High-Throughput and Area-Efficient Finite Field GF(2^m) Multiplier Architectures for Internet of Things Security Algorithms

Submitted in partial fulfilment of the requirements

for the award of the degree of

Doctor of Philosophy

by

Siva Ramakrishna P

(Roll No: 701623)

Under the supervision of

Dr. B. Lakshmi

Department of Electronics & Communication Engineering

National Institute of Technology Warangal

Telangana, India - 506004

2021

Dedicated

To

My Family,

Teachers & Friends

APPROVAL SHEET

The thesis entitled "**High-Throughput and Area-Efficient Finite Field GF(2^m) Multiplier Architectures for Internet of Things Security Algorithms**" by **Mr. Siva Ramakrishna P** is approved for the degree of Doctor of Philosophy.

Examiners

[illegible]

____

Prof. Shaik Rafi Ahamed
E.E.E Department,
I.I.T. Guwahati, INDIA

____

Supervisor

Dr. B. Lakshmi
Associate Professor,
E.C.E Department,
NIT, Warangal

____

Chairman

Prof. L. Anjaneyulu
Head of the E.C.E. department
NIT, Warangal

Date: **10-08 -2021**

Declaration

This is to certify that the work presented in this thesis entitled **High-Throughput and Area-Efficient Finite Field GF(2^m) Multiplier Architectures for Internet of Things Security Algorithms** is a bonafied work done by me under the supervision of **Dr. 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.

Siva Ramakrishna P

Research Scholar

Roll No.: 701623

Place:

Date:

NATIONAL INSTITUTE OF TECHNOLOGY

WARANGAL, INDIA-506004

Department of Electronics & Communication Engineering

CERTIFICATE

This is to certify that the thesis work entitled **High-Throughput and Area-Efficient Finite Field GF(2^m) Multiplier Architectures for Internet of Things Security Algorithms** is a bonafide record of work carried out by **Siva Ramakrishna P** 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.

Dr. B. Lakshmi
Research Supervisor
Associate Professor
Department of ECE
NIT Warangal, India-506 004.

Place:

Date:

Acknowledgements

First, I take immense pleasure to convey my sincere gratitude to my supervisor Dr. 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. L. Anjaneyulu, Prof. C. B. Rama Rao, Dr. P. Muralidhar, Prof. N. Subrahmanyam (Dept. of EEE), Prof. N. V. S. N. Sarma, Prof. T. Kishore Kumar, and Prof. N. Bheema Rao 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 Sri Chalavadi Mallikarjuna Rao (Founder President, PSCMR College of Engg. & Tech., Vijayawada), Sri Ravi Kishore Kodali, Dr. Sudha Ellison Mathe, Dr. Ashok Agarwal, Mr. Phani Kumar Vadali, Mr. Ramaswamy Venkateswarlu, Mr. Vijay Avinash Prathipati, Mr. Rambabu Kusuma, Mr. Apparao Kandula, Dr. Venkata Ramana Kollipara, Mr. Pradeep Kumar Nadikuda, Mr. Sai Kumar Aruru, Mr. S N SandeepRaju Pathapati, Mr. Venkata Rao Marpudi, Mr. Kiran Kumar Gowdiperu, and Mr. Narasimha Rao Bandi 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.

Siva Ramakrishna P

Abstract

Internet of Things (IoT) is a state of the art rapidly emerging communication technology with many application areas. It connects many physical objects such as domestic appliances, vehicles, and healthcare devices over its network apart from conventional computing devices such as desktop computers. The connectivity to these physical objects is made possible by equipping them with the low-end/constrained devices called IoT devices. The success of this new IoT technology depends on how securely the data is communicated over the network. However, security in IoT is a major concern and it must be addressed using various cryptography algorithms.

Cryptography deals with the study of encryption/decryption algorithms to transform messages into a hidden form to make them secure and immune to attacks. The security challenges posed by the emerging applications such as IoT prompt this conventional cryptography into a new direction, namely, lightweight cryptography. Lightweight cryptography is suitable for resource-constrained devices to make them secure in the network. Elliptic curve cryptography (ECC) is one such system requiring a shorter key length for the same level of security compared to other available cryptosystems. This cryptography heavily uses finite field GF(2^m) arithmetic in its underlying operations.

Finite field GF(2^m) is an algebraic structure with 2^m elements where arithmetic operations such as addition, multiplication, and inversion are defined. GF(2^m) multiplication is complex and also a performance-critical operation, hence, it requires efficient hardware implementations. GF(2^m) multipliers designed using polynomial basis gives more efficient, simple, and regular structures compared to other available bases. Various classes of irreducible polynomials can also be used for efficient implementation of GF(2^m) multipliers. Many efficient GF(2^m) multipliers 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 and time efficient hardware architectures for GF(2^m) multiplication targeting the implementation of security in IoT devices. Accordingly, some GF(2^m) multiplication algorithms or formulations are proposed based on the available algorithms in the literature and subsequently efficient multiplier architectures are realized for these proposed algorithms. Firstly, two bit-serial sequential multiplier architectures over GF(2^m) for general irreducible polynomials are proposed. One of these multipliers is based on the Interleaved modular reduction multiplication algorithm and the other is based on the Montgomery multiplication algorithm. Secondly, three bit-parallel systolic multiplier architectures are proposed based on the formulations developed for two specific classes of trinomials. These multipliers are realized by representing the developed formulations using signal flow graphs (SFGs) and applying suitable cutset pipelining techniques. Lastly, two digit-serial sequential multiplier architectures over a specific class of trinomials are proposed. The area and time complexities of all the proposed architectures are computed analytically for $m = 409$ using Silvaco's FreePDK NanGate 45nm standard gate estimations and the efficiency of these hardware architectures are verified by comparing them with the related available architectures in the literature. The HDL (Hardware description language) models of these proposed architectures are also implemented using Synopsys Design Compiler tool employing FreePDK NanGate 45nm technology libraries. It is observed from the comparison of the results that the proposed architectures outperform the existing architectures in terms of area and delay complexities. These proposed area and time efficient GF(2^m) multipliers may be used in the implementation of security in IoT devices.

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
LSD	Least Significant Digit
MUX	Multiplexer
ADP	Area-Delay-Product
GE	Gate Equivalent
SFG	Signal Flow Graph
PE	Processing Element
VLSI	Very Large Scale Integration
ASIC	Application Specific Integrated Circuit
VHDL	Very High Speed Integrated Circuit Hardware Description Language
RTL	Register Transfer Level
nm	nanometer
ns	nanosecond
μm^2	Square Micrometer
LFSR	Linear Feedback Shift Register

Chapter 1

Introduction

Internet of Things (IoT) is a state of the art rapidly emerging communication technology with many application areas. It can be considered as an extension to the Internet to include connectivity to constrained devices as well. IoT can connect many physical things such as domestic appliances, vehicles, smart wearables, health devices, and smart grids over the network apart from conventional resourceful computing devices such as desktop computers [1]. On this basis, many of the physical things/objects around us will be on the network. These physical objects are equipped with small computing devices that have connectivity capabilities which help to enable the objects to participate in communication over the network. These small computing devices are called IoT End devices. Many IoT end devices that are used under a specific application can be connected to relatively high-computing IoT devices called IoT Edge devices/IoT Gateways. This possibility of connecting end devices to an edge device facilitates a new computing paradigm called Edge computing [2]. The generic architecture for IoT edge computing which includes IoT end devices and IoT edge devices is shown in Fig. 1.1. This architecture demonstrates that a group of end devices, which are attached to the physical objects such as a projector or refrigerator, are connected to an edge device. Typically, end devices are battery-powered, low-cost, and have low computational resources while edge devices need to have high performance in terms of speed or throughput to accommodate connectivity to all its end devices [3]. The success of this new emerging IoT technology is majorly challenged by its security i.e. how securely Information or Data is communicated over the IoT network. Hence, it is indispensable to implement security features in IoT devices for

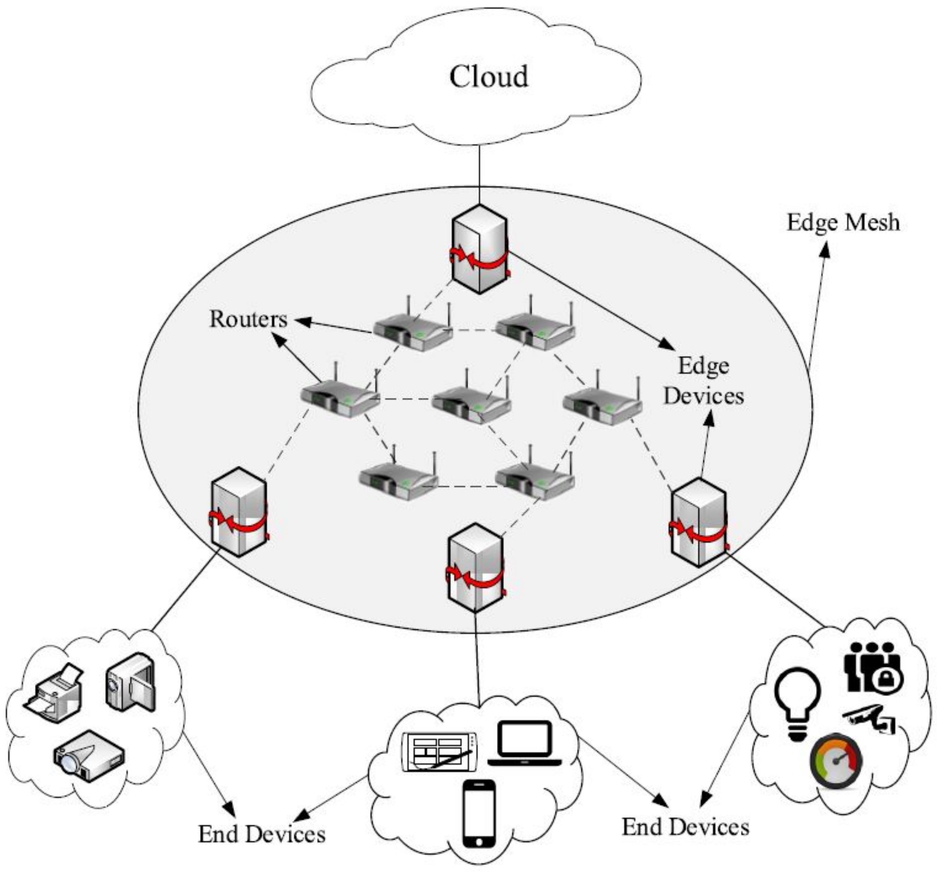

Figure 1.1 Generic IoT edge computing architecture with end and edge computing devices.

achieving secure communication and also to avoid many network-based attacks [4]. IoT devices are typically characterized by low hardware, hence, they have low computational power and small data bus width typically 8/16/32 bits. Moreover, these IoT devices must be cost-effective and also suitable for today's high data speeds [3]. Hence, implementation of security into these low-hardware and low-cost IoT devices is a major challenge to be addressed to make IoT widespread. Nevertheless, on the other hand, we have Cryptography which can be used to achieve some of the major security services including data confidentiality, authentication, non-repudiation, and data integrity in IoT devices.

Cryptography deals with the design and analysis of various data encryption and decryption algorithms. It can be broadly categorized into Private key cryptography and Public key cryptography. Private key cryptography uses the same key for both encryption and decryption. Examples for this cryptography include the algorithms such as Data encryption standard (DES), Advanced encryption standard (AES), and Twofish. On the other hand, public key cryptography employs a pair of related keys where one key is used for encryption and the other key is used for decryption. Examples for this cryptography include the algorithms such as RSA (Rivest-Shamir-Adleman), Elliptic curve Diffie-Hellman (ECDH) key exchange, and Elliptic curve digital signature algorithm (ECDSA) [5].

Conventional cryptography which is targeted for resourceful devices is not suitable for resource-constrained devices such as IoT devices. Moreover, the need for billions of IoT devices has prompted this cryptography in a new direction called Lightweight cryptography [6]. Many private-key lightweight cryptographic algorithms such as CLEFIA and PRESENT are evolved, while, elliptic curve cryptography, apart from being a conventional public key algorithm, remains the best candidate for public key lightweight cryptography because of its more security per key bit [7]. Hence, ECC (Elliptic curve cryptography)-based public key schemes such as ECDH and ECDSA are adopted widely to implement security in IoT devices.

Elliptic curve cryptography uses the elliptic curves that are defined over GF(p) or GF(2^m) finite fields. ECC over prime fields GF(p) is typically used for software implementations where applications may have general processors, however, resource-constrained devices such as IoT devices require hardware implementations. Moreover, hardware implementation of ECC is more practical for IoT applications as it provides efficient solutions in terms of area, delay, and power compared to software implementations. Furthermore, hardware implementation of ECC over binary fields GF(2^m) exhibit substantially lower hardware apart from low power and less delay compared to its prime field GF(p) counterparts [8,9]. Hence, hardware implementation of ECC using GF(2^m) fields is preferable for IoT applications [10–12]. 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 addition, point doubling, and scalar multiplication. The hierarchy of the arithmetic operations involved in ECC-based schemes is shown in Fig. 1.2. The performance of these schemes depends on the implementation of the low-level arithmetic operations, especially GF(2^m) multiplication [13]. Thus, the performance of applications that use ECC for implementing security can be improved by employing an efficient suitable GF(2^m) finite field multiplier.

A GF(2^m) finite field is an algebraic structure where arithmetic operations such as addition, multiplication, and inversion can be performed without leaving the structure.

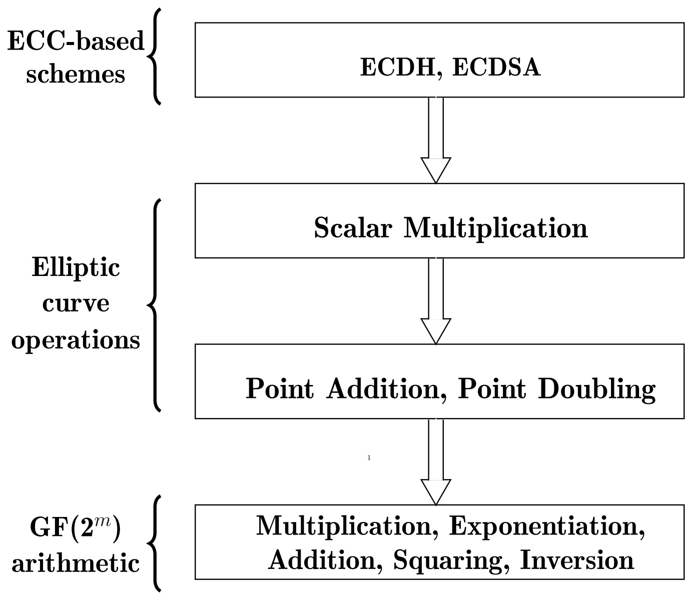
ECC-based schemes
ECDH, ECDSA
Elliptic curve operations
Scalar Multiplication
Point Addition, Point Doubling
GF(2^m) arithmetic
Multiplication, Exponentiation, Addition, Squaring, Inversion

Figure 1.2 ECC arithmetic hierarchy.

The finite field GF(2^m) has 2^m elements, where the field elements can be represented using various bases such as dual basis, normal basis, polynomial basis, and redundant basis. Polynomial basis is one of the bases recommended by many standard institutes including National Institute of Standards and Technology (NIST), and multipliers based on this basis are simpler, regular, and modular. In polynomial basis representation, the complexity of multiplication depends on field generating m^{th} degree polynomial called an irreducible polynomial [14]. There are various types of irreducible polynomials such as general irreducible polynomials, all one polynomials, pentanomials, and trinomials. The fields defined over general irreducible polynomials are suitable for general applications such as domestic IoT devices. Besides, standard institutes recommend sparse polynomials such as trinomials and pentanomials as they result in low hardware and time complexities. Hence, fields defined over trinomials are more suitable for high-performance applications such as Industrial IoT devices. In finite field GF(2^m) arithmetic, multiplication is the important and most frequent operation and is repeatedly used in other operations such as exponentiation and inversion. The GF(2^m) polynomial basis multiplication is defined as follows: Let $A(x)$ and $B(x)$ be the two field elements to be multiplied and $T(x)$ be the field irreducible polynomial. Then the polynomial basis is constituted by $(1, x, x^2, x^3, ..., x^{m-1})$, where x is the root of the irreducible polynomial $T(x)$. Let $C(x)$ be the finite field product of the two elements $A(x)$ and $B(x)$. Then GF(2^m) finite field multiplication of $A(x)$ and $B(x)$ is given by $C(x) = A(x)B(x) \bmod T(x)$, i.e. usual multiplication of $A(x)$ and $B(x)$ followed by modulo reduction using $T(x)$.

Many efficient GF(2^m) multiplication algorithms and architectures have been proposed in the literature to achieve reduction in area and time complexities. The GF(2^m) multiplication can be realized using various algorithms such as the Interleaved multiplication algorithm [15, 16], Karatsuba algorithm [17, 18], Montgomery algorithm [19, 20], and Mastrovito multiplication [21, 22], and can be implemented employing various architectural styles. Depending on the style of implementation, various architectures for finite field GF(2^m) multipliers can be developed. Based on the style of input/output, the architectures can be bit-serial, bit-parallel, and digit-serial. In bit-serial architectures, at least one input/output enters/generates serially while others can be parallel [23]. In bit-parallel architectures, all inputs and outputs appear in parallel [24]. Depending on the structure of the architecture various implementations such as sequential [16, 25], parallel [26], and systolic [27] can be developed. Sequential structures take less hardware at the expense of more computational delay. The output of a sequential structure is available only after more than one clock cycle, typically in m clock cycles for a GF(2^m) multiplication. Parallel structures generate output in a single clock cycle at the expense of excessive hardware. Systolic structures offer advantages such as regularity, modularity, concurrency, local interconnections, and are more suitable for VLSI (Very Large Scale Integration) implementation. These systolic structures can accommodate high throughput rates while their area and latency are usually very large. Digit-level [18] architectures process a group of bits, called a digit, at a time and these architectures can facilitate the area-delay trade-off. Bit-serial sequential implementations are of interest for IoT end devices, bit-parallel systolic implementations are desirable for IoT edge devices, and digit-level architectures are suitable for both IoT end/edge devices.

1.1 Motivation

Advances in computing and communication technologies have prompted the evolution of Internet of things (IoT). Internet of Things is a new computing environment where many constrained devices called IoT end and edge devices are connected to the Internet. It is estimated that more than 40 billion devices will be connected to the IoT network by 2022. Hence, there is a great demand for secure and low-cost IoT devices to transmit data securely and to avoid many network-based attacks. Elliptic curve cryptography (ECC) is an efficient public key cryptosystem that is used to achieve some of the security features in IoT devices. The constrained nature of IoT devices demands low-hardware and low-cost implementations of ECC and its underlying GF(2^m) finite field operations while maintaining adequate performance in terms of speed. Finite field GF(2^m) multiplication is the most performance-critical operation in ECC, hence, it requires efficient realizations and hardware implementations. Low-hardware and low-cost implementations can be achieved by designing area-efficient multipliers and high-performance implementations can be achieved by designing high-throughput multipliers. Further, scalable multipliers are also required to achieve area-delay trade-off which is required for a wide variety of IoT applications that require moderate performance. Also, the design of GF(2^m) finite field multipliers using polynomial basis gives more efficient architectures compared to the other available bases. Hence, it is necessary to design high-throughput and area-efficient polynomial basis GF(2^m) multipliers targeting IoT devices.

1.2 Research Objectives

The objective of this research is to design and implement efficient polynomial basis finite field GF(2^m) multipliers to improve the performance of IoT security algorithms.

- Due to the wide range of application areas, IoT end devices must be cost-effective and are desirable to be available as generic off-the-shelf components. Hence, it is required to design area-efficient bit-serial sequential GF(2^m) multipliers using general irreducible polynomials. It is also required to verify the performance of these multipliers using analytical and ASIC (Application Specific Integrated Circuit) implementation comparisons.
- Many IoT end devices working under a specific application can be connected to an IoT edge device for Edge computing. The performance of IoT edge devices must

be high in terms of speed or throughput while having moderate area complexities. Hence, it is required to design high-throughput, low-latency, and area-efficient systolic GF(2^m) multipliers. It is also required to verify the performance of these multipliers using analytical and ASIC implementation comparisons.

- In addition to low-end (cost-effective) and high-end (high-performance) IoT devices, many applications need middle-end IoT devices that have performance and cost requirements in between to low-end and high-end devices. Scalable architectures such as digit-serial architectures are suitable for implementing these middle-end IoT devices. Hence, it is required to design efficient (high-throughput or low-hardware) digit serial multipliers and also to verify the performance of these multipliers through analytical and ASIC implementation comparisons.

1.3 Thesis Contributions

The contributions of the thesis are summarized as follows:

- **Area-Efficient Bit-Serial Sequential Finite Field GF(2^m) Multipliers for General Irreducible Polynomials** Two bit-serial sequential multiplier architectures using polynomial basis that perform multiplication of any two finite field elements for any irreducible polynomial are proposed. The performance of these proposed architectures is evaluated through theoretical analysis and ASIC implementations. The contributions of this work are briefly described as:
  - **Proposed Area-Efficient Bit-Serial Sequential Polynomial Basis GF(2^m) Multiplier** In this work, a modified interleaved modular reduction multiplication algorithm over general irreducible polynomials and its realization using a bit-serial sequential architecture are presented. The modification in the algorithm involves employing more efficient logical relations to achieve reduction in hardware complexities. The proposed architecture achieves a minimum reduction of about 31% in area and 5% in ADP (Area-Delay-Product) compared to the previous works for the field of order $m = 409$. The ASIC implementation of the proposed architecture indicates a minimum reduction of about 28% in

area and 3% in ADP compared to the existing works.

- **Proposed Low-Complexity Bit-Serial Sequential Polynomial Basis Montgomery GF(2^m) Multipliers** In this work, we propose modified MSB (most significant bit)-first and LSB (Least significant bit)-first algorithms for Montgomery multiplication. These proposed modified algorithms are realized using bit-serial sequential architectures. The proposed MSB architecture involves less area and time complexities and achieves a minimum reduction of about 16% in ADP compared to the previous works for the field of order $m = 409$. Further, the ASIC implementation of this proposed architecture indicate a minimum reduction of about 12% in ADP compared to the existing works. Similarly, the proposed LSB architecture involves less area and time complexities and achieves a minimum reduction of about 13% in ADP compared to the previous works for the field of order $m = 409$. Further, the ASIC implementation of this proposed architecture indicate a minimum reduction of about 11% in ADP compared to the existing works.

• **Low-Latency and High-Throughput Bit-Parallel Systolic Finite Field GF(2^m) Multipliers for Specific Classes of Trinomials** Three polynomial basis systolic multiplier architectures that perform multiplication of any two finite field elements defined over specific classes of trinomials are proposed. The performance of these proposed architectures is evaluated through theoretical analysis and ASIC implementations. The contributions of this work are briefly described as:

  - **Proposed Area-Efficient Low-Latency Bit-Parallel Systolic Polynomial Basis GF(2^m) Multiplier** In this work, we develop formulations for GF(2^m) multiplication applicable for a class of trinomials for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even), where k is the degree of the middle term of trinomial and m is the order of the field GF(2^m). Based on the developed formulations we present a systolic architecture for the finite field GF(2^m) multiplication. The proposed multiplier achieves a minimum reduction of about 11% in area complexity and 14% in latency compared to previous works for the field of order $m = 409$. Also, the ASIC implementation of the proposed multiplier indicate a minimum reduction of about 9% in area and

12% in latency compared to the existing works.

- **Proposed High-Throughput Area-Delay-Efficient Bit-Parallel Systolic Polynomial Basis GF(2^m) Multiplier** This work presents a high-throughput systolic multiplier architecture based on the formulations proposed for the above area-efficient low-latency multiplier. Thus, this proposed multiplier is also applicable for the class of trinomials for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). The proposed multiplication method is realized using a systolic architecture where efficient cutset pipelining techniques are applied to the derived signal flow graph (SFG) to reduce time complexities. The proposed multiplier achieves a minimum increase of about 42% in throughput rate and a minimum reduction of about 6% in ADP compared to previous works for the field of order $m = 409$. Also, the ASIC implementation of the proposed multiplier indicate a minimum increase of about 35% in throughput and a minimum reduction of about 5% in ADP compared to the existing works.

- **Proposed Low-Latency Area-Efficient Bit-Parallel Systolic Polynomial Basis GF(2^m) Multiplier** In this work, we propose a new area-efficient and low-latency GF(2^m) systolic multiplier applicable for a narrow class of trinomials for which $k \leq m-2\lceil m/3 \rceil$ which includes both the NIST recommended trinomials for $m = 233$ and 409 fields. The proposed multiplier achieves a minimum reduction of about 6% in area complexity and a 32% in latency compared to previous works for the field of order $m = 409$. Also, the ASIC implementation of the proposed multiplier indicate a minimum reduction of about 4% in area and 25% in latency compared to the existing works.

• **High-Throughput and Low-hardware Digit-Serial Multipliers for a Specific Class of Trinomials** Two digit-serial polynomial basis multiplier architectures that perform multiplication of any two finite field elements defined over a specific class of trinomials are proposed. The performance of these proposed architectures is evaluated through theoretical analysis and ASIC implementations. The contributions of this work are briefly described as:

- **Proposed High-Throughput Digit-Serial Sequential Polynomial Basis**

GF(2^m) Multiplier In this work, a digit-serial GF(2^m) multiplication algorithm is proposed and the corresponding architecture is also presented. The proposed algorithm is based on a redundant basis digit-serial multiplication algorithm available in the literature. This available redundant basis multiplication algorithm is modified to work for polynomial basis multiplication. The proposed modified algorithm is suitable for polynomial basis GF(2^m) multiplication over a class of trinomials for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even), where k is the degree of the middle term of trinomial and m is the order of the field GF(2^m). The proposed digit-serial multiplier takes both the operands simultaneously digit-wise to perform computation. The proposed multiplier achieves reduction in critical path delay and ADP for the field of order $m = 409$. Also, the ASIC implementation of the proposed multiplier indicates a minimum increase of about 26% in throughput compared to the existing works.

– **Proposed Low-Hardware Digit-Serial Sequential Polynomial Basis GF(2^m) Multiplier** In this work, a new formulation for the digit-serial finite field multiplication over the class of trinomials for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even) and its hardware structure are presented. Based on a available Mastrovito multiplier, an optimized parallel multiplier for the considered class of trinomials is designed and employed in the proposed digit-serial multiplier architecture. The proposed multiplier achieves a marginal reduction of area and ADP compared to previous works for the field of order $m = 409$. Also, the ASIC implementation of the proposed multiplier indicates a minimum reduction of about 3% in ADP compared to the existing works.

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 finite field multiplication operation. It also presents a few available multiplication algorithms and examples.

Chapter 3 presents the review of the architectures proposed in the literature for polynomial basis GF(2^m) multiplication. It presents the available bit-serial sequential multiplier architectures for general irreducible polynomials followed by the available bit-parallel systolic multiplier architectures for trinomials. The chapter also presents the review of the available digit-serial sequential multipliers for trinomials. The review also includes detailed discussions on the performance of these architectures in terms of area complexity, latency, throughput, and critical path delay.

Chapter 4 presents a modified interleaved modular reduction multiplication algorithm and a bit-serial sequential architecture over GF(2^m) for general irreducible polynomials. This chapter also presents the design of two bit-serial sequential Montgomery multipliers over GF(2^m) for general irreducible polynomials using modified Montgomery algorithms. Analysis and ASIC implementations followed by a comparison of results with existing works are presented.

Chapter 5 presents the design of three bit-parallel systolic multipliers over GF(2^m) using two specific classes of trinomials. Analysis and ASIC implementations followed by a comparison of results with existing works is also presented.

Chapter 6 presents the design of two digit-serial sequential multipliers over GF(2^m) using a specific class of trinomials. Analysis and ASIC implementations 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 GF(2^m) multiplication operation along with a few available multiplication algorithms.

Chapter 2

Finite Field GF(2^m) Multiplication

This chapter presents a brief overview of some mathematical concepts about 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 finite field GF(2^m) arithmetic operations. Finally, GF(2^m) multiplication operation over polynomial basis is presented along with a few multiplication algorithms to describe the operation in detail.

2.1 Finite Fields

This section presents the definitions and properties of Groups, Rings, and Fields. Further, it also presents the definitions of finite field, binary finite field GF(2^m), irreducible polynomial, and various bases.

2.1.1 Groups

Definition 1. A Group denoted by (G, $*$) is a set of elements G along with a binary operator $*$, such that, for any a, $b \in$ G, the result of the group operation between a and b must be in G i.e. $a * b \in$ G, and it satisfies the following properties:

(1) Identity - There is an element e in G, such that for every $a \in$ G, $e*a = a*e = a$.

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

(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 $a + (b + c) \equiv (a + b) + c \bmod n$.

Further, on the other hand, it may be observed that the set $\mathbb{Z}_n$ does not have multiplicative inverses for all its elements and hence is not a group under multiplication modulo n operation.

(2) The set of integers $\mathbb{Z}$ under addition is a group with identity element 0.

(3) The set of real numbers $\mathbb{R}$ is

i) a group under the addition operation with identity element 0,

ii) a group under the multiplication operation with identity element 1.

A group (G, $*$) is said to be "abelian" if $a * b = b * a$ for every a, b $\in$ G. The above three examples are abelian groups.

2.1.2 Rings

Definition 2. A Ring (R, +, ×) is a set R which is closed under two operations + and ×, and satisfying the following properties:

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

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

(3) The operation × obeys distributive law over + operation, i.e., for every $a, b, c \in$ R, the following identities hold: $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = b \times a + c \times a$.

Examples:

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

(2) Another example is the set of integers $\mathbb{Z}$ along under the usual addition and multiplication operations can be considered as a ring $(\mathbb{Z}, +, \times)$.

A ring is said to be a ”commutative ring” if the operation × obeys 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 (F, +, ×) is a set F which is closed under two operations + and ×, such that

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

(2) F-{0} (the set F without the additive identity 0) is an abelian group under ×.

Examples:

(1) Some examples of fields are the set of all real numbers $\mathbb{R}$, set of all complex numbers $\mathbb{C}$, and the set of all rational numbers $\mathbb{Q}$.

2.1.4 Finite Fields

Finite fields or Galois fields are the fields that have a finite number of elements. The number of elements in a field is called the order of the field.

Finite fields are of two types.

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, 0 is the additive identity and 1 is the multiplicative 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}).

Thus, GF(2^m) is a binary extension finite field which is generated using the base field GF(2), where GF(2) is the finite field with two elements 0 and 1. The elements of binary finite fields GF(2^m) can be represented with the polynomials of degree less than

m over GF(2) i.e. the coefficients of the polynomials come from the base field GF(2). Let $A(x)$ be an arbitrary element of the field GF(2^m), thus, it can be represented as a polynomial of degree $(m-1)$, given by

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

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

Moreover, every GF(2^m) is characterized by its field-defining m^{th} degree polynomial called an irreducible polynomial. An irreducible polynomial of the finite field GF(2^m) is an m^{th} degree monic polynomial which cannot be factored into two non-trivial polynomial elements over the same field. For a GF(2^m), the general form of the irreducible polynomial $T(x)$ is given by a monic polynomial of the form, $T(x) = x^m + \sum_{j=1}^{m-1} t_j x^j + 1$ with at least one of t_js to be non zero and all $t_j \in$ GF(2).

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

$$\text{GF}(2^m) = \{A(x) \,|\, A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + a_{m-3}x^{m-3} + ... + a_1 x + a_0; \\ \forall\, a_i \in \text{GF}(2), i = m-1 \text{ to } 0\}, \tag{2.2}$$

where, x is a root of the irreducible polynomial $T(x)$.

Field irreducible polynomials $T(x)$ are categorized into general irreducible polynomials, all one polynomials, equally spaced polynomials, pentonomials, and trinomials. General irreducible polynomials bears no specific constraints on the structure of the polynomial and are of the form given by

$$T(x) = x^m + t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + t_{m-3}x^{m-3} + ... + t_1 x + 1; \\ \forall\, t_i \in \text{GF}(2), i = m-1 \text{ to } 1 \tag{2.3}$$

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

$$x^m = t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + t_{m-3}x^{m-3} + ... + t_1 x + 1$$

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

$$T(x) = x^m + x^{m-1} + x^{m-2} + x^{m-3} + ... + x + 1 \tag{2.4}$$

further, since x is the root of $T(x)$, 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

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

also, since x is the root of $T(x)$, 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,

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

moreover, since x is the root of $T(x)$, 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,

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

further, since x is the root of $T(x)$, one can also have $x^m = x^k + 1$. Examples for trinomials include the trinomials $T(x) = x^{233} + x^{74} + 1$ and $T(x) = x^{409} + x^{87} + 1$ (which are also the NIST (National Institute of Standards and Technology) recommended trinomials for elliptic curve cryptography).

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, any element of a field GF(2^m) can be represented as a linear combination of the m basis elements. A finite field can have more than one basis, thus, the elements of a finite field GF(2^m) can be represented using various bases such as polynomial basis, normal basis, redundant basis, dual basis, and weakly dual basis.

Polynomial Basis: The polynomial basis is defined with the set $(1, x, x^2, x^3, ...$ $.., x^{m-2}, x^{m-1})$ where x is the root of the irreducible polynomial $T(x)$ of the field GF(2^m).

An element $A(x) \in$ GF(2^m) represented using the polynomial basis is of the form,

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0, \quad a_i \in \text{GF}(2)$$

Dual Basis: Let $(x_0, x_1, x_2, x_3, ..., x_{m-1})$ and $(y_0, y_1, y_2, y_3, ..., y_{m-1})$ be the bases of a field GF(2^m). Then, the bases are said to be dual to each other if they satisfy the following condition.

$$Tr(x_iy_j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{2.8}$$

where $Tr(\alpha)$ (in Eq. 2.8, $\alpha = x_iy_j$) is a trace function defined over GF(2^m) as

$$Tr(\alpha) = \sum_{j=0}^{m-1} x^{2^i} \tag{2.9}$$

Weakly Dual Basis: Let $(x_0, x_1, x_2, x_3, ..., x_{m-1})$ and $(y_0, y_1, y_2, y_3, ..., y_{m-1})$ be the bases of a field GF(2^m) and $\gamma \in$ GF(2^m), $\gamma \neq 0$. Then, the bases are said to be weakly dual to each other if they satisfy the following condition.

$$Tr(\gamma x_iy_j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{2.10}$$

Clearly, for $\gamma = 1$, weakly dual basis becomes the dual basis. Thus, the dual basis is a special case of weakly dual basis.

Normal Basis: The normal basis is defined with the set $(x, x^2, x^{2^2}, x^{2^3}, ..., x^{2^{m-2}}, x^{2^{m-1}})$ where x is the root of the irreducible polynomial $T(x)$ of the field GF(2^m). An element $A(x) \in$ GF(2^m) represented using the normal basis is of the form,

$$A(x) = a_{m-1}x^{2^{m-1}} + a_{m-2}x^{2^{m-2}} + ... + a_1x^2 + a_0x, \quad a_i \in \text{GF}(2)$$

Redundant Basis: The splitting field of the polynomial $x^n - 1$ is known as a cyclotomic field which is denoted by $K^{(n)}$. Let β be the n^{th} roots of unity. The field $K^{(n)}$ can be generated by β over K, and an element of $K^{(n)}$ is given by

$$A(\beta) = a_0 + a_1\beta + a_2\beta^2 + ... + a_{n-1}\beta^{n-1}, \text{where, all } a_i \in K \tag{2.11}$$

Thus, the set $(1, \beta, \beta^2, \beta^3, ..., \beta^{n-1})$ forms the basis for the cyclotomic field $K^{(n)}$. This basis also forms the basis for any subfield of $K^{(n)}$ and is called redundant basis.

2.2 Finite Field GF(2^m) Arithmetic

Arithmetic operations such as addition, multiplication, inversion, squaring, division, and exponentiation are defined in the GF(2^m) field. Conventionally, these field operations can be performed in two steps, usual arithmetic operation followed by modulo reduction using the irreducible polynomial, $T(x)$. Moreover, both the steps follow modulo 2 arithmetic for operations among the polynomial coefficients.

However, GF(2^m) addition is simple and can also be performed as modulo 2 addition of polynomials. For example, consider 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. Then,

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

2.3 Finite Field GF(2^m) Multiplication

Among all the arithmetic operations, multiplication requires more attention as it is frequently used in the realization of other operations. Further, polynomial basis multiplication 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.

Let $A(x)$ and $B(x)$ be two elements of the field GF(2^m) represented using polynomial basis as

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + a_{m-3}x^{m-3} + \ldots\ldots + a_1x + a_0; \qquad \forall\ a_i \in \text{GF}(2), i = m-1 \text{ to } 0 \quad (2.12)$$

$$B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + b_{m-3}x^{m-3} + \ldots\ldots + b_1x + b_0; \qquad \forall\ b_i \in \text{GF}(2), i = m-1 \text{ to } 0 \quad (2.13)$$

Then, the GF(2^m) multiplication is defined as

$$C(x) = A(x)B(x) \bmod T(x) \quad (2.14)$$

The product $C(x)$ can be obtained by usual multiplication of the polynomials $A(x)$ and $B(x)$, followed by modulo reduction using $T(x)$ (This is called Classical method of computing finite field multiplication).

Example: Consider a finite field GF(2^8) with the field irreducible polynomial $T(x) = x^8+x^4+x^3+x+1$. Also consider two elements of this field as $A(x) = x^6+x^4+x^2+x+1$ and $B(x) = x^7+x+1$. Then, multiplication of these two elements $C(x)$ can be obtained as follows.

$$C(x) = A(x) \times B(x) \bmod T(x)$$

$$\begin{aligned} A(x) \times B(x) &= (x^6+x^4+x^2+x+1) \times (x^7+x+1) \\ &= x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 \end{aligned}$$

This result of the usual multiplication $(A(x) \times B(x))$ must be modulo reduced using the irreducible polynomial $T(x) = x^8+x^4+x^3+x+1$ as follows.

$$\begin{array}{rl}
x^8+x^4+x^3+x+1 \big) & x^{13}+x^{11}+x^9+x^8 \qquad +x^6+x^5+x^4+x^3+1 \ \big(\ x^5+x^3 \\
& x^{13} \qquad +x^9+x^8 \qquad +x^6+x^5 \\
\cline{2-2}
& \quad x^{11} \qquad\qquad\qquad\qquad +x^4+x^3+1 \\
& \quad x^{11} \qquad\qquad +x^7+x^6 \quad +x^4+x^3 \\
\cline{2-2}
& \qquad\qquad\qquad x^7+x^6 \qquad\qquad +1
\end{array}$$

Therefore, $C(x) = A(x)B(x) \bmod T(x) = x^7+x^6+1$.

In the classical approach mentioned in the above example, multiplication of polynomials is performed first, and then, it is followed by the reduction using the irreducible polynomial $T(x)$. Further, it is also possible to interleave the modular reduction step as shown in algorithm 2.1 for computing the required product, $C(x)$. This method is based on the observation that $A(x)B(x) \bmod T(x) = a_{m-1}(x^{m-1}B(x) \bmod T(x)) + ... + a_1(xB(x) \bmod T(x)) + B(x) \bmod T(x)$. Thus, $x^iB(x) \bmod T(x)$ can be successively computed for all $0 \leq i \leq m-1$, and all the reults are added for which $a_i = 1$. The successive computation can be started with $xB(x) \bmod T(x)$ (for $i = 1$), and it can be computed as follows using the propoerty of the irreducible polynomial $T(x)$ that $x^m = t(x) = t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + t_{m-3}x^{m-3} + \ldots\ldots + t_1x + 1$.

$$\begin{aligned} xB(x) \bmod T(x) &= b_{m-1}x^m + b_{m-2}x^{m-1} + ... + b_1x^2 + b_0x \bmod T(x) \\ &= b_{m-1}\big(t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + t_{m-3}x^{m-3} + \ldots\ldots + t_1x + 1\big) + b_{m-2}x^{m-1} \\ &\quad + ... + b_1x^2 + b_0x \end{aligned}$$

Therefore, $xB(x) \bmod T(x)$ can be computed by a left-shift of the vector representation of $B(x)$ (which invariably consists m coordinates) and adding $t(x)$ to it if $b_{m-1} = 1$. Similarly, other terms $x^iB(x \bmod T(x)$ can be successively obtained using previous terms and can be used in the computation of $C(x)$ as described in the algorithm.

Algorithm 2.1: Interleaved modular reduction multiplication algorithm

Input: $A = \sum_{j=0}^{m-1} a_jx^j$, $B = \sum_{j=0}^{m-1} b_jx^j$ both w.r.t. polynomial basis, and $T = x^m + \sum_{j=1}^{m-1} t_jx^j + 1$.

Output: $C = (A \times B) \bmod T = \sum_{j=0}^{m-1} c_jx^j$ also w.r.t. polynomial basis.

```
1: if $a_0 = 1$ then
2: $C(x) \leftarrow B(x)$
3: else
4: $C(x) \leftarrow 0$
5: end if
6: for $i = 1$ to $m - 1$ do
7: $B(x) \leftarrow B(x)x \bmod T(x)$
8:    if $a_i = 1$ then
9:      $C(x) \leftarrow B(x) + C(x)$
10:   end if
11: end for
12: return $C(x)$
```

Apart from the above mentioned classical (polynomial multiplication followed by modulo reduction) and Interleaved modulo reduction multiplication methods for computing the finite field GF(2^m) multiplication, various other algorithms/methods proposed in the literature such as Mastrovito multiplication, Montgomery multiplication, and Karatsuba-Ofman multiplication can also be used for efficient computation of multiplication. Further, the computational complexity of multiplication can also be reduced using various classes of irreducible polynomials, particularly, trinomials.

2.4 Conclusions

In this chapter, a brief overview of the fundamental concepts of groups, rings, fields, finite fields, polynomial basis representation, and the description of polynomial basis multiplication is presented. The next chapter presents the review of finite field multiplication architectures over GF(2^m) available in the literature.

Chapter 3

Polynomial Basis GF(2^m) Multiplier Architectures

Hardware implementation of IoT (Internet of Things) devices typically require low-cost, high-performance, and scalable multipliers as mentioned in Chapter 1. Low-cost implementations can be achieved using bit-serial sequential multipliers while high-throughput implementations can be achieved using bit-parallel systolic multipliers. Moreover, scalable multipliers that facilitate area-delay trade-off can be realized using digit-serial sequential multipliers. This current chapter presents the survey of different related architectures proposed in the literature for polynomial basis GF(2^m) multiplication. Firstly, the bit-serial sequential multipliers proposed in the literature over GF(2^m) for general irreducible polynomials are presented. Secondly, the bit-parallel systolic multipliers proposed in the literature over GF(2^m) for irreducible trinomials are presented. Finally, the digit-serial sequential multipliers proposed in the literature over GF(2^m) for irreducible trinomials are presented. In addition, the performance improvements achieved by these multipliers in terms of area complexity, latency, throughput, and critical path delay are also presented.

3.1 Review of Bit-Serial Sequential Multipliers for General Irreducible Polynomials

Several bit-serial sequential multipliers proposed in the literature for the finite field multiplication over GF(2^m) for general irreducible polynomials are reviewed and the performance of these multipliers in terms of area and time complexities is presented in Table 3.1. The area and time complexities of the multipliers are expressed using the notations m, T_A, T_{NA}, T_X, T_{XN}, T_{tsb}, and T_M which represent field order, the propagation delays of a 2-input AND gate, 2-input NAND gate, 2-input XOR gate, 2-input XNOR gate, tristate buffer, and 2-to-1 MUX (multiplexer), respectively. Furthermore, these notations are used to compute areas and delays of all the architectures presented in this thesis.

Table 3.1 Area and time complexities of the available bit-serial sequential multipliers.

Multiplier	AND	XOR	MUX	Register	Latency	Critical path
[28]	$2m$	$2m-1$	0	$4m+2$	$m+1$	T_A+T_X
MSB-first [14]	$2m$	$2m-1$	0	$3m$	m	T_A+2T_X
LSB-first [14]	$2m$	$2m-1$	m	$3m$	m	T_A+T_X
[25]	$(m^2+m)/2$	$(m^2+m)/2$	$4m$	$5m-1$	$2k_t^a+1$	$T_X\lceil log_2 m\rceil+2T_M+T_A$
[29]	$4m$	$2m$	$(m-1)^b+m^c+m^d$	$3m$	m	$T_A+T_X+T_{tsb}$
[30]	0	$6m+18$	$14m+26$	$6m+7$	$m/4$	$2T_M+4T_X$
[16]	$2m$	$2m$	$2m$	$3m$	m	T_A+T_X
MSB-first [31]	$2m-1$	$2m-1$	0	$2m$	m	T_A+T_X
LSB-first [31]	$2m-1$	$2m-1$	0	$2m$	m	T_A+2T_X
[32]	$2m-1$	$2m-1$	0	$2m$	m	T_A+T_X
[33]	$2m-1$	$2m-1$	0	$2m$	m	T_A+2T_X
[34]	m	$2m-1$	0	$2m$	m	$T_A+\lceil\log_2 m\rceil T_X$

[a]the second highest degree of the irreducible polynomial, [b]OR gates, [c]inverters, [d]tristate buffers

Various algorithms such as Interleaved multiplication algorithm [14], Montgomery multiplication algorithm [35], and Karatsuba-Ofman multiplication algorithm [36] have been proposed in the literature to realize efficient bit-serial multiplier architectures. Based on these algorithms, many architectures have been proposed in the literature to achieve reduction in area and time complexities. Beth et al. [37] in 1989 presented a most significant bit (MSB) first and a least significant bit (LSB) first bit-level serial-in parallel-out architectures using trinomials. These architectures use a linear feedback shift register with feedback added into the register in the positions defined by the irreducible polynomial. In these architectures, one input is loaded in parallel and another one is loaded serially one bit per clock cycle requiring a total of m clock cycles for a single multiplication. Song et al. [28] in 1996 presented a new polynomial basis bit-serial multiplier which achieves smaller critical path delay and lower latency. Further, this multiplier, using sub-structure sharing technique, can also achieve hardware saving when it is used in large systems. This multiplier requires $2m$ AND gates, $(2m-1)$ XOR gates, and $(4m+2)$ registers. The latency of this multiplier is $(m+1)$ clock cycles and the critical path delay is given by (T_A+T_X). In 2001, Johann Großschädl [38] presented a low power MSB-first bit-serial architecture. This multiplier also performs addition operation and operates over a wide range of finite fields. Major advantages of this multiplier are low power and versatility in terms of being applicable to a wide range of finite field orders, m.

Deschamps et al. [14] in 2009 presented implementation of bit-serial multipliers for general irreducible polynomials using the available [37] MSB-first and LSB-first interleaved multiplication algorithms. The MSB-first multiplier requires $2m$ AND gates, $(2m-1)$ XOR gates, and $3m$ registers. The latency of this multiplier is m clock cycles and the critical path delay is given by $(T_A + 2T_X)$. The LSB-first multiplier requires $2m$ AND gates, $(2m-1)$ XOR gates, m multiplexers, and $3m$ registers. The latency of this multiplier is also m clock cycles and the critical path delay is given by $(T_A + T_X)$. Both of these multipliers require less number of registers compared to the previously available multiplier [28], however, the MSB multiplier requires more critical path delay while LSB multiplier requires the same critical path delay.

In 2010, Imaña [25] proposed a new low latency parallel-in/parallel-out sequential polynomial basis multiplier over GF(2^m). It is a partially versatile multiplier as its datapath can also be used for fields GF(2^n) such that $1 \leq n \leq m$. This multiplier achieves low latency when the condition $m \geq 2k_t - 1$ is met, where k_t is the second highest degree of the field irreducible polynomial. This condition is specifically important as the NIST (National Institute of Standards and Technology) recommended five binary fields verify this condition. This multiplier requires $(m^2+m)/2$ AND gates, $(m^2+m)/2$ XOR gates, $4m$ MUXes and $(5m-1)$ registers. The latency of this multiplier is $(2k_t+1)$ clock cycles and the critical path delay is given by $(T_X\lceil log_2 m\rceil + 2T_M + T_A)$. Though this multiplier achieves low latency, it requires more hardware and critical path delay compared to the previously available multipliers [14,28]. Although this multiplier may achieve low latency, due to the high area complexity it is not useful for low-hardware applications.

A low-power and high-speed bit-serial versatile multiplier [29] was proposed by Zakerolhosseini et al. in 2013 which is flexible with field size as well as field irreducible polynomial. This multiplier uses tri-state buffers to achieve power reduction and it also achieves lower area and critical path delays. This multiplier requires $4m$ AND gates, $2m$ XOR gates, $(m-1)$ OR gates, m inverters, m tristate buffers, and $3m$ registers. The latency of this multiplier is m clock cycles and the critical path delay is given by

$T_A + T_X + T_{tsb}$. This multiplier achieves low power while requiring more area and delay complexities compared to the available multipliers [14, 28].

A sequential multiplier was proposed by Ho [30] in 2013 based on the condition $m \geq k_t + 4$, where k_t is the second highest degree of the irreducible polynomial. This architecture requires $(6m + 18)$ XOR gates, $(14m + 26)$ MUXes and $(6m + 7)$ registers. The critical path delay is given by the expression $(2T_M + 4T_X)$ with a latency of $m/4$ clock cycles. This multiplier achieves reduction in latency compared to the sequential multipliers [14, 28, 29], however, it requires high area complexity and critical path delay compared to these sequential multipliers. Also, this multiplier requires low latency for certain field orders including $m = 233$ for the NIST recommended trinomial, when compared to the multiplier [25]. Although this multiplier has achieved low latency, due to the high area complexity it is not useful for low-hardware applications.

Mathe et al. [16] presented an LSB-first sequential polynomial basis multiplier for generic irreducible polynomials with a latency of m clock cycles. This architecture is designed to take one operand in parallel and another operand serially during computation. This multiplier requires $2m$ AND gates, $2m$ XOR gates, $2m$ MUXes, and $3m$ registers. The latency of this multiplier is m clock cycles and the critical path delay is given by $(T_A + T_X)$. This multiplier slightly requires more hardware when compared to the LSB-first multiplier [14], however, this multiplier achieves more regularity as it does not require shifting for any of its registers.

Two bit-serial Montgomery multipliers [31] namely MSB-first multiplier and LSB-first multiplier using general irreducible polynomials were presented by Hariri et al. in 2009. In this work, the authors studied the role of Montgomery factor and identified that the appropriate factor as x^{m-1}. The Montgomery multipliers proposed in this work are faster than the previously available Montgomery multiplier [35]. The MSB-first multiplier requires $(2m - 1)$ AND gates, $(2m - 1)$ XOR gates, and $2m$ registers. The latency of this multiplier is m clock cycles and the critical path delay is given by $(T_A + T_X)$. The LSB-first multiplier requires $(2m-1)$ AND gates, $(2m-1)$ XOR gates, and $2m$ registers. The latency of this multiplier is also m clock cycles and the critical path delay is given by $(T_A + 2T_X)$.

In 2011, Morales et al. [32] presented an MSB-first Mongomery multiplier for two different Montgomery factors (x^m and x^{m-1}) using various classes of irreducible polynomials. This multiplier architecture is designed using a new approach where a linear feedback shift register (LFSR) is used as the main building block. Due to the low area complexity and high performance of the LFSR, this multiplier achieves low area and time complexities. This multiplier achieves reduction in area and time complexities when irreducible polynomials are selected as trinomials and all one polynomials, however, for general irreducible polynomials the complexities of this multiplier are matched to the avaialabe MSB-first multiplier [31]. For the case of general irreducible polynomials, this multiplier requires $(2m-1)$ AND gates, $(2m-1)$ XOR gates, and $2m$ registers. The latency of this multiplier is m clock cycles and the critical path delay is given by $(T_A + T_X)$.

In 2013, Huapeng Wu presented a linear feedback shift register (LFSR) based LSB-first bit-serial Montgomery multiplier [33]. The architecture of this multiplier is designed employing the LFSR approach, however, the area and time complexities of this multiplier are similar to the available LSB-first multiplier [31]. This multiplier requires $(2m-1)$ AND gates, $(2m-1)$ XOR gates, and $2m$ registers. The latency of this multiplier is m clock cycles and the critical path delay is given by $(T_A + 2T_X)$.

An efficient bit-serial Montgomery multiplier [34] was presented by Huapeng Wu in 2014 using weakly dual basis. The architecture of this multiplier is designed using a compact and highly regular Fibonacci type LFSR. This multiplier achieves more regularity and requires less area compared to previously available Montgomery multipliers [31–33], however, this multiplier may require basis conversion depending on the application. This multiplier requires m AND gates, $(2m-1)$ XOR gates, and $2m$ registers. The latency of this multiplier is m clock cycles and the critical path delay is given by $(T_A + \lceil \log_2 m \rceil T_X)$.

3.2 Review of Bit-Parallel Systolic Multipliers for Trinomials

Several bit-parallel systolic multipliers proposed in the literature for the finite field multiplication over GF(2^m) for irreducible trinomials are reviewed and the performance of these multipliers in terms of area and time complexities is presented in Table 3.2.

In 1984, Yeh et al. [47] presented a serial-in serial-out array multiplier with two

Table 3.2 Area and time complexities of the available bit-parallel systolic multipliers.

Design	**AND**	**XOR**	**Register**	**Latency**	**Critical Path**
[47]	$2m^2$	$2m^2$	$7m^2$	$3m$	$T_A + T_X$
[48]	$2m^2$	$2m^2$	$7m^2$	$3m$	$T_A + T_X$
[39](i)	$(m+1)^2$	$(m+1)^2$	$4(m+1)^2$	$m+1$	$T_A + T_X$
[39](ii)	$(m+1)^2$	$(m+1)(m+2)$	$5(m+1)^2$	$m+1$	T_X
[40]	m^2	m^2+m-1	$3m^2+2m-2$	$2m-1$	$T_A + T_X$
[41]	m^2	m^2+ml	$4m^2+2lm$	$m+l+1$	$T_A + T_X$
[42]	$(3m^2-m)/2$	m^2+m	$4m^2+m$	$m+1$	$T_A + T_X$
[43]	m^2	m^2+m	$3.5m^2+3m$	$m+2$	$T_A + T_X$
[27]	m^2	m^2-1	$2m(m-1)$	m	$T_A + T_X$
[44]	m^2	m^2+m	$2m^2$	m	$T_A + 2T_X$
[45]	m^2	m^2+m	$2m^2$	$m/2+2$	$2T_X$
[46]	m^2	$1.5m^2+0.5m$	$1.5m^2+2m-1$	$m+2$	$T_A + T_X$

$l = \lfloor (m-2)/(m-k) \rfloor + 1.$

control signals and a parallel-in parallel-out array multiplier with bidirectional data flow. Wang et al., in 1991, presented a two-dimensional parallel-in parallel-out and a one-dimensional serial-in serial-out systolic architectures [48]. These multipliers [47, 48] are based on the direct unrolling of iterative algorithms and not exploited the fully inherent parallelism. In 1998, various finite field arithmetic structures including multipliers [49] were explored to achieve reduction in latency using semi-systolic structures. For large values of m these semi-systolic structures are not suitable because of increased and longer broadcast signals. Further, all these multipliers [47–49] are defined over general irreducible polynomials and hence require large area and time complexities. Following this, many multipliers were proposed in the literature using various classes of irreducible polynomials, particularly trinomials, to achieve reduction in area and time complexities.

Two bit-parallel systolic multipliers [39] were presented by Lee et al. in 2001 over all one and equally spaced polynomials where one multiplier is area-efficient while the other is time optimal. Few properties of all one polynomials (AOP) are used in the formulation of the algorithms of these multipliers. These two systolic multipliers are also applicable to trinomials that come under equally spaced polynomials (ESP). The area-efficient multiplier architecture [39](i) requires $(m+1)^2$ AND gates, $(m+1)^2$ XOR gates, and $4(m+1)^2$ registers. The latency of this multiplier is $(m+1)$ clock cycles and the critical path delay is given by $(T_A + T_X)$. The time-efficient multiplier architecture [39](ii) requires $(m+1)^2$ AND gates, $(m+1)(m+2)$ XOR gates, and $5(m+1)^2$ registers. The latency of this multiplier is $(m+1)$ clock cycles and the critical path delay is given by T_X. The short critical path delay (T_X) of this time-efficient multiplier facilitates high throughput rates and high frequency of operation. Both of these multipliers achieve low hardware and low latency compared to the available systolic multipliers [47, 48], however, these multipliers are suitable for very limited applications due to the scarcity of AOPs.

A trinomial based low-complexity and low-latency bit-parallel systolic multiplier [40] was presented in 2003 by Chiou-Yng Lee. This multiplier requires low hardware compared to the previously available multipliers [39,47,48]. Also, this multiplier achieves low latency when compared to the multipliers [47, 48], however, it requires more latency compared to the multipliers [39]. This multiplier requires m^2 AND gates, $(m^2 + m - 1)$ XOR gates, and $(3m^2 + 2m - 2)$ registers. The latency of this multiplier is $(2m - 1)$ clock cycles and the critical path delay is given by $(T_A + T_X)$.

In 2003, Lee et al. presented a low-latency bit-parallel systolic multiplier [41] applicable for the class of trinomials $x^m + x^k + 1$ for which $\gcd(m, n) = 1$. This multiplier is based on a new algorithm that uses permutation polynomials. The algorithm is similar to conventional interleaved multiplication algorithm where reduction and multiplication are performed interleavingly. The latency of this multiplier is at least $(m + 2)$. For $2 \leq k \leq \lceil m/2 \rceil$, the latency of this multiplier is $m+3$, which is lower than the latency of the available multipliers [40, 47, 48]. This multiplier requires m^2 AND gates, $(m^2 + ml)$ XOR gates, and $(4m^2 + 2lm)$ registers. The latency of this multiplier is $(m+l+1)$ clock cycles and the critical path delay is given by $(T_A + T_X)$.

A low-complexity bit-parallel systolic Montgomery multiplier [42] using a transformation method is proposed by Lee et al. in 2005. The latency of this multiplier is lower compared to the multipliers [40, 41, 47, 48]. This multiplier requires $(3m^2 - m)/2$ AND gates, $(m^2 + m)$ XOR gates, and $(4m^2 + m)$ registers. The latency of this multiplier is $(m + 1)$ clock cycles and the critical path delay is given by $(T_A + T_X)$.

An area-efficient bit-parallel systolic Montgomery multiplier [43] for trinomials using

Toeplitz matrix-vector representation was presented in 2008 by Chiou-Yng Lee. This multiplier requires m^2 AND gates, $(m^2 + m)$ XOR gates, and $(3.5m^2 + 3m)$ registers. The critical path delay is given by the expression $(T_A + T_X)$ with a latency of $(m + 2)$ clock cycles. This multiplier achieves reduction in area complexity compared to the systolic multipliers [39, 41] while requiring high area complexity compared to the systolic multiplier [40]. Moreover, it achieves low latency compared to the systolic multipliers [40, 41] while requiring more latency compared to the multipliers [39].

A bit-level pipelined area-efficient systolic multiplier architecture [27] was presented in 2008 by Pramod Kumar Meher, where the architecture is derived using the signal flow graph approach [50] and applying appropriate cutset retiming techniques. This multiplier requires m^2 AND gates, $(m^2 - 1)$ XOR gates, and $(2m^2 - 2m)$ registers. The critical path delay is given by the expression $(T_A + T_X)$ with a latency of m clock cycles. This multiplier achieves low area complexity and low latency compared to the systolic multipliers [39–43, 47, 48].

A polynomial basis systolic multiplier [44] which can be extended to have concurrent error detection was presented by Bayat-Sarmadi et al. in 2009. The latency of this multiplier is similar to the latency of the multiplier [27] while area is marginally higher. The critical path delay of this multiplier is more than the previously available multipliers [27, 39–43]. This multiplier requires m^2 AND gates, $(m^2 + m)$ XOR gates, and $2m^2$ registers. The critical path delay is given by the expression $(T_A + 2T_X)$ with a latency of m clock cycles.

In 2012, Jia-feng et al. presented a low-latency systolic architecture [45], which is derived using the signal flow graph approach applying appropriate cut-set retiming techniques. The area required by this multiplier is similar to the multiplier [44] while critical path delay is lower. The latency of this multiplier is lower compared to the previously available multipliers [27, 39–44]. This multiplier requires m^2 AND gates, $(m^2 + m)$ XOR gates, and $2m^2$ registers. The critical path delay is given by the expression $2T_X$ with a latency of $(m/2 + 2)$ clock cycles.

Bayat-Sarmadi et al. [46] proposed a systolic multiplier for trinomials in 2015 based on the Montgomery multiplication algorithm. This multiplier achieves low area complexity compared to the available systolic multipliers [27, 39–45]. This multiplier requires m^2

AND gates, $(1.5m^2 + 0.5m)$ XOR gates, and $(1.5m^2 + 2m - 1)$ registers. The critical path delay is given by the expression $(T_A + T_X)$ with a latency of $(m + 2)$ clock cycles.

3.3 Review of Digit-Serial Sequential Multipliers for Trinomials

Several digit-serial sequential multipliers proposed in the literature for the finite field multiplication over GF(2^m) for irreducible trinomials are reviewed and the performance of these multipliers in terms of area and time complexities is presented in Table 3.3.

Table 3.3 Area and time complexities of the available digit-serial sequential multipliers.

Design	AND	XOR	Register	Latency (clock cycles)	Critical path delay
MSD-first [51]	wm	$wm + 3w$	$2m + w$	$n + 2$	$T_A + (\lceil log_2^{2w+1} \rceil)T_X$
LSD-first [51]	wm	$wm + 3w - 2$	$3m + w - 1$	$n + 2$	$T_A + (\lceil log_2^{w+1} \rceil)T_X$
[52]	wm	$wm + (w^2 + w)/2$	$2m + w$	$n - 1$	$T_A + (\lceil log_2^{w} \rceil + 2)T_X$
[53]	wm	$wm + (w^2 + w)/2$	$2m + w$	$n + 1$	$T_A + (\lceil log_2^{w} \rceil + 2)T_X$
[54]	$m^{log_4^6}$	$69/20m^{log_4^6} - 1/4m^{log_4^2} - 11/5$	$2m - 1$	$n + 1$	$T_A + (1 + 3log_4^m)T_X$
[55]	wm	$wm + w^2/2 + 3w/2 - 1$	$2m$	$n + 1$	$T_A + (\lceil log_2^{w} \rceil + 2)T_X$

In 1998, Song et al. [51] presented two digit-level LSD (Least significant digit)-first and MSD (most significant digit)-first serial-in parallel-out multipliers. The design of these multipliers is based on a new approach where an array-type algorithm is combined with a parallel algorithm to realize efficient architectures. The digit-level array-type algorithm reduces the latency of the multiplier while the parallel structure inside each digit-cell reduces the critical path delay as well as the switching activities. The MSD-first multiplier requires wm AND gates, $(wm + 3w)$ XOR gates, and $(2m + w)$ registers. The latency of this multiplier is $(n + 2)$ clock cycles and the critical path is given by $(T_A + (\lceil log_2^{2w+1} \rceil)T_X)$. The LSD-first multiplier requires wm AND gates, $(wm + 3w - 2)$ XOR gates, and $(3m + w - 1)$ registers. The latency of this multiplier is also $(n + 2)$ clock cycles and the critical path delay is given by $(T_A + (\lceil log_2^{w+1} \rceil)T_X)$. In these complexities expressions, m represents field order, n represents number of digits, and w represents digit size. These LSD-first and MSD-first multipliers achieve a substantial reduction in computational delay and energy consumption, when compared to the multipliers obtained by folding the available semi-systolic multipliers [49].

An MSD-first bit-parallel word-serial (BPWS) multiplier for GF(2^{233}) [52] is presented in 2005 by Tang et al., where, the field order 233 was selected as it provided enough security and also it is one of the field orders recommended by NIST. In this multiplier, an 8×233 parallel multiplier is employed to generate the finite field partial products. The proposed digit-serial multiplier requires wm AND gates, $(wm + (w^2 + w)/2)$ XOR gates, and $(2m + w)$ registers. The latency of this multiplier is $(n-1)$ clock cycles and the critical path delay is given by $(T_A + (\lceil log_2^w \rceil + 2)T_X)$. This multiplier achieves reduction in area and time complexities while consuming less power compared to the MSD-first multiplier [51].

In 2007, a high-throughput digit-serial multiplier for trinomials is presented in [53] by Meher. In this multiplier, T-flipflops are used for accumulation instead of D-flipflops to achieve a reduction in critical path delay as well as in hardware. This design saves m number of XOR gates and eliminates the feedback loops. Moreover, this multiplier is more regular and modular. This digit-serial multiplier requires wm AND gates, $(wm + (w^2 + w)/2)$ XOR gates, and $(2m + w)$ registers. The latency of this multiplier is $(n + 1)$ clock cycles and the critical path delay is given by $(T_A + (\lceil log_2^w \rceil + 2)T_X)$. This multiplier achieves lower area, critical path delay, and area-delay-product compared to the multipliers [51,52].

A digit-serial multiplier [54] based on (b, 2)-way Karatsuba algorithm decomposition is proposed in 2014 by Lee et al. for achieving sub-quadratic space complexity. This multiplier is based on Shifted Polynomial Basis (SPB), which is a variant of the polynomial basis. In this work, authors extended the Karatsuba decomposition to generalize (b, 2)-way KA decomposition, and based on this a low-complexity digit-serial multiplier is developed. This digit-serial multiplier requires $m^{log_4^6}$ AND gates, $(69/20m^{log_4^6} - 1/4m^{log_4^2} - 11/5)$ XOR gates, and $(2m - 1)$ registers. The latency of this multiplier is $(n + 1)$ clock cycles and the critical path delay is given by $(T_A + (\lceil log_2^w \rceil + 2)T_X)$. This multiplier is more regular, modular, and achieves significantly less area-delay-product compared to the available SPB multiplier [56].

In 2017, Namin et al. presented a low-power digit-serial multiplier [55], where, a new factoring technique is employed to minimize the switching power and logic gate substitution is used to reduce the internal power. Moreover, logic gate substitution also results in reduction in area complexity. This digit-serial multiplier requires wm AND gates, $(wm + w^2/2 + 3w/2 - 1)$ XOR gates, and $2m$ registers. The latency of this multiplier is $(n+1)$ clock cycles and the critical path delay is given by $(T_A + (\lceil log_2^w \rceil + 2)T_X)$. This multiplier achieves reduction in area, energy, and area-energy product compared to the multipliers [51–54, 57].

In 2017, Namin et al. presented a digit-serial multiplier using redundant basis representation. Redundant basis is also an appealing representation for field elements as its arithmetic does not require modulo reduction operation. This multiplier is a fully serial-in parallel-out multiplier where both the operands enter the architecture simultaneously. Thus, this multiplier does not require preloading of operands, thereby, it saves the delay required for preloading. This multiplier achives lower multiplication delay compared to the related previous multipliers [58, 59]. The fully serial-in paralle-out approach employed for this multiplier can be further explored on polynomial basis also.

3.4 Conclusions

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

Chapter 4

Area-Efficient Bit-Serial Sequential Multipliers for General Irreducible Polynomials

This chapter presents the design of bit-serial sequential multipliers for general irreducible polynomials targeting domestic IoT (Internet of Things) end devices. These multipliers include an MSB (most significant bit) first multiplier based on the proposed modified interleaved multiplication algorithm and an MSB as well as an LSB (least significant bit) first multipliers based on the proposed modified Montgomery algorithms. The modifications in the proposed algorithms involve employing more efficient logical relations in place of the existing relations. The analytical complexities in terms of area and delay are obtained and also computed for $m = 409$. Furthermore, the proposed multipliers are modeled using VHDL (Very High Speed Integrated Circuit Hardware Description Language) and implemented using Synopsys Design Compiler employing NanGate 45nm open cell library files. The obtained analytical and implementation results show that the proposed multipliers are area-efficient compared to the respective multipliers available in the literature.

4.1 Introduction

Internet of Things comprises billions of end devices attached to physical objects to sense and transmit data. These devices should be implemented with less hardware so as to reduce the cost, particularly, for domestic applications. Hence, implementation of security in these end devices also requires hardware efficient architectures. Elliptic curve cryptography (ECC) is an efficient public key cryptosystem used for security implementation in IoT and heavily uses GF(2^m) multiplications in its underlying operations. This multiplication is the performance-critical operation in ECC and its implementation targeting IoT end devices requires area-efficient architectures. Achieving area reduction for GF(2^m) multipliers is a continuous effort to meet the challenges raised from the various evolving applications such as IoT and WSNs (wireless sensor networks). These applications in domestic environments tend to focus more on cost reduction rather than targeting high performance, thereby requiring low hardware implementations. Though many architectural styles are available such as bit-parallel, digit-serial, bit-serial, systolic, sequential, and parallel for the implementation of GF(2^m) multiplication, it is observed that bit-serial sequential architectures require the lowest area compared to other architectural styles. Hence, the design of area-efficient bit-serial sequential GF(2^m) multipliers is desirable for the security implementation in the domestic IoT end devices. Furthermore, it is desirable for the domestic applications to have generic off-the-shelf components available in the market since the domestic applications may have a wide variety of requirements in terms of specifications. Hence, the design of GF(2^m) multipliers using general irreducible polynomials is also required.

Many bit-serial sequential architectures are proposed in the literature [14,16,25,28–34] for polynomial basis GF(2^m) multiplication using general irreducible polynomials to achieve reduction in area and time complexities. In this work, firstly, we have derived a modified interleaved multiplication algorithm based on the conventional interleaved multiplication algorithm [38] available in the literature. Subsequently, an efficient MSB-first sequential polynomial basis multiplier, which supports the multiplication of any two arbitrary finite field elements over GF(2^m) for generic irreducible polynomials, is designed based on the proposed algorithm. The area and delay complexities of this sequential multiplier are estimated and its performance is compared with the existing sequential multipliers [14,16,25,28–30]. Secondly, we have also derived modified Montgomery multiplication algorithms based on the Montgomery multiplication algorithms [31] available in the literature. Subsequently, efficient MSB-first and LSB-first sequential polynomial basis multipliers, which support the multiplication of any two arbitrary finite field elements over GF(2^m) for generic irreducible polynomials, are designed based on the proposed algorithms. The area and delay complexities of these sequential multipliers are estimated and the performance is compared with the existing Montgomery sequential multipliers [31–34]. It is observed that the proposed Montgomery sequential multipliers achieve reduction in area and area-delay-product (ADP) over the existing sequential multipliers when verified using a field of order $m = 409$. Further, all the proposed multipliers and some existing multipliers are implemented using ASIC (Application specific integrated circuit) technologies and the implementation results show that the proposed sequential multipliers achieve reduction in area and ADP over existing multipliers.

4.2 Area-Efficient Bit-Serial Sequential GF(2^m) Multiplier

In this section, the design and performance analysis of the proposed bit-serial sequential multiplier are presented. First, we present the mathematical formulations for the proposed modified interleaved GF(2^m) multiplication algorithm and its realization using the bit-serial architecture. Following this, analytical comparisons of area and time complexities and implementation results are presented.

4.2.1 Design

Mathematical Formulation

Let $A(x)$ and $B(x)$ be the field GF(2^m) elements and $T(x)$ be the field irreducible polynomial. Let $C(x)$ be the product of the elements $A(x)$ and $B(x)$ given by

$$C(x) = \big(A(x) \times B(x)\big) \bmod T(x) \tag{4.1}$$

Consider the expression for $C(x)$ presented in Eq. 4.1 for developing the proposed formulations, we have

$$C(x) = \big(A(x) \times B(x)\big) \bmod T(x)$$

$$= \big(A(x) \times (b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + ... + b_1x + b_0)\big) \bmod T(x)$$

$$= \big(A(x) \times ((......(((b_{m-1}x + b_{m-2})x + b_{m-3})x + b_{m-4})x$$
$$+ .... + b_1)x + b_0)\big) \bmod T(x)$$

$$= \big((.....(((b_{m-1}A(x)x + b_{m-2}A(x))x + b_{m-3}A(x))x + b_{m-4}A(x))x$$
$$+ ...... + b_1A(x))x + b_0A(x)\big) \bmod T(x) \quad (4.2)$$

Application of mod operation on the overall right side expression in Eq. 4.2 is replaced by interleaving it throughout the equation as

$$C(x) = \Big(......\Big(\Big(((b_{m-1}A(x))x \bmod T(x) + b_{m-2}A(x)\big)x \bmod T(x)$$
$$+ b_{m-3}A(x)\Big)x \bmod T(x) + b_{m-4}A(x)\Big)x \bmod T(x)$$
$$+ ...... + b_1A(x)\Big)x \bmod T(x) + b_0A(x) \quad (4.3)$$

The right side expression of Eq. 4.3 is evaluated in m iterations as follows. Beginning from the leftmost product term $b_{m-1}A$, modulo reduction is needed to be performed a total of $(m-1)$ times in the entire evaluation. In each iteration, the reduction is applied on the expression of the form $K^{(l-1)}(x) \times x$, where $K^{(l-1)}(x) = k^{(l-1)}_{m-1}x^{m-1} + k^{(l-1)}_{m-2}x^{m-2} + ... + k^{(l-1)}_1 x + k^{(l-1)}_0$ is considered to be a partial-product polynomial of degree $(m-1)$ which is obtained from the $(l-1)^{th}$ iteration. Furthermore, the partial-product $K^{(m)}(x)$ that is obtained from the m^{th} iteration is the final product required (i.e. $C(x) = K^{(m)}(x)$). During the first iteration, the partial-product to be used is $K^{(0)}(x) = 0$. This partial-product is accumulated with the leftmost product term $b_{m-1}A(x)$. Hence, the new partial-product term to be used for the second iteration is $K^{(1)}(x) = b_{m-1}A(x)$, and now $(K^{(1)}(x) \times x) \bmod T(x)$ is evaluated by first finding product expression $(K^{(1)}(x) \times x)$. Then, this is to be further modulo reduced using $T(x)$. This reduction using $T(x)$ requires consideration of the property of $T(x)$ that $T(x) = 0$, where x is a root of $T(x)$. i. e.

$$x^m = t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + ... + t_1x + 1 \quad (4.4)$$

We have, $K^{(1)}(x) \times x = k^{(1)}_{m-1}x^m + k^{(1)}_{m-2}x^{m-1} + ... + k^{(1)}_1 x^2 + k^{(1)}_0 x$. Modulo reduction of this m^{th} degree polynomial is performed using Eq. 4.4 as

$$(K^{(1)}(x) \times x) \bmod R(x) = k^{(1)}_{m-1}(t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + ...$$
$$... + t_1x + 1) + k^{(1)}_{m-2}x^{m-1} + ... + k^{(1)}_1 x^2 + k^{(1)}_0 x$$

$$= (k^{(1)}_{m-1}t_{m-1} + k^{(1)}_{m-2})x^{m-1} + (k^{(1)}_{m-1}t_{m-2}$$
$$+ k^{(1)}_{m-3})x^{m-2} + ... + (k^{(1)}_{m-1}t_1 + k^{(1)}_0)x + k^{(1)}_{m-1} \quad (4.5)$$

Thus, Eq. 4.5 denotes an $(m-1)^{th}$ degree polynomial obtained from the modulo reduction step performed in the second iteration. The right side expression in Eq. 4.5 is added to the corresponding product term $b_{m-2}A$. The result of this addition is represented by $K^{(2)}(x)$ to be used in next iteration. Hence, in general, for each of the subsequent $(m-2)$ iterations, the partial products can be obtained as

$$K^{(l)}(x) = K^{(l-1)}(x) + b_{m-l}A(x) \tag{4.6}$$

where $l \leq m$ is the iteration count. Applying the similar simplification steps as in Eqs. 4.4, 4.5, and 4.6 in the remaining $(m-2)$ iterations, the interleaved modular reduction multiplication suggested in Eq. 4.3 is evaluated as $C(x) = K^{(m)}$. The multiplication and addition operations between coefficients of the polynomials in Eqs. 4.6, 4.5 and 4.3 can be mapped to boolean algebra AND and boolean algebra XOR operations, respectively, and can be written as

$$K^{(l)}(x) = K^{(l-1)}(x) \oplus \big(b_{m-l}A(x)\big) \tag{4.7}$$

$$\begin{aligned}(K^{(1)}(x) \times x) \bmod T(x) = {} & \big((k^{(1)}_{m-1} \wedge t_{m-1}) \oplus k^{(1)}_{m-2}\big)x^{m-1} + \big((k^{(1)}_{m-1} \wedge t_{m-2}) \\ & \oplus k^{(1)}_{m-3}\big)x^{m-2} + ... + \big((k^{(1)}_{m-1} \wedge t_1) \oplus k^{(1)}_0\big)x + k^{(1)}_{m-1}\end{aligned} \tag{4.8}$$

$$\begin{aligned}C(x) = {} & \Big(....\Big(\Big(\big((b_{m-1}A(x))x \bmod T(x) \oplus b_{m-2}A(x)\big) \\ & x \bmod T(x) \oplus b_{m-3}A(x)\Big)x \bmod T(x) \oplus b_{m-4}A(x)\Big)x \bmod T(x) \oplus ..... \\ & ..... \oplus b_1A(x)\Big)x \bmod T(x) \oplus b_0A(x)\end{aligned} \tag{4.9}$$

where $\wedge$ and $\oplus$ represent logical AND and XOR operations, respectively, and $b_{m-l}A$ represents that each bit of operand A is AND operated with the bit b_{m-l}. This logical operations mapping is possible from the definitions of arithmetic operations of the base field GF(2). These logical operations mapped equations are further evaluated in view of performance gain in hardware implementation as explained below. Consider the boolean equation presented in Eq. 4.7, which can be rewritten as

$$\begin{aligned}K^{(l)}(x) &= K^{(l-1)}(x) \oplus \big(b_{m-l}A(x)\big) \\ &= \big(K^{(l-1)} \wedge (\overline{b_{m-l}A})\big) \ \vee \ \big(\overline{K^{(l-1)}} \wedge (b_{m-l}A)\big) \\ &= \big(K^{(l-1)} \wedge (\overline{b_{m-l}A})\big) \ \vee \ \big(\overline{K^{(l-1)}} \wedge \overline{\overline{(b_{m-l}A)}}\big) \\ &= K^{(l-1)} \ \odot \ (\overline{b_{m-l}A})\end{aligned} \tag{4.10}$$

where $\vee$ and $\odot$ represent logical OR and XNOR operations, respectively, and $\overline{b_{m-l}A}$ $\big($logical NOT of $(b_{m-l}A)\big)$ represents that each bit of operand A is NAND operated with the bit b_{m-l}. Similarly, Eqs. 4.8 and 4.9 can be equivalently rewritten using logical NAND and XNOR operators as

$$(K^{(1)}(x) \times x) \bmod T(x) = \big((k^{(1)}_{m-1} \overline{\wedge} t_{m-1}) \odot k^{(1)}_{m-2}\big)x^{m-1} + \big((k^{(1)}_{m-1} \overline{\wedge} t_{m-2}) \odot k^{(1)}_{m-3}\big)x^{m-2} + ... + \big((k^{(1)}_{m-1} \overline{\wedge} t_1) \odot k^{(1)}_0\big)x + k^{(1)}_{m-1} \quad (4.11)$$

$$C(x) = \Big(....\big(\big((\overline{b_{m-1}A}x \bmod T(x) \;\odot\; \overline{b_{m-2}A})x \bmod T(x) \;\odot\; \overline{b_{m-3}A}\big)x \bmod T(x) \;\odot\; \overline{b_{m-4}A}\big)x \bmod T(x) \odot .... \odot \overline{b_1 A}\Big) x \bmod T(x) \;\odot\; \overline{b_0 A} \quad (4.12)$$

where $\overline{\wedge}$ represents logical NAND operation.

Proposed Modified Interleaved Modular Reduction Multiplication Algorithm

The proposed algorithm to compute polynomial basis multiplication is presented in algorithm 4.1. This algorithm is developed to reduce the hardware required for the implementation of polynomial basis multiplication. In the algorithm formulation, Eq. 4.10 is used along with Eqs. 4.11 and 4.12 to compute the desired final product in m iterations, where m is the order of field GF(2^m) in which multiplication is being computed. The three Eqs. 4.10, 4.11, and 4.12 involve logical NAND and logical XNOR operations which prompts efficient hardware realization than that of conventional equations based on logical AND and logical XOR operations. Furthermore, the algorithm is formulated such that step 2 and step 3 involve a similar type of computations thereby their implementation needs similar hardware. The computations in the algorithm can be described as follows. Let A and B be two arbitrary field GF(2^m) elements while T be an arbitrary field irreducible polynomial all represented as m-bit vectors. Assume C is an m-bit variable that holds the multiplication result at the end of the m^{th} loop iteration. Step 1 of the algorithm indicates a total of m iterations in the computation of $AB \bmod T$. The operations to be performed in each iteration are modulo reduction followed by accumulation.

In l^{th} iteration, modulo reduction is applied to the $K^{(l-1)}$ resulted from $(l-1)^{th}$ iteration. However, for the first iteration $K^{(0)} = 0$ is considered which is suggested by the initialization step in the algorithm. The modular reduction is interleaved throughout the computation as in step 2 of the algorithm. The accumulation that adds all the partial-products, from each iteration, to finally give the desired multiplication result is described by step 3. In each iteration, one bit of B is considered serially starting from most-significant-bit.

Algorithm 4.1: Modified interleaved modular reduction multiplication algorithm

```
Input: A=$(a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$, B=$(b_{m-1}, b_{m-2}, \ldots, b_1, b_0)$ both w.r.t. polynomial basis, and T =$(t_{m-1}, t_{m-2}, \ldots, t_1, 1)$
Output: C = (A × B) mod T = $(c_{m-1}, c_{m-2}, \ldots, c_1, c_0)$ also w.r.t. polynomial basis
    Initialization : $K^{(0)} \leftarrow 0$;
1: for $l = 1$ to $m$ do
2:     $K^{(l-1)} \leftarrow (\overline{k_{m-1}^{(l-1)}T}\ ) \odot xK^{(l-1)}$;        ▷ $xK^{(l-1)}$: Left shifting $K^{(l-1)}$
3:     $K^{(l)} \leftarrow K^{(l-1)} \odot \overline{(b_{m-l}A)}$;        ▷ $K^{(l)} = (k_{m-1}^{(l)}, k_{m-2}^{(l)}, \ldots\ldots, k_1^{(l)}, k_0^{(l)})$
4: end for
5: return $K^{(m)}$
```

The term $(\overline{k_{m-1}^{(l-1)}T}\ )$ in step 2 represents the NAND operation of the bit $k_{m-1}^{(l-1)}$ with each bit of T. Similarly, step 3 involves the NAND operation of the bit b_{m-l} of B with each bit of A. The term $xK^{(l-1)}$ indicates the usual multiplication of $K^{(l-1)}(x)$ with x. Since $K^{(l-1)}$ is an m-bit variable, $xK^{(l-1)}$ is realized by performing a left shift operation on $K^{(l-1)}$ by one bit. Thus, the proposed algorithm performs the modulo reduction step and accumulation step in each iteration and finally computes the desired multiplication result at the end of the m^{th} iteration.

Proposed Multiplier Architecture

In this section, the hardware realization of the proposed modified interleaved modular reduction multiplication algorithm is presented. The proposed serial-in parallel-out sequential multiplier architecture is shown in Fig. 4.1. The architecture comprises two

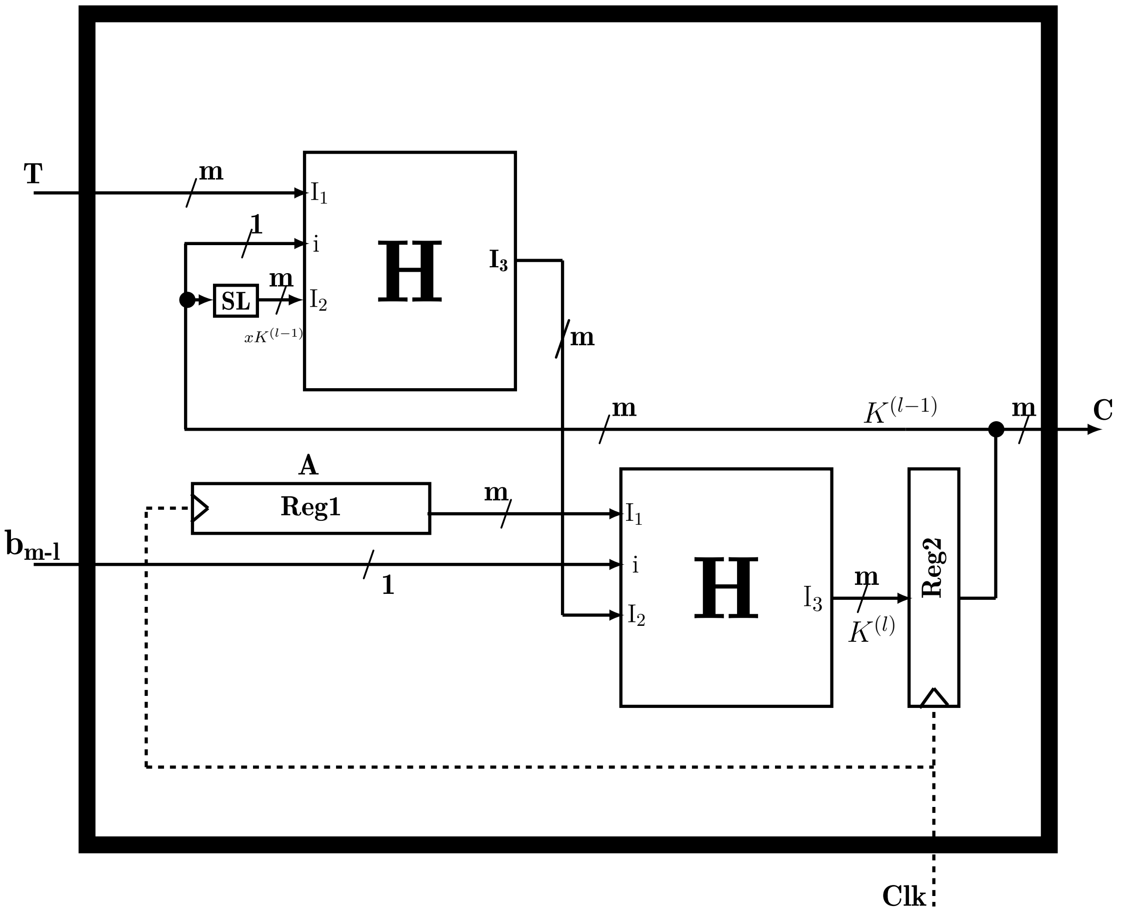

Figure 4.1 Top-level block diagram of the proposed bit-serial sequential multiplier.

H blocks, two registers (Reg1, Reg2), and a shift left (SL) block. The architecture takes a single-bit input (b_{m-l}) for every clock cycle and generates an m-bit product (C) in parallel after m clock cycles. In addition, the architecture also takes an m-bit input (T) to accommodate selected irreducible polynomial. Assume that the field polynomials to be multiplied are A and B, and field irreducible polynomial is T. All these three polynomials A, B, and T, using their vector representations, are treated as m-bit operands for the multiplier architecture. Operand A is preloaded into the m-bit register, Reg1, and it is kept constant throughout the multiplication operation. The operand B enters the architecture serially (b_{m-l}) with one bit per cycle starting from the most significant bit. The m-bit operand T should be available throughout the multiplication operation. Since the field multiplication involves modulo reduction and accumulation steps, these operations are realized using H blocks in the proposed architecture. Step 2 of the algorithm is realized in hardware using upper H block which performs the modulo reduction of $(K^{(l-1)}(x) \times x)$ with the irreducible polynomial $T(x)$. Step 3 of the algorithm refers

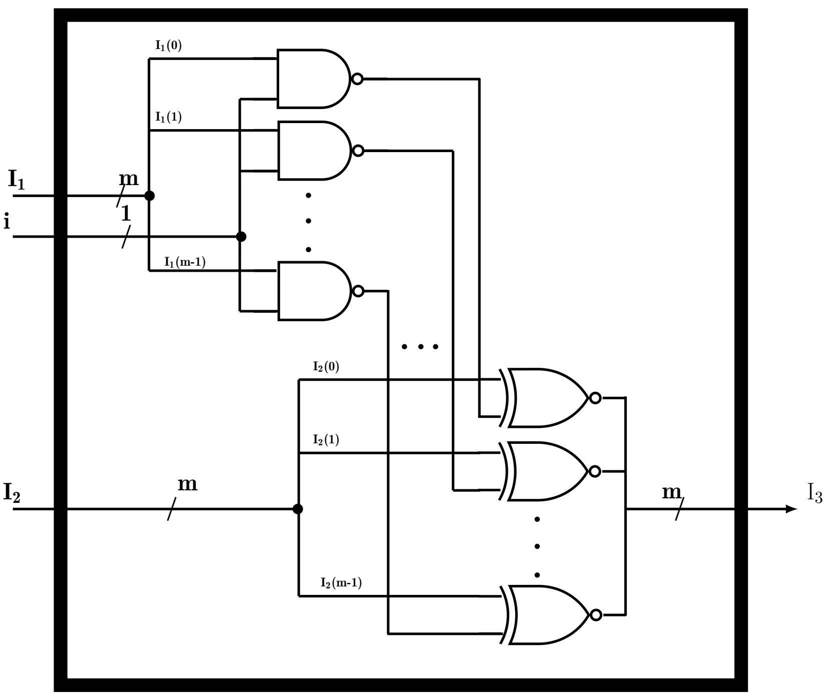

Figure 4.2 Gate level schematic of H block.

accumulation of partial products and it is realized by lower H block.

Figure 4.2 shows the gate level schematic of the H block. The H block contains two levels of logic gates. The first level of logic gates is constructed with an array of m NAND gates to implement the NAND operations suggested in step 2 and step 3 of the algorithm. The second level of logic gates is constructed with an array of m XNOR gates to implement the XNOR operations suggested in step 2 and step 3 of the algorithm. The m-bit partial product, $K^{(l)}$, generated during l^{th} clock cycle is given as input for the m-bit register Reg2 which is initialized to zero. The term $xK^{(l-1)}$ in Step 2 of the algorithm, which is a simple left shift of $K^{(l-1)}$, is realized in SL block using hardware-free routing. The proposed multiplier takes m clock cycles to complete one multiplication operation. In each clock cycle, this multiplier performs modulo reduction of the previous partial-product followed by accumulation. The partial product generated during the m^{th} clock cycle is the desired result of the field multiplication and is available at Reg2 after the m^{th} clock cycle. In addition, it is also noted that the two H blocks, being identical, improve the regularity of the multiplier architecture.

4.2.2 Analytical Results

In this section, the area and time complexities of the proposed sequential multiplier are compared with that of similar multipliers available in the literature. In Table 4.1, the analytical expressions for area and time complexities of the proposed multiplier together with that of available multipliers [14, 16, 25, 28–30] are presented. The analytical expressions presented in Table. 4.1 are evaluated for $m = 409$ using the area and time complexity estimations of logic gates from NanGate 45nm technology library files and presented in Table 4.2.

Table 4.1 presents the analytical comparison of area complexity, latency, and critical path delay of the proposed multiplier with the multipliers considered for comparison. The area required for the proposed multiplier is computed in terms of the number of AND gates, NAND gates, XOR/XNOR gates, multiplexers, and registers. We also present the area complexity for the other multipliers in a similar way to compare with that of the proposed multiplier. The expressions for area complexities of the proposed multiplier are derived using Figs. 4.1 and 4.2. Figure 4.1 contains two identical H blocks along with two m-bit registers. Each H block (See Fig. 4.2) contains an array of m-NAND gates and an array of m-XNOR gates. Hence, the proposed multiplier architecture requires $2m$ NAND gates, $2m$ XNOR gates, and $2m$ registers. Note that m represents the order of the GF(2^m). The time complexities of the proposed architecture and other multipliers considered for comparison are computed by assuming T_A, T_X, T_{NA}, T_{XN}, T_{FF}, T_M and T_{tsb} denotes the delays of 2-input AND gate, 2-input XOR gate, 2-input NAND gate, 2-input XNOR gate, D flip-flop, 2:1 1-bit multiplexer, and tristate buffer, respectively.

Table 4.1 Area and time complexities comparison for GF(2^m).

Design	AND	NAND	XOR/XNOR*	MUX	Register	Latency	Critical path
[28]	$2m$	0	$2m-1$	0	$4m+2$	$m+1$	$T_A + T_X$
[14]	$2m$	0	$2m-1$	m	$3m$	m	$T_A + T_X$
[25]	$(m^2+m)/2$	0	$(m^2+m)/2$	$4m$	$5m-1$	$2k_t^a+1$	$T_X\lceil log_2 m\rceil + 2T_M + T_A$
[29]	$4m$	0	$2m$	$m^b+m^c+m^d$	$3m$	m	$T_A + T_X + T_{tsb}$
[30]	0	0	$6m+18$	$14m+26$	$6m+7$	$m/4$	$2T_M + 4T_X$
[16]	$2m$	0	$2m$	$2m$	$3m$	m	$T_A + T_X$
Proposed	0	$2m$	$2m$	0	$2m$	m	$T_{NA} + 2T_{XN}$

*XOR and XNOR have the same area and time complexities, [a]the second highest degree of the irreducible polynomial, [b] OR gates, [c]inverters, [d]tristate buffers

The critical path and latency of the proposed multiplier computed from the proposed architecture are $T_{NA}+2T_{XN}$ and m clock cycles, respectively. It is observed from Table. 4.1 that the multipliers [14, 16, 28, 29] require less critical path delay at the expense of more number of registers compared to the proposed multiplier. It may also be observed that the multipliers [25, 30] require less latency compared to the proposed multiplier, however, they require more hardware and critical path delay. In addition, it is also noted that the proposed multiplier requires the lowest number of registers, which also results in less area requirement.

The analytical comparisons presented in Table 4.1 can be better understood by evaluating them for a specific value of field order along with a specific technology-based area and time complexity estimations of gates. The field order, m, can be selected as 409 which is one of the field sizes recommended by National Institute of Standards and Technology (NIST) for Elliptic curve cryptographic applications. For the estimation of area and time complexities of the gates, NanGate 45nm technology-based open cell library statistics [46, 60] is adopted as follows: The area complexities in terms of the NAND gate equivalents (GE) for a NOT gate, a 2-input AND gate, a 2-input XOR gate, a 2-input XNOR gate, a 2-1 MUX (Multiplexer), and a D flip-flop with set/reset capabilities are taken as 0.5, 1.4, 2, 2, 1.4, and 5.7, respectively. The delays of a 2-input NAND gate, a 2-input AND gate, and a 2-input XOR gate, a 2-input XNOR gate, a 2-1 MUX, a tristate buffer, and a D flip-flop with set/reset are 0.015, 0.025, 0.035, 0.035, 0.025, 0.020and 0.060 nanoseconds, respectively. It is observed that the area required for a 2-input NAND gate based on Synopsys design compiler synthesis using 45nm NanGate open cell libraries is $0.8\mu m^2$.

Table 4.2 presents the comparison of estimated area (μm^2), delay, and area-delay-product (ADP) of the proposed multiplier with that of the same multipliers considered for comparison in Table 4.1. It is observed that the proposed multiplier requires the lowest area. It is clear from Table. 4.2 (% reduction in area column) that the proposed multiplier achieves area efficiency of 41%, 31%, 97%, 39%, 73%, and 34% when compared with multipliers [28], [14], [25], [29], [30], [16], respectively. It may be noted that multipliers [28], [14], [29], [16] have lower critical path, and multipliers [28], [14], [29], [30], [16] take less delay compared to the proposed multiplier. However, area-delay-product (ADP)

Table 4.2 Comparison of area, delay, and area-delay-product estimations for GF(2^{409}).

Design	Critical path (ns)	Latency (clock cycles)	Delay (ns)	Area (μm^2)	ADP ($\times 10^6$)	% reduction in Area	% reduction in ADP
[28]	0.060	410	24.60	9692	0.23	41	17
[14]	0.060	409	24.54	8276	0.20	31	5
[25]	0.390	175	68.25	239210	16.3	97	99
[29]	0.080	409	32.7	9390	0.30	39	36
[30]	0.190	103	19.57	21619	0.42	73	54
[16]	0.060	409	24.54	8736	0.21	34	9
Proposed	0.085	409	34.8	5693	0.19	–	–

parameter is the balanced parameter for overall performance comparison of various architectures rather than area and delay individually. The proposed multiplier achieves the best area-delay-product compared to the multipliers considered for comparison. It is evident from Table 4.2 (% reduction in ADP column) that the proposed multiplier achieves area-delay efficiency of 17%, 5%, 99%, 36%, 54%, and 9% when compared with multipliers [28], [14], [25], [29], [30], [16], respectively. Hence, it is clear from the estimation values presented in Table. 4.2 that the proposed multiplier is area and area-delay-product efficient.

4.2.3 Implementation Results

It is observed from Table. 4.2 that the multipliers [14,16] require less area compared to the other multipliers considered for comparison. Hence, the proposed multiplier and the two multipliers [14,16] are modeled using VHDL for GF(2^{409}). The RTL (Register Transfer Level) designs are simulated using Vivado Simulator to verify the functionality. The netlists of these models are synthesized using Synopsys Design Compiler tool employing NanGate 45nm open cell libraries [60] to obtain the area and time complexities. The area

Table 4.3 Comparison of ASIC implementation results for GF(2^{409}).

Design	Multiplier area (μm^2)	Critical path delay (ns)	Multiplication delay (ns)	Area $\times$ Delay ($\mu m^2 \times ns$)	% reduction in Area	% reduction in ADP
[14]	9687	0.20	82	794334	28	3
[16]	10213	0.20	82	837466	31	8
Proposed	6971	0.27	110	766810	–	–

and time complexities obtained for all the three multipliers are tabulated in Table 4.3.

It is observed from the ASIC implementation results that the proposed multiplier achieves 28% hardware efficiency and 3% area-delay-product improvement compared to the best available multiplier [14]. It may also be observed from implementation results that the proposed multiplier implementation achieves 31% reduction in area complexity and 8% reduction in area-delay-product compared to the sequential multiplier [16]. Hence, the ASIC implementation results confirm that the proposed multiplier achieves better area as well as area-delay-product complexities.

4.3 Low-Complexity Bit-Serial Sequential Montgomery GF(2^m) Multipliers

In this section, the design and performance analysis of the proposed Montgomery bit-serial sequential multipliers are presented. First, we present the mathematical formulations for the proposed MSB-first and LSB-first modified GF(2^m) Montgomery multiplication algorithms and the realization of these algorithms using the bit-serial architectures. Following this, analytical comparisons of area and time complexities and implementation results are also presented.

4.3.1 Design

Let $\beta(x)$ and $\gamma(x)$ are two GF(2^m) elements to be multiplied, and let $\phi(x) = \beta(x)\gamma(x) \bmod T(x)$ be the product. However, multiplication using Montgomery technique requires the elements converted to be Montgomery residues. Let $A(x)$ and $B(x)$ be Montgomery residues of the elements $\beta(x)$ and $\gamma(x)$, respectively, given by

$$A(x) = \beta(x)r(x) \bmod T(x) = \sum_{i=0}^{m-1} a_i x^i \tag{4.13}$$

and

$$B(x) = \gamma(x)r(x) \bmod T(x) = \sum_{i=0}^{m-1} b_i x^i \tag{4.14}$$

where, $r(x)$ is a fixed field element called Montgomery factor that satisfies the relation $\gcd(T(x), r(x)) = 1$. The Montgomery multiplication can be defined as

$$C(x) = A(x)B(x)r^{-1}(x) \bmod T(x) \tag{4.15}$$

Algorithm 4.2: Montgomery multiplication in GF(2^m)

```
Input: $A, B, r, T(x), T'(x)$
Output: $C = ABr^{-1} \bmod T(x)$
1: $q := AB$
2: $u := qT'(x) \bmod r$
3: $C := (q + uT(x))/r$
```

where, $r^{-1}(x)$ is the inverse of $r(x)$ in GF(2^m) and $C(x)$ is the Montgomery residue of $\phi(x)$. The conventional Montgomery algorithm that computes the Montgomery product C given in Eq. 4.15 is presented in algorithm 4.2 [35]. In this algorithm, $r^{-1}(x)$ and $T'(x)$ are two polynomials such that $r(x)r^{-1}(x) + T(x)T'(x) = 1$. This work [35] employs x^m as the Montgomery factor and also presented an LSB-first bit-serial multiplication algorithm. However, the work [31] analyzed the role of Montgomery factor and found out that x^{m-1} gives time-efficient bit-serial architectures for GF(2^m) over general irreducible polynomials. Hence, we consider x^{m-1} as the selected Montgomery factor as it results in more efficient architectures than other factors.

Consider the relation $T(x) = 0$, where x is the root of irreducible polynomial $T(x)$, which gives that

$$T(x) = x^m + t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + ... + t_1x + 1 = 0$$

It can be rewritten as

$$x^m + t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + ... + t_1x + 1 = 0 \tag{4.16}$$

Equation 4.16 can be equivalently rewritten as

$$x^m + t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + ... + t_1x = 1 \tag{4.17}$$

Multipliying both sides of Eq. 4.17 with α^{-1} gives the relation,

$$x^{-1} \bmod T(x) = x^{m-1} + t_{m-1}x^{m-2} + t_{m-2}x^{m-3} + ... + t_1 \tag{4.18}$$

MSB-first Montgomery Multiplication

Formulations

We have the Montgomery multiplication over GF(2^m) given in Eq. 4.15 as

$$C = ABr^{-1} \bmod T(x)$$

Using Eq. 4.14 and the selected Montgomery factor, $r = x^{m-1}$, it can be written as,

$$\begin{aligned}
C &= A\sum_{i=0}^{m-1} b_i x^i x^{-(m-1)} \bmod T(x)\\
&= A\sum_{i=0}^{m-1} b_i x^{(i-(m-1))} \bmod T(x)\\
&= \big(Ab_{m-1} + Ab_{m-2}x^{-1} + Ab_{m-3}x^{-2} + ... + Ab_1x^{-(m-2)} + Ab_0x^{-(m-1)}\big) \bmod T(x)\\
&= \big(b_{m-1}A + b_{m-2}Ax^{-1} + b_{m-3}Ax^{-2} + ... + b_1Ax^{-(m-2)} + b_0Ax^{-(m-1)}\big) \bmod T(x)
\end{aligned} \tag{4.19}$$

The computation of Eq. 4.19 can be performed by defining a recursive relation, $A^{(i)} = A^{(i-1)}x^{-1} \bmod T(x)$, where $A^{(i)}$ is a reduced form of Ax^{-i}, and also $A^{(0)} = A$. The expression for $A^{(i-1)}$ can be written as

$$A^{(i-1)}(x) = a_{m-1}^{(i-1)}x^{m-1} + a_{m-2}^{(i-1)}x^{m-2} + ... + a_1^{(i-1)}x + a_0^{(i-1)} \tag{4.20}$$

Now, the next recursive form $A^{(i)}$ can be obtained using $A^{(i-1)}$ as follows,

$$\begin{aligned}
A^{(i)} &= \big(a_{m-1}^{(i-1)}x^{m-1} + a_{m-2}^{(i-1)}x^{m-2} + ... + a_1^{(i-1)}x + a_0^{(i-1)}\big)x^{-1} \bmod T(x)\\
&= \big(a_{m-1}^{(i-1)}x^{m-2} + a_{m-2}^{(i-1)}x^{m-3} + ... + a_1^{(i-1)} + a_0^{(i-1)}x^{-1}\big) \bmod T(x)
\end{aligned} \tag{4.21}$$

Using Eq. 4.18, Eq. 4.21 can be rewritten as

$$\begin{aligned}
A^{(i)} = \Bigg(& a_{m-1}^{(i-1)}x^{m-2} + a_{m-2}^{(i-1)}x^{m-3} + ... + a_1^{(i-1)}\\
&+ a_0^{(i-1)}\big(x^{m-1} + t_{m-1}x^{m-2} + t_{m-2}x^{m-3} + ... + t_1\big)\Bigg)
\end{aligned} \tag{4.22}$$

Following Eq. 4.22, the computation of $A^{(i)}$ can be performed using the relations as follows,

$$\begin{aligned}
a_{m-1}^{(i)} &= a_0^{(i-1)}\\
a_k^{(i)} &= a_{k+1}^{(i-1)} + a_0^{(i-1)}t_{k+1}, \text{ where, } 0 \le k \le m-2
\end{aligned} \tag{4.23}$$

Now, the computation of Montgomery product, C, can be performed using Eq. 4.19 and Eq. 4.23, and using the following recursive relation as

$$C^{(i)} = C^{(i-1)} + b_{m-i}A^{(i-1)}, \text{ where, } 1 \leq i \leq m, \text{ and } C^{(0)} = 0 \tag{4.24}$$

The computation of C using the recursive relation in Eq. 4.24 can be performed in m iterations as follows. During the i^{th} iteration, the computation of the partial product, $C^{(i)}$, is performed by accumulating the previous partial product, $C^{(i-1)}$, which is generated during the $(i-1)^{th}$ iteration, to the present product term $b_{m-i}A^{(i-1)}$. The product term $b_{m-i}A^{(i-1)}$ denotes that each bit of $A^{(i-1)}$ is multiplied with the bit b_{m-i} i.e., $b_{m-i}A^{(i-1)} = (b_{m-i}a_{m-1}^{(i-1)}, b_{m-i}a_{m-2}^{(i-1)}, ..., b_{m-i}a_1^{(i-1)}, b_{m-i}a_0^{(i-1)})$. The partial product $C^{(m)}$ generated during the m^{th} iteration is the required Montgomery product, C. The hardware realization of computation in Eq. 4.24 requires transformation of usual arithmetic operations into logical operations using the definitions of finite field GF(2^m) arithmetic. In the base field GF(2) of GF(2^m), addition is realized using the XOR operation and multiplication is realized using the AND operation. Based on this fact, the computation in Eq. 4.24 can be rewritten as,

$$C^{(i)} = C^{(i-1)} \oplus (b_{m-i} \wedge A^{(i-1)}) \tag{4.25}$$

where, $\oplus$ represents the logical XOR operator, and $b_{m-i} \wedge A^{(i-1)}$ denotes that each bit of $A^{(i-1)}$ is performed logical AND operation with the bit b_{m-i} i.e., $b_{m-i} \wedge A^{(i-1)} = (b_{m-i} \wedge a_{m-1}^{(i-1)}, b_{m-i} \wedge a_{m-2}^{(i-1)}, ..., b_{m-i} \wedge a_1^{(i-1)}, b_{m-i} \wedge a_0^{(i-1)})$, where, $\wedge$ represents logical AND operator. Further modification of Eq. 4.25, with the view that it results in more efficient hardware realization, can be performed as follows.

$$\begin{aligned}
C^{(i)} &= C^{(i-1)} \oplus (b_{m-i} \wedge A^{(i-1)}) \\
&= \left(C^{(i-1)} \wedge (\overline{b_{m-i} \wedge A^{(i-1)}})\right) \vee \left(\overline{C^{(i-1)}} \wedge (b_{m-i} \wedge A^{(i-1)})\right) \\
&= \left(C^{(i-1)} \wedge (\overline{b_{m-i} \wedge A^{(i-1)}})\right) \vee \left(\overline{C^{(i-1)}} \wedge (\overline{\overline{b_{m-i} \wedge A^{(i-1)}}})\right) \\
&= C^{(i-1)} \odot (\overline{b_{m-i} \wedge A^{(i-1)}})
\end{aligned} \tag{4.26}$$

where, $\odot$ and $\vee$ represent the logical XNOR and logical OR operators, respectively. The term $\overline{b_{m-i} \wedge A^{(i-1)}}$ denotes that each bit of $A^{(i-1)}$ is performed logical NAND operation with the bit b_{m-i} i.e., $\overline{b_{m-i} \wedge A^{(i-1)}} = (b_{m-i}\overline{\wedge}a_{m-1}^{(i-1)}, b_{m-i}\overline{\wedge}a_{m-2}^{(i-1)}, ..., b_{m-i}\overline{\wedge}a_1^{(i-1)}, b_{m-i}\overline{\wedge}a_0^{(i-1)})$, where, $\overline{\wedge}$ represents logical NAND operator. Similarly, Eq. 4.23 can be rewritten employing these efficient logical relations as,

$$\begin{aligned} a_{m-1}^{(i)} &= a_0^{(i-1)} \\ a_k^{(i)} &= a_{k+1}^{(i-1)} \odot (a_0^{(i-1)} \overline{\wedge} t_{k+1}), \text{ where, } 0 \le k \le m-2 \end{aligned} \tag{4.27}$$

Algorithm

The computation of the Montgomery product C using the Eq. 4.26 and Eq. 4.27 is described in algorithm 4.3.

Algorithm 4.3: Modified MSB-first Montgomery multiplication in GF(2^m)

Input: A=$(a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$, B=$(b_{m-1}, b_{m-2}, \ldots, b_1, b_0)$ both w.r.t. polynomial basis, and T' =$(1, t_{m-1}, t_{m-2}, \ldots, t_1)$.

Output: $C = ABx^{-(m-1)} \bmod T(x) = (c_{m-1}, c_{m-2}, \ldots, c_1, c_0)$ also w.r.t. polynomial basis

```
    Initialization : $A^{(0)} \leftarrow$ A, $C^{(0)} \leftarrow 0$;
1: for $i = 1$ to $m$ do
2:     $C^{(i)} \leftarrow C^{(i-1)} \odot (\overline{b_{m-i} \wedge A^{(i-1)}})$;
3:     $A^{(i)} \leftarrow A^{(i-1)}x^{-1} \odot (\overline{a_0^{(i-1)} \wedge T'})$;    ▷ $A^{(i-1)}x^{-1}$: Right shifting $A^{(i-1)}$ by 1-bit
4: end for
5: return $C^{(m)}$
```

Let A and B be the two Montgomery residues to be multiplied, and T' be an m-bit element defined to be $T' = (1, t_{m-1}, t_{m-2}, \ldots, t_1)$ using the irreducible polynomial T. Let the partial product $C^{(i)}$ and the reduced term $A^{(i)}$ be initialized for $i = 0$ as $C^{(0)} = 0$ and $A^{(0)} = A$, respectively. The computation of the Montgomery product C is performed in m iterations, which is described in Step 1. In each of the i^{th} iteration, one bit of the operand B, b_{m-i}, is involved in computation. In Step 2, the partial product, $C^{(i)}$, is computed during the i^{th} iteration using the previous partial product, $C^{(i-1)}$, and reduced term $A^{(i-1)}$ generated during the $(i-1)^{th}$ iteration. This step of the algorithm is based

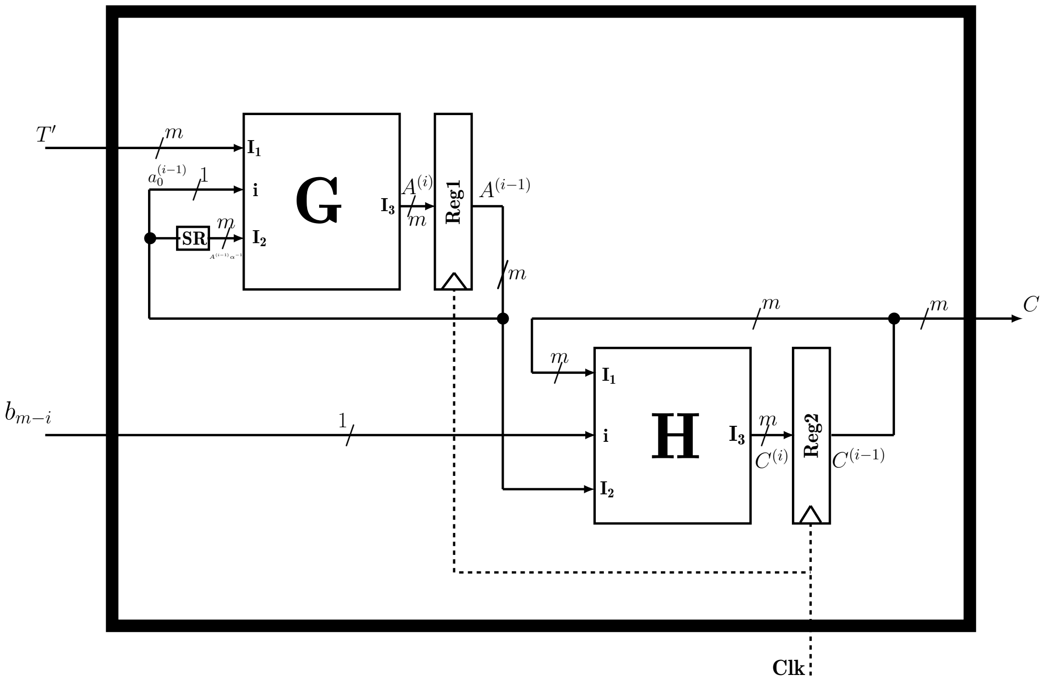

Figure 4.3 Top-level block diagram of the proposed MSB-first bit-serial sequential multiplier.

on Eq. 4.26. In Step 3 of the algorithm, the reduced term $A^{(i)}$ is computed during the i^{th} iteration using the previous reduced term $A^{(i-1)}$ generated during the $(i-1)^{th}$ iteration. This step of the algorithm is based on Eq. 4.27. During the m^{th} iteration, the partial product $C^{(m)}$ is computed which is the required Montgomery product, C.

Architecture

The hardware realization of the proposed MSB-first algorithm (algorithm 4.3) is presented in this section. The proposed architecture, shown in Fig. 4.3, comprises of two lower blocks G and H whose detailed architectures are presented in Fig. 4.4 and Fig. 4.5, respectively. In addition to these two lower blocks, the architecture also includes two m-bit registers and an SR block. The architecture requires an m-bit input, T', a single bit input, b_{m-i}, one of the operands, A, to be preloaded into Reg1, and then m clock cycles in order to generate the required m-bit multiplication product, C. As Step 1 of the algorithm indicates that it requires m iterations to compute the product C, the proposed bit-serial architecture realizes this step requiring m clock cycles where in each clock cycle the iterative computations presented in Step 2 and Step 3 are processed. The computation suggested by Step 3 of the algorithm is realized using the G block, SR block,

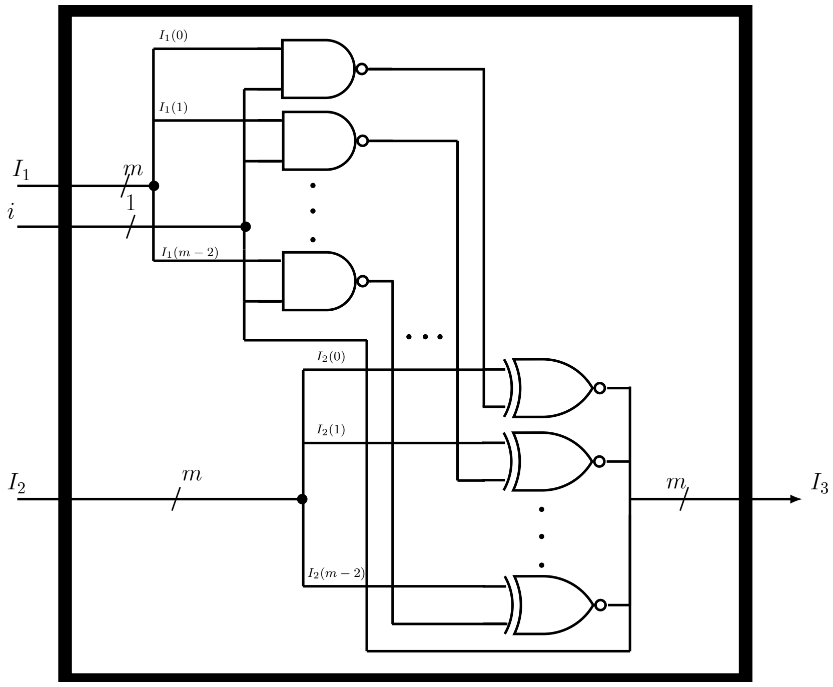

Figure 4.4 Gate level architecture of G block.

and the register Reg1. The G block has a feedback path through the register Reg1. During the i^{th} clock cycle, the G block generates the reduced term, $A^{(i)}$, using its inputs $A^{(i-1)}x^{-1}$, $a_0^{(i-1)}$, and T'. The term $A^{(i-1)}x^{-1}$ is obtained from $A^{(i-1)}$ using the SR block. The SR block, which shifts the input right by a 1-bit position, simply routes the input without requiring any hardware. The computations suggested by Step 2 of the algorithm is realized using the H block and the register Reg2. The H block has a feedback path through the register Reg2. During the i^{th} clock cycle, the H block generates the partial product, $C^{(i)}$, using its inputs $C^{(i-1)}$, b_{m-i}, and $A^{(i-1)}$. Furthermore, during the m^{th} clock cycle the H block generates the partial product $C^{(m)}$ which is available at the output after this clock cycle.

The detailed architectures of the two lower blocks G and H are presented in Fig. 4.4 and Fig. 4.5, respectively. The G block realizes the computation of the expression $A^{(i-1)}x^{-1} \odot (\overline{a_0^{(i-1)} \wedge T'})$ using two levels of logic gates ($A^{(i-1)}x^{-1}$ is realized using SR block). The first level is realized using an array of $(m-1)$ NAND gates while the second level requires an array of $(m-1)$ XNOR gates. The input i is applied to all NAND gates, and also it is routed directly to the output, I_3. The H block realizes the computation of the expression $C^{(i-1)} \odot (\overline{b_{m-i} \wedge A^{(i-1)}})$ using two levels of logic gates. The first level

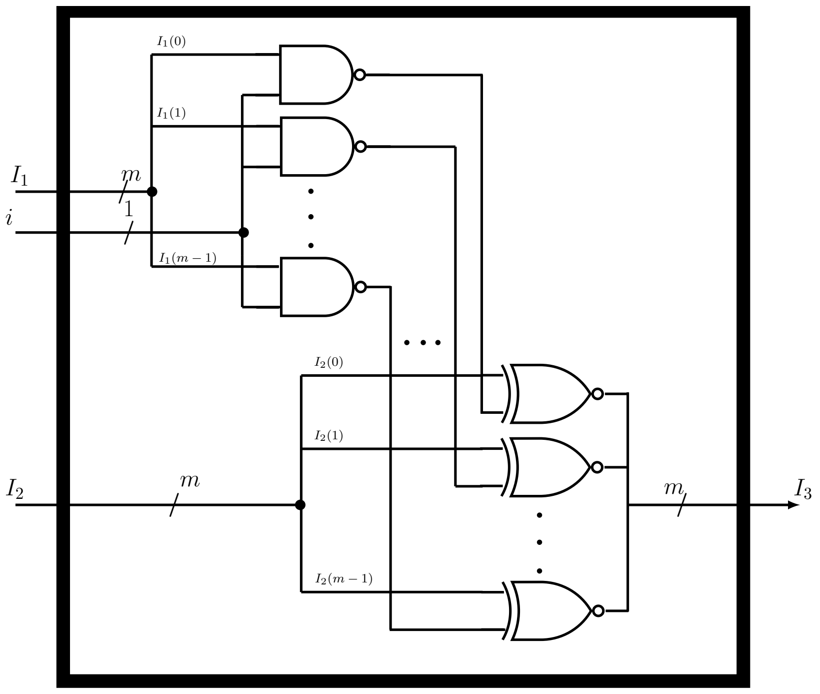

Figure 4.5 Gate level architecture of H block.

requires an array of m NAND gates while the second level requires an array of m XNOR gates.

LSB-first Montgomery Multiplication

Formulations

The formulations for the least significant bit (LSB) first multiplication approach can be developed by considering Eq. 4.19, and then rewriting it using Horner's rule as

$$\begin{aligned} C &= \big(b_{m-1}A + b_{m-2}Ax^{-1} + b_{m-3}Ax^{-2} + ... + b_1Ax^{-(m-2)} + b_0Ax^{-(m-1)}\big) \bmod T(x) \\ &= \big((.....((b_0Ax^{-1} + b_1A)x^{-1} + b_2A)x^{-1} + ... + b_{m-2}A)x^{-1} + b_{m-1}A\big) \bmod T(x) \end{aligned} \tag{4.28}$$

The computation of Eq. 4.28 can be performed using a recursive relation defined as

$$C^{(i)} = C^{(i-1)}x^{-1} \bmod T(x) + b_{i-1}A \tag{4.29}$$

where, $1 \leq i \leq m$, and $C^{(0)} = 0$ and $C^{(m)} = C$. Computation of $C^{(i)}$ in Eq. 4.29 requires the computation of the expression $C^{(i-1)}x^{-1} \bmod T(x)$. Evaluating this expression results in an $(m-1)^{th}$ degree polynomial which can be denoted with $C^{(i-1)}_{\alpha^{-1}}$. It follows that rewriting Eq. 4.29 as,

$$C^{(i)} = C^{(i-1)}_{x^{-1}} + b_{i-1}A \tag{4.30}$$

where $C^{(i-1)}_{x^{-1}}$ can be obtained as follows.

$$\begin{aligned} C^{(i-1)}_{x^{-1}} &= C^{(i-1)}x^{-1} \bmod T(x) \\ &= (C^{(i-1)}_{m-1}x^{m-1} + C^{(i-1)}_{m-2}x^{m-2} + ... + C^{(i-1)}_{1}x + C^{(i-1)}_{0})x^{-1} \bmod T(x) \\ &= C^{(i-1)}_{m-1}x^{m-2} + C^{(i-1)}_{m-2}x^{m-3} + ... + C^{(i-1)}_{1} + C^{(i-1)}_{0}x^{-1} \bmod T(x) \end{aligned} \tag{4.31}$$

Using Eq. 4.18, substitute the expression for $x^{-1} \bmod T(x)$ in Eq. 4.31,

$$\begin{aligned} C^{(i-1)}_{x^{-1}} = C^{(i-1)}_{m-1}x^{m-2} + C^{(i-1)}_{m-2}x^{m-3} + ... + C^{(i-1)}_{1} + C^{(i-1)}_{0}(x^{m-1} \\ + t_{m-1}x^{m-2} + t_{m-2}x^{m-3} + ... + t_1) \\ = C^{(i-1)}_{0}x^{m-1} + (C^{(i-1)}_{m-1} + C^{(i-1)}_{0}t_{m-1})x^{m-2} + (C^{(i-1)}_{m-2} \\ + C^{(i-1)}_{0}t_{m-2})x^{m-3} + ... + (C^{(i-1)}_{1} + C^{(i-1)}_{0}t_1) \end{aligned} \tag{4.32}$$

Based on the fact that addition and multiplication operations in GF(2) can be realized using logical XOR and logical AND operations, respectively, Eq. 4.30 and Eq. 4.32 can be rewritten as

$$C^{(i)} = C^{(i-1)}_{x^{-1}} \oplus (b_{i-1} \wedge A) \tag{4.33}$$

$$\begin{aligned} C^{(i-1)}_{x^{-1}} = C^{(i-1)}_{0}x^{m-1} + (C^{(i-1)}_{m-1} \oplus (C^{(i-1)}_{0} \wedge t_{m-1}))x^{m-2} + (C^{(i-1)}_{m-2} \\ \oplus (C^{(i-1)}_{0} \wedge t_{m-2}))x^{m-3} + ... + (C^{(i-1)}_{1} \oplus (C^{(i-1)}_{0} \wedge t_1)) \end{aligned} \tag{4.34}$$

The term $(b_{i-1} \wedge A)$ represents that each bit of A is AND operated with the bit b_{i-1}. Further modification of Eq. 4.33, with the view that it results in more efficient hardware realization, can be performed as follows.

$$\begin{aligned} C^{(i)} &= C^{(i-1)}_{x^{-1}} \oplus (b_{i-1} \wedge A) \\ &= (\overline{C^{(i-1)}_{x^{-1}}} \wedge (b_{i-1} \wedge A)) \vee (C^{(i-1)}_{x^{-1}} \wedge (\overline{b_{i-1} \wedge A})) \\ &= (\overline{C^{(i-1)}_{x^{-1}}} \wedge (\overline{\overline{b_{i-1} \wedge A}})) \vee (C^{(i-1)}_{x^{-1}} \wedge (\overline{b_{i-1} \wedge A})) \\ &= C^{(i-1)}_{x^{-1}} \odot (\overline{b_{i-1} \wedge A}) \end{aligned} \tag{4.35}$$

Similarly, Eq. 4.34 can also be rewritten as

$$C_{x^{-1}}^{(i-1)} = C_0^{(i-1)}x^{m-1} + (C_{m-1}^{(i-1)} \odot (\overline{C_0^{(i-1)} \wedge t_{m-1}}))x^{m-2} + (C_{m-2}^{(i-1)} \odot (\overline{C_0^{(i-1)} \wedge t_{m-2}}))x^{m-3} + ... + (C_1^{(i-1)} \odot (\overline{C_0^{(i-1)} \wedge t_1})) \quad (4.36)$$

Algorithm

The computation of the Montgomery product C using Eq. 4.35 and Eq. 4.36 is described in algorithm 4.4.

Algorithm 4.4: Modified LSB-first Montgomery multiplication in GF(2^m)

Input: A=$(a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$, B=$(b_{m-1}, b_{m-2}, \ldots, b_1, b_0)$ both w.r.t. polynomial basis, and T' $=(1, t_{m-1}, t_{m-2}, \ldots, t_1)$.

Output: $C = ABx^{-(m-1)} \bmod T(x) = (c_{m-1}, c_{m-2}, \ldots, c_1, c_0)$ also w.r.t. polynomial basis

```
    Initialization : $C^{(0)} \leftarrow 0$;
1: for $i = 1$ to $m$ do
2:     $C_{x^{-1}}^{(i-1)} \leftarrow C^{(i-1)}x^{-1} \odot (\overline{C_0^{(i-1)} \wedge T'})$;     ▷ $C^{(i-1)}x^{-1}$: Right shifting $C^{(i-1)}$ by 1-bit
3:     $C^{(i)} \leftarrow C_{x^{-1}}^{(i-1)} \odot (\overline{b_{i-1} \wedge A})$;
4: end for
5: return $C^{(m)}$
```

Let the partial product $C^{(i)}$ be initialized for $i = 0$ as $C^{(0)} = 0$. The computation of the Montgomery product C is performed in m iterations which is described in Step 1. In each of the i^{th} iteration, one bit of the operand B, b_{i-1}, is involved in the computation. In Step 2 of the algorithm, the term $C_{x^{-1}}^{(i-1)}$ is computed during the i^{th} iteration using the previous partial product term $C^{(i-1)}$ generated during the $(i-1)^{th}$ iteration. This step of the algorithm is based on Eq. 4.36. In Step 3 of the algorithm the partial product $C^{(i)}$ is computed during the i^{th} iteration using the term $C_{x^{-1}}^{(i-1)}$ computed during this i^{th} iteration. This step of the algorithm is based on Eq. 4.35. During the m^{th} iteration, the partial product $C^{(m)}$ is computed which is the required Montgomery product, C.

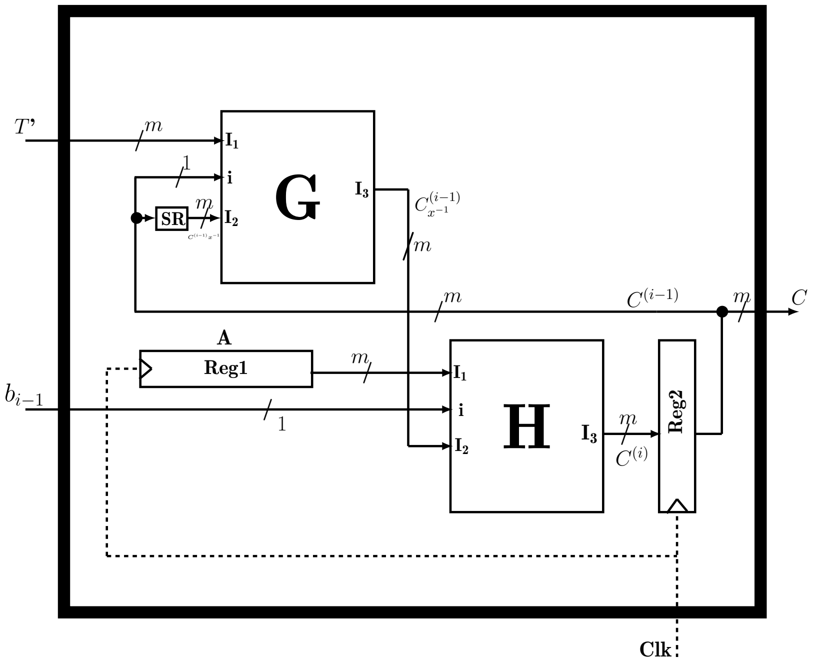

Figure 4.6 Top-level block diagram of the proposed LSB-first bit-serial sequential multiplier.

Architecture

The hardware realization of the proposed LSB-first algorithm (algorithm 4.4) is presented in this section. The proposed architecture, shown in Fig. 4.6, consists of two lower blocks G and H, whose detailed architectures are presented in Fig. 4.4 and Fig. 4.5, respectively. In addition to these two lower blocks, the architecture also includes two m-bit registers and an SR block. The architecture requires an m-bit input, T', a single bit input, b_{i-1}, one of the operands, A, to be preloaded into Reg1, and then m clock cycles in order to generate the required m-bit multiplication product, C. As Step 1 of the algorithm indicates that it requires m iterations to compute the product C, the proposed bit-serial architecture realizes this step requiring m clock cycles where in each clock cycle the iterative computations presented in Step 2 and Step 3 are processed. The computation suggested by Step 2 of the algorithm is realized using the G block and the SR block. The G block has a feedback path through the H block and the register Reg2. During the i^{th} clock cycle, the G block generates the term, $C^{(i-1)}_{x^{-1}}$, using its inputs $C^{(i-1)}x^{-1}$, $C^{(i-1)}_0$ and T'. The term $C^{(i-1)}x^{-1}$ is obtained from $C^{(i-1)}$ using the SR block. The SR block, which shifts the input right by 1-bit position, simply routes the input without requiring any hardware. The computations suggested by Step 3 of the algorithm is realized using the H block and the register Reg1. The H block also has the same feedback path as the G block through the G block and register Reg2. During the i^{th} clock cycle, the H block generates the partial product, $C^{(i)}$, using its inputs A, b_{i-1}, and $C_{x^{-1}}^{(i-1)}$. Furthermore, during the m^{th} clock cycle the H block generates the partial product $C^{(m)}$ which is available at the output after this clock cycle.

The detailed architecture of the two lower blocks G and H are the same as Fig. 4.4 and Fig. 4.5, respectively. The G block along with the SR block realizes the computation of the expression $C^{(i-1)}x^{-1} \odot (\overline{C_0^{(i-1)} \wedge T'})$ using two levels of logic gates. ($C^{(i-1)}x^{-1}$ is realized using the SR block). The first level comprises an array of $(m-1)$ NAND gates while the second level comprises an array of $(m-1)$ XNOR gates. The input i is applied to all NAND gates and also it is routed directly to the output, I_3. The H block realizes the computation of the expression $C_{x^{-1}}^{(i-1)} \odot (\overline{b_{i-1} \wedge A})$ using two levels of logic gates. The first level requires an array of m NAND gates while the second level requires an array of m XNOR gates.

4.3.2 Analytical Results

This section presents the estimation and comparison of area and time complexities of the proposed Montgomery multipliers. Table 4.4 presents the comparison of analytical expressions for area and time complexities of the proposed MSB multiplier with the similar available MSB Montgomery multipliers. These expressions are evaluated for a specific field order $m = 409$ using the estimations of the logic gates from NanGate 45nm libraries, and presented in Table 4.5. Similarly, analytical expressions and evaluations are also presented for the proposed LSB multiplier in Table 4.6 and Table 4.7, respectively.

The analytical expressions for the proposed MSB multiplier in terms of area, latency, and critical path delay are presented in Table 4.4 along with other comparable multipliers [31, 32, 34]. The area complexity for the proposed multiplier and the other multipliers in Table 4.4 are expressed in terms of 2-input AND, 2-input NAND, 2-input XOR/XNOR, and registers. The gate counts for the proposed multiplier are obtained using Figs. 4.3, 4.4, and 4.5. The top level block diagram as shown in Fig. 4.3 contains two sub blocks G and

Table 4.4 Area and time complexities comparison for GF(2^m) (MSB multipliers).

Design	**Basis**	**Architecture Style**	**AND**	**NAND**	**XOR/XNOR***	**Register**	**Latency**	**Critical Path Delay**
[31]	PB	Non-LFSR	$2m-1$	0	$2m-1$	$2m$	m	$T_A + T_X$
[32]	PB	LFSR (Galois)	$2m-1$	0	$2m-1$	$2m$	m	$T_A + T_X$
[34]	WDB	LFSR (Fibonacci)	m	0	$2m-1$	$2m$	m	$T_A + \lceil \log_2 m \rceil T_X$
Proposed	PB	Non-LFSR	0	$2m-1$	$2m-1$	$2m$	m	$T_{NA} + T_{XN}$

*XOR and XNOR have the same area and time complexities, PB: Polynomial basis, WDB: Weakly dual basis, LFSR: Linear feedback shift register

H along with two m-bit registers, Reg1 and Reg2. The G block as shown in Fig. 4.4 contains an array of $(m-1)$ 2-input NAND gates and an array of $(m-1)$ 2-input XNOR gates. The H block shown in Fig. 4.5 contains an array of m 2-input NAND gates and an array of m 2-input XNOR gates. Hence, the proposed MSB Montgomery multiplier architecture requires $(2m-1)$ 2-input NAND gates, $(2m-1)$ 2-input XNOR gates, and $2m$ registers. Note that m represents the order of the GF(2^m) field. The computation of time complexities in terms of critical path delay and latency for the proposed MSB multiplier and other multipliers considered for comparison is performed by assuming T_A, T_{NA}, T_X, and T_{XN} represents the delays of 2-input AND gate, 2-input NAND gate, 2-input XOR gate, and 2-input XNOR gate, respectively. The critical path delay and latency of the proposed MSB multiplier computed from the proposed architecture are $T_{NA} + T_{XN}$ and m clock cycles, respectively.

The analytical comparisons presented in Table 4.4 can be better understood by evaluating them for a specific value of field order m using a specific technology based gate area and time estimations. The field order m is selected to be 409 as it is one of the fields suggested by National Institute of Standards and Technology (NIST) for elliptic curve digital signature algorithm implementation. Furthermore, NanGate 45nm technology-based open cell library statistics [46, 60] are adopted for the gates estimated area and time complexities. Based on these library files the area complexities for the basic gates are given as follows: The area complexities in terms of the NAND gate equivalents (GE) for a 2-input AND gate, a 2-input XOR gate, a 2-input XNOR gate, and a D flip-flop with set/reset capabilities are taken as 1.4, 2, 2, and 5.7, respectively. The delays of a 2-input

Table 4.5 Comparison of area, delay, and area-delay-product for GF(2^{409}) (MSB multipliers).

Design	Critical path delay (ns)	Latency (*clock cycles*)	Delay (ns)	Area (μm^2)	ADP ($\times 10^6$)	% reduction in ADP
[31]	0.06	409	24.54	7440	0.18	16.66
[32]	0.06	409	24.54	7440	0.18	16.66
[34]	0.34	409	139.06	6869	0.95	84.21
Proposed	0.05	409	20.45	7113	0.15	–

NAND gate, a 2-input AND gate, and a 2-input XOR gate, a 2-input XNOR gate, and a D flip-flop with set/reset are 0.015, 0.025, 0.035, 0.035, and 0.060 nanoseconds, respectively. It is observed that the area required for a 2-input NAND gate based on Synopsys design compiler synthesis using 45nm NanGate open cell libraries is $0.8\mu m^2$.

Table 4.5 presents the estimated area and time complexities in terms of number of Area (μm^2), critical path delay (ns), latency (clock cycles), and computational delay (ns) for the field GF(2^m) using FreePDK NanGate 45nm technology based gates estimations. It may be observed that the proposed MSB multiplier requires less area when compared to the multipliers [31] [32]. Though the multiplier [34] requires less area than the proposed MSB multiplier, it requires substantially more delay. It may be noted that the proposed multiplier requires the lowest critical path delay and the lowest delay. Furthermore, the overall performance of these multipliers can be better compared using the balanced parameter area-delay-product (ADP). It may be observed from the table that the proposed MSB multiplier achieves the lowest area-delay-product compared to other multipliers considered for comparison. It is evident from the table (% reduction in ADP column) the proposed MSB multiplier achieves reduction in ADP of around 16%, 16%, and 84%,when compared with multipliers [31], [32], and [34], respectively. Hence, it is clear from the estimated complexities presented in the table that the proposed MSB multiplier is time (critical path delay and computation delay) and area-delay-product efficient.

Similarly, the area and time expressions for LSB multiplier (using Figs. 4.6, 4.4, and 4.5) are derived. This LSB multiplier architecture requires $(2m - 1)$ NAND gates,

Table 4.6 Area and time complexities comparison for the field GF(2^m) (LSB multipliers).

Design	**Basis**	**Architecture Style**	**AND**	**NAND**	**XOR/XNOR**	**Register**	**Latency**	**Critical path**
[31]	PB	Non-LFSR	$2m-1$	0	$2m-1$	$2m$	m	T_A+2T_X
[33]	PB	LFSR (Galois)	$2m-1$	0	$2m-1$	$2m$	m	T_A+2T_X
Proposed	PB	Non-LFSR	0	$2m-1$	$2m-1$	$2m$	m	$T_{NA}+2T_{XN}$

Table 4.7 Comparison of area, delay, and area-delay-product for GF(2^{409}) (LSB multipliers).

Design	**Critical path delay** (ns)	**Latency** ($clock\ cycles$)	**Delay** (ns)	**Area** (μm^2)	**ADP** ($\times 10^6$)	**% reduction in ADP**
[31]	0.095	409	38.86	7440	0.29	13.79
[33]	0.095	409	38.86	7440	0.29	13.79
Proposed	0.085	409	34.77	7113	0.25	–

$(2m-1)$ XNOR gates, and $2m$ registers. These analytical complexities of the proposed LSB multiplier along with the available similar multipliers [31] [33] are presented in Table 4.6. Similar to the proposed MSB multiplier, the analytical expressions of this LSB multiplier along with compared multipliers are evaluated for a specific field order, $m = 409$, and using the NanGate 45nm technology-based standard cell library estimations from Silvaco company [46, 60], and presented in Table 4.7. It is observed from Table 4.7 that the proposed LSB multiplier achieves lower area and lower delay complexities when compared with the multipliers [31] [33].

4.3.3 Implementation Results

The proposed MSB and LSB multipliers, and the best of the multipliers considered for comparison [32] and [33], respectively, are considered for Application Specific Integrated Circuit (ASIC) implementation. These multipliers are modeled using VHDL for the field order of $m = 409$, and verified the functionalities through simulations using the

Table 4.8 Comparison of the ASIC implementation results for GF(2^{409}) (MSB multipliers).

Design	**Multiplier area** (μm^2)	**Critical path delay** (ns)	**Multiplication delay** (ns)	**ADP** $\times$ (10^6) ($\mu m^2 \times ns$)	**% reduction in ADP**
[32]	8593	0.20	81.80	0.70	12.85
Proposed	8322	0.18	73.62	0.61	–

Table 4.9 Comparison of the ASIC implementation results for GF(2^{409}) (LSB multipliers).

Design	**Multiplier area** (μm^2)	**Critical path delay** (ns)	**Multiplication delay** (ns)	**ADP** $\times$ (10^6) ($\mu m^2 \times ns$)	**% reduction in ADP**
[33]	8593	0.32	130.88	1.12	11.60
Proposed	8322	0.29	118.61	0.99	–

Xilinx Vivado simulator. Further, gate level netlists are generated for the VHDL models and synthesized these netlists using the Synopsys Design Compiler tool employing FreePDK NanGate 45nm library files to obtain area and time complexities. Table 4.8 and Table 4.9 present the obtained area and time complexities of all these implemented multipliers.

It is observed from the ASIC implementation results presented in Table 4.8 that the proposed MSB multiplier achieves high-speed (due to less critical path delay), less area, less delay, and area-delay-product improvement of around 12% when compared with the multiplier [32]. Similarly, it is observed from the ASIC implementation results presented in Table 4.9 that the proposed LSB multiplier achieves area-delay-product improvement of around 11% when compared with the multiplier [33]. Hence, the ASIC implementation results confirm that the proposed multipliers achieve better area as well as time complexities compared to the available multipliers in the literature.

4.4 Conclusions

In this chapter, the design of area-efficient bit-serial GF(2^m) multipliers is presented. The design of the proposed multipliers involves the modification of the conventional algorithms using the NAND-XNOR logical relations rather than the AND-XOR relations. It includes the design of the bit-serial sequential multiplier based on the proposed modified interleaved modular reduction algorithm. Further, the performance analysis of this multiplier using analytical and ASIC implementation results is presented. The analysis shows that the proposed multiplier is area and area-delay efficient compared to the available multipliers in the literature. The proposed multipliers also include the design of the MSB-first and the LSB-first multipliers based on the proposed modified Montgomery algorithms. The performance analysis of these multipliers using analytical and ASIC implementation results is presented which shows that the proposed multipliers are area and time efficient compared to the available multipliers in the literature. These proposed area-efficient bit-serial sequential multipliers are highly suitable for domestic IoT end devices, and obiviously not appropriate for high-performance IoT edge devices. Hence, it is also desirable to design high-throughput multiplier architectures such as systolic multipliers to improve the performance of the IoT applications. Consequently, the next chapter focuses on the design of efficient systolic multipliers that are suitable for IoT edge devices.

Chapter 5

Low-Latency and High-Throughput Bit-Parallel Systolic Multipliers for Specific Classes of Trinomials

This chapter presents the design of a few time-efficient systolic multipliers that are suitable for high-performance IoT devices such as IoT edge devices. First, we present the design and analysis of a low-latency area-efficient systolic multiplier using a specific class of trinomials. Further, this multiplier is modified with respect to its architecture to realize another multiplier that achieves high-throughput. This chapter also presents the design and analysis of another low-latency and area-efficient systolic multiplier using a narrow class of trinomials. Analysis of all these proposed multipliers is performed using the comparison of analytical and implementation results with the available multipliers. The analytical comparisons are based on the complexities computed for $m = 409$ using FreePDK NanGate 45nm gate estimations and the implementation comparisons are based on the results obtained from Synopsis Design Compiler employing NanGate 45nm libraries. The comparisons show that proposed multipliers are more time-efficient compared to the available multipliers.

5.1 Introduction

Edge devices which are used in IoT edge computing need to have high-performance implementations since these devices are required to process large volumes of data collected from a large number of end nodes. Hence, the hardware blocks including the finite field

GF(2^m) multipliers that are employed in the implementation of edge devices must have high throughput rates. Design of finite field GF(2^m) multipliers using systolic structures can give high-throughput rates and also offer advantages such as regularity, modularity, and concurrency. The performance of these multipliers can further be improved by selecting low-weight irreducible polynomials such as trinomials or subclasses of trinomials. These irreducible polynomials reduce the computations compared to other types of polynomials and allow high-speed implementations. Hence, it is desirable for IoT edge devices to design efficient systolic multipliers using trinomials or subclasses of trinomials.

Many bit-parallel systolic multipliers are proposed in the literature [27, 39–46] for polynomial basis GF(2^m) multiplication using trinomials/subclasses of trinomials to achieve reduction in area and time complexities. In this work, we consider two classes of trinomials that result in more efficient implementation of GF(2^m) multipliers. The first of these two classes is the class of irreducible trinomials of the form $x^m + x^k + 1$ for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even), and the second class is of the trinomials of the form $x^m + x^k + 1$ for which $k \leq m - 2\lceil m/3 \rceil$. There exists a large number of these classes of trinomials (for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even)/$k \leq m - 2\lceil m/3 \rceil$) for different values of field order m [24]. Moreover, for elliptic curve cryptography, it is recommended that if an irreducible trinomial $x^m + x^k + 1$ exists, then k should be chosen as small as possible i.e. lowest possible degree for the middle term x^k [61]. Also, for the finite fields GF(2^{233}) and GF(2^{409}), National Institute of Standards and Technology (NIST) has recommended the trinomials $x^{233} + x^{73} + 1$ and $x^{409} + x^{87} + 1$, respectively, where the values of k are less than half the value of m [61]. In addition, for the trinomial $x^{409} + x^{87} + 1$ the value of k is also less than the value of $m - 2\lceil m/3 \rceil = 135$.

In this chapter, first, we present a low-latency area-efficient bit-parallel systolic finite field multiplier for the class of trinomials for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). It includes the formulations developed for this multiplier and its architecture realized using a systolic structure employing efficient cutset pipelining techniques. Analysis of this multiplier using analytical and implementation results is also presented. Next, this multiplier is further modified with respect to its architecture using additional horizontal cutset pipelining to achieve a high throughput multiplier. The performance of this multiplier is also verified using analytical and implementation results.

Finally, we present another low-latency area-efficient bit-parallel systolic multiplier using a narrow class of trinomials of the form $x^m + x^k + 1$ for which $k \leq m - 2\lceil m/3 \rceil$. Its formulations and the architecture along with the performance comparisons are also presented. The analytical results for all the proposed multipliers are computed for $m = 409$ using NanGate 45nm gate complexities and compared with the available multipliers [27,39–46]. The implementation results for the proposed multipliers are obtained using Synopsis Design Compiler tool employing NanGate 45nm technology libraries. The comparisons show that the proposed systolic multipliers are more time-efficient compared to the available multipliers.

5.2 Area-Efficient Low-Latency Bit-Parallel Systolic Multiplier

In this section, design and performance analysis of the proposed bit-parallel systolic multiplier for a class of trinomials for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even) are presented. First, we present the mathematical formulations for the proposed GF(2^m) multiplication and its realization using the bit-parallel systolic architecture. Following this, area and time complexities comparisons of analytical and implementation results are presented.

5.2.1 Design

Mathematical Formulation

Consider A and B are two arbitrary elements of the finite field GF(2^m), and C is the product of the elements A and B. Then, the product C can be written as

$$\begin{aligned} C(x) &= (A(x) \times B(x)) \bmod T(x) \qquad (5.1)\\ &= A(x) \sum_{j=0}^{m-1} b_i x^i \bmod T(x) \\ &= \sum_{j=0}^{m-1} b_i A(x) x^i \bmod T(x) \end{aligned}$$

$$= \big(b_{m-1}Ax^{m-1} + b_{m-2}Ax^{m-2} + \text{........} + b_1Ax + b_0A\big) \bmod T(x)$$

$$= \big(b_{m-1}(Ax^{m-1}) + b_{m-2}(Ax^{m-2}) + \ldots\ldots.. + b_1(Ax) + b_0(A)\big) \bmod T(x) \quad (5.2)$$

Equation 5.2 can be rewritten, where the lower part of polynomial whose powers of x from 0 to $(m/2-1)$ and the upper part of polynomial whose powers from $m/2$ to $m-1$ are shown explicitly, as

$$\begin{aligned} = \Bigg(\Bigg(& b_{m-1}(Ax^{m-1}) + b_{m-2}(Ax^{m-2}) + \ldots\ldots..+ \\ & b_{m/2+1}(Ax^{m/2+1}) + b_{m/2}(Ax^{m/2})\Bigg) \\ & + \Bigg(b_{m/2-1}(Ax^{m/2-1}) + b_{m/2-2}(Ax^{m/2-2}) + \ldots\ldots..+ \\ & b_1(Ax) + b_0(A)\Bigg)\Bigg) \bmod T(x) \quad (5.3)\end{aligned}$$

Now, by taking out $x^{m/2}$ as a common term from the upper part, we have

$$\begin{aligned} C(x) = \Bigg(\Bigg(& b_{m-1}(Ax^{m/2-1}) + b_{m-2}(Ax^{m/2-2}) + \ldots\ldots..+ \\ & b_{m/2+1}(Ax) + b_{m/2}(A)\Bigg)x^{m/2} \\ & + \Bigg(b_{m/2-1}(Ax^{m/2-1}) + b_{m/2-2}(Ax^{m/2-2}) + \ldots\ldots..+ \\ & b_1(Ax) + b_0(A)\Bigg)\Bigg) \bmod T(x) \quad (5.4)\end{aligned}$$

Now, consider the following recursive notations, $A^{(0)} = A$, $A^{(1)} = A^{(0)}x$, $A^{(2)} = A^{(1)}x$, $A^{(3)} = A^{(2)}x$, ..., $A^{(m-1)} = A^{(m-2)}x$. From these recursive relations, it can be defined that

$$A^{(j)} = A^{(j-1)}x \bmod T(x) \quad (5.5)$$

where,

$$A^{(j-1)} = \sum_{i=0}^{m-1} a_i^{(j-1)}x^i, \forall\, a_i^{(j-1)} \in GF(2) \quad (5.6)$$

Furthermore, from the fact that irreducible polynomial $T(x) = 0$, it is implied that

$$x^m = t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + \ldots\ldots. + t_2x^2 + t_1x + t_0 \quad (5.7)$$

The expression for the left side of Eq. 5.5 can be written as

$$A^{(j)} = a_{m-1}^{(j)}x^{m-1} + a_{m-2}^{(j)}x^{m-2} + \ldots.. + a_2^{(j)}x^2 + a_1^{(j)}x + a_0^{(j)} \quad (5.8)$$

The coefficients of $A^{(j)}$ can be obtained in terms of the coefficients of $A^{(j-1)}$ as

$$a_0^{(j)} = a_{m-1}^{(j-1)} \tag{5.9}$$

$$a_i^{(j)} = a_{i-1}^{(j-1)} + a_{m-1}^{(j-1)} t_i, \text{where } i = 1, 2, ..., m-1 \tag{5.10}$$

With these implications, the expression for $C(x)$ in Eq. 5.4 can be rewritten as

$$\begin{aligned} C(x) = ((b_{m-1}(A^{(m/2-1)}) + b_{m-2}(A^{(m/2-2)}) + ...+ \\ (b_{m/2+1}(A^{(1)}) + b_{m/2}(A^{(0)}))x^{m/2} \\ + (b_{m/2-1}(A^{(m/2-1)}) + b_{m/2-2}(A^{(m/2-2)}) + ...+ \\ (b_1 A^{(1)}) + b_0(A^{(0)}))) \bmod T(x) \end{aligned} \tag{5.11}$$

The formulation for the product $C(x)$ expression in Eq. 5.11 contains two parts, part1 as

$$\begin{aligned} ((b_{m-1}(A^{(m/2-1)}) + b_{m-2}(A^{(m/2-2)}) + ...+ \\ (b_{m/2+1}(A^{(1)}) + b_{m/2}(A^{(0)})) \bmod T(x) \end{aligned} \tag{5.12}$$

and part2 as

$$\begin{aligned} (b_{m/2-1}(A^{(m/2-1)}) + b_{m/2-2}(A^{(m/2-2)}) + ...+ \\ (b_1 A^{(1)}) + b_0(A^{(0)}))) \bmod T(x) \end{aligned} \tag{5.13}$$

It may be observed that Eq. 5.12 and Eq. 5.13 are similar in computation, and these can be combined to compute the $C(x)$ (See Eq. 5.11). It is noted that the computational complexity of Eq. 5.11 depends on the reduction polynomial $T(x)$. The complexity can be reduced by selecting trinomials as reduction polynomials. It is also observed that for certain class of trinomials, $x^m + x^k + 1$, where $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even), the computational complexity further decreases. It is possible from the fact that for the class of trinomials specified, the product of any field element A and x^k, where $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even), can be computed using either simple one level of binary addition or permutation of the field element coordinates. To explain this point clearly, an example is presented in Table 5.1 for GF(2^{10}) over a trinomial $x^{10} + x^3 + 1$. It is evident from this table that the computation of the coordinates of Ax^i, until $m/2$ ($= 5$) powers of x, involves either simple permutation or one level of binary addition of the field element A coordinates ($[a_9, a_8, a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0]$).

Table 5.1 Coordinate representation of $Ax^i \bmod T(x)$.

i	$Ax^i \bmod T(x)$
1	$[a_8, a_7, a_6, a_5, a_4, a_3, a_2 \oplus a_9, a_1, a_0, a_9]$
2	$[a_7, a_6, a_5, a_4, a_3, a_2 \oplus a_9, a_1 \oplus a_8, a_0, a_9, a_8]$
3	$[a_6, a_5, a_4, a_3, a_2 \oplus a_9, a_1 \oplus a_8, a_0 \oplus a_7, a_9, a_8, a_7]$
4	$[a_5, a_4, a_3, a_2 \oplus a_9, a_1 \oplus a_8, a_0 \oplus a_7 \oplus a_6, a_9, a_8, a_7, a_6]$
5	$[a_4, a_3, a_2 \oplus a_9, a_1 \oplus a_8, a_0 \oplus a_7 \oplus a_6, a_9 \oplus a_5, a_8, a_7, a_6, a_5]$

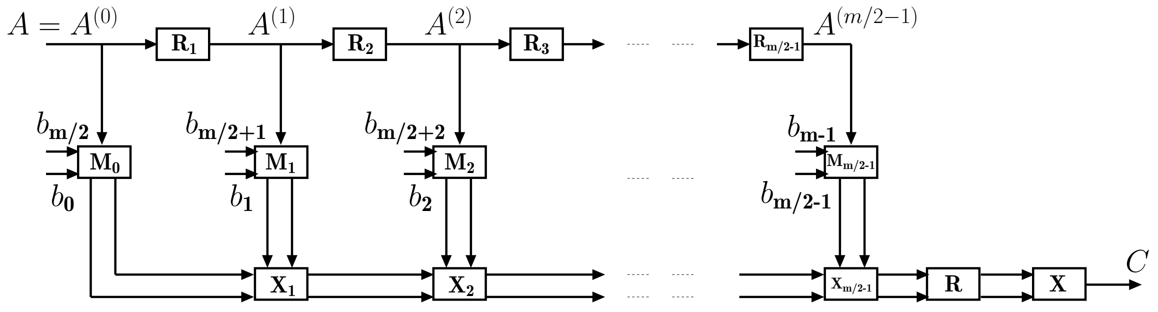

Figure 5.1 The signal flow graph (SFG) of the proposed finite field GF(2^m) multiplication.

Maximum number of computations are involved for the considered class of trinomial fields GF(2^m) when the reduction polynomial is of the form $x^m + x^{m/2} + 1$ (see last row of Table 5.1). For all other trinomials of interest, $x^m + x^k + 1$ with $k < m/2$, the number of computations are lesser than the maximum number of computations (see the cases for $i = 1, 2, 3, 4$ in Table 5.1).

Proposed Systolic Multiplier for a Class of Trinomials

The proposed systolic multiplier over GF(2^m) for a class of trinomials for which the power of second highest degree term of the irreducible polynomial is less than or equal to half the field order m, is presented in this section.

In the proposed multiplier design, we consider m to be even for keeping $m/2$ to be an integer for simplicity. The equation specified by Eq. 5.11 can be represented with a signal flow graph (SFG) as shown in Fig. 5.1. The SFG consists of $(m/2 - 1)$ reduction of degree by one nodes R$_i$ ($i = 1, 2, ...m/2 - 1$), $m/2$ number of multiplication nodes M$_i$

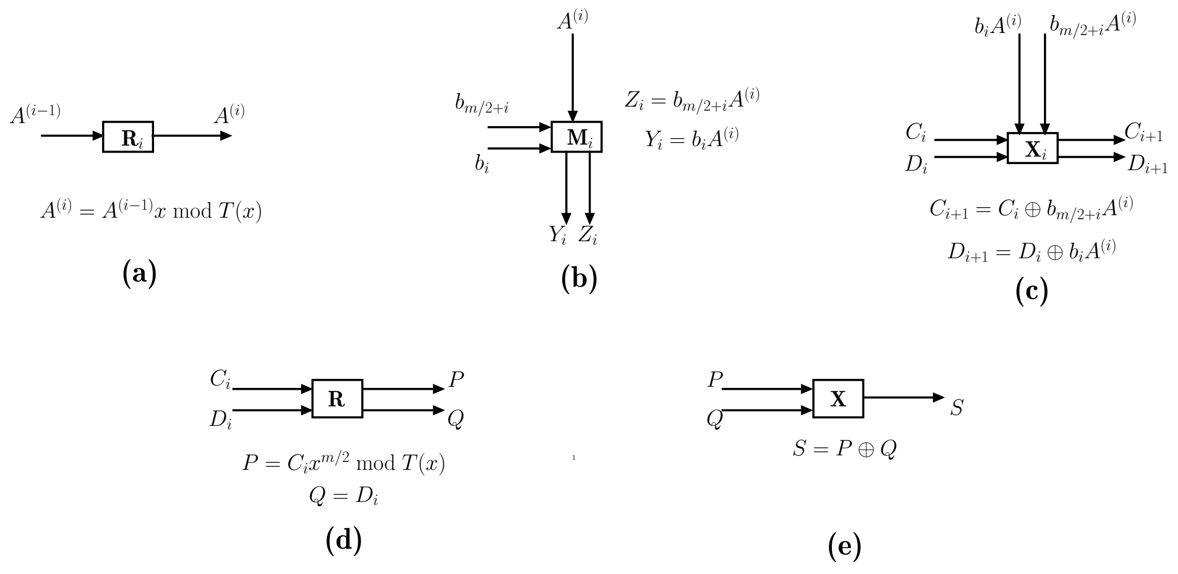

Figure 5.2 a) Functional description of the i^{th} reduction of degree by one node R$_i$. b) Functional description of the i^{th} multiplication node M$_i$. c) Functional description of the i^{th} addition node X$_i$. d) Functional description of the reduction of degree by $m/2$ node R. e) Functional description of the addition node X.

$(i = 0, 1, 2, ...m/2-1)$, and $(m/2-1)$ addition nodes X$_i$ $(i = 1, 2, ...m/2-1)$. In addition, the SFG also contains a reduction of degree by $m/2$ node R and an addition node X. The functional description of the nodes is presented in Fig. 5.2.

The reduction of degree by one node R$_i$ shown in Fig. 5.2(a) performs modulo reduction operation. It takes $A^{(i-1)}$, the reduced form of A, as input and performs modulo $T(x)$ multiplication by x on it. The multiplication node M$_i$ realizes the formulations presented in Fig. 5.2(b). It performs the two sets of multiplications, where for each multiplication input $A^{(i)}$ is multiplied by a specific bit of input $B(b_i$ or $b_{m/2+i})$. Figure 5.2(c) presents the functional description of addition node X$_i$ which performs two sets of m bit additions, where in each case one input from left and one respective input from the top are added as suggested by the formulations presented. The reduction node R in Fig. 5.2(d) performs modulo $T(x)$ multiplication by $x^{m/2}$ on the upper input, while the lower input to this node remains unchanged. The functional description of the addition node X is given in Figure 5.2(e), which computes the addition of its two m-bit inputs.

It may be noted that Eq. 5.11 has two parts (Eq. 5.12 and Eq. 5.13) where each part is a summation of $m/2$ product terms. The SFG (See Fig. 5.1) successively generates these product terms and aggregates their sum. In this SFG, $A^{(i)}$ generated by node R$_i$

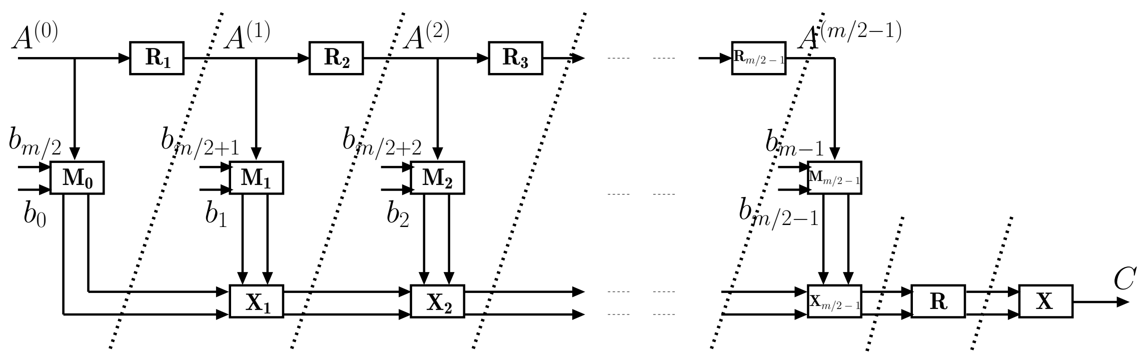

Figure 5.3 The pipelined SFG of the proposed finite field GF(2^m) multiplier.

is used by the multiplication M$_i$ and addition X$_i$ nodes. The combined effect of these three respective nodes R$_i$, M$_i$, and X$_i$ is equivalent to the generation of i^{th} product term and aggregating it to the previous $(i-1)^{th}$ aggregation. Once two parts of Eq. 5.11 are generated, they are combined according to this equation using the nodes R and X. The multiplication of Eq. 5.12 by $x^{m/2}$ as suggested by Eq. 5.13 is realized by node R while the summation of the reduced form of Eq. 5.12 with Eq. 5.13 is performed by the node X.

The input A applied to the SFG (Fig. 5.1) is processed at a stretch through the nodes to give the output C. Hence, the long critical path of the SFG can be reduced by applying suitable cutset pipelining on it. Figure 5.3 shows the pipelined SFG obtained by applying feed-forward cutset, which involves inserting storage elements at cutset points. The cutset performed here eliminates the dependency, in terms of sequential processing, between reduction node R$_i$ and its corresponding multiplication M$_i$ and addition X$_i$ nodes to generate the i^{th} product term. The processing section between any two adjacent cutset lines can be called a processing element (PE). The formation of all the PEs obtained by the cutset pipelining is shown in Fig. 5.4. Each pair of adjacent PEs are to be separated by the storage elements as inferred by the cutset lines.

The proposed systolic structure derived from the SFG in Fig. 5.4 for GF(2^m) multiplier is shown in Fig. 5.5. There are a total of $(m/2+2)$ PEs in the structure. The structure contains five types of PEs. The regular PEs from PE[1] to PE[$m/2-2$] are of the same type. The first PE, PE[0], and the last three PEs, PE[$m/2-1$], PE[$\alpha^{m/2}$], and PE[Out] are all individually distinct PE types. The functionality of each type of PEs is described in Fig. 5.6. The first processing element PE[0] presented in Fig. 5.6(a) performs

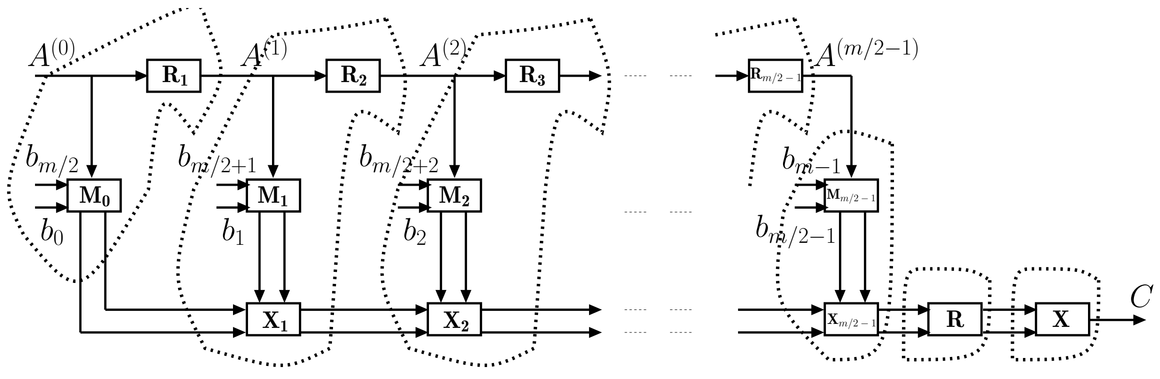

Figure 5.4 Formation of the processing elements (PEs).

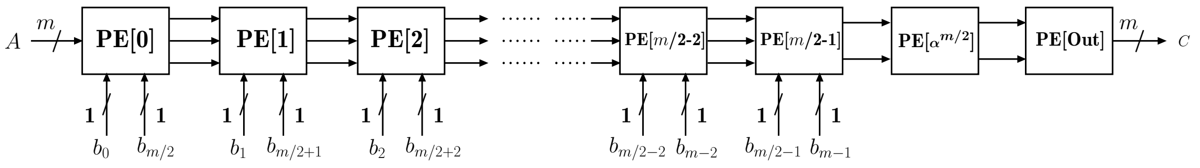

Figure 5.5 Proposed systolic structure of the GF(2^m) multiplier.

the modular reduction of degree by one operation. It also performs one more type of operation where a one-bit input ($Y1in$ or $Y2in$) is logically ANDed with every bit of m-bit input (Xin). The functionality of this PE (PE[0]) is represented by the corresponding mathematical equations in the same figure. The functionality of regular PE, applicable to PE[1] to PE[m/2-2], is presented in Fig. 5.6(b). This PE, in addition to operations described for PE[0], also performs two m-bit additions as described by the corresponding equations presented in the figure. As shown in Fig. 5.6(c), the function of PE[m/2-1] same as regular PE except for the absence of the modulo reduction operation. The processing element PE[$x^{m/2}$] as shown in Fig. 5.6(d) performs modulo reduction of degree by $m/2$ on its upper input while lower input remains unchanged. The last processing element PE[Out] shown in Fig. 5.6(e) performs the addition of its two m-bit inputs.

The detailed architecture of the regular PE is presented in Fig. 5.7. The reduction unit RU comprises a single XOR gate for the reduction of degree by one operation. In the implementation of RU, the placement of the XOR gate is suggested by k value (See the first row of Table 5.1, where $k = 3$). The left AND cell comprises an array of m two-input AND gates where i^{th} AND gate takes one input as $Y1in$ while another input is i^{th} bit of $X1in$ for $i = 1, 2, ..., m$. Similarly, the right AND cell also designed to perform the

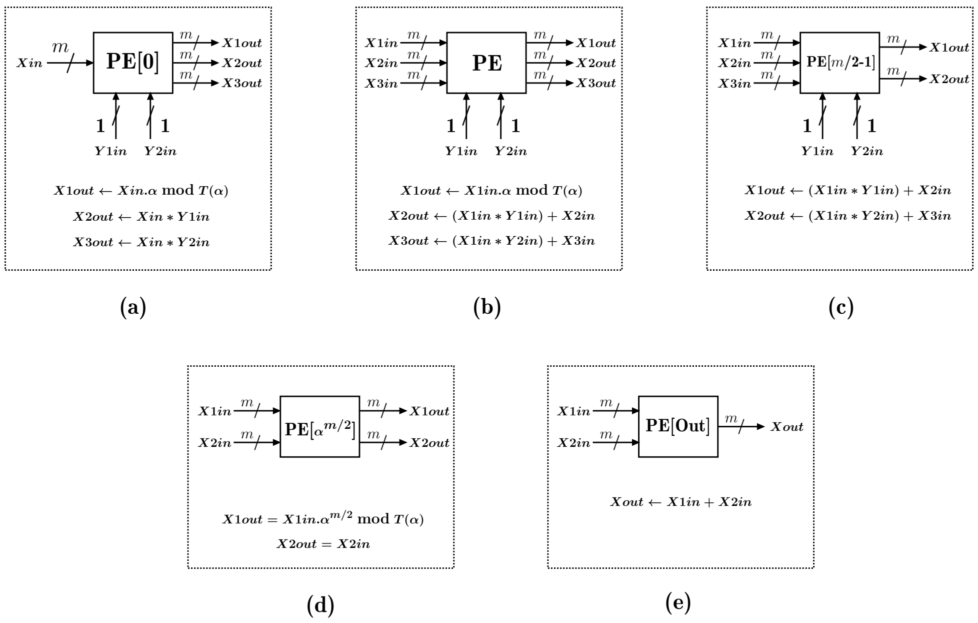

Figure 5.6 a) Functional description of the PE[0] node. b) Functional description of the regular PE (PE[1] to PE[$m/2-2$]). c) Functional description of the PE[$m/2-1$]. d) Functional description of PE[$\alpha^{m/2}$] node. e) Functional description of the PE[Out] node.

same function on $Y2in$. The XOR cell comprises an array of m-two input XOR gates. Each XOR cell is responsible for the addition of one of the product terms generated by the AND gate to the $X2in/X3in$ input of the processing element. In addition, since the functions of processing elements PE[0] and PE[$m/2-1$] are sub-functions for regular PE (See Fig. 5.6), the architectures for PE[0] and PE[$m/2-1$] can be derived by modifying the architecture of regular PE. The PE[$x^{m/2}$] comprises an array of $m/2$ XOR gates (See the last row of Table 5.1, where $m = 10$). The architecture of the PE[out] comprises an array of $m/2$ XOR gates to realize an m-bit addition. The proposed multiplier generates one output for every clock cycle with an initial latency of $(m/2+2)$ clock cycles.

5.2.2 Analytical Results

In this section, the area and time complexities of the proposed bit-parallel systolic multiplier are estimated and compared with that of similar multipliers available in the literature. Table 5.2 presents the analytical expressions for area and time complexities of the proposed multiplier and similar multipliers considered for comparison. Furthermore, these analytical expressions are computed for $m = 409$ using the area and time complexity

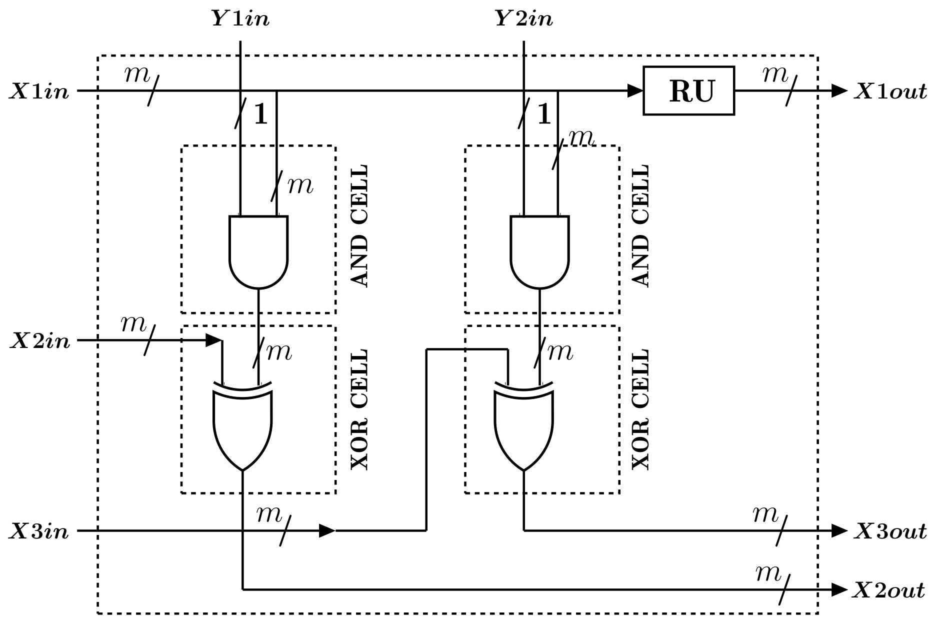

Figure 5.7 Detailed architecture of the regular PE.

estimations of logic gates from FreePDK45 NanGate open cell library statistics [46, 60] and presented in Table 5.3.

Table 5.2 presents the analytical comparison of area complexity, latency, and critical path delay of the proposed systolic multiplier with the available systolic multipliers considered for comparison [27, 39–46]. All these multipliers including the proposed multiplier are applicable for either general trinomials or a class of trinomials. The area required for the proposed multiplier is computed in terms of the number of AND gates, XOR gates, and registers. We also present the area complexity for the other multipliers in a similar way to compare with that of the proposed multiplier. The time complexities of the proposed multiplier and other multipliers considered for comparison are computed by assuming that T_A, T_X, and T_{NA} denote the delays of 2-input AND gate, 2-input XOR gate, and 2-input NAND gate, respectively.

The area and time complexities of the proposed multiplier can be computed from the detailed gate level circuitry of all PEs of the structure shown in Fig. 5.5. The structure in this figure contains $(m/2+2)$ PEs. The first processing element PE[0] requires one XOR gate and $2m$ AND gates. Each one of the regular PEs from PE[1] to PE[$m/2-2$] requires $2m+1$ XOR gates and $2m$ AND gates. The PE[$m/2-1$] needs $2m$ XOR gates and $2m$ AND gates. The last two PEs, PE[$x^{m/2}$] and PE[Out], require $m/2$ XOR gates and m

Table 5.2 Area and time complexities comparison for GF(2^m).

Design	**AND**	**XOR**	**Register**	**Latency**	**Critical Path**
[39]	$(m+1)^2$	$(m+1)^2$	$4(m+1)^2$	$m+1$	T_A+T_X
[40]	m^2	m^2+m-1	$3m^2+2m-2$	$2m-1$	T_A+T_X
[41]	m^2	m^2+ml	$4m^2+2lm$	$m+l-1$	T_A+T_X
[42]	$(3m^2-m)/2$	m^2+m	$4m^2+m$	$m+1$	T_A+T_X
[43]	m^2	m^2+m	$3.5m^2+3m$	$m+2$	T_A+T_X
[27]	m^2	m^2-1	$2m(m-1)$	m	T_A+T_X
[44]	m^2	m^2+mt^a-m	$2m^2$	m	T_A+2T_X
[45]	m^2	m^2+mt^a-m	$2m^2$	$m/2+2$	$2T_X$
[46]	m^{2*}	$1.5m^2+0.5m$	$1.5m^2+2m-1$	$m+2$	$T_{NA}+T_X$
Proposed	m^2	m^2-1	$1.5m^2+m$	$m/2+2$	T_A+T_X

$l=\lfloor(m-2)/(m-k)\rfloor+1$.

* m^2 NAND gates along with $(1.5m^2-2.5m+3)$ number of inverters.

^{a} t=1 for the class of polynomials considered in this work.

XOR gates, respectively. Hence, the proposed systolic structure requires m^2 AND gates, (m^2-1) XOR gates and $(1.5m^2+m)$ number of registers. Latency and Critical path (See Fig. 5.7) of the proposed multiplier structure are $(m/2+2)$ and (T_A+T_X), respectively. It may be observed from Table 5.2 that all the multipliers in the table require nearly the same critical path delay. However, the proposed systolic multiplier has the lowest latency. The multiplier [45] has nearly same time complexities as the proposed multiplier (See Latency and Critical path columns of Table 5.2), but it requires more amount of hardware. In addition, it may be noted that the proposed multiplier requires the least number of registers suggesting less area requirement.

The analytical comparisons presented in Table 5.2 can be better understood by evaluating them for a specific value of the field order along with a specific technology-based area and time complexity estimations of gates. The field order, m, can be selected as 409, which is one of the field sizes recommended by NIST for Elliptic curve cryptographic applications [61]. For the estimation of area and time complexities of the gates, NanGate 45nm technology-based open cell library statistics [46, 60] is adopted as follows: The area complexities in terms of the NAND gate equivalents (GE) for a NOT gate, a 2-input

Table 5.3 Area and time complexities comparison for GF(2^{409}).

Design	**Critical Path** (ns)	**Latency** ($clock\ cycles$)	**Latency** (ns)	**Area** ($\times 10^6$) (μm^2)	**% reduction in Latency** (ns)	**% reduction in Area**
[39]	0.06	410	24.60	3.5	50	54
[40]	0.06	817	49.02	2.7	75	41
[41]	0.06	412	24.72	3.8	50	56
[42]	0.06	410	24.60	4.5	49	64
[43]	0.06	411	24.66	3.1	50	48
[27]	0.06	409	24.54	2	49	20
[44]	0.09	409	36.81	2	72	20
[45]	0.07	207	14.49	2	14	20
[46]	0.05	411	20.55	1.8	40	11
Proposed	0.06	207	12.42	1.6	–	–

AND gate, a 2-input XOR gate, and a D flip-flop with set/reset capabilities are taken as 0.5, 1.4, 2, and 5.7, respectively. The delays of a 2-input NAND gate, a 2-input AND gate, and a 2-input XOR gate are 0.015, 0.025, and 0.035 nanoseconds, respectively. It is observed that the area required for a 2-input NAND gate based on Synopsys design compiler synthesis using 45nm NanGate open cell libraries is $0.8\mu m^2$.

Table 5.3 presents the comparison of estimated critical path delay, latency, and area complexity of the proposed multiplier with that of the same multipliers considered for comparison in Table 5.2. It is observed that the proposed multiplier requires the lowest area. It is clear from Table 5.3 (% reduction in Area column) that the proposed multiplier achieves area efficiency of 54%, 41%, 56%, 64%, 48%, 20%, 20%, 20%, and 11% when compared with multipliers [39], [40], [41], [42], [43], [27], [44], [45], [46], respectively. It is observed that the multipliers [27, 39–43] have similar critical path delays when compared with the proposed multiplier. However, the proposed multiplier achieves low latency (ns) compared to all these multipliers. The critical path of the multiplier [46] is less than that of the proposed multiplier, but its latency (ns) is higher than the proposed multiplier. Though the multiplier [45] requires a similar number of clock cycles to generate the first output as the proposed multiplier, its latency (ns) is higher. From Table 5.3, it is observed that the proposed multiplier achieves the lowest latency (ns). Hence, it is clear from the estimated values presented in Table 5.3 that the proposed multiplier achieves both low-latency and low-area when compared with the similar multipliers available in the

Table 5.4 Comparison of ASIC implementation results for GF(2^{409}).

Design	**Critical Path (ns)**	**Latency (ns)**	**Multiplier Area (μm^2)**	**% reduction in Latency (ns)**	**% reduction in Area**
[45]	0.24	49.68	3052297	12	17
[46]	0.17	78.09	2783963	37	9
Proposed	0.21	43.47	2533407	–	–

literature.

5.2.3 Implementation Results

It is observed from Table 5.3 that the multipliers [45, 46] require less latency and less area, respectively, compared to the other available multipliers considered for comparison. Hence, the proposed multiplier and the two multipliers [45, 46] are modeled using VHDL for GF(2^{409}). The RTL models are simulated using Vivado Simulator to verify the functionality. Also, these RTL models are synthesized using the Synopsys Design Compiler tool employing NanGate 45nm open cell library [60] to obtain the area and time complexities. The area and time complexities obtained for all the three multipliers are tabulated in Table 5.4. The experimental results obtained confirm that the proposed multiplier requires less area and less latency (ns) than the other previous multipliers.

5.3 High-Throughput Area-Delay-Efficient Bit-Parallel Systolic Multiplier

This section presents a high-throughput multiplier whose architecture is realized using the formulations of the area-efficient low-latency multiplier presented in the previous section. The formulations for this multiplier are the same as the previous multiplier and the architecture realization involves further cutset pipelining compared to the previous multiplier. This section also presents the performance analysis of this proposed high-throughput multiplier through analytical and implementation results comparisons with the available related multipliers.

5.3.1 Design

Mathematical Formulation

The formulations for this multiplier are the same as the formulations presented for the multiplier in the previous section (Section 5.2.1). Similar to the previous multiplier case, this multiplier is also applicable for the same class of trinomials $x^m + x^k + 1$ for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even).

Proposed Systolic Architecture

This section presents the proposed GF(2^m) multiplier systolic structure, which is applicable for the specified class of trinomials for which the degree of the middle term is less than half of the order of the field.

Consider the pipelined signal flow graph (SFG) presented in Fig. 5.3. The critical path delay for this pipelined SFG shown in Fig. 5.3 is $\max(T_{\mathrm{R}_i N}, T_{\mathrm{M}_i\mathrm{N}} + T_{\mathrm{X}_i\mathrm{N}}, T_{\mathrm{RN}}, T_{\mathrm{XN}})$. However, it is clear from the hardware implementation point of view that $T_{\mathrm{M}_i\mathrm{N}} + T_{\mathrm{X}_i\mathrm{N}}$ constitutes the critical path delay. This delay can be further reduced by applying a horizontal cutset as shown in Fig. 5.8. The critical path delay for this pipelined SFG shown in Fig. 5.8 is $\max(T_{\mathrm{R}_i\mathrm{N}}, T_{\mathrm{M}_i\mathrm{N}}, T_{\mathrm{X}_i\mathrm{N}}, T_{\mathrm{RN}}, T_{\mathrm{XN}})$. It may be noted that wherever the cutset lines intersect the SFG, hardware registers are to be placed in the hardware implementation. Based on the vertical cutsets, the processing elements (PEs) can be formed for the pipelined SFG (Fig. 5.8) as shown in Fig. 5.9.

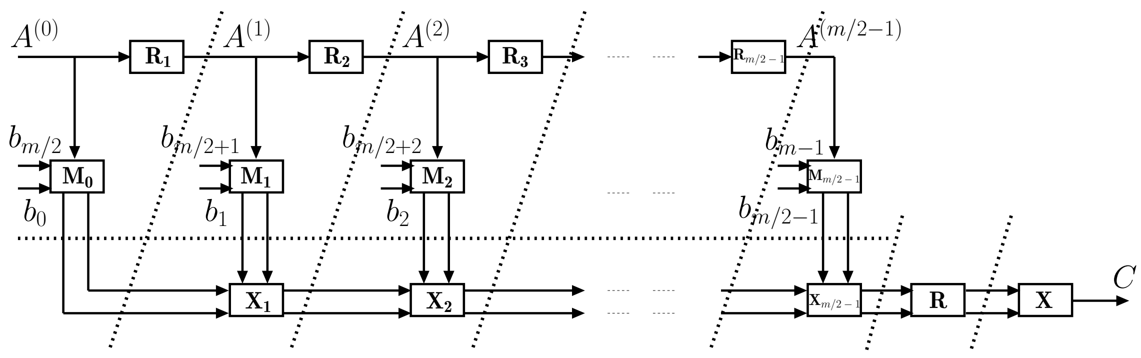

Figure 5.8 Further pipelined SFG using horizontal cutset.

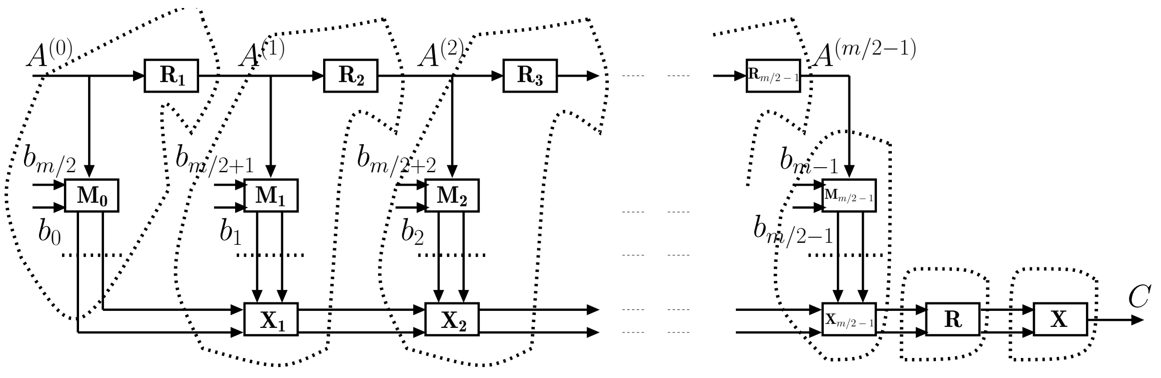

Figure 5.9 Formation of the processing elements (PEs).

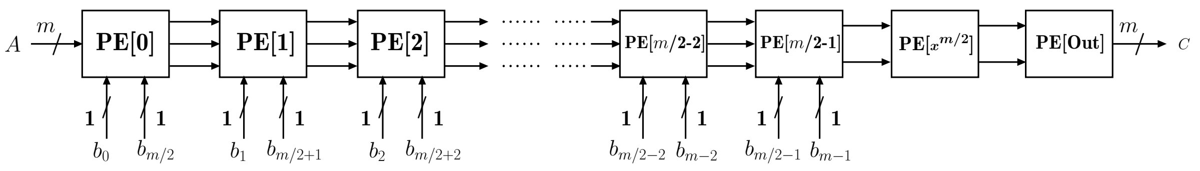

Figure 5.10 Proposed systolic structure of the GF(2^m) multiplier.

The proposed systolic structure for the GF(2^m) multiplier derived from the SFG (Fig. 5.9) is shown in Fig. 5.10. This structure has five types of processing elements (PEs). The processing elements PE[1] through PE[m/2-2] are similar in architecture and can be referred to as regular PEs. The remaining processing elements PE[0], PE[m/2-1], PE[$x^{m/2}$], and PE[Out] are different from regular PEs, and also individually distinct from one another. The functional descriptions of all the PEs are presented in Fig. 5.11. It may be noted that processing elements PE[0] through PE[m/2-1] are required to include two m-bit registers in the architecture to realize the horizontal cutset which is described in Fig. 5.8. In Fig. 5.11, the subscript, (t), used for the input and output variables represents the current clock cycle and the subscript ($t-1$) indicates the previous clock cycle when implemented the structure in hardware. For the PE[0] shown in Fig. 5.11(a), $Xin_{(t)}, Y1in_{(t)}$, and $Y2in_{(t)}$ are the inputs. The output $X1out_{(t)}$ is obtained by performing reduction of degree by one operation on the input $Xin_{(t)}$. The output $X2out_{(t)}$ is obtained by multiplying the previous m-bit input $Xin_{(t-1)}$, with the previous 1-bit input $Y1in_{(t-1)}$. Here, the previous inputs means that the inputs available to PE[0] during the previous clock cycle. Similarly, the output $X3out_{(t)}$ is obtained by multiplying the previous m-bit input $Xin_{(t-1)}$, with the previous 1-bit input $Y2in_{(t-1)}$. Figure 5.11(b) presents the functional description of a regular PE. When compared with PE[0], it has two more inputs

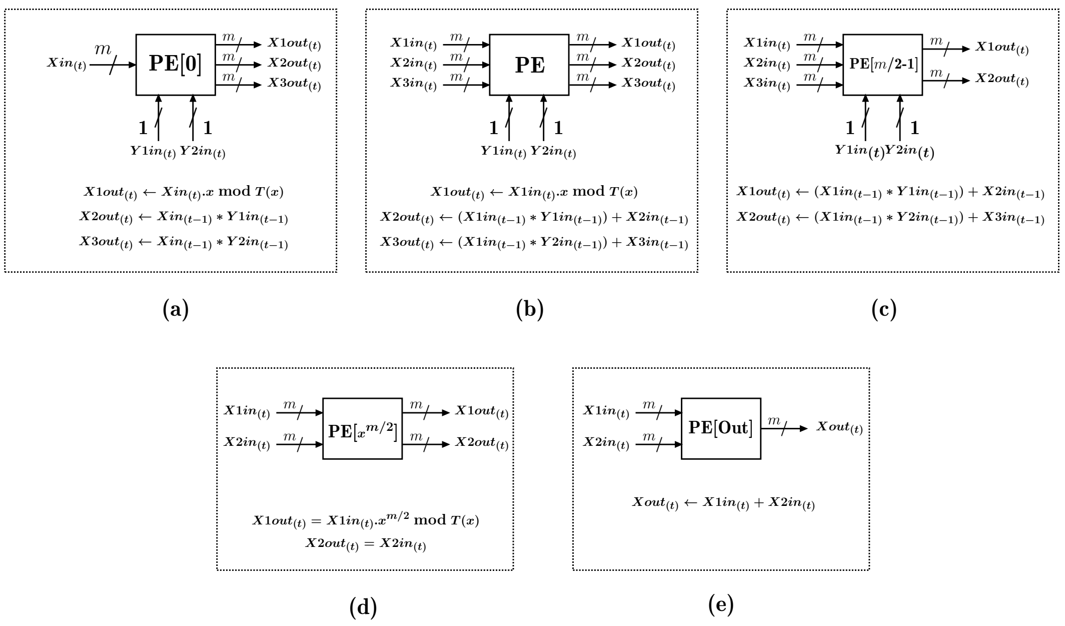
$Xin_{(t)}$
m
PE[0]
$X1out_{(t)}$
$X2out_{(t)}$
$X3out_{(t)}$
1
1
$Y1in_{(t)}$ $Y2in_{(t)}$
$X1out_{(t)} \leftarrow Xin_{(t)}.x \bmod T(x)$
$X2out_{(t)} \leftarrow Xin_{(t-1)} * Y1in_{(t-1)}$
$X3out_{(t)} \leftarrow Xin_{(t-1)} * Y2in_{(t-1)}$
(a)
$X1in_{(t)}$
$X2in_{(t)}$
$X3in_{(t)}$
PE
$X1out_{(t)}$
$X2out_{(t)}$
$X3out_{(t)}$
$Y1in_{(t)}$ $Y2in_{(t)}$
$X1out_{(t)} \leftarrow X1in_{(t)}.x \bmod T(x)$
$X2out_{(t)} \leftarrow (X1in_{(t-1)} * Y1in_{(t-1)}) + X2in_{(t-1)}$
$X3out_{(t)} \leftarrow (X1in_{(t-1)} * Y2in_{(t-1)}) + X3in_{(t-1)}$
(b)
PE[m/2-1]
$X1out_{(t)} \leftarrow (X1in_{(t-1)} * Y1in_{(t-1)}) + X2in_{(t-1)}$
$X2out_{(t)} \leftarrow (X1in_{(t-1)} * Y2in_{(t-1)}) + X3in_{(t-1)}$
(c)
PE[$x^{m/2}$]
$X1out_{(t)} = X1in_{(t)}.x^{m/2} \bmod T(x)$
$X2out_{(t)} = X2in_{(t)}$
(d)
PE[Out]
$Xout_{(t)}$
$Xout_{(t)} \leftarrow X1in_{(t)} + X2in_{(t)}$
(e)

Figure 5.11 a) Functional description of the PE[0] node. b) Functional description of the regular PE (PE[1] to PE[$m/2-2$]). c) Functional description of the PE[$m/2-1$]. d) Functional description of the PE[$x^{m/2}$] node. e) Functional description of the PE[Out] node.

$X2in_{(t)}$ and $X3in_{(t)}$. The output $X2out_{(t)}$ which realizes the accumulation is obtained by multiplying the previous m-bit input $X1in_{(t-1)}$, with the previous 1-bit input $Y1in_{(t-1)}$, followed by adding the previous m-bit input $X2in_{(t-1)}$. Similarly, the output $X3out_{(t)}$ can be obtained. The operations performed by PE[m/2-1] are similar to the regular PE, however, it does not perform reduction of degree by one operation. For the PE[$x^{m/2}$], $X1out_{(t)}$ is obtained by performing reduction of degree by $x^{m/2}$ operation on the input $X1in_{(t)}$. The other input of this PE simply routs to output without any modification. The processing element PE[Out] adds its two m-bit inputs $X1in_{(t)}$ and $X2in_{(t)}$ to generate the output $Xout_{(t)}$. The processing elements PE[$x^{m/2}$] and PE[Out] do not require any storage elements in the architectures.

The detailed architecture for the regular PE is shown in Fig. 5.12. The reduction unit RU, which performs the reduction of degree by one operation, comprises a single XOR gate. The placement of this single XOR gate in the RU block is determined by the k value (See the first row of Table 5.1, where $k = 3$). The left AND cell consists of an array of m two-input AND gates where i^{th} AND gate takes one input as $Y1in$ while another input is i^{th} bit of $X1in$ for $i = 1, 2, ..., m$. Similarly, the right AND cell also designed to perform the same function on $Y2in$. The XOR cell consists of an array of m two-input XOR gates.

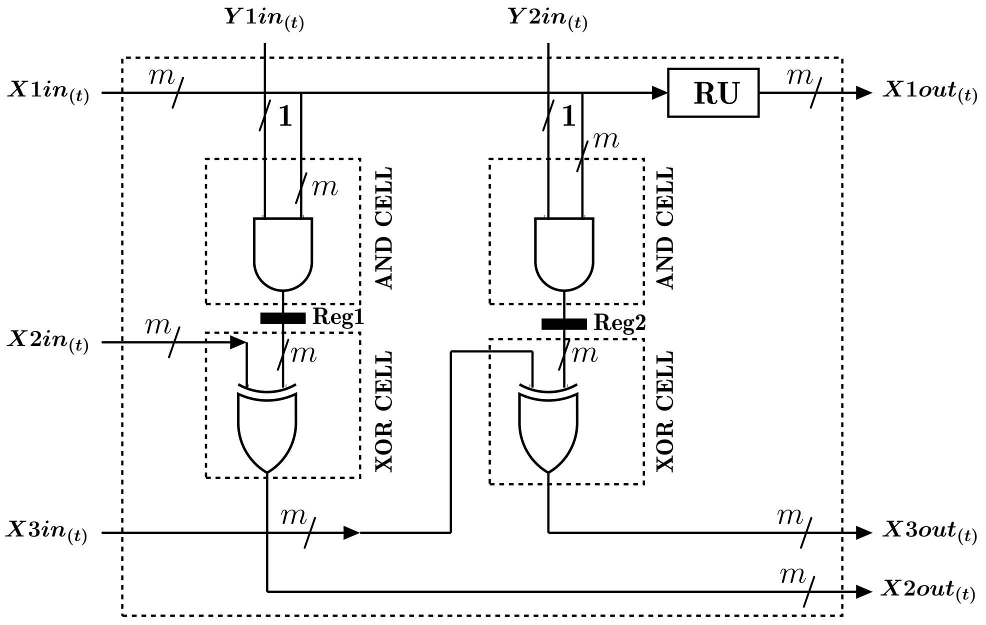

Figure 5.12 Detailed architecture of the regular PE.

The XOR cells are responsible for the addition of the product terms generated by the AND cells (in the previous clock cycle) to the respective inputs ($X2in_{(t)}$ and $X3in_{(t)}$) of the processing element. Two registers Reg1 and Reg2 are placed in the architecture to realize the horizontal cutsets that appeared in the processing elements PE[0] through PE[m/2-1] (See Fig. 5.8). It may be noted that since the functions of processing elements PE[0] and PE[$m/2-1$] are sub-functions for regular PE (See Fig. 5.7), the architectures for PE[0] and PE[$m/2-1$] can easily be obtained by modifying the architecture of the regular PE. The processing element PE[$x^{m/2}$] can be realized using an array of $m/2$ two-input XOR gates (See the last row of Table 5.1, where $m = 10$). The architecture of PE[out] requires an array of m two-input XOR gates to implement the m-bit addition. The proposed systolic multiplier generates one output per clock cycle with an initial latency of $(m/2 + 3)$ clock cycles.

5.3.2 Analytical Results

In this section, the area and time complexities of the proposed systolic multiplier are estimated and compared with the available multipliers in the literature. Table 5.5 presents the comparison of the analytical complexities of the proposed multiplier with the available multipliers. Furthermore, these analytical complexities are evaluated for a specific field order, $m = 409$, using FreePDK45 NanGate cell library statistics [46, 60], and presented in Table 5.6.

Table 5.5 Area and time complexities comparison for GF(2^m).

Design	AND	XOR	Register	Latency	Critical Path
[40]	m^2	m^2+m-1	$3m^2+2m-2$	$2m-1$	T_A+T_X
[41]	m^2	m^2+ml	$4m^2+2lm$	$m+l-1$	T_A+T_X
[42]	$(3m^2-m)/2$	m^2+m	$4m^2+m$	$m+1$	T_A+T_X
[43]	m^2	m^2+m	$3.5m^2+3m$	$m+2$	T_A+T_X
[27]	m^2	m^2-1	$2m(m-1)$	m	T_A+T_X
[44]	m^2	m^2+mt^a-m	$2m^2$	m	T_A+2T_X
[45]	m^2	m^2+mt^a-m	$2m^2$	$m/2+2$	$2T_X$
[46]	m^{2*}	$1.5m^2+0.5m$	$1.5m^2+2m-1$	$m+2$	$T_{NA}+T_X$
[62]	m^2	m^2-1	$1.5m^2+m$	$m/2+2$	T_A+T_X
Proposed	m^2	m^2-1	$2.5m^2+m$	$m/2+3$	T_X

$l = \lfloor (m-2)/(m-k) \rfloor + 1$.

* m^2 NAND gates along with $(1.5m^2 - 2.5m + 3)$ number of inverters.

^{a} t=1 for the class of polynomials considered in this work.

Table 5.5 presents the analytical comparison of area complexity, latency, and critical path delay of the proposed systolic multiplier with the available systolic multipliers considered for comparison [27, 40–46, 62]. All these multipliers including the proposed multiplier are applicable for either general trinomials or a class of trinomials. The area required for the proposed multiplier is computed in terms of the number of AND gates, XOR gates, and registers. We also present the area complexity for the other multipliers in a similar way to compare with that of the proposed multiplier. The time complexities of the proposed multiplier and other multipliers considered for comparison are computed by assuming that T_A, T_X, and T_{NA} denote the delays of 2-input AND gate, 2-input XOR gate, and 2-input NAND gate, respectively.

The area and time complexities of the proposed multiplier can be computed from the detailed gate level circuitry of all PEs of the structure shown in Fig. 5.10 where the structure contains $(m/2+2)$ PEs. The first processing element PE[0] requires one XOR gate, $2m$ AND gates, and two m-bit registers. Each one of the regular PEs from PE[1]

Table 5.6 Area and time complexities comparison for GF(2^{409}).

Design	**Critical Path** (ns)	**Area** ($\times 10^6$) (μm^2)	**Throughput** ($\times 10^9$)	**ADP** ($\times 10^{-3}$)	**% increase in Throughput**	**% reduction in ADP**
[40]	0.06	2.7	16.66	162	71.48	48.15
[41]	0.06	3.8	16.66	228	71.48	63.15
[42]	0.06	4.5	16.66	270	71.48	68.88
[43]	0.06	3.1	16.66	186	71.48	54.83
[27]	0.06	2	16.66	120	71.48	30
[44]	0.095	2	10.53	190	171.32	55.79
[45]	0.07	2	14.29	140	99 93	40
[46]	0.05	1.8	20	90	42.85	6.66
[62]	0.06	1.6	16.66	96	71.48	12.50
Proposed	0.035	2.4	28.57	84	–	–

to PE[$m/2 - 2$] requires $2m + 1$ XOR gates, $2m$ AND gates, and two m-bit registers. The PE[$m/2 - 1$] needs $2m$ XOR gates, $2m$ AND gates, two m-bit registers. The last two PEs, PE[$x^{m/2}$] and PE[Out], require $m/2$ XOR gates and m XOR gates, respectively. Hence, the proposed systolic structure requires m^2 AND gates, $(m^2 - 1)$ XOR gates and $(2.5m^2 + m)$ number of registers. Latency and Critical path delay (See Fig. 5.12) of the proposed multiplier structure are $(m/2 + 3)$ and T_X, respectively. It may be noted that all the multipliers in Table 5.5 have nearly the same AND and XOR gate complexities while the proposed multiplier requires more registers when compared with the multipliers [27,44–46,62]. However, when compared with all the other multipliers in Table 5.5, the proposed multiplier requires the lowest critical path delay T_X, thereby, able to give high throughput rates (Number of GF(2^m) multiplications/sec).

The analytical comparisons presented in Table 5.5 can be better understood by evaluating them for a specific value of the field order along with a specific technology-based area and time complexity estimations of gates. The field order, m, can be selected as 409, which is one of the field sizes recommended by NIST for Elliptic curve cryptographic applications [61]. For the estimation of area and time complexities of the gates, NanGate 45nm technology-based open cell library statistics [46,60] is adopted as follows: The area complexities in terms of the NAND gate equivalents (GE) for a NOT gate, a 2-input AND gate, a 2-input XOR gate, and a D flip-flop with set/reset capabilities are taken as 0.5, 1.4, 2, and 5.7, respectively. The delays of a 2-input NAND gate, a 2-input AND gate, and a 2-input XOR gate are 0.015, 0.025, and 0.035 ns, respectively. It is observed that the area required for a 2-input NAND gate based on Synopsys design compiler synthesis using 45nm NanGate open cell libraries is $0.8\mu m^2$.

Table 5.6 presents the estimated critical path delay and area complexity for the proposed multiplier along with same multipliers considered for analytical comparison. Area complexities are estimated in the units of μm^2 and it may be noted that the proposed multiplier requires more number of area when compared with the multipliers [27,44–46,62]. However, the proposed multiplier requires the lowest critical path delay which indicates that it achieves high throughput rates when compared with other multipliers. Though the proposed multiplier is not area-efficient, it achieves reduction in critical path delay and improvement in throughput rates. It is observed that the proposed multiplier achieves around 42% more throughput rates when compared with the best multiplier [46] (See % increase in Throughput column). The overall efficiency of the proposed multiplier can be compared with other multipliers in terms of area-delay-product (ADP). It is observed that the proposed multiplier achieves around 6% less area-delay-product when compared with the best multiplier [46] (See % reduction in ADP column). Hence, it is clear from the estimated values presented in Table 5.6 that the proposed multiplier has high throughput rates and also achieves a marginal reduction in area-delay-product when compared with the similar multipliers available in the literature.

Table 5.7 Comparison of ASIC implementation results for GF(2^{409}).

Design	**Critical Path (ns)**	**Throughput ($\times 10^9$)**	**Multiplier Area (μm^2)**	**ADP**	**% increase in Throughput**	**% reduction in ADP**
[46]	0.19	5.26	2783963	528953	35.74	5.7
[62]	0.21	4.76	2533407	532015	50	6.3
Proposed	0.14	7.14	3559604	498345	–	–

5.3.3 Implementation Results

It is observed from Table 5.6 that the multipliers [46, 62] have the nearest complexities to that of the proposed multiplier with respect to throughput and area-delay-product compared to the other multipliers considered for comparison. Hence, these two multipliers [46, 62] and the proposed multiplier are modeled using VHDL for GF(2^{409}). The RTL models are simulated using Vivado Simulator to verify the functionality. Also, these RTL models are synthesized using Synopsys Design Compiler tool employing NanGate 45nm open cell library [60] to obtain the area and time complexities. The area and time complexities obtained for all the three multipliers are tabulated in Table 5.7. It may be concluded from the synthesized results obtained that the proposed multiplier achieves more throughput and less area-delay-product than that of the other previous multipliers.

5.4 Low-Latency Area-Efficient Bit-Parallel Systolic Multiplier

In this section, the design and performance analysis of the proposed bit-parallel systolic multiplier for a narrow of trinomials for which $k \leq m - 2\lceil m/3 \rceil$ is presented. First, we present the mathematical formulations for the proposed GF(2^m) multiplication and its realization using the bit-parallel systolic architecture. Following this, analytical comparisons of area and time complexities and implementation results are presented.

5.4.1 Design

This section presents the formulations and the multiplier architecture for the proposed GF(2^m) multiplication. First, we develop the formulations for the proposed GF(2^m) multiplication method. Following this, the proposed multiplier architecture realized using these formulations is presented.

Mathematical Formulation

Consider Eq. 5.1, where $C(x)$ is the product of the two field elements $A(x)$ and $B(x)$, given by

$$\begin{aligned} C(x) &= (A(x) \times B(x)) \bmod T(x) \\ &= A(x)\sum_{i=0}^{m-1} b_i x^i \bmod T(x) \\ &= \sum_{i=0}^{m-1} b_i A(x) x^i \bmod T(x) \end{aligned}$$

$$= \big(b_{m-1}Ax^{m-1} \; + \; b_{m-2}Ax^{m-2} \; + \; .... \; + \; b_1 Ax \; + \; b_0 A\big) \bmod T(x)$$

$$= \big(b_{m-1}(Ax^{m-1}) + b_{m-2}(Ax^{m-2}) + ... + b_1(Ax) + b_0(A)\big) \bmod T(x) \quad (5.14)$$

This Eq. 5.14 is the same as Eq. 5.2, however, it is renumbered here for convenience. Equation 5.14 can be explicitly rewritten as a summation of three sub-expressions as shown in Eq. 5.15 where each sub-expression consists $m/3$ terms. The lower sub-expression contains the terms which have the powers of x from 0 to $(m/3 - 1)$. The middle sub-expression contains the terms which include the powers of x from $m/3$ to $(2m/3 - 1)$. The upper sub-expression contains the remaining terms that have the powers from $2m/3$ to $(m - 1)$.

$$\begin{aligned} C(x) = \Big(\Big(& b_{m-1}(Ax^{m-1}) + b_{m-2}(Ax^{m-2}) + ... + b_{2m/3+1}(Ax^{2m/3+1}) + b_{2m/3}(Ax^{2m/3})\Big) \\ &+ \Big(b_{2m/3-1}(Ax^{2m/3-1}) + b_{2m/3-2}(Ax^{2m/3-2}) + ... \\ &\quad ..... + b_{m/3+1}(Ax^{m/3+1}) + b_{m/3}(Ax^{m/3})\Big) \\ &+ \Big(b_{m/3-1}(Ax^{m/3-1}) + b_{m/3-2}(Ax^{m/3-2}) + ...+ \\ &\quad b_1(Ax) + b_0(A)\Big)\Big) \bmod T(x) \quad (5.15) \end{aligned}$$

Now, by bringing out $x^{m/3}$ as a common term from the middle sub-expression and $x^{2m/3}$ as a common term from the upper sub-expression, we can have

$$\begin{aligned} C(x) = \Big(\Big(& b_{m-1}(Ax^{m/3-1}) + b_{m-2}(Ax^{m/3-2}) + ... + b_{2m/3+1}(Ax) + b_{2m/3}(A)\Big)x^{2m/3} \\ &+ \Big(b_{2m/3-1}(Ax^{m/3-1}) + b_{2m/3-2}(Ax^{m/3-2}) + ... \\ &..... + b_{m/3+1}(Ax) + b_{m/3}(A)\Big)x^{m/3} \\ &+ \Big(b_{m/3-1}(Ax^{m/3-1}) + b_{m/3-2}(Ax^{m/3-2}) + ...+ \\ &b_1(Ax) + b_0(A)\Big)\Big) \bmod T(x) \end{aligned} \quad (5.16)$$

Now, we can define the following recursive notations, $A^{(0)} = A$, $A^{(1)} = A^{(0)}x$, $A^{(2)} = A^{(1)}x$, $A^{(3)} = A^{(2)}x$, ..., $A^{(m/3-1)} = A^{(m/3-2)}x$. These recursive relations can also be represented using the following recursive equation as,

$$A^{(j)} = A^{(j-1)}x \bmod T(x), \text{where } j = (m/3-1), ..., 2, 1 \quad (5.17)$$

where,

$$A^{(j-1)} = \sum_{i=0}^{m-1} a_i^{(j-1)}x^i, \forall\, a_i^{(j-1)} \in \mathrm{GF}(2) \quad (5.18)$$

Furthermore, from the fact that x is a root of the irreducible polynomial $T(x)$, hence, using Eq. 5.7 we have

$$x^m = t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + ... + t_2x^2 + t_1x + t_0$$

The expression for the element $A^{(j)}$ (Eq. 5.17) can be written as

$$A^{(j)} = a_{m-1}^{(j)}x^{m-1} + a_{m-2}^{(j)}x^{m-2} + ... + a_2^{(j)}x^2 + a_1^{(j)}x + a_0^{(j)} \quad (5.19)$$

The coefficients of the element $A^{(j)}$ can be obtained in terms of the coefficients of $A^{(j-1)}$ as

$$a_0^{(j)} = a_{m-1}^{(j-1)} \quad (5.20)$$

$$a_i^{(j)} = a_{i-1}^{(j-1)} + a_{m-1}^{(j-1)}t_i, \text{where } i = 1, 2, ..., m-1 \quad (5.21)$$

From these implications, the expression for $C(x)$ in Eq. 5.16 can be rewritten as

$$\begin{aligned} C(x) = \Big(\Big(& b_{m-1}(A^{(m/3-1)}) + b_{m-2}(A^{(m/3-2)}) + ... + b_{2m/3+1}(A^{(1)}) + b_{2m/3}(A^{(0)})\Big)x^{2m/3} \\ &+ \Big(b_{2m/3-1}(A^{(m/3-1)}) + b_{2m/3-2}(A^{(m/3-2)}) + ... \\ &..... + b_{m/3+1}(A^{(1)}) + b_{m/3}(A^{(0)})\Big)x^{m/3} \\ &+ \Big(b_{m/3-1}(A^{(m/3-1)}) + b_{m/3-2}(A^{(m/3-2)}) + ...+ \\ &b_1(A^{(1)}) + b_0(A^{(0)})\Big)\Big) \bmod T(x) \end{aligned} \quad (5.22)$$

It can be noted that the expression for $C(x)$ in Eq. 5.22 contains three similar sub-expressions, namely, sub-expression-1

$$\Big(b_{m-1}(A^{(m/3-1)}) + b_{m-2}(A^{(m/3-2)}) + ... + b_{2m/3+1}(A^{(1)}) + b_{2m/3}(A^{(0)})\Big) \bmod T(x) \quad (5.23)$$

and sub-expression-2 as

$$\begin{aligned} &\Big(b_{2m/3-1}(A^{(m/3-1)}) + b_{2m/3-2}(A^{(m/3-2)}) + ... \\ &..... + b_{m/3+1}(A^{(1)}) + b_{m/3}(A^{(0)})\Big) \bmod T(x) \end{aligned} \quad (5.24)$$

and sub-expression-3 as

$$\begin{aligned} &\Big(b_{m/3-1}(A^{(m/3-1)}) + b_{m/3-2}(A^{(m/3-2)}) + ...+ \\ &b_1(A^{(1)}) + b_0(A^{(0)})\Big)\Big) \bmod T(x) \end{aligned} \quad (5.25)$$

It can be observed that the above three sub-expressions (Eqs. 5.23, 5.24, and 5.25) are similar in computation and each one requires the elements $A^{(0)}$, $A^{(1)}$, ..., $A^{(m/3-2)}$, $A^{(m/3-1)}$ for its computation. Further, these $m/3$ elements can be computed recursively from the element A using Eq. 5.17 where each recursion requires modulo reduction using the irreducible polynomial $T(x)$. The computational complexity of the elements $A^{(j)}, j = 1, 2, ..., (m/3-1)$, can be reduced by selecting the low-weight irreducible polynomials such as trinomials. Moreover, it is observed that the complexity can be further reduced for a specific class of trinomials x^m+x^k+1 for which $k \leq m-2\lceil m/3\rceil$. This narrow class of trinomials that results in low-hardware complexities are highly desirable for constrained and cost-effective applications such as IoT applications. The complexity reduction using these trinomilas can be achieved as the multiplication of any field element, A, with x^i, where $i = 1, 2, ..., 2m/3$, can be realized using one level of i number of XOR gates. To make it more clear, we have considered the field GF(2^{12}) over the irreducible trinomial $T(x) = x^{12} + x^3 + 1$ where this trinomial obeys the condition that is mentioned for the specified narrow class of trinomials, i.e. $k \leq m - 2\lceil m/3 \rceil$. This field is considered to show that $Ax^i \bmod T(x)$, for $i = 1, 2, ..., 8$, requires one level of i XOR gates, and same is described in Table 5.8. The co-ordinate representation of the element $A \in$ GF(2^{12}) can be given as $(a_{11}, a_{10}, a_9, a_8, a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$. It can be observed from Table 5.8

Table 5.8 Coordinate representation of $Ax^i \bmod T(x)$.

i	$Ax^i \bmod T(x)$
1	$[a_{10}, a_9, a_8, a_7, a_6, a_5, a_4, a_3, a_2 \oplus a_{11}, a_1, a_0, a_{11}]$
2	$[a_9, a_8, a_7, a_6, a_5, a_4, a_3, a_2 \oplus a_{11}, a_1 \oplus a_{10}, a_0, a_{11}, a_{10}]$
3	$[a_8, a_7, a_6, a_5, a_4, a_3, a_2 \oplus a_{11}, a_1 \oplus a_{10}, a_0 \oplus a_9, a_{11}, a_{10}, a_9]$
4	$[a_7, a_6, a_5, a_4, a_3, a_2 \oplus a_{11}, a_1 \oplus a_{10}, a_0 \oplus a_9, a_{11} \oplus a_8, a_{10}, a_9, a_8]$
5	$[a_6, a_5, a_4, a_3, a_2 \oplus a_{11}, a_1 \oplus a_{10}, a_0 \oplus a_9, a_{11} \oplus a_8, a_{10} \oplus a_7, a_9, a_8, a_7]$
6	$[a_5, a_4, a_3, a_2 \oplus a_{11}, a_1 \oplus a_{10}, a_0 \oplus a_9, a_{11} \oplus a_8, a_{10} \oplus a_7, a_9 \oplus a_6, a_8, a_7, a_6]$
7	$[a_4, a_3, a_2 \oplus a_{11}, a_1 \oplus a_{10}, a_0 \oplus a_9, a_{11} \oplus a_8, a_{10} \oplus a_7, a_9 \oplus a_6, a_8 \oplus a_5, a_7, a_6, a_5]$
8	$[a_3, a_2 \oplus a_{11}, a_1 \oplus a_{10}, a_0 \oplus a_9, a_{11} \oplus a_8, a_{10} \oplus a_7, a_9 \oplus a_6, a_8 \oplus a_5, a_7 \oplus a_4, a_6, a_5, a_4]$

that realization of an i^{th} row requires i number of XOR gates. It can also be noted from the table that the realization of $Ax^i \bmod T(x)$ for $i = 8$ $(2m/3)$ requires the maximum number of one level of XOR gates i.e., 8 gates, while for all other considered values of i, it requires less than this maximum value $(2m/3)$.

Proposed Systolic Multiplier Architecture

This section presents the proposed systolic architecture based on the formulations developed for GF(2^m) multiplication. This multiplier architecture is suitable for multiplication in GF(2^m) fields that are generated using trinomials $x^m + x^k + 1$ for which $k \leq m - 2\lceil m/3 \rceil$. A signal flow graph (SFG) is developed based on the formulations (Eq. 5.22), and then a set of suitable cutset pipelining techniques are applied on it to realize the gate level structure for the proposed systolic multiplier.

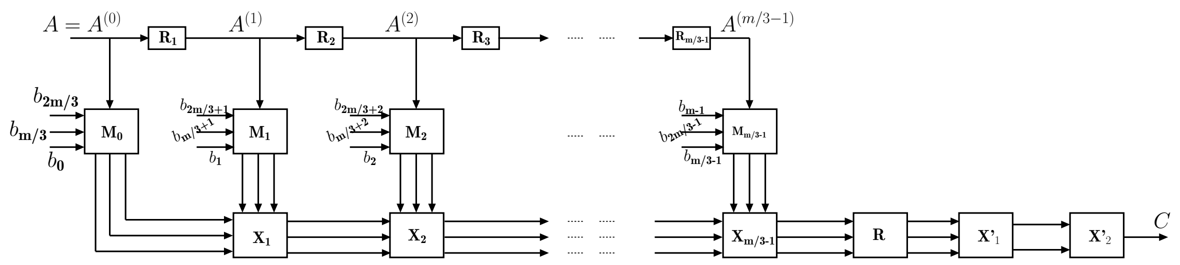

Figure 5.13 Signal flow graph (SFG) representation of the proposed GF(2^m) multiplication.

The expression for the multiplication of elements A and B, $C(x)$, given by Eq. 5.22 can be represented using the signal flow graph (SFG) shown in Fig. 5.13. The structure of this SFG is developed based on the observations that Eq. 5.22 comprises the three similar sub-expressions (See Eqs. 5.23, 5.24, and 5.25) whose computation can be performed in parallel and the terms $A^{(i)}$, which are used in the computation of the three sub-expressions, can be obtained from the recursive computations (See Eq. 5.17). The SFG contains five types of nodes namely reduction nodes (R$_i$), multiplication nodes (M$_i$), addition nodes (X$_i$), final reduction node (R), and final addition nodes (X'$_i$). Specifically, the SFG contains $(m/3 - 1)$ reduction nodes (R$_i$, for i=1, 2, 3, ..., $m/3 - 1$), $m/3$ multiplication nodes (M$_i$, for i=0, 1, 2, ..., $m/3 - 1$), $(m/3 - 1)$ addition nodes (X$_i$, for i=1, 2, 3, ..., $m/3 - 1$), one final reduction node (R), and two final addition nodes (X'$_i$, for i=1, 2). The input operands, A and B, are applied in parallel to the SFG and the output, C, is also available in parallel.

The functional description of the nodes of the SFG is depicted in Fig. 5.14. The reduction node R$_i$ presented in Fig. 5.14(a) performs reduction by degree one on its input, $A^{(i-1)}$, after multiplying it with x. The equation that describes the function of this node is also presented in this figure where $A^{(i)}$ is considered as the reduced form of $A^{(i-1)}$. The multiplication node M$_i$ presented in Fig. 5.14(b) performs three simultaneous multiplications, where, in the each multiplication the single bit input $(b_i/b_{m/3+i}/\mathrm{b}_{2m/3+i})$ is multiplied with the m-bit input $A^{(i)}$. The formulations that represent the function of this node are also presented in the figure. The addition node X$_i$ shown in Fig. 5.14(c) performs three simultaneous additions where the m-bit inputs C_i, D_i, and E_i are added to the inputs $b_{2m/3+i}A^{(i)}$, $b_{m/3+i}A^{(i)}$, and $b_iA^{(i)}$, respectively. The final reduction node R shown in Fig. 5.14(d) performs reduction by degree $2m/3$ on the input C_i and reduction

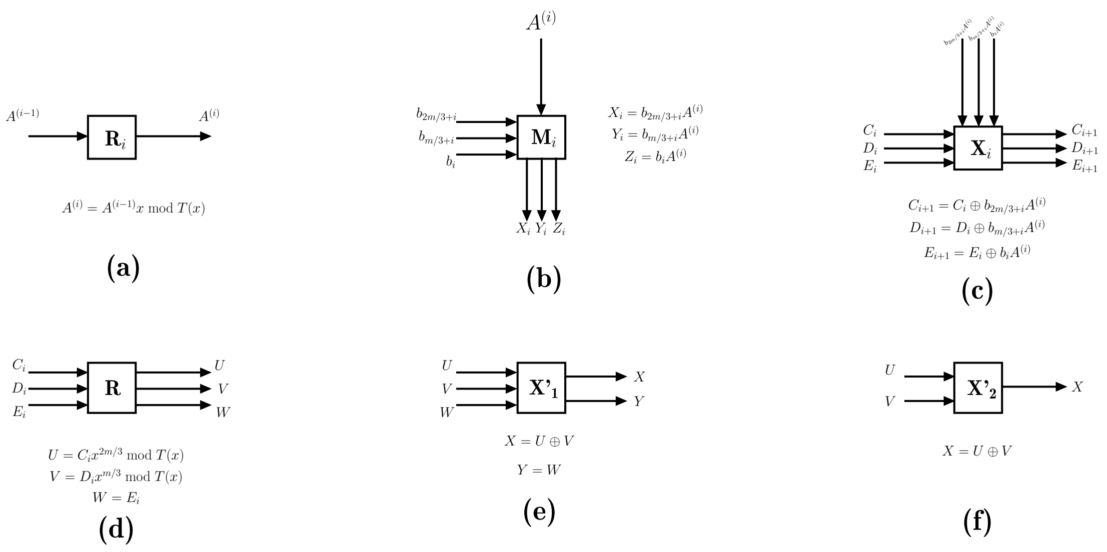

Figure 5.14 Functional description of a) ith reduction node R$_i$ b) ith multiplication node M$_i$ c) ith addition node X$_i$ d) final reduction node R e) final addition node X'$_1$ f) final addition node X'$_2$.

by degree $m/3$ on the input D_i. The other input E_i is unchanged and is simply routed to the output W. The formulations representing the functionality of this node are also presented in the figure. The final addition node X'$_1$ presented in Fig. 5.14(e) performs the addition of the upper two m-bit inputs (U and V). The other input (W) remains unchanged and is made available at the output (Y). The equations that describe the functionality of this node are also presented. The final addition node X'$_2$ presented in Fig. 5.14(f) performs the addition of the two m-bit inputs (U and V). The equation that describes the functionality of this node is also presented.

It may be noted that Eq. 5.22 representing the SFG (Fig. 5.13) consists of the three sub-expressions as specified using Eqs. 5.23, 5.24, and 5.25. The computations involved in the evaluation of these three sub-expressions are similar and requires the reduced forms of A from $A^{(0)}$ to $A^{(m/3-1)}$. Moreover, these three expressions can be evaluated simultaneously as shown in the SFG. The reduced forms of input A are obtained using reduction nodes R$_i$ and used by multiplication nodes M$_i$ to generate the product terms of the form $b_iA^{(i)}/b_{m/3+i}A^{(i)}/b_{2m/3+i}A^{(i)}$. These product terms are accumulated using X$_i$ nodes to obtain the evaluation of the sub-expressions (Eqs. 5.23, 5.24, and 5.25). Once these three expressions are computed, the evaluation of Eq. 5.22 to obtain the required product C can be performed by modulo multiplying Eq. 5.23 with $x^{2m/3}$ and Eq. 5.24 with $x^{m/3}$ using the R node followed by the addition of both of these equations to Eq. 5.25 using the X'$_1$ and X'$_2$ nodes.

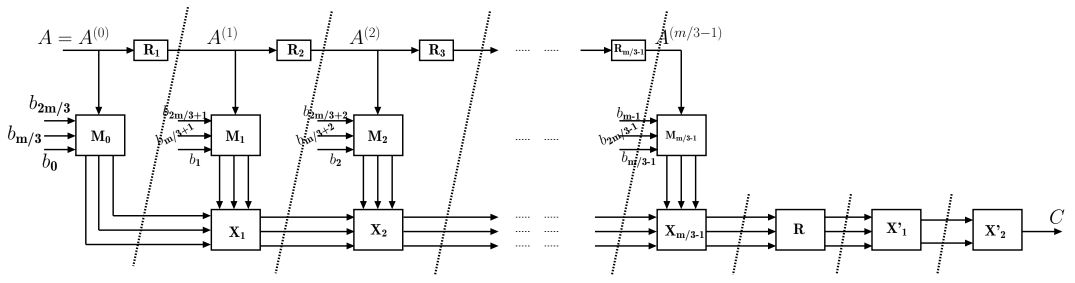

Figure 5.15 Pipelined SFG using vertical cutsets.

It can be observed from the SFG (Fig. 5.13) that input A is processed through all the nodes at a stretch to obtain the output C. This kind of processing at a stretch incurs a long critical path delay when this SFG is realized in hardware. Hence, it is required to apply suitable cutset pipelining to this SFG to reduce the critical path delay. It is identified that the vertical cutsets as shown in Fig. 5.15 are best suitable for the efficient realization of the SFG. These cutsets are feed-forward cutsets and indicate the placing of registers in hardware realization at the points where the cutset lines intersect the SFG. The vertical cutsets are applied such that the sequential dependency between the R$_i$ node and the M$_i$ & X$_i$ nodes is eliminated i.e. during any clock cycle the M$_i$ & X$_i$ nodes need not wait until the output at the R$_i$ node available. Instead, the M$_i$ & X$_i$ nodes process the data generated by R$_i$ node during the previous clock cycle. The sections of the SFG that are separated by cutset lines can be considered as processing elements (PEs) and the SFG (Fig. 5.15) contains $(m/3 + 3)$ such sections. The formation of PEs obtained from the pipelined SFG is presented in Fig. 5.16, and there are $(m/3 + 3)$ PEs. The hardware realization of this SFG requires to insert registers (incurred by cutset lines) between PEs.

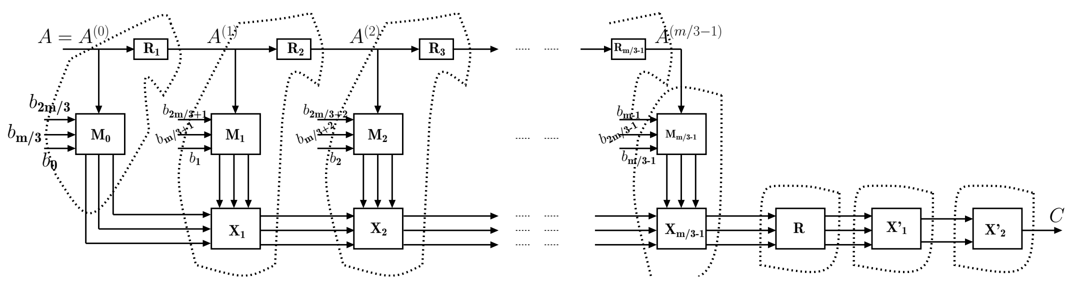

Figure 5.16 Formation of the processing elements (PEs).

The proposed systolic structure for obtaining the required finite field GF(2^m) product, $C(x)$ given by Eq. 5.22, is presented in Fig. 5.17. This structure is developed using the SFG shown in Fig. 5.16. This systolic structure consists of a total of $(m/3+3)$ PEs and these PEs can be categorized into six types based on the functionality. The PEs from PE[1] to PE[m/2-2] are of the same type and this type is denoted as regular PE (PE). The other PEs, PE[0], PE[m/3-1], PE[$\boldsymbol{x^{m/3}}$&$\boldsymbol{x^{2m/3}}$], PE[X'$_1$], and PE[X'$_2$] are of distinct PEs and are also different from the regular PEs.

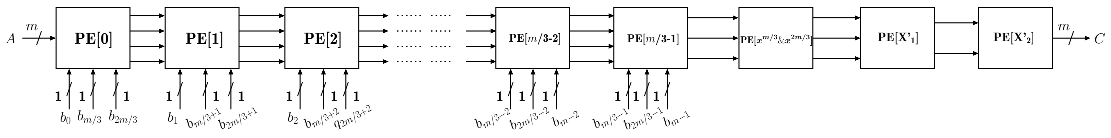

Figure 5.17 Proposed systolic structure for GF(2^m) multiplication.

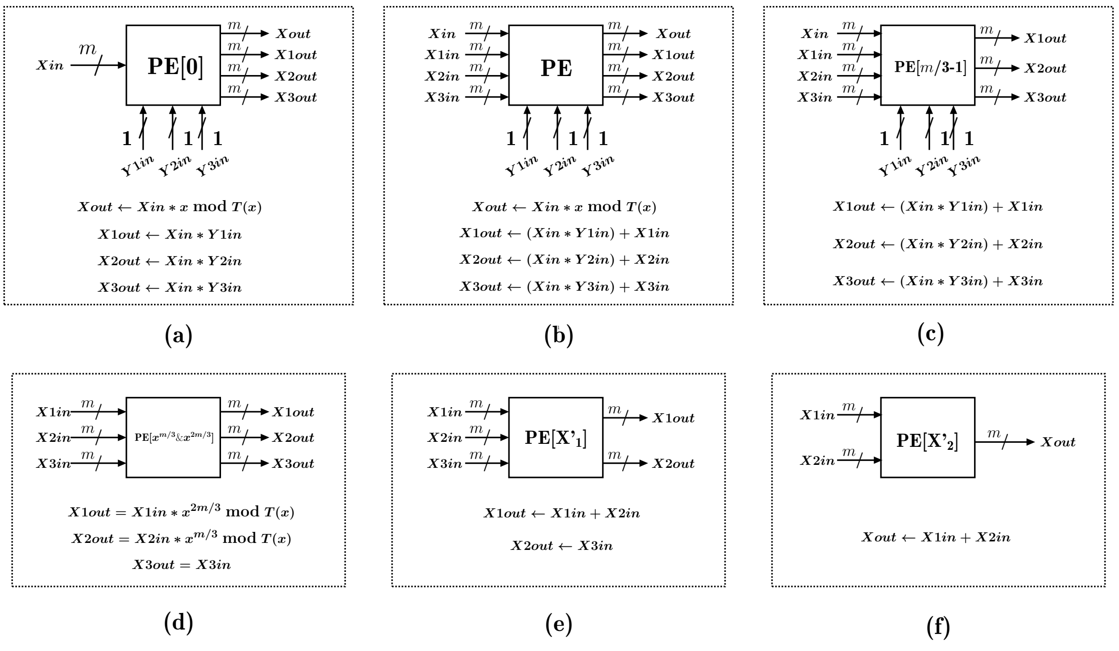

Figure 5.18 Functional description of a) PE[0] b) Regular PE c) PE[m/3-1] d) PE[$\boldsymbol{x^{m/3}}$&$\boldsymbol{x^{2m/3}}$] e) PE[X'$_1$] f) PE[X'$_2$].

The functional description of all the PEs is presented in Fig. 5.18. The functional description of the first PE of the systolic structure, PE[0], is presented in Fig. 5.18(a). This PE performs a reduction by degree one operation and three multiplication operations. The multiplication operations involve the multiplication of an m-bit operand with a 1-bit operand. The functional description of the regular PEs is presented in Fig. 5.18(b).

These PEs perform a reduction by degree one operation and three multiply-add operations. The multiply-add operations involve the multiplication of an m-bit operand with a 1-bit operand followed by the addition of an m-bit operand. Figure 5.18(c) presents the functional description of the PE[m/3-1] where this PE performs three multiply-add operations. Figure 5.18(d) presents the functional description of the PE[$\boldsymbol{x^{m/3}}$&$\boldsymbol{x^{2m/3}}$] where the functionalities of this PE include a reduction by degree $2m/3$ operation and a reduction by degree $m/3$ operation. The functional description of the PE[X'$_1$] is presented in Fig. 5.18(e). This PE performs the addition of the two upper inputs while simply routing the other input to the output. The functional description of the PE[X'$_2$] is presented in Fig. 5.18(f) where this PE performs the addition of its two inputs. Formulations that represent the functionalities are also included for all the PEs presented in Fig. 5.18.

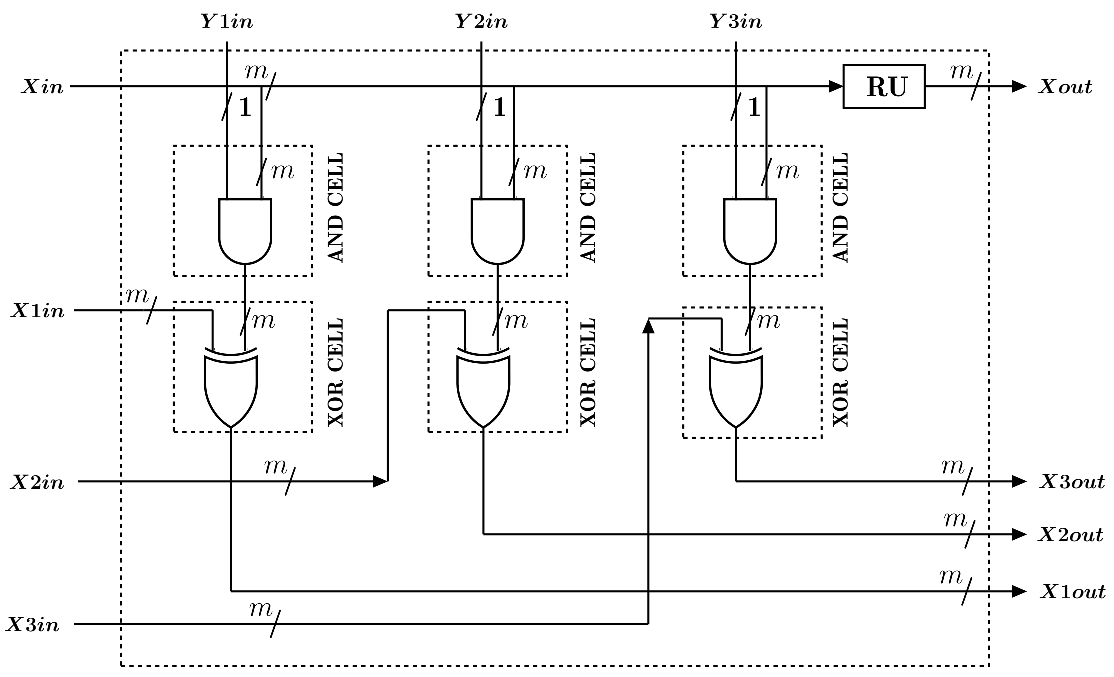

Figure 5.19 Detailed gate level architecture of the regular PE.

The detailed gate-level architecture of the regular PE is presented in Fig. 5.19. The inputs Xin, $X1in$, $X2in$, and $X3in$ are of m-bits and the inputs $Y1in$, $Y2in$, and $Y3in$, are of 1-bit. Also, there are four outputs and all are of m-bit width. The architecture contains a reduction unit (RU), three AND cells, and three XOR cells. The reduction unit RU performs the reduction by degree one operation on the input Xin. It realizes the reduction node R$_i$ of a regular PE. It comprises a single XOR gate to realize the reduction operation. The exact placement of this XOR gate in the RU depends on the degree of the middle term of the trinomial selected. For instance, the first row of Table 5.8 represents the reduction by degree one of the element A using the trinomial $x^{12} + x^3 + 1$. For this case, it requires an XOR gate at the fourth place from the least-significant-bit position to realize the reduction operation. The three AND cells perform the multiplication of the three 1-bit inputs $Y1in$, $Y2in$, and $Y3in$ with the Xin. These three AND cells jointly denote the realization of the multiplication node M_i of a regular PE. Each AND cell contains an array of m AND gates where an i^{th} gate takes the i^{th} bit of Xin as one input and another input is $Y1in$ (for the left cell)/ $Y2in$ (for the middle cell)/$Y3in$ (for the right cell). The three XOR cells perform the addition of the three m-bit inputs $X1in$, $X2in$, and $X3in$ with the respective output of the three AND cells as shown in the figure. These three XOR cells jointly denote the realization of the addition node X_i of a regular PE. The left XOR cell contains an array of m XOR gates where an i^{th} gate takes the i^{th} bit of $X1in$ as one input and another input is the i^{th} bit of the output of the left AND cell. Similarly, the same is the case with middle and right XOR cells.

The gate level architectures for the PE[0] and PE[$m/3$-1] can be easily obtained from the architecture of the regular PE as the functionalities of these PEs are sub-functions of the regular PE. The processing element PE[$\boldsymbol{x^{m/3}}$&$\boldsymbol{x^{2m/3}}$] can be realized using m XOR gates. The reduction by $2m/3$ operation of this PE requires $2m/3$ XOR gates and the reduction by $m/3$ operation requires $m/3$ XOR gates. The placement of these gates depends on the trinomial selected for the design of GF(2^m) multiplier (For example, see the eighth and fourth rows of Table 5.8 for the trinomial $x^{12} + x^3 + 1$). The gate level architectures for the PE[X'$_1$] and PE[X'$_2$] require m XOR gates each. The proposed systolic multiplier has a latency of $(m/3 + 3)$ clock cycles and generates a new output for each clock cycle.

5.4.2 Analytical Results

This section presents the area and time comparisons of the proposed systolic multiplier with the related systolic multipliers available in the literature. Table 5.9 presents the analytical comparisons of the area and time complexities and Table 5.10 presents the comparison of the estimated area and time complexities computed for $m = 409$ using the FreePDK 45nm NanGate open cell library statistics.

Table 5.9 presents the analytical comparison of the proposed multiplier with the available multipliers [27, 39–46, 62] in terms of gate complexities, critical path delay, and latency. All of these multipliers listed in Table 5.9 are applicable for either trinomials or a specific class of trinomials. The area complexities for the proposed multiplier are computed in terms of the number of 2-input XOR gates, 2-input AND gates, and registers. The area complexities of the multipliers considered for comparison are also presented using similar gates. The time complexities for the proposed multiplier and the available multipliers are presented in terms of 2-input AND gate delay (T_A) and 2-input XOR gate delay (T_X).

Table 5.9 Comparison of area and time complexities for GF(2^m).

Design	**AND**	**XOR**	**Register**	**Latency**	**Critical Path**
[39]	$(m+1)^2$	$(m+1)^2$	$4(m+1)^2$	$m+1$	T_A+T_X
[40]	m^2	m^2+m-1	$3m^2+2m-2$	$2m-1$	T_A+T_X
[41]	m^2	m^2+ml	$4m^2+2lm$	$m+l-1$	T_A+T_X
[42]	$(3m^2-m)/2$	m^2+m	$4m^2+m$	$m+1$	T_A+T_X
[43]	m^2	m^2+m	$3.5m^2+3m$	$m+2$	T_A+T_X
[27]	m^2	m^2-1	$2m(m-1)$	m	T_A+T_X
[44]	m^2	m^2+mt^a-m	$2m^2$	m	T_A+2T_X
[45]	m^2	m^2+mt^a-m	$2m^2$	$m/2+2$	$2T_X$
[46]	m^{2*}	$1.5m^2+0.5m$	$1.5m^2+2m-1$	$m+2$	$T_{NA}+T_X$
[62]	m^2	m^2-1	$1.5m^2+m$	$m/2+2$	T_A+T_X
Proposed	m^2	$10m^2/9-m/3$	$4m^2/3+5m$	$m/3+3$	T_A+T_X

$l = \lfloor (m-2)/(m-k) \rfloor + 1$.

* m^2 NAND gates along with $(1.5m^2 - 2.5m + 3)$ number of inverters.

^{a} t=1 for the class of polynomials considered in this work.

The area and time complexities for the proposed systolic multiplier can be obtained from the gate level architectures of the PEs of the systolic structure presented in Fig. 5.17. The area complexities for the regular PEs (PE[1] to PE[m/2-2]) can be computed from the detailed gate-level architecture presented in Fig. 5.19. This architecture comprises a reduction unit RU and three sets of AND and XOR cells. The reduction unit requires one XOR gate, a AND cell requires m AND gates, and an XOR cell requires m XOR gates.

Hence, the regular PE requires a total of $3m$ AND gates and $3m+1$ XOR gates. The architecture of the processing element PE[0] is similar to regular PE except that it does not require XOR cells. Hence, this PE requires $3m$ AND gates and a single XOR gate. The architecture of the processing element PE[m/3-1] is also similar to regular PE except that it does not require the reduction unit RU. Hence, this PE requires $3m$ AND gates and $3m$ XOR gates. The processing element PE[$\boldsymbol{x^{m/3}}$&$\boldsymbol{x^{2m/3}}$] requires m XOR gates which include $2m/3$ XOR gates for reduction by degree $2m/3$ operation and $m/3$ XOR gates for reduction by degree $m/3$ operation. The final addition nodes PE[X'$_1$] and PE[X'$_2$] require m XOR gates each. Furthermore, to realize the cutset lines (See Fig. 5.15) it is required to place the registers between any two adjacent PEs i.e. each horizontal line in the proposed systolic structure (See Fig. 5.17) requires m registers. Hence, the proposed multiplier requires a total of m^2 AND gates, $10m^2 - m/3$ XOR gates, and $4m^2/3 + 5m$ registers. Critical path delay and latency of the proposed multiplier are $T_A + T_X$ and $(m/3+3)$ clock cycles (See Fig. 5.19), respectively. These area and time complexities are presented in Table 5.9 along with the complexities of the available multipliers. It can be observed from this table that the proposed multiplier requires the lowest latency compared to all other multipliers. Also, the critical path delay of the proposed multiplier is nearly the same as the available multipliers.

The comparisons presented in Table 5.9 can be better assessed using a specific field order m and a specific implementation technology. The field order is selected as $m = 409$ since it is the field order recommended by NIST for implementation of the Elliptic curve digital signature algorithm (ECDSA) [61, 63]. We have considered Nangate 45nm open cell library [46, 60] statistics to obtain the estimated standard gate complexities. Based on this technology, the estimations for area and time complexities are adopted as follows. The area complexities for all the required gates are represented in terms of the 2-input NAND gate equivalents (GE), where a NOT gate, a 2-input XOR gate, a 2-input AND gate, and a D flip-flop with set/reset capabilities are equivalent to 0.5, 2, 1.4 and 5.7 GEs, respectively. It is observed that the area required for a 2-input NAND gate when synthesized using Synopsys design compiler employing 45 nm NanGate open cell libraries is 0.8 μm^2. The delays of a 2-input NAND gate, a 2-input XOR gate, and a 2-input AND gate are 0.015, 0.035, and 0.025 ns, respectively.

Table 5.10 Comparison of area and time complexities for GF(2^{409}).

Design	**Critical Path** (ns)	**Latency** ($clock\ cycles$)	**Latency** (ns)	**Area** ($\times 10^6$) (μm^2)	**% reduction in Latency** (ns)	**% reduction in Area**
[39]	0.06	410	24.60	3.5	66	57.14
[40]	0.06	817	49.02	2.7	83	44.44
[41]	0.06	410	24.60	3.7	67	59.45
[42]	0.06	410	24.60	4.5	66	66.66
[43]	0.06	411	24.66	3.1	67	51.61
[27]	0.06	409	24.54	2	66	25
[44]	0.09	409	36.81	2	78	25
[45]	0.07	207	14.49	2	42	25
[46]	0.05	411	20.55	1.8	59	16.67
[62]	0.06	207	12.42	1.6	32	6.67
Proposed	0.06	139	8.34	1.5	–	–

Table 5.10 presents the estimated area and time complexities computed using $m = 409$ and the NanGate 45nm open cell library statistics for the proposed multiplier and the multipliers considered for comparison. The table presents the comparison of critical path delay (ns), latency ($clock\ cycles$), latency (ns), and area complexity (NAND gate equivalents, GE). It is observed that the proposed multiplier requires the same critical path delay ($0.06ns$) when compared to the multipliers [39], [40], [41], [42], [43], [27], [62] and more delay when compared to the multiplier [45]. However, the proposed multiplier requires the lowest latency both in terms of clock cycles and delay (ns) compared to all other multipliers (See third and fourth columns). The proposed multiplier achieves 66%, 83%, 67%, 66%, 67%, 66%, 78%, 42%,59%, and 32% reduction in latency (ns) compared to the multipliers [39], [40], [41], [42], [43], [27], [44], [45], [46], [62], respectively. It is also observed from the table that the proposed multiplier requires the lowest area (See the fifth column) when compared to all other multipliers considered for comparison. The proposed multiplier achieves 57%, 44%, 59%, 66%, 51%, 25%, 25%, 25%, 16% and 6% reduction in area compared to the multipliers [39], [40], [41], [42], [43], [27], [44], [45], [46], [62], respectively. Hence, it is clear from this table that the proposed multiplier achieves low latency and low area complexities compared to the similar multipliers available in the literature.

Table 5.11 Comparison of ASIC implementation results for GF(2^{409}).

Design	Critical Path (ns)	Latency (ns)	Multiplier Area (μm^2)	% reduction in Latency	% reduction in Area
[46]	0.19	78.09	2783963	58.67	13.08
[62]	0.21	43.47	2533407	25.76	4.49
Proposed	0.21	32.27	2419701	–	–

5.4.3 Implementation Results

It may be observed from Table 5.10 that the two multipliers [46,62] have the nearest complexities to the proposed multiplier compared to the other multipliers. These two nearest multipliers and the proposed multiplier are modeled for the field GF(2^{409}) using VHDL and simulated using the Vivado simulator to verify the functionality. Also, these models are synthesized using Synopsys design compiler tool employing the NanGate 45nm open cell libraries [60] to obtain the critical path delay (ns) and area (μm^2). Table 5.11 presents the comparison of ASIC implementation results obtained using the Synopsys design compiler tool. It is observed that the proposed multiplier requires the lowest area compared to the other two multipliers. The proposed multiplier requires 13% and 4% less area when compared to the multipliers [46, 62], respectively. It is also observed that the proposed multiplier requires the lowest latency compared to the other two multipliers. The proposed multiplier achieves a 58% and 25% reduction in latency when compared to the multipliers [46, 62], respectively. Hence, the implementation results also confirm that the proposed multiplier achieves low latency and low area.

5.5 Conclusions

In this chapter, three high-performance multipliers employing systolic architectures using two specific classes of trinomials targeting IoT edge devices are presented. The design of the proposed multipliers involves developing the formulations followed by signal graph representation and applying efficient cutset pipelining techniques. It includes the design of a bit-parallel systolic multiplier using the class of trinomials $x^m + x^k + 1$ for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). Analysis of this multiplier is performed using analytical and implementation results which show that the proposed multiplier achieves low-latency and low-area compared to the available multipliers. This low-latency and low-area multiplier is further modified by employing a few more cutsets to reduce the critical path delay. The analysis of this modified multiplier shows that it achieves improved throughput compared to the available related multipliers. Design of the proposed multipliers also includes another systolic multiplier using the class of trinomials $x^m + x^k + 1$ for which $k \leq m - 2\lceil m/3 \rceil$. Analysis of this multiplier is performed using analytical and implementation results which show that the proposed multiplier achieves low-latency and low-area compared to available related multipliers including our first proposed low-latency multiplier (for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even)). For all the proposed multipliers, analytical comparisons are performed by computing the expressions for $m = 409$ using NanGate 45nm FreePDK open cell library statistics and implementaion comparisons are performed for $m = 409$ using Synopsis Design Compiler employing NanGate 45nm FreePDK open cell library files. The proposed systolic multipliers generate a new product for every clock cycle and are suitable for high-performance IoT devices such as edge devices.

The bit-serial sequential multipliers presented in the previous chapter (Chapter 4) are suitable for low-cost IoT devices where performance is not an important criterion (Domestic IoT), and the bit-parallel systolic multipliers presented in this chapter are suitable for high-performance IoT devices where the cost is not a primary objective (Industrial IoT). However, there is also a need for scalable (digit-serial) multipliers, which are in between the bit-serial and the bit-parallel multipliers with respect to performance as well as cost, required for the IoT devices that are employed in a wide variety of applications such as agriculture and healthcare. Hence, the design of scalable multipliers such as digit-serial multipliers is desirable to address the wide variety of performance requirements arising from various application domains. Consequently, the next chapter presents the design of efficient digit-serial multipliers that are suitable for a wide range of IoT applications.

Chapter 6

High-Throughput and Low-hardware Digit-Serial Sequential Multipliers for a Specific Class of Trinomials

Digit-serial multipliers facilitate scaling of area/delay or trade-off between area and delay. Depending on the application, the digit-size of these multipliers can be selected anywhere between a single-bit to m-bits. In this chapter, we present the design of two digit-serial sequential multipliers using a specific class of trinomials (This class of trinomials is the same as the class that is used for the design of the area-efficient low-latency systolic multiplier presented in the previous chapter, Section 5.2). First, we present a fully digit-serial sequential multiplier using the class of trinomials $x^m + x^k + 1$ for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). The algorithm of this multiplier is based on a redundant basis multiplication algorithm available in the literature. Next, we present another digit-serial sequential multiplier using the same class of trinomials $x^m + x^k + 1$ for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). The area and time complexities of these multipliers are obtained analytically and compared with the related available multipliers. Also, these analytical complexities are computed for $m = 409$ using the gate estimations from the FreePDK NanGate 45nm technology standard cell libraries. Further, the proposed multipliers are modeled using VHDL (Very High Speed Integrated Circuit Hardware Description Language) and synthesized using Synopsys Design Compiler employing FreePDK NanGate 45nm technology libraries, and compared with the best available multipliers. The comparisons show that the proposed fully digit-serial multiplier is time-efficient and the other digit-serial multiplier is hardware-efficient.

6.1 Introduction

Bit-serial and bit-parallel multipliers are two extremes with respect to hardware requirement and speed performance. Though bit-serial multipliers are highly area-efficient, they can not be used for applications that require high/moderate data speeds. On the other hand, bit-parallel multipliers provide high throughput rates, however, they fail to meet the area requirements of resource-constrained applications. Neverthless, digit-serial multipliers allow the scaling of area or throughput rates, hence, facilitate the flexible implementation. Hence, the design of digit-serial multipliers is required targeting a wide range of IoT (Internet of Things) applications that require moderate performance such as smart agriculture and smart healthcare.

Many digit-level finite field multipliers [51–55] are proposed in the literature to achieve better area and time complexities. In this chapter, first, we present a modified digit-serial polynomial basis multiplication algorithm and its fully digit-serial architecture. The proposed algorithm is based on an algorithm presented for the redundant basis multiplier [64]. Multipliers based on redundant basis do not involve modulo reduction step, however, they take a large amount of excessive hardware due to the embedding of the finite field GF(2^m) in a larger cyclotomic field. Hence, redundant basis multipliers are not suitable for most of the fields including fields recommended by National Institute of Standards and Technology (NIST) for implementing the elliptic curve digital signature algorithm (ECDSA). Since polynomial basis multiplier is more efficient compared to other bases multipliers, we propose a fully digit-serial architecture for hardware realization of polynomial basis multiplier for trinomials based on the methodology presented in [64]. The proposed multiplier is applicable for the class of trinomials $x^m + x^k + 1$ where $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even) which includes two of the NIST recommended fields (GF(2^{233}) and GF(2^{409})), and is highly suitable for cost-effective IoT edge devices employed for high data rate applications. The allowable digit-size (w) for this multiplier is assumed to be $w \leq (m-1)/4$ (if m is odd) or $w \leq m/4$ (if m is even). Next, we present another polynomial basis digit-level multiplier whose structure comprises of a parallel multiplier followed by an accumulation unit. The parallel multiplier is based on the approach proposed for an available parallel multiplier for all trinomials [65], and this approach when applied for the class of trinomials, $x^m + x^k + 1$ where $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even), gives low hardware implementations [66]. This proposed low-hardware digit-serial multiplier is suitable for end devices used in IoT applications.

6.2 High-Throughput Fully Digit-Serial Sequential Multiplier

In this section, the design of the proposed digit-serial multiplier and its performance analysis using analytical and implementation comparisons with the related multipliers are presented.

6.2.1 Design

In this section, mathematical formulations for the proposed fully digit-serial multiplication scheme are developed. The formulations are based on recursive definitions of the input elements. Based on the formulations developed for polynomial basis multiplication, a digit-level finite field GF(2^m) multiplier architecture is proposed. One digit from each element starting from MSD (most significant digit) enters the architecture in each clock cycle. The architecture takes $(n+1)$ clock cycles to perform one multiplication operation.

Mathematical Formulations

Let GF(2^m) be a binary finite field defined over an irreducible polynomial $T(x)$, where $T(x)$ be a trinomial of the form $x^m + x^k + 1$ for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). Let A and B be two arbitrary m-bit field elements represented using polynomial basis.

Assume A is divided into n digits where each digit contains w bits. Further, this digit-size (w) allowed is assumed to be such that $w \leq (m-1)/4$ (if m is odd) or $w \leq m/4$ (if m is even). Then, we have $n = \lceil \frac{m}{w} \rceil$, and

$A = \underbrace{a_0 a_1 ...... a_{w-1}}_{A_0} \underbrace{a_w ...... a_{2w-1}}_{A_1} .... \underbrace{a_{(k-1)w} .... a_{m-1} 0 ..... 0}_{A_{n-1}}$

The MSD is appended with $(nw - m)$ zeros to make it a digit of w bits. The element A can be reconstructed recursively using its n digits as

$$A^{(l)} = A_{n-l} + (x^w A^{(l-1)}), \quad l = 1, 2, ...n \tag{6.1}$$

where, $A^{(0)} = 0$, $A_{n-l} = \sum_{i=0}^{w-1} a_{(n-l)w+i} x^i$, and $A^{(n)} = A$.

Let C be the product of A and B. Then, we have $C = AB \bmod T(x)$. Using the recursive definitions of elements A and B, then C can be expressed as

$$\begin{aligned} C &= A^{(l)} B^{(l)} \Big|_{l=n} \bmod T(x) \\ &= A^{(n)} B^{(n)} \bmod T(x) \\ &= \big(A_0 + (x^w A^{(n-1)})\big)\big(B_0 + (x^w B^{(n-1)})\big) \bmod T(x) \end{aligned} \tag{6.2}$$

It follows,

$$C \;=\; \big(A_0(B_0 \;+\; x^w B^{(n-1)}) \;+\; B_0(x^w A^{(n-1)}) \;+\; A^{(n-1)} B^{(n-1)} x^{2w}\big) \bmod T(x) \tag{6.3}$$

$$= \big(A_0(B_0 + x^w B^{(n-1)}) + B_0(x^w A^{(n-1)})\big) \bmod T(x) + A^{(n-1)} B^{(n-1)x^{2w}} \bmod T(x) \tag{6.4}$$

Equation 6.4 can be evaluated by defining two intermediate vectors $V_i^{(l)}$, and $C^{(l)}$ as

$$V_i^{(l)} = \big(a_{w(n-l)+i} B^{(l)} + b_{w(n-l)+i}(x^w A^{(l-1)})\big) x^i \bmod T(x) \tag{6.5}$$

and

$$C^{(l)} = \sum_{i=0}^{w-1} V_i^{(l)} + (C^{(l-1)} x^{2w} \bmod T(x)) \tag{6.6}$$

After developing the above formulations, the proposed polynomial basis multiplication operation is computed as given in algorithm 6.1.

Algorithm 6.1: Proposed fully digit-serial multiplication algorithm

Input: $A_{n-l}, B_{n-l}, l = 1, ..., n$, w.r.t. polynomial basis

Output: $C = C^{(n)} = AB \bmod T(x) = (c_{m-1}, c_{m-2}, \ldots, c_1, c_0)$ also w.r.t. polynomial basis

Initialisation: $n = \lceil \frac{m}{w} \rceil$, $C^{(0)} = 0$;

```
1: for $l = 1$ to $n$, compute in serial do
2:     for $i = 0$ to $w-1$, compute in parallel do
3:         if $i = 0$ then
4:             $V_0^{(l)} = \big(a_{w(n-l)}B^{(l)} + b_{w(n-l)}(x^w A^{(l-1)})\big) + \big(C^{(l-1)}x^{2w} \bmod T(x)\big)$
5:         else
6:             $V_i^{(l)} = \big(\big(a_{w(n-l)+i}B^{(l)} + b_{w(n-l)+i}(x^w A^{(l-1)})\big)x^i\big) \bmod T(x)$
7:         end if
8:     end for
9:     $C^{(l)} = \sum_{i=0}^{w-1} V_i^{(l)}$
10: end for
```

Proposed Architecture

In this section, the hardware realization of the proposed algorithm for digit-level polynomial basis finite field GF(2^m) multiplication is presented. The proposed architecture is shown in Fig. 6.1. The operands A and B concurrently enter the architecture

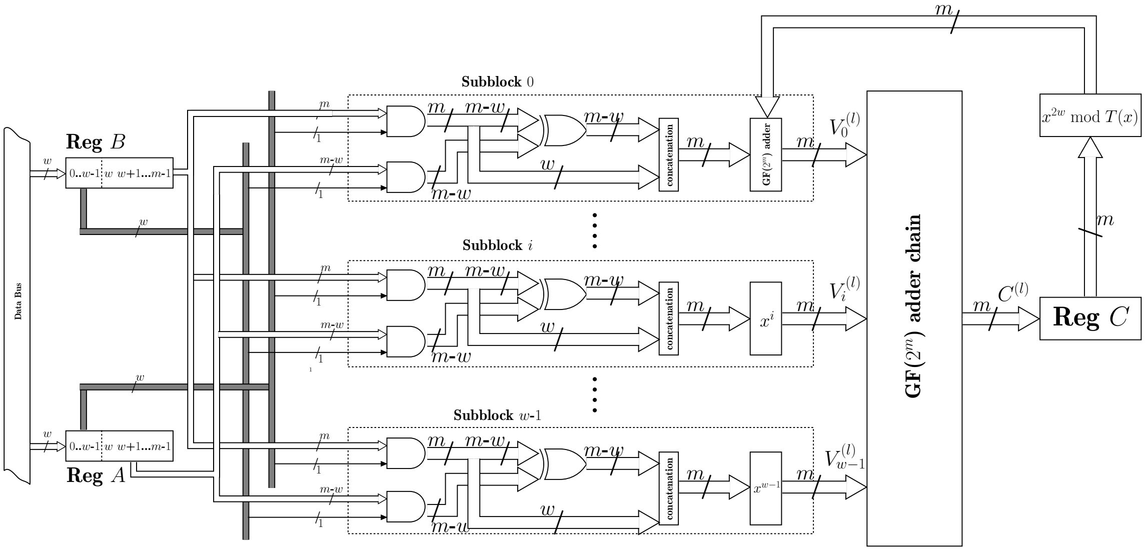

Figure 6.1 Proposed fully digit-serial polynomial basis GF(2^m) multiplier architecture.

digit-wise starting from the MSD. The architecture takes $(n+1)$ clock cycles to perform one multiplication operation. The input digits of each operand are loaded into m-bit input registers with one digit per clock cycle. During l^{th} clock cycle, the corresponding w partial products, $V_i^{(l)}, i = 0, 1, .., (w-1)$ are generated in parallel. The first partial product before its generation does not involve any multiplication with the powers of x (unlike the other partial products). Instead, it involves the adding of feedback m-bit data ($C^{(l-1)}x^{2w} \bmod T(x)$) to itself. The m-bit feedback data is obtained from the output register after it is multiplied with x^{2w}. Once these w partial products are generated, these are added by the GF(2^m) adder chain block. The mapping of the algorithm (algorithm 6.1) onto its hardware realization is as follows: Step 2 of the algorithm is performed by w subblocks. Step 4 is performed by subblock 0 which generates $V_0^{(l)}$, while step 6 is realized with the other $(w-1)$ lower subblocks. Step 9 in the algorithm is implemented with GF(2^m) adder chain.

6.2.2 Analytical Results

The hardware and time complexities of the proposed multiplier can be obtained from Fig. 6.1. The multiplier requires two m-bit input registers and one m-bit output register. Each subblock (Subblock i, for $i = 0, ..., i, .., w-1$) contains $(2m-w)$ AND gates (to realize the terms $a_{w(n-l)+i}(B^{(l)})$ and $b_{w(n-l)+i}(x^w A^{(l-1)})$), and $(m-w)$ number of XOR gates (to realize the addition of the terms $a_{w(n-l)+i}(B^{(l)})$ and $b_{w(n-l)+i}(x^w A^{(l-1)})$). The concatenation block requires simple wire routing and does not need any logic gates. In addition, each of the x^i multiplication blocks contains i number of XOR gates while the GF(2^m) adder of subblock 0 contains m XOR gates. Hence, all the w subblocks in the architecture require $(2m-w)w$ AND gates and $(m-w)w + m + w(w-1)/2$ XOR gates. The GF(2^m) adder chain requires $(w-1)m$ number of XOR gates to add the w number of $m-$bit data ($V_i^{(l)}, i = 0, 1, ..., w-1$). The $x^{2w} \bmod T(x)$ block in the feedback path requires $2w$ number of XOR gates. Hence, the proposed multiplier architecture requires $3m$ registers, $((2m-w)w)$ AND gates, and $(2mw - (w^2/2) + (3w/2))$ XOR gates. The area complexities in terms of 2-input AND gates, 2-input XOR gates, and registers are presented in Table 6.1 for the proposed multiplier along with similar multipliers [51–55] considered for comparison.

The time complexity of the architecture is given in terms of delays of logic gates. Assume T_A and T_X denote the delays of 2-input AND and 2-input XOR logic gates. The

Table 6.1 Area complexities comparison for GF(2^m).

Design	AND	XOR	Register
[51]	wm	$wm + 3w$	$2m + w$
[52]	wm	$wm + (w^2 + w)/2$	$2m + w$
[53]	wm	$wm + (w^2 + w)/2$	$2m + w$
[54]	$m^{log_4^6}$	$69/20m^{log_4^6} - 1/4m^{log_4^2} - 11/5$	$2m - 1$
[55]	wm	$wm + w^2/2 + 3w/2 - 1$	$2m$
Proposed	$(2m - w)w$	$2mw - w^2/2 + 3w/2$	$3m$

Table 6.2 Time complexities comparison for GF(2^m).

Design	Latency (*clock cycles*)	Critical path delay
[51]	$n + 2$	$T_A + (\lceil log_2^{2w+1} \rceil)T_X$
[52]	$n - 1$	$T_A + (\lceil log_2^w \rceil + 2)T_X$
[53]	$n + 1$	$T_A + (\lceil log_2^w \rceil + 2)T_X$
[54]	$n + 1$	$T_A + (1 + 3log_4^m)T_X$
[55]	$n + 1$	$T_A + (\lceil log_2^w \rceil + 2)T_X$
Proposed	$n + 1$	$T_A + (\lceil log_2^w \rceil + 2)T_X$

critical path of the architecture comprises of the delays through a subblock and delays through GF(2^m) adder chain. Hence, the delay of critical path is $T_A + (\lceil log_2^w \rceil + 2)T_X$. The time complexities for the proposed multiplier along with multipliers considered for comparison are presented in Table 6.2. It is noted that the multipliers proposed in the literature [51–55] require extra clock cycles for the preloading of one operand before the start of multiplication operation, resulting in increase in the latency. In these multipliers, after the loading of one operand, the entire data bus available is dedicated to loading another operand digit-wise. In the case of the proposed multiplier, both operands enter the architecture digit-wise requiring the available data bus to be shared between two operands. Hence, for a given data bus width, the proposed multiplier digit size is half of the digit size compared to other multipliers [51–55] that are considered for comparison.

Table 6.3 presents the estimation of complexities for the field order GF(2^{409}) as-

Table 6.3 Area and time complexities comparison for GF(2^{409}).

Design	**Area** (μm^2)	**Latency (including preloading)** (*clock cycles*)	**Critical Path Delay** (ns)	**Delay** (ns)	**Area-Delay-Product** ($\mu m^2 \times ns$)
[51]	11024	106	0.2	21.2	233709
[52]	11043	103	0.2	20.6	227485
[53]	11043	105	0.2	21	231903
[54]	18639	105	0.585	61.43	1144993
[55]	11030	105	0.2	21	231630
Proposed	12169	104	0.165	17.16	208820

suming a data bus width of 8. It implies $w = 8$ for all the architectures considered for comparison [51–55], and $w = 4$ for the proposed architecture. NanGate 45nm standard library statistics [46, 60] are used to estimate the area and time complexities. With this technology, the NAND gate equivalents for AND gate, XOR gate, and register are considered to be 1.4, 2, and 5.7. The delays for AND gate and XOR gate are considered to be 0.025 and 0.035. It is observed that the area required for a 2-input NAND gate based on Synopsys design compiler synthesis using 45nm NanGate open cell libraries is $0.8\mu m^2$. It is noted that the latency of all the multipliers used for comparison includes preloading of the first operand which requires 52 clock cycles for the considered case of $m = 409$ and $w = 8$. It is observed from Table 6.3 that the proposed architecture requires the lowest delay. Hence, the proposed multiplier achieves higher throughput than the other compared multipliers. In addition, it is observed that the proposed multiplier also achieves a reduction in critical path delay and area-delay-product.

6.2.3 Implementation Results

It is observed from Table. 6.3 that the multiplier [55] requires less area compared to the other multipliers considered for comparison. Hence, the proposed multiplier and the multiplier [55] are modeled using VHDL for GF(2^{409}). The RTL (Register Transfer Level) designs are simulated using Vivado Simulator to verify the functionality. The netlists of these models are synthesized using Synopsys Design Compiler tool employing NanGate 45nm open cell libraries [60] to obtain the area and time complexities. The area and time complexities obtained for these multipliers are tabulated in Table 6.4. It is observed from

Table 6.4 Comparison of ASIC implementation results for GF(2^{409}).

Design	Multiplier area (μm^2)	Critical path delay (ns)	Multiplication delay (ns)	Area × Delay ($\mu m^2 \times ns$)	Throughput ($\times 10^6$)	% increase in Throughput
[55]	13072	0.65	68.25	892164	14.6	26.71
Proposed	14116	0.52	54.08	763394	18.5	–

the ASIC (Application specific integrated circuit) implementation results that the proposed multiplier achieves 26% improvement in throughput compared to the multiplier [55]. The ASIC implementation results confirm that the proposed multiplier achieves better throughput rates compared to the available multipliers. Hence, the proposed multiplier is suitable for constrained devices in high-speed applications.

6.3 Low-Hardware Digit-Serial Sequential Multiplier

In this section, the formulations for the proposed low-hardware digit-serial multiplier and its architecture are presented. Analysis of this multiplier using analytical and implementation results and the comparisons with the related multipliers are also presented.

6.3.1 Design

Mathematical Formulations

Let $T(x) = x^m + x^k + 1$, where $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even), be an irreducible trinomial polynomial over which the field GF(2^m) is defined. Let $A(x) = \sum_{j=0}^{m-1} a_j x^j$ and $B'(x) = \sum_{j=0}^{w-1} b'_j x^j$ be two elements, where $w \leq m$. Let $D(x)$ denote the product of polynomials A and B' as $D(x) = \sum_{j=0}^{m+w-2} d_j x^j = AB'$. This product expression $D(x) = AB'$ can be expressed using a $(m+w-1) \times w$ matrix M as shown below.

The product polynomial $D(x)$ includes the terms whose degree is more than $m-1$ and these terms can be modulo reduced using the identity $x^m = x^k + 1$. From this identity we can have $x^{m+i} = (x^k + 1)x^i = x^{k+i} + x^i$, where the range of i is assumed to be in the range $0 \leq i \leq (w-2)$. Also, assume $w \leq (m-1)/4$ (if m is odd) or $w \leq m/4$ (if m is even), then we have $k+i \leq k+w-2 \leq m/2 + \lceil m/2 \rceil - 2 < m$. Thus each term in the product polynomial $D(x)$ whose degree is $(m+i)$ can be reduced to a polynomial of at most degree $(m-w)$ with two terms, $x^{k+i} + x^i$. By this modulo reduction, each $(m+i)^{th}$ row of matrix M for $0 \leq i \leq (w-2)$ is added to the i^{th} and $(k+i)^{th}$ rows of it.

$$\begin{bmatrix} d_0 \\ d_1 \\ . \\ . \\ . \\ d_{w-1} \\ d_l \\ . \\ . \\ . \\ d_{m-1} \\ d_m \\ . \\ . \\ . \\ d_{m+w-2} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 & .\ .\ .\ . & 0 & 0 \\ a_1 & a_0 & 0 & .\ .\ .\ . & 0 & 0 \\ . & . & . & .\ .\ .\ . & . & . \\ . & . & . & .\ .\ .\ . & . & . \\ . & . & . & .\ .\ .\ . & . & . \\ a_{w-1} & a_{w-2} & a_{w-3} & .\ .\ .\ . & a_1 & a_0 \\ a_w & a_{w-1} & a_{w-2} & .\ .\ .\ . & a_2 & a_1 \\ . & . & . & .\ .\ .\ . & . & . \\ . & . & . & .\ .\ .\ . & . & . \\ . & . & . & .\ .\ .\ . & . & . \\ a_{m-1} & a_{m-2} & a_{m-3} & .\ .\ .\ . & a_{m-w+1} & a_{m-w} \\ 0 & a_{m-1} & a_{m-2} & .\ .\ .\ . & a_{m-w+2} & a_{m-w+1} \\ . & . & . & .\ .\ .\ . & . & . \\ . & . & . & .\ .\ .\ . & . & . \\ . & . & . & .\ .\ .\ . & . & . \\ 0 & 0 & 0 & .\ .\ .\ . & 0 & a_{m-1} \end{bmatrix} \times \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ . \\ . \\ . \\ b'_{w-1} \end{bmatrix}$$

Let Q be a $m \times w$ matrix which is obtained from the matrix M, after the modulo reduction process applied. Let matrix Q be decomposed into the sum of three $m \times w$ matrices X, Y, and Z, such that $Q = X + Y + Z$. These three matrices X, Y, and Z can be defined as follows.

$$X = \begin{bmatrix} a_0 & 0 & 0 & . & . & . & . & 0 & 0 \\ a_1 & a_0 & 0 & . & . & . & . & 0 & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ a_{w-2} & a_{w-3} & a_{w-4} & . & . & . & . & a_0 & 0 \\ a_{w-1} & a_{w-2} & a_{w-3} & . & . & . & . & a_1 & a_0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ a_{m-1} & a_{m-2} & a_{m-3} & . & . & . & . & a_{m-w+1} & a_{m-w} \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & . & . & . & . & a_{m-w+2} & a_{m-w+1} \\ 0 & 0 & a_{m-1} & . & . & . & . & a_{m-w+3} & a_{m-w+2} \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & . & . & 0 & a_{m-1} \\ 0 & 0 & 0 & . & . & . & . & 0 & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & . & . & 0 & 0 \end{bmatrix}$$

$$Z = \begin{array}{c@{\quad}c}
\left[\begin{array}{ccccccccc}
0 & 0 & 0 & . & . & . & . & 0 & 0 \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & . & . & . & . & 0 & 0 \\
0 & a_{m-1} & a_{m-2} & . & . & . & . & a_{m-w+2} & a_{m-w+1} \\
0 & 0 & a_{m-1} & . & . & . & . & a_{m-w+3} & a_{m-w+2} \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & . & . & . & . & 0 & a_{m-1} \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & . & . & . & . & 0 & 0
\end{array}\right] &
\begin{array}{c}
0^{th} row \\ \\ \\ \\ k^{th} row \\ \\ \\ \\ (k+w-2)^{th} row \\ \\ \\ \\
\end{array}
\end{array}$$

Matrix Z is equivalent to a matrix that can be obtained by shifting matrix Y down by k rows and filling the first k rows with zeros. By employing similar method presented in [65], any i^{th} row of matrix Q can be obtained with simple rewiring of the k^{th} row, Q_k, of the matrx Q. The row Q_k can be computed as

$$Q_k = \begin{cases} (a_k a_{k-1} ... a_0 a_{m-1} a_{m-2} .... a_{m-w+k+1}) + (0 a_{m-1} ... a_{m-w+1}), & \text{if } k \leq w \\ (a_k a_{k-1} ..... a_{k-w+1}) + (0 a_{m-1} ... a_{m-w+1}), & \text{if } k > w \end{cases}$$

To compute Q_k, it requires $(w-1)$ two-input XOR gates, and a delay of T_X, where T_X is a dealy of a two-input XOR gate. Since Q is an $m \times w$ matrix, $Q.B'$ needs wm AND gates, $(w-1)m$ XOR gates, and $T_A + \lceil log_2^w \rceil T_X$ delays, where $B' = (b_0, b_1, ..., b_{l-1})^t$. After formulating this method of computing the multiplication for $D(x)$ of an m-bit element with a w-bit element where $w \leq (m-1)/4$ (if m is odd) or $w \leq m/4$ (if m is even), the multiplication of any two arbitrary field elements is considered now as follows.

Let $A(x) = \sum_{i=0}^{m-1} a_i x^i$ and $B(x) = \sum_{i=0}^{m-1} b_i x^i$ be two arbitrary field elements. Let $C(x) = \sum_{i=0}^{m-1} c_i x^i = AB \bmod T(x)$ be the product of the elements A and B. The computation of $C(x)$ can be performed as follows.

Let the element $B(x)$ is partitioned into s digits where each digit of size w bits. Then, we have $n = \lceil \frac{m}{w} \rceil$. It follows,

$$\begin{aligned} B(x) &= \sum_{j=0}^{w-1} b_j x^j + \sum_{j=l}^{2w-1} b_j x^j + ... + \sum_{j=(n-1)w}^{nw-1} b_j x^j \\ &= \sum_{j=0}^{w-1} b_j x^j + x^w \sum_{j=0}^{w-1} b_{w+j} x^j + +x^{2w} \sum_{j=0}^{w-1} b_{2w+j} x^j ...... + x^{(n-1)w} \sum_{j=0}^{w-1} b_{(n-1)w+j} x^j, \end{aligned} \quad (6.7)$$

where all b_js for $j \geq m$ are zero. Now, the product $C(x)$ can be computed as

$$\begin{aligned} C(x) = A(x)B(x) \bmod T(x) = A(x) \sum_{j=0}^{w-1} b_j x^j \bmod T(x) + \\ x^w A(x) \sum_{j=0}^{w-1} b_{w+j} x^j \bmod T(x) + x^{2w} A(x) \sum_{j=0}^{w-1} b_{2w+j} x^j \bmod T(x) + ... \\ ..... + x^{(n-1)w} A(x) \sum_{j=0}^{w-1} b_{(n-1)w+j} x^j \bmod T(x) \end{aligned}$$

$$= P_0 \bmod T(x) + x^w P_1 \bmod T(x) + x^{2w} P_2 \bmod T(x) + ... + x^{(n-1)l} P_{n-1} \bmod T(x)$$

$$\begin{aligned} = (.....((P_{n-1} x^w \bmod T(x) + P_{n-2}) x^w \bmod T(x) + P_{n-3}) x^w \bmod T(x) + ... \\ ..... + P_1) x^w \bmod T(x) + P_0, \end{aligned} \quad (6.8)$$

where

$$P_i = A(x) \sum_{j=0}^{w-1} b_{iw+j} x^j \bmod T(x). \quad (6.9)$$

The computation of P_i can be performed using the procedure shown to compute $D(x)$. In the computation of $D(x)$, note that the value of digit-size, w, is taken at most half the value of field order, m. It is acceptable for the constrained devices since the data bus width of these devices is typically 8/16/32 bits only. As per today's security requirements, a field order of at least 233 is required. Hence, the selected w range is quite applicable to today's security requirements for Wireless Sensor Network (WSN) nodes and IoT end-nodes/edge devices.

Proposed Structure of the Multiplier

Based on the proposed formulations, a conceptual block diagram of the digit-serial multiplier is shown in Fig. 6.2. The structure shown in Fig. 6.2 realizes the expression given in Eq. 6.8. Node M1 is a partial parallel $m \times w$ multiplier that multiplies an m-bit element with an w-bit element. It realizes the computation of P_i as given in Eq. 6.9. Node A_1 performs the additions that are involved in the computation of the expression in Eq. 6.8. Similarly, Node M_2 performs the interleaved multiplications of partial output

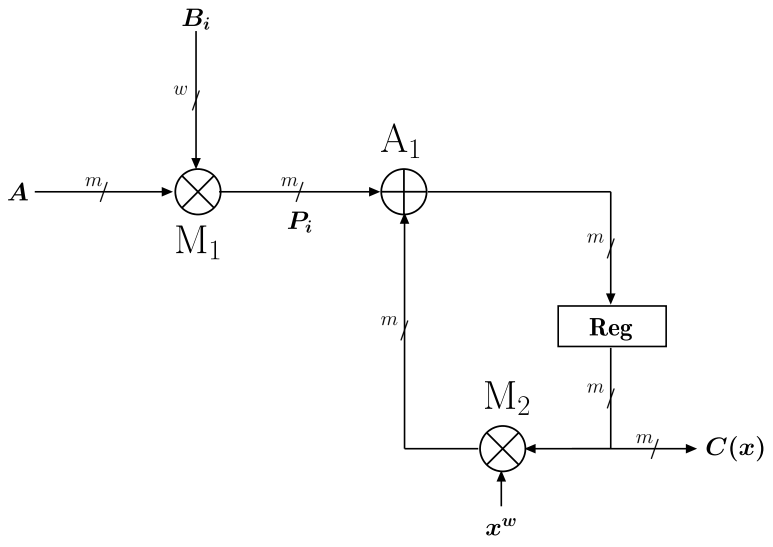

Figure 6.2 The proposed structure of the digit-serial multiplier.

product with x^w that are involved in the computation of the expression in Eq. 6.8. The multiplicand A is made available throughout the computation, while multiplier B enters the structure digit-wise starting from the most significant digit (MSD). The structure produces the required multiplication result after a delay of n clock cycles.

6.3.2 Analytical Results

In this section, the area and time complexities of the proposed multiplier are obtained and compared with the existing similar multipliers. The node M_1 which computes P_i performs a similar computation presented for computing $Q.B^{'}$. Hence, it requires wm AND gates and $(w-1)(m+1)$ XOR gates. The node A_1 requires m XOR gates while the

Table 6.5 Comparison of area complexities for GF(2^m).

Design	XOR	AND	Register
[51]	$wm + 3w$	wm	$2m + w$
[52]	$wm + (w^2 + w)/2$	wm	$2m + w$
[53]	$wm + (w^2 + w)/2$	wm	$2m + w$
[54]	$69/20m^{log_4^6} - 1/4m^{log_4^2} - 11/5$	$m^{log_4^6}$	$2m - 1$
[55]	$wm + w^2/2 + 3w/2 - 1$	wm	$2m$
Proposed	$wm + (2w - 1)$	wm	$2m$

node M_2 requires w XOR gates. The structure also requires two m-bit registers, one at the input to register multiplicand A while another as output register, Reg. The delays of the nodes M_1, A_1 and M_2 are $T_A + (\lceil log_2^w \rceil + 1)T_X, T_X$, and T_X respectively. The critical path of the structure is $T_A + (\lceil log_2^w \rceil + 2)T_X$. The area and time complexities for the proposed multiplier and the other similar multipliers [51], [52], [53], [54], [55] are presented in Table 6.5 and Table 6.6, respectively.

Table 6.6 Comparison of time complexities for GF(2^m).

Design	Critical path	Latency (*clock cycles*)
[51]	$T_A + (\lceil log_2^{2w+1} \rceil)T_X$	$n + 2$
[52]	$T_A + (\lceil log_2^w \rceil + 2)T_X$	$n - 1$
[53]	$T_A + (\lceil log_2^w \rceil + 2)T_X$	$n + 1$
[54]	$T_A + (1 + 3log_4^m)T_X$	$n + 1$
[55]	$T_A + (\lceil log_2^w \rceil + 2)T_X$	$n + 1$
Proposed	$T_A + (\lceil log_2^w \rceil + 2)T_X$	n

The analytical comparisons presented in Table 6.5 and Table 6.6 can be better understood by considering a specific field order m and a specific digit size w. By selecting the field to be GF(2^{409}) over an irreducible polynomial $x^{409} + x^{87} + 1$ with a digit-size $w = 8$, the complexities presented in Table 6.5 and Table 6.6 are computed and presented in Table 6.7.

Table 6.7 Area and time complexities comparison for GF(2^{409}) over $x^{409} + x^{87} + 1$ with $w = 8$.

Design	**Area** (μm^2)	**Latency** (*clock cycles*)	**Critical Path Delay** (ns)	**Delay** (ns)	**Area-Delay-Product** ($\mu m^2 \times ns$)
[51]	11024	54	0.2	10.80	119059
[52]	11043	51	0.2	10.20	112638
[53]	11043	53	0.2	10.60	117056
[54]	18639	53	0.585	31	577809
[55]	11030	53	0.2	10.60	116918
Proposed	10986	52	0.2	10.40	114254

We have NanGate 45nm standard library statistics [46, 60] to estimate the time and area requirements. With this technology, the NAND gate equivalents for XOR gate, AND gate, and register are assumed to be 2, 1.4, and 5.7. The delays for XOR gate and AND gate are assumed to be 0.035 and 0.025. It is observed that the area required for a 2-input NAND gate based on Synopsys design compiler synthesis using 45nm NanGate open cell libraries is $0.8\mu m^2$. It is observed from Table 6.7, the proposed multiplier requires marginally less hardware when compared with other similar multipliers. It is also observed that the proposed multiplier achieves low area-delay-product as well.

6.3.3 Implementation Results

The proposed multiplier and the multiplier [55] are modeled using VHDL for GF(2^{409}). The RTL designs are simulated using Vivado Simulator to verify the functionality. The netlists of these models are synthesized using Synopsys Design Compiler tool employing NanGate 45nm open cell libraries [60] to obtain the area and time complexities. The area and time complexities obtained for these multipliers are tabulated in Table 6.8. It is observed from the ASIC implementation results that the proposed multiplier achieves a marginal reduction in area and in ADP (Area-delay-product) (3%) compared to the

Table 6.8 Comparison of ASIC implementation results for GF(2^{409}).

Design	**Multiplier area (μm^2)**	**Critical path delay (ns)**	**Multiplication delay (ns)**	**Area × Delay ($\mu m^2 \times ns$)**	**% reduction in ADP**
[55]	13072	0.65	34.45	450330	3.08
Proposed	12912	0.65	33.80	436425	–

multiplier [55]. Hence, the proposed digit-serial sequential multiplier is suitable for IoT end devices which typically have a bus width of 8/16/32 bits.

6.4 Conclusions

In this chapter, the design of the two digit-serial multipliers and the performance analysis of these multipliers using analytical and implementation results are presented. First, a fully digit-serial multiplier, where both the operands enter the multiplier architecture simultaneously, is presented. Design of this multiplier is based on a redundant basis multiplier available in the literature. Comparisons of this multiplier with the available multipliers show that it achieved improved throughput rates. Next, another digit-serial multiplier is presented whose parallel multiplier is based on the Mastrovito multiplier. The available Mastrovito multiplier is tailored for the considered class of trinomials to achieve hardware efficiency. The comparisons show that the proposed digit-serial multiplier achieves a marginal reduction in area compared to the available multipliers. These proposed scalable multipliers are suitable for a wide range of IoT applications where the data bus width of the processor can be 8/16/32/64 bits.

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

Internet of Things (IoT) is the state-of-the-art widely used communication technology having numerous application areas. This technology includes the constrained devices namely IoT end devices and edge devices. End devices are required to be low-cost while edge devices are required to have high-performance. Further, security is a major concern in these devices to be addressed, and elliptic curve cryptography (ECC) provides some of the security features required in these devices. GF(2^m) multiplication is a performance-critical operation in this cryptography, which requires efficient hardware implementations. Bit-serial sequential multipliers using general irreducible polynomials are suitable for end devices, since they offer low hardware complexities that result in low-cost implementations. Bit-parallel systolic multipliers using trinomials are suitable for edge devices, since they offer high-throughput rates that result into high-performance impementations. Further, digit-serial multipliers which are scalable are also required for IoT devices, since these multipliers can provide area-delay trade-off as required by the application at hand. This thesis aims at offering some area and time efficient multiplier architectures that are targeted for security implementation in IoT devices. The contributions of the work are concluded as follows:

- In Chapter 4, we have presented the design of bit-serial sequential multipliers over general irreducible polynomials, since these multipliers are suitable for security implementation in low-cost IoT devices. In this regard, we have presented an area-efficient bit-serial sequential multiplier using the proposed modified interleaved multiplication algorithm. Through the comparisons of analytical and ASIC (Apllication specific integrated circuit) implementation results obtained for $m = 409$, we have shown that the proposed multiplier is indeed area-efficient and achieves a minimum of 28% reduction in area and a minimum of 3% reduction in ADP (Area-delay-product) compared to existing multipliers. We have also presented two area and time efficient bit-serial sequential multipliers using the proposed modified Montgomery MSB (most significant bit)-first and LSB-first bit-serial algorithms. Through analytical and ASIC implementation comparisons, we have shown that the proposed multipliers are indeed area and time efficient compared to existing multipliers. The proposed MSB-multiplier and LSB (least significant bit)-multiplier achieve a minimum of 12% and 11% reduction in ADP, respectively. These proposed multipliers are suitable for generic IoT devices as they are defined over general irreducible polynomials.

- Chapter 5 has presented the design of bit-parallel systolic multipliers over a few specific classes of trinomials, since these multipliers are suitable for security implementation in high-performance IoT devices. Though many multipliers presented in the literature have been defined over trinomials, we have observed that multipliers defined over a few specific classes also include NIST (National Institute of Standards and Technology) recommended trinomials and result in further less area and time complexities as well. Hence, multipliers defined over these classes are particularly suitable for IoT devices as they require low hardware and time complexities. Consequently, we have proposed a low-latency area-efficient multiplier and a high-throughput multiplier over the class of trinomials x^m+x^k+1 for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). In addition to that we have also proposed another low-latency area-efficient multiplier for a further narrow class of trinomials for which $k \leq m - 2\lceil m/3 \rceil$. Through the comparisons of analytical and ASIC implementation results obtained for $m = 409$, we have shown that the proposed low-latency multipliers are indeed area and time efficient and the proposed high-throughput multiplier is indeed time efficient compared to existing multipliers. The low-latency systolic multiplier defined over the trinomials for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even) achieves a minimum of 9% reduction in area and a minimum of 12% reduction in latency and the other low-latency multiplier defined over the trinomials for which $k \leq m - 2\lceil m/3 \rceil$ achieves a minimum 4% reduction in area and a minimum 25% reduction in latency. Also, the proposed high-throughput multiplier achives a minimum of 35% increase in throughput and a minimum of 5% reduction in ADP compared to the existing works.

- In Chapter 6, we have presented the design of digit-serial sequential multipliers over a specific class of trinomials, since these multipliers due to their scalability are suitable for security implementation in IoT end/edge devices that can be used in a wide variety of applications. The class of trinomials considered in this chapter is the same class of trinomials considered in Chapter 5 for the high-throughput systolic multiplier, i.e., the trinomials $x^m + x^k + 1$ for which $k \leq (m-1)/2$ (if m is odd) or $k \leq m/2$ (if m is even). Consequently, we have presented a high-throughput digit-serial sequential multiplier using the proposed fully digit-serial multiplication algorithm. This algorithm is based on a redundant basis multiplication algorithm and is obtained by adapting it to polynomial basis. This is the first time in polynomial basis literature to design a fully digit-serial multiplier where both the operands enter the architecture simultaneously. Through the comparisons of analytical and ASIC implementation results obtained for $m = 409$, we have shown that the proposed multiplier is indeed high-throughput and achieves a minimum of 26% increase in throughput compared to existing multipliers. We have also presented another low-hardware digit-serial sequential multiplier where the area reduction is achieved using a modified parallel multiplier. We have obtained this parallel multiplier by tailoring the available Mastrovito multiplier for the class of trinomials considered. Through analytical and ASIC implementation comparisons, we have shown that the proposed multiplier is indeed area-efficient and achieves a marginal reduction in area compared to the best existing multiplier. This proposed multiplier achieves a minimum of 3% reduction in ADP compared to existing multipliers.

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 architectures proposed in this thesis achieve reduction in area and time complexities. Further, apart from these area and time complexities reduction, power reduction is also an important requirement for IoT devices as in many cases these devices are battery-powered. Hence, an important extension would be to implement low-power techniques into these designs to achieve reduction in power.
- In finite field arithmetic, inversion and exponentiation are computationally intensive operations and repeatedly use multiplication. Hence, using the multipliers proposed in this thesis, efficient field inversion and exponentiation operations can be realized to improve the efficiency of the overall finite field arithmetic.
- Fault tolerance in field multipliers is a method that ensures reliability, and also prevents many fault-based attacks on cryptosystems that use field arithmetic. Multipliers with concurrent error detection and correction capabilities support testing and correcting cryptosystems while they are in operation. Our future research includes the development of some new fault-tolerant techniques and applying these techniques to multipliers to further strengthen the security of the IoT applications.

Publications

List of International Journals:

1. Pillutla Siva Ramakrishna and Boppana Lakshmi, "An area-efficient bit-serial sequential polynomial basis finite field GF(2^m) multiplier," *AEU-International Journal of Electronics and Communications*, 114 (2020): 153017. **(SCI-Indexed, Elsevier)**

2. Pillutla Siva Ramakrishna and Boppana Lakshmi, "Area-efficient low-latency polynomial basis finite field GF(2^m) systolic multiplier for a class of trinomials," *Microelectronics Journal*, 97 (2020): 104709. **(SCI-Indexed, Elsevier)**

3. Pillutla Siva Ramakrishna and Boppana Lakshmi, "High-Throughput Area-Delay-Efficient Systolic Multiplier over GF(2^m) for a Class of Trinomials," *Microprocessors and Microsystems*, (2020): 103173. **(SCI-Indexed, Elsevier)**

4. Pillutla Siva Ramakrishna and Boppana Lakshmi, "Low-complexity bit-serial sequential polynomial basis finite field GF(2^m) Montgomery multipliers," *Microprocessors and Microsystems*, (2021): 104053. **(SCI-Indexed, Elsevier)**

5. Pillutla Siva Ramakrishna and Boppana Lakshmi, "Low-Latency Area-Efficient Systolic Bit-parallel GF(2^m) Multiplier for a Narrow Class of Trinomials," *Microelectronics Journal.* **(Under Review) (SCI-Indexed, Elsevier)**

List of International Conferences:

1. Pillutla Siva Ramakrishna and Boppana Lakshmi, "A high-throughput fully digit-serial polynomial basis finite field GF(2^m) multiplier for IoT applications," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON), IEEE, 2019.*

2. Pillutla Siva Ramakrishna and Boppana Lakshmi, ”Low-hardware Digit-serial Sequential Polynomial Basis Finite Field GF(2^m) Multiplier for Trinomials,” in *Advances in Communications, Signal Processing, and VLSI Conference. Springer, 2019.*

Bibliography

[1] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[2] M. Capra, R. Peloso, G. Masera, M. R. Roch, and M. Martina, "Edge computing: A survey on the hardware requirements in the Internet of Things world," *Future Internet*, vol. 11, no. 4, p. 100, 2019.

[3] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end IoT devices," *IEEE Access*, vol. 6, pp. 70 528–70 554, 2018.

[4] M. A. Razzaq, S. H. Gill, M. A. Qureshi, and S. Ullah, "Security issues in the Internet of Things (IoT): a comprehensive study," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 8, no. 6, pp. 383–388, 2017.

[5] W. Stallings, *Cryptography and network security, 4/E.* Pearson Education India, 2006.

[6] M. Katagi and S. Moriai, "Lightweight cryptography for the internet of things," *Sony Corporation*, pp. 7–10, 2008.

[7] L. Marin, M. Pawlowski, and A. Jara, "Optimized ECC implementation for secure communication between heterogeneous IoT devices," *Sensors*, vol. 15, no. 9, pp. 21 478–21 499, 2015.

[8] L.-Y. Yeh, P.-J. Chen, C.-C. Pai, and T.-T. Liu, "An Energy-Efficient Dual-Field Elliptic Curve Cryptography Processor for Internet of Things Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 9, pp. 1614–1618, 2020.

[9] E. Wenger and M. Hutter, "Exploring the design space of prime field vs. binary field ecc-hardware implementations," in *Nordic Conference on Secure IT Systems.* Springer, 2011, pp. 256–271.

[10] Y. Chen, S. Lu, C. Fu, D. Blaauw, R. Dreslinski Jr, T. Mudge, and H.-S. Kim, "A programmable Galois field processor for the internet of things," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 55–68.

[11] M. Imran, I. Shafi, A. R. Jafri, and M. Rashid, "Hardware design and implementation of ECC based crypto processor for low-area-applications on FPGA," in *2017 International Conference on Open Source Systems & Technologies (ICOSST).* IEEE, 2017, pp. 54–59.

[12] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Lightweight elliptic curve cryptography accelerator for internet of things applications," *Ad Hoc Networks*, p. 102159, 2020.

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

[14] J. Deschamps, J. Imaña, and G. Sutter, *Hardware Implementation of Finite-Field Arithmetic.* New York: McGraw-Hill, 2009.

[15] G. R. Blakely, "A computer algorithm for calculating the product AB modulo M," *IEEE Transactions on Computers*, vol. 100, no. 5, pp. 497–500, 1983.

[16] S. E. Mathe and L. Boppana, "Design and Implementation of a Sequential Polynomial Basis Multiplier over GF(2^m)," *KSII Transactions on Internet & Information Systems*, vol. 11, no. 5, 2017.

[17] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," in *Doklady Akademii Nauk*, vol. 145, no. 2. Russian Academy of Sciences, 1962, pp. 293–294.

[18] T.-Y. Lee, M.-J. Liu, C.-H. Huang, C.-C. Fan, C.-C. Tsai, and H. Wu, "Design of a digit-serial multiplier over GF(2^m) using a karatsuba algorithm," *Journal of the Chinese Institute of Engineers*, vol. 42, no. 7, pp. 602–612, 2019.

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

[20] J. Xie, J. jun He, and P. K. Meher, "Low latency systolic Montgomery multiplier for finite field GF(2^m) based on pentanomials," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 21, no. 2, pp. 385–389, 2012.

[21] E. D. Mastrovito, "VLSI designs for multiplication over finite fields GF(2^m)," in *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes.* Springer, 1988, pp. 297–309.

[22] Y. Li, X. Ma, Y. Zhang, and C. Qi, "Mastrovito form of non-recursive Karatsuba multiplier for all trinomials," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1573–1584, 2017.

[23] A. Reyhani-Masoleh, "A new bit-serial architecture for field multiplication using polynomial bases," in *Proc. 10th Int. Workshop on Cryptographic Hardware and Embedded Systems*, vol. 5154, Heidelberg, 2008, pp. 300–314.

[24] J. L. Imaña, J. M. Sanchez, and F. Tirado, "Bit-parallel finite field multipliers for irreducible trinomials," *IEEE Transactions on Computers*, vol. 55, no. 5, pp. 520–533, 2006.

[25] J. L. Imaña, "Low latency GF(2^m) polynomial basis multiplier," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 5, pp. 935–946, 2010.

[26] A. Cilardo, "Fast Parallel GF(2^m) Polynomial Multiplication for All Degrees," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 929–943, 2012.

[27] P. K. Meher, "Systolic and super-systolic multipliers for finite field GF(2^m) based on irreducible trinomials," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 4, pp. 1031–1040, 2008.

[28] L. Song and K. K. Parhi, "Efficient finite field serial/parallel multiplication," in *Proceedings of International Conference on Application Specific Systems, Architectures and Processors: ASAP'96.* IEEE, 1996, pp. 72–82.

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

[30] H. Ho, “Design and Implementation of a Polynomial Basis Multiplier Architecture Over GF(2^m),” *Journal of Signal Processing Systems*, vol. 75, no. 3, pp. 203–208, 2014.

[31] A. Hariri and A. Reyhani-Masoleh, “Bit-serial and bit-parallel montgomery multiplication and squaring over GF(2^m),” *IEEE Transactions on Computers*, vol. 58, no. 10, pp. 1332–1345, 2009.

[32] M. Morales-Sandoval, C. Feregrino-Uribe, and P. Kitsos, “Bit-serial and digit-serial GF(2^m) montgomery multipliers using linear feedback shift registers,” *IET Computers & Digital Techniques*, vol. 5, no. 2, pp. 86–94, 2011.

[33] H. Wu, “Low complexity LFSR based bit-serial montgomery multiplier in GF(2^m),” in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*. IEEE, 2013, pp. 1962–1965.

[34] ——, “Efficient bit-serial finite field montgomery multiplier in GF(2^m),” in *2014 4th IEEE International Conference on Information Science and Technology*. IEEE, 2014, pp. 527–530.

[35] C. K. Koc and T. Acar, “Montgomery multiplication in GF(2^k),” *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57–69, 1998.

[36] H. Fan, J. Sun, M. Gu, and K.-Y. Lam, “Overlap-free Karatsuba–Ofman polynomial multiplication algorithms,” *IET Information security*, vol. 4, no. 1, pp. 8–14, 2010.

[37] T. Beth and D. Gollmann, “Algorithm engineering for public key algorithms,” *IEEE Journal on selected areas in communications*, vol. 7, no. 4, pp. 458–466, 1989.

[38] J. Grossschadl, “A low-power bit-serial multiplier for finite fields GF(2^m),” in *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No. 01CH37196)*, vol. 4. IEEE, 2001, pp. 37–40.

[39] C. Lee, E. Lu, and J. Lee, "Bit-parallel systolic multipliers for GF(2^m) fields defined by all-one and equally spaced polynomials," *IEEE Transactions on Computers*, vol. 50, no. 5, pp. 385–393, 2001.

[40] C.-Y. Lee, "Low complexity bit-parallel systolic multiplier over GF(2^m) using irreducible trinomials," *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 1, pp. 39–42, 2003.

[41] C. Y. Lee, "Low-latency bit-parallel systolic multiplier for irreducible $x^m + x^n + 1$ with gcd(m, n)= 1," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 86, no. 11, pp. 2844–2852, 2003.

[42] C.-Y. Lee, J.-S. Horng, I.-C. Jou, and E.-H. Lu, "Low-complexity bit-parallel systolic Montgomery multipliers for special classes of GF(2^m)," *IEEE Transactions on Computers*, vol. 54, no. 9, pp. 1061–1070, 2005.

[43] C.-Y. Lee, "Low-complexity parallel systolic Montgomery multipliers over GF(2^m) using Toeplitz Matrix-vector representation," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 91, no. 6, pp. 1470–1477, 2008.

[44] S. Bayat-Sarmadi and M. A. Hasan, "Concurrent error detection in finite-field arithmetic operations using pipelined and systolic architectures," *IEEE Transactions on computers*, vol. 58, no. 11, pp. 1553–1567, 2009.

[45] J.-f. Xie, J.-j. He, and W.-h. Gui, "Low latency systolic multipliers for finite field GF(2^m) based on irreducible polynomials," *Journal of Central South University*, vol. 19, no. 5, pp. 1283–1289, 2012.

[46] S. Bayat-Sarmadi and M. Farmani, "High-throughput low-complexity systolic Montgomery multiplication over GF(2^m) based on trinomials," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 4, pp. 377–381, 2015.

[47] C.-S. Yeh, I. S. Reed, and T.-K. Truong, "Systolic multipliers for finite fields GF(2^m)," *IEEE Computer Architecture Letters*, vol. 33, no. 04, pp. 357–360, 1984.

[48] C.-L. Wang and J.-L. Lin, “Systolic array implementation of multipliers for finite fields GF(2^m),” *IEEE Transactions on Circuits and Systems*, vol. 38, no. 7, pp. 796–800, 1991.

[49] S. K. Jain, L. Song, and K. K. Parhi, “Efficient semisystolic architectures for finite-field arithmetic,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 101–113, 1998.

[50] K. K. Parhi, *VLSI digital signal processing systems: design and implementation.* John Wiley & Sons, 2007.

[51] 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, no. 2, pp. 149–166, 1998.

[52] W. Tang, H. Wu, and M. Ahmadi, “VLSI implementation of bit-parallel word-serial multiplier in GF(2^{233}),” in *The 3rd International IEEE-NEWCAS Conference, 2005.* IEEE, 2005, pp. 399–402.

[53] 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.

[54] C.-Y. Lee, C.-S. Yang, B. K. Meher, P. K. Meher, and J.-S. Pan, “Low-complexity digit-serial and scalable SPB/GPB multipliers over large binary extension fields using (b, 2)-way Karatsuba decomposition,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 11, pp. 3115–3124, 2014.

[55] 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, 2017.

[56] A. Hariri and A. Reyhani-Masoleh, “Digit-serial structures for the shifted polynomial basis multiplication over binary extension fields,” in *International Workshop on the Arithmetic of Finite Fields.* Springer, 2008, pp. 103–116.

[57] S. Kumar, T. Wollinger, and C. Paar, “Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography,” *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1306–1311, 2006.

[58] A. H. Namin, H. Wu, and M. Ahmadi, “Comb Architectures for Finite Field Multiplication in F_{2^m},” *IEEE Transactions on Computers*, vol. 56, no. 7, pp. 909–916, 2007.

[59] ——, “A High-Speed Word Level Finite Field Multiplier in F_{2^m} Using Redundant Representation,” *IEEE transactions on very large scale integration (VLSI) systems*, vol. 17, no. 10, pp. 1546–1550, 2009.

[60] “NanGate Standard Cell Library. [Online]. Available: http://www.si2. org/.”

[61] F. PUB, “186-2 digital signature standard (DSS),” *National Institute of Standards and Technology (NIST)*, vol. 20, p. 13, 2000.

[62] S. R. Pillutla and L. Boppana, “Area-efficient low-latency polynomial basis finite field GF(2^m) systolic multiplier for a class of trinomials,” *Microelectronics Journal*, p. 104709, 2020.

[63] M. Morales-Sandoval, C. Feregrino-Uribe, P. Kitsos, and R. Cumplido, “Area/performance trade-off analysis of an FPGA digit-serial GF(2^m) montgomery multiplier based on LFSR,” *Computers & Electrical Engineering*, vol. 39, no. 2, pp. 542–549, 2013.

[64] P. H. Namin, R. Muscedere, and M. Ahmadi, “A fully serial-in parallel-out digit-level finite field multiplier in F_{2^m} using redundant representation,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 11, pp. 1337–1341, 2017.

[65] B. Sunar and C. K. Koc, “Mastrovito multiplier for all trinomials,” *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 522–527, 1999.

[66] Y. Choi, K.-Y. Chang, D. Hong, and H. Cho, “Hybrid multiplier for GF(2^m) defined by some irreducible trinomials,” *Electronics Letters*, vol. 40, no. 14, pp. 852–853, 2004.