High Speed and Area-Time Efficient Finite field GF(2^m) Point Multiplication Architectures for Elliptic Curve Cryptographic Applications

Submitted in partial fulfilment of the requirements

for the award of the degree of

Doctor of Philosophy

by

Pradeep Kumar Goud N

(Roll No: 717026)

Under the supervision of

Prof. B. Lakshmi

Department of Electronics & Communication Engineering

National Institute of Technology Warangal

Telangana, India - 506004

2024

Dedicated

To

My Mother,

Teachers & Friends

Approval Sheet

This thesis entitled **High Speed and Area-Time Efficient Finite Field GF(2^m) Point Multiplication Architectures for Elliptic Curve Cryptographic Applications** by **Pradeep Kumar Goud N** is approved for the degree of **Doctor of Philosophy**.

Examiners

Research Supervisor

Prof. B. Lakshmi
Department of ECE
NIT Warangal, India-506004

Chairman & Head

Prof . D. Vakula
Department of ECE
NIT Warangal, India-506004

Place:

Date:

Declaration

This is to certify that the work presented in this thesis entitled **High Speed and Area-Time Efficient Finite field GF(2^m) Point Multiplication Architectures for Elliptic Curve Cryptographic Applications** is a bonafied work done by me under the supervision of **Prof. B. Lakshmi** and was not submitted elsewhere for the award of any degree.

I declare that this written submission represents my own ideas and even considered others ideas which are adequately cited and further referenced the original sources. I understand that any violation of the above will cause disciplinary action by the institute and can also evoke panel action from the sources or from whom proper permission has not been taken when needed. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea or data or fact or source in my submission.

Place:

Date:

Pradeep Kumar Goud N
Research Scholar
Roll No.: 717026

NATIONAL INSTITUTE OF TECHNOLOGY

WARANGAL, INDIA-506004

Department of Electronics & Communication Engineering

CERTIFICATE

This is to certify that the thesis work entitled **High Speed and Area-Time Efficient Finite Field GF(2^m) Point Multiplication Architectures for Elliptic Curve Cryptographic Applications** is a bonafide record of work carried out by **Pradeep Kumar Goud N** submitted to the faculty of **Electronics & Communication Engineering** department, in partial fulfilment of the requirements for the award of the degree of **Doctor of Philosophy** in **Electronics and Communication Engineering, National Institute of Technology Warangal, India-506004**. The contributions embodied in this thesis have not been submitted to any other university or institute for the award of any degree.

Prof. B. Lakshmi
Research Supervisor
Department of ECE
NIT Warangal, India-506 004.

Place:

Date:

Acknowledgements

First, I take immense pleasure to convey my sincere gratitude to my supervisor Prof. B. Lakshmi for her perpetual encouragement and supervision. Her guidance has oriented me in a proper direction and supported me with promptness and care.

I take this privilege to thank all my Doctoral Scrutiny Committee members Prof . P. Sreehari Rao, Prof. D. Vakula, Prof. S. Anuradha, and Prof. R.B.V Subramanyam for their detailed review, constructive suggestions, and excellent advice during the progress of this research work. I would also like to thank all the faculty of Dept.of ECE who helped me during the course.

I would like to convey my sincere gratitute to Dr. Siva Rama Krishna, Dr. Sudha Ellison Mathe, Dr. Ashok Agarwal, Mr. Pavan kumar Reddy, Mr. Deepak Chinnapilla, Mr. Naresh kumar, Mr. Jayanth yadav, Mr. Atul Shriwastava, Mr. Partha Saradhi, Mr. Sunny Yadav, Mr. Santosh kumar Singh, Mr. Naresh Bashetty, Mr. Narsingh Bahadur, Mr. Sher Bahadur, Mr. Rajsekar Reddy, Mr. Gopi, Mr. Ravi kanth, Mr. Dattatreya, Mr. Kondal Reddy, Mr. koushik, and Mr. Ravi kumar for thier invaluable guidance and support. I would also like to extend my heartfelt appreciation to my family, colleague scholars, friends, and well-wishers who helped to write my thesis with their support. Finally, I thank my nation India, for giving me the opportunity to carry out my research work at the NIT Warangal. A special thanks to MHRD for its financial support.

Pradeep Kumar Goud N

Abstract

Advances in communication technology and availability of high bandwidth enables billions of devices communicate confidential information over the Internet. Hence, it is required to introduce some data security technique while transmitting the information which can be addressed using various cryptographic algorithms.

Cryptography is one of the profound technique used to secure the data by employing encryption and decryption algorithms. Over the last few years, technological advances in the implementation of smart sensors, processing elements and communication services in resource-constrained devices have enabled the rapid growth and emergence of evolving technologies like Wireless sensor-networks (WSNs) and Internet-of-Things (IoT). These technologies elevated the need for efficient and high-performance cryptographic algorithms. The hardware realization of these algorithms involve the selection of an appropriate architecture with constraints on area, speed and power. The conventional approaches such as Elgamal, and RSA are found to be impractical to implement on resource constrained devices. The Elliptic Curve-Cryptography (ECC) suggested by Koblitz and Miller has captivated considerable attention when compared to similar cryptosystems available in the literature owing to its high security per bit ratio and small key size.

The performance of ECC relies on point multiplication operation and its underlying finite field operations. Each point multiplication operation is realized by a series of finite-field addition, finite-field inversion and finite-field multiplication operations. FF-multiplication and FF-inversion are the two area and time critical operations in point multiplication. Hence, the efficiency of point multiplication relies on efficiency of these two finite field operations. In addition, the type of irreducible polynomial used also impacts the performance of point multiplication architecture. Many high performance point multiplication architectures for various classes of irreducible polynomials using polynomial

basis are proposed in the literature to achieve reduction in area and time complexities.

In this thesis, we focus on the design of area-time efficient hardware architectures for point multiplication targeting the implementation of security in ECC applications. Accordingly, some GF(2^m) point multiplication algorithms and formulations are proposed based on the available algorithms in the literature and subsequently efficient point multiplication architectures are realized for these proposed algorithms. We first studied the efficient implementation of FF-inversion and FF-multiplication operations. We proposed a FF-inversion architecture over GF(2^m) for general irreducible polynomials based on the Itoh-Tsujii algorithm. We then proposed an area-time efficient point multiplication architecture employing the proposed FF-inversion architecture. In addition, a digit-serial FF-multiplier and parallel Itoh-Tsujii algorithm for inversion are proposed to reduce the computation time. An area-time efficient and high speed point multiplication architectures are proposed by employing the proposed FF-multiplier and FF-inversion modules. A detailed analysis of the possible parallelization and scheduling schemes employed to reduce the latency of proposed point multiplication architectures is presented. The area and time complexities of all the proposed architectures are computed analytically for various m values and compared with similar architectures in the literature. It is observed from the comparison of the results that the proposed architectures outperform the existing architectures in terms of area-time complexities. These proposed area-time efficient GF(2^m) point multiplication architectures can be preferred in the implementation of security in ECC applications.

Contents

List of Figures

List of Tables

List of Abbreviations

IoT	Internet of Things
DES	Data Encryption Standard
AES	Advanced Encryption Standard
RSA	Rivest-Shamir-Adleman algorithm
ECDH	Elliptic Curve Diffie-Hellman key exchange algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
ECC	Elliptic Curve Cryptographgy
NIST	National Institute of Standards and Technology
MSB	Most Significant Bit
LSB	Least Significant Bit
MSD	Most Significant Digit
ITA	Itoh Tsujii Algorithm
EEA	Extended Euclidean algorithm
MUX	Multiplexer
ATP	Area-Time-Product
PA	Point Addition
PD	Point Doubling
PM	Point Multiplication
LUT	Look Up Table
SAR	Shift and Reduction Module
VLSI	Very Large Scale Integration
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
nm	nanometer
us	nanosecond

Chapter 1

Introduction

In today's era, wireless networks facilitate communication over billions of devices. The mass exchange of information over the open-ended Internet architecture sometimes leads to eavesdropping of confidential, private, and sensitive information. Hence, it is required to introduce some data security techniques while transmitting the information. Cryptography is one of the profound technique used to secure the data by employing encryption and decryption algorithms. Encryption is performed on the original message at the sender side using a secret key to obtain an unreadable cipher message that can be sent over a communication channel.The receiver employs the secret key to decrypt the received cipher message and retrieve the original plain-text message. Based on the key sharing strategy, the techniques in modern cryptography can be divided into symmetric-key-cryptography and asymmetric-key-cryptography. In symmetric-key cryptography, the same secret key is used for both encryption and decryption.The major challenge with this approach is the exchange of common key securely. This problem is addressed by the Diffie and Hellman in 1976, resulting in the development of public key cryptography algorithm known as RSA.

The development of small, always-connected devices in recent years, including wireless sensor nodes (WSNs), smart cards, mobile-hand-held devices, near-field communication (NFC) devices, and RFID-tags has necessitated the development of efficient, high-performance cryptographic computation schemes.The conventional schemes such as RSA is found to be impractical to implement in resource constrained devices.This led to the adoption of a new technology based on elliptic curves for implementing public key cryp-

tography, known as elliptic curve cryptography (ECC), proposed by Neil Koblitz [2] and Victor Miller, and it received significant attention in the most recent research works published in the literature. IEEE and NIST standards propose using ECC as an effective way to implement public key cryptography in resource-constrained and embedded environments. It is observed from the analysis of ECC that it requires less key size while providing the same level of security as that offered by RSA technique. The reduced key size necessitates less hardware, making it particularly suitable for devices with limited resources, like storage, silicon-area, and bandwidth. The inherent difficulty of solving the elliptic curve discrete logarithm problem further enhances the security provided by ECC [2].

Elliptic curve cryptography utilizes elliptic curves defined over prime-fields (GF(p)) or binary-fields (GF(2^m)). When developing applications on general-purpose processors, the software implementation of security using ECC over prime-fields GF(p) is often preferred. However, the resource-constrained applications always target to reduce hardware and power while consuming less resources to perform any task. Hence, even for implementing security in these devices also, resources can be reduced by realizing ECC algorithm over binary-fields.

ECC over GF(2^m) heavily uses GF(2^m) arithmetic in its low-level operations to realize the other high-level operations such as point multiplication, point doubling, and point addition. The hierarchy of the arithmetic operations involved in ECC-based schemes is shown in Fig. 1.1. The efficiency of these schemes relies heavily on the implementation of the low-level arithmetic operations, especially GF(2^m) multiplication [3] and GF(2^m) inversion. Thus, the performance of applications implementing security using ECC can be improved by realizing finite field multiplication and inversion over GF(2^m) employing efficient architectures.

A few efficient architectures have been proposed in the literature to implement GF(2^m) multiplication and GF(2^m) inversion operations in hardware, aiming to reduce both area and computation time.

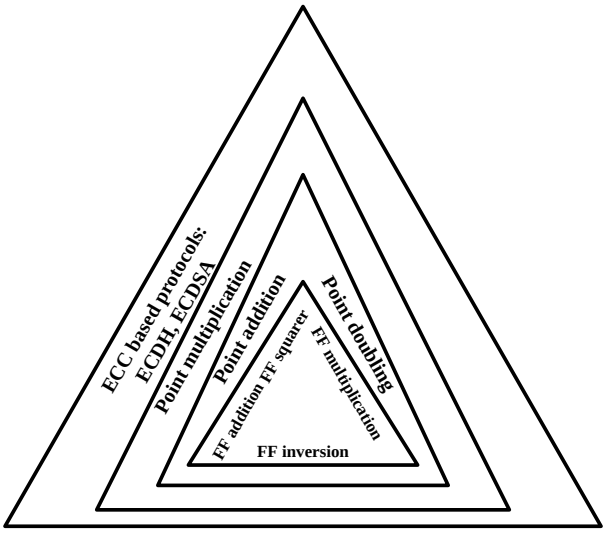

Figure 1.1 Implementation of ECC Hierarchy

1.1 Motivation

In recent times, technological advancements have facilitated extensive utilization of resource-constrained devices and high-performance web servers across various applications. Securing these resource-constrained devices, including RFID-tags, hand-held devices, smart-cards, and high-performance servers involved in secure online banking, and e-commerce transactions demands efficient implementation of Elliptic Curve Cryptography (ECC). The former applications face challenges due to limited silicon area availability, while the latter ones encounter issues related to the relatively slow speed of existing security protocols. Additionally, the rise in the number of always-connect and limited resource devices to the servers, necessitates efficient cryptographic computation protocols.

Prime fields and binary extension fields can be used to model elliptic curves over finite fields and the literature has numerous ECC implementations using both the fields. However, the usage of particular field is chosen based on the application and resource availability. Binary fields and prime fields are understood to outperform over each other in hardware and software implementations, respectively [4]. It is observed that the recently presented methods in the literature did not take into consideration a systematic implementation of ECC over binary fields. For example, they used finite field multipliers with large digit sizes without evaluating their suitability for the implementation of proposed crypto-processors [5, 6]. The hierarchy of ECC implementation necessitates an

efficient implementation of lower level finite field arithmetic, followed by curve level and protocol level operations. Hence, it becomes one crucial task to explore a bottom-up approach when designing an ECC crypto-processor for specific applications. In addition, parallelism is found to be only method to improve the performance of point multiplication at curve level hierarchy. However, one should note that there is a data dependency between curve level operations and these data dependencies vary with the type of ECC curve being used for point multiplication. Hence, parallelism is not applicable to all the ECC curves and this limits the performance of the ECC designs according to the number of parallel processors. In this research, various architectures proposed in the literature to realize point multiplication are analyzed to propose area-time efficient architectures by attempting to reduce area-time complexities of the primitive building blocks such as FF-inversion and FF parallel multiplication. In this work, we have also attempted to exploit the data dependency between curve level operations to improve the computation time of point multiplication.

1.2 Research Objectives

The objective of this research is to design and implement efficient polynomial basis finite field GF(2^m) point multiplication architectures to enhance the performance of ECC algorithms.

- Due to the wide range of application of ECC algorithms in resource constrained devices, it is required to design area-efficient finite-field inversion and finite-field multiplication architectures using general irreducible polynomials. It is also required to verify the performance of these finite field inversion architectures using analytical and FPGA implementation comparisons.
- Many ECC applications communicating over open ended Internet architecture are prone to various types of attacks, namely, power and time analysis attacks. Hence, it is required to develop efficient point multiplication algorithms.
- In addition, many ECC applications need optimization in terms of both hardware and speed. Scalable architectures such as digit-serial architectures are suitable for

implementing ECC applications. Hence, it is required to design efficient digit-serial multipliers and point multiplication architectures employing digit-serial multipliers. It is also required to verify the performance of these multipliers through analytical and FPGA implementation comparisons.

1.3 Thesis Contributions

The contributions of the thesis are summarized as follows:

- **Area-Efficient Finite Field GF(2^m) Inversion Architecture for General Irreducible Polynomials** A finite field inversion architecture using polynomial basis that performs inversion of any finite field element for any irreducible polynomial is proposed. The performance of this proposed architecture is evaluated through theoretical analysis and FPGA implementations. The key contributions of this study are briefly described as follows:
  - **Proposed Area-Efficient Architecture for Finite Field Inversion over GF(2^m) using Polynomial Basis** In this work, a modified Itoh-Tsujii algorithm over general irreducible polynomials is proposed and the corresponding architecture to realize the finite filed inversion algorithm is presented. The high speed and area efficient 4^k exponentiation modules are employed to achieve reduction in hardware complexities. The proposed architecture achieves reduction in area around 15% to 70% and reduction in ATP (area-time-product) around 5% to 15% compared to the existing architectures for the field orders $m = 193$ and $m = 409$, respectively.
- **Low area-time complexity point multiplication architecture for ECC over GF(2^m) using polynomial basis** A point multiplication architecture employing a digit-serial multiplier is proposed over GF(2^m) using polynomial basis. The performance of the proposed architecture is evaluated through theoretical analysis and FPGA implementations. The key contributions of this study are briefly described as follows:

  - **Proposed Low Area-time complexity Point Multiplication Architecture using Polynomial Basis** In this work, a modified Montgomery algorithm over general irreducible polynomials is presented and the corresponding architecture to realize the point multiplication algorithm is presented. A digit-serial multiplier is used to realize FF-multiplication operation and a modified Itoh-Tsujii algorithm is employed to realize FF-inversion. The proposed point multiplication architecture achieves reduction in ATP(area-time-product) around 4% to 85% and 35% to 80% compared to the existing architectures for the field of orders $m = 163$ and $m = 233$, respectively.

- **High Speed and Area-Time Efficient Point Multiplication Architectures over GF(2^m) using Polynomial Basis** In this work we present two point multiplication architectures over GF(2^m) using general irreducible polynomials and irreducible trinomials. The performance of these proposed architectures is evaluated through theoretical analysis and FPGA implementations. The key contributions of this study are briefly described as follows:

  - **Proposed Area-Time Efficient Point Multiplication Architecture for ECC over GF(2^m) using Polynomial Basis** In this work, a modified Montgomery ladder algorithm for point multiplication is proposed and the corresponding architecture to realize point multiplication is presented. A modified Itoh-Tsujii algorithm is employed to achieve reduction in hardware complexities for the realization of FF-inversion. The proposed point multiplication architecture achieves reduction in ATP(area-time-product) around 15% to 80% and 40% to 95% compared to the existing architectures for the field of orders $m = 163$ and $m = 233$, respectively.
  - **Proposed High Speed Point Multiplication Architecture over GF(2^m) for the case of Irreducible Trinomials** In this work, a modified Montgomery ladder algorithm for point multiplication is proposed to reduce the computation time of point multiplication and the corresponding architecture to realize point multiplication is presented. A parallel Itoh-Tsujii algorithm is developed to achieve reduction in the computation time for the realization of FF-inversion. The proposed point multiplication architecture achieves reduc-

tion in computation time around 10% to 95% and 3% to 90% compared to the existing works for the field orders $m = 233$and $m = 409$, respectively.

1.4 Thesis Organization

The rest of the thesis is structured as follows:

Chapter 2 presents an overview of the mathematical concepts of GF(2^m) finite fields and point multiplication operation.

Chapter 3 presents the review of the architectures proposed in the literature for ECC point multiplication. It presents the available finite field inversion architectures for general irreducible polynomials followed by the available finite field multiplier architectures for general irreducible polynomials. This chapter also presents the review of available point multiplication architectures. The review also includes detailed discussions on the performance of these architectures in terms of area complexity, latency, and critical path delay.

Chapter 4 presents a modified finite filed inversion algorithm and its corresponding finite field inversion architecture over GF(2^m) for general irreducible polynomials. This chapter also presents the formulations derived for 4^k exponentiation over irreducible trinomials. Analysis and FPGA implementations results followed by a comparison of results with existing works are presented.

Chapter 5 presents the design of digit-serial multiplier and point multiplication architectures over GF(2^m) using two specific classes of trinomials and pentanomials. Analysis and FPGA implementations results followed by a comparison of results with existing works is also presented.

Chapter 6 presents the design of two point multiplication architectures over GF(2^m) using polynomial basis. Analysis and FPGA implementations results followed by a comparison of results with existing works is also presented.

Chapter 7 draws conclusions from the earlier chapters and concludes the thesis.

1.5 Conclusions

In this chapter, a brief overview of the entire research work along with the motivation behind this research and its objectives are presented. The next chapter presents an overview of the mathematical concepts of GF(2^m) finite fields and point multiplication operation along with a few available point multiplication algorithms.

Chapter 2

Mathematical Background

This chapter presents a brief summary of some mathematical theory of finite fields. First, we present the definitions and properties of Groups, Rings, and Fields. Following this, we present the definitions of finite field, binary finite-field GF(2^m), and GF(2^m) arithmetic operations. Finally, elliptic curve cryptography and its underlying operations are presented along with a few point multiplication algorithms available in the literature.

2.1 Finite Field Arithmetic

2.1.1 Groups

Definition 1. A Group is defined as a set G combined with a binary operator $*$, where for any elements a and b in G, the result of the group operation between a and b must be in G i.e. $a * b \in$ G, and it fulfills the subsequent properties:

(1) Identity - In the set G, there exists an element $e \in$ G, such that the operation $a * e = e * a = a$ holds true for every $a \in$ G.

(2) Inverse - For each element a in G, there exists an element a' in G such that $a * a' = a' * a = e$, where e represents the identity element.

(3) Associativity - For every $a, b, c \in$ G, the following identity holds: $a * (b * c) = (a * b) * c$.

Examples:

(1) The set of integers mod n, $\mathbb{Z}_n$, under addition is a group $(\mathbb{Z}_n, +)$, where the group operator + describes the addition modulo n operation. This group satisfies the axioms as explained below,

(a) Closure: For any given two integers mod n, their sum, defined as addition modulo n, is also an integer mod n.

(b) Identity: $0 \bmod n$ is the identity of the group, since for any $a \in (\mathbb{Z}_n, +)$, it follows that $0+a = (0+a) \bmod n = a \bmod n$ as well as $a+0 = (a+0) \bmod n = a \bmod n$.

(c) Inverse: For any given $a \bmod n$, we can find an inverse a' in the group such that $a + a' = e$, i.e. $a + a' \equiv 0 \bmod n$. The inverse of a (a') in this case is $n - a$.

(d) Associativity: From the basic rules of addition associativity of the integers hold true, hence, the integers $\bmod n$ are also associative. That is, since $a+(b+c) = (a+b)+c$, it is also true that $a + (b + c) \equiv (a + b) + c \bmod n$.

A group (G, $*$) is considered "abelian" if for every element a and b in G, it satisfies $a * b = b * a$.

2.1.2 Rings

A Ring, denoted as (R, +, $\times$), is a set of elements equipped with two operations, + and $\times$, and it fulfills the subsequent properties:

(1) The group (R, +) must be an abelian group.

(2) The operation $\times$ adheres to the associative law, i.e., $(a \times b) \times c = a \times (b \times c)$, for every $a, b, c \in$ R.

(3) The operation $\times$ follows the distributive law over the + operation, i.e., for every $a, b, c \in$ R, the ensuing equations are valid: $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = b \times a + c \times a$.

Examples:

(1) The set of integers modulo n, denoted as $\mathbb{Z}_n$, with the multiplication and addition modulo n operations forms a ring $(\mathbb{Z}_n, +, \times)$.

(2) Another example is the set of integers $\mathbb{Z}$ with the usual multiplication and addition operations, forming a ring $(\mathbb{Z}, +, \times)$.

A ring is termed a "commutative ring" if the operation $\times$ follows the commutative law, i.e., $a \times b = b \times a$. The above two examples are commutative rings.

2.1.3 Fields

Definition 3. A field denoted as (F, $+, \times$) is a set F which is closed under two operations $\times$ and $+$, such that

(1) (F, $+$) is an abelian group and

(2) F-$\{0\}$ is an abelian group under $\times$.

Examples of fields include the set of all real numbers $\mathbb{R}$, the set of all complex numbers $\mathbb{C}$, and the set of all rational numbers. $\mathbb{Q}$.

2.1.4 Finite Fields

Finite-fields are named as Galois-fields honoring Evariste Galois a French mathematician. A Galois field is a field that consists of a finite number of elements and can be of two types: a prime-field GF(p) and a binary-field GF(2^m).

1. Prime Fields, GF(p): The order of this field is 'p', which must be a prime number.

Ex: GF(2), GF(5), and GF(29).

The GF(2) field is the smallest finite field containing two elements. It can be written as GF(2) = $\{0, 1\}$. In this field, 1 is the multiplicative identity and 0 is the additive identity. Arithmetic operations in this field follow modulo 2 arithmetic:

GF(2) addition: $0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0$, and

GF(2) multiplication: $0 \times 0 = 0, 0 \times 1 = 0, 1 \times 0 = 0, 1 \times 1 = 1$.

It can be observed that GF(2) addition is the same as logical XOR operation, and GF(2) multiplication is the same as logical AND operation. Hence, these operations can be

implemented using XOR and AND gates.

2. Extension Fields, GF(p^m): The order of this field is 'p^m', where 'p' must be a prime number and m is any positive integer greater than 1.

Ex: GF(2^{409}), GF(5^7), and GF(29^4).

2.1.5 Binary Finite Fields, GF(2^m)

Binary finite fields are the fields of the form GF(2^m). These fields can be obtained from the extension fields GF(p^m) by selecting $p = 2$. In other words, these are the extension fields of the prime field GF(2).

Ex: GF(2^8), GF(2^{169}), GF(2^{233}), and GF(2^{409}).

Hence, GF(2^m) represents a binary extension finite field, generated over the base field GF(2), where GF(2) is a finite field comprising the elements 0 and 1. The elements within the binary finite field GF(2^m) can be expressed using polynomials of degree less than m over GF(2), implying that the coefficients of these polynomials originate from the base field GF(2). Consider $A(x)$ as an arbitrary element of the field GF(2^m) that can be represented as a polynomial of degree $(m-1)$ as follows:

$$A(x) = \sum_{j=0}^{m-1} a_j x^j = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1 x + a_0 \tag{2.1}$$

where all $a_j \in$ GF(2). This element can also be represented using the coordinate notation as $(a_{m-1}, a_{m-2}, ..., .a_1, a_0)$.

Furthermore, each GF(2^m) possesses a distinguishing m^{th} degree polynomial known as an irreducible polynomial. An irreducible polynomial in the finite field GF(2^m) is a monic polynomial of degree m that cannot be factored into two non-trivial polynomial elements over the same field. The general form of the irreducible polynomial $R(x)$ for GF(2^m) takes the shape of a monic polynomial given by $R(x) = x^m + \sum_{j=1}^{m-1} r_j x^j + 1$, where at least one of the coefficients r_j is non-zero, and all r_j belong to GF(2).

Therefore, the finite field GF(2^m) can be represented as a set of all its 2^m polynomial elements, denoted by:

$$\mathrm{GF}(2^m) = A(x)|A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + a_{m-3}x^{m-3} + \ldots + a_1x + a_0;\ \text{where } a_i \in \mathrm{GF}(2),\ \text{for} \tag{2.2}$$

Here, x represents a root of the irreducible polynomial $R(x)$.

The field irreducible polynomials $R(x)$ fall into different categories, including general irreducible polynomials, equally spaced polynomials, all-one polynomials, pentanomials,and trinomials. General irreducible polynomials have no specific constraints on their structure and are represented in the following form:

$$R(x) = x^m + r_{m-1}x^{m-1} + r_{m-2}x^{m-2} + r_{m-3}x^{m-3} + ... + r_1x + 1; \quad \forall\, r_i \in \mathrm{GF}(2), i = m-1 \text{ to } 1 \tag{2.3}$$

moreover, since x is the root of $R(x)$, one can also have

$$x^m = r_{m-1}x^{m-1} + r_{m-2}x^{m-2} + ... + r_1x + 1$$

All one polynomials (AOP) are the class of irreducible polynomials that have all of its polynomial coefficients equal to 1, i.e. all $r_i = 1$, and are of the form

$$R(x) = x^m + x^{m-1} + x^{m-2} + ... + x + 1 \tag{2.4}$$

further, since x is the root of $R(x)$, one can also have

$$x^m = x^{m-1} + x^{m-2} + x^{m-3} + ... + x + 1$$

Equally spaced polynomials (ESP) are the class of irreducible polynomials that have equal spacing with respect to degree of the polynomial terms. Clearly, AOPs are are ESPs with spacing of 1. The general form of ESPs is given by

$$\begin{aligned} R(x) &= \sum_{j=0}^{l} x^{js}, \text{for } j = 0, 1, 2, ..., l \\ &= x^{sl} + x^{s(l-1)} + ... + x^s + 1 \end{aligned} \tag{2.5}$$

also, since x is the root of $R(x)$, one can also have $x^{sl} = x^{s(l-1)} + ... + x^s + 1$.
Pentanomials are the class of irreducible polynomials that have only five terms and are of the form,

$$R(x) = x^m + x^{m_3} + x^{m_2} + x^{m_1} + 1, \text{where, } 1 \leq m_1 < m_2 < m_3 \leq (m-1) \tag{2.6}$$

moreover, since x is the root of $R(x)$, one can also have $x^m = x^{m_3} + x^{m_2} + x^{m_1} + 1$. Trinomials are the class of irreducible polynomials that have only three terms and are of the form,

$$R(x) = x^m + x^k + 1, \text{where, } 1 \leq k \leq (m-1) \tag{2.7}$$

Further, since x is the root of $R(x)$, one can also have $x^m = x^k + 1$. Examples for trinomials include the trinomials $R(x) = x^{193} + x^{15} + 1$ and $R(x) = x^{409} + x^{87} + 1$ recommended by National Institute of Standards and Technology (NIST) for ECC.

Finite fields GF(2^m) can also be viewed as vector spaces of dimension, 'm'. Hence, a finite field GF(2^m), which consists 2^m elements, can also be represented using a specific set of any of its m linearly independent elements called basis. Thus, every element in the field GF(2^m) can be expressed as a linear combination of its m basis elements. It's important to note that a finite field may possess multiple bases. Consequently, elements within the finite field GF(2^m) are typically represented using either polynomial basis or normal basis.

Polynomial Basis: The polynomial basis is constructed from the set $(1, x, x^2, x^3, ..., x^{m-2}, x^{m-1}$ where x is a root of the irreducible polynomial $R(x)$ in the field GF(2^m). When using the polynomial basis to represent an element $A(x) \in$ GF(2^m), it takes the following form:

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_0, \quad a_i \in \text{GF}(2)$$

Normal Basis: The normal basis is constructed from the set $(x, x^2, x^{2^2}, x^{2^3}, ..., x^{2^{m-2}}, x^{2^{m-1}})$, where x is a root of the irreducible polynomial $R(x)$ in the field GF(2^m). When using the normal basis to represent an element $A(x) \in$ GF(2^m), it takes the following form:

$$A(x) = a_{m-1}x^{2^{m-1}} + a_{m-2}x^{2^{m-2}} + \ldots + a_0x, \quad a_i \in \text{GF}(2)$$

2.2 Finite Field GF(2^m) Arithmetic

Various arithmetic operations are carried out in binary fields namely, addition, subtraction, multiplication, inversion, and squaring. Addition and subtraction are two simple

operations and are realized by using logical XOR circuits. However, multiplication and squaring operations are realized in two steps: first, the usual arithmetic operation is carried out, and then it is followed by a modulo reduction.

Considering two elements $A(x) = x^7 + x^5 + x^4 + x^3 + x + 1$ and $B(x) = x^7 + x^6 + x^4 + x^3 + x^2 + x + 1$ from the GF(2^8) field, the addition can be realized as

$$A(x) + B(x) = x^6 + x^5 + x^2$$

2.3 Finite Field GF(2^m) Multiplication

Multiplication is one complex operation determining the efficiency of point multiplication. Multiplication in polynomial basis is simpler and also gives more regular and compact realizations compared to other bases. Hence, we have selected polynomial basis for realizing GF(2^m) multiplication and the same basis is adopted throughout the thesis.

Consider two arbitrary elements in GF(2^{m}), denoted as A(x) and B(x), which can be expressed as follows:

$$A(x) = \sum_{l=0}^{m-1} a_l x^l = a_0 + a_1 x a_2 x^+ \ldots + a_{m-2} x^{m-2} + a_{m-1} x^{m-1} \tag{2.8}$$

$$B(x) = \sum_{l=0}^{m-1} b_l x^l = b_0 + b_1 x + \ldots + b_{m-2} x^{m-2} + b_{m-1} x^{m-1} \tag{2.9}$$

where a_l and b_l belong to the field GF(2) for $0 \leq l \leq m-1$.

The product of these field elements, A(x) and B(x), over GF(2^{m}) is given by:

$$C(x) = A(x) \cdot B(x) \bmod R(x) \tag{2.10}$$

Here, $R(x)$ represents a general irreducible polynomial.

2.4 Elliptic Curve Cryptography

Elliptic curve cryptography was introduced by Victor Miller and Neil Koblitz [3] and it gained significant attention in the recent research works available in the literature. IEEE

and NIST standards recommended ECC as an effective approach for implementing public key cryptography within resource-constrained and embedded environments. It is observed from the analysis of ECC that it requires less key size while providing the same level of security as that offered by RSA technique. The reduced key size necessitates less hardware, making it particularly suitable for devices with limited resources, like storage, silicon-area, and bandwidth. The challenge posed by solving the discrete logarithm problem in elliptic curves contributes to enhancing the security provided by ECC [3].

2.4.1 Elliptic Curve Point-Multiplication

An elliptic curve E over GF(2^m) is represented by the following expression [2]

$$E : y^2 + x \cdot y = x^3 + a \cdot x^2 + b \tag{2.11}$$

where a and $b \in GF\left(2^m\right), b \neq 0$. A pair of elements $x, y \in GF\left(2^m\right)$ represent a point on the elliptic-curve E and the point $P = (x, y) \in E\left(GF\left(2^m\right)\right)$, satisfies the tangent and chord laws. There are various coordinate systems used to represent the point P=(x, y) on the elliptic curve. Considering the computational complexity, the most commonly used coordinate systems are affine-coordinate system and projective-coordinate system. Within the affine-coordinate system, a point P is denoted by its two coordinates (x, y), whereas the projective-coordinate system employs three coordinates (X, Y, Z) to represent a point P.

Let P and Q be two points on the curve E, and K a positive integer. The computation of Q such that it is a multiple of point P is referred to as point-multiplication. The point Q can be obtained through the following equation:

$$Q = kP = P + P + P + \ldots + P \tag{2.12}$$

Here, k takes values within the range $[0, m-1]$. Point-multiplication is achieved by a series of point-addition (PA) and point-doubling (PD) operations.

Given two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ on the elliptic curve E, combining these points results in another point P_3 on the same curve. This operation, known as point-addition, is expressed as $P_3 = (x_3, y_3) = P_1 + P_2$. When the point-addition involves the

same point $P_1 = P_2$, it is termed a point-doubling operation, represented as $P_3 = 2P_1 = 2P_2$. The expressions for PA and PD in the affine-coordinate systems are provided in [7].

$$x_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + \mathrm{a}, & \mathrm{P}_1 \neq \mathrm{P}_2 \\ x_1^2 + \frac{\mathrm{b}}{x_1^2}, & \mathrm{P}_1 = \mathrm{P}_2 \end{cases} \tag{2.13}$$

$$y_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1 + x_3) + x_3 + y_1, & \mathrm{P}_1 \neq \mathrm{P}_2 \\ x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right)(x_3) + x_3, & \mathrm{P}_1 = \mathrm{P}_2 \end{cases} \tag{2.14}$$

It can be observed from the Eq.(2.13) and Eq.(2.14) that each PA and PD operation involves an FF-inversion, which is computationally intensive FF-operation requiring more area and time. If m is the order of the field then it involves around 2m number of FF-inversion operations for point-multiplication using affine-coordinate systems. In order to avoid the FF-inversion operations, it is suggested to use projective-coordinate systems. Lopez-Dahab, standard, and Jacobian are the most preferred projective coordinate systems. For any affine point P(x,y), the projective coordinate point P is given as $\left(X/Z^{\mathrm{c}}, Y/Z^{\mathrm{d}}\right)$ where, Z$\neq$ 0. The values of c, d for Lopez-Dahab, standard, and Jacobian are given as c=1 & d=2, c=1 & d=1 and c=2 & d=3, respectively [8].

In the proposed work we have used Lopez-Dahab projective-coordinates and the expressions for PA and PD operations using Lopez-Dahab projective-coordinates are given as respectively [9],

$$\begin{aligned} \mathrm{X}_3 &= x \cdot \mathrm{Z}_3 + (\mathrm{X}_1 \cdot \mathrm{Z}_2) \cdot (\mathrm{X}_2 \cdot \mathrm{Z}_1) \\ \mathrm{Z}_3 &= (\mathrm{X}_1 \cdot \mathrm{Z}_2 + \mathrm{X}_2 \cdot \mathrm{Z}_1)^2 \end{aligned} \tag{2.15}$$

$$\begin{aligned} X_3 &= \mathrm{X}_1^4 + \mathrm{b}.\mathrm{Z}_1^4 \ \text{ or } \ \mathrm{X}_3 = \mathrm{X}_2^4 + \mathrm{b}.\mathrm{Z}_2^4 \\ Z_3 &= \mathrm{Z}_1^2 \cdot \mathrm{X}_1^2 \ \text{ or } \ \mathrm{Z}_3 = \mathrm{Z}_2^2 \cdot \mathrm{X}_2^2 \end{aligned} \tag{2.16}$$

2.4.2 Point-Multiplication Algorithms

There are several algorithms presented in the literature to achieve point multiplication namely, double-and-add algorithm(DAA), Non-adjacent-form(NAF) algorithm, window-NAF (width-w NAF) algorithm, Left-to-right algorithm, Right-to-left algorithm,

Montgomery-ladder algorithm,Sliding-window-algorithm, and τ-adic-NAF (τ-NAF) algorithm. The review of these algorithms is presented in [10].

Algorithm 2.1 and Algorithm 2.2 illustrate the point multiplication algorithms for right-to-left and left-to-right approaches, respectively. Algorithm 2.1 operates by processing the bits of scalar k from right to left, while Algorithm 2.2 processes the bits from left to right. In both algorithms, every bit of the scalar k is examined, and depending on whether the bit value is '0' or '1', either a point-doubling (PD) operation or a combination of PD and point-addition (PA) operations is executed. Notably, the right-to-left algorithm enables parallel execution of PD and PA operations.

The computational cost of each point multiplication algorithm depends on the Hamming weight $H(k)$ of secret key k. The double and add approach requires around $l-1$ number of point doubling operations and $H(k)-1$ number of point addition operations. One efficient approach to reduce Hamming weight is to use Non-Adjacent Form (NAF), wherein consecutive bits are never simultaneously nonzero. Through the NAF method, the Hamming weight is reduced to approximately $H(k) \approx l/3$.

Algorithm 2.1 Right-to-left Algorithm [10]

Input: $k = (k_{l-1}, k_{l-2}, \ldots, k_2, k_1, k_0), P \in$

Output: kP

1. $Q \leftarrow \mathcal{O}$
2. for i from 0 downto $l-1$ do
3. if $k_i = 1$ then $Q \leftarrow P + Q$
4. $P \leftarrow 2P$
5. end if
6. end for
7. Return Q

Algorithm 2.2 Left-to-right Algorithm [10]

Input: $k = (k_{l-1}, k_{l-2}, \ldots, k_2, k_1, k_0), P \in$

Output: kP

1. $Q \leftarrow \mathcal{O}$
2. for i from $l-1$ downto 0 do
3. $Q \leftarrow 2Q$
4. if $k_i = 1$ then $Q \leftarrow P + Q$
5. end if
6. end for
7. Return Q

Algorithm 2.3 Binary NAF Algorithm [10]

Input: $k = (k_{l-1}, k_{l-2}, \ldots, k_2, k_1, k_0), P \in \mathbb{F}_q$

Output: kP

1. Compute $\mathrm{NAF}(k) = \sum_{i=0}^{l-1} k_i 2^i$
2. $Q \leftarrow \mathcal{O}$
3. for i from $l-1$ downto 0 do
4. $Q \leftarrow 2Q$
5. if $k_i = 1$ then $Q \leftarrow Q + P$
6. end if
7. if $k_i = -1$ then $Q \leftarrow Q - P$
8. end if
9. end for
10. Return Q

Algorithm 2.4 Window NAF Algorithm [10]

Input: $k = (k_{l-1}, k_{l-2}, \ldots, k_2, k_1, k_0), P \in \mathbb{F}_\varphi$

Output: kP

1. Compute $\mathrm{NAF}_w(k) = \sum_{i=0}^{l-1} k_i 2^i$
2. Compute $P_i = iP$ for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
3. $Q \leftarrow \mathcal{O}$
4. for i from $l-1$ downto 0 do
5. $Q \leftarrow 2Q$
6. if $k_i \neq 0$ then
7. if $k_i > 0$ then $Q \leftarrow Q + P_{k_i}$
8. else $Q \leftarrow Q - P_{-k_i}$
9. end if
10. end if
11. end for
12. Return Q

Algorithm 2.5 τ NAF Algorithm [10]

Input: $k = (k_{l-1}, k_{l-2}, \ldots, k_2, k_1, k_0), P \in \mathbb{F}_q$

Output: kP

1. Compute $\tau\mathrm{NAF}(k) = \sum_{i=0}^{l-1} u_i \tau^i$
2. if $u_{l-1} = 1$ then $Q \leftarrow P$ else $Q \leftarrow -P$
3. end if
4. for i from $l-2$ downto 0 do
5. $Q \leftarrow \varphi(Q)$
6. if $u_i = 1$ then $Q \leftarrow Q + P$
7. end if
8. if $u_i = -1$ then $Q \leftarrow Q - P$
9. end if
10. end for
11. Return Q

Algorithm 2.6 Montgomery ladder Algorithm [10]

Input: $k = (k_{l-1}, k_{l-2}, \ldots, k_2, k_1, k_0)$, with $k_{l-1} = 1, P \in \mathbb{F}_q$

Output: kP

1. $Q_1 \leftarrow P, Q_2 \leftarrow 2P$
2. for i from $l-2$ downto 0 do
3. if $k_i = 1$ then
4. $Q_1 \leftarrow Q_1 + Q_2, Q_2 \leftarrow 2Q_2$
5. else
6. $Q_2 \leftarrow Q_1 + Q_2, Q_1 \leftarrow 2Q_1$
7. end if
8. end for
9. Return Q_1

The operational cost of these algorithms for different levels of optimization is presented in Table 2.1.

Table 2.1 Comparison of various point multiplication algorithms

Algorithm	Operational cost (best optimization)	(least Optimization)
Double and ADD (left to right, Right to left and Montgomery)	m/2 PA's and m PD's	m PA's and m PD's
Addition-Subtraction (NAF,Window-NAF and τ-NAF)	m/3 PA's and m PD's	m/2 PA's and m PD's

where, PA point addition, PD point doubling, and m field size

The implementations of these algorithms are susceptible to various forms of attacks, with side-channel attacks being the most prevalent among the existing vulnerabilities in point-multiplication algorithms [2].In these attacks, adversary tries to encrypt the secret key either by analyzing the timing or power behavior of the set of finite-field operations involved in the algorithm. The mathematical modeling for PA and PD operations are quiet different and therefore their power traces and timing traces can be easily distinguished. In timing-attacks, the adversary tries to encrypt the secret key by analyzing timing of each Finite-field operation, whereas the power of each FF-operation is traced in power analysis attack. The Power analysis attacks are of two types: the simple power analysis (SPA) attack and the differential power analysis (DPA) attack. In an SPA attack, the power trace of an individual key operation is analyzed to uncover the secret key. In

contrast, a DPA attack involves an adversary collecting a series of power traces and subsequently analyzing the secret key by grouping these collected traces. Considering the timing and power-analysis attacks, Montgomery [11] has derived an algorithm, which performs same set of operations in every iteration of the main-loop irrespective of the secret key value, avoiding timing and power analysis attacks. In addition, Montgomery-ladder algorithm reduces the computational complexity by using only affine x coordinates during initialization, and projective X and Z coordinates in the main loop of point-multiplication algorithm. The y coordinate is recovered at the end of main-loop using X and Z coordinates.

2.5 Conclusions

This chapter presents a brief summary of finite field arithmetic. First, we present the definitions and properties of Groups, Rings, and Fields. Followed by the definitions of finite field, binary finite field GF(2^m), and finite field GF(2^m) arithmetic operations. Finally, elliptic curve cryptography and its underlying operations are presented along with a few point multiplication algorithms available in the literature. The next chapter presents the review of finite field GF(2^m) point multiplication architectures available in the literature.

Chapter 3

Polynomial Basis GF(2^m) Point Multiplication Architectures

The hardware realization of ECC applications require a low complexity and high performance point multiplication architecture. The performance of point multiplication architecture depends on the underlying finite field arithmetic. Finite field inversion and finite field multiplication are two important arithmetic operations that dominate the performance of point multiplication operation. In this chapter, a brief review of the different FF-inversion architectures and FF-multiplication architectures proposed in the literature are presented. In addition, a review of different point multiplication architectures available in the literature are presented.

3.1 Review of Finite Field Inversion Architectures

In this section, different FF-inversion architectures proposed in the literature over GF(2^m) for general irreducible polynomials based on the Itoh-Tsujii algorithm (ITA) and Extended Euclidean algorithm (EEA) are presented. The EEA computes FF-inversion using the greatest common divisor approach, while the ITA uses Fermat's little theorem [8].

3.1.1 EEA based Finite Field Inversion Architectures

Several architectures are presented in the literature for computing FF-inversion based on EEA. The implementation of traditional EEA requires repeated sequential division and multiplication operations which increases the hardware and time complexities [12]. These problems are addressed by proposing reformulations to EEA and implementing FF-inversion using parallelism techniques [13–20].

In 2006, a reformulated EEA for FF-inversion using a digit-serial systolic architecture is presented by Yan [13]. This architecture is developed employing unfolding techniques to realize FF-inversion in a systematic approach to achieve throughput of L/m and area-time complexity of O(Lm), where L is the digit size. It is observed that the architecture achieves reduction of about 50% in critical path delay while achieving the same latency and throughput L/m compared to architectures available in the literature [21].

A low area and low power scalable systolic architecture for FF-inversion, suitable for fixed size processors is introduced by Ibrahim et al. [15] in 2016. This architecture is modular and has a latency of $(2m-1)(m/T+1)$ with critical path delay of $2T_{mux}$, where T is number of processing elements. The design achieves reduction in area of about 88% compared to available FF-inversion architectures and profound to be suitable for resource constrained cryptographic implementations [13].

In 2017, Ibrahim et al. [16] presented a bit-serial systolic array architecture with low area complexity and moderate speed. This systolic architecture has simple structure and the local processing elements can communication with each other. The latency of this architecture is $(2m-1)$ and the critical path delay is $2T_{mux}$.

Ibrahim et al. [17] in 2017 proposed a unified and scalable architecture for FF-inversion based on modified EEA. In this unified design, the FF-multiplier and FF-inverter share the data-path to save area and power resources. Furthermore, the architectures scalability makes it suitable for fixed size processor without varying the field size m. The latency of the design is $(m)(m/N+1)$ and the critical path delay is $T_A + T_X + 2T_{MUX}$.

Hazmi et al. [19] proposed in 2019 a modified EEA based FF-inverter to explain the possibilities of parallelism and concurrency to minimize the design space. Polynomial multiplication and division are analyzed to enable concurrent execution of EEA, resulting

in enhanced area and time complexities. The latency of this design is $(2m-2))$ and the critical path delay is $T_A + 2T_X$.

In 2019, Rashidi et al. [20] proposed a modified EEA algorithm for FF-inversion over GF(2^m) using polynomial basis. The design is able to perform FF-operations corresponding to two iterations in just one iteration compared to the other EEA-based inversion algorithms reported in the literature. The latency of this design is m and the critical path delay is $T_A + T_X + T_O + T_{4MUX}$.

3.1.2 ITA based Finite Field Inversion Architectures

The computation of FF-inversion using ITA requires a sequence of FF-Multiplications and FF-Squaring operations. The ITA was initially presented using the normal basis (NB), since FF-squaring with NB is just a cyclic shift [22, 23]. However, the polynomial basis is preferred for the realization of FF-inversion algorithm as the normal basis increases the computational complexity. In the recent past, several Itoh-Tsujiis algorithm (ITA) based architectures are proposed for the realization of Finite-field inversion [8, 22–36].

In 2005, Rodriguez et al. [24] presented an FF-inversion architecture in GF(2^m) based on Itoh-Tsujii algorithm. The design is able reduce the number of clock-cycles required to implement ITA by cascading 2^1 exponential blocks, and the number of cascade blocks required to realize the FF-inversion depends on the addition-chain. The main building blocks of this design are FF-multiplier and FF-squarer. The design is able to achieve FF-inversion using $(m-1)$ FF-squarings and $|\log_2(\mathrm{m}-1)| + \mathrm{H}(\mathrm{m}-1) - 1$ FF-multiplications, where m, is field order and H is hamming weight.

Rodriguez et al. [26] in 2007 proposed an FF-inversion architecture based on a novel parallel Itoh-Tsujii algorithm. The formulations of the proposed parallel Itoh-Tsujii algorithm involves parallel implementation of the square and square-root operations. The simultaneous implementation of these FF-operations resulted in the reduction of clock cycles required to realize FF-inversion. This design is able to achieve FF-inversion using $(m-1)$ FF-squaroots and $|\log_2(\mathrm{m}-1)| + \mathrm{H}(\mathrm{m}-1) - 1$ FF-multiplications, where m is field order and H is hamming weight.

In 2011, Rebeiro et al. [25] presented a FF-inversion architecture based on Quad-

ITA. This architecture accelerates the FF-inversion operation by using quad circuits (2^2) in place of traditional square circuits (2^2). The area-time complexities of an FF-inversion architecture are influenced by the use addition chain. This architecture is able to reduce the length of addition chain from $(m-1)$ to $(m-1)/2)$ and is able to realize FF-inversion using $(m-1)/2)$-1 FF-quads and $|\log_2(\mathrm{m}-1)| + \mathrm{H}(\mathrm{m}-1)$ FF-multiplications, where m is field order and H is hamming weight.

Roy et al. [27] presented the analysis of ITA algorithm to determine the effect of various design methodologies on the delay, area, and clock cycle requirements for FPGA implementation in 2011. This developed model is able to estimate the ideal number of cascade blocks and the best optimal exponential circuit to be used to obtain maximum performance in realizing FF-inversion.

A generalized model was developed by Roy et al. [28] in 2012 to compute FF-inversion using two sets of exponentiation 2^k blocks in parallel to enhance the clock frequency. This design uses either 2^k or 2^{-k} cascade blocks in parallel with a large exponentiation block to realize FF-inversion. This work involves study and analysis to determine the best combination of 2^k or 2^{-k} cascade block and large exponentiation block to be used for any NIST recommended field order.

Rashidi et al. [30] presented in 2016, high-performance implementation of polynomial basis ItohTsujii inversion algorithm (ITA) over GF(2^m) using irreducible trinomials and pentanomials. This implementation uses k-times squarer blocks and high-speed GF(2^m) digit-serial multiplier to realize FF-inversion. The critical path of the design is decreased by placing registers at appropriate k-times squarer blocks output.

In 2017, Li et al. [29] proposed an FF-inversion architecture based on modified Itoh-Tsujii algorithm. This modified ITA along with optimal addition chains enables parallel FF-multiplication and FF-squaring operations resulting in reduction of number of clock cycles required to realize FF-inversion without incurring any additional hardware.

Two novel architectures, namely, High-speed(HS) and Least-clock-cycle(LCC)-ITA were presented by Li et al. [31] to improve the computation time using 2^k blocks. Several high exponential 2^k blocks are precomputed and used to realize FF-inversion and this resulted in achieving any exponentiation of 2 in a single clock cycle at the cost of a

substantial increase in area.

It is observed in the above-mentioned methodologies that the parallel-ITA architectures and higher exponentiation circuits may achieve reduction in the number of clock cycles at the cost of a significant increase in hardware LUTs. An FF-inversion architecture with balanced time complexity and hardware complexity, can be achieved by exploring strategies to find the optimal exponential circuit.

3.2 Review of Digit-Serial Multiplier Architectures

In general, GF(2^m) multipliers can be realized using digit-serial, bit-parallel and bit-serial multiplier architectures [26]. The bit-serial multiplier processes a single-bit of m input data for each clock cycle and achieves multiplication over m iterations. The digit-serial multiplier, on the other hand, processes a group of D bits in a single clock cycle and accomplishes multiplication within m/D iterations. Lastly, the bit-parallel multiplier simultaneously processes m bits during a single clock cycle, completing the multiplication in a single iteration.

Bit-serial architectures achieve reduced hardware complexity by employing the same hardware across multiple iterations, but exhibit high latency. Conversely, bit-parallel architectures attain high throughput by replicating arithmetic blocks, at the expense of increased implementation hardware. The digit-serial architectures are versatile and they facilitate in area-delay tradeoff. However, it is observed with the increment of D value the area and time complexities increase.

Several GF(2^m) digit-serial multipliers are presented in the literature for general irreducible polynomials using polynomial-basis [21, 37–45]. The performance of these digit- serial architectures in terms of area and time is presented in Table 3.2.

In 1998, two digit-serial multipliers namely MSD-first and LSD-first multipliers are presented by Song et al. [41]. These designs introduced a new approach of realizing FF-multiplication by combining array-type algorithm and parallel algorithm. The array-type approach reduces the latency while the parallel structure inside each digit-cell reduces the critical path delay of the multiplier. The latency of the both the architectures is same

Table 3.1 Area and Time Complexities of the available Digit-serial Multipliers

Design	AND	XOR/XNOR	NAND	MUX	Register	Latency	Critical path
Song [41]	Dm	Dm+3D-2	0	m	3m+D-1	(m/D) +2	$(\lceil log_2(D) \rceil + 1)T_X$ $+T_A + T_D + T_M$
Song [41]	mD	Dm+3D	0	0	2m+D	(m/D) +2	$(\lceil log_2(D) \rceil + 2)T_X$ $+T_A + T_D$
Kim [38]	m(2D+1)	2mD	0	2m	10m+(m/D)	3(m/D)	$D(T_X + T_A + T_M)$ $+T_M$
Tang [42]	Dm	Dm+D(D+1)/2	0	0	m	(m/D)-1	$(\lceil log_2(D) \rceil + 2)T_X$ $+T_A + T_D$
Kumar [37]	mD	m(D+1)+3D-3	0	m	4m+D-2	(m/D) +2	$(\lceil log_2(D/2) \rceil + 1)T_X$ $+T_A + T_D + T_M$

and is equal to $(m/D) + 2$, while critical path delay of LSD-First multiplier is given as $(\lceil log_2(D) \rceil + 1)T_X + T_A + T_D + T_M$ and MSD-first multiplier is given as $(\lceil log_2(D) \rceil + 2)T_X + T_A + T_D + T_M$.

Tang et al. [42] proposed in 2005 a MSD-first bit-parallel word-serial multiplier for GF(2^{233}). In this design, an 8×233 parallel multiplier is used to generate the FF-multiplication partial products. The latency and critical path delay of this design are given as $(m/D) - 1$ and $(\lceil log_2(D) \rceil + 2)T_X + T_A + T_D$, respectively. This multiplier achieves considerable reduction in area and time complexities compared to the MSD-first multiplier [41].

An efficient digit-serial systolic array multiplier over GF(2^m) using standard basis is presented by Kim et al. [38] in 2005. In this design a new dependence graph is obtained to design an efficient digit-serial systolic multiplier based on LSD-first multiplication algorithm. It is observed for a continuous input of data the design is able to produce multiplication results at a rate of one per every m/D clock cycles. The latency and critical path delay of the design are given as $3(m/D)$ and $D(T_X + T_A + T_M) + T_M$, respectively.

In 2006, Kumar et al. [37] proposed two architectures based on the least significant digit first(LSD-first) algorithm namely, N-accumulator multiplier and double accumulator multiplier. These designs achieved considerable throughput by utilizing registers to retain partial products at the expense of high area and power complexities. The latency and critical path delay of these designs are given as $(m/D) + 2$ and $(\lceil log_2(D/2) \rceil + 1)T_X + T_A + T_D + T_M$, respectively.

3.3 Review of Point Multiplication Architectures

In 1980s, ECC a asymmetric-key algorithm proposed by Neal Koblitz [3] and Victor Miller [46]. The small key size, less computation time, and low-area complexity preserving the same level of security attracts ECC in comparison to other asymmetric-key cryptosystems. The performance of ECC in resource-constrained devices relies on the performance of the point multiplication operation. Many architectures are proposed in the literature

to design high-speed and low complexity ECC [5, 6, 47–58]. Most of these works aimed to reduce computation time by minimizing the latency for computation of point multiplication. The performance comparison of these point multiplication architectures in terms of critical path delay and latency are presented in Table 3.2.

Table 3.2 Time-complexities of the available Point Multiplication Architectures

Design	latency(clock cycles)	critical-path delay
[51]	$2mL + \left[L(l+1) + \sum_{i=2}^{l} \left\lceil \frac{u_i - u_{i-1}}{u_s} \right\rceil\right] + [7 + 9L]$	$\log_{\mathrm{d}}(2 + \log_2 2) + \log_{\mathrm{d}}(4 + \log_2 4) + 3$
[47]	$m \cdot 7 + \frac{m}{3} \cdot 15 + \left[L(l+1) + \sum_{i=2}^{l} \left\lceil \frac{u_i - u_{i-1}}{u_s} \right\rceil\right] + 7$	$2\log_{\mathrm{d}}(4 + \log_4 2) + 2$
[59]	$5 + (6)(m-1) + (m-1)/2 + 3(\mathrm{H(m-1)} + \lvert \log_2(\mathrm{m}-1)\rvert\text{-}1) + 31$	$\log_2(n + 2r) + \log_{\mathrm{d}}(2 + \log_2 2) + 1$
[54]	$3(w-1)d + 5(2^w - (w+1)) + 7d$	$log_{\mathrm{d}}(\frac{\mathrm{m}}{\tau})) + log_{\mathrm{d}} 2\tau)$
[55]	$3\mathrm{m} + \sum_{i=2}^{l} \lceil \frac{u_i}{5} \rceil + 2(\mathrm{H(m-1)} + \lvert \log_2(\mathrm{m}-1)\rvert - 1)$	–

L is number of pipeline stages, l is length of addition chain, u_s is maximum cascade size, H is hamming weight of m-1,u_i is element in the addition chain, d is digit size, w is window size,n is number of segments in a multiplier, r is r-nomial irreducible polynomial, τ is threshold multiplier size

Sakiyama et al. [48] presented in 2006 a super scalar ECC co-processor over GF(2^m) using polynomial basis. This architecture is able to accelerate the point multiplication operation by employing instruction level parallelism. However, the processor is profound to be impractical in cases of slow I/O communication channels.

In 2008, Chelton et al. [6], developed a pipelined architecture for point-multiplication over GF(2^m) using polynomial basis. The architecture is investigated with different levels of pipelining for optimal data path. A new combined algorithm is developed to achieve point-addition and point-doubling in nine instructions. The computation time of this point multiplication when implemented on Xilinx XC4VLX200 over GF(2^{163}) is 19.55 us. However, because of more pipelined stages as high as seven, the latency becomes very high.

A high throughput point multiplication architecture was proposed by Kim et al. [49] in 2008 based on Lopez-Dahab algorithm for point multiplication. This design employs

Gaussian normal basis multipliers and parallelized point-doubling and point-addition units developed with uniform addressing to realize point multiplication. The computation time of this point multiplication over GF(2^{163}) on Xilinx XC4VLX80 is 10 us.

A high speed point multiplication architecture based on Montgomery algorithm is proposed by Hasan et al. [5] in 2008. A Pseudo-pipelined word-serial GF(2^m) multiplier is developed to perform underlying FF-multiplication operations and a scheduling scheme is adopted to carry out FF-operations with no idle clock cycles. The presented architecture is able to achieve point multiplication in $25(m-1)$ clock cycles and is found to be 1.6 times faster than previous works reported in the literature. This point multiplication architecture over GF(2^{163}) requires 4050 clock cycles with computation time of 41 us when implemented on Xilinx XC2V2000.

Azarderakhsh et al. [50] proposed in 2011 an efficient point multiplication architecture on Generalized Hessian and Binary Edwards curves. A digit level Gaussian normal basis GF(2^m) multiplier is presented and the area-time trade-off has been studied for various digit sizes. The architecture is able to reduce the computation time of GF(2^m) point multiplication by using two pipelined Gaussian normal basis multipliers operated in parallel. The computation time for this point multiplication architecture over GF(2^{163}) is 12.9 us, when implemented on Virtex-5 FPGA.

In 2012, Rebeiro et al. [51] presented a low clock cycle model for point multiplication by employing a bit-parallel Karatsuba multiplier. The architecture presented is mathematically analyzed to explore the best optimal pipelined path.The lower level arithmetic operations are scheduled appropriately to support data forwarding. The design uses only single GF(2^m) multiplier to preform point multiplication and hence, it is able to achieve considerable reduction in area compared to similar point multiplication architectures available in the literature. The computation times of this point multiplication are 8.6 us, and 12.3 us, when implemented on Xilinx XC5VLX85t over the fields GF(2^{163}), and GF(2^{233}), respectively. However, the architecture have longer critical path delay due to additional finite field arithmetic primitives in the critical path other than FF-multiplier.

In 2013, Roy et al. [47] proposed a pipelined point multiplication architecture employing bit-parallel Karatsuba multiplier. A theoretical model is developed to analyze the critical path and estimate the optimal pipelined stages to achieve point multiplication.

The design is able to accelerate the FF-inversion operation using a series of power blocks. It is reported that the computation time of this point multiplication architecture over GF(2^{233}) is 9.5 us, when implemented on Xilinx XC5VSX240. However, the architecture have low clock frequency due to additional finite field arithmetic primitives in the critical path other than FF-multiplier.

Sutter et al. [52] in 2013 presented a point multiplication architecture using a digit-serial FF-multiplier. The architecture is investigated with different digit sizes to perform GF(2^m) multiplication and division operations over each of the five NIST recommend elliptic curves. The architecture is able to speed up the point multiplication operation by using three digit-serial multipliers operated in parallel. The computation times of this point multiplication are 5.5 us and 17.8 us, over GF(2^{163}), and GF(2^{233}), respectively, when implemented on Xilinx XC5VLX110-3. However, there is considerable increase in area because of three parallel multipliers employed to realize GF(2^m) multiplication.

An area-time efficient architecture for point multiplication employing bit serial GF(2^m) multiplier was proposed by Nguyen et al. [53] in 2016. The architecture developed is based on the left-to-right and Lopez-Dahab point multiplication algorithms. To improve the latency of point multiplication a series of bit serial GF(2^m) multipliers are used to operate in parallel for realizing GF(2^m) multiplication. The computation time of this point multiplication architecture over GF(2^{163}) is 94.6 us, when implemented on Xilinx Virtex-5 FPGA.

Khan et al. [59] in 2017 presented two point multiplication architectures based on Montgomery ladder algorithm. A segment pipelined GF(2^m) multiplier is used to reduce the point multiplication computation time and a scheduling scheme alongside cascaded blocks of FF-squarer, FF-adder and FF-multiplier blocks are employed to achieve point multiplication with least clock cycles. The high-speed and low latency architectures are able to achieve point multiplication in six clock-cycles and two clock-cycles, respectively. The computation time of this point multiplication over GF(2^{163}) is 4.91 us when implemented targeting Xilinx XC5VLX50.

In 2018, Salarifard et al. [54] proposed two low complexity and low latency architectures based on fixed comb point multiplication algorithm. The fixed comb point multiplication algorithm has a precomputation phase which reduces the number of point

addition operations and hence, reduces the latency of the point multiplication. The computation time of this point multiplication architecture over GF(2^{163}), and GF(2^{233}), are 2.22 us, and 5.79 us, respectively, when implemented on Xilinx Virtex-5 FPGA.

Li et al. [55] in 2019, proposed a high performance point multiplication architecture and an FF-inversion architecture using two parallel balanced precision multipliers to reduce the latency. The computation time of this point multiplication architecture over GF(2^{163}) is 2.6 us when implemented on Xilinx Virtex-5 FPGA.

3.4 Conclusions

In this chapter, a survey of different architectures of finite field inversion, finite field multiplication and point multiplication available in the literature and the performance analysis of these architectures in terms area and time complexities are presented. The next chapter presents the design of the proposed finite-field inversion architecture for general irreducible polynomials.

Chapter 4

An Area-Efficient Architecture for Finite Field Inversion over GF(2^m) using Polynomial Basis

This chapter presents the design of Finite-field inversion architecture for general irreducible polynomials targeting ECC applications. The FF-inversion architecture is developed based on the modified ItohTsujii algorithm. The classic ItohTsujii algorithm is modified to realize ff-inversion using 4^k exponentiation modules in place of traditional 2^k exponentiation modules. The formulations for area and time complexities are made and evaluated for the field orders $m = 193, 233, 409$. The proposed architecture is modeled using verilog HDL and simulated to verify the functionality using Xilinx Vivado tools. The HDL netlist is synthesized targeting Xilinx Virtex-5 FPGA and implemented to compare with the similar FF-inversion architectures available in the literature.

4.1 Introduction

Finite fields are widely used in various cryptographic schemes such as asymmetric and symmetric key cryptosystems. ECC is one of the reliable and high-speed asymmetric key cryptosystems. The complexity of hardware realization of ECC depends on the arithmetic operations in finite field. The arithmetic operations used in ECC are namely, FF-addition, FF-subtraction, FF-multiplication, and FF-inversion. Since FF-inversion is the most time-critical and resource-consuming operation, it is required to develop an area efficient and high speed architecture for the realization of FF-inversion algorithm. In ad-

dition, the performance of FF-inversion architectures depends on the type of irreducible polynomial employed for realizing the architecture. The trinomials and pentanomials with less computations involved are more suitable for fast ECC applications. Hence, it is suggestable to design high performance FF-inversion architectures using irreducible trinomials and irreducible pentanomials.

Several architectures are presented in the literature for computing FF-inversion based on EEA [13–20] and ITA [22–36]. These architectures explored the possibilities of reducing the area and computation time required to realize FF-inversion. In this work we derived a modified ITA based on the ITA [22] presented in the literature. Subsequently, the architecture for the proposed modified Itoh-Tsujii algorithm is developed to realize FF-inversion of any arbitrary element $a \in GF(2^m)$. Finally, the area and delay formulations of the proposed architecture are estimated and the performance is compared with the existing FF-inversion architectures in the literature.

4.2 Area-Efficient Finite-field Inversion over GF(2^m)

In this section, the design and performance analysis of the proposed finite field inversion architecture are presented. First, we present the mathematical formulations for the proposed modified Itoh-Tsujii algorithm and its realization using two-stage pipelined architecture. Followed by, formulations to derive the area and time complexities of the proposed architecture are presented. Finally, the comparison of implementation results of the proposed architecture with the similar finite field inversion architectures available in the literature are presented.

4.2.1 Mathematical Formulations

Fermat's Little Theorem [8] states that for any integer a and prime number p,

$$a^{p-1} \equiv 1(\mathrm{mod}p) \tag{4.1}$$

where a is not divisible by p. We can rewrite Eq.(4.1) as

$$a.a^{p-2} \equiv 1(\mathrm{mod}p), \tag{4.2}$$

and

$$a^{-1} \equiv a^{p-2} (\mathrm{mod} p) \tag{4.3}$$

The multiplicative inverse of a nonzero element $a \in GF\,(2^m)$ is derived by using Eq.(4.3), and is given by

$$a^{-1} \equiv a^{2^{\mathrm{m}}-2} \tag{4.4}$$

Using binary exponentiation, Eq.(4.4) can be realized using the Eq.(4.5), and it requires (m-2) FF-multiplications and (m-1) FF-squarers [22]. The main drawback of this method is that it requires $(m-2)$ multiplications for each inversion.

$$a^{2^{\mathrm{m}}-2} = a^{2^1} \times a^{2^2} \times \cdots a^{2^{\mathrm{m}-1}} \tag{4.5}$$

Itoh-Tsujii [22] suggested a more efficient approach to compute each inversion using (m-1) FF-squares and $(|\log_2(\mathrm{m}-1)|+\mathrm{H}(\mathrm{m}-1)-1)$ FF-multiplications, with H being Hamming-weight of the binary m. This approach achieves the reduction in the number of FF-multiplications using addition-chains [60].

Addition-chain

For a positive integer (m-1), an addition-chain is a finite set of positive integers represented as $U = (u_1 ..... u_e)$, where $u_1 = 1$, $u_e = m-1$, and $u_i = u_j + u_l$ for $1 \leq j \leq l \leq i \leq e$. To further simplify, Brauer formulated that for binary sequence $(m-1) = \langle m_{b-1} \ldots m_0 \rangle_2$ with $m_{b-1} = 1$, the addition-chain grows as $u_i = 2u_{i-1}$ for each m_i and $u_{i+1} = u_i + u_1$ for each m_i= 1. For example, the binary sequence $192 = \langle 11000000 \rangle_2$, can utilize the Brauer-chain {1,2,3,6,12,24,48,96,192} for inversion over $GF\,(2^{193})$.

4.2.2 FF-Inversion Algorithm over GF(2^m)

To explain how ITA functions, the Eq.(4.5) can be modified as

$$a^{-1} = [\gamma_{m-1}(a)]^2 \tag{4.6}$$

and

$$\gamma_q(a) = a^{2^q-1} \tag{4.7}$$

with $\gamma_q(a) \in GF\,(2^m)$ and $q \in$ N, where N is a set of natural numbers.
For simplicity, we shall denote $\gamma_q(a)$ by γ_q.

For any two integers $q, r \geq 0$, Eq.(4.7) can be written as

$$\gamma_{q+r}(a) = \gamma_q(a)^{2^r}\gamma_r(a) \tag{4.8}$$

with

$$\begin{aligned}\gamma_{q+r}(a) = a^{2^{q+r}-1} &= \frac{a^{2^{q+r}}}{a} = \frac{\left(a^{2^q}\right)^{2^r}}{a} \\ &= \left(\frac{a^{2^q}}{a}\right)^{2^r}\frac{a^{2^r}}{a} = \left(a^{2^q-1}\right)^{2^r}a^{2^r-1} \\ &= \gamma_q(a)^{2^r}\gamma_r(a)\end{aligned}$$

Eq.(4.8) can be reformulated in terms of index pairs u_{i_1} and u_{i_2} as

$$\gamma_{u_{i_2}+u_{i_1}}(a) = \left[\gamma_{u_{i_1}}(a)\right]^{2^{u_{i_2}}}\gamma_{u_{i_2}}(a) = \gamma_{u_i}(a) = a^{2^{u_i}-1} \tag{4.9}$$

where, $\gamma_{u_1}(a) = a^{2^1-1} = a$. The FF-inverse of a such that $a^{-1} \in GF\,(2^m)$ can be computed by recursively applying Eq.(4.9) for each u_i and applying squaring at the final step u_e. It is given by,

$$\left[\gamma_{u_e}(a)\right]^2 = \left(a^{2^{m-1}-1}\right)^2 = \left(a^{2^m-2}\right) = a^{-1} \tag{4.10}$$

The FF-inverse of $a \in GF\,(2^{193})$ defined over the irreducible trinomial $R(X) = X^{193} + X^{15} + 1$ computed by recursively applying Eq.(4.9) and is presented in Table 4.1. It can be observed from Table 4.1 that each intermediate result requires a series of 2^k exponentiation blocks and an FF-multiplication operation. The number of FF-squaring operations required to generate the final result are as high as m, with each intermediate step requiring $u_i/2$ number of FF-squaring operations. Several architectures are proposed in the literature [24–27, 29, 30] to reduce the computational delay by employing cascade blocks of 2^k with exponentiation k value in and around the value of a certain element (u_i) in the addition-chain. The computational delay of FF-inversion block is further reduced by selecting 2^k blocks with k value equivalent to the required exponentiation in that particular step to complete the execution of this step in one clock cycle [31,32]. However, this technique increases the hardware. The objective of reducing the computational delay and hardware of FF-inversion block is addressed by using 4^k exponentiation modules instead of 2^k modules [25]. It is observed that 4^k exponentiation modules offer the best LUT utilization compared to the 2^k circuits with the same delay as that of 2^k modules.

Table 4.1 FF-Inverse realization of a such that $a^{-1} \in GF\left(2^{193}\right)$ using classical ITA

	$\gamma_{\mathbf{u_i}}(\mathbf{a})$	$\gamma_{\mathbf{u_j}+\mathbf{u_l}}(\mathbf{a})$	Exponentiation
1	$\gamma_1(a)$		a
2	$\gamma_2(a)$	$\gamma_{1+1}(a)$	$(\gamma_1)^{2^1}\gamma_1 = a^{2^2-1}$
3	$\gamma_3(a)$	$\gamma_{2+1}(a)$	$(\gamma_2)^{2^1}\gamma_1 = a^{2^3-1}$
4	$\gamma_6(a)$	$\gamma_{3+3}(a)$	$(\gamma_3)^{2^3}\gamma_3 = a^{2^6-1}$
5	$\gamma_{12}(a)$	$\gamma_{6+6}(a)$	$(\gamma_6)^{2^6}\gamma_6 = a^{2^{12}-1}$
6	$\gamma_{24}(a)$	$\gamma_{12+12}(a)$	$(\gamma_{12})^{2^{12}}\gamma_{12} = a^{2^{24}-1}$
7	$\gamma_{48}(a)$	$\gamma_{24+24}(a)$	$(\gamma_{24})^{2^{24}}\gamma_{24} = a^{2^{48}-1}$
8	$\gamma_{96}(a)$	$\gamma_{48+48}(a)$	$(\gamma_{48})^{2^{48}}\gamma_{48} = a^{2^{96}} - 1$
9	$\gamma_{192}(a)$	$\gamma_{96+96}(a)$	$(\gamma_{96})^{2^{96}}\gamma_{96} = a^{2^{192}-1}$

For any arbitrary element a, the procedure to obtain $a^{-1} \in GF\left(2^m\right)$ using 4^k exponentiation is presented in Algorithm 4.1. In the proposed algorithm, we use $\gamma_{\mathbf{u_i}}(a) = a^{4^{\mathbf{u_i}}-1}$ instead of $\gamma_{\mathbf{u_i}}(a) = a^{2^{\mathbf{u_i}}-1}$ used in classical-ITA. FF-inversion computation starts with pre-computation a^3. Steps 3 to 13 compute the required exponentiation in each intermediate step with maximum exponentiation of u_C per clock cycle. The area overhead of using higher exponentiation is avoided by selecting the value of u_C such that it doesn't alter the critical path delay. If the value of variable k(power of exponentiation 4^k) is greater than u_C, the exponentiation 4^k operation can be computed in multiple clock cycles, which is performed using steps 6 to 8. However, the 4^k operation can be completed in single clock-cycle using step 12, if the value of k is less than u_C. The steps 14 to 18 perform the FF-multiplication operation, in which either step 15 or 17 is implemented based on the k value. Steps 4 to 17 are repeated for all the elements of addition chain until u_f. Finally, the step 20 gives the final result with the FF-squaring operation on last updated value of α_m.

Algorithm 4.1 Modified Itoh-Tsujii Algorithm using exponentiation 4^k

Input: The element $a \in GF\,(2^m)$, addition chain length f, cascade size u_{C} and the addition chain $U = \left\{1, 2, \ldots, \frac{m-1}{2}\right\}$

Output: Multiplicative inverse $a^{-1} \in GF\,(2^m)$;

begin

1: $P_0 = a^2$; $\alpha_s = P_0$;

2: $M_0 = \alpha_s * a$; $\alpha_1 = M_0$;$\alpha_m = M_0$;

3: **for each** $u_i \in U\,(2 \leq i \leq f)$ **do**

4: $k = u_i - u_{i-1}$;

5: **if**(k $> u_{\mathrm{C}}$) **then**

6: $P_0 = \alpha_m^{4^{u_C}}$, $\alpha_s = P_0$; k = k - u_{C}

7: **While**(k$> u_{\mathrm{C}}$)**do**

8: $P_0 = \alpha_s^{4^{u_C}}$, $\alpha_s = P_0$; k = k - u_{C}

9: **end while**

10: $P_0 = \alpha_s^{4^k}$, $\alpha_s = P_0$

11: **else**

12: $P_0 = \alpha_m^{4^k}$, $\alpha_s = P_0$

13: **end if**

14: **if**(k $\neq 1$) **then**

15: $M_0 = \alpha_s * \alpha_m; \alpha_m = M_0$;

16: **else**

17: $M_0 = \alpha_s * \alpha_1; \alpha_m = M_0$;

18: **end if**

19: **end for**

20: $P_0 = \alpha_m^2$; $\alpha_s = P_0$; $a^{-1} = \alpha_s$;

end

where, $f = |\log_2(\mathrm{m} - 1)| + \mathrm{H}(\mathrm{m} - 1) - 2$

The inverse of $a \in GF\,(2^{193})$ over the irreducible trinomial $R(X) = X^{193} + X^{15} + 1$ computed based on the modified-ITA is presented in Table 4.2.

Table 4.2 FF-Inverse realization of a such that $a^{-1} \in GF\left(2^{193}\right)$ using Modified-ITA

	$\gamma_{\mathbf{u_i}}(\mathbf{a})$	$\gamma_{\mathbf{u_j+u_l}}(\mathbf{a})$	Exponentiation
1	$\gamma_1(a)$		a^3
2	$\gamma_2(a)$	$\gamma_{1+1}(a)$	$(\gamma_1)^{4^1}\gamma_1 = a^{4^2-1}$
3	$\gamma_3(a)$	$\gamma_{2+1}(a)$	$(\gamma_2)^{4^1}\gamma_1 = a^{4^3-1}$
4	$\gamma_6(a)$	$\gamma_{3+3}(a)$	$(\gamma_3)^{4^3}\gamma_3 = a^{4^6-1}$
5	$\gamma_{12}(a)$	$\gamma_{6+6}(a)$	$(\gamma_6)^{4^6}\gamma_6 = a^{4^{12}-1}$
6	$\gamma_{24}(a)$	$\gamma_{12+12}(a)$	$(\gamma_{12})^{4^{12}}\gamma_{12} = a^{4^{24}-1}$
7	$\gamma_{48}(a)$	$\gamma_{24+24}(a)$	$(\gamma_{24})^{4^{24}}\gamma_{24} = a^{4^{48}-1}$
8	$\gamma_{96}(a)$	$\gamma_{48+48}(a)$	$(\gamma_{48})^{4^{48}}\gamma_{48} = a^{4^{96}-1}$

4.2.3 Proposed FF-inversion Architecture

The architecture developed for the computation of FF-Inversion over GF(2^m) using the Algorithm 4.1 is shown in Fig. 4.1. It is a two-stage pipelined architecture which computes the exponentiation using power-block followed by multiplication with FF-multiplier. The inputs to the Power-block and FF-multiplier are applied through 4:1 multiplexers M1 and M2. During every clock-cycle, a set of control signals are issued by the control unit to the registers and multiplexers. The input data is received by FF-multiplier and power-block either from external source or from the registers based on the control signals. The results are stored in registers at the end of each clock cycle and this process is repeated for f number of iterations, where $f = |\log_2(\mathrm{m}-1)| + \mathrm{H}(\mathrm{m}-1) - 2$. The functionality of the building blocks of the proposed architecture are explained in the subsequent sub-sections.

FF-Multiplier

The FF-multiplication required in the hardware realization of the proposed modified-ITA algorithm is performed by employing the bit-parallel hybrid-Karatsuba multiplier [61]. This FF-multiplier receives two inputs from the multiplexers M1 and M2 and recursively divides the operands size by half to reduce the computational complexity. The

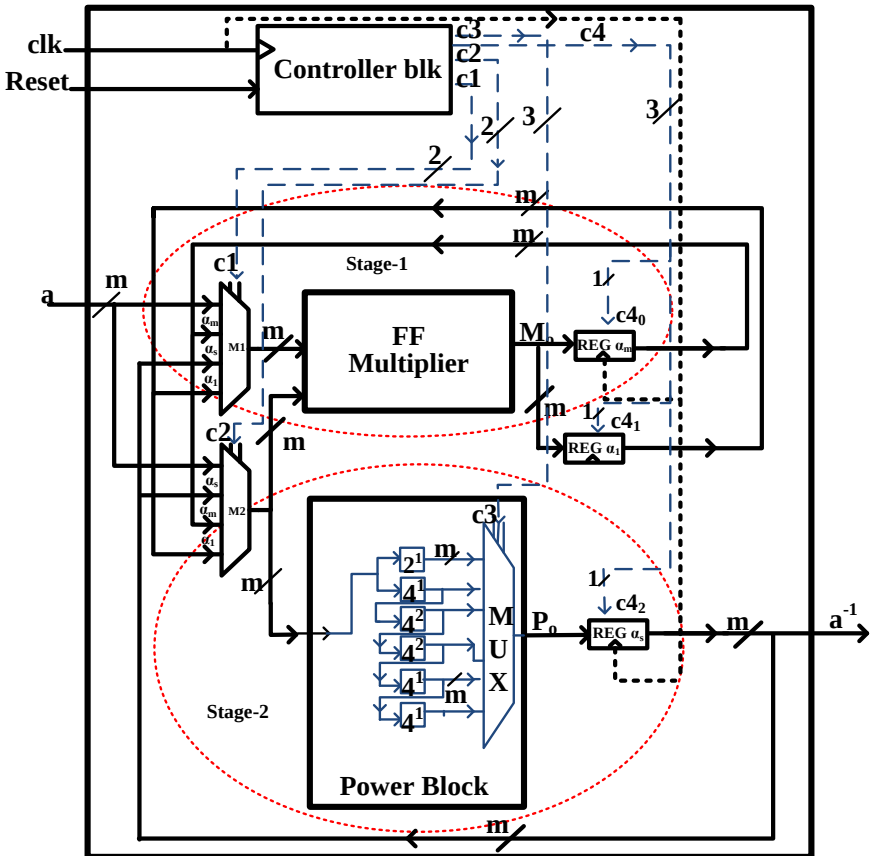

Figure 4.1 Proposed Architecture of FF-Inversion

operands are divided until a threshold (τ) is achieved [51, 62] and these are multiplied using classical-multiplication(CM) followed by performing modular-reduction operation as shown in Fig. 4.2. This Karatsuba multiplier is employed in the proposed architecture as it reduces the computational complexity to O($m^{1.58}$) compared to that of conventional multiplier (O(m^2)).

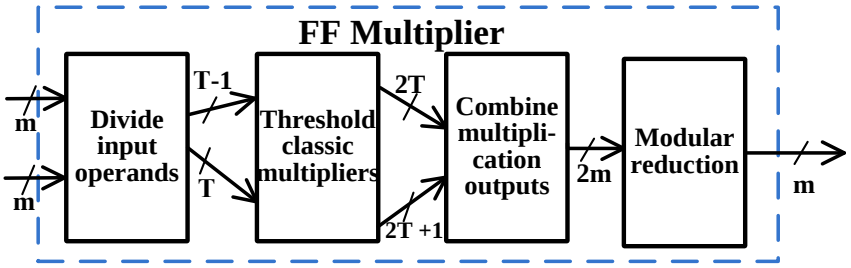

Figure 4.2 Architecture of the FF-Multiplier.

Let A(x) and B(x) be any arbitrary elements in GF(2^{m}) and can be represented as

$$A(x) = \sum_{l=0}^{m-1} a_l x^l = a_0 + a_1 x + \ldots + a_{m-2} x^{m-2} + a_{m-1} x^{m-1} \tag{4.11}$$

$$B(x) = \sum_{l=0}^{m-1} b_l x^l = b_0 + b_1 x + \ldots + b_{m-2} x^{m-2} + b_{m-1} x^{m-1} \tag{4.12}$$

$a_l, b_l \in GF(2)$ for $0 \leq l \leq m-1$. The field element product of A(x) and B(x) over GF(2^{m}) is given by

$$\begin{aligned} A \cdot B &= (a_1 x^{\mathrm{n}} + a_0)(b_1 x^{\mathrm{n}} + b_0) \\ &= a_1 b_1 x^{2\mathrm{n}} + (a_0 b_1 + a_1 b_0) x^{\mathrm{n}} + a_0 b_0 \\ &= a_1 b_1 x^{2\mathrm{n}} + [(a_1 + a_0)(b_1 + b_0) + a_1 b_1 + a_0 b_0] x^{\mathrm{n}} + a_0 b_0 \end{aligned} \tag{4.13}$$

where, $\mathrm{n} = \lfloor \mathrm{m}/2 \rfloor, A = a_1 x^{\mathrm{n}} + a_0,$ and $B = b_1 x^{\mathrm{n}} + b_0$

where, m is operand size.

Power block

The power-block consists of a 2^1 module and a set of cascaded 4^k modules as shown in Fig. 4.1. The output of the multiplexer M2 is fed as input to the power-block. The multiplexer within the power-block generates the output depending on the control signal C3. Here, the number of cascade blocks of 4^k are chosen such that the combinational delay of the cascade blocks together is equivalent to the combinational delay of the FF-multiplier to maintain optimal frequency in all the critical paths. The functionality of 2^1 and 4^k modules are explained below:

2^1 **and** 4^k : Let $R(x)$ be an m degree irreducible polynomial and $A(x)$ be any field element defined over the field GF(2^m). The polynomial representation of A and A^2 are given by the following expressions

$$A(x) = \left(a_{m-1} x^{m-1} + a_{m-2} x^{m-2} + \ldots + a_2 x + a_1 x + a_0\right) \bmod R \tag{4.14}$$

$$A^2(x) = \left(a_{m-1} x^{2m-2} + a_{m-2} x^{2m-4} + \ldots + a_2 x^4 + a_1 x^2 + a_0\right) \bmod R \tag{4.15}$$

Here, each binary coefficient of the 2^1(FF-squaring) module is realized by a set of logical XOR circuits. For the faster accomplishment of 2^1 exponentiation the reduction polynomial $R(x)$ is chosen to be National Institute of Standard and Technology (NIST)

trinomial or pentanomial. For example, the 2^1 exponentiation module in GF(2^{193}) with NIST irreducible trinomial R(x)=$x^{193} + x^{15} + 1$ is shown in the Fig. 4.3. This 2^1 module is realized based on the following assumptions [8] with R(x)=$x^m + x^g + 1$ where, m , g are odd numbers and g $< \frac{m+1}{2}$.

$$s_i = \begin{cases} a_{\frac{i}{2}} & i \text{ even}, i < g \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+\frac{m-g}{2}} + a_{\frac{i}{2}+(m-g)} & i \text{ even}, g < i < 2g \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+\frac{m-g}{2}} & i \text{ even}, i \geq 2g \\ a_{\frac{m+i}{2}} + a\frac{m+i}{2} + \frac{m-g}{2} & i \text{ odd }, i < g \\ a_{\frac{m+i}{2}} & i \text{ odd }, i \geq g \end{cases} \tag{4.16}$$

For the field element A, the 4^1 exponentiation can be computed by the following equation

$$A^4(x) = \left(a_{m-1}x^{4m-4} + a_{m-2}x^{4m-8} + \ldots + a_2x^8 + a_1x^4 + a_0\right) \bmod R \tag{4.17}$$

The logical diagrams are realized for 4^1 and 4^2 modules by recursively applying the assumptions in Eq.4.16, and are shown in the Fig. 4.4 and Fig. 4.5, respectively. It may be noted that 4^2 module can be generated directly using a series of logical XOR circuits instead of cascading two 4^1 modules, which results in reduction in combinational delay. Since the delay of 4^1 module is equal to delay of two XOR gates($2D_{xor}$), and the delay of 4^2 is equal to the delay of three XOR gates($3D_{xor}$), we can achieve 25% reduction in the delay by employing a single 4^2 module instead of two cascaded 4^1 modules.

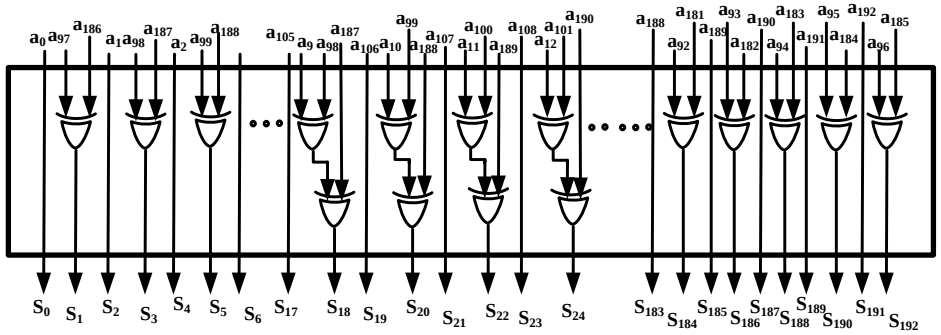

Figure 4.3 Architecture of the 2^1 module over GF(2^{193})

It is to be noted that the assumptions in Eq.(4.17), vary with m and g values. For example, realizing 2^1 in GF(2^{233}) with irreducible trinomial R(x)=$x^{233} + x^{74} + 1$ is

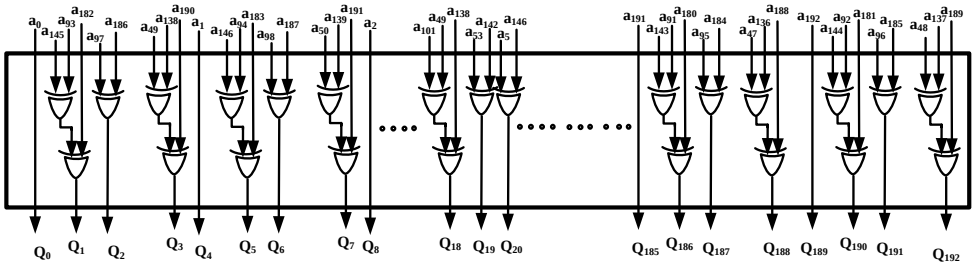

Figure 4.4 Architecture of the 4^1 module over GF(2^{193})

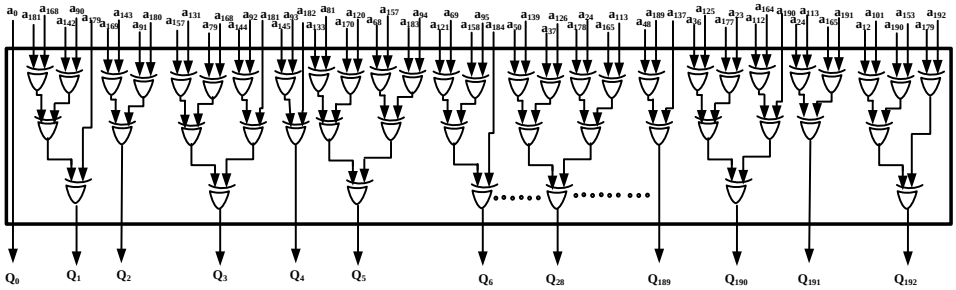

Figure 4.5 Architecture of 4^2 module over GF(2^{193})

computed by the following expression

$$s_i = \begin{cases} a_{\frac{i}{2}} + a_{\frac{i}{2}+m-\frac{g}{2}} & i \text{ even, } i < g, \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+m-g} & i \text{ even, } g \leq i < 2g \\ a_{\frac{i}{2}} & i \text{ even, } i \geq 2g, \\ a_{\frac{m+i}{2}} & i \text{ odd, } i < g \\ a_{\frac{m+i}{2}} + a_{\frac{m+i}{2}-\frac{g}{2}} & i \text{ odd, } i > g \end{cases} \tag{4.18}$$

where, m is odd, g is even and g $< \frac{m+1}{2}$.

Registers

The registers α_m, and α_s store the intermediate results of FF-multiplier and power-block, respectively. Whereas, α_1 stores the precomputation value that is generated by steps 1 and 2 of Algorithm 4.1, which is to be used in the later steps of the algorithm. The value to be stored in each register for a particular clock-cycle is controlled by the control signal C4. The register values are applied as inputs to the FF-multiplier and power-block via multiplexers M1 and M2 using the control signals C1 and C2, respectively.

4.2.4 Analytical Formulations

Hardware Complexity Analysis

The hardware complexity of the proposed architecture to compute FF-inversion is estimated in terms of d-input LUTs, where d represents the number of inputs to the LUT which depends on the technology of the target FPGA device. For a single variable logic function, zero LUTs are required. If the number of input variables of the logic function are less than or equal to d then a single LUT can configure the logic function. The number of LUTs required to realize a logic function with number of input variables greater than d can be computed using the following expression [27, 61]

$$\text{lut}(x) = \begin{cases} \left\lfloor \frac{x-d}{d-1} \right\rfloor + 2 & \text{if } x > d \text{ and } (d-1) \nmid (x-d) \\ \frac{x-d}{d-1} + 1 & \text{if } x > d \text{ and } (d-1) \mid (x-d) \end{cases} \tag{4.19}$$

As explained in Section 4.2.3, the Hybrid Karatsuba multiplier splits the m-bit operand into multiple m/2-bit operands recursively, until it reaches a threshold τ value. The experimental results for selecting the τ value for a particular d-input LUT FPGA are presented in [27, 62]. It may be noted that an m-bit hybrid Karatsuba multiplier requires three $\frac{m}{2}$ multipliers if m value is even. If m value is odd one $\frac{m-1}{2}$ multiplier and two $\frac{m+1}{2}$ bit multipliers are required. A zero bit is to be added in the MSB bit position of the m bit operand, if m value is odd.The total LUT estimate of the FF-Multiplier can be computed by recursively applying the following expressions depending on the value of m

$$\begin{aligned} \#LUT_{hkmul}(m) =& 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \\ & LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) + 2m - 1 \end{aligned} \tag{4.20}$$

$$\begin{aligned} \#LUT_{hkmul}(m) =& 2 \times LUT_{hkmul}\left(\left\lceil \frac{m+1}{2} \right\rceil\right) + \\ & LUT_{hkmul}\left(\left\lfloor \frac{m-1}{2} \right\rfloor\right) + 2m - 1 \end{aligned} \tag{4.21}$$

The expression to compute the LUT estimate for the second iteration can be derived by taking m/2 as the field order in Eq.4.20 when m is even and is given by,

$$\begin{aligned} \#LUT_{hkmul}(m) =& 2 \times (2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{4} \right\rceil\right) + LUT_{hkmul}\left(\left\lfloor \frac{m}{4} \right\rfloor\right) + m - 1) + \\ & (2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{4} \right\rceil\right) + LUT_{hkmul}\left(\left\lfloor \frac{m}{4} \right\rfloor\right) + m - 1) + 2m - 1 \end{aligned} \tag{4.22}$$

The LUT estimate of the FF-multiplier in the remaining iterations can be computed in a similar manner by evaluating Eq.4.20, with the field order value as half of that value taken for the previous iteration. This procedure is to be repeated in every iteration of the remaining iterations. The number of iterations depends upon the time required for the field order, which is divided in every iteration to reach the threshold value τ. A classical multiplier is invoked once the operand size is less than or equal to τ value. The LUT estimate of classical multiplier can be computed as

$$\mathcal{L}_{cm}(\tau) = \sum_{y=0}^{\tau-1} \left\lceil \frac{2y+1}{d-1} \right\rceil + \sum_{y=\tau}^{2\tau-2} \left\lceil \frac{4\tau-2y-3}{d-1} \right\rceil \tag{4.23}$$

In the proposed architecture, M1 & M2 multiplexers are used to apply the operands to the FF-multiplier and M3 multiplexer is used in the power block to select the appropriate exponentiation term. The generalized expression for the LUT estimate of the Multiplexers with x inputs is

$$\#LUT_{MUX} = m \times \text{lut}\,(x + \log_2 x) \tag{4.24}$$

where, $\log_2 x$ gives the number of selection lines.
The LUT estimate of the power block depends upon the 2^k or 4^k modules. The hardware complexity of 2^k or 4^k modules depends upon the binary sequence which is the outcome of logical XOR network given by Eq.(4.16). The LUT estimate of 2^k or4^k modules can be computed as

$$\#LUT_{2^k or 4^k} = \sum_{i=0}^{m-1} \text{lut}\,(b_i) \tag{4.25}$$

The total area complexity of the proposed FF-inversion architecture shown in the Fig. 4.1 can be computed by the summation of area complexity of each building block as

$$\begin{aligned} ITA_{Area} =& 2m \times \text{lut}\,(4 + \log_2 4) + m \times \text{lut}\,(8 + \log_2 8) + \frac{(m-1)}{2} + 3m \times lut(4) \\ &+ 2m \times lut(8) + \sum_{i=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \sum_{i=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) \\ &+ 2m - 1 \end{aligned} \tag{4.26}$$

where, $LUT_{hkmul}(\frac{m}{2}) = 2 \times LUT_{hkmul}(\frac{m}{4}) + LUT_{hkmul}(\frac{m}{4}) + m - 1$

Delay Complexity Analysis

The delay of the proposed two-stage pipelined FF-inversion architecture (see Fig.4.1) can be computed by estimating the stage delay and number of clock-cycles required to compute the FF-inversion for a given input. The stage delay of the pipelined architecture is the delay of the critical-path. It may be noted from the Fig. 4.1 that one stage(stage-1) of pipelined data-path contains FF-multiplier and the second stage(stage-2) contains power-block. The critical path of this pipelined architecture can be computed by determining the stage with maximum delay. The delays of multiplexer, FF-multiplier and Power-block are estimated in terms of the delay of a d-input LUT [27, 61] to compare the performance of this proposed FF-inversion architecture with that of architectures available in the literature [25, 27–31, 33, 34, 63].

FF-multiplier: The hybrid Karatsuba multiplier employed in the proposed architecture to compute FF-inversion consists of four stages namely splitting stage(dividing the operands until τ), classical-multiplier stage(multiplication of operands with τ as operand size), alignment stage(combine the outputs of classical-multiplier stage) and modular reduction stage ($(2m-1)$-bit output reduced to m-bit output). The delays of all these four stages of FF-multiplier are estimated in terms of d-input LUT as $D_{\mathrm{sp}} = log_{\mathrm{d}}(\frac{\mathrm{m}}{\tau})$, $D_{\mathrm{Th}} = log_{\mathrm{d}}2\tau$, $D_{\mathrm{cm}} = log_2(\frac{\mathrm{m}}{\tau})$, and $D_{\mathrm{mod}} = log_{\mathrm{d}}\mathrm{t}$ respectively. The delay of FF-multiplier is the sum of the delays of four stages and is given by

$$D_{\mathrm{FFmul}} = D_{\mathrm{sp}} + D_{\mathrm{Th}} + D_{\mathrm{cm}} + D_{\mathrm{mod}} \tag{4.27}$$

Multiplexer: The delay of an x:1 multiplexer with $\log_2 x$ selection lines is given by $D_{mux} = \log_{\mathrm{d}}(x + \log_2 x)$.

Power-block: The delay across power-block of the proposed FF-inversion architecture is the sum of the delays across cascaded 4^k exponentiation modules and x:1 multiplexer and is given by

$$\begin{aligned} \#D_{\mathrm{Powblk}} &= D_{pb} + D_{MUX_P} \\ &= \sum D_{4^1} + \sum D_{4^2} + \lceil \log_d(x + \log_2 x) \rceil \end{aligned} \tag{4.28}$$

where, x is the number of inputs applied to the multiplexer within power-block. Here, D_{pb} denotes the delay of the 4^k circuits comprising the total delay across D_{4^1} and D_{4^2} modules and D_{MUX_P} denotes delay of the Multiplexer within the power-block.

The stage delay of the Proposed pipelined FF-inversion architecture will be the maximum value of the delays of stage-1 and stage-2. The expressions for the delay of stage1 and stage2 are given as

$$D_{stage1} = D_{FFmul} + D_{MUX(M1)} \tag{4.29}$$

$$D_{stage2} = D_{MUX(M2)} + D_{pb} + D_{MUX_P} \tag{4.30}$$

Using Eq.(4.29) and Eq.(4.30) the stage delay of the Proposed FF-inversion architecture can be computed as

$$ITA_{delay} = Max(D_{stage1}, D_{stage2}) \tag{4.31}$$

where, ITA_{delay} varies with the order of the field.

It may be noted that K value in 4^k is chosen such that the total delay of stage-2 is less than or equal to the delay of stage-1. Therefore the delay of proposed inversion architecture is equal to delay of stage 1 and is given as

$$\mathrm{ITA}_{delay} = \log_d\left(\frac{\mathrm{m}}{\tau}\right) + \log_d 2\tau + \log_2\left(\frac{\mathrm{m}}{\tau}\right) + \log_d \mathrm{t} + \log_d(\mathrm{x} + \log_2 \mathrm{x}) \tag{4.32}$$

where, x is the number of inputs to the multiplexer, τ is threshold multiplier size.

The delay of the proposed architecture can be computed with respect to Virtex-4 and Virtex-5 FPGAs, by simply substituting the value of d (using Eq.(4.32)) as 4 and 6, respectively. For example, the delay computation of the proposed FF-inversion architecture for an irreducible trinomial (GF(2^{193})) is explained with the help of the Fig. 4.6 and is equal to 10 LUT's (10 $\times$ the d-input LUT delay of the target FPGA).

The computation time of the proposed FF-inversion architecture is

$$T_{INV} = C_{INV} \times T_P \tag{4.33}$$

where,

- C_{INV} denotes the number of clock-cycles required to compute FF-Inversion
- T_P represents the time period of the clock

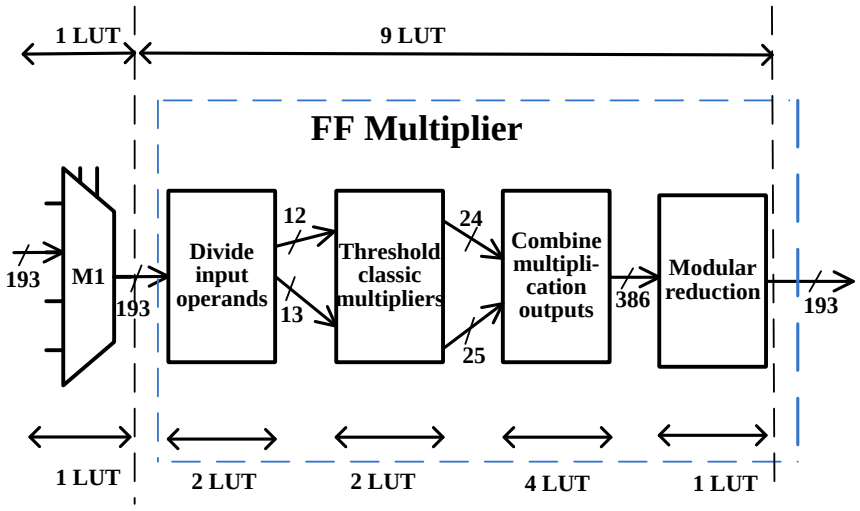

Figure 4.6 The critical-path of the proposed Architecture (GF(2^{193})).

As shown in the Algorithm 4.1, the computation of FF-inversion requires f number of iterations with each iteration computed in c number of clock-cycles. The value of c varies with each iteration depending on the exponentiation required in that particular iteration. Each iteration consists of a 4^k exponentiation and FF-multiplication. Therefore, total number of clock-cycles required for the computation of FF-inversion is given by the following expression

$$C_{INV} = C_{Quad} + M_{Inv} \tag{4.34}$$

and M_{Inv} is computed as

$$M_{Inv} = (|\log_2(\mathrm{m}-1)| + \mathrm{H}(\mathrm{m}-1) - 2) \tag{4.35}$$

where,

- C_{Quad} denotes total clock-cycles required for exponentiation
- M_{Inv} represents clock-cycles required for multiplications

The latency of the proposed architecture can be computed by using Eqs.(4.34 and 4.35) and is given as

$$ITA_{\text{latency}} = \sum_{i=2}^{f} \left\lceil \frac{u_i}{u_C} \right\rceil + (\mathrm{H}(\mathrm{m}-1) + |\log_2(\mathrm{m}-1)| - 2) + 2 \tag{4.36}$$

where, $\sum_{i=2}^{f} \left\lceil \frac{u_i}{u_C} \right\rceil$ is the number of clock cycles required for generating required exponentiation, (H(m−1)+|$\log_2$(m−1)|−2) is the clock cycles required for multiplication, 2 represents the number of clock cycles required for precomputation, f is length of addition chain, H is hamming weight of m-1, u_i is the element in the addition chain, u_C is the maximum cascade size as explained in the Algorithm 1.

It is to be observed from the Eq.(4.34) that the clock-cycles required for FF-inversion is directly proportional to number of clock cycles required for total exponentiation. Hence, we need to design power-block with suitable value of k in 4^k module such that it completes FF-inversion in minimum clock-cycles. It may also be noted that the area complexity increases with increase in the k value of 4^k module. Hence, the power-block must be designed with optimal area and time complexity by choosing suitable value of k. The area

Table 4.3 Area-time estimates of the Proposed FF-inversion Architectures for different 4^k exponentiation modules over GF(2^{193})

Exponentiation	**FPGA**	**Area** (LUT's)	**clock cycles**	**Time** (ns)	**ATP** ($\times 10^{-6}$)
4^6	virtex-4	12794	28	308	3940
4^7	virtex-4	12939	26	286	3700
4^8	virtex-4	13084	25	302	3951
4^9	virtex-4	13229	25	330	4365
4^{10}	virtex-4	13374	24	343	4589
4^6	virtex-5	8915	28	252	2246
4^7	virtex-5	9060	26	234	2120
4^8	virtex-5	9205	25	247	2273
4^9	virtex-5	9350	25	270	2524
4^{10}	virtex-5	9495	24	280	2658

and time complexity values of the proposed FF-inversion architecture over the irreducible trinomial GF(2^{193}) is estimated for various 4^k exponentiation modules by varying the cascade size(u_C) in the Algorithm 4.1 and are presented in Table 4.3. It may be observed from the Table 4.3 that the increase in value of k in 4^k module from 6 to 10 results in

decreasing the clock-cycles from 28 to 24 and increasing the area complexity from 12794 to 13374. It may be concluded that this architecture for GF(2^{193}) must be designed with $k = 7$ as 4^7 exponentiation provides minimum area-time product.

4.2.5 Implementation Results

In this section, we present the performance comparison of the proposed FF-inversion architecture and the FF-inversion architectures available in the literature [25, 27–31, 33, 34, 63]. The performance of the proposed architecture can be estimated theoretically by using the Eqs.(4.26, (4.32) and (4.36).

When targeted device is Virtex-4 FPGA, the total area and delay complexities of the proposed FF-inversion architecture can be computed using the following expressions,

$$ITA_{Area} = \frac{(31m-1)}{2} + \sum_{i=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \sum_{i=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) + 2m - 1 \quad (4.37)$$

where, $LUT_{hkmul}(\frac{m}{2}) = 2 \times LUT_{hkmul}(\frac{m}{4}) + LUT_{hkmul}(\frac{m}{4}) + m - 1$

$$\mathrm{ITA}_{delay} = \log_4\left(\frac{\mathrm{m}}{\tau}\right) + \log_4 2\tau + \log_2\left(\frac{\mathrm{m}}{\tau}\right) + \log_4 \mathrm{t} + \log_4 (4 + \log_2 4) \quad (4.38)$$

When targeted device is Virtex-5 FPGA, the total area and delay complexities of the proposed FF-inversion architecture can be computed using the following expressions

$$ITA_{Area} = \frac{(23m-1)}{2} + \sum_{i=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \sum_{i=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) + 2m - 1 \quad (4.39)$$

$$\mathrm{I}TA_{delay} = \log_6\left(\frac{\mathrm{m}}{\tau}\right) + \log_6 2\tau + \log_2\left(\frac{\mathrm{m}}{\tau}\right) + \log_6 \mathrm{t} + \log_6 (4 + \log_2 4) \quad (4.40)$$

Table 4.4 presents the performance comparison of the proposed FF-inversion architecture over GF(2^{193}) with that of others architectures available in the literature [27, 28, 30, 31, 63]. It is evident from these results that the proposed FF-inversion architecture achieves area efficiency of around 14%, 39%, 15%, and 37% compared with

Table 4.4 Performance comparison of the Proposed FF-inversion Architecture with the existing FF-inversion Architectures over GF(2^{193}).

Design	**Field**	**FPGA**	**Area** (LUTs)	**Fmax** (MHz)	**clock cycles**	**Time** (ns)	**ATP** ($\times 10^{-6}$)	% **Reduction in Area**	% **Reduction in ATP**
[27]	193	virtex-4	77.52	26	15043	335	5039	14	26.57
[28]	193	virtex-4	89.28	18	21105	201	4242	39	12.77
[30]	193	virtex-4	255.44	49	15179	192	2914	15	-21.2
[63]	193	virtex-4	-	18	20843	190	3960	37.9	6.6
Proposed	193	virtex-4	90.9	26	12939	286	3700	-	–
[27]	193	virtex-5	112.2	22	11254	196	2206	19	3.89
[28]	193	virtex-5	124.1	18	15137	145	2194	40	3.37
[31]	193	virtex-5	124.7	9	31050	64	1987	70	-6.2
[63]	193	virtex-5	-	18	14993	131	1964	39.5	-7.3
Proposed	193	virtex-5	111.1	26	9060	234	2120	–	–

FF-inversion architectures [27, 28, 30, 63], respectively, when targeted on virtex-4 FPGA. It may be be noted that the proposed architecture achieves reduction in area-time product of around 26%, 12%, and 6% when compared with [27, 28, 63], respectively. However, the proposed FF-inversion architecture achieves area efficiency of around 15% compared to the FF-inversion architecture [30] at the expense of more computation time.

Table 4.4 also presents the performance comparison of the proposed FF-inversion architecture over GF(2^{193}) with that of others architectures available in the literature [27, 28, 31, 63] when targeted device is virtex-5 FPGA. It can be observed from these results that the proposed FF-inversion architecture achieves area efficiency of around 19%, 40%, 70%, and 39% compared with FF-inversion architectures [27,28,31,63], respectively. It is also be noted that the proposed architecture achieves reduction in area-time product of around 3.8%, and 3.3% when compared with [27, 28], respectively. Since the proposed FF-inversion architecture requires more area-time product of about 6%- 8% than the architectures proposed in [31,63], this may be preferred for the area constrained applications at the expense of speed.

Table 4.5 presents the performance comparison of the proposed FF-inversion architecture over GF(2^{233}) with that of the architectures available in the literature [25, 27–

Table 4.5 Performance comparison of the Proposed FF-inversion Architecture with the existing FF-inversion Architectures over GF(2^{233}).

Design	**Field**	**FPGA**	**Area** (LUTs)	**Fmax** (MHz)	**clock cycles**	**Time** (ns)	**ATP** ($\times 10^{-6}$)	**% Reduction in Area**	**% Reduction in ATP**
[25]*QITA	233	virtex-4	26122	97.08	30	309	8071	32.9	23.62
[25]*SITA	233	virtex-4	27897	98.09	36	367	10238	37.2	39.79
[27]	233	virtex-4	21464	73.71	30	407	8737	18.4	29.44
[28]	233	virtex-4	23734	71.87	23	320	7594	26.2	18.83
[29]	233	virtex-4	19414	-	26	287.3	5577	10	-9
[30]	233	virtex-4	20988	213.84	58	271	5687	16.5	-7.7
[33]	233	virtex-4	24713	100	29	290	7166	29.1	13.98
[34]*ITA	233	virtex-4	14982	-	91	896.8	13435	-14	54.1
[34]*TITA	233	virtex-4	34947	-	82	677.9	23690	49.8	73.9
[34]*PITA	233	virtex-4	27620	-	64	642.2	17737	36.5	65.2
[63]	233	virtex-4	22917	-	23	273	6256	23.5	2
Proposed	233	virtex-4	17512	90.90	32	352	6164	-	–
[25]*QITA	233	virtex-5	20950	128.75	30	233	4881	45.5	32.7
[25]*SITA	233	virtex-5	21379	141.02	33	234	5002	46.6	34.38
[27]	233	virtex-5	13962	109.31	27	247	3453	18.3	4.95
[28]	233	virtex-5	16256	109	23	211	3430	29.8	4.31
[63]	233	virtex-5	15956	-	23	186	2967	28.5	-9.5
[20]*EEA	233	virtex-5	10990	-	47	312	3428	-3.5	4.25
Proposed	233	virtex-5	11399	111.1	32	288	3282		–

30, 33, 34, 63] when targeted device is Virtex-4 FPGA. It can be observed from these results that the proposed architecture achieves area efficiency of around 32%, 37%, 18%, 26%, 37%, 10%, 16%, 29%, 49% and 36% when compared with the FF-inversion architectures [25]*QITA, [25]*SITA, [27–30,33,34], [34]*TITA, [34]*PITA, respectively. It is to be noted that the proposed architecture achieves reduction in area-time product of around 23%,39%,29%,18%,13%,54%,73% and 65% when compared with the FF-inversion architectures [25]*QITA, [25]*SITA, [27, 28, 33], [34]*ITA, [34]*TITA, [34]*PITA, respectively. It may be observed that the proposed FF-inversion architecture requires less area-time product than FF-inversion architecture [34]*ITA while consuming more hardware than that of the architecture [34]*ITA. However, the proposed architecture requires less area of around 10%-16% and more area-time product of around 7%-9% when compared with

Table 4.6 Performance comparison of the Proposed FF-inversion Architecture with the existing FF-inversion Architectures over GF(2^{409}).

Design	**Field**	**FPGA**	**Area** (LUTs)	**Fmax** (MHz)	**clock cycles**	**Time** (ns)	**ATP** ($\times 10^{-6}$)	% **Reduction in Area**	% **Reduction in ATP**
[25]*SITA	409	virtex-4	64970	77.51	40	516	33524	34.5	30.90
[25]*QITA	409	virtex-4	60644	63.29	32	505.6	30661	29.9	24.44
[34]*ITA	409	virtex-4	19934	-	221	2254	44931	-53.1	48
[34]*TITA	409	virtex-4	54114	-	141	1765	95511	21	75.7
[34]*PITA	409	virtex-4	36462	-	161	1706	62204	-14.2	62.7
Proposed	409	virtex-4	42545	82.64	45	544.5	23165		–
[25]*SITA	409	virtex-5	47785	116.8	40	342.2	16258	36.9	17.42
[25]*QITA	409	virtex-5	44948	108.69	35	322	14473	32.9	5.16
Proposed	409	virtex-5	30136	101.01	45	445.5	13425		–

the architectures [29, 30].

Table 4.5 also presents the performance comparison of the proposed FF-inversion architecture over GF(2^{233}) with that of the architectures available in the literature [20, 25, 27, 28, 63] when targeted device is Virtex-5 FPGA. It can be observed from these results that the proposed architecture achieves area efficiency of around 45%, 46%, 18%, 29%, and 28% when compared with the FF-inversion architectures[[25]*QITA, [25]*SITA, [27,28,63], respectively. It is to be noted that the proposed architecture achieves reduction in area-time product of around 32%,34%,4%, 4% and 4% when compared with the FF-inversion architectures [25]*QITA, [25]*SITA, [25, 28], [20]*EEA, respectively. It may be observed that the proposed FF-inversion architecture achieves area efficiency of around 28% than FF-inversion architecture [63] at the expense of more area-time product of around 9%.

Table 4.6 presents the performance comparison of the proposed FF-inversion architecture over the GF(2^{409}) with that of the architectures available in the literature [25, 34] when targeted device is Virtex-4 FPGA. It can be observed from these results that the proposed architecture achieves area efficiency of around 29.9%,34.5% and 21% when compared with the FF-inversion architectures [25]*QITA, [25]*SITA, [34]*TITA, respectively. It is to be noted that the proposed architecture achieves reduction in area-time product

of around 24.44%,30.9%,48%, 75.7 and 62.7% when compared with the FF-inversion architectures [25]*QITA, [25]*SITA, [34]*ITA, [34]*TITA, [34]*PITA, respectively. It may also be noted that the proposed FF-inversion architecture achieves around 62% reduction in area-time product than [25]*PITA at the expense of increase in the area around 14%. When compared with [34]*ITA, the proposed architecture achieves reduction in area-time product of around 48% at the expense of around 53% increase in area.

Table 4.6 also presents the performance comparison of the proposed FF-inversion architecture over the GF(2^{409}) with that of the architecture [25] when targeted device is Virtex-5 FPGA. It can be observed from these results that the proposed architecture achieves area efficiency of around 36.9%, and 32.9% when compared with the FF-inversion architectures [25]*QITA, [25]*SITA, respectively. It is also to be noted that the proposed architecture achieves reduction in area-time product of around 17.42%, and 5.16% when compared with the FF-inversion architectures [25]*QITA, [25]*SITA, respectively.

4.3 Conclusions

In this chapter, the conventional Itoh-Tsujii algorithm is modified to reduce computation time by employing 4^k exponentiation modules rather than 2^k exponentiation modules. A two-stage FF-inversion architecture is developed for this modified Itoh-Tsujii algorithm. Further, the formulations for area and delay are presented and evaluated for the field orders m=193, 233, 409. The implementation results show that the proposed FF-inversion architecture is area and area-time efficient compared to the similar FF-inversion architectures in the literature. Hence, the proposed FF-inversion architecture is profound to be suitable for area-time efficient ECC applications. However, the performance of the ECC design is dominated by the performance of the point-multiplication operation. Hence it is desirable to design area-time efficient point multiplication architecture suitable for ECC applications. The next chapter presents the design of efficient point multiplication architecture suitable for ECC applications.

Chapter 5

Low Area-time Complexity Point Multiplication Architecture over GF(2^m) using Polynomial Basis

In this chapter, we present a point-multiplication architecture developed for the proposed modified Montgomery ladder algorithm targeting ECC applications. A digit-serial multiplier is used to implement FF-multiplication in the realization of proposed point multiplication algorithm. The area and time complexities of the proposed point multiplication (PM) architecture are computed for irreducible trinomial GF(2^{233}) and irreducible pentanomial GF(2^{163}). The proposed architecture is modeled using verilog HDL and simulated to verify the functionality using Xilinx Vivado tools. The HDL netlist is synthesized targeting Xilinx Virtex-5 FPGA and implemented to compare with the similar point multiplication architectures available in the literature.

5.1 Introduction

Over the last few years, technological advances in the implementation of smart sensors, processing elements, and communication services in resource-constrained devices have enabled the rapid growth and emergence of evolving technologies like Internet-of-Things (IoT) and Wireless-sensor-networks (WSNs) [64]. These technologies elevated the need for data security services like data confidentiality, non-repudiation, digital signature, and data integrity, considering the number of devices and the secure information they carry. Public-Key-Cryptography is proven to suffice the need of data security and

is based on a presumed hard problem, such as elliptic-curve-discrete-logarithm(ECDL), integer-factorization, and discrete-logarithm. The Elliptic-Curve-Cryptography (ECC) suggested by Koblitz [3] and Miller [46], has captivated considerable attention in recent years owing to its high security per bit ratio and small key size. The performance of ECC in these resource-constrained devices relies on the performance of the point multiplication operation. The efficiency of point multiplication relies on the point multiplication algorithm and underlying finite field operations namely, FF-multiplication, and FF-inversion. A modified Itoh-Tsujii algorithm is presented in the previous chapter to improve the computation time of FF-inversion. So it is required to develop an area efficient and high speed FF-multiplier architecture for the realization of point multiplication algorithm. The performance of point multiplication is also dominated by type of irreducible polynomial employed for the realizing the architecture. The NIST trinomials and pentanomials offer less area-time complexities and are suitable for fast ECC applications. Hence, it is recommended to design high performance point multiplication architectures using irreducible trinomials and pentanomials.

Many architectures are proposed in the literature to design high-speed and low complexity ECC [5, 6, 47–58]. Most of these works aimed to reduce computation time by minimizing the latency for computation of point multiplication.

In this work, firstly, we have presented a digit-serial multiplier and its corresponding area and delay formulations. Secondly, we have derived modified Montgomery point multiplication algorithm based on the classic Montgomery point multiplication algorithms [10] available in the literature. Subsequently, the architecture for the proposed modified Montgomery point multiplication algorithm is developed. Finally, the area and delay formulations of the proposed architecture are estimated and the performance is compared with the existing point multiplication architectures in the literature.

5.2 Digit-serial Multiplier over GF(2^m)

The hardware realization of the point multiplication algorithm requires FF-multiplier. The performance of point multiplication architecture depends on the efficiency of FF-multiplication architecture. In general, GF(2^m) multipliers can be realized using digit-

serial, bit-serial, and bit-parallel multiplier architectures [65]. The bit-serial multiplier processes a single-bit of m input data for each clock-cycle and completes multiplication in m iterations. The digit-serial multiplier processes a set of D bits per iteration reducing the latency from m to m/D cycles. However, it is observed with the increment of D value the area and time complexities increase.

In this section, we present the design and performance analysis of the proposed digit- serial multiplier developed for a MSD-first digit-serial multiplication algorithm [66]. First, the design and preliminaries of the proposed digit-serial multiplier are presented. Followed by, the architecture developed for the realization of FF-multiplication is discussed. Subsequently, the area and delay formulations are derived to compare with the similar digit-serial architectures available in the literature.

5.2.1 Mathematical Formulations

An irreducible polynomial R(x) of degree m defines the binary field GF(2^{m}) and is given by the following expression [2],

$$R(x) = x^m + r_{m-1}x^{m-1} + r_{m-2}x^{m-2} + \ldots + r_1x + 1 \tag{5.1}$$

where, $r_i \in GF(2)$ for $1 \leq i \leq m$. If $\beta \in$ GF (2^m) is assumed to be the root of irreducible polynomial $R(x)$ over GF(2), then it follows $R(\beta) = 0$, and the vector $(1, \beta, \beta^2, \beta^3, \ldots\ldots\ldots \beta^{m-1})$ constitutes the polynomial basis [2].

Let A(x) and B(x) be any two arbitrary elements in GF(2^{m}) and can be represented in polynomial basis as

$$A(x) = \sum_{l=0}^{m-1} a_l x^l = a_0 + a_1x + \ldots + a_{m-2}x^{m-2} + a_{m-1}x^{m-1} \tag{5.2}$$

$$B(x) = \sum_{l=0}^{m-1} b_l x^l = b_0 + b_1x + \ldots + b_{m-2}x^{m-2} + b_{m-1}x^{m-1} \tag{5.3}$$

$a_l, b_l \in GF(2)$ for $1 \leq l \leq m-1$

Then the product of polynomials A(x) and B(x) over GF(2^{m}) is given by

$$P(x) = A(x)B(x) \bmod R(x) \tag{5.4}$$

The binary set of R, A and B over GF(2^{m}) may be defined as, R=$(1, r_1, \ldots r_{m-2}, r_{m-1})$, A=$(a_0, a_1, \ldots, a_{m-2}, a_{m-1})$, and B=$(b_0, b_1, \ldots, b_{m-2}, b_{m-1})$ respectively.

5.2.2 MSD-first Digit-serial Multiplication Algorithm

Let A and B be any two elements of order m defined over GF(2^m). Let R(x) be the irreducible polynomial of order m and P(x) be the accumulator for partial products. The Algorithm 5.1 presents MSD first digit-serial multiplication algorithm [66] to be performed on the operands A and B. It may be noted from this algorithm that each iteration involves the multiplication of operand $A(x)$ multiplied by D bits of operand $B(x)$ followed by modulo reduction by $R(x)$. The generated partial product is accumulated in the register $P(x)$. For each iteration the partial product $P(x)$ is multiplied by the constant x^D followed by XOR operation with new partial product. This process is repeated for m/D iterations and the final product is accumulated into the register $P(x)$ after m/D iterations.

Algorithm 5.1 MSD-first digit-serial multiplication Algorithm [66]

Input: A and B are arbitrary elements in GF (2^m)

Output: $P = AB \bmod R(x)$

1. $P = 0$
2. $B = B_0 + B_1 x^D + \cdots + B_{s-1} x^{D(s-1)}$, where $B_l = \sum_{j=0}^{D-1} b_{lD+j} x^j$
3. For $l = s - 1$ to 0
4. $P = Px^D \bmod R(x)$
5. $P = P + AB_l \bmod R(x)$
6. end for

5.2.3 Proposed Digit-serial Multiplier

The digit-serial multiplier designed to perform FF-multiplication operation on A and B operands of m-bit size is shown in the Fig. 5.1. The functionality of each building block of digit-serial multiplier(see Fig. 5.1) are explained in the following sub sections.

Proposed Bit-Parallel Multiplier

The performance of digit-serial multiplier can be improved by improving the performance of bit parallel multiplier. Hence, we present the bit parallel multiplier architecture developed by modifying the architecture [67] to improve the area-time efficiency (see

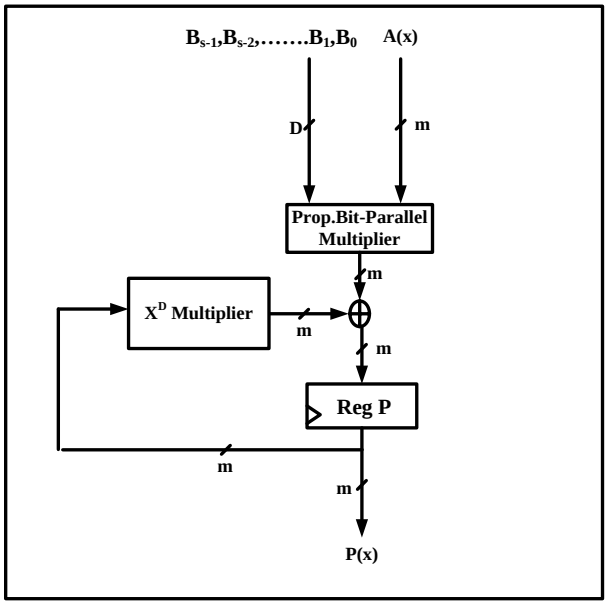

Figure 5.1 Digit-serial Multiplier

Fig. 5.2). The bit-parallel multiplication of a $m-1$ degree polynomial $A(x)$ and a (D-1) degree polynomial $K(x)$ is given by the following expression

$$\begin{aligned} K(x)A(x) \bmod R(x) =& x^{D-1}k_{D-1}A(x) \bmod R(x) \\ &+ x^{D-2}k_{D-2}A(x) \bmod R(x) \\ &+ \cdots + xk_1A(x) \bmod R(x) + A(x)k_0 \bmod R(x) \end{aligned} \tag{5.5}$$

Each element of the multiplication product ($x^l k_l A(x) \bmod R(x)$) is realized in three steps. First, a shift left operation of l times is performed on $A(x)$, followed by the reduction over $R(x)$. In the second step, logical NAND operation is performed on the resultant SAR module with the respective k_l bit in parallel. Finally, the summation is performed on the partial products generated in step 2 by logical XOR operation. Since the area, and time complexities of NAND gate are less than that of AND gate, we have employed NAND gates to realize the partial products. Shift and reduction(SAR) block is used to realize the $xA(x) \bmod R(x)$ operation and is explained in detail in the following subsection.

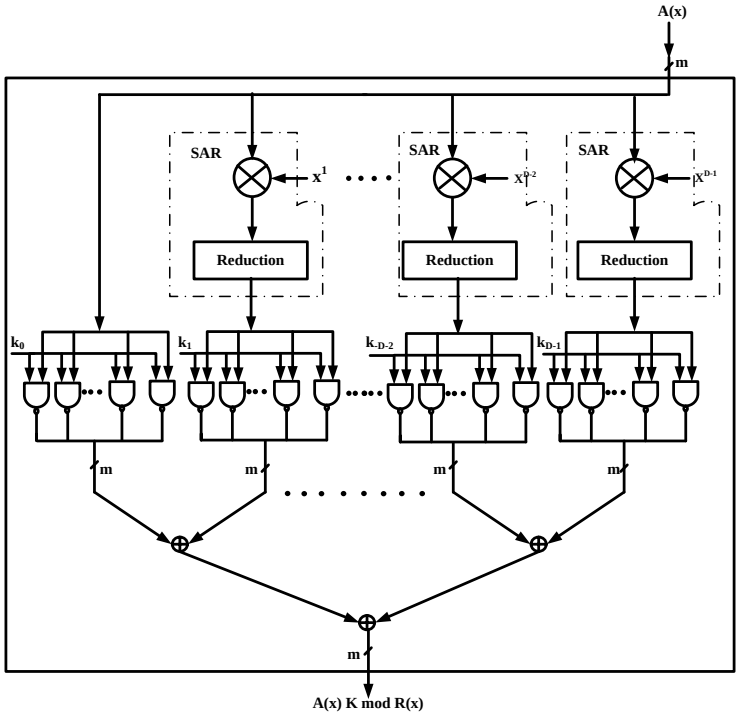

Figure 5.2 Proposed Bit-Parallel Multiplier

Shift and Reduction Module

The hardware realization of SAR module varies based on the form of irreducible polynomial. For example, consider an m degree general irreducible polynomial $R(x)$

$$R(x) = x^m + r_{m-1}x^{m-1} + \ldots + r_1 x + 1 \tag{5.6}$$

and field element A(x), the SAR block can be realized by the following expression

$$\begin{aligned} A_{l+1}(x) &= (A_l(x) \times x) \bmod R(x) \\ &= (a_0 x + a_1 x^2 + \ldots + a_{m-1}x^m) \bmod (1 + r_1 x + \ldots r_{m-1}x^{m-1} + x^m) \end{aligned} \tag{5.7}$$

If we assume x is the root of the Eq.(5.6) then we have $R(x)$=0, and it follows $x^m = r_{m-1}x^{m-1} + \ldots + r_1 x + 1$. Modulo reduction is performed by substituting x^m in Eq.(5.7)

$$A_{l+1}(x) = (a_0 x + a_1 x^2 + \ldots + a_{m-1}(r_{m-1}x^{m-1} + \ldots + r_1 x + 1)) \tag{5.8}$$

Finally rearranging the coefficients in the increasing degree of x we have

$$A_{l+1}(x) = a_{m-1} + (a_{m-1}r_1 + a_0)\, x + (a_{m-1}r_2 + a_1)\, x^2 + \ldots + (a_{m-1}r_{m-1} + a_{m-2})\, x^{m-1} \tag{5.9}$$

It can be noted that each logical AND, and logical XOR gates can be replaced by logical NAND, and logical XNOR gates respectively keeping the final outcome intact.

$$A_{l+1}(x) = a_{m-1} + (\overline{a_{m-1}r_1} \odot a_0)\, x + (\overline{a_{m-1}r_2} \odot a_1)\, x^2 + \ldots + (\overline{a_{m-1}r_{m-1}} \odot a_{m-2})\, x^{m-1} \tag{5.10}$$

where, $\odot$ is logical XNOR operation. For example, let $a_{m-1} = 1, r_1 = 0$, and $a_0 = 1$, then AND operation over a_{m-1} and r_1 will be 0. When this result is XORed with 1, we get 1 as final outcome. Similarly, NAND operation over a_{m-1} and r_1 will be 1, and XNOR operation of the result with 1 will be 1.

Fig. 5.3 shows the hardware realization of SAR module for general irreducible polynomial.

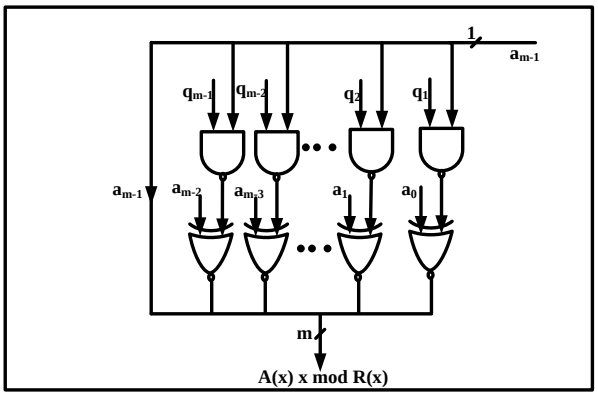

Figure 5.3 Shift and Reduction Module for general irreducible polynomials

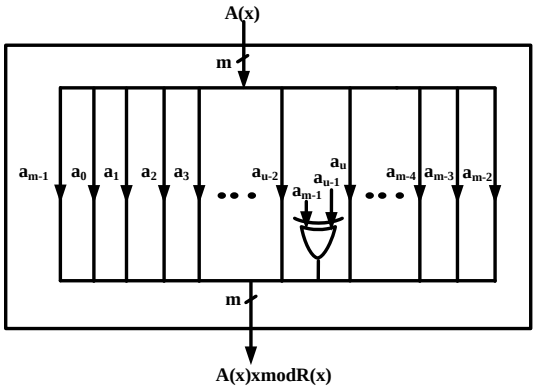

Figure 5.4 Shift and Reduction Module for Trinomials

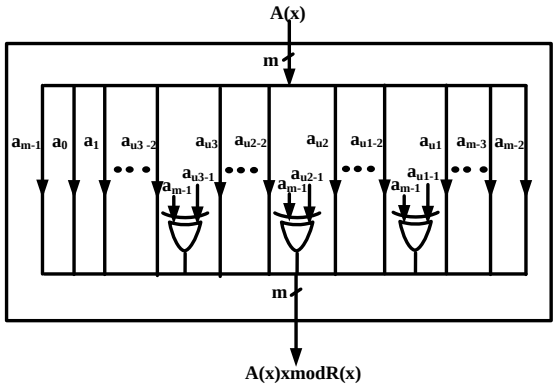

Figure 5.5 Shift and Reduction Module for Pentanomials

The hardware realization of SAR module for an irreducible trinomial and pentanomial are shown in Fig. 5.4 and Fig 5.5, respectively. The computations carried out by each SAR block for an irreducible trinomial $R(x) = x^m + x^u + 1, (0 < u < m)$ is given by

$$\begin{aligned} A_{l+1}(x) = xA_l(x) &= x\sum_{j=0}^{m-1} a_u^{(l)} x^j \\ &= a_{m-1}^{(l)} + \sum_{\substack{j=1 \\ j\neq u}}^{m-1} a_{j-1}^{(l)} x^j + \left(a_{u-1}^{(l)} + a_{m-1}^{(l)}\right) x^u \end{aligned} \tag{5.11}$$

where, A_{l+1} and A_l represent the input and its corresponding output of each SAR module, respectively. Each SAR module is realized by a single XOR gate as shown in Fig. 5.4. Similarly, the computations performed by SAR block for an irreducible pentanomial $R(x) = x^m + x^{u1} + x^{u2} + x^{u3} + 1, (0 < u3 < u2 < u1 < m)$ is given by

$$\begin{aligned} A_{l+1}(x) = xA_l(x) &= x\sum_{j=0}^{m-1} a_j^{(l)} x^j \\ &= a_{m-1}^{(l)} + \sum_{\substack{j=1 \\ j\neq u1,u2,u3}}^{m-1} a_{j-1}^{(i)} x^j + \left(a_{u1-1}^{(l)} + a_{m-1}^{(l)}\right) x^{u1} + \left(a_{u2-1}^{(l)} + a_{m-1}^{(l)}\right) x^{u2} \\ &+ \left(a_{u3-1}^{(l)} + a_{m-1}^{(l)}\right) x^{u3} \end{aligned} \tag{5.12}$$

The hardware realization each SAR module requires three XOR gates as shown in Fig. 5.5.

For example: Design of bit parallel multiplier in GF(2^5) based on the irreducible trinomial $R(x) = x^5 + x^2 + 1$. Let A(x) and K(x) are two arbitrary elements in GF(2^5) and are given as

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \tag{5.13}$$

$$K(x) = k_0 + k_1 x + k_2 x^2 + k_3 x^3 + k_4 x^4 \tag{5.14}$$

$a_l, k_l \in GF(2)$ for $1 \leq l \leq 4$

The field product of the elements A(x), and K(x) over GF(2^5) is given as

$$\begin{aligned} K(x)A(x) \bmod R(x) =& k_4 x^4 A(x) \bmod R(x) \\ &+ k_3 x^3 A(x) \bmod R(x) + k_2 x^2 A(x) \bmod R(x) \\ &+ k_1 x A(x) \bmod R(x) + A(x) k_0 \bmod R(x) \end{aligned} \tag{5.15}$$

The hardware realization of each element in Eq.(5.15) requires shift and reduction module

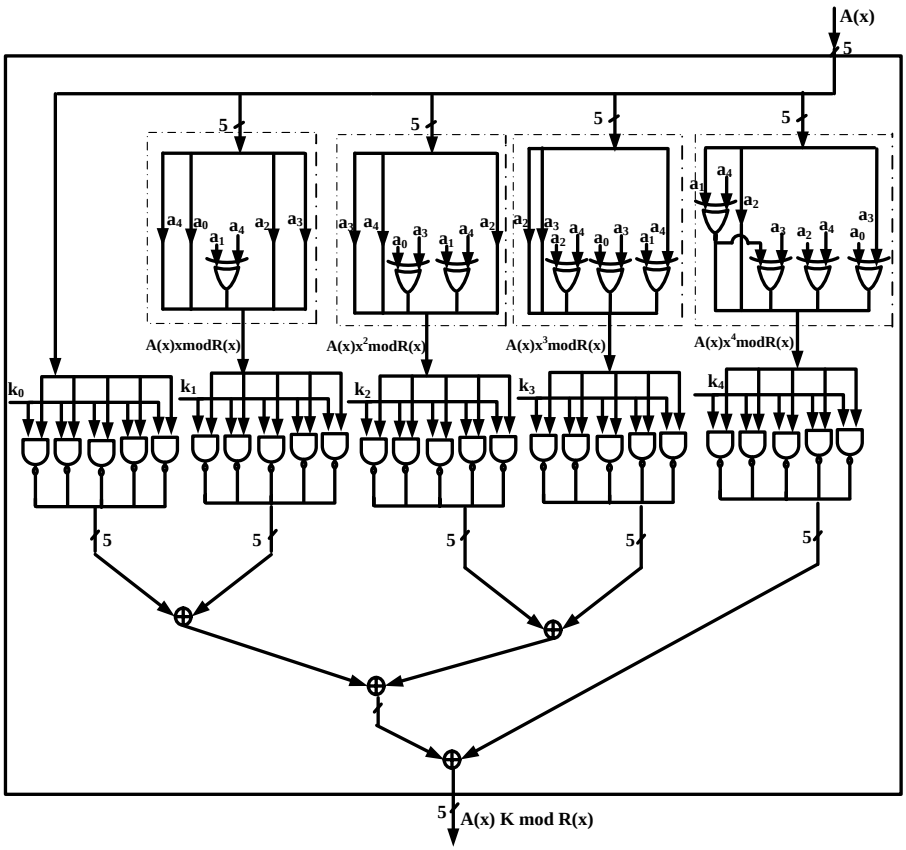

Figure 5.6 An example : Bit parallel multiplier over GF(2^5) based on the irreducible trinomial $R(x) = x^5 + x^2 + 1$.

followed by multiplier. Since the reduction polynomial is trinomial, the SAR modules are

realized based on the Eq.(5.11) and are given by the following expressions,

$$\begin{aligned} xA(x) \bmod R(x) =& x(a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4) \bmod (x^5 + x^2 + 1) \\ &= (a_0x + a_1x^2 + a_2x^3 + a_3x^4 + a_4x^5) \bmod (x^5 + x^2 + 1) \\ &= (a_0x + a_1x^2 + a_2x^3 + a_3x^4 + a_4(x^2 + 1)) \\ &= a_4 + a_0x + (a_1 + a_4)x^2 + a_2x^3 + a_3x^4 \end{aligned} \tag{5.16}$$

$$x^2A(x) \bmod R(x) = a_3 + a_4x + (a_0 + a_3)x^2(+a_1 + a_4x^3) + a_2x^4 \tag{5.17}$$

$$x^3A(x) \bmod R(x) = a_2 + a_3x + (a_2 + a_4)x^2(+a_0 + a_3x^3) + (a_1 + a_4)x^4 \tag{5.18}$$

$$x^4A(x) \bmod R(x) = (a_1 + a_4) + a_2x + ((a_1 + a_4) + a_3)x^2(+a_2 + a_4x^3) + (a_0 + a_3)x^4 \tag{5.19}$$

Fig. 5.6 shows the architecture of bit parallel multiplier (GF(2^5)) developed by realizing Eq.(5.16), Eq.(5.17), Eq.(5.18) and Eq.(5.19). It is observed from these equations that each reduction operation costs a series of logical XOR gates and it is equal to number of shift operations for the case of irreducible trinomials. In this case, a total of 10 XOR gates are used to realize the SAR modules. If D is the operand size, it will costs a total of D(D-1)/2 XOR gates to realize SAR modules in GF(2^D) over the irreducible trinomial $R(x) = x^D + x^g + 1$.

The output of each SAR module is multiplied with respective bit of operand K in parallel satisfying the Eq.(5.15), followed by XOR operation on the resultant partial products. If m and D are the operand sizes of elements $A(x)$ and $K(x)$, then it takes mD number of NAND gates to realize all the partial products and m(D-1) number of XOR gates to realize the final outcome($K(x)A(x)$ mod $R(x)$). The hardware complexity of this example (see Fig. 5.6) requires 25 NAND gates for realization of partial products and 20 XOR gates for XOR tree.

x^D Multiplier

x^D multiplier is used to multiply the partial product generated at each iteration with the constant x^D. This operation is realized by using SAR module which performs left shift operations on the partial product by D number of bits followed by mod R(x)

operation. The hardware realization of x^D multiplier for the case of irreducible trinomials is shown in the Fig. 5.7. It may be noted that the x^D Multiplier requires only D number of XOR gates.

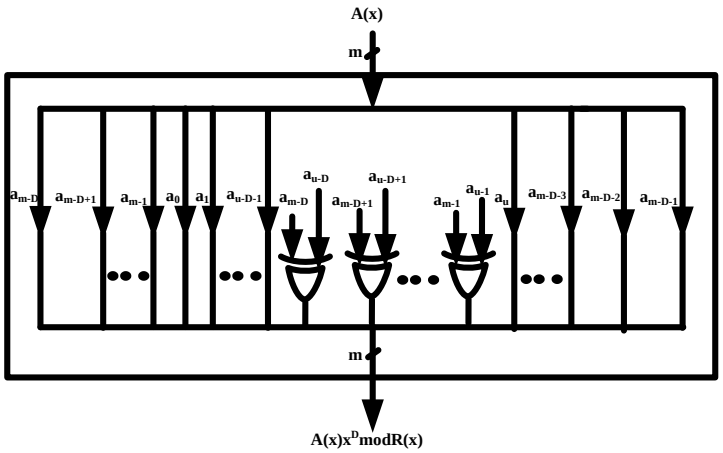

Figure 5.7 x^D Multiplier

5.2.4 Analytical Results

This section presents the analysis of area and time complexities of the digit-serial multiplier. The area complexity of the digit-serial multiplier (see Fig. 5.1) is computed as the sum of area complexities of bit-parallel multiplier, m-bit adder, x^D multiplier, and m-bit register. The area complexity of this digit-serial multiplier is computed in terms of the area of 2-i/p NAND gates, 2-i/p XOR gates, 2-i/p XNOR gates, and D-flipflops where, i/p: input. The time complexity is analyzed by assuming T_{NA}, T_X, T_{NX}, and T_D, as the delays of a 2-i/p NAND gate, a 2-i/p XOR gate, a 2-i/p XNOR gate, and a D FF respectively.

The bit-parallel multiplier requires a total of $(D-1)$ SAR modules, a network of NAND gates and XOR gates. Each SAR module consists of a series of NAND gates and XOR/XNOR gates. The number of NAND and XOR/XNOR gates varies based on the type of irreducible polynomial used, as explained in the sec 4.2.1. The bit parallel multiplier for a general irreducible polynomial(see Fig. 5.2) requires $(m-1)$ NAND gates and $(m-1)$ XNOR gates to implement the SAR modules. In addition, it requires a total of (mD) NAND gates and (m$(D-1)$) XOR gates for implementing NAND and

XOR networks, respectively. Similarly, the area complexities of bit parallel multiplier are computed for an irreducible trinomials and pentanomials and presented in Table 5.1.

The other building blocks of digit-serial multiplier (see Fig. 5.1), m-bit adder, x^D multiplier, and m-bit register for an irreducible polynomial, requires m XOR gates, D XOR gates, and m D-FFs, respectively. The area complexity of the digit-serial multiplier for general irreducible polynomial is presented in Table 5.2. The area complexities of digit-serial multiplier for irreducible trinomials and pentanomials are computed by employing the SAR modules (see Fig. 5.4 & Fig. 5.5) and x^D modules (see Fig. 5.7) and presented in the Table 5.2.

Table 5.1 Area and time complexities of Proposed Bit-Parallel Multiplier

F(x)	NAND	XNOR/XOR*	Latency	CriticalpathDelay
General	(m-1)(D-1)+mD	(m-1)(D-1)+m(D-1)	1	$(\lceil log_2(D) \rceil)T_X + T_{NX} + 2T_{NA}$
Trinomial	mD	D(D-1)/2+m(D-1)	1	$(\lceil log_2(D) \rceil + 1)T_X + T_{NA}$
Pentanomial	mD	3D(D-1)/2+m(D-1)	1	$(\lceil log_2(D) \rceil + 1)T_X + T_{NA}$

* XNOR/XOR have same area & time complexities

Since the area and time complexities of the digit-serial multiplier depends on the digit size D, these complexities are computed for various values of D (Virtex-5 XC5VLX110) and presented in Table 5.3 for a specific field GF(2^{163}). It may be noted from the Table 5.3 that the increase of digit size from 6 to 82 results increase in the area complexity from 4280 LUTs to 68546 LUTs. It may also be noted that the computation time decreases from 7.84 us to 0.86 us with the increase in digit size from 6 to 82. Hence, the appropriate value of D must be selected based on the target area and time complexities of a specific application.

Table 5.4 presents the area and time complexities comparison of the proposed digit-serial multiplier with the similar digit-serial multipliers available in the literature for the case of irreducible trinomials. The area complexities for the architectures considered for comparison in the Table 5.4 are computed for m=233 and presented in Table 5.5. The total area column of the Table 5.5 indicates that the area complexity of the proposed architecture is less than that of other architectures considered for comparison.

Table 5.2 Area and time complexities of Proposed Digit-serial Multiplier

F(x)	XOR/XNOR	NAND	Register	Latency	Criticalpath
General	m+2(m-1)(D-1)+m(D-1)	mD+2(m-1)(D-1)	m	(m/D)	$(\lceil log_2(D) \rceil + 1)T_X + T_{NX} + 2T_{NA} + T_D$
Trinomial	m+D+D(D-1)/2+m(D-1)	mD	m	(m/D)	$(\lceil log_2(D) \rceil + 2)T_X + T_{NA} + T_D$
Pentanomial	m+3D+3D(D-1)/2+m(D-1)	mD	m	(m/D)	$(\lceil log_2(D) \rceil + 2)T_X + T_{NA} + T_D$

* XNOR/XOR have same area & time complexities

Table 5.3 Digit-serial Multiplication over GF(2^{163}).

Digit size	clock cycles	Area (LUTs)	Critical-path (ns)	Delay (ns)
6	28	4280	0.28	7.84
8	21	5438	0.3	6.3
11	15	7111	0.31	6.3
15	11	9054	0.33	3.63
24	7	15742	0.36	2.12
33	5	22230	0.38	1.9
55	3	40754	0.41	1.23
82	2	68546	0.43	0.86

5.3 Low Area-time complexity Point Multiplication Architecture over GF(2^m)

In this section, the design and performance analysis of the proposed point multiplication architecture developed for the realization of modified Montgomery algorithm is presented. First, the mathematical preliminaries for the proposed modified Montgomery point algorithm are presented. Followed by, the design of architecture to realize the proposed algorithm and its corresponding area and time complexities are presented. Finally, the comparison of implementation results with the similar point multiplication architectures available in the literature are presented.

5.3.1 Mathematical Formulations

An elliptic curve E over GF(2^m) is defined by the following expression [2]

$$E : y^2 + x \cdot y = x^3 + a \cdot x^2 + b \tag{5.20}$$

where a and $b \in GF(2^m), b \neq 0$. A point on the elliptic curve is represented by pair of elements $x, y \in GF(2^m)$ and the point $P = (x, y)$ fulfills the chord and tangent

Table 5.4 Area and time complexities comparison of Digit-serial multipliers over GF(2^m)

Design	AND	XOR/XNOR	NAND	MUX	Register	Latency	Critical path
Song [41]	Dm	Dm+3D-2	0	m	3m+D-1	(m/D) +2	$(\lceil log_2(D) \rceil + 1)T_X + T_A + T_D + T_M$
Song [41]	mD	Dm+3D	0	0	2m+D	(m/D) +2	$(\lceil log_2(D) \rceil + 2)T_X + T_A + T_D$
Kim [38]	m(2D+1)	2mD	0	2m	10m+(m/D)	3(m/D)	$D(T_X + T_A + T_M) + T_M$
Tang [42]	Dm	Dm+D(D+1)/2	0	0	m	(m/D)-1	$(\lceil log_2(D) \rceil + 2)T_X + T_A + T_D$
Kumar [37]	mD	m(D+1)+3D-3	0	m	4m+D-2	(m/D) +2	$(\lceil log_2(D/2) \rceil + 1)T_X + T_A + T_D + T_M$
Proposed	0	m+D+D(D-1)/2 +m(D-1)	mD	0	m	(m/D)	$(\lceil log_2(D) \rceil + 2)T_X + T_{NA} + T_D$

Table 5.5 Area and time complexities comparison of Digit-serial Multipliers over GF(2^{233})

Design	AND	XOR/XNOR	NAND	MUX	Register	Total area (# transistors)	% Reduction in area
Song [41]	54522	110424	-	2796	22110	189852	14
Song [41]	54522	110448	-	-	15150	180120	10
Kim [38]	110442	218088	-	5592	70079	293759	44
Tang [42]	54522	118404	-	-	6990	179916	10
Kumar [37]	54522	113208	-	2796	29070	199596	18
Proposed	-	118404	36348	-	6990	161742	-

laws. Projective-coordinate system (P (X, Y, Z)), and affine-coordinate system(P(x, y)) are the two widely used coordinate systems to represent the point P on the elliptic curve.

Let P be a base point on the curve E and k is a positive integer, then the product kP is another point and is called as point-multiplication.

$$Q = kP = P + P + P + ... + P \tag{5.21}$$

where, $k \in [0, m-1]$.

Point-addition(PA) and point-doubling(PD) are the two curve operations used to achieve point-multiplication. If $P1(x1, y1)$, $P2(x2, y2)$, and $P3 = (x3, y3)$ are three points on the curve E, with $P3 = (x3, y3) = P1 + P2$. Then $P3$ is called point-addition of the points $P1$, and $P2$. When $P1 = P2$, the curve operation is called point-doubling and it is given by $P3 = 2P1 = 2P2$. The hardware realization of PA and PD operations in affine-coordinate are given by the following expressions [68]

$$x_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + \mathrm{a}, & \mathrm{P}_1 \neq \mathrm{P}_2 \\ x_1^2 + \frac{\mathrm{b}}{x_1^2}, & \mathrm{P}_1 = \mathrm{P}_2 \end{cases} \tag{5.22}$$

$$y_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1 + x_3) + x_3 + y_1, & \mathrm{P}_1 \neq \mathrm{P}_2 \\ x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right)(x_3) + x_3, & \mathrm{P}_1 = \mathrm{P}_2 \end{cases} \tag{5.23}$$

It can be noted from the Eq.(5.22) and Eq.(5.23) that each Point-addition and Point-doubling operation requires computationally complex finite-field inversion operation and

it involves around 2m number of FF-inversion operations when using affine-coordinate systems. It is recommended to use projective-coordinate systems to avoid the FF-inversion, and Lopez-Dahab($X/Z^1, Y/Z^2$), standard($X/Z^1, Y/Z^1$), and Jacobian($X/Z^2, Y/Z^3$) are the most chosen projective coordinate systems [2]. Table 5.6 presents the comparison of the field operations in various projective systems, and it can be observed that the hardware realization of curve operations using Lopez Dahab projective coordinate systems is lesser complex compared to other projective coordinate systems.

Table 5.6 Comparison of Field operations in various Projective coordinate systems

Coordinate System	PA	PD
Lopez Dahab	9M + 4S	4M + 5S
Jacobian	10M + 4S	5M + 5S
Standard	12M + 1S	7M +5S

M is number of FF-multiplications,and S is number of FF-squaring Operations

Considering the computational complexity and hardware complexity of various projective systems we have employed the PA and PD expressions derived using Lopez-Dahab projective-coordinates and are given as [9],

$$\begin{aligned} \mathrm{X_3} &= x \cdot \mathrm{Z_3} + (\mathrm{X_1} \cdot \mathrm{Z_2}) \cdot (\mathrm{X_2} \cdot \mathrm{Z_1}) \\ \mathrm{Z_3} &= (\mathrm{X_1} \cdot \mathrm{Z_2} + \mathrm{X_2} \cdot \mathrm{Z_1})^2 \end{aligned} \tag{5.24}$$

$$\begin{aligned} X_3 &= \mathrm{X_1^4} + \mathrm{b.Z_1^4} \text{ or } \mathrm{X_3} = \mathrm{X_2^4} + \mathrm{b.Z_2^4} \\ Z_3 &= \mathrm{Z_1^2} \cdot \mathrm{X_1^2} \text{ or } \mathrm{Z_3} = \mathrm{Z_2^2} \cdot \mathrm{X_2^2} \end{aligned} \tag{5.25}$$

5.3.2 Point Multiplication Algorithm

There are several algorithms presented in the literature to achieve point multiplication namely, double-and-add algorithm(DAA), Non-adjacent-form(NAF) algorithm, window-NAF (width-w NAF) algorithm, Right-to-left algorithm, Left-to-right algorithm, Montgomery-ladder algorithm,Sliding-window-algorithm, and τ-adic-NAF (τ-NAF) algorithm. The review of these algorithms is presented in [10]. Implementations of these

algorithms are prone to different types of attacks and side-channel attacks are the most dominant of the available attacks on the point-multiplication algorithms [2]. Each curve operation is performed by a set of field operations, and each field operation has different timing and power behavior. In SAA the timing(Timing-attack), and power(Power-attack) variations of the curve operations are observed to encrypt the secret key. Power analysis attack can be done by tracing power of a single key operation (simple power analysis attack) or multiple traces of power (differential power analysis attack). In order to overcome the timing and power-analysis attacks, Montgomery [11] developed an efficient algorithm for point-multiplication to perform same set of curve operations independent of the key value.

A projective-coordinate Montgomery point multiplication algorithm is presented in Algorithm 5.2 [11], with point multiplication carried out in three stages. The first stage is initialization stage, where affine coordinates are converted to projective coordinates. The second stage is Main-loop, with set of PA and PD operations performed for m iterations. The final stage is post process used to recover affine coordinates. It can be observed from the algorithm that only projective X and Z coordinates along with affine x coordinate are used to realize the curve operations(main-loop) to reduce computational complexity of the algorithm. A modified Montgomery point multiplication algorithm is presented with uniform addressing [5] by merging the execution paths of k=0 and k=1 using swapping (X1 with X2 and Z1 with Z2). In [4] the initialization stage is merged with the main loop to execute both stages with a single data-path.

The proposed modified Montgomery algorithm for point multiplication is presented in Algorithm 5.3. This algorithm involves performing the finite field multiplication and finite field squaring operations in parallel to each other without data-dependency. The curve operations, PA and PD performed at each iteration remain to be same independent of the key value, with each individual curve operation realized in multiple clock-cycles to defend the side channel attacks. The field operations used to carry out curve operations(PA, PD) are scheduled appropriately to avoid data-dependency with no idle-cycles.

Algorithm 5.2 Montgomery point-multiplication over GF (2^m) in projective coordinates [11]

Input: k = $(k_{i-1}, ..., k_1, k_0)$ with $k_{i-1} = 1, P = (x_P,\ y_P) \in E/GF(2^m)$. Output: $Q = kP = (x_3,\ y_3)$.

Initialization : Affine to projective

1: $X_1 \leftarrow x_P, Z_1 \leftarrow 1, X_2 \leftarrow x_P^4 + b, Z_2 \leftarrow x_P^2$.

Main loop : Projective point addition and doubling

2: for t from $i-2$ downto 0 do

3: if $k_t = 1$ then

4: $R \leftarrow Z_1, Z_1 \leftarrow (X_1Z_2\ + X_2Z_1)^2, X_1 \leftarrow x_PZ_1 + X_1X_2RZ_2$.

$$R \leftarrow X_2,\ X_2 \leftarrow X_2^4 + bZ_2^4,\ Z_2 \leftarrow R^2Z_2^2.$$

5: else

6: $R \leftarrow Z_2, Z_2 \leftarrow (X_1Z_2\ +\ X_2Z_1)^2, X_2 \leftarrow x_PZ_2 + X_1X_2RZ_1$.

$$R \leftarrow X_1,\ X_1\ \leftarrow X_1^4 + bZ_1^4,\ Z_1\ \leftarrow R^2Z_1^2.$$

7: end if

8: end for

Post-process : Recover y and projective to affine

9: $x_3 \leftarrow X_1/Z_1$.

10:$y_3 \leftarrow (x_P + X_1/Z_1)[(X_1 + x_PZ_1)(X_2\ + x_PZ_2) + (x_P^2 + y_P)(Z_1Z_2)](x_PZ_1Z_2)^{-1} + y_P$

11: return $(x_3,\ y_3)$.

Algorithm 5.4 presents the post process stage of projective to affine conversion. It is realized in twelve steps and each step involves an FF-multiplication operation. It is also to be noted that step 9 of the Algorithm 5.4 involves an FF-inversion operation. In general, FF-inversion is computed using Itoh-Tsujii's algorithm [22] which employs Fermat's little theorem [8] or extended Euclidean algorithm (EEA) which employs greatest common division algorithm. Since the hardware realization of EEA is complex compared to that of ITA, the latter is preferred to compute FF-inversion, which requires a series of FF-Squaring, and FF-Multiplications operations. In this paper, we have considered the modified Itoh-Tsujii algorithm to compute FF-inversion improving the computation

delay by using 4^k exponentiation [1]. The data-path to accomplish FF-inversion is bound to use same data-path of main-loop without additional hardware.

Algorithm 5.3 Modified Montgomery point-multiplication over GF (2^m) in Lopez-Dahab projective coordinates

Input: $k = (k_{i-1}, ..., k_1, k_0)$ with $k_{i-1} = 1, P = (x_P, y_P) \in E/GF(2^m)$ and $c, b \in GF(2^m)$, where $c = \sqrt{b}$.

Output: $Q = kP = (x_3, y_3)$.

Initial Values: $X_1 \leftarrow 1, Z_1 \leftarrow 0, X_2 \leftarrow x_P, Z_2 \leftarrow 1$.

Affine to projective conversion and PA and PD FF-operations

for t from $i-1$ downto 0 do

Finite-field multiplication	Finite-field squaring
1: $Z_1 \leftarrow X_2 Z_1$	1: $T_2 \leftarrow X_2^2$
2: $X_1 \leftarrow X_1 Z_2$, $X_2 \leftarrow c$	2: $T_1 \leftarrow Z_2^2$, $Z_2 \leftarrow (X_1 + Z_1)^2$
3: $X_2 \leftarrow X_2 T_1 + T_2$	3: $X_2 \leftarrow X_2^2$
4: $X_1 \leftarrow X_1 Z_1$	4: No operation
5: $T_2 \leftarrow T_1 T_2$, $Z_1 \leftarrow Z_2$	5: No operation
6: $X_1 \leftarrow x_p Z_1 + X_1$, $Z_2 \leftarrow T_2$	6: No operation

if ($t \neq 0$ and $k_i \neq k_{i-1}$) or ($t = 0$ and $k_i = 0$) then

Swap (X_1, X_2) , Swap (Z_1, Z_2)

end if

end for

Post process: Recovering affine coordinates

$x_3 \leftarrow X_1/Z_1$.

$y_3 \leftarrow (x_P + X_1/Z_1)[(X_1 + x_P Z_1)(X_2 + x_P Z_2) + (x_P^2 + y_P)(Z_1 Z_2)](x_P Z_1 Z_2)^{-1} + y_P$

return (x_3, y_3)

Algorithm 5.4 Recovering affine coordinates

1: $X_2 \leftarrow X_2Z_1$	7: $T_2 \leftarrow X_2Z_1 + T_2$
2: $T_2 \leftarrow X_1X_2$	8: $Z_2 \leftarrow T_1Z_1$
3: $Z_2 \leftarrow X_1Z_2 + X_2$	9: $X_2 \leftarrow X_1\mathbf{inv}(Z_2) + x_p,\ T_1 \leftarrow T_2$
4: $Z_1 \leftarrow Z_1Z_2,\ T_1 \leftarrow x_p$	10: $Z_1 \leftarrow T_1\mathbf{inv}(Z_2),\ T_2 \leftarrow Z_1$
5: $X_1 \leftarrow T_1X_2$	11: $X_1 \leftarrow X_1\mathbf{inv}(Z_2),\ Z_1 \leftarrow y_p$
6: $T_2 \leftarrow T_1Z_2 + T_2,\ \ X_2 \leftarrow y_p$	12: $X_2 \leftarrow X_2T_2 + Z_1$

5.3.3 Proposed Point Multiplication Architecture

The proposed point multiplication architecture developed for the modified Montgomery Algorithm 5.3 is shown in Fig. 5.8. This architecture requires digit-serial multiplier, power-block, Registers, Multiplexers and a control unit to perform point multiplication. Point multiplication is realized by performing PA and PD operations on the point P(x_P, y_P) for each bit of m-bit key K. During every clock-cycle a set of inputs are applied to the digit-serial multiplier and power-block via multiplexers ($M_1, M_2, \& M_3$) and the outputs are stored into the registers T_1, T_2, X_1, X_2, X_3 and X_4. The inputs to be applied to any FF-operation unit and outputs to be stored into the particular register at each clock cycle is controlled by the control unit. As explained in the algorithm 5.3, each main loop iteration involves FF-multiplication, FF-addition, and FF-squaring operations.

FF-multiplication is realized by the digit-serial multiplier presented in Sec 5.2, which processes D bits of data at each clock cycle. If A, B are two m-bit operands then m bits of operand A are multiplied with D bits of operand B and this process is repeated for m/D iterations. The core multiplication of m-bits of A and D-bits of B is performed by the proposed bit-parallel multiplier.

The post process step in Algorithm 5.3 involves FF-inversion operation realized by using FF-squaring and 4^1 operations based on the analysis presented in [8, 69]. The hardware realization of FF-squaring and 4^1 operations for order m=163 are shown in the Figs. 5.9 and 5.10, respectively. It is to be observed that 4^1 is realized as a direct structure instead of cascading two FF-squaring modules to reduce the computational delay. It may

be noted that each iteration of main loop in algorithm 5.2 involves FF-addition operation and this FF-addition operation is implemented by bit-wise XOR operation on the given two operands.

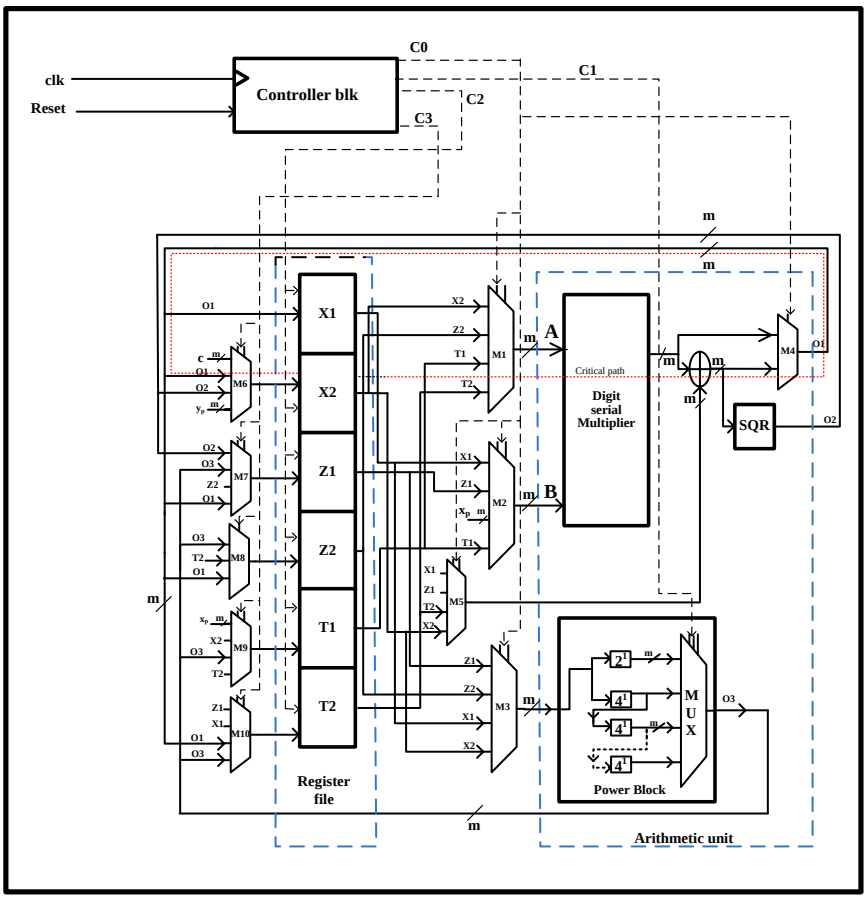

Figure 5.8 Proposed Architecture of Point Multiplication.

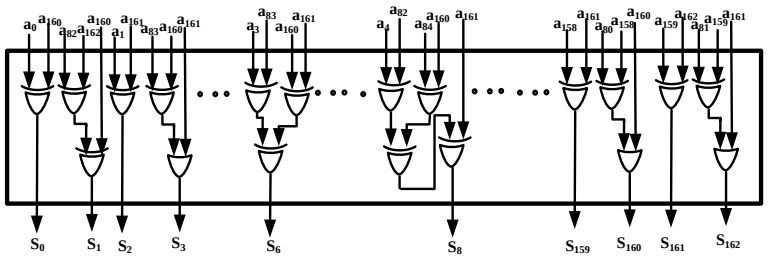

Figure 5.9 Architecture of the 2^1 module over GF(2^{163}).

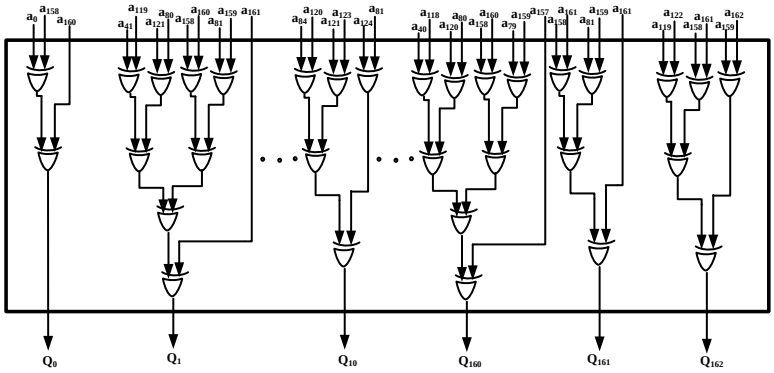

Figure 5.10 Architecture of the 4^1 module over GF(2^{163}).

Analytical Results

Critical Path Delay

The critical path of proposed point multiplication architecture consists of 2:1 multiplexer, FF-adder, FF-multiplier, and two 4:1 multiplexers. The critical path delay of the digit-serial multiplier for general irreducible polynomial is derived in sec 5.2.4 and is given by

$$T_{mul} = (\lceil log_2(D) \rceil + 1)T_X + T_{NX} + 2T_{NA} + T_D \tag{5.26}$$

where, D is digit size,T_{NA}, T_{NX}, T_X, and T_D are delays of two-input NAND gate, two-input XNOR gate, two-input XOR gate, and D-flipflop respectively. The delay of m-bit FF-adder realized using m XOR gates is equal to the delay of two-input XOR gate. If T_{mul}, T_X, T_{mux} denote delays of FF-multiplier, FF-adder, and two-input multiplexer, respectively, then critical path delay is given by the following expression

$$T_{CP} = T_{mul} + T_X + 5T_{mux} \tag{5.27}$$

Substituting the T_{mul} value(Eq.(5.26)) in the Eq.(5.27) we have

$$T_{CP} = (\lceil log_2(D) \rceil + 1)T_X + T_{NX} + 2T_{NA} + T_D + T_X + 5T_{mux} \tag{5.28}$$

Similarly, the critical path delay of the proposed point multiplication architecture for irreducible trinomials and irreducible pentanomials are computed and presented in Table 5.7.

It is to be noted that the critical-path delay of the point multiplication architecture depends on the critical-path delay of the FF-multiplier. In addition, the performance

Table 5.7 Critical-path delay of Proposed Point Multiplication Architecture

F(x)	Critical path delay
General	$(\lceil log_2(D) \rceil + 1)T_X + T_{NX} + 2T_{NA} + T_D + T_X + 5T_{mux}$
Trinomial	$(\lceil log_2(D) \rceil + 2)T_X + T_{NA} + T_D + T_X + 5T_{mux}$
Pentanomial	$(\lceil log_2(D) \rceil + 2)T_X + T_{NA} + T_D + T_X + 5T_{mux}$

of point multiplication architecture depends on the number of clock cycles required for point-multiplication. The number of clock cycles needed to perform point-multiplication is equal to the sum of the clock cycles required for main loop and for the post process stage. Since the main loop involves m iterations and each iteration takes six clock cycles, the total number clock cycles required for the main loop are equal to 6 × m. The clock cycles required for post process stage is equal to the clock-cycles required to recover affine coordinates (clock cycles for FF-inversion and others). The total number of clock cycles required to compute point multiplication are

$$C_{PM} = C_{ML} + C_{Quad} + M_{Inv} + C_{others} \tag{5.29}$$

where, C_{ML} denotes the clock cycles required for main loop, C_{Quad} denotes total clock cycles required to achieve $4^{((m-1)/2)}$exponentiation in FF-inversion, M_{Inv} represents clock cycles required for multiplications in FF-inversion and C_{others} denotes clock-cycles required for post process stage excluding FF-inversion.

Table 5.8 presents the clock cycles required for each FF-operation in the main loop and the post process stage of point multiplication algorithm. The number clock cycles required to complete point multiplication using the proposed architecture for GF(2^{163}) and GF(2^{233}) (m=163, 233 in Eq.(5.29) are 1020 and 1450 respectively.

5.3.4 Implementation Results

This section presents the implementation results (place and route) of the proposed point multiplication architecture over GF(2^{163}) and GF(2^{233}) on Virtex-5 (XC5VLX110) using Xilinx ISE tool. The performance analysis of the proposed point multiplication architecture for different digit sizes over GF(2^{163}) is presented in Table 5.9. It may be noted that as the digit sizes varies from 6 to 82 the computation time decreases from 8.19

Table 5.8 Clock cycles for Point Multiplication

Stage	FF-operation	clock-cylces
main loop	m iterations	$C_{ML} = \mathrm{m} \times C_m$
Post process	FF-inversion	$C_{Quad} = \sum_{i=2}^{l} \frac{u_i}{u_s}$, $M_{Inv} = (\lvert\log_2(m-1)\rvert + \mathrm{H(m-1)} - 1)$
	others	$C_{others} = 12$

where, m denotes field order, u_i addition chain element, u_s represents maximum cascade size of 4^k exponentiation, H is hamming weight of m-1, l is length of addition chain, and C_m denotes clock cycles for one iteration(main loop)

us to 1.07 us. It may also be noted that the area-time product decreases from 63.1 to 49 as the digit size is varied from 6 to 15 and increases from 49 to 77.6 as the digit size is varied from 15 to 82. Hence, the digit size of D=15 gives best area-time efficiency for the proposed point multiplication architecture over over GF(2^{163}).

Table 5.9 Performance comparison of the Proposed Point Multiplication Architecture for different digit size multiplication over GF(2^{163}).

Digit size	Fmax (MHz)	Area (LUTs)	critical-path (ns)	Time (us)	ATP ($\times 10^{-6}$)
6	124	7792	8.03	8.19	63.1
8	154	8950	6.49	6.62	59
11	206	10623	4.84	4.94	52
15	261	12566	3.82	3.9	49
24	369	19254	2.71	2.76	51.9
33	478	25742	2.09	2.13	54
55	704	44266	1.42	1.44	61.9
82	952	72058	1.05	1.07	77.6

The performance analysis of the proposed point multiplication architecture for different digit sizes over GF(2^{233}) is presented in Table 5.10. It may be noted that as the digit sizes varies from 16 to 59 the computation time decreases from 7.67 us to 2.65 us. It may also be noted that the area-time product decreases from 140 to 135 as the digit size is varied from 16 to 39 and increases from 139 to 143 as the digit size is varied from

47 to 59. Hence, the digit size of D=39 gives best area-time efficiency for the proposed point multiplication architecture over GF(2^{233}).

Table 5.10 Performance comparison of the Proposed Point Multiplication Architecture for different digit size multiplication over GF(2^{233}).

Digit size	Fmax (MHz)	Area (LUTs)	critical-path (ns)	Time (us)	ATP ($\times 10^{-6}$)
16	189	18314	5.29	7.67	140
39	395	37008	2.53	3.66	135
47	454	43808	2.2	3.19	139
59	546	54296	1.83	2.65	143

Table 5.11 presents the performance comparison of the proposed point multiplication architecture over GF(2^{163}) with that of other point multiplication architectures reported in the literature [47, 51–55]. It may be concluded that the proposed architecture achieves reduction in computation time of around 58%, 54%, 28%, 95%, and 20% when compared to the architectures [47, 51–53]. It may also be concluded that the architectures [54, 55] achieve better computational time compared to the proposed architecture at the cost of more area. However, the proposed architecture achieves area-time efficiency of around 48%, 43%, 60%, 89%, 4%, 35%, and 37% when compared to the architectures reported in the literature [47, 51–55], respectively.

Table 5.12 presents the performance comparison of the proposed point multiplication architecture over GF(2^{233}) with that of other point multiplication architectures reported in the literature [51, 52, 56–58, 70]. It can be concluded that the proposed architecture achieves reduction in computation time of around 70%, 81%, 85%, 96%, 94%, and 46% when compared to the architectures [51, 52, 56–58, 70]. It may also be concluded that the proposed architecture achieves area-time efficiency of around 40%, 69%, 83%, 85%, 80% and 53% when compared to the architectures reported in the literature [51, 52, 56–58, 70], respectively.

Table 5.11 Area and time complexities comparison for GF(2^{163}).

Design	Field	FPGA	Fmax (MHz)	clock cycles	Area (LUTs)	Time (us)	ATP ($\times 10^{-3}$)	% Reduction in ATP
[47]	163	virtex-5	147	-	10195	9.5	96	48
[51]	163	virtex-5	167	1429	10176	8.6	87	43
[52]	163	virtex-5	250	1371	22936	5.48	125	60
[53]	163	virtex-5	550	52012	4807	94.6	454	89
[54]	163	virtex-5	145	-	23135	2.22	51	4
[55]	163	virtex-5	211	547	29309	2.6	76	35
[59]	163	virtex-5	228	1119	16090	4.91	79	37
Proposed	163	virtex-5	261	1021	12566	3.9	49	–

Table 5.12 Area and time complexities comparison for GF(2^{233}).

Design	Field	FPGA	Fmax (MHz)	clock cycles	Area (LUTs)	Time (us)	ATP ($\times 10^{-3}$)	% Reduction in ATP
[51]	233	virtex-5	156	-	18097	12.3	225	40
[52]	233	virtex-5	192	3825	22340	19.9	444	69
[56]	233	virtex-5	132	3277	32874	25	821	83
[57]	233	virtex-5	50	5613	8612	112	964	85
[58]	233	virtex-5	264	18900	9576	71.5	684	80
[70]	233	virtex-5	360	-	42404	6.84	288	53
Proposed	233	virtex-5	395	1450	37008	3.66	135	–

5.4 Conclusions

In this chapter, a modified Montgomery-ladder algorithm is presented to reduce the number of clock cycles required for point-multiplication. An area-time efficient architecture is developed for the implementation of the proposed point-multiplication algorithm over irreducible pentanomial GF(2^{163}) and irreducible trinomial GF(2^{233}). The FF-inversion and FF-multiplication are two resource consuming and time critical opera-

tions in point-multiplication. Hence, area-time efficient digit-serial multiplier is designed by operating shift and reduction operations(SARs) in parallel (for every bit of the digit) and employing NAND gates in the computation of partial products. The reduction in the computation time is achieved by realizing modified Itoh-Tsujii algorithm using 4^K exponentiation modules to implement FF-inversion. The implementation results of the proposed architecture on Virtex-5 FPGA over the fields GF(2^{163}) and GF(2^{233}) show that the proposed design has less computation time and less area-time product compared to the similar architectures reported in the literature. Hence, the proposed point multiplication architecture may be recommended for ECC applications targeting less area-time product. However, some ECC applications demand high speed architectural design and is achieved by pipelining and parallelism techniques. Consequently, the next chapter focuses on the design of high-speed and low area-time complexity architectures for point multiplication suitable for ECC applications.

Chapter 6

High Speed and Area-Time Efficient Point Multiplication Architectures over GF(2^m) using Polynomial Basis

This chapter presents the design of high speed and area-time efficient point multiplication architectures targeting ECC applications. These architectures are developed based on modified Montgomery algorithm over GF(2^m) using polynomial basis. The proposed architectures are modeled using verilog HDL and simulated to verify the functionality using Xilinx Vivado tools. The HDL netlist is synthesized targeting Xilinx Virtex-5 FPGA and implemented to compare with the similar point multiplication architectures available in the literature.

6.1 Introduction

Advances in communication technology and availability high bandwidth enables millions of devices communicate confidential information over the Internet. Securing these technological devices has become an absolute necessity in this emerging digital world. The Elliptic Curve-Cryptography (ECC) suggested by Koblitz and Miller has captivated considerable attention when compared to similar cryptosystems available in the literature owing to its high security per bit ratio and small key size. The high performance of ECC relies on the finite-field arithmetic operations. In the previous chapter, we have presented

a low area-time complexity point multiplication architecture based on the modified Montgomery algorithm. This architecture is found to achieve best area-time product compared to the similar architectures at the cost of more area. However, some ECC applications demand both high speed and low area-time which can be achieved by exploring strategies like pipelining and Parallelism. In addition, NIST recommended irreducible trinomials and irreducible pentanomials require less hardware and computational complexity compared to the case of general irreducible polynomials. Hence, it is attempted in this work to design a high speed and a low area-time complexity point multiplication architectures employing the pipelining and parallelism strategies for the case of NIST irreducible trinomials and Pentanomials.

Some architectures are proposed to minimize the clock cycles required for point-multiplication by introducing parallelism at instruction level and architecture level [6, 47–49, 51].

In this chapter, we have presented a two stage point multiplication architecture and its corresponding area and delay formulations. The proposed point multiplication is developed based on the modified Montgomery point multiplication algorithm. The classic Montgomery algorithm for point multiplication is modified to perform parallel FF-multiplication and FF-squaring operations. The area and delay formulations are derived and the performance is compared with the existing point multiplication architectures in the literature. In addition, a three stage point multiplication architecture proposed for the case of irreducible trinomials is also presented. This architecture is developed by realizing the FF-inversion operation required for point multiplication using the proposed parallel Itoh-Tsujii algorithm.

6.2 Area-time Efficient point multiplication architecture over GF(2^m)

In this section, the design and performance analysis of the proposed two stage point multiplication architecture are presented. First, mathematical preliminaries for the proposed modified Montgomery algorithm are presented. Followed by, hardware realization

of the proposed point multiplication algorithm and the corresponding area and time complexity formulations are presented. Finally, the comparison of implementation results with the similar point multiplication architectures available in the literature are presented.

6.2.1 Point Multiplication Algorithm over GF(2^m)

Algorithm 6.1 presents the proposed modified Montgomery algorithm for point multiplication. In this Algorithm 6.1, the finite field multiplication and finite field squaring operations are subjected to perform in parallel to each other without data-dependency. The FF-operations performed at each iteration remain to be same irrespective of k value so as to defend the timing and power analysis attack. In the proposed scheme, the point addition (PA) and point doubling (PD) operations are performed in multiple steps making it difficult to trace particular FF-operation in a particular step of the algorithm. Each main loop iteration is executed in six clock cycles with each FF-operation timed appropriately to avoid data-dependency with no idle-cycles. The post process stage is realized using Algorithm 6.2 and it involves an FF-inversion operation. The Itoh-Tsujii's algorithm [22], and extended euclidean algorithm (EEA), are the most used algorithms for computing FF-inversion. The ITA uses Fermat's little theorem [8], while the EEA computes FF-inversion using the greatest common divisor approach. ITA is the most commonly used FF-inversion algorithm, which requires a sequence of FF-Squaring and FF-Multiplications operations to compute FF-inversion.

In this work, the modified Itoh-Tsujii algorithm [1] is used to compute FF-inversion for reducing the computational delay by employing 4^k exponentiation. The data-path to accomplish FF-inversion is bound to use the same data-path of main-loop costing no extra hardware.

Algorithm 6.1 Modified Montgomery-ladder point-multiplication algorithm over GF (2^m) in projective-coordinates

Input: $k = (k_{i-1}, ..., k_1, k_0)$ with $k_{i-1} = 1, P = (x_P, y_P) \in E/GF(2^m)$ and $c, b \in GF(2^m)$, where $c = \sqrt{b}$.

Output: $Q = kP = (x_3, y_3)$.

Initial Values: $X_1 \leftarrow 1, Z_1 \leftarrow 0, X_2 \leftarrow x_P, Z_2 \leftarrow 1$.

Main-loop: PA and PD operations

for t from $i-1$ downto 0 do

FF-multiplication	FF-squarer
cc1: $Z_1 \leftarrow X_2 Z_1$	cc1: $T_1 \leftarrow Z_2^2$
cc2: $X_2 \leftarrow cT_1 + T_2$	cc2: $T_2 \leftarrow X_2^2$
cc3: $X_1 \leftarrow X_1 Z_2$,	cc3: $X_2 \leftarrow X_2^2$
cc4: $Z_2 \leftarrow T_1 T_2$	cc4: No operation
cc5: $T_2 \leftarrow X_1 Z_1 \quad T_1 \leftarrow x_p$	cc5: $Z_1 \leftarrow (X_1 + Z_1)^2$
cc6: $X_1 \leftarrow T_1 Z_1 + T_2$	cc6: No operation

if ($t \neq 0$ and $k_i \neq k_{i-1}$) or ($t = 0$ and $k_i = 0$) then

Swap (X_1, X_2) , Swap (Z_1, Z_2)

end if

end for

projective to affine conversion

$x_3 \leftarrow X_1/Z_1$.

$y_3 \leftarrow (x_P + X_1/Z_1)[(X_1 + x_P Z_1)(X_2 + x_P Z_2) + (x_P^2 + y_P)(Z_1 Z_2)](x_P Z_1 Z_2)^{-1} + y_P$

return (x_3, y_3)

Algorithm 6.2 Projective to affine conversion

cc1: $X_2 \leftarrow X_2Z_1$	cc7: $T_2 \leftarrow X_2Z_1 + T_2$
cc2: $T_2 \leftarrow X_1X_2$	cc8: $Z_2 \leftarrow T_1Z_1$
cc3: $Z_2 \leftarrow X_1Z_2 + X_2$	cc9: $X_2 \leftarrow X_1\mathbf{inv}(Z_2) + x_p$, $T_1 \leftarrow T_2$
cc4: $Z_1 \leftarrow Z_1Z_2,\ T_1 \leftarrow x_p$	cc10: $Z_1 \leftarrow T_1\mathbf{inv}(Z_2),\ T_2 \leftarrow Z_1$
cc5: $X_1 \leftarrow T_1X_2$	cc11: $X_1 \leftarrow X_1\mathbf{inv}(Z_2),\ Z_1 \leftarrow y_p$
cc6: $T_2 \leftarrow T_1Z_2 + T_2,\ X_2 \leftarrow y_p$	cc12: $X_2 \leftarrow X_2T_2 + Z_1$

Table 6.1 Operation-wise comparison of the Point-multiplication Algorithm

Design	Algorithm	# FF Squarings	# FF Multiplications	# FF Additions	#PAs	#PDs
[47, 51–53, 55, 59]	Montgomery Ladder	5(m-1)	6(m-1)	3(m-1)	m-1	m-1
Proposed	Modified Montgomery Ladder	4m	6m	3m	m	m

In Table 6.1, the comparison of main-loop FF-operations and curve operations of the proposed point-multiplication algorithm with the existing point-multiplication algorithms are presented.

It can be observed from the Table 6.1 that the proposed modified Montgomery point multiplication algorithm requires around m number of FF-squarings operations less when compared to the existing Montgomery point multiplication algorithms presented in the literature [47, 51–53, 55, 59]. It may also be noted that the number of PA and PD operations are increased by one when compared to [47, 51–53, 55, 59]. The increased PA and PD operations are due to the merging of initialization stage with the main loop so as to avoid the separate hardware for initialization.

Table 6.2 presents the hardware and latency comparison of FF-inversion realized using the proposed modified Itoh-Tsujii algorithm [1] with that of the FF-inversion operation used by the point-multiplication architectures [47, 51–53, 55, 59] considered for comparison.

Table 6.2 Hardware and Latency comparison of modified Itoh-Tsujii Algorithm

Design	# FF Squarings	# FF Multiplications	#FF Quad	latency (clock cycles)
[51]	(m-1)	l + 1	-	L(l+1)+ $\left[L(l+1)+\sum_{i=2}^{l}\left\lceil\frac{u_i-u_{i-1}}{u_s}\right\rceil\right]$
[47]	-	l+1	(m-1)/2	L(l+1)+ $\left[(l+1)+\sum_{i=2}^{l}\left\lceil\frac{u_i-u_{i-1}}{u_s}\right\rceil\right]$
[52]	-	-	-	2m/d
[59]	(m-1)	$\lvert\log_2(m-1)\rvert + H(m-1) - 1$	-	(m-1)/2 + 3(H(m − 1) +$\lvert\log_2(m-1)\rvert$-1) + 3
[54]	(m-1)	$\lvert\log_2(m-1)\rvert + H(m-1) - 1$	-	(m-1)/2 + 2(H(m − 1) +$\lvert\log_2(m-1)\rvert$-1)
[55]	(m-1)	$\lvert\log_2(m-1)\rvert + H(m-1) - 1$	-	$\sum_{i=2}^{l}\lceil\frac{u_i}{5}\rceil$ ++2(H(m − 1) +$\lvert\log_2(m-1)\rvert - 1$)
Proposed MITA [1]	-	$\lvert\log_2(m-1)\rvert + H(m-1) - 1$	(m-1)/2	$\sum_{i=2}^{l}\left\lceil\frac{u_i}{u_s}\right\rceil$ +2(H(m − 1) +$\lvert\log_2(m-1)\rvert - 1$)

6.2.2 Proposed Point Multiplication Architecture

In this work, we have developed the architecture shown in the Fig. 6.1 for the hardware realization of the point multiplication over GF(2^m) using Algorithm 6.1. This architecture is designed using FF-multiplier, multiplexers, and power-block employing two-stage pipelining. The functionality of FF-multiplier and power block are explained in Chapter 4 (Sec 4.2.3). The multiplexers M1, M2,and M3 are used to provide inputs to the FF-multiplier and power-block. During every clock-cycle, the control unit issues a set

of control signals to the multiplexers and registers. The input data to the FF-multiplier and power-block is received either from the registers (X_1, X_2, Z_1, Z_1, T_1, and T_2) or from the external source(x_p, y_p) through M5 multiplexer and it depends on the status of control signals. At the end of each clock-cycle, the results are stored in the registers (X_1, X_2, Z_1, Z_1, T_1, and T_2) and this procedure is repeated for m iterations, where m is the order of the field.

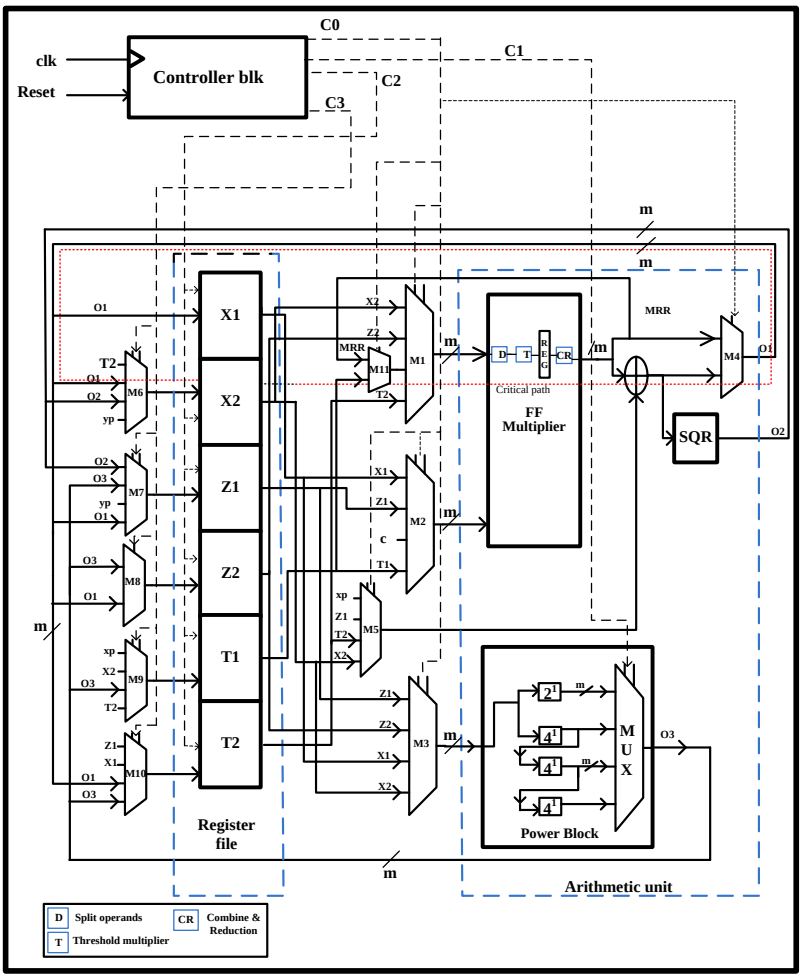

Figure 6.1 Proposed Architecture of Point Multiplication.

The functionals blocks required to realize the point multiplication architecture are presented below.

Registers

The registers X_1, X_2, Z_1, Z_1, T_1, and T_2, store the intermediate results of power-block and FF-multiplier. The value to be stored in registers X_2, Z_1, Z_1, T_1, and T_2, for a particular clock-cycle is applied via multiplexers M6, M7, M8, M9 and M10,respectively, and is controlled by the control signals C4 and C5. The register X_1 is applied with output O1 based on the control signal C5. The values stored in the registers are applied as inputs to the power-block and FF-multiplier through multiplexers M1 ,M2 and M3 based on the control signals C1, C2 and C3 respectively.

FF-adder

The FF-adder is placed at the output of FF-multiplier to perform add on fly FF-addition avoiding an extra clock cycle for FF- addition. The FF-addition is performed by applying logical XOR operation on m bits of the two operands in parallel.

Data flow diagram

Fig. 6.2 demonstrates the data flow diagram for the proposed Modified Montgomery-ladder point-multiplication algorithm. It explains the organization of hardware blocks employed in the proposed architecture to accomplish the main loop iteration in six clock cycles (except the first iteration). It is to be observed that, at each clock cycle FF-squaring and FF-multiplication are executed in parallel. It is also to be observed that FF-multiplication accomplished in two clock cycles(two stage pipelining) and FF-squaring is accomplished in one clock cycle. It may be noted that, one clock cycle is saved per each iteration by overlapping the FF-addition and FF-multiplication operations of the current iteration with the FF-operations of next iteration without any data discrepancy.

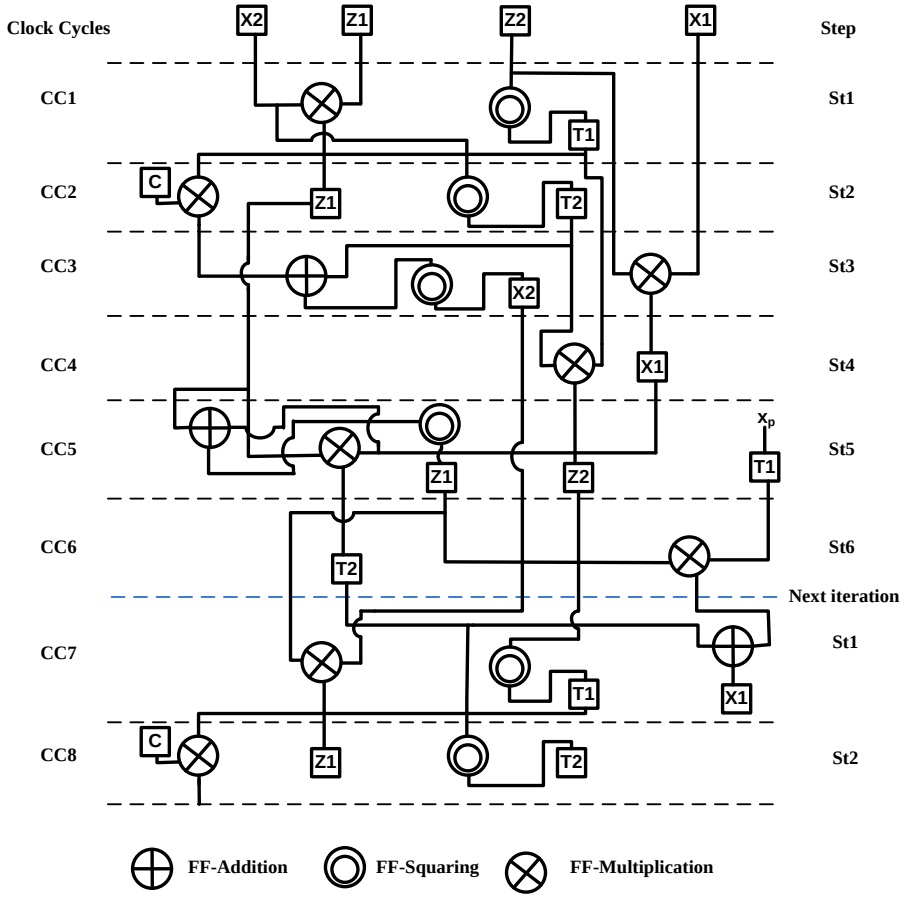

Figure 6.2 Data flow diagram for the Proposed Modified Montgomery Point Multiplication Algorithm

6.2.3 Hardware and Delay Complexity Analysis

Hardware Complexity Analysis

This section presents the estimation of hardware complexity of the proposed point multiplication architecture. The hardware complexity is expressed in terms of d input LUTs, and the d value varies with target FPGA device. The complexities are computed based on the analysis presented in [1].

The total area complexity of the proposed point multiplication architecture shown in Fig. 6.1 can be computed by the summation of area complexity of each building block as

$$\begin{aligned} PMA_{Area} =& 3m \times \text{lut}\,(2+\log_2 2) + 9m \times \text{lut}\,(4+\log_2 4) + \frac{2(m-1)}{2} \\ &+ 2m \times lut(x) + (\sum_{i=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \sum_{i=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) \\ &+ 2m-1) \end{aligned} \tag{6.1}$$

where, $LUT_{hkmul}(\frac{m}{2}) = 2 \times LUT_{hkmul}(\frac{m}{4}) + LUT_{hkmul}(\frac{m}{4}) + m - 1$ and x is the size of bit element in the binary sequence with maximum number of inputs in a 4^1 exponentiation block(x varies with field order).

For example, the size of x will be equal to 9 for an irreducible pentanomial GF(2^163) and the total area complexity is given as

$$\begin{aligned} PMA_{Area} =& 17m - 1 + (\sum_{i=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \sum_{i=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) \\ &+ 2m-1) \end{aligned} \tag{6.2}$$

Similarly for an irreducible trinomial GF(2^{233}), the size of x will be equal to 4 and the total area complexity is given as

$$\begin{aligned} PMA_{Area} =& 15m - 1 + (\sum_{i=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \sum_{i=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) \\ &+ 2m-1) \end{aligned} \tag{6.3}$$

Delay Complexity Analysis

The architecture developed for point multiplication is a two-stage pipelined structure (see Fig. 6.1) and the delay can be estimated by computing the maximum stage delay and the clock cycles needed to perform ECC point-multiplication operation. The critical path delay of the pipelined architecture is equal to the maximum stage delay. It may be observed from the Fig. 6.1 that the proposed Point multiplication architecture consists of two critical paths, one through the power-block and the other via FF-multiplier. The delays of each building block of the proposed architecture are computed in terms of d-input LUT delay [1] and are estimated as follows,

FF-multiplier: The hybrid Karatsuba multiplier employed consists of four stages and the delay of FF-multiplier is equal to the summation of delays of splitting stage ($D_{\text{sp}} =$

$log_{\mathrm{d}}(\frac{\mathrm{m}}{\tau})$), classical-multiplier stage ($D_{\mathrm{Th}} = log_{\mathrm{d}}2\tau$), alignment stage ($D_{\mathrm{cm}} = log_2(\frac{\mathrm{m}}{\tau})$) [62] and modular reduction stage ($D_{\mathrm{mod}} = log_{\mathrm{d}}\mathrm{t}$). The FF-multiplier delay is given by the following expression

$$D_{\mathrm{FFmul}} = D_{\mathrm{sp}} + D_{\mathrm{Th}} + D_{\mathrm{cm}} + D_{\mathrm{mod}} \tag{6.4}$$

Multiplexer: The multiplexer delay with x inputs and $\log_2 x$ selection lines can be computed by $D_{mux} = \log_{\mathrm{d}}(x + \log_2 x)$.

FF-adder: Since the FF-addition involves logical XOR operation on two bits, the delay of FF-adder is equal to delay of a single XOR gate(one LUT delay).

Power-block: The power-block delay is the summation of the delays across x:1 multiplexer and the cascaded 4^1 modules as shown in the Fig. 6.1. It can be computed by the following expression,

$$\begin{aligned} \#D_{\mathrm{Powblk}} &= D_{pb} + D_{MUX_P} \\ &= \sum D_{4^1} + \lceil \log_d (x + \log_2 x) \rceil . \end{aligned} \tag{6.5}$$

where, D_{pb} is the cascaded delay of 4^1 modules and D_{MUX_P} is the Multiplexer delay. The delay of single 4^1 is equal to longest of delay taken to generate output bit sequence of 4^1.

The critical-path delay(D_{PM}) of the Proposed pipelined point-multiplication architecture is the maximum of delays across the critical-path1(CP-1) and critical-path2(CP-2). The delay expressions of critical-path1 and critical-path2 are given as

$$D_{CP-1} = D_{FFmul} + D_{MUX(M4)} + D_{MUX(M6)} + D_{MUX(M11)} + D_{ADD} \tag{6.6}$$

$$D_{CP-2} = D_{MUX(M3)} + D_{pb} + D_{MUX_P} \tag{6.7}$$

Using Eq.(6.6) and Eq.(6.7), the critical path delay of the Proposed multiplication architecture can be computed as

$$D_{PM} = Max(D_{CP-1}, D_{CP-2}) \tag{6.8}$$

where, D_{PM} varies with the order of the field.

The performance of the point-multiplication architecture depends on the critical-path delay of the proposed architecture. The higher the critical-path delay, the lower the operating frequency of the architecture. To improve the operating frequency, the point-multiplication architecture should be pipelined at the appropriate stages of the critical path. Considering a L staged pipelined architecture, the critical path delay D_{PM} reduces to D_{PM}/L. But, increasing the number of pipelined stages will increase the clock cycles required for multiplication. So the number of pipelined stages required for the point-multiplication architecture are chosen in such a way that it maintains optimal frequency in all the critical paths with minimal number of clock-cycles required for point-multiplication.

The critical path delay of the proposed two-staged point multiplication architecture is given as

$$\begin{aligned} PMA_{delay} =& D_{MUX(M11)} + D_{MUX(M1)} + D_{\mathrm{sp}} + D_{\mathrm{Th}} \\ & (or) D_{\mathrm{cm}} + D_{\mathrm{mod}} + D_{MUX(M4)} + D_{MUX(M6)} + D_{ADD} \\ PMA_{delay} =& \log_d (2 + \log_2 2) + \log_d (4 + \log_2 4) + \log_d \left(\frac{\mathrm{m}}{\tau}\right) + \log_d 2\tau \\ & (or) \log_2 \left(\frac{\mathrm{m}}{\tau}\right) + \log_d \mathrm{t} + \log_d (2 + \log_2 2) + \log_d (4 + \log_2 4) + 1 \end{aligned} \tag{6.9}$$

where, τ is threshold multiplier size. For example, the critical path delay for GF(2^{163})) is shown in the Fig. 6.3 and it is to be noted that the delay is equal to 7 LUT's(7 $\times$ target FPGA LUT delay).

Estimation of computation time

The computation time of the proposed point multiplication architecture is

$$T_{PM} = C_{PM} \times T_P \tag{6.10}$$

where, C_{PM} denotes clock cycles required to perform point-multiplication and T_P represents the time period of the clock

It may be noted from Algorithm 6.1 that the number of clock cycles required for point-multiplication is equal to the summation of clock cycles required for m main-loop iterations and the clock cycles required for post process stage and is given by

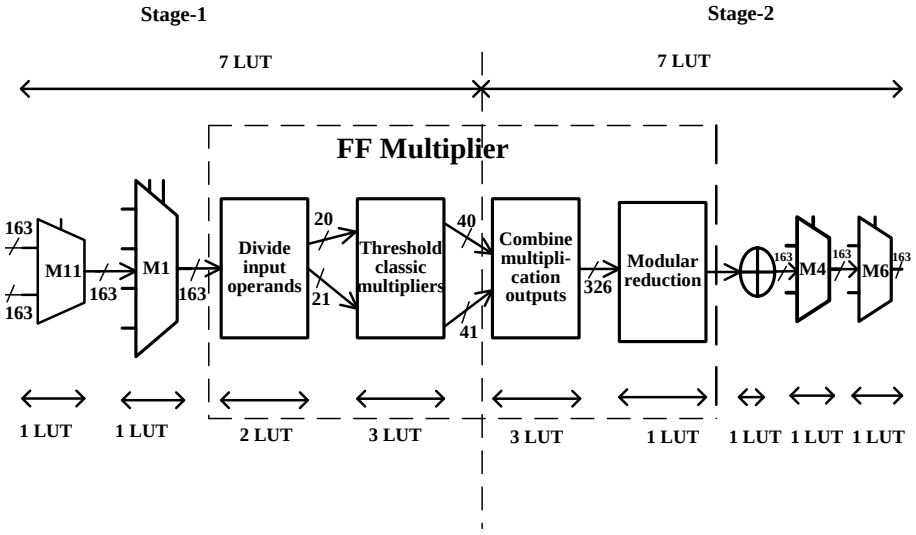

Figure 6.3 The critical-path of the proposed architecture (GF(2^{163})).

$$C_{PM} = C_{ML} + C_{PA} \tag{6.11}$$

where, C_{ML} denotes total clock-cycles required for main loop and C_{PA} represents clock-cycles required for post process stage.

$$C_{ML} = \mathrm{m} \times C_m \tag{6.12}$$

where, m denotes field order and C_m denotes number of clock cycles required for one iteration of main loop

$$C_{PA} = C_{INV} + C_{Additional} \tag{6.13}$$

where, C_{INV} represents clock cycles required to compute FF-inversion and $C_{Additional}$ denotes clock-cycles required for post process stage excluding C_{INV}

$$C_{INV} = C_{Quad} + LM_{Inv} \tag{6.14}$$

where, C_{Quad} denotes total clock-cycles required for exponentiation and M_{Inv} represents clock-cycles required for multiplications in FF-inversion

C_{Quad} and M_{Inv} are computed as

$$C_{Quad} = \sum_{i=2}^{l} \left\lceil \frac{u_i}{u_s} \right\rceil \tag{6.15}$$

where, u_i element in the addition chain and u_s maximum cascade size of 4^k exponentiation

$$M_{Inv} = (|\log_2(\mathrm{m}-1)| + \mathrm{H}(\mathrm{m}-1) - 1) \tag{6.16}$$

$$C_{PM} = \mathrm{m} \times C_m + C_{Quad} + LM_{Inv} + C_{Additional} \tag{6.17}$$

where, L denotes number of pipelined stages

For example, the clock cycles required for the proposed two stage point multiplication architecture to perform one point multiplication over GF(2^{163}) is given as $6 \times 163 + 33 + 18 + 13 = 1042$

6.2.4 Implementation Results

The performance of the proposed point-multiplication architecture is compared with the point-multiplication architectures reported in the literature [47,51–55,59]. The architectures proposed for hardware realization of Point multiplication over Galois field with field orders m=163, and m=233 are modeled using verilog and implemented on Virtex-5 FPGA technology to enable fair comparison with the relevant reported architectures implemented using the same FPGA technology. The comparison of time-complexity of the proposed two stage point-multiplication architecture is presented in Table 6.3 to compare with that of other architectures reported in the literature.

The performance of the proposed point-multiplication architecture over GF(2^{163}) is presented in Table 6.4 to compare with that of others architectures reported in the literature [47, 51–53, 55, 58, 59, 71]. It may be observed from the results that the architectures [47, 51, 52, 55, 59] require 4%, 4%, 60%, 66%,and 40%, respectively, more area compared with the proposed architecture when implemented using Virtex-5 FPGA technology. It may also be noted that the architectures [47,51–53,55,58,59,71]reported in the literature require more area-time complexity of around 34%, 27%, 49%, 86%, 17%, 79%, 20%, and 76%, respectively, when compared with the proposed architecture.

Table 6.5 presents the performance of the proposed architecture over GF(2^{233}) and other architectures reported in the literature [51,52,56–58,70,72]. It may be noted from the results that the architectures [51,52,56,70] require 21%, 37%, 56%, and 66%, respectively,

Table 6.3 Time-complexity comparison of the Proposed Point Multiplication architecture over GF(2^m)

Design	latency(clock cycles)	critical-path delay
[51]	$2mL + \left[L(l+1) + \sum_{i=2}^{l} \left\lceil \frac{u_i - u_{i-1}}{u_s} \right\rceil \right] + [7 + 9L]$	$\log_d(2 + \log_2 2) + \log_d(4 + \log_2 4) + 3$
[47]	$m \cdot 7 + \frac{m}{3} \cdot 15 + \left[L(l+1) + \sum_{i=2}^{l} \left\lceil \frac{u_i - u_{i-1}}{u_s} \right\rceil \right] + 7$	$2\log_d(4 + \log_4 2) + 2$
[59]	$5 + (6)(m-1) + (m-1)/2 + 3(\mathrm{H(m-1)}$ $+\lvert \log_2(\mathrm{m}-1) \rvert \text{-}1) + 31$	$\log_2(n + 2r) + \log_d(2 + \log_2 2) + 1$
[54]	$3(w-1)d + 5(2^w - (w+1)) + 7d$	$log_d(\frac{m}{\tau})) + log_d 2\tau)$
[55]	$3\mathrm{m} + \sum_{i=2}^{l} \lceil \frac{u_i}{5} \rceil +$ $+2(\mathrm{H(m-1)} + \lvert \log_2(\mathrm{m}-1) \rvert - 1)$	–
proposed work	$6\mathrm{m} + \sum_{i=2}^{l} \left\lceil \frac{u_i}{u_s} \right\rceil + 2(\mathrm{H(m-1)}$ $+\lvert \log_2(\mathrm{m}-1) \rvert - 1) + 13$	$log_2(\frac{m}{\tau}) + log_d t + \log_d(4 + \log_2 4)$ $+\log_d(2 + \log_2 2) + 1$

L is number of pipeline stages, l is length of addition chain, u_s is maximum cascade size, H is hamming weight of m-1,u_i is element in the addition chain, d is digit size, w is window size,n is number of segments in a multiplier, r is r-nomial irreducible polynomial, τ is threshold multiplier size

Table 6.4 Performance comparison of the proposed Point Multiplication Architecture with the existing Point Multiplication Architectures over GF(2^{163})

Design	Field	FPGA	Fmax (MHz)	clock cycles	Area (LUTs)	Time (us)	ATP ($\times 10^{-3}$)	% Reduction in ATP
[47]	163	virtex-5	147	-	10195	9.5	96	34
[51]	163	virtex-5	167	1429	10176	8.6	87	27
[52]	163	virtex-5	250	1371	22936	5.48	125	49
[53]	163	virtex-5	550	52012	4807	94.6	454	86
[55]	163	virtex-5	211	547	29309	2.6	76	17
[58]	163	virtex-5	290	13000	6959	44.69	310	79
[59]	163	virtex-5	228	1119	16090	4.91	79	20
[71]	163	virtex-5	106	3426	8457	32.3	270	76
Proposed	163	virtex-5	158	1042	9760	6.5	63	–

Table 6.5 Performance comparison of the proposed Point Multiplication Architecture with the existing Point Multiplication Architectures over GF(2^{233}).

Design	Field	FPGA	Fmax (MHz)	clock cycles	Area (LUTs)	Time (us)	ATP ($\times 10^{-3}$)	% Reduction in ATP
[51]	233	virtex-5	156	-	18097	12.3	225	42
[52]	233	virtex-5	192	3825	22340	19.9	444	70
[56]	233	virtex-5	132	3277	32874	25	821	84
[57]	233	virtex-5	50	5613	8612	112	964	86
[58]	233	virtex-5	264	18900	9576	71.5	684	80
[70]	233	virtex-5	360	-	42404	6.84	288	54
[72]	233	virtex-5	119	174047	6912	1462	10105	98
Proposed	233	virtex-5	158	1476	14137	9.2	130	–

more area than the proposed point-multiplication architecture when targeted on Virtex-5 FPGA technology. It may be also observed that the architectures [[51, 52, 56–58, 70, 72] require more area-time complexity of around 42%, 70%, 84%, 86%, 80%, 54%, and 98%, respectively, than the proposed architecture.

6.3 High speed Point Multiplication Architecture over GF(2^m) for the case of irreducible Trinomials

In this section, the design and performance analysis of the proposed three stage point multiplication architecture are presented. A parallel Itoh-Tsujii algorithm is derived to realize the FF-inversion operation and a modified Montgomery algorithm is developed to realize the point multiplication for the case of irreducible trinomials. The area and time complexity formulations are derived and implemented for the case of irreducible trinomials. A verilog model is developed to verify the functionality and the modeled designed is implemented on Virtex-5 FPGA to compare with the similar architectures available in the literature.

6.3.1 Point multiplication Algorithm over GF(2^m)

In the literature, several algorithms are presented to perform point-multiplication, namely NAF algorithm, τ-NAF algorithm, Window-NAF algorithm, Montgomery algorithm, double-add algorithm [10]. Each algorithm has its unique advantages and are used based on the application requirement and the type of binary elliptic curve employed. For example, NAF, and Window-NAF algorithms are used for applications with less hardware complexity(Hamming weight is at most m/3, and the number of PA operations are reduced) and τ-NAF algorithm is used in cases of Koblitz curves.

Realization of these point-multiplication algorithms are subjected to various kinds of attacks and side channel attack is the predominant attack on point multiplication algorithm [2]. In side channel attacks, the adversary studies the timing and power behavior of each curve operation for multiple iterations of point multiplication algorithm to encrypt the secret key. These side-channel attacks are of two sub-types namely timing attacks and Power analysis attacks and the description of these attacks is presented in [73].

In order to overcome the timing and power-analysis attacks, Montgomery [11] has presented a point multiplication algorithm with each iteration involving both PA and PD curve operations. It will be difficult to trace a particular curve operation as timing and power behavior remain to be same for each iteration of point multiplication algorithm.

Algorithm 6.3 presents the proposed modified Montgomery-algorithm for point-multip-lication. It is to be observed that Algorithm 6.3 is realized in two steps, step one performs initialization and curve operations, and step two performs projective to affine conversion. It is also to be observed that the PA and PD curve operations at each iteration remained to be same and the set of finite field operations required to realize each of these curve operations are rearranged and regrouped to perform in parallel. The regrouping of finite field operations makes it difficult to trace the type of curve operation.

Algorithm 6.3 Modified Montgomery algorithm for point multiplication over GF (2^m)

Input: $k = (k_{i-1}, ..., k_1, k_0)$ with $k_{i-1} = 1, P = (x_P, y_P) \in E/GF(2^m)$ and $c, b \in GF(2^m)$, where $c = \sqrt{b}$.

Output: $Q = kP = (x_3, y_3)$.

Initial Values: $X_1 \leftarrow 1, Z_1 \leftarrow 0, X_2 \leftarrow x_P, Z_2 \leftarrow 1$.

Main-loop: Point Addition and Point Doubling operations

for t from $i-1$ downto 0 do

FF-multiplication	FF-squarer
cc1: $Z_1 \leftarrow X_2 Z_1$	cc1: $T_1 \leftarrow Z_2^2$
cc2: $X_1 \leftarrow X_1 Z_2$	cc2: $T_2 \leftarrow X_2^2$
cc3: $X_2 \leftarrow cT_1 + T_2$	cc3: No operation
cc4: $T_2 \leftarrow X_1 Z_1$	cc4: $Z_1 \leftarrow (X_1 + Z_1)^2$
cc5: $Z_2 \leftarrow T_1 T_2$	cc5: $X_2 \leftarrow X_2^2$
cc6: $X_1 \leftarrow Z_1 x_p + T_2$	cc6: No operation

if ($t \neq 0$ and $k_i \neq k_{i-1}$) or ($t = 0$ and $k_i = 0$) then

Swap (X_1, X_2) , Swap (Z_1, Z_2)

end if

end for

Recovering Affine Coordinates

$x_3 \leftarrow X_1/Z_1$.

$y_3 \leftarrow (x_P + X_1/Z_1)[(X_1 + x_P Z_1)(X_2 + x_P Z_2) + (x_P^2 + y_P)(Z_1 Z_2)](x_P Z_1 Z_2)^{-1} + y_P$

return (x_3, y_3)

The second step of recovering the affine coordinates from the projective coordinates is presented in Algorithm 6.4 and it involves computationally complex finite-field inversion operation. The Itoh-Tsujii algorithm [22] based on Fermat's little theorem and extended euclidean algorithm (EEA) based on greatest common divisor approach, are two predominantly used algorithms for the realization of finite field inversion. The performance of the

point multiplication architecture depends on the performance of finite-field multiplication and finite field inversion.

In this work, we present the parallel Itoh-Tsujii algorithm(see Algorithm 6.5) for the realization of FF-inversion and a cascade block of 4^k exponentiation modules are employed to reduce number of clock cycles required to realize FF-inversion.

Algorithm 6.4 Recovering Affine Coordinates

cc1: $X_2 \leftarrow X_2Z_1$	cc7: $T_2 \leftarrow X_2Z_1 + T_2$
cc2: $T_2 \leftarrow X_1X_2$	cc8: $Z_2 \leftarrow T_1Z_1$
cc3: $Z_2 \leftarrow X_1Z_2 + X_2$	cc9: $X_2 \leftarrow X_1\mathbf{inv}(Z_2) + x_p$, $T_1 \leftarrow T_2$
cc4: $Z_1 \leftarrow Z_1Z_2,\ T_1 \leftarrow x_p$	cc10: $Z_1 \leftarrow T_1\mathbf{inv}(Z_2),\ T_2 \leftarrow Z_1$
cc5: $X_1 \leftarrow T_1X_2$	cc11: $X_1 \leftarrow X_1\mathbf{inv}(Z_2)$
cc6: $T_2 \leftarrow T_1Z_2 + T_2,\ X_2 \leftarrow y_p$	cc12: $X_2 \leftarrow X_2T_2 + y_p$

FF-Inversion Algorithm over GF(2^m)

For any nonzero element $a \in GF\,(2^m)$, the FF-inversion is given by the following expression

$$a^{-1} \equiv a^{2^m-2} \tag{6.18}$$

Eq.(6.7) can also be expressed as

$$a^{-1} = \left[\gamma_{m-1}(a)\right]^2, \tag{6.19}$$

where,

$$\gamma_q(a) = a^{2^q-1} \tag{6.20}$$

with $\gamma_q(a) \in GF\,(2^m)$ and $q \in$ N, with N being a set of natural-numbers.
The notation $\gamma_q(a)$ is denoted as γ_q for further formulations.

The Eq.(6.20) can be generalized in-terms of two integers $q, r \geq 0$, and is given as

$$\gamma_{q+r}(a) = \gamma_q(a)^{2^r}\gamma_r(a) = \gamma_r(a)^{2^q}\gamma_q(a) \tag{6.21}$$

The Eq.(6.21) can be expressed in terms of addition-chain index pair elements u_{i_1} and u_{i_2} as

$$\gamma_{u_{i_2}+u_{i_1}}(a) = \left[\gamma_{u_{i_1}}(a)\right]^{2^{u_{i_2}}} \gamma_{u_{i_2}}(a) = \gamma_{u_i}(a) = a^{2^{u_i}-1} \tag{6.22}$$

where, $\gamma_{u_1}(a) = a^{2^1-1} = a$. The FF-inverse of an element a such that $a^{-1} \in GF\left(2^m\right)$ can be derived by successively applying Eq.(6.12) for each addition chain index element u_i and performing FF-squaring operation at the final step u_f. It is expressed as,

$$\left[\gamma_{u_f}(a)\right]^2 = \left(a^{2^{m-1}-1}\right)^2 = \left(a^{2^m-2}\right) = a^{-1} \tag{6.23}$$

Table 6.6 presents the realization of FF-inverse of a such that $a^{-1} \in GF\left(2^{233}\right)$ defined over a irreducible trinomial $R(x) = x^{233}+x^{74}+1$ based on the assumption $\gamma_{\mathrm{u_i}}(a) = a^{4^{\mathrm{u_i}}-1}$. It can be observed from the table that each step consists of 4^k exponentiation followed by FF-multiplication. It is also to be observed that the 4^k exponentiation is performed on the FF-multiplication outcome of the previous step.

Table 6.6 FF-Inverse of a such that $a^{-1} \in GF\left(2^{233}\right)$ [1]

	$\gamma_{\mathbf{u_i}}(\mathbf{a})$	$\gamma_{\mathbf{u_j}+\mathbf{u_l}}(\mathbf{a})$	Exponentiation
1	$\gamma_1(a)$		a^3
2	$\gamma_2(a)$	$\gamma_{1+1}(a)$	$(\gamma_1)^{4^1}\,\gamma_1 = a^{4^2-1}$
3	$\gamma_3(a)$	$\gamma_{2+1}(a)$	$(\gamma_2)^{4^1}\,\gamma_1 = a^{4^3-1}$
4	$\gamma_6(a)$	$\gamma_{3+3}(a)$	$(\gamma_3)^{4^3}\,\gamma_3 = a^{4^6-1}$
5	$\gamma_7(a)$	$\gamma_{6+1}(a)$	$(\gamma_6)^{4^1}\,\gamma_1 = a^{4^7-1}$
6	$\gamma_{14}(a)$	$\gamma_{7+7}(a)$	$(\gamma_7)^{4^7}\,\gamma_7 = a^{4^{14}-1}$
7	$\gamma_{28}(a)$	$\gamma_{14+14}(a)$	$(\gamma_{14})^{4^{14}}\,\gamma_{14} = a^{4^{28}-1}$
8	$\gamma_{29}(a)$	$\gamma_{28+1}(a)$	$(\gamma_{28})^{4^1}\,\gamma_1 = a^{4^{29}-1}$
8	$\gamma_{58}(a)$	$\gamma_{29+29}(a)$	$(\gamma_{29})^{4^{29}}\,\gamma_{29} = a^{4^{58}-1}$
9	$\gamma_{116}(a)$	$\gamma_{58+58}(a)$	$(\gamma_{58})^{4^{58}}\,\gamma_{58} = a^{4^{116}-1}$

It may be noted from the Table 6.6 that steps 2, 3, 5 and 8 involves FF-multiplication of $\gamma_{\mathbf{u_j}}(\mathbf{a})$ exponentiation with γ_1 and these steps can be alternatively realized by the

following expressions using Eq.(6.21).

$$\begin{aligned}(\gamma_2)^{4^1}\gamma_1 &= (\gamma_1)^{4^2}\gamma_2 \\ (\gamma_6)^{4^1}\gamma_1 &= (\gamma_1)^{4^6}\gamma_6 \\ (\gamma_{28})^{4^1}\gamma_1 &= (\gamma_1)^{4^{28}}\gamma_{28}\end{aligned} \tag{6.24}$$

γ_1 is a precomputed value and is stored in a register. The exponentiation involving γ_1 in each subsequent step can be computed in parallel with FF-multiplication operation of the current step to reduce the computation time of FF-inversion. Based on the idea of parallel exponentiation, and FF-multiplication, a parallel Itoh-Tsujii algorithm developed for FF-inversion is presented in Algorithm 6.5. The algorithm starts with precomputation of the element Z_2^3. The steps 4 to 23 generate the required exponentiation, while steps 24 to 27 perform the FF-multiplication. For each iteration, the difference between two consecutive addition chain elements is performed. Based on the difference value, required exponentiation is performed with a maximum of 4^{u_c} exponentiation per clock cycle. If the required exponentiation is greater than 4^{u_c} then it is achieved in multiple clock cycles(steps 6 to 10). If the difference value is equal to 1, the required exponentiation is generated using steps 13 to 21. The parallel execution of exponentiation and FF-multiplication is done at steps 25 and 27. The register T_1 stores the initial value Z_2^3(see step 2). Depending on exponentiation value that is to be achieved in the most recent subsequent step involving T_1, the required exponentiation is performed on T_1 in parallel with current step FF-multiplication. Step 25 performs an exponentiation equal to 4^{u_c}, while step 27 performs an exponentiation less than 4^{u_c} and execution of these steps is based on the values of t, u_c, and u_{i-1}. The steps 4 to 28 are repeated for each element of addition-chain and a Finite-field squaring operation is performed on X_2 at the final step to complete inversion (Z_2^{-1}).

Algorithm 6.5 Parallel Itoh-Tsujii Algorithm using exponentiation 4^k

Input: The element $Z_2 \in GF\,(2^m)$, cascade size $u_{\rm C}$, addition chain length l, and the addition chain $U = \{1, 2, \ldots, \frac{m-1}{2}\}$
Output: Multiplicative inverse $Z_2^{-1} \in GF\,(2^m)$;
begin
1: $Z_1 = Z_2^2$; t=0; s=0;
2: $X_2 = Z_1 * Z_2$; $T_1 = X_2$;
3: **for each** $u_i \in U\,(2 \leq i \leq f)$ **do**
4: $s = u_i - u_{i-1}$;
5: **if**(s $\neq$ 1 and s $>u_{\rm C}$) **then**
6: $Z_1 = X_2^{4^{u_C}}$, $Z_2 = Z_1$; s = s - $u_{\rm C}$
7: **While**(s$>u_{\rm C}$)**do**
8: $Z_1 = Z_2^{4^{u_C}}$, $Z_2 = Z_1$; s = s - $u_{\rm C}$
9: **end while**
10: $Z_1 = Z_2^{4^s}$
11: **else if** (s $\neq$ 1 and s $<u_{\rm C}$) **then**
12: $Z_1 = X_2^{4^s}$
13: **else if** (s = 1 and $u_{\rm i-1}$ =2) **then**
14: $Z_1 = T_1$
15:: **else**
16: $Z_1 = T_1^{4^{u_C}}$, $Z_2 = Z_1$; s = $s - t - u_{\rm C}$
17: **While**(s$>u_{\rm C}$)**do**
18: $Z_1 = Z_2^{4^{u_C}}$, $Z_2 = Z_1$; s = s - $u_{\rm C}$
19: **end while**
20: $Z_1 = Z_2^{4^s}$
21: **end if**
22: **end if**
23: **end if**
24: **if**(t+ $u_{\rm C}$ $<u_{\rm i-1}$) **then**
25: $X_2 = Z_1 * X_2$; $T_1 = T_1^{4^{u_C}}$ t= t+$u_{\rm C}$ **/* parallel*/**
26: **else**
27: $X_2 = Z_1 * X_2$; $T_1 = T_1^{4^{t+u_C-u_{i-1}}}$ t= t+$u_{\rm C}$-$u_{\rm i-1}$ **/* parallel*/**
28: **end if**
26: **end for**
23: $Z_2 = X_2^2$; $Z_2^{-1} = Z_2$;
end

where, $f = \lceil \log_2(\mathrm{m}-1) \rceil + \mathrm{H}(\mathrm{m}-1) - 2$

Table 6.7 presents the realization of FF-inverse of a such that $a^{-1} \in GF\,(2^{233})$ defined over an irreducible trinomial $R(x) = x^{233} + x^{74} + 1$ based on the proposed parallel Itoh-Tsujii algorithm. It can be observed from this table that the exponentiation over γ_1 required at subsequent steps is realized in parallel along with FF-multiplication operation at steps 2, 3, 5, 6, and 7.

Table 6.7 FF-Inverse of a such that $a^{-1} \in GF\left(2^{233}\right)$ using Parallel-ITA

	$\gamma_{\mathbf{u_i}}(\mathbf{a})$	$\gamma_{\mathbf{u_j+u_l}}(\mathbf{a})$	Exponentiation	Parallel $(\gamma_1)^{4^k}$
1	$\gamma_1(a)$		a^3	–
2	$\gamma_2(a)$	$\gamma_{1+1}(a)$	$(\gamma_1)^{4^1}\gamma_1 = a^{4^2-1}$	$(\gamma_1)^{4^2}$
3	$\gamma_3(a)$	$\gamma_{2+1}(a)$	$(\gamma_1)^{4^2}\gamma_2 = a^{4^3-1}$	$(\gamma_1)^{4^5}, (\gamma_1)^{4^6}$
4	$\gamma_6(a)$	$\gamma_{3+3}(a)$	$(\gamma_3)^{4^3}\gamma_3 = a^{4^6-1}$	–
5	$\gamma_7(a)$	$\gamma_{6+1}(a)$	$(\gamma_1)^{4^6}\gamma_6 = a^{4^7-1}$	$(\gamma_1)^{4^9}, (\gamma_1)^{4^{12}}, (\gamma_1)^{4^{15}}$
6	$\gamma_{14}(a)$	$\gamma_{7+7}(a)$	$(\gamma_7)^{4^7}\gamma_7 = a^{4^{14}-1}$	$(\gamma_1)^{4^{18}}, (\gamma_1)^{4^{21}}, (\gamma_1)^{4^{24}}$
7	$\gamma_{28}(a)$	$\gamma_{14+14}(a)$	$(\gamma_{14})^{4^{14}}\gamma_{14} = a^{4^{28}-1}$	$(\gamma_1)^{4^{27}}, (\gamma_1)^{4^{28}}$
8	$\gamma_{29}(a)$	$\gamma_{28+1}(a)$	$(\gamma_1)^{4^{28}}\gamma_{28} = a^{4^{29}-1}$	–
9	$\gamma_{58}(a)$	$\gamma_{29+29}(a)$	$(\gamma_{29})^{4^{29}}\gamma_{29} = a^{4^{58}-1}$	–
10	$\gamma_{116}(a)$	$\gamma_{58+58}(a)$	$(\gamma_{58})^{4^{58}}\gamma_{58} = a^{4^{116}-1}$	–

The hardware and latency comparison of the proposed Parallel-ITA with the FF-inversion algorithms [47,51–53,55,59] reported in the literature is presented in Table 6.8.

6.3.2 Proposed Point Multiplication Architecture for the case of Irreducible Trinomials

This section presents the proposed point multiplication architecture for the realization of modified Montgomery algorithm(see Fig. 6.4) employing the proposed parallel-ITA for FF-inversion. The architecture consists of power-block, FF-adder, FF-multiplier, and a Register file. The control unit is the major building block of the architecture and it is enabled or disabled based on the status signals, start and reset. For each clock signal, a series of control signals are generated to the arithmetic unit and the register file. At the first clock cycle, the base point (xp,yp) and curve constant are initialized into the register file. In the subsequent clock cycles, a set of FF-operations are performed according to the Algorithm 6.5. Fig. 6.5 shows the detailed data path of the proposed architecture. It can be observed from the Fig. 6.5 that the Finite-field multiplier and power-block receive

Table 6.8 Hardware and Latency comparison of FF-inversion Algorithms

Design	latency (clock cycles)	# FF Multiplications	# FF Squarings	#FF Quad
[51]	L(l+1)+ $\left[L(l+1)+\sum_{i=2}^{l}\left\lceil\frac{u_i-u_{i-1}}{u_s}\right\rceil\right]$	l + 1	(m-1)	-
[47]	L(l+1)+ $\left[(l+1)+\sum_{i=2}^{l}\left\lceil\frac{u_i-u_{i-1}}{u_s}\right\rceil\right]$	l+1	-	(m-1)/2
[52]	2m/d	-	-	-
[59]	(m-1)/2 + 3($H(m-1)$ +$\lceil\log_2(m-1)\rceil$-1) + 3	$\lceil\log_2(m-1)\rceil + H(m-1) - 1$	(m-1)	-
[54]	(m-1)/2 + 2(H(m − 1) +$\lceil\log_2(\mathrm{m}-1)\rceil$-1)	$\lceil\log_2(\mathrm{m}-1)\rceil + \mathrm{H}(\mathrm{m}-1) - 1$	(m-1)	-
[55]	$\sum_{i=2}^{l}\lceil\frac{u_i}{5}\rceil$ ++2(H(m − 1) +$\lceil\log_2(\mathrm{m}-1)\rceil - 1$)	$\lceil\log_2(\mathrm{m}-1)\rceil + \mathrm{H}(\mathrm{m}-1) - 1$	(m-1)	-
Proposed PITA	$\sum_{i=2}^{l}\left\lceil\frac{u_i}{u_s}\right\rceil$ +3(H(m − 1) +$\lceil\log_2(\mathrm{m}-1)\rceil - 2$)	$\lceil\log_2(\mathrm{m}-1)\rceil + \mathrm{H}(\mathrm{m}-1) - 1$	-	(m-1)/2

a set of m-bit data via the multiplexers M1, M2,and M3, for each clock cycle and the processed data is written into the registers (X_1, X_2, Z_1, Z_1, T_1, and T_2) at the end of each clock-cycle. This is a continuous process and is repeated m times, where m is the field order. The functionality of FF-multiplier and power-blocks of the proposed point multiplication architecture are explained in Chapter 4 (Sec 4.2.3).

Data flow diagram

The Algorithm 6.3 can be better understood by using a data flow diagram and is shown in the Fig. 6.6. It explains the realization of each step of Algorithm 6.3 in hardware, with a clear picture of execution time of each FF-operation. It may be observed from the Fig. 6.6 that it takes seven clock cycles to complete one main loop iteration. It may also be observed that the execution time of each FF-squaring and FF-multiplication operation are one and three clock cycles, respectively. The grouping of FF-operations without data

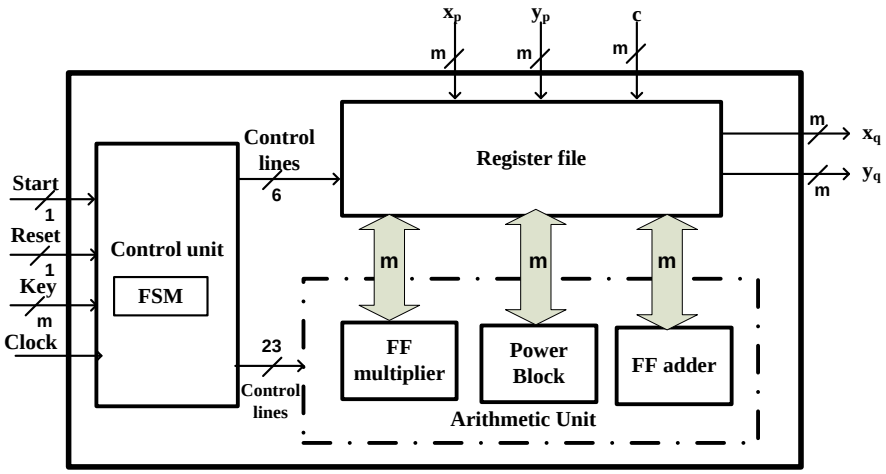

Figure 6.4 Block diagram of Proposed Point Multiplication Architecture for the case of Irreducible Trinomials.

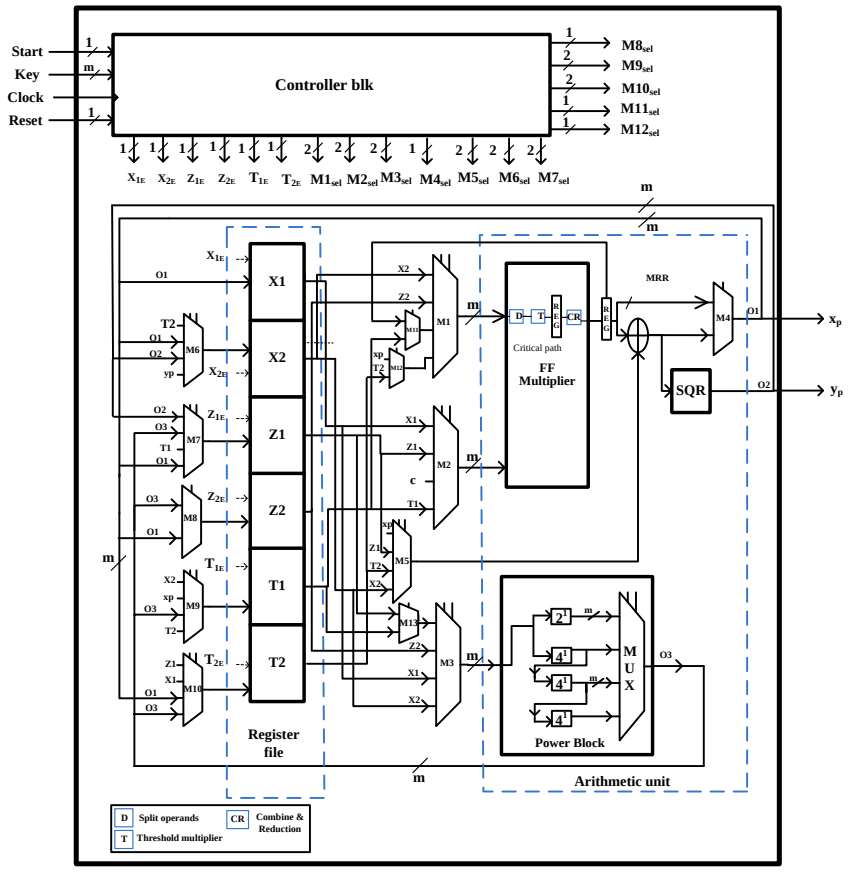

Figure 6.5 Proposed Point Multiplication Architecture for the case of Irreducible Trinomials.

dependency saves a total of m clock cycles with one clock cycle saved per iteration.

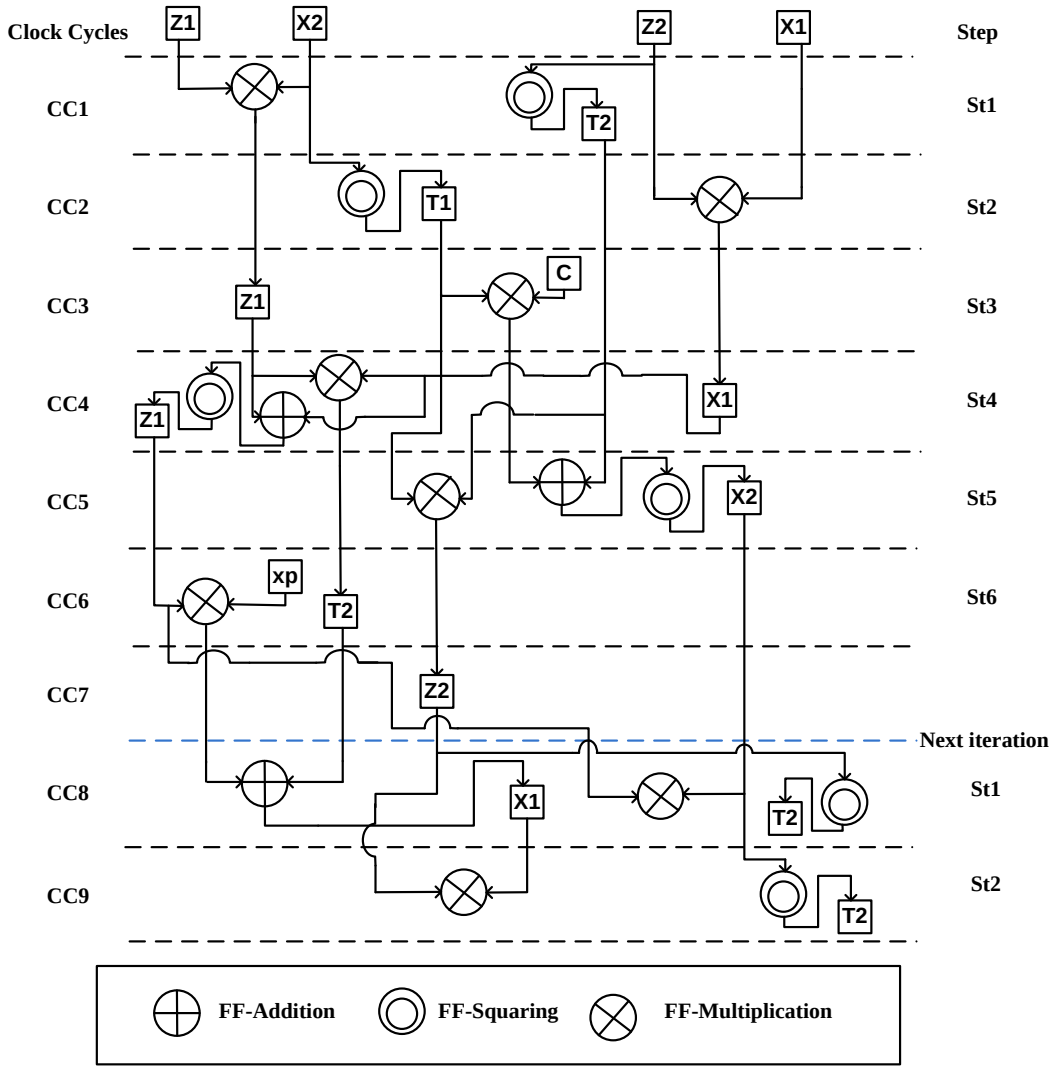

Figure 6.6 Data flow diagram of the point Multiplication Architecture

6.3.3 Analysis of the Proposed Architecture

Area Complexity Analysis

The total area complexity is computed by the summation of each individual block hardware (see Fig. 6.5) and is given as

$$\begin{aligned} PMA_{Area} =& 5m \times \text{lut}\,(2+\log_2 2) + 9m \times \text{lut}\,(4+\log_2 4) + \frac{2(m-1)}{2} \\ &+ \frac{3(m-1)}{4} + \Big(\sum_{m=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{m}{2} \right\rceil\right) + \sum_{m=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{m}{2} \right\rfloor\right) \\ &+ 4m - 2\Big) \end{aligned} \tag{6.25}$$

where, $LUT_{hkmul}(\frac{m}{2}) = 2 \times LUT_{hkmul}(\frac{m}{4}) + LUT_{hkmul}(\frac{m}{4}) + m - 1$

The total area complexity (Eq.(6.23) can be reformulated as,

$$PMA_{Area} = \frac{79m-13}{4} + (\sum_{i=m}^{2\tau} 2 \times LUT_{hkmul}\left(\left\lceil \frac{i}{2} \right\rceil\right) + \sum_{i=m}^{2\tau} LUT_{hkmul}\left(\left\lfloor \frac{i}{2} \right\rfloor\right)) \quad (6.26)$$

Time Complexity Analysis

The total time complexity of the proposed point multiplication architecture is computed by estimating the individual block delays (see Fig. 6.5). The proposed architecture is three staged pipelined architecture and the critical path delay is calculated by estimating maximum path delay through FF-multiplier and Power block. The delays of individual blocks are estimated based on the analysis presented in [1].

The critical path delay of the proposed architecture is given as

$$\begin{aligned} PMA_{delay} =& D_{MUX(M1)} + D_{MUX(M11)} + D_{\mathrm{sp}} + D_{\mathrm{Th}} \\ & (or) D_{\mathrm{cm}} + D_{\mathrm{mod}} (or) D_{pb} + D_{MUX_P} \\ PMA_{delay} =& \log_d (2 + \log_2 2) + \log_d (4 + \log_2 4) + \log_d \left(\frac{\mathrm{m}}{\tau}\right) + \log_d 2\tau \\ & (or) \log_2 \left(\frac{\mathrm{m}}{\tau}\right) + \log_d \mathrm{t} (or) \sum D_{4^1} + \lceil \log_d (x + \log_2 x) \rceil \end{aligned} \quad (6.27)$$

The critical path delay estimated for the irreducible trinomial R(x)=$x^{233} + x^{74} + 1$ in GF(2^{233})) is equal to 7 LUT's(7 $\times$ target FPGA LUT delay). The total computation time to perform one point multiplication is derived based on the analysis presented in section and is given as

$$C_{PM} = \mathrm{m} \times C_m + C_{Quad} + LM_{Inv} + C_{Additional} \quad (6.28)$$

with, L being number of pipelined stages. For example, to perform one point multiplication in GF(2^{233}) takes about $7 \times 233 + 72 + 14 = 1717$ clock cycles.

6.3.4 Implementation Results

This section presents the implementation results of the proposed point multiplication algorithm for the class of irreducible trinomials. The NIST trinomials $x^{193} + x^{15} + 1$, $x^{233} + x^{74} + 1$ and $x^{409} + x^{87} + 1$ are considered for experimentation to compare with the similar architectures reported in the literature. The verilog model of the proposed

architecture is implemented on the target platform Xilinx (XC5VLX110) Virtex-5 using Xilinx ISE tools to have a fair comparison with the similar architectures reported in the literature, which are implemented using the same FPGA technology.

The results of proposed architecture implemented on Virtex-5 target platform (Xilinx XC5VLX110) over GF(2^{233}) are presented in Table 6.9. The area-time complexity comparison shows that the proposed architecture achieves less area time product of about 30%, 64%, 80%, 83%, 77% and 98%, respectively, when compared to the available architectures [51, 52, 56–58, 72]. In addition, these existing architectures [51, 52, 56–58, 72] require more computation time of about 12%, 45%, 56%, 90%, 84%, and 99%, respectively.

The results of proposed architecture implemented on Virtex-5 target platform (Xilinx XC5VLX110) over GF(2^{409}) are presented in Table 6.10. The area-time complexity comparison shows that the architectures[[52]a, [52]b, [52]c, [4]d, [4]e, [4]f] require more area-time product of around 84%, 80%, 78%, 10%, 7%, and 11%, respectively, when compared to the proposed architecture. In addition, these existing architectures [[52]a, [52]b, [52]c, [4]d, [4]e, [4]f] require more computation time of about 92%, 86%, 81%, 7%, 3%,and 10%, respectively

Table 6.9 Performance comparison of Point Multiplication over GF(2^{233})

Design	Field	FPGA	Fmax (MHz)	clock cycles	Area (LUTs)	Time (us)	ATP ($\times 10^{-3}$)	% Reduction in Time	% Reduction in ATP
[51]	233	virtex-5	156	-	18097	12.3	225	12	30
[52]	233	virtex-5	192	3825	22340	19.9	444	45	64
[56]	233	virtex-5	132	3277	32874	25	821	56	80
[57]	233	virtex-5	50	5613	8612	112	964	90	83
[58]	233	virtex-5	264	18900	9576	71.5	684	84	77
[72]	233	virtex-5	119	174047	6912	1462	10105	99	98
Proposed	233	virtex-5	158	1717	14603	10.8	157	–	

Table 6.10 Performance comparison of Point Multiplication over GF(2^{409}).

Design	Field	FPGA	Fmax (MHz)	clock cycles	Area (LUTs)	Time (us)	ATP ($\times 10^{-3}$)	% Reduction in Time	% Reduction in ATP
[52]a	409	virtex-5	181	4551	16169	250.3	4047	92	84
[52]b	409	virtex-5	163	23373	22315	142.6	3182	86	80
[52]c	409	virtex-5	161	16541	28503	102.6	2924	81	78
[4]d	409	virtex-5	143	2906	34778	20.3	705	7	10
[4]e	409	virtex-5	172	3337	35313	19.4	685	3	7
[4]f	409	virtex-5	200	4177	34389	20.9	718	10	11
Proposed	409	virtex-5	158	2983	33950	18.7	634	–	

6.4 Conclusions

In this chapter, the design of area-time efficient point multiplication architecture for irreducible polynomials and high speed point multiplication architecture for NIST recommended irreducible trinomials are presented. These proposed point multiplication architectures are designed employing pipelining and parallelism strategies to improve the computation time. In addition, modified-ITA and parallel Itoh-Tsujii algorithm are also employed in point multiplication architectures for irreducible polynomials and NIST recommended irreducible trinomials, respectively, to further reduce the computation time. Further, the formulations for area and delay complexities for the proposed architectures are presented.The implementation results for the proposed point multiplication architecture over an irreducible polynomial requires less area-time product compared to the available architectures considered for comparison. It is also observed from the implementation results of the proposed point multiplication architecture for the NIST recommended trinomials that it achieves less computation time compared to the existing architectures in the literature. Hence, the proposed point multiplication architectures are suitable for high speed and low area-time ECC applications.

Chapter 7

Conclusions and Future Scope

This chapter concludes the thesis by underlining the main contributions. It also presents the possible directions of future work.

7.1 Conclusions

In this thesis we have investigated various finite field arithmetic operations used for the construction of point multiplication architecture. The most hardware critical finite field operations contributing for the development of point multiplication architecture are finite-field inversion and finite-field multiplication operations. Various design methodologies are explored to optimize the area-time product at the different hierarchical levels of ECC point multiplication. The following summarizes the contributions of this work.

- In Chapter 4, we have presented a finite field inversion architecture over GF(2^m) using polynomial basis, since finite field inversion is one of the resource consuming and time critical operation in realizing point multiplication. In this regard, we have presented an area-efficient finite filed inversion architecture using the proposed modified Itoh-Tsujii algorithm. The performance of finite field inversion architecture depends on the exponentiation module employed and the number of cascaded exponentiation modules. It is observed that the 4^k exponentiation modules outperform the traditional 2^k exponentiation modules by 30% and 50% in terms of area and time, respectively. Hence, a series of cascade blocks of 4^k exponentiation mod-

ules have been employed to realize the exponentiation required at each intermediate step of finite field inversion. Through the comparisons of analytical and FPGA implementation results obtained for the field orders $m = 193$, and $m = 409$, we have shown that the proposed finite field inversion architecture achieves reduction in area around 15% to 70% and reduction in ATP (area-time-product) around 5% to 15% compared to the existing architectures for the field orders m=193 and m=409, respectively

- In Chapter 5, we have presented the design of digit serial multiplier over a specific classes of trinomials and pentanomials to achieve the reduction in area-time product of the proposed point multiplication architecture. It is observed that the logical NAND gates consume less hardware compared to the traditional AND gates to generate the partial products. Hence, traditional AND gates are replaced by logical NAND gates to generate the partial products in realizing the FF-multiplication. Through the comparisons of analytical and FPGA implementation results obtained, we have shown that the proposed point multiplication architecture achieves reduction in ATP(area-time-product) around 4% to 85% and 35% to 80% compared to the existing architectures for the field of orders m = 163 and m = 233, respectively.

- In Chapter 6, we have presented the development of proposed point multiplication architectures over GF(2^m) using polynomial basis, since these point multiplication architectures determine the performance of ECC applications. An area-time efficient point multiplication architecture is developed using the proposed modified Montgomery point multiplication algorithm. The lower level finite field operations like FF-multiplication and FF-squaring are designed to perform in parallel while realizing the proposed Montgomery point multiplication algorithm. Through the comparisons of analytical and FPGA implementation results obtained, we have shown that the proposed point multiplication architecture achieves reduction in ATP(area-time-product) around 15% to 80% and 40% to 95% compared to the existing architectures for the field of orders m = 163 and m = 233, respectively. In addition, we have also proposed an high speed point multiplication architecture for a class of trinomials, since the point multiplication architectures developed for a class of trinomials found to consume less hardware and time. Through the comparisons

of analytical and FPGA implementation results obtained, we have shown that the proposed point multiplication architecture achieves reduction in computation time around 10% to 95% and 3% to 90% compared to the existing works for the field orders m=233 and m=409, respectively.

7.2 Future Scope

The work proposed in this thesis can be extended for future research. Some of the possible directions in which the problems can be further pursued are:

- The finite-field inversion and finite-field multiplication are two resource consuming and time critical operations in point multiplication. The proposed architectures can be investigated with the available FF-multipliers for better area-time product. In addition k-chains can be explored for realizing FF-inversion, instead of traditional addition chains (2-chain).

- The most works available in the literature used Binary Weierstrass curves for FPGA implementation of point multiplication. There are other binary elliptic curves which can be investigated for the FPGA implementation of point multiplication, namely Koblitz curves, Generalized Hessian curves, Binary Huff curves and Binary Edward curves. Each of these curves have its unique advantages. Point multiplication using Koblitz curves can be implemented without point doubling curve operation. Generalized Hessian curves defend side-channel attacks with unified addition formulas. Binary Huff curves exhibit resistance against power attacks with unified point doubling and point addition formula.

- The proposed architectures have been developed based on the Modified Montgomery algorithm in Lopez-Dahab projective coordinates. There are several other algorithms to implement point multiplication: Left-to-right, right-to-left, Non-adjacent-form (NAF) method, τ-adic NAF (NAF) method, window NAF method (width-w NAF), and Sliding window method. For example in NAF method implementation of point multiplication, the hamming weight of the secret key is reduced by 3 times of that of the original key and it results in the reduction of curve level operations

ultimately reducing the area-time complexity of point multiplication. Hence, these algorithms can be investigated for better Area and time complexity trade-off.

Publications

List of International Journals:

---

1. Nadikuda Pradeep Kumar Goud and Boppana Lakshmi, "An area-efficient architecture for finite field inversion over GF(2^m) using polynomial basis," *Microprocessors and Microsystems*, (2022): 104439. **(SCI-Indexed, Elsevier)**

2. Nadikuda Pradeep Kumar Goud and Boppana Lakshmi, "An area-time efficient point multiplication architecture for ECC over GF(2^m) using polynomial basis," *Microprocessors and Microsystems*, (2022): 104525. **(SCI-Indexed, Elsevier)**

3. Nadikuda Pradeep Kumar Goud and Boppana Lakshmi, "Low area-time complexity point multiplication architecture for ECC over GF(2^m) using polynomial basis," *Journal of Cryptographic Engineering.* **(SCI-Indexed, Springer Nature)**

4. Nadikuda Pradeep Kumar Goud and Boppana Lakshmi, "High-speed point multiplication architecture over GF(2^m) for a class of trinomials," *The Journal of Supercomputing.* **(Under Review) (SCI-Indexed, Springer Nature)**

Bibliography

[1] P. K. G. Nadikuda and L. Boppana, "An area-efficient architecture for finite field inversion over gf (2m) using polynomial basis," *Microprocessors and Microsystems*, p. 104439, 2022.

[2] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

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

[4] L. Li and S. Li, "High-performance pipelined architecture of elliptic curve scalar multiplication overgf(2^m)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 4, pp. 1223–1232, 2015.

[5] B. Ansari and M. A. Hasan, "High-performance architecture of elliptic curve scalar multiplication," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1443–1453, 2008.

[6] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on fpga," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 16, no. 2, pp. 198–205, 2008.

[7] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in gf (2n)," in *Selected Areas in Cryptography: 5th Annual International Workshop, SAC98 Kingston, Ontario, Canada, August 17–18, 1998 Proceedings*. Springer, 2002, pp. 201–212.

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

[9] J. López and R. Dahab, "Fast multiplication on elliptic curves over gf (2 m) without precomputation," in *Cryptographic Hardware and Embedded Systems: First InternationalWorkshop, CHES99 Worcester, MA, USA, August 12–13, 1999 Proceedings 1*. Springer, 1999, pp. 316–327.

[10] B. Rashidi, "A survey on hardware implementations of elliptic curve cryptosystems," *arXiv preprint arXiv:1710.08336*, 2017.

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

[12] I. Hazmi, "Systolic design space exploration of eea-based inversion over binary and ternary fields," Ph.D. dissertation, 2018.

[13] Z. Yan, "Digit-serial systolic architectures for inversions over gf (2m)," in *2006 IEEE Workshop on Signal Processing Systems Design and Implementation*. IEEE, 2006, pp. 77–82.

[14] C.-Y. Lee and C. W. Chiou, "New bit-parallel systolic architectures for computing multiplication, multiplicative inversion and division in gf (2m) under polynomial basis and normal basis representations," *Journal of Signal Processing Systems*, vol. 52, no. 3, pp. 313–324, 2008.

[15] A. Ibrahim, T. F. Al-Somani, and F. Gebali, "Efficient scalable digit-serial inverter over gf(2m) for ultra-low power devices," *IEEE access*, vol. 4, pp. 9758–9763, 2016.

[16] A. Ibrahim and F. Gebali, "Scalable and unified digit-serial processor array architecture for multiplication and inversion over gf(2m)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 11, pp. 2894–2906, 2017.

[17] A. Ibrahim, T. F. Al-Somani, and F. Gebali, "New systolic array architecture for finite field inversion," *Canadian Journal of Electrical and Computer Engineering*, vol. 40, no. 1, pp. 23–30, 2017.

[18] U. Banerjee, C. Juvekar, A. Wright, A. P. Chandrakasan *et al.*, "An energy-efficient reconfigurable dtls cryptographic engine for end-to-end security in iot applications," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 42–44.

[19] I. Hazmi, F. Gebali, and A. Ibrahim, "High speed and low area complexity extended euclidean inversion over binary fields," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 408–417, 2019.

[20] B. Rashidi, "Efficient hardware structure for extended euclidean-based inversion over f 2m," *IET Computers & Digital Techniques*, vol. 13, no. 4, pp. 282–291, 2019.

[21] J.-H. Guo and C.-L. Wang, "Digit-serial systolic multiplier for finite fields gf (2m)," *IEE Proceedings-Computers and Digital Techniques*, vol. 145, no. 2, pp. 143–148, 1998.

[22] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in gf (2m) using normal bases," *Information and computation*, vol. 78, no. 3, pp. 171–177, 1988.

[23] G.-L. Feng, "A vlsi architecture for fast inversion in gf (2/sup m/)," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1383–1386, 1989.

[24] F. Rodríguez-Henríquez, N. Cruz-Cortes, and N. Saqib, "A fast implementation of multiplicative inversion over gf (2/sup m/)," in *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, vol. 1. IEEE, 2005, pp. 574–579.

[25] C. Rebeiro, S. S. Roy, D. S. Reddy, and D. Mukhopadhyay, "Revisiting the itoh-tsujii inversion algorithm for fpga platforms," *IEEE Transactions on very large scale integration (vlsi) systems*, vol. 19, no. 8, pp. 1508–1512, 2010.

[26] F. Rodríguez-Henríquez, G. Morales-Luna, N. A. Saqib, and N. Cruz-Cortés, "Parallel itoh–tsujii multiplicative inversion algorithm for a special class of trinomials," *Designs, Codes and Cryptography*, vol. 45, no. 1, pp. 19–37, 2007.

[27] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical modeling of the itoh-tsujii inversion algorithm for enhanced performance on k-lut based fpgas," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[28] ——, "Generalized high speed itoh–tsujii multiplicative inversion architecture for fpgas," *Integration*, vol. 45, no. 3, pp. 307–315, 2012.

[29] L. Li and S. Li, "Fast inversion in gf (2m) with polynomial basis using optimal addition chains," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[30] B. Rashidi, R. R. Farashahi, and S. M. Sayedi, "High-performance and high-speed implementation of polynomial basis itoh–tsujii inversion algorithm over gf (2 m)," *IET Information Security*, vol. 11, no. 2, pp. 66–77, 2017.

[31] J. Li, Z. Li, C. Xue, J. Zhang, W. Gao, and S. Cao, "A fast modular inversion fpga implementation over gf (2 m) using modified x 2n unit," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[32] L. Parrilla, A. Lloris, E. Castillo, and A. Garcia, "Minimum-clock-cycle itoh-tsujii algorithm hardware implementation for cryptography applications over gf (2m) fields," *Electronics letters*, vol. 48, no. 18, pp. 1126–1128, 2012.

[33] V. VR, N. Murali, and S. Rajaram, "An improved quad itoh-tsujii algorithm for fpgas," *IEICE Electronics Express*, vol. 10, no. 18, pp. 20 130 612–20 130 612, 2013.

[34] J. Hu, W. Guo, J. Wei, and R. C. Cheung, "Fast and generic inversion architectures over gf(2^m) using modified itoh–tsujii algorithms," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 4, pp. 367–371, 2015.

[35] V. Trujillo-Olaya and J. Velasco-Medina, "Hardware architectures for inversion in gf (2 m) using polynomial and gaussian normal basis," in *2010 IEEE ANDESCON*. IEEE, 2010, pp. 1–5.

[36] N. Takagi, J.-i. Yoshiki, and K. Takagi, "A fast algorithm for multiplicative inversion in gf (2/sup m/) using normal basis," *IEEE Transactions on Computers*, vol. 50, no. 5, pp. 394–398, 2001.

[37] S. Kumar, T. Wollinger, and C. Paar, "Optimum digit serial gf (2ˆ m) multipliers for curve-based cryptography," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1306–1311, 2006.

[38] C. H. Kim, C. P. Hong, and S. Kwon, "A digit-serial multiplier for finite field gf (2/sup m/)," *IEEE Transactions on very large scale integration (vlsi) systems*, vol. 13, no. 4, pp. 476–483, 2005.

[39] C. H. Kim, S. Kwon, and C. P. Hong, "A fast digit-serial systolic multiplier for finite field gf (2 m)," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, 2005, pp. 1268–1271.

[40] S. H. Namin, H. Wu, and M. Ahmadi, "Low-power design for a digit-serial polynomial basis finite field multiplier using factoring technique," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 441–449, 2016.

[41] L. Song and K. K. Parhi, "Low-energy digit-serial/parallel finite field multipliers," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 19, pp. 149–166, 1998.

[42] W. Tang, H. Wu, and M. Ahmadi, "Vlsi implementation of bit-parallel word-serial multiplier in gf (2/sup 233/)," in *The 3rd International IEEE-NEWCAS Conference, 2005.* IEEE, 2005, pp. 399–402.

[43] C.-H. Liu, C.-Y. Lee, and P. K. Meher, "Efficient digit-serial ka-based multiplier over binary extension fields using block recombination approach," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 8, pp. 2044–2051, 2015.

[44] M. Morales-Sandoval, C. Feregrino-Uribe, P. Kitsos, and R. Cumplido, "Area/performance trade-off analysis of an fpga digit-serial gf (2m) montgomery multiplier based on lfsr," *Computers & Electrical Engineering*, vol. 39, no. 2, pp. 542–549, 2013.

[45] P. Meher, "High-throughput hardware-efficient digit-serial architecture for field multiplication over gf (2^m)," in *2007 6th International Conference on Information, Communications & Signal Processing.* IEEE, 2007, pp. 1–5.

[46] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques.* Springer, 1985, pp. 417–426.

[47] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical modeling of elliptic curve scalar multiplier on lut-based fpgas for area and speed," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 901–909, 2012.

[48] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Superscalar coprocessor for high-speed curve-based cryptography," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 415–429.

[49] C. H. Kim, S. Kwon, and C. P. Hong, "Fpga implementation of high performance elliptic curve cryptographic processor over gf (2163)," *Journal of Systems Architecture*, vol. 54, no. 10, pp. 893–900, 2008.

[50] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient fpga implementations of point multiplication on binary edwards and generalized hessian curves using gaussian normal basis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1453–1466, 2011.

[51] C. Rebeiro, S. S. Roy, and D. Mukhopadhyay, "Pushing the limits of high-speed gf (2 m) elliptic curve scalar multiplication on fpgas," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 494–511.

[52] G. D. Sutter, J.-P. Deschamps, and J. L. Imaña, "Efficient elliptic curve point multiplication using digit-serial binary field operations," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 1, pp. 217–225, 2012.

[53] T. T. Nguyen and H. Lee, "Efficient algorithm and architecture for elliptic curve cryptographic processor," *JSTS: Journal of Semiconductor Technology and Science*, vol. 16, no. 1, pp. 118–125, 2016.

[54] R. Salarifard, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "A low-latency and low-complexity point-multiplication in ecc," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 9, pp. 2869–2877, 2018.

[55] J. Li, Z. Li, S. Cao, J. Zhang, W. Wang *et al.*, "Speed-oriented architecture for binary field point multiplication on elliptic curves," *IEEE Access*, vol. 7, pp. 32 048–32 060, 2019.

[56] A. P. Fournaris, N. Sklavos, and C. Koulamas, "A high speed scalar multiplier for binary edwards curves," in *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, 2016, pp. 41–44.

[57] L. Parrilla, J. A. Álvarez-Bermejo, E. Castillo, J. A. López-Ramos, D. P. Morales-Santos, and A. García, "Elliptic curve cryptography hardware accelerator for high-performance secure servers," *The Journal of Supercomputing*, vol. 75, no. 3, pp. 1107–1122, 2019.

[58] S. Harb and M. Jarrah, "Fpga implementation of the ecc over gf (2m) for small embedded applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 2, pp. 1–19, 2019.

[59] Z. U. Khan and M. Benaissa, "High-speed and low-latency ecc processor implementation over gf(2^m) on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 165–176, 2016.

[60] E. G. Thurber, "Efficient generation of minimal length addition chains," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1247–1263, 1999.

[61] C. Rebeiro and D. Mukhopadhyay, "Power attack resistant efficient fpga architecture for karatsuba multiplier," in *21st International Conference on VLSI Design (VLSID 2008)*. IEEE, 2008, pp. 706–711.

[62] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity analysis and efficient implementations of bit parallel finite field multipliers based on karatsuba-ofman algorithm on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1057–1066, 2009.

[63] M. Kalaiarasi, V. Venkatasubramani, and S. Rajaram, "A parallel quad itoh-tsujii multiplicative inversion algorithm for fpga platforms," in *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*. IEEE, 2020, pp. 31–35.

[64] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1250–1258, 2017.

[65] A. Zakerolhosseini and M. Nikooghadam, "Low-power and high-speed design of a versatile bit-serial multiplier in finite fields gf (2m)," *Integration*, vol. 46, no. 2, pp. 211–217, 2013.

[66] P. K. Meher, "Systolic and non-systolic scalable modular designs of finite field multipliers for reed–solomon codec," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 17, no. 6, pp. 747–757, 2009.

[67] B. Rashidi, R. R. Farashahi, and S. M. Sayedi, "High-speed and pipelined finite field bit-parallel multiplier over gf (2 m) for elliptic curve cryptosystems," in *2014 11th International ISC Conference on Information Security and Cryptology*. IEEE, 2014, pp. 15–20.

[68] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in gf (2 n)," in *International Workshop on Selected Areas in Cryptography*. Springer, 1998, pp. 201–212.

[69] X. Xiong and H. Fan, "Gf (2 n) bit-parallel squarer using generalised polynomial basis for new class of irreducible pentanomials," *Electronics Letters*, vol. 50, no. 9, pp. 655–657, 2014.

[70] B. Rashidi, S. M. Sayedi, and R. R. Farashahi, "High-speed hardware architecture of scalar multiplication for binary elliptic curve cryptosystems," *Microelectronics journal*, vol. 52, pp. 49–65, 2016.

[71] M. Imran, M. Rashid, and I. Shafi, "Lopez dahab based elliptic crypto processor (ecp) over gf (2 163) for low-area applications on fpga," in *2018 International Conference on Engineering and Emerging Technologies (ICEET)*. IEEE, 2018, pp. 1–6.

[72] M. Kashif and I. Cicek, "Field-programmable gate array (fpga) hardware design and implementation of a new area efficient elliptic curve crypto-processor," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 29, no. 4, pp. 2127–2139, 2021.

[73] P. K. G. Nadikuda and L. Boppana, "An area-time efficient point-multiplication architecture for ecc over gf (2m) using polynomial basis," *Microprocessors and Microsystems*, vol. 91, p. 104525, 2022.

Curriculum Vitae

Personal Information

Name: Pradeep Kumar Goud N

Address: H. No. 5 - 4 - 103/1, Bhavani Colony, Rajendranagar, Rangareddy Dt., Telanaga, India, Pin Code - 500030.

Ph: +91-7207726897 | **Email:**pradeep.nadikuda@gmail.com

Research Interests

VLSI architectures, Digital system design, Microprocessor systems, and Hardware implementation of finite field arithmetic and cryptographic algorithms.

Experience Summary

Assistant Professor, KL University, Hyderabad (2023-Present).

Assistant Professor, Geethanjali college of Engineering and Technology, Hyderabad (2022-2023).

Assistant Professor, ACE college of Engineering, Hyderabad (2016-2017).

Assistant Professor, Aurora group of Intuitions, Hyderabad (2012-16).

Education

Ph.D, National Institute of Technology Warangal, Telangana, India

M.Tech, Gokararju Rangaraju Institute of Engineering and Technology, Telangana, India

B.Tech, TKR Institute of Engineering & Technology, Telangana, India

---