3 respectively.

It is worth mentioning using the DA method EC PD and PA operations can not be executed concurrently. It is also visible from Table 5.2 that scope of parallelism inside these operations are also very limited, therefore, the EC scalar multiplier architecture incorporated a single A/S, Mul and Div units as shown in Figure 5.1.

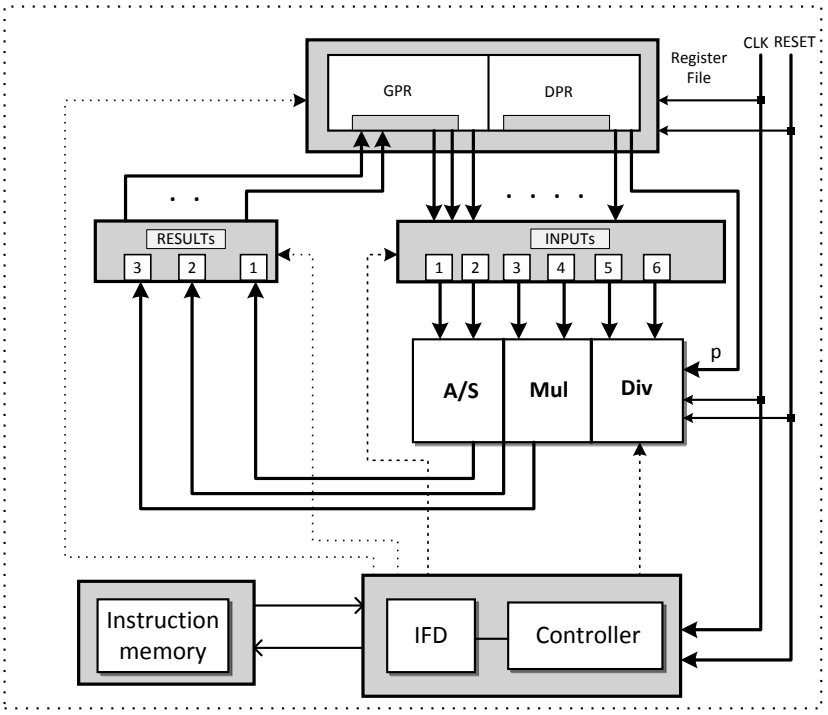


Figure 5.1: EC scalar multiplier architecture using affine coordinates

Normally, to achieve a better performance of EC point multiplication on dedicated hardware, multiple copies of modular adder, subtractor, multiplier and divider units are integrated. These multiple copies can help to execute several operations in parallel at the expense of extra area and cost. As mentioned before when using the DA method, EC PD and PA operations can not be computed in parallel. The scope of parallelism in affine coordinates is also very limited, therefore integration of multiple arithmetic units can not be fully exploited to achieve a significant performance increase.

However, EC point PA and EC point PD operations can be executed in parallel using DAA method for EC point multiplication irrespective of the d_i as shown in algorithm 12. As these EC point operations do not depend on the i^{th} bit of the scalar d, hence, timing and power consumption of these operations are symmetric and it is not be possible for an attacker to extract any information regarding secret value d. Therefore this technique provides a protection against timing and simple power analysis attacks. To defy timing and simple power analysis attacks, the DAA method computes a PA operation in every iteration irrespective of the respective scalar bit. Therefore, it computes 100% more PA operations as compared to the DA method. However, DAA provides a flexibility to compute EC PD and PA operations concurrently. Therefore, this work integrates two instances of the arithmetic unit (AU1 and AU2) to execute EC PD and PA operations as shown in Figure 5.2. Each of these arithmetic units in-

Table 5.2: Scheduling of **PD** operation in affine coordinates

A/S	Mul	Div
$A_1 = y_1 + y_1$ $A_2 = x_1 + x_1$	$M_1 = x_1 \times x_1$	
$A_3 = M_1 + M_1$ $A_4 = A_3 + M_1$ $A_5 = A_4 + a$		
		$D_1 = A_5 \div A_1$
	$M_2 = D_1 \times D_1$	
$A_6(x_3) = M_2 - A_2$ $A_7 = x_1 - A_6$		
	$M_3 = A_7 \times D_1$	
$A_8(y_3) = M_3 - y_1$		

Table 5.3: Scheduling of **PA** operation in affine coordinates

A/S	Mul	Div
$A_1 = y_2 - y_1$ $A_2 = x_2 - x_1$		
$A_3 = x_1 + x_2$		$D_1 = A_1 \div A_2$
	$M_1 = D_1 \times D_1$	
$A_4(x_3) = M_1 - A_3$ $A_5 = x_1 - A_4$		
	$M_2 = D_1 \times A_5$	
$A_6(y_3) = M_2 - y_1$		

corporates a single unit of A/S, Mul and Div, therefore, in total two A/S, two Mul and two Div units are integrated in the overall architecture. AU1 is responsible to execute EC PD operation while AU2 computes EC PA operation according to Table 5.2 and 5.3 respectively. From Table 5.2 and Table 5.3 it is evident that the computational time of an EC PD operation is more than that of an EC PA operation.Therefore, the overall computational time of a scalar multiplication is dependent on the time taken by a PD operation in case of the DAA method given in algorithm 12.

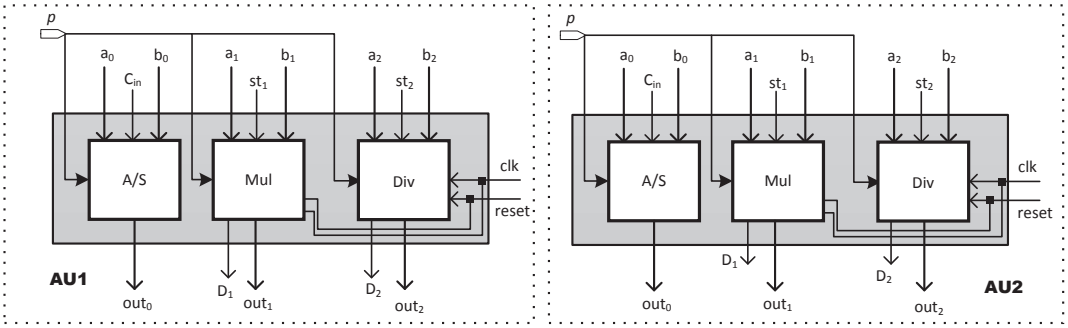


Figure 5.2: Arithmetic units for parallel execution of PD and PA operations

5.3.1 Latency

An EC PD operation using the R4PIM multiplier can be completed in $(7\lfloor n/2 \rfloor + 24)$ clock cycles and an EC PA operation requires $(3n + 19)$ clock cycles.

5.3.2 Using double-and-add (DA) method

As it has been pointed out that the DA method does not provide the flexibility to compute PD and PA operations in parallel. Therefore, using DA method the computational complexity of EC scalar multiplication operation is $n\times$ PD $+n/2\times$ PA. Therefore the latency of the EC scalar multiplication operation is given below.

$$T_{DA} = n(7\lfloor n/2 \rfloor + 24) + n/2(3n + 19) \tag{5.5}$$

Equation (5.5) shows the latency of an EC scalar multiplication operation using the R4PIM multiplier.

5.3.3 Using double-and-always-add (DAA) method

Using the DAA method PD and PA operations can be performed in parallel, therefore the latency of an EC scalar multiplication operation is $n\times$ PD (PD is slower than PA) using the R4PIM multiplier is given in equation (5.6).

$$T_{DAA} = n(7\lfloor n/2 \rfloor + 24) \tag{5.6}$$

5.4 Implementation Results

The implementation results of the EC scalar multiplier architecture using affine coordinates on Virtex-6 FPGA is given in Table 5.4. It is obvious that the the proposed EC scalar multiplier architecture based on DA method is slower than the DAA method. This is because of the parallel execution of the EC PA and PD operations on two arithmetic units using the DAA method. In the case of DA method these EC group operations are executed on a single arithmetic unit.

Table 5.4: Implementation of EC scalar multiplier using affine coordinates

Field size	Area (slices)	Freq. (MHz)	clock cycles	Time (*ms*)
Using DA method on a single AU unit				
192-any	3551	137	191, 904	1.4
224-any	4023	134	258, 384	1.93
256-any	4807	131	336, 256	2.6
Using DAA method on two parallel AU units				
192-any	7152	137	133, 632	0.98
224-any	7976	134	180, 992	1.35
256-any	9213	131	235, 520	1.8

Computation time of an EC scalar multiplication of a 256-bit field size using the DA method with a single arithmetic unit takes 2.6 *ms* and 336,256 clock cycles. It occupies 4807 FPGA slices and is able to operate at a maximum clock frequency of 131 MHz. Using the DAA method and two parallel arithmetic units, a 256-bit EC scalar multiplication operation is completed in 1.8 *ms* in 235,520 clock cycles. It shows that the DAA method, due to parallel execution of the PA and PD operations is 1.4 times faster than the DA method. However, due to utilizing two separate copies of the AU unit it consumes 9213 slices which is 1.9 times more than the design based on the DA method. In [56] dual core architecture of the EC scalar multiplier using DAA method completes the 256-bit operation in 7.7 *ms* which is almost 4.3 times slower than the proposed EC scalar multiplier.

5.5 EC Point Operations Using Projective Coordinates

Elliptic curve point representation in affine coordinates requires modular inversion or division operations to compute both PD and PA operations. It is the most expensive operation in terms of computation time and resource consumption. In order to speed up these group operations different projective coordinates systems have been explored. Using projective coordinates has the advantage of eliminating modular inversion/division from the EC group operations at the cost of more modular multiplication operations. Typically at the end, one or two modular inversions are required to re-map from projective to affine space as shown in Figure 2.4. In this work standard projective coordinates are used in an EC scalar multiplier design.

Table 5.5: EC PD operation in standard projective coordinates [28], [29]

Point addition, **PD**	No Of field operation**(FOP)**
$w = 3(X_1 - Z_1) \times (X_1 + Z_1)$ $s = 2Y_1 \times Z_1$ $ss = s \times s$ $sss = ss \times s$ $R = Y_1 \times s$ $RR = R \times R$ $B = 2 \times X_1 \times R$ $h = w^2 - 2B$ $X_3 = h \times s$ $Y_3 = w \times (b - h) - 2RR$ $Z_3 = sss$	10M+11A

Point addition (**PA**), Point doubling (**PD**), Field operation (**FOP**), modular addition/-subtraction (A),

- In **Projective coordinates** space an affine point $P(x, y)$ corresponds to the point $P(XZ^{-1}, YZ^{-1}, Z)$, where $Z \neq 0$. If set $Z = 1$ then it is trivial to map the points from affine to projective space as shown below.

$$(x, y) \longmapsto (X, Y, 1), \quad X = x, \quad Y = y, \quad Z = 1$$

- At the end of the EC scalar multiplication operation, conversion from projective to affine space is performed as follows:

$$x = XZ^{-1}, \qquad y = YZ^{-1}$$

An elliptic curve in short Weierstrass form in affine coordinates after transformation to projective coordinates is given in equation (5.7) [15], [16]

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{5.7}$$

The EC PA and EC PD formulae reported in [28], [29] and are listed in Tables 5.5 and 5.6 respectively. The number of field operations (FOP) required for an EC PD operation in projective settings is ten modular multiplications (M) plus eleven modular additions (A) i.e., 10M+11A while for EC PA operation the number of required FOP is 14M+7A.

Therefore, a single iteration of DA and DAA algorithms require twenty four time critical modular multiplications and relatively cheaper eighteen modular addition op-

Table 5.6: EC PA operation in standard projective coordinates [28], [29]

Point addition, **PA**	No Of field operations **(FOP)**
$Y_1Z_2 = Y_1 \times Z_2$ $X_1Z_2 = X_1 \times Z_2$ $Z_1Z_2 = Z_1 \times Z_2$ $u = Y_2 \times Z_1 - Y_1Z_2$ $uu = u \times u$ $v = X_2 \times Z_1 - X_1Z_2$ $vv = v \times v$ $vvv = vv \times v$ $R = vv \times X_1Z_2$ $A = uu \times Z_1Z_2 - vvv - 2R$ $X_3 = v \times A$ $Y_3 = u \times (R - A) - vvv \times Y_1Z_2$ $Z_3 = vvv \times Z_1Z_2$	14M+7A

Point addition (**PA**), Point doubling (**PD**), Field operation (**FOP**), modular addition/subtraction (A),

erations. The chosen EC point doubling and EC point addition formulae are based on the assumption that the EC parameter $a = -3$, which is commonly used in many standard elliptic curves. For further details see [28], [29].

5.6 EC Scalar Multiplier Architecture in Projective Coordinates

In this section, a high performance architecture to perform the EC scalar multiplication operation over general prime field is developed. It performs the EC scalar multiplication operation using standard projective coordinates. To compute EC group operations i.e., EC PD and EC PA, a high speed arithmetic unit (AU) is developed first and is described in the next section.

5.6.1 Arithmetic Unit

When EC points are represented in affine coordinates, the scope of parallelism among the underlying finite field arithmetic operations is very limited, integration of multiple arithmetic units does not significantly reduce the computation time of EC scalar multiplication operation. EC points are represented using standard projective coordinates

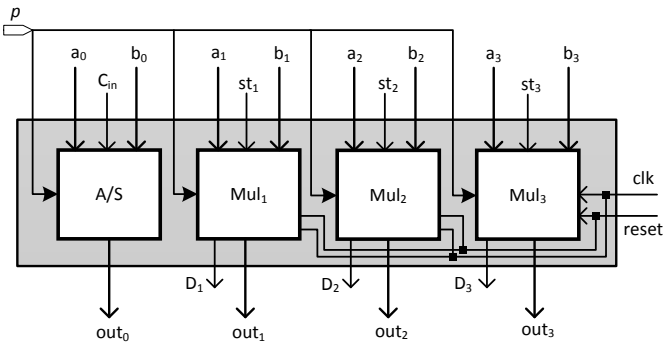


Figure 5.3: Proposed arithmetic unit (AU)

Table 5.7: Field operations on AU unit

Control signals	Field operation	Execution unit	clock cycles
$C_{in} = 0$	$out_0 = a_0 + b_0$	A/S	1
$C_{in} = 1$	$out_0 = a_0 - b_0$	A/S	1
$st_1 = 1$	$out_1 = a_1 \times b_1$	MUL_1	$\lfloor \frac{n}{2} \rfloor + 5$
$st_2 = 1$	$out_2 = a_2 \times b_2$	MUL_2	$\lfloor \frac{n}{2} \rfloor + 5$
$st_3 = 1$	$out_3 = a_3 \times b_3$	MUL_3	$\lfloor \frac{n}{2} \rfloor + 5$

in this section. It is evident from Tables 5.5 and 5.6 that there is a significant scope of parallelism which can be exploited to speed up EC scalar multiplication operation, this is the motivation of integrating parallel dedicated modular multipliers in an AU unit. Latency of a modular addition/subtraction is only a single clock cycle so only a single A/S unit is integrated to reduce the area cost. Therefore a single addition or subtraction instruction can be performed by the AU unit at a time.

The arithmetic unit (AU) shown in Figure 5.3 consists of three dedicated modular multipliers and a single modular adder/subtracter (A/S) units. The three modular multiplication units are named as MUL_1, MUL_2, and MUL_3, where each MUL unit is based on the R4PIM multiplier given in chapter 4 so it performs a modular multiplication instruction in $\lfloor \frac{n}{2} \rfloor + 5$ clock cycles, whereas the A/S unit takes a single cycle to execute a modular addition/subtraction instruction [98]. Therefore, the AU unit is able to execute four independent modular instructions on their respective execution units, concurrently. The AU unit receives eight independent operands (a_0-a_3, b_0-b_3) and produces four independent results (out_0-out_3). Carry in (C_{in}) signal in the A/S execution unit behaves as an operation selection signal, which determine whether modular addition or subtraction is performed. If $C_{in} = 1$, then the A/S unit outputs $out_0 = (a_0 - b_0)$ modulo p, otherwise $out_0 = (a_0 + b_0)$ modulo p. The MUL_{1-3} execution units perform multiplication operations when their respective start signals ($st_1 - st_3$) are set to one. The done flags ($D_1 - D_3$) indicate that the output is available

Table 5.8: Scheduling of **PD** operation using three multipliers in projective coordinates

A/S	Mul_1	Mul_2	Mul_3
$A_1 = X_1 - Z_1$	$M_1 = Y_1 \times Z_1$		
$A_2 = X_1 + Z_1$			
$A_3 = A_1 + A_1$			
$A_4 = A_1 + A_3$			
		$M_2 = A_4 \times A_2$	
$A_5 = M_1 + M_1$			
	$M_3 = A_5 \times A_5$	$M_4 = M_2 \times M_2$	$M_5 = Y_1 \times A_5$
	$Z_3 \leftarrow M_6 = M_3 \times A_5$	$M_8 = M_5 \times M_5$	$M_7 = X_1 \times M_5$
$A_6 = M_7 + M_7$			
$A_7 = A_6 + A_6$			
$A_8 = M_4 - A_7$			
$A_9 = A_6 - A_8$	$X_3 \leftarrow M_9 = A_8 \times A_5$		
			$M_{10} = M_2 \times A_9$
$A_{10} = M_8 + M_8$			
$Y_3 \leftarrow A_{11} = M_{10} - A_{10}$			

at the respective output ports (out_1-out_3), which are set to one after $\lfloor \frac{n}{2} \rfloor + 5$ clock cycles. The instructions with their corresponding execution units and control signals are shown in Table 5.7, where on the basis of the control signals the respective execution unit is active and performs the respective operation. The given AU unit takes eight n-bit input operands produces four n-bit outputs, while the prime modulus p is made directly available to the four execution units.

5.6.2 Scheduling of PD and PA Operations

The architecture in Figure 5.3 computes EC group operations **PD** and **PA** in standard projective coordinates [28]. These **PD** and **PA** operations require a number of modular additions, subtractions, and multiplications which are listed in Tables 5.5 and 5.6. Table 5.5 also depicts that a **PD** computation requires 10M+11A operations, while 14M+7A operations are required to perform a **PA** operation in Table 5.6. As in the DA algorithm these point operations can not be performed in parallel, therefore, a single iteration of the algorithm requires twenty four modular multiplications and eighteen modular addition/subtraction operations. Scheduling of these modular operations to perform PD and PA operations are demonstrated in Table 5.8 and 5.9 respectively. These operation schedules are based on the AU unit that consists of three dedicated multipliers and a single A/S unit. Table 5.8 depicts that the field operations sequence

Table 5.9: Scheduling of **PA** using three multipliers in projective coordinates

A/S	Mul_1	Mul_2	Mul_3
	$M_1 = Y_1 \times Z_2$	$M_2 = X_1 \times Z_2$	$M_3 = Z_1 \times Z_2$
	$M_4 = Y_2 \times Z_1$	$M_5 = X_2 \times Z_1$	
$A_1 = M_4 - M_1$			
$A_2 = M_5 - M_2$	$M_6 = A_1 \times A_1$		
		$M_7 = A_2 \times A_2$	
	$M_8 = M_7 \times M_2$	$M_9 = M_6 \times M_3$	$M_{10} = M_7 \times A_2$
$A_3 = M_8 + M_8$			$M_{12} = M_{10} \times M_1$
$A_4 = M_9 - M_{10}$			
$A_5 = A_4 - A_3$			
$A_6 = M_8 - A_5$	$X_3 \leftarrow M_{11} = A_2 \times A_5$	$Z_3 \leftarrow M_{14} = M_{10} \times M_3$	
			$M_{13} = A_1 \times A_6$
$Y_3 \leftarrow A_7 = M_{13} - M_{12}$			

for a PD operation in which M_{1-10} indicate modular multiplications and A_{1-11} represent modular addition/subtraction operations. X_3, Y_3, Z_3 are the resultant coordinates of **PD** operation of point $R(X_1, Y_1, Z_1)$.

A PA operation requires 14M+7A field operations, which are executed in the sequence given in Table 5.9, where X_3, Y_3, Z_3 are the coordinates of the resulting point of adding points $Q(X_1, Y_1, Z_1)$ and $R(X_2, Y_2, Z_2)$. Note that, these low level field operations can be scheduled in several ways, however the schedule in the tables require minimum number of clock cycles when using three parallel modular multipliers. Data dependency graphs of PD and PA operations using three parallel multipliers are shown in Figures 5.4 and 5.5 respectively.

The scheduling of the underlying field operations in the DAA algorithm is given in Table 5.10 and a data dependency graph is shown in Figure 5.6. It is shown that using four parallel multipliers a single iteration of the DAA algorithm can be completed in $n+35$ clock cycles where each MUL_{1-4} takes $\lfloor n/2 \rfloor + 5$ clock cycles to perform an n-bit modular multiplication operation using the R4PIM multiplier.

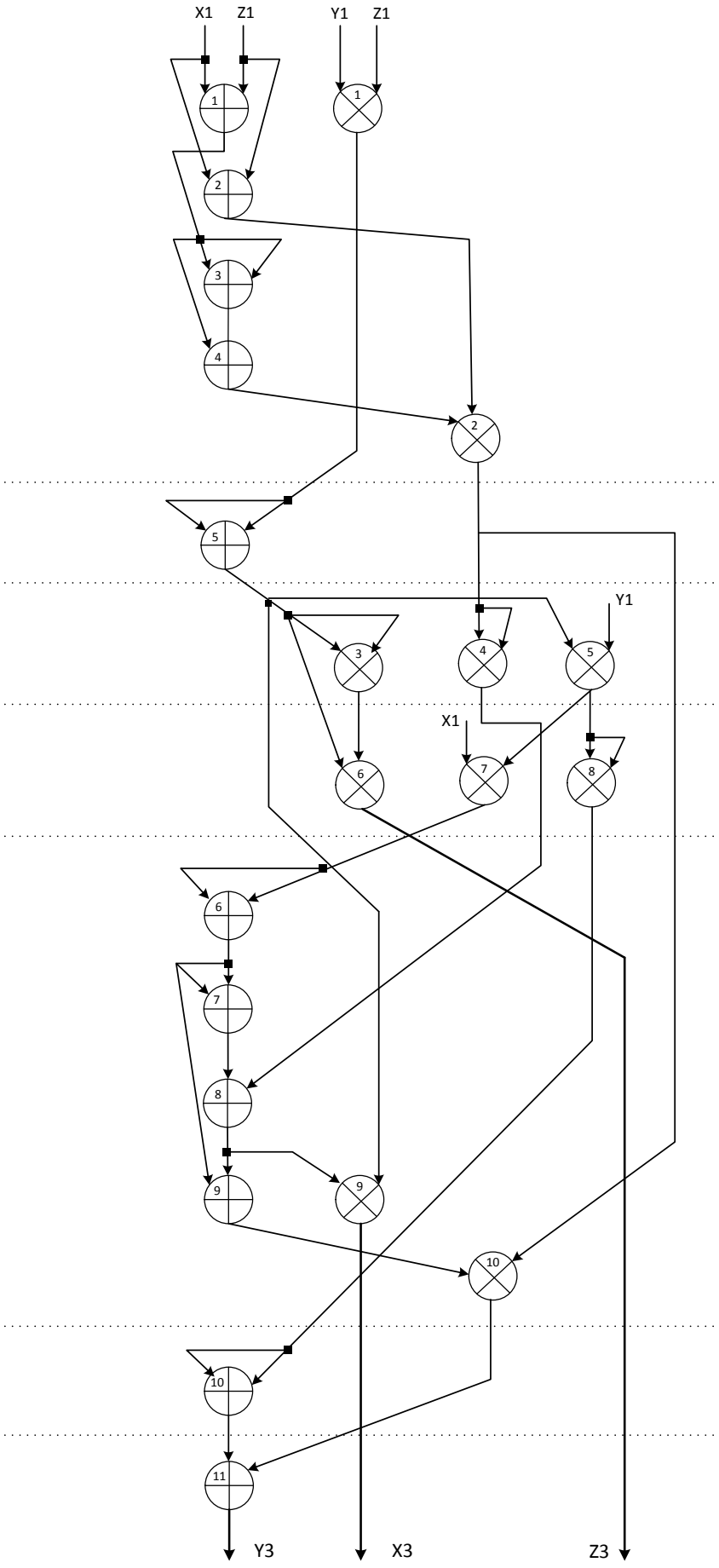


Figure 5.4: Data dependency graph of PD operation using three multipliers

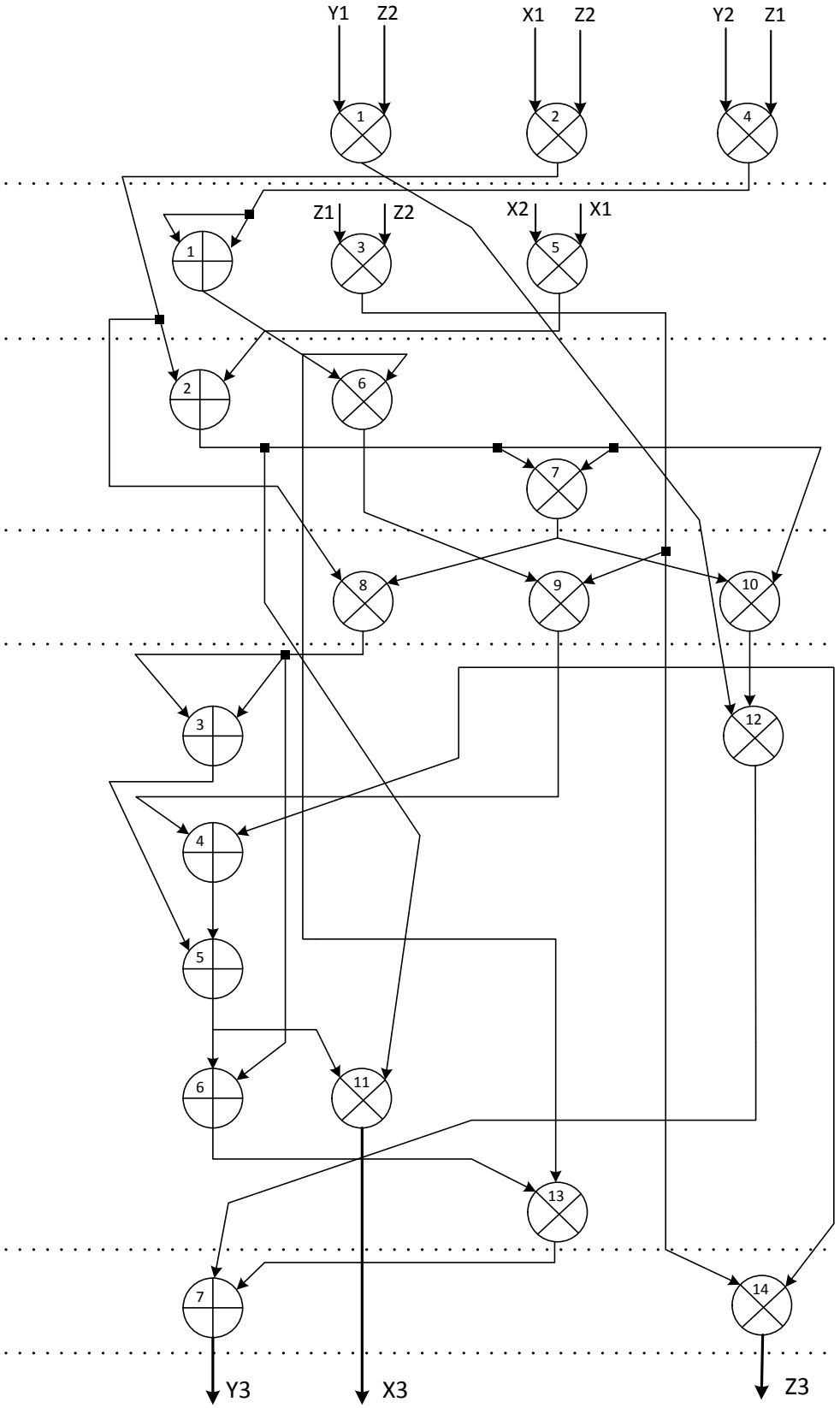


Figure 5.5: Data dependency graph of PA operation using three multipliers

Table 5.10: Scheduling of parallel **PD PA** operations using four multipliers

A/S	Mul_1	Mul_2	Mul_3	Mul_4
$A_1 = X_1 - Z_1$	$M_1 = Y_1 \times Z_2$	$M_2 = X_1 \times Z_2$	$M_3 = Z_1 \times Z_2$	$M_4 = Y_1 \times Z_1$
$A_2 = X_1 + Z_1$				
$A_3 = A_1 + A_1$				
$A_4 = A_3 + A_1$	$M_5 = A_4 \times A_2$	$M_6 = Y_2 \times Z_1$	$M_7 = X_2 \times A_1$	$M_8 = A_5 \times A_5$
$A_5 = M_4 + M_1$				
		$Z_3^d \leftarrow M_{10} = M_8 \times A_5$		
$A_6 = M_6 - M_1$	$M_9 = Y_1 \times A_5$		$M_{11} = A_6 \times A_6$	$M_{12} = A_7 \times A_7$
$A_7 = M_7 - M_2$				$M_{16} = M_5 \times M_5$
	$M_{13} = M_9 \times M_9$	$M_{14} = X_1 \times M_9$	$M_{15} = M_{12} \times A_7$	
$A_8 = M_{14} + M_{14}$	$M_{17} = M_{15} \times M_2$	$M_{18} = M_{11} \times M_3$	$M_{19} = M_{15} \times M_1$	
$A_9 = A_8 + A_8$				
$A_{10} = M_{16} - A_8$				$X_3^d \leftarrow M_{20} = A_{10} \times A_5$
$A_{11} = M_{13} + A_{13}$				
$A_{12} = M_{13} + M_{13}$				
$A_{13} = M_{17} + M_{17}$	$Z_3^a \leftarrow M_{21} = M_{15} \times M_3$			
$A_{14} = M_{15} + A_{13}$				
$A_{15} = M_{18} - A_{14}$				
$A_{16} = M_{17} - A_{15}$		$X_3^a \leftarrow M_{22} = A_{15} \times A_7$	$M_{23} = A_{11} \times M_5$	$M_{24} = A_{16} \times A_6$
$Y_3^d \leftarrow A_{17} = M_{23} - A_{12}$				
$Y_3^a \leftarrow A_{18} = M_{24} - M_{19}$				

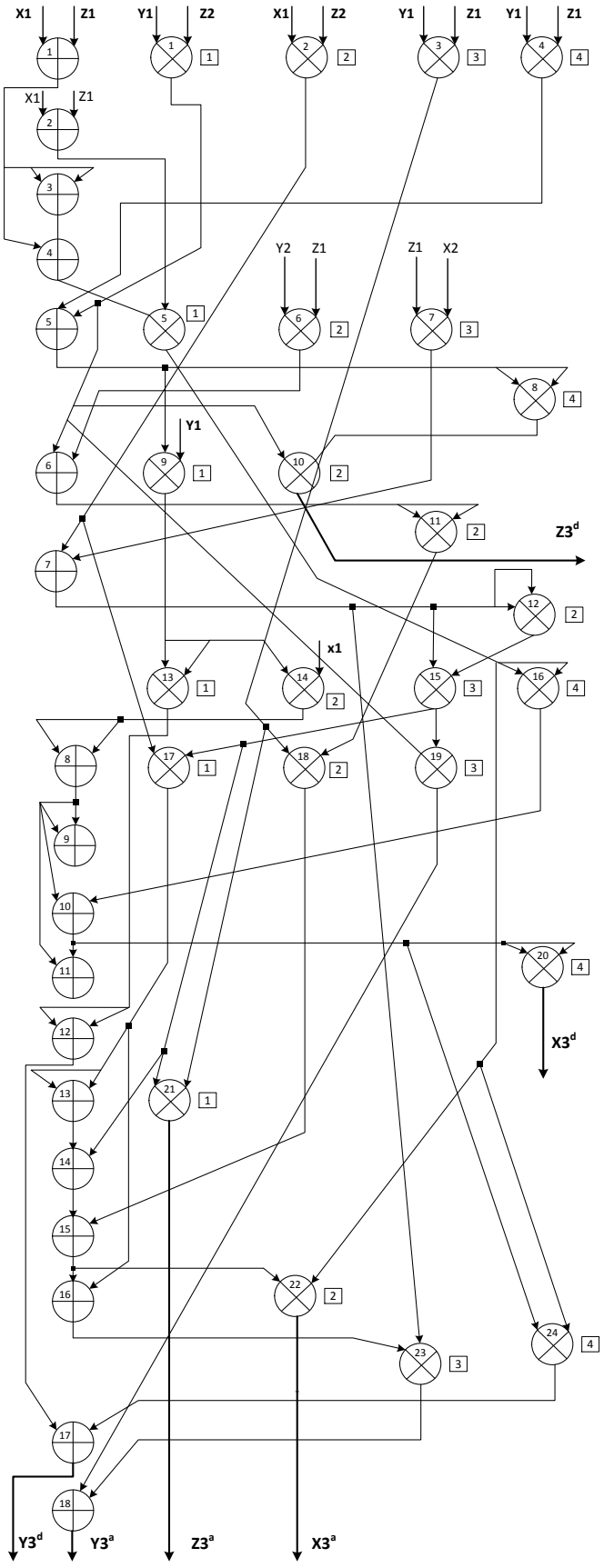


Figure 5.6: Data dependency graph of concurrent PA and PD operations using four multipliers

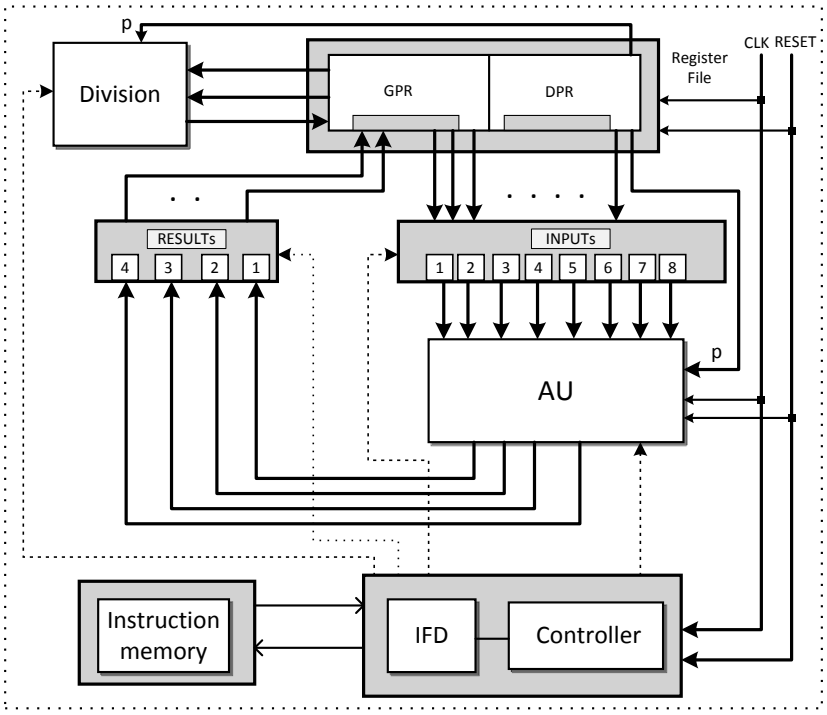


Figure 5.7: EC scalar multiplier architecture

5.6.3 Overall Execution

The overall architecture of the proposed EC scalar multiplier using projective coordinates is given in Figure 5.7. In addition to the AU unit, it also contains a register file, instruction fetch and decode unit (IFD), instruction memory, and a controller. In order to configure the AU unit to perform the respective operations, instructions are loaded in the instruction memory. The controller generates a request to the IFD unit. Then, the IFD unit fetches and decodes an instruction and generates appropriate signals to configure the AU unit for smooth execution of the instruction.

As the instruction data dependencies and data hazards are known in advance, therefore these microcoded instructions can be scheduled in a manner to get maximum utilization of MUL_1, MUL_2, and MUL_3 execution units of the AU based on the schedules demonstrated in the Tables 5.8, 5.9, 5.10.

The AU unit performs four instructions in parallel, therefore it accesses eight independent input operands in parallel and produces four independent results. The register file stores input points coordinates, modulus p, intermediate results, and the coordinates of the resultant point. A very simple read and write mechanism for the register file is adopted, where each data access (read/write) is based on hard-coded control signals for the inputs and results multiplexing blocks, which are generated and

Table 5.11: No of Clock cycles of EC scalar multiplication in projective coordinates

Method	No of Mul	PD	PA	EC scalar multiplication
DA	2	$(5n/2+32)$	$(7n/2+38)$	$[n(5n/2+32)+n/2(7n/2+38)]+4n$
	3	$(2n+27)$	$(3n+37)$	$[n(2n+27)+n/2(3n+37)]+4n$
	4	$(2n+27)$	$(2n+28)$	$[n(2n+27)+n/2(2n+28)]+4n$
DAA	2	$6n+61$		$[n(6n+61)]+4n$
	3	$4n+43$		$[n(4n+43)]+4n$
	4	$3n+35$		$[n(3n+35)]+4n$

managed by the controller and this avoids additional software development cost. It is also worth mentioning that read and write to specific registers are completed in the same clock cycle, which avoids unnecessary delay. The register file is grouped into general purpose (GPR) and dedicated purpose registers (DPR). The GPR is updated with intermediate results, while DPR holds the input operands in every iteration of the DA and DAA algorithms.

5.6.4 Final Conversion

At the end of an EC scalar multiplication, the resultant coordinates (X, Y, Z) in standard projective coordinates need to be converted back to affine coordinates (x, y). This conversion is done as $x = X/Z, y = Y/Z$, which requires two modular division instructions. These modular division instructions are executed on the dedicated division units (presented in chapter 3) based on extended Euclidean algorithm [16] in $4n$ clock cycles ($2n$ for each).

5.6.5 Latecny

Latencies of a single iteration of DA and DAA algorithms using multiple parallel multipliers are given in Table 5.11. The table shows that using four parallel multipliers a single iteration of DAA algorithm is completed in $(3n+35)$ clock cycles. In total, there are n iterations therefore EC scalar multiplication operation is completed in $n(3n+35)+4n$ clock cycles. The table also shows that using DAA algorithm, using four multipliers the latency of a EC scalar multiplication operation is $[n(2n+27)+n/2(2n+28)]+4n$

Table 5.12: Latency of EC scalar multiplication in projective coordinates

Method	Mul	n	Clock cycles			Freq (MHz)	Time (ms)
			PD	PA	EC Scalar Multiplication		
DA	2	192	512	710	167, 232	154	1.08
		224	592	822	225, 568	150	1.5
		256	672	934	276, 096	146	2.01
	3	192	411	613	138, 528	152	0.91
		224	475	709	185, 808	147	1.26
		256	539	805	242, 048	143	1.69
	4	192	410	412	119, 232	151	0.79
		224	475	486	160, 832	145	1.10
		256	539	540	208, 128	141	1.47
Parallel PA and PD operations							
DAA	2	192	1213		233, 664	154	1.51
		224	1405		315, 616	150	2.1
		256	1597		409, 856	146	2.8
	3	192	811		156, 480	152	1.03
		224	939		211, 232	147	1.44
		256	1067		274, 176	143	1.92
	4	192	611		118, 080	149	0.78
		224	707		159, 264	145	1.09
		256	803		206, 592	141	1.46

clock cycles. Note that extra $4n$ clock cycles in Table 5.11 are consumed in the final conversion i.e., from projective to affine coordinates.

5.7 Implementation and Results

The EC scalar multiplier architectures are coded in Verilog (HDL). Xilinx ISE 14.1 design suite is used for synthesis, mapping, placement, and routing purposes and Xilinx ISim simulator is used for simulation purposes. The R4PIM modular multiplier described and implemented in chapter 4 is used. The EC points are represented in standard projective coordinates and the EC group operations are computed over standard form of an EC. Tables 5.12 depicts latencies of EC PA , PD and scalar multiplication operation against different field sizes when computed using DA and DAA methods by incorporating different number of parallel multiplier units.

During synthesis Xilinx Virtex-6 device is selected as the target implementation platform. The available fast carry chains of Xilinx FPGA are utilized to perform addition and subtraction operations. On the top level, the EC scalar multiplier is based

Table 5.13: Implementation results of EC scalar multiplier in projective coordinates

Method	Multipliers	n	Area (slices)	ATB	TP (ops)
DA	2	192	5251	29.5	925.9
		224	6048	40.5	666.6
		256	7089	55.6	497.5
	3	192	6952	32.9	1098.9
		224	8108	45.6	793.6
		256	9372	61.8	591.7
	4	192	8731	35.9	1265.8
		224	10, 115	49.6	909
		256	11, 655	66.9	680.3
DAA	2	192	5432	42.7	666.6
		224	6341	59.4	476.2
		256	7235	79.1	357.1
	3	192	7113	38.15	970.8
		224	8341	53.6	694.4
		256	9588	71.9	520.8
	4	192	8897	36.14	1282
		224	10, 291	50.07	917.4
		256	11, 791	67.24	684.9

(Area × Time)/bits (ATB), Throughput (TP), EC scalar multiplication operations per second (ops)

on the DA and DAA algorithms. The DAA technique offers inherent protection against timing and simple power analysis attacks. The presented designs are programmable for various field sizes of $p \leq 256$-bit.

It is obvious that incorporating more multiplier units reduces the computation time of an EC scalar multiplication operation at the cost of more hardware resources. Note that in the case of using four parallel multiplier units computation time of an EC scalar multiplication operation using DA and DAA methods are comparable. Table 5.12 shows that a 256-bit EC scalar multiplication operation using DA method is completed in 208,128 clock cycles using four multipliers. The same bit length EC scalar multiplication operation with the same multipliers using DAA method is completed in 206,592 clock cycles.

Tables 5.12 and 5.13 show that the DA method with three multipliers on Virtex-6 platform, a 256-bit EC scalar multiplication is completed in 1.69 *ms* in a cycle count of 242K. It consumes 9372 slices and runs at a maximum frequency of 143 MHz with a throughput of almost 592 operations per second (*ops*). Similarly, it is depicted that when using four multipliers the same bit length operation is completed in 1.47 *ms*,

consumes 11.65K slices and achieves a throughput of 680 ops. Which is 15% faster as compared to the case of using three multipliers. However due to the integration of an extra multiplier unit it consumes 20% more FPGA slices. Note that in the DA method PA and PD operations are easily distinguishable because PA operation is dependent on the bit value of the scalar d. Thus, the scalar d can be potentially revealed by measuring timing or power consumption of the cryptographic device. Therefore DA method is very susceptible to these side channel attacks.

Tables 5.12 and 5.13 also demonstrate an implementation of scalar multiplication using DAA method on Virtex-6 FPGA platform. In the DAA method PD and PA operations can be performed in an indistinguishable manner irrespective of the bit value of the scalar d. A 256-bit EC scalar multiplication operation using four multipliers is completed in 1.46 ms, consumes 11.79K slices and attains a maximum frequency of 141 MHz with a throughput of 684.9 ops. Using DAA method for EC scalar multiplication can protect the scalar d against timing and simple power analysis attacks.

It is obvious that incorporating more parallel multipliers will increase area consumption and reduce the computation time of a EC scalar multiplication operation. Area-delay product per bit (ATB) for a number of parallel multipliers is also given in Table 5.13. ATB is higher for more multipliers with increased throughput rate. Therefore, a choice of the number of multipliers depends on an application requirements. If the application demands a very high speed design then four multipliers is the most suitable option.

Table 5.14: Comparison of FPGA implemented EC scalar multipliers

Ref.	Platform	Algorithm	$\|p\|$ (bits)	Area	Freq. (MHz)	cycle count	Time (ms)	TP (ops)	**TPAR**	**DPR**
This work	Virtex-6	DA[a]	192-any	**6.95K Slices**	152	**138.5K**	**0.91**	1.1K	×	×
			224-any	**8.1K Slices**	147	**185.8K**	**1.26**	794	×	×
			256-any	**9.37K Slices**	143	**242.04K**	**1.69**	592	×	×
		DA[b]	192-any	**8.73K Slices**	151	**119.2K**	**0.79**	1.4K	×	×
			224-any	**10.11K Slices**	145	**160.8K**	**1.10**	909	×	×
			256-any	**11.65K Slices**	141	**190K**	**1.47**	680	×	×
		DAA[b]	192-any	**8.9K Slices**	149	**118K**	**0.78**	1.28K	√	×
			224-any	**10.3K Slices**	145	**159.3K**	**1.09**	917	√	×
			256-any	**11.8K Slices**	141	**206.5K**	**1.46**	684.9	√	×
[55]	Virtex-II pro	DAA	192-any	09.0K slices	43	192K	4.47	223.7	√	√
			224-any	10.4K slices	40	260K	6.50	153.8	√	√
			256-any	12.0K slices	36	338K	9.38	106	√	√
[55]	Virtex-4	DAA	192-any	-	61	192K	3.15	317	√	√
			224-any	-	58	260K	4.49	223	√	√
			256-any	-	54	338K	6.26	158	√	√
[56]	Virtex-4	DAA	192-any	14.9K slices	53	186K	3.5	286	√	×
			224-any	17.3K slices	47	253K	5.4	185	√	×
			256-any	20.1K slices	43	330K	7.7	130	√	×
[46]	Virtex-4	DA	192-NIST	20.8K slices + 32 DSPs	60		4.8	208	×	×
			224-NIST	20.8K slices + 32 DSPs	60	-	5.8	172.4	×	×
			256-NIST	20.8K slices + 32 DSPs	43		6.9	145	×	×
[71]	Virtex-5		192-any	6.1K LUTs	96.6	-	2.05	487.8	×	×
			256-any	7.8K LUTs	81.7		4.04	247.5	×	×
[47]	Virtex-6	DAA	192-NIST	33K LUTs + 289 DSPs	100	-	0.30-3.91	255-1.02K	√	√
[60]	Virtex-II pro	DA	192-any	15.8K slices + 256 DSPs	40	151.4K	3.86	259	×	×
[63]	Virtex-II pro	NAF	192-any	40.3K slices	94.7	118.1K	1.25	800	×	×
			256-any	41.6K slices	94.7	252K	2.66	376	×	×
Our	Virtex-4	DAA[b]	192-any	**27.3K slices**	71	**117.8K**	**1.66**	602	√	×
			224-any	**33.1K slices**	67	**159.3K**	**2.37**	422	√	×
			256-any	**37.9K slices**	63	**206.5K**	**3.27**	306	√	×

Throughput (TP), operations per second (ops), Timing and simple power analysis resistance (**TPAR**), Differential power analysis resistance (**DPR**) Double-and-add (DAA), Always double-and-add (ADA), Non-adjacent-form (NAF), Three multipliers (a), Four multipliers (b).

5.7.1 Performance Evaluation

FPGA implementation of several other related designs are listed in Table 5.14. These listed designs differ in different aspects: underlying implementation platform, chosen prime number characteristics, underlying elliptic curve representation, points representation, and the ability to countering different side channel attacks. A braod overview the proposed design against other designs in different performance metrics are listed in Table 5.14.

Designs reported in [46], [47] are based on **NIST** recommended elliptic curves over prime fields, where a prime modulus p is of special form (close to a power of 2) called pseudo-Mersenne Prime. Modular multiplications over this form of prime are much faster due to simpler reduction steps which can be achieved by a few addition and subtraction operations instead of division as required in the case of a general prime field. Therefore, typically implementations of NIST recommended curves are faster than the implementations of general curves. However, these designs are not compatible to any other prime numbers. Hence, NIST based EC scalar multiplier designs are not flexible to support any prime numbers of the chosen bit sizes. Moreover, scalability of these designs is also a major problem. For example, it is not easy to extend a 192-bit EC scalar multiplier over NIST curves to a multiplier over 224-bit NIST curves.

Virtex-4 implementation of [46] completes a 256-bit EC scalar multiplication in $6.1ms$ at 43 MHz clock frequency, occupies 20.1K slices and 32 DSPs blocks (16×16 embedded multipliers). At the top level DA algorithm is used for EC scalar multiplication operation. It is 2.2 times slower than the proposed design, and it also lacks the ability to resist timing and simple power analysis attacks. [47] extends the design in [46] to increase performance and side channel attacks resistivity. Its implementation on a Virtex-6 computes an EC scalar multiplication between $0.3ms$ to $3.91ms$ for 192 to 512-bits prime modulus p. It occupies 11.2K slices (33K LUTs), 289 DSPs blocks (18×18 multipliers), and 128 RAMB36 (36K random access memory). On the same platform it consumes the same amount of FPGA slices as compared to our design, note that it does not include the logic utilization of 289 DSPs and the 128 RAM36 blocks, moreover its flexibility is only limited to the NIST recommend curves.

Designs in [60], [63] and [71] support general prime field, with $p \leq 256$-bits, however these designs are either based on double-and-add (DA) or non-adjacent-form

(NAF) techniques to perform EC scalar multiplication. Using the DA method, an n-bit EC scalar multiplication takes (n) **PD** + $\lceil \frac{n}{2} \rceil$ **PA**, whereas using NAF, it costs (n) **PD** + $\lceil \frac{n}{3} \rceil$ **PA** which is almost 33% decrease in **PA** operations as compared to the DA method. However, as data dependencies and computational complexities of PD and PA are different therefore, these methods are very susceptible to side channel attacks (even vulnerable against timing and simple power analysis (SPA) attacks). The design presented in this thesis is superior to these in terms of resistivity to timing and SPA attacks.

In [56], propose an elliptic curve scalar multiplier architecture over general prime field resilient to timing and power analysis attacks. The design performs **PD** and **PA** operations using affine coordinates (x, y) which also require modular inversion/division operations in addition to the field addition, subtraction, and multiplication. Its arithmetic unit employs two modular addition, two modular subtraction, two modular multiplication, and two modular division units. The Virtex-4 FPGA implementation of the design computes a 256-bit EC scalar multiplication in 7.7 *ms*, cycle count of 330K, runs at a maximum clock frequency of 43 MHz, and occupies 20.1K slices. The design presented in this thesis (DAA using four R4PIM multipliers) on the same platform is 2.25 times faster and consumes only 1.8 times more Virtex-4 FPGA slices. The increase in slice area is mainly due to the parallel multiplier units, whereas [56] employs serial radix-2 multiplier. The design in [56] also requires 1.92 times more clock cycles to compute a single EC scalar multiplication operation as compared to the presented design.

The design reported in [55] proposes a compact programmable arithmetic unit (PAU) to perform finite field arithmetic operations. Then, an EC scalar multiplier architecture is presented based on dual instances of the PAU. EC points are represented in affine coordinates, and Montgomery laddering method for EC scalar multiplication is adopted to perform **PD** and **PA** operations in parallel. Its implementation on Virtex-2 pro completes a 256-bit EC scalar multiplication in 9.38 *ms*, achieves maximum frequency of 36 MHz, cycle count of 338K, and consumes 12K slices. In [55] (Table XIV) only timing results of its implementation on Virtex-4 are also given, which show that, it completes a 256-bit EC scalar multiplication in 6.26 *ms*, which is 2.23 times slower, and its cycle count is 1.97 times higher as compared to the presented design on the same Virtex-4 platform.

As in Table 5.4 implementation results of the EC scalar multiplier in affine coordinates are presented. Using DA method, a 256-bit EC scalar multiplication is completed in 2.6 *ms* and consumes 4807 Virtex-6 FPGA slices in 336,256 clock cycles. On the other hand, Table 5.14 shows the same bit length operation on the same FPGA device using projective coordinates is completed in 1.69 *ms* and consumes 9370 Virtex-6 FPGA slices in 242,004 clock cycles, which is almost 1.54 times faster than the affine coordinates implementation. However, due to integration of multiple copies of the R4PIM multipliers in projective coordinates, its area consumption is 1.94 times of the implementation in affine coordinates. Similar conclusions can be made for the DAA method in affine and projective coordinates. Therefore, projective coordinates offers better performance which relies only on the performance of a finite field multiplier. In the case of affine coordinates, optimization of a finite field multiplier and a divider is important to boost the overall performance.

Most of the available high-speed EC scalar multiplier designs on FPGA platforms extensively use dedicated built-in blocks such as digital signal processing (DSP) and embedded multipliers. This research work does not use these embedded blocks except look-up-tables and the fast carry chains (FCC) of the FPGA is used to reduce the long carry propagation delay in an adder circuit. Hence, presented designs in this work are more portable to other FPGA devices and ASIC technologies.

All of the designs in Table 5.14 are based on standard elliptic curve representation (Weierstrass). The number of Underlying field operations to compute PD and PA operations are more than the some other forms of ECs. Recent advances in EC cryptography is to perform scalar multiplication on new forms of EC other than the Weierstrass form. Examples of these are Edwards curves [115], Twisted Edwards curves [116], Montgomery curves [117, 118]. These curves require fewer field multiplications to compute PD and PA operations as compared to the curves in Weierstrass form [29]. Therefore computation time of scalar multiplication on these curves are lower than the standard Weierstrass form. Some of these curves have unified formula for PD and PA [116], thus they may have inherent resistance for different side channel attacks. Hardware analysis of these curves are reported in [43, 70, 130]. Despite their numerous advantages these curves are not standardized yet. Presented designs for low level field operations and EC scalar multiplication in this work supports general prime field which means that these designs are flexible to work for any prime number p. There-

fore, the presented architectures for finite field arithmetic in this work can be utilized to construct elliptic curve cryptographic processors over several new forms of elliptic curves [74].

It is worth mentioning that most of the designs in Table 5.14 including the architectures presented in this thesis are not capable of resisting differential power analysis attacks [32]. DPA tries to reveal the secret by monitoring several power traces and find patterns by applying some statistical methods.

5.8 Conclusion

This chapter introduces novel hardware architectures to compute the EC point multiplication operation over general prime field. Two EC scalar multiplication algorithms, double-and-add and double-and-always-add are implemented. The chapter first presents hardware architectures for EC scalar multiplication algorithms by using affine coordinates representation of the EC points. As in affine coordinates the scope of parallelism in the underlying field operations is very limited therefore multiple copies of an arithmetic unit can not accelerate the respective operations. The division operation required to compute EC PA and PD operations in affine coordinates also limits the performance of an EC scalar multiplier in affine coordinates.

This chapter also presents an EC scalar multiplier architecture using projective coordinates. In projective coordinates EC PD and PA operations are inversion/division free, and there are also opportunities to exploit parallelism in the computation of EC PD and PA operations. A number of modular multipliers are used in the design of the arithmetic unit to exploit the available parallelism.

The presented architectures are synthesised targeting Xilinx Virtex-6 FPGA platform. Using the DA algorithm and employing three multipliers, a 256-bit EC scalar multiplication can be computed in 1.69 *ms* in a cycle count of 242K. When four multipliers are employed the computation time for the same operation is reduced to 1.47 *ms* with a cycle count of 208.1K. Similarly, using the DAA algorithm and employing four multipliers, computation time for a 256-bit EC scalar multiplication operation is 1.46 *ms* in a cycle count of 206.59K.

The synthesis results confirm that the EC scalar multiplier presented in this thesis is a good trade-offs of performance and flexibility. The other major advantage of the proposed designs using DAA method is the ability to counter side channel attacks based on timing and simple power consumption analysis of the cryptographic device.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The contribution of this research work is mainly comprises of efficient hardware architectures for finite arithmetic operations including addition, subtraction, multiplication, and division. Based on these optimized arithmetic primitives, high performance hardware architectures for elliptic curve scalar multiplication operation are proposed.

Chapter 2 discusses essential background knowledge of elliptic curve cryptography (ECC) which establishes that an elliptic curve (EC) scalar multiplication is the fundamental operation in the construction of ECC based crypto-systems. In this regards, common optimization techniques and available hardware implementations of EC scalar multiplication are analysed.

In Chapter 3, strategies to perform addition, subtraction, multiplication and division in a finite field of prime characteristics are discussed. Hardware architectures to execute these basic finite field operations are also presented. For finite field inversion/division, the extended Euclidean algorithm is adopted while the interleaved modular multiplication algorithm is used to perform a finite field multiplication. Finite field multiplication is a core operation in all public key based cryptosystems. The performance of these cryptosystems can be significantly enhanced by incorporating an optimized finite field multiplier. Therefore, this chapter also presents modifications to

the interleaved modular multiplication algorithm based on radix-4, radix-8 and Booth encoding techniques. Subsequently, efficient finite field multiplier architectures are presented based on the modified interleaved modular multiplication algorithms.

In Chapter 4, the finite field multipliers presented in Chapter 3 are further optimized by introducing parallelism to perform critical operations concurrently. Due to the introduced parallelism, the parallel finite field multipliers have shorter critical path delays and are able to achieve higher operating clock frequencies. Then, the performance of these parallel multipliers are evaluated on the basis of computation time, resource consumption, throughput and operating frequency.

Chapter 5 presents hardware architectures to execute EC scalar multiplications. The underlying finite field operations to perform an EC scalar multiplication operation in affine coordinates are modular addition, subtraction, multiplication, and division. Therefore, the arithmetic unit in an EC scalar multiplier design is comprised of modular addder/subtractor, modular multiplier and modular divider units. On the system level the standard double-and-add (DA) and double-and-always-add (DAA) methods are adopted to perform an EC scalar multiplication operation. In the DA algorithm point addition and point doubling operations can not be performed concurrently. On the other hand the DAA algorithm provides a flexibility to execute these point addition and doubling operations in parallel. Therefore, in the implementation of the EC scalar multiplier using the DAA algorithm, two arithmetic units are incorporated to execute point addition and doubling operations concurrently.

As the computational complexity of a modular division operation is more than a modular multiplication, the standard projective coordinates are used to remove this division operation from the computation of EC group operations. At the end of an EC scalar multiplication operation in projective coordinates, two division operations are needed to convert the result back to affine coordinates. This work uses the standard projective coordinate system to perform these point operations. In projective coordinates, there are many opportunities to execute the underlying field operations in parallel. Therefore, the EC scalar multiplier using standard projective coordinates employs different numbers of parallel multipliers. A performance evaluation is presented based on computation time, resource consumption and throughput. Using the DA method, performance is evaluated by employing three and four parallel multipliers and for the DAA method, performance is shown for the architectures using four

modular multipliers. Finally, comparison with state-of-the-art designs in the literature is presented which shows that the architectures proposed in this work are good trade-offs between performance and flexibility. The presented designs provide the flexibility for the user to select a prime number and the EC parameters. Therefore, the user can update these parameters to avoid any security breach.

Implementation results of EC scalar multipliers based on the other modular multipliers presented in Chapter 4 are given in Appendix A.

6.2 Future Work

There are several possible extensions of the work presented in this thesis. Some of these are highlighted below.

- In this thesis, the fast carry chains (FCC) of FPGA are used to reduce long carry propagation delay in adders. An important extension of the work presented in this thesis is to design a high-speed adder circuit incorporating different fast addition techniques such as carry save addition, redundant signed digit addition etc. Then analyse the performance of modular multipliers and EC scalar multipliers based on the high-speed adder.

- Resource requirements of high-radix parallel modular multipliers presented in Chapter 4 are significantly higher than their radix-2 counterparts. One possible future direction is to eliminate redundant operations to optimize the resource consumption of higher-radix parallel modular multipliers.

- The general side-channel attack is based on timing and is known as the timing attack. The EC scalar multiplier design in this work is resilient to this attack by using the DAA method to compute the EC scalar multiplication operation. Side-channel attack based on power consumption has been categorized into simple power analysis (SPA) and differential power analysis (DPA). SPA uses a single trace of power consumption and the DAA method is resistant to this type of attacks. On the other hand DPA uses statistical analysis of several power traces and the DAA method is not resistant to the DPA attacks. Randomization techniques are recommended to defeat DPA [131]. Therefore, one possible future direction is to add resistance to DPA attacks in the presented architectures in this work.

- Several new forms of EC curves have been proposed such as Edwards, twisted Edwards, Montgomery, etc. These curves require fewer number of underlying arithmetic operations and are considered more secure against different side-channel attacks as compared to the curves presented in the other Weierstrass form. Thus, point multiplication operations over these curves are faster than the Weierstrass form. As the presented modular multipliers work for any general prime number, therefore, a possible future extension is to design an EC scalar multiplier to perform point multiplication operations over these new forms of EC curves.

- Modern FPGAs are equipped with different high speed components such as digital signal processing (DSP) blocks, block RAMs (BRAM) depending on vendors and device type, a DSP block integrates an integer multiplier of different sizes. For example, the Virtex-6 DSP block incorporates a 25×18-bit multiplier. A possible future extension of the work is to use these built-in FPGA components in the design of point multiplier over standard and the new forms of ECs.

Appendix A

Appendix

A.1 Implementation results of EC scalar multiplier using modular multipliers presented in Chapter 4

Table A.1: Number of clock cycles required for EC scalar multiplication in projective coordinates

Latency (T_n)	EC scalar Multiplier
R4BIM ($\lfloor n/2 \rfloor + 3$)	
DA[a]	$[n(2n+16)+\lfloor n/2 \rfloor(3n+16)]+4n$
DA[b]	$[n(2n+13)+\lfloor n/2 \rfloor(3n/2+14)]+4n$
DAA[b]	$n(3n+35)+4n$
R8BIM ($\lfloor n/3 \rfloor + 5$)	
DA[a]	$[n(4n/3+29)+\lfloor n/2 \rfloor(2n+33)]+4n$
DA[b]	$[n(4n/3+23)+\lfloor n/2 \rfloor(5n/3+27)]+4n$
DAA[b]	$n(2n+38)+4n$
R8PIM ($\lfloor n/3 \rfloor + 7$)	
DA[a]	$[n(4n/3+37)+\lfloor n/2 \rfloor(2n+41)]+4n$
DA[b]	$n(4n+31)+\lfloor n/2 \rfloor(5n/3+37)+4n$
DAA[b]	$n(2n+55)+4n$
R4BPIM ($\lfloor n/2 \rfloor + 3$)	
DA[a]	$[n(2n+16)+\lfloor n/2 \rfloor(3n+16)]+4n$
DA[b]	$[n(2n+13)+\lfloor n/2 \rfloor(3n/2+14)]+4n$
DAA[b]	$n(3n+24)+4n$
R8BPIM ($\lfloor n/3 \rfloor + 5$)	
DA[a]	$[n(4n/3+29)+\lfloor n/2 \rfloor(2n+33)]+4n$
DA[b]	$n(4n/3+25)+\lfloor n/2 \rfloor(5n/3+27)+4n$
DAA[b]	$n(2n+40)+4n$

* Double-and-Add (DA)
* Double-and-always-add (DAA)
* Three parallel multipliers (a)
* Four parallel multipliers (b)

Table A.2: Cycle count of EC scalar multiplication using DA algorithm and three multipliers in projective coordinates

Field size	Point doubling	Point addition	EC scalar multiplication
R4BIM			
192-any	400	592	133, 632
224-any	464	688	180, 992
256-any	528	784	235, 520
R8BIM			
192-any	285	417	94, 752
224-any	328	481	127, 344
256-any	371	545	164, 736
R8PIM			
192-any	293	425	97, 056
224-any	336	489	130, 032
256-any	371	553	167, 808
R4BPIM			
192-any	400	592	133, 632
224-any	464	688	180, 992
256-any	528	784	235, 520
R8BPIM			
192-any	285	417	94, 752
224-any	328	481	127, 344
256-any	371	545	164, 736

EC scalar multiplication (ECSM)

Table A.3: Cycle count of EC scalar multiplication using DA algorithm and four multipliers in projective coordinates

Field size	Point doubling	Point addition	EC scalar multiplication
R4BIM			
192-any	397	302	105, 216
224-any	461	350	142, 464
256-any	525	398	185, 344
R8BIM			
192-any	279	347	86, 880
224-any	322	401	117, 040
256-any	365	454	151, 552
R8PIM			
192-any	287	357	89, 376
224-any	330	411	119, 952
256-any	373	464	154, 880
R4BPIM			
192-any	397	302	105, 216
224-any	461	350	142, 464
256-any	525	398	185, 344
R8BPIM			
192-any	279	347	86, 880
224-any	322	401	117, 040
256-any	365	454	151, 552

EC scalar multiplication (ECSM)

Table A.4: Implementation of DAA algorithm using four multipliers in projective coordinates

Field size	Area	Freq. (MHz)	cycle count	Time (*ms*)	TP
R4BIM					
192-any	7,853 slices	77	115, 712	1.51	662
224-any	8,718 slices	73	156, 352	2.14	467
256-any	9,505 slices	68	203, 264	2.98	335
R8BIM					
192-any	8,569 slices	63	84, 068, 888	1.29	775
224-any	9,426 slices	58	109, 312	1.88	531
256-any	10,585 slices	53	141, 312	2.66	375
R8PIM					
192-any	15,565 slices	118	84, 672	0.71	1408
224-any	17,454 slices	112	113, 120	1	1000
256-any	19,361 slices	107	145, 664	1.36	735
R4BPIM					
192-any	8,381 slices	131	115, 712	0.88	1136
224-any	8,914 slices	127	156, 352	1.23	813
256-any	10,131 slices	122	203, 264	1.66	602
R8BPIM					
192-any	13,781 slices	115	81, 792	0.71	1408
224-any	16,234 slices	110	109, 760	0.99	1000
256-any	17,793 slices	105	141, 824	1.35	735

Throughput (TP), operations per second (*ops*)

Bibliography

[1] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology–CRYPTO85 Proceedings*. Springer, 1986, pp. 417–426.

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

[3] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and Public-Key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[4] NIST, "Data encryption standard (DES), FIPS (46-3)," *National Institute of Standards and Technology*, vol. 25, no. 10, pp. 1–22, 1999.

[5] , "Advanced encryption standard (AES), FIPS (197)," *National Institute of Standards and Technology*, vol. 197, pp. 441–0311, 2001.

[6] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[7] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

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

[9] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC Press, 2014.

[10] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST special publication 800-57," *NIST Special Publication*, vol. 800, no. 57, pp. 1–142.

[11] ——, "Recommendation for key management-part 1: General (revised," in *NIST special publication*. Citeseer, 2006.

[12] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *Journal of cryptology*, vol. 14, no. 4, pp. 255–293, 2001.

[13] "keylength," www.keylength.com, accessed: 2015-09-20.

[14] ECRYPT, "ECRYPT II yearly report on algorithms and keysizes (ict-2007-216676)," European Network of Excellence in Cryptology II, revision 1, Tech. Rep., 2012.

[15] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005.

[16] D. Hankerson, S. Vanstone, and A. J. Menezes, *Guide to elliptic curve cryptography*. Springer, 2004.

[17] S. Ghosh, "Design and analysis of pairing based cryptographic hardware for prime fields," Ph.D. dissertation, Department of Computer Science and Engineering, Indian Institute OF Technology Kharagpur, 2011.

[18] B. Baldwin, "Hardware design of cryptographic accelerators," Ph.D. dissertation, Department of Electrical Engineering University College Cork, 2013.

[19] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p," *Mathematics of computation*, vol. 44, no. 170, pp. 483–494, 1985.

[20] R. Schroeppel, H. Orman, S. ÓMalley, and O. Spatscheck, *Fast key exchange with elliptic curve systems*. Springer, 1995.

[21] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[22] D. Stebila and J. Green, "Elliptic-curve algorithm integration in the secure shell transport layer," 2009.

[23] S. Blake-Wilson, B. Moeller, V. Gupta, C. Hawk, and N. Bolyard, "Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS)," 2006.

[24] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[25] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 157–175.

[26] Certicom, "Certicom research. Standards for efficient cryptography 1: Elliptic curve cryptography. Standard SEC1," 2009, http://www.secg.org/SEC1-Ver-1.0.pdf.

[27] E. B. Barker, D. Johnson, and M. E. Smid, "Sp 800-56A. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised)," 2007, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.415.8236&rep=rep1&type=pdf.

[28] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Advances in cryptology–ASIACRYPT 2007*. Springer, 2007, pp. 29–50.

[29] "Explixit-Formaula-database." [Online]. Available: https://www.hyperelliptic.org/EFD/

[30] E. Brier and M. Joye, "Weierstraß elliptic curves and side-channel attacks," in *International Workshop on Public Key Cryptography*. Springer, 2002, pp. 335–345.

[31] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology–CRYPTO96*. Springer, 1996, pp. 104–113.

[32] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.

[33] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, "State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures," in *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 76–87.

[34] J. Fan and I. Verbauwhede, "An updated survey on secure ECC implementations: Attacks, countermeasures and cost," in *Cryptography and Security: From Theory to Applications*. Springer, 2012, pp. 265–282.

[35] J.-P. Deschamps, *Hardware implementation of finite-field arithmetic*. McGraw-Hill, Inc., 2009.

[36] E. Wenger and M. Hutter, "Exploring the design space of prime field vs. binary field ECC-hardware implementations," in *Information Security Technology for Applications*. Springer, 2011, pp. 256–271.

[37] M. N. Hassan and M. Benaissa, "Efficient time-area scalable ECC processor using μ-coding technique," in *Arithmetic of Finite Fields*. Springer, 2010, pp. 250–268.

[38] M. Benaissa and W. M. Lim, "Design of flexible $GF(2^m)$ elliptic curve cryptography processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 6, pp. 659–662, 2006.

[39] G. D. Sutter, J. Deschamps, and J. L. Imaña, "Efficient elliptic curve point multiplication using digit-serial binary field operations," *Industrial Electronics, IEEE Transactions on*, vol. 60, no. 1, pp. 217–225, 2013.

[40] S. Antao, R. Chaves, and L. Sousa, "Efficient FPGA elliptic curve cryptographic processor over $GF(2^m)$," in *ICECE Technology, 2008. FPT 2008. International Conference on*. IEEE, 2008, pp. 357–360.

[41] B. Ansari and H. Wu, "Efficient finite field processor for $GF(2^m)$ and its vlsi implementation," in *Information Technology, 2007. ITNG'07. Fourth International Conference on*. IEEE, 2007, pp. 1021–1026.

[42] Y. Wang and R. Li, "A unified architecture for supporting operations of AES and ECC," in *Parallel Architectures, Algorithms and Programming (PAAP), 2011 Fourth International Symposium on*. IEEE, 2011, pp. 185–189.

[43] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 8, pp. 1453–1466, Aug 2012.

[44] G. Meurice de Dormale and J.-J. Quisquater, "High-speed hardware implementations of elliptic curve cryptography: A survey," *Journal of systems architecture*, vol. 53, no. 2, pp. 72–84, 2007.

[45] J. Guajardo, T. Güneysu, S. S. Kumar, C. Paar, and J. Pelzl, "Efficient hardware implementation of finite fields with applications to cryptography," *Acta Applicandae Mathematica*, vol. 93, no. 1-3, pp. 75–118, 2006.

[46] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over NIST prime fields," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 8, pp. 1099–1112, Aug 2009.

[47] H. Alrimeih and D. Rakhmatov, "Fast and flexible hardware support for ECC over multiple standard prime fields," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 12, pp. 2661–2674, Dec 2014.

[48] T. Güneysu and C. Paar, "Ultra high performance ecc over NIST primes on commercial FPGAs," in *Cryptographic Hardware and Embedded Systems–CHES 2008*. Springer, 2008, pp. 62–78.

[49] A. P. Fournaris, N. Klaoudatos, N. Sklavos, and C. Koulamas, "Fault and power analysis attack resistant RNS based Edwards curve point multiplication," in *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, ser. CS2 '15. New York, NY, USA: ACM, 2015, pp. 43:43–43:46. [Online]. Available: http://doi.acm.org/10.1145/2694805.2694814

[50] M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi, "Efficient RNS implementation of elliptic curve point multiplication over GF(p)," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 8, pp. 1545–1549, Aug 2013.

[51] 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*, vol. 24, no. 1, pp. 151–164, Jan 2016.

[52] M. Varchola, T. Guneysu, and O. Mischke, "MicroECC: A lightweight reconfigurable elliptic curve crypto-processor," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, Nov 2011, pp. 204–210.

[53] D. J. Bernstein and T. Lange, "Failures in NIST ECC standards," 2015, https://cr.yp.to/newelliptic/nistecc-20160106.pdf.

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

[55] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Petrel: Power and timing attack resistant elliptic curve scalar multiplier based on programmable GF(p) arithmetic unit," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, no. 8, pp. 1798–1812, Aug 2011.

[56] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta, "Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks," *Computers & Electrical Engineering*, vol. 35, no. 2, pp. 329–338, 2009.

[57] G. Chen, G. Bai, and H. Chen, "A high-performance elliptic curve cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 54, no. 5, pp. 412–416, May 2007.

[58] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.

[59] M. Joye and S.-M. Yen, "The Montgomery powering ladder," in *Cryptographic Hardware and Embedded Systems-CHES 2002*. Springer, 2003, pp. 291–302.

[60] C. J. Mcivor, M. Mcloone, and J. V. Mccanny, "Hardware elliptic curve cryptographic processor over GF(p)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 9, pp. 1946–1957, Sept 2006.

[61] J.-H. Chen, M.-D. Shieh, and W.-C. Lin, "A high-performance unified-field reconfigurable cryptographic processor," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 8, pp. 1145–1158, Aug 2010.

[62] S.-C. Chung, J.-W. Lee, H.-C. Chang, and C.-Y. Lee, "A high-performance elliptic curve cryptographic processor over GF(p) with SPA resistance," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 1456–1459.

[63] J.-Y. Lai and C.-T. Huang, "Elixir: High-throughput cost-effective dual-field processors and the design framework for elliptic curve cryptography," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 11, pp. 1567–1580, Nov 2008.

[64] S. Ors, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of an elliptic curve processor over GF(p)," in *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*, June 2003, pp. 433–443.

[65] W. Shuhua and Z. Yuefei, "A timing-and-area tradeoff GF(p) elliptic curve processor architecture for FPGA," in *Communications, Circuits and Systems, 2005. Proceedings. 2005 International Conference on*, vol. 2, May 2005, pp. –1312.

[66] J. Fan, K. Sakiyama, and I. Verbauwhede, "Elliptic curve cryptography on embedded multicore systems," *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 231–242, 2008.

[67] E. Öztürk, B. Sunar, and E. Savaş, "Low-power elliptic curve cryptography using scaled modular arithmetic," in *Cryptographic Hardware and Embedded Systems–CHES 2004*. Springer, 2004, pp. 92–106.

[68] G. Orlando and C. Paar, "A scalable GF(p) elliptic curve processor architecture for programmable hardware," in *Cryptographic Hardware and Embedded Systems–CHES 2001*. Springer, 2001, pp. 348–363.

[69] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "An FPGA implementation of a GF(p) ALU for encryption processors," *Microprocessors and Microsystems*, vol. 28, no. 5, pp. 253 – 260, 2004.

[70] B. Baldwin, R. Moloney, A. Byrne, G. McGuire, and W. P. Marnane, "A hardware analysis of twisted Edwards curves for an elliptic curve cryptosystem," in *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2009, pp. 355–361.

[71] 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*, vol. 2, no. 4, pp. 221–240, 2012.

[72] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over Fp," in *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010, pp. 48–64.

[73] H. Marzouqi, M. Al-Qutayri, and K. Salah, "Review of elliptic curve cryptography processor designs," *Microprocessors and Microsystems*, vol. 39, no. 2, pp. 97 – 112, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933115000137

[74] D. J. Bernstein and T. Lange, "Safecurves: choosing safe curves for elliptic-curve cryptography," *Disponıvel em http://safecurves. cr. yp. to/rigid. html*, 2013.

[75] S. Brown and J. Rose, "FPGA and CPLD architectures: A tutorial," *IEEE Design & Test of Computers*, no. 2, pp. 42–57, 1996.

[76] "Xilinx," www.xilinx.com, accessed: 2015-09-20.

[77] "Altera," https://www.altera.com/, accessed: 2015-09-10.

[78] Xilinx, *Virtex-6 family overview, (DS150)*, August 2015.

[79] Xilinx, *Virtex-6 FPGA Configurable Logic Block, (UG364)*, Feburary 2012.

[80] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb 2007.

[81] F. Rodríguez-Henríquez, N. A. Saqib, A. D. Pérez, and C. K. Koc, *Cryptographic algorithms on reconfigurable hardware*. Springer Science & Business Media, 2007.

[82] P. S. Zuchowski, C. B. Reynolds, R. J. Grupp, S. G. Davis, B. Cremen, and B. Troxel, "A hybrid ASIC and FPGA architecture," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*. ACM, 2002, pp. 187–194.

[83] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 4, pp. 545–557, Aug 2001.

[84] F. Rodríguez-Henríquez, N. A. Saqib, A. D. Perez, and C. K. Koc, *Cryptographic algorithms on reconfigurable hardware*. Springer Science & Business Media, 2007.

[85] P. FIPS, "186-2. digital signature standard (dss)," *National Institute of Standards and Technology (NIST)*, 2000.

[86] G. Blakely, "A computer algorithm for calculating the product AB modulo M," *Computers, IEEE Transactions on*, vol. C-32, no. 5, pp. 497–500, May 1983.

[87] K. R. Sloan Jr, "Comments on a computer algorithm for calculating the product AB modulo M," *IEEE Transactions on Computers*, vol. 34, no. 3, pp. 290–292, 1985.

[88] H. Alrimeih and D. Rakhmatov, "Pipelined modular multiplier supporting multiple standard prime fields," in *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, June 2014, pp. 48–56.

[89] K. Kelley and D. Harris, "Very high radix scalable Montgomery multipliers," in *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*, July 2005, pp. 400–404.

[90] A. Tenca and L. Tawalbeh, "An efficient and scalable radix-4 modular multiplier design using recoding techniques," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, vol. 2, Nov 2003, pp. 1445–1450 Vol.2.

[91] Ç. K. Koç, T. Acar, and B. S. Kaliski Jr, "Analyzing and comparing Montgomery multiplication algorithms," *Micro, IEEE*, vol. 16, no. 3, pp. 26–33, Jun 1996.

[92] D. Narh Amanor, C. Paar, J. Pelzl, V. Bunimov, and M. Schimmler, "Efficient hardware architectures for modular multiplication on FPGAs," in *Field Programmable Logic and Applications, 2005. International Conference on*, Aug 2005, pp. 539–542.

[93] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods," *Computers, IEEE Transactions on*, vol. 59, no. 12, pp. 1715–1721, Dec 2010.

[94] V. Bunimov and M. Schimmler, "Area and time efficient modular multiplication of large integers," in *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*. IEEE, 2003, pp. 400–409.

[95] K. Shigemoto, K. Kawakami, and K. Nakano, "Accelerating Montgomery modulo multiplication for redundant radix-64k number system on the FPGA using dual-port block rams," in *Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on*, vol. 1. IEEE, 2008, pp. 44–51.

[96] K. Javeed and X. Wang, "Efficient Montgomery multiplier for pairing and elliptic curve based cryptography," in *Communication Systems, Networks & Digital Signal Processing (CSNDSP), 2014 9th International Symposium on*. IEEE, 2014, pp. 255–260.

[97] A. Mondal, S. Ghosh, A. Das, and D. R. Chowdhury, "Efficient FPGA implementation of Montgomery multiplier using DSP blocks," in *Progress in VLSI Design and Test*. Springer, 2012, pp. 370–372.

[98] K. Javeed and X. Wang, "Radix-4 and radix-8 Booth encoded interleaved modular multipliers over general Fp," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–6.

[99] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Secure dual-core cryptoprocessor for pairings over Barreto-Naehrig curves on FPGA platform," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 3, pp. 434–442, March 2013.

[100] S. Ghosh, D. Mukhopadhyay, and D. Chowdhury, "High speed Fp multipliers and adders on FPGA platform," in *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*, Oct 2010, pp. 21–26.

[101] K. Javeed, X. Wang, and M. Scott, "Serial and parallel interleaved modular multipliers on FPGA platform," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, Sept 2015, pp. 1–4.

[102] K. Javeed and X. Wang, "Speed and area optimized parallel higher-radix modular multipliers," Cryptology ePrint Archive, Report 2016/053, 2016, http://eprint.iacr.org/.

[103] S. Ghosh, M. Alam, I. Gupta, and D. Chowdhury, "A robust GF(p) parallel arithmetic unit for public key cryptography," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, Aug 2007, pp. 109–115.

[104] A. M. AbdelFattah, A. M. B. El-Din, and H. M. Fahmy, "An efficient architecture for interleaved modular multiplication."

[105] K. Javeed, D. Irwin, and X. Wang, "Design and performance comparison of modular multipliers implemented on FPGA platform," in *ICCCS*. Springer, 2016, p. in press.

[106] B. S. Kaliski, "The Montgomery inverse and its applications," *IEEE transactions on computers*, no. 8, pp. 1064–1065, 1995.

[107] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "Fast modular division for application in ECC on reconfigurable logic," in *Field Programmable Logic and Application*. Springer, 2003, pp. 786–795.

[108] A. D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[109] S. A. Khan, *Digital design of signal processing systems A paratical approach*. Wiley, 2011.

[110] R. P. Brent and P. Zimmermann, *Modern computer arithmetic*. Cambridge University Press, 2010, vol. 18.

[111] L. Hars, "Long modular multiplication for cryptographic applications," in *Cryptographic Hardware and Embedded Systems-CHES 2004*. Springer, 2004, pp. 45–61.

[112] "IEEE standard specifications for Public-Key cryptography - amendment 1: Additional techniques," *IEEE Std 1363a-2004 (Amendment to IEEE Std 1363-2000)*, pp. 1–167, Sept 2004.

[113] K. Lauter, "The advantages of elliptic curve cryptography for wireless security," *Wireless Communications, IEEE*, vol. 11, no. 1, pp. 62–67, Feb 2004.

[114] F.-X. Standaert, "Introduction to side-channel attacks," in *Secure Integrated Circuits and Systems*. Springer, 2010, pp. 27–42.

[115] H. Edwards, "A normal form for elliptic curves," *Bulletin of the American Mathematical Society*, vol. 44, no. 3, pp. 393–422, 2007.

[116] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves," in *Progress in Cryptology–AFRICACRYPT 2008*. Springer, 2008, pp. 389–405.

[117] R. Moloney, G. McGuire, and M. Markowitz, "Elliptic curves in Montgomery form with b=1 and their low order torsion," Cryptology ePrint Archive, Report 2009/213, 2009, http://eprint.iacr.org/.

[118] K. Okeya, H. Kurumatani, and K. Sakurai, "Elliptic curves with the Montgomery-form and their cryptographic applications," in *Public Key Cryptography*. Springer, 2000, pp. 238–257.

[119] T. Güneysu, "Utilizing hard cores of modern FPGA devices for high-performance cryptography," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 37–55, 2011.

[120] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[121] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault attacks on RSA with CRT: Concrete results and practical countermeasures," in *Cryptographic hardware and embedded systems–CHES 2002*. Springer, 2002, pp. 260–275.

[122] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible..." in *Advances in Cryptology–ASIACRYPT 2000*. Springer, 2000, pp. 489–502.

[123] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems–CHES 2002*. Springer, 2002, pp. 13–28.

[124] R. Bevan and E. Knudsen, "Ways to enhance differential power analysis," in *Information Security and Cryptology–ICISC 2002*. Springer, 2002, pp. 327–342.

[125] E. Brier, C. Clavier, and F. Olivier, "Optimal statistical power analysis." *IACR Cryptology ePrint Archive*, vol. 2003, p. 152, 2003.

[126] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of eliminating errors in cryptographic computations," *Journal of cryptology*, vol. 14, no. 2, pp. 101–119, 2001.

[127] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology–CRYPTO97*. Springer, 1997, pp. 513–525.

[128] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Cryptographic Hardware and Embedded Systems–CHES 2001*. Springer, 2001, pp. 251–261.

[129] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): Measures and counter-measures for smart cards," in *Smart Card Programming and Security*. Springer, 2001, pp. 200–210.

[130] P. Sasdrich and T. Güneysu, "Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices," in *Reconfigurable Computing: Architectures, Tools, and Applications*. Springer, 2014, pp. 25–36.

[131] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Cryptographic Hardware and Embedded Systems*. Springer, 1999, pp. 292–302.