RECONFIGURABLE AND COMPACT MODULAR POLYNOMIAL MULTIPLIER IN GALOIS FIELD FOR THE SECURITY OF IOT

FARIHA HAROON
Bachelor of Science, University of Delhi, 2022

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

RECONFIGURABLE AND COMPACT MODULAR POLYNOMIAL MULTIPLIER IN GALOIS FIELD FOR THE SECURITY OF IOT

FARIHA HAROON

Date of Defence: August 07, 2025

Dr. Hua Li Thesis Supervisor	Associate Professor	Ph.D.
Dr. Robert Benkoczi Thesis Examination Committee Member	Professor	Ph.D.
Dr. Wendy Osborn Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Andrew Fiori Chair, Thesis Examination Committee Member	Associate Professor	Ph.D.

Abstract

The rise of the Internet of Things (IoT) has intensified the need for secure, low-power cryptographic hardware capable of operating efficiently in constrained environments. This thesis presents the design and implementation of reconfigurable and compact modular polynomial multipliers over Galois Fields (GF), specifically tailored for the security demands of IoT devices. Leveraging polynomial basis arithmetic in $GF(2^m)$, the proposed designs emphasize hardware efficiency, adaptability, and cryptographic robustness for Elliptic Curve Cryptography (ECC) applications.

The proposed multipliers were synthesized and validated on multiple FPGA platforms including Spartan-7, Zynq UltraScale+, and Artix-7 using the AMD Xilinx Vivado toolchain. The Karatsuba reconfigurable multiplier achieved the best area-delay product (ADP) across platforms. Analytical and experimental comparisons confirm that the reconfigurable approaches significantly outperform conventional designs in terms of efficiency and scalability. This research provides a practical and versatile foundation for cryptographic accelerators in future low-power and high-security IoT systems.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Hua Li, for his guidance, and support throughout the course of this research. His expertise and thoughtful insights were helpful in shaping both the technical direction and overall quality of this thesis.

I would also like to sincerely thank my thesis committee members, Dr. Robert Benkoczi and Dr. Wendy Osborn, for their valuable feedback, and helpful suggestions that greatly improved the depth and clarity of this work.

A heartfelt thank you goes to my family and friends for their constant support, patience, and belief in me. Their encouragement through challenging times kept me focused and motivated to achieve my goals. This journey would not have been possible without the guidance and support of all those mentioned above. I am truly grateful.

Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	**Full Form**
ADP	Area-Delay Product
AMD	Advanced Micro Devices
AOP	All-One Polynomial
BL-SIPO	Bit-Level Serial-In Parallel-Out
CA	Conventional Algorithm
CMOS	Complementary Metal–Oxide–Semiconductor
DFG	Data Flow Graph
DLP	Discrete Logarithm Problem
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
FPGA	Field-Programmable Gate Array
$GF(2^n)$	Galois Field of 2 to the power n
IoT	Internet of Things
KA	Karatsuba Algorithm
KOA	Karatsuba-Ofman Algorithm
NIST	National Institute of Standards and Technology
OKA	Overlap-Free Karatsuba Algorithm
PE	Processing Element
RSA	Rivest-Shamir-Adleman
RTL	Register Transfer Level
SoC	System-on-Chip
TA	Delay of AND gate
TMUX	Delay of Multiplexer
TNAND	Delay of NAND gate
TX	Delay of XOR gate
VLSI	Very Large Scale Integration

Chapter 1

Introduction

The introduction outlines the motivation, challenges, and contributions of this thesis in the context of securing Internet of Things (IoT). With the rapid growth of IoT applications, particularly in resource-constrained devices, ensuring efficient and robust cryptographic operations has become essential. The chapter begins by examining IoT security needs and the computational challenges of cryptography, followed by an overview of key establishment methods, the limitations of traditional public-key algorithms, and the advantages of ECC for lightweight systems. It then highlights the performance bottleneck in ECC—finite-field polynomial multiplication—and presents the proposed hardware-efficient, reconfigurable multiplier architectures designed to enhance speed, flexibility, and scalability in FPGA implementations.

1.0.1 IoT Security Needs and Computational Challenges

The IoT refers to a network of interconnected physical devices, embedded with sensors, software, and communication technologies, that can collect, exchange, and process data over the internet. IoT is widely used in various domains such as smart homes, healthcare, industrial automation, transportation, and environmental monitoring due to its ability to improve efficiency, enable real-time decision-making, and provide intelligent services. These IoT devices can be broadly divided into two categories: *resource-rich devices*, which have high processing power, memory, and energy availability, and *resource-constrained devices*, which operate with limited computational capabilities, storage, and power supply. In this thesis, the focus is on resource-constrained devices, as their limited resources make the implementation of secure and efficient cryptographic solutions particularly challenging.

The increasing reliance on IoT has led to a growing demand for efficient cryptographic solutions. A critical challenge in cryptography is the intensive computational complexity associated with large integer and polynomial multiplications, which form the backbone of many encryption algorithms. Given the proliferation of IoT-based systems, ensuring data security through lightweight cryptographic algorithms is critical to prevent network-based attacks.

1.0.2 Key Establishment and Digital Signatures in IoT

IoT devices must securely create cryptographic keys (i.e. key establishment) and use digital signatures to authenticate and validate messages. These defensive mechanisms aid in securing private information and protecting against network-based threats such as spoofing, data manipulation, and man-in-the-middle attacks.

Key establishment methods can be classified into two categories: symmetric and asymmetric (Schneier, 1996). Symmetric key establishment requires that a shared secret key be securely exchanged before any encrypted communication can begin.

1.0.3 Symmetric vs. Asymmetric Cryptography

Asymmetric key establishment cryptography uses a set of two key known as public/private key pair, where the public key can be shared openly (e.g., posted on a website or sent via email), while the private key is kept secret by its owner. This eliminates the need for a secure channel to distribute keys, greatly simplifying key management. Public-key cryptographic algorithms such as Diffie–Hellman (Diffie and Hellman, 2022), Rivest-Shamir-Adleman (RSA) (Rivest, 1978), ElGamal (ElGamal, 1985), and Elliptic Curve Cryptography (ECC) (Koblitz, 1987) are commonly used for this purpose.

1.0.4 Limitations of RSA and Diffie–Hellman

RSA which was first proposed algorithm has relatively slow encryption and decryption speeds, making it impractical to encrypt large messages, particularly on mobile devices. As a result, RSA is primarily utilized for tasks such as key exchange and digital signatures, while the actual data are secured using symmetric encryption algorithms with short-lived session keys. Similarly, the security of Diffie–Hellman algorithm depends on the Discrete Logarithm Problem (DLP) in finite fields. To achieve strong security, they require very large key sizes (e.g., 2048-bit or higher).

1.0.5 Advantages of ECC for IoT

In contrast, ECC provides comparable or even superior security with significantly shorter key lengths, making it more efficient in terms of computational performance and bandwidth usage. ECC is the most suitable for IoT applications due to its ability to provide strong security with shorter key sizes, making it efficient and lightweight for devices limited in resources (Miller, 1985).

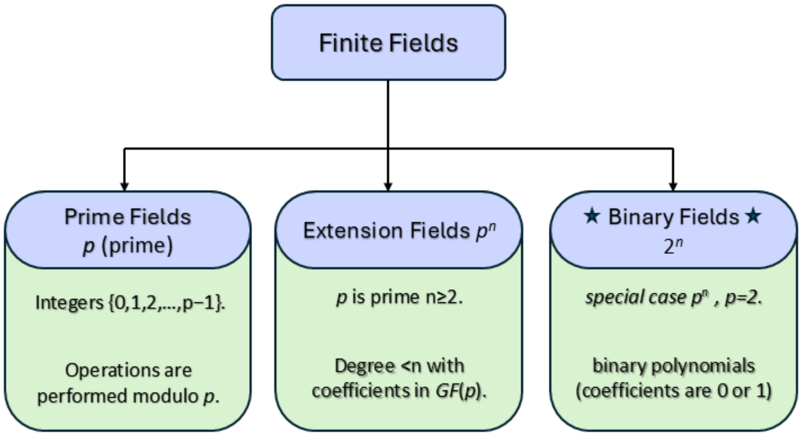
Finite Fields
Prime Fields
p (prime)
Integers {0,1,2,...,p−1}.
Operations are performed modulo p.
Extension Fields p^n
p is prime n≥2.
Degree <n with coefficients in $GF(p)$.
★ Binary Fields ★
2^n
special case p^n , $p=2$.
binary polynomials (coefficients are 0 or 1)

Figure 1.1: Different Types of Finite Field for Arithmetic Operations

The core arithmetic operations in ECC rely on finite-field (Odlyzko, 1984) $GF(2^n)$ computations. As illustrated in Figure 1.1, finite fields can be broadly classified into three main categories: *prime fields* $GF(p)$, *extension fields* $GF(p^n)$, and *binary fields* $GF(2^n)$. In this work, we will focus on examining binary fields $GF(2^n)$. A $GF(2^n)$ is a finite field containing exactly 2^n elements, where n is a positive integer. It is constructed as an extension field of the prime field $GF(2)$, whose elements are $\{0,1\}$ with arithmetic performed modulo 2. The elements of $GF(2^n)$ can be represented as polynomials of degree less than n with coefficients in $GF(2)$, and addition and multiplication are carried out modulo an irreducible polynomial of degree n over $GF(2)$. The computation of elliptic curve points is heavily based on finite-field arithmetic operations, with finite-field multiplication being the most critical task in the hardware implementation of ECC. The size and delay of the multiplier significantly impact the area utilization, path delay, and throughput of the entire system, making it a key factor in optimizing ECC hardware designs. These elliptic curves are described by equations of the form,

$$y^2 = x^3 + ax + b \tag{1.1}$$

where a and b are constants, and the curve is defined over a finite field. When plotted, an elliptic curve exhibits a distinctive red symmetrical shape as shown in Figure 1.2. ECC is built upon a branch of advanced mathematics involving elliptic curves.

One of the notable properties of elliptic curves is horizontal symmetry—reflecting the curve across the x-axis produces a mirror image. Another important characteristic is that any non-vertical straight line drawn through the curve intersects it at no more than three points, typically labelled as (A, B, C) or (F, E, D). From the graph, the security of ECC relies on the complexity of the ECDLP. Specifically, given a point A on the curve and a scalar k, it is computationally infeasible to determinent B = k × A. On an elliptic curve, we can take a point A and “multiply” it by a number k (this is scalar multiplication in ECC, not normal arithmetic multiplication). Multiplying A by k means repeatedly adding A to itself k times, the result is another point B on the curve. This problem is significantly harder than factoring large numbers, which underpins the security of traditional algorithms such as RSA. Due to this computational difficulty, ECC achieves strong security with much smaller key sizes compared to RSA and other public-key cryptosystems, making it particularly well suited for resource-constrained environments such as IoT devices.

Despite the efficiency of ECC for IoT applications, the underlying finite field arithmetic—particularly polynomial multiplication—remains a bottleneck in hardware implementations. Existing architectures either lack reconfigurability, suffer from high area and delay overhead, or are not optimized for recursive computation. Thus, there is a need for a compact, flexible, and low-latency multiplier design that balances performance and resource usage. The scope of this work is limited to hardware implementation of polynomial multiplication in $GF(2^n)$ fields used in ECC, with a specific focus on optimizing logic for FPGA and ASIC platforms. Software-level cryptographic protocols, side-channel attack resistance, and power analysis are considered outside the scope of this thesis.

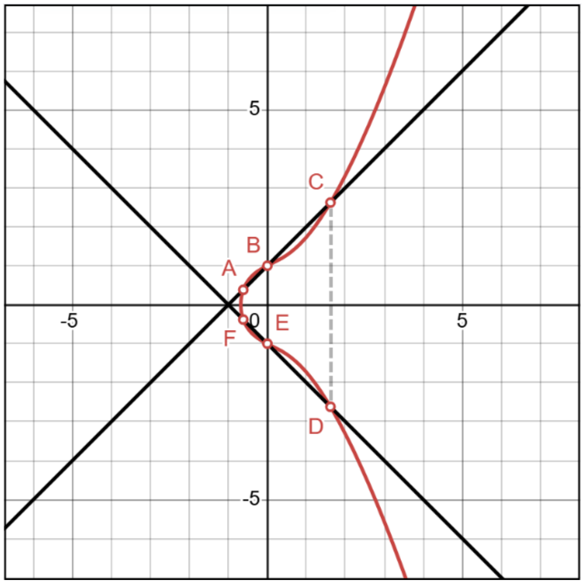

Figure 1.2: Graph of Elliptic Curve Points on a Finite Field

1.1 Key Innovations

This study presents a hardware-efficient polynomial multiplier architecture with two key innovations to overcome these limitations: a reconfigurable polynomial modular multiplier and a reconfigurable karatsuba algorithm optimized using an overlap strategy.

1.1.1 Reconfigurable Polynomial Multiplier Algorithm

This architecture enables effective implementation without sacrificing performance. A key advantage of the proposed design is its reconfigurability. The modulus polynomial is treated as a flexible input parameter, enabling the multiplier to support a wide range of modulus polynomials without any need for hardware changes. This adaptability makes the architecture particularly valuable for applications that require dynamic or customizable configurations. The architecture reduces the number of I/O pins needed while preserving parallel output capabilities by serializing a single multiplicand.

1.1.2 Reconfigurable Karatsuba Algorithm

The optimized version of the reconfigurable algorithm incorporates an overlap-free technique that effectively reduces the delay by half. This method employs a recursive $n/2$ strategy, dividing a $n-bit$ multiplication into two concurrent $n/2-bit$ operations, thereby significantly minimizing latency. For example, in the case of an $8-bit$ multiplier, the critical path latency is reduced from $n/2$ clock cycles by executing two $4-bit$ operations in parallel. The hierarchical reuse of processing elements not only enhances performance by twofold but also maintains the simplicity of the hardware architecture.

In this thesis, Chapter 2 provides a comprehensive overview of finite field arithmetic, mathematical foundations, and key concepts in ECC, particularly in the context of hardware implementations. Chapter 3 includes related work, reviews existing approaches to polynomial multiplier designs, and highlights the limitations of traditional architectures in terms of performance tradeoffs and flexibility. Chapter 4 presents the proposed architectures Reconfigurable and the Karatsuba reconfigurable algorithm, and discusses the RTL schematics, theoretical analysis, and gate-level substitution strategy to further reduce delay. Chapter 5 presents a comprehensive comparison between the proposed design and existing multipliers, evaluating theoretical metrics such as area complexity, time complexity, transistor count, and delay performance. Finally, Chapter 6 concludes the thesis by highlighting the key findings and outlining future research directions, including scaling the architecture and extending its applicability to additional cryptographic primitives.

Chapter 2

Preliminaries

This chapter establishes the theoretical framework required for the reader to comprehend the essential concepts of reconfigurable and compact modular polynomial multipliers in Galois Fields for IoT security. This thesis incorporates the notation and definitions utilized throughout the thesis.

2.1 Polynomial Basis Representation and Multiplication in GF(2^n)

This section presents the representation and multiplication of field elements in the finite field GF(2^m) using the PB. The field GF(2^n) is an extension of the binary field GF(2), which consists of only two elements: $\{0, 1\}$. Within this extension, field elements can be represented as n-dimensional vectors on GF(2). In GF(2), arithmetic operations are defined as follows: Addition and subtraction are performed using the logical XOR (exclusive-OR) operation, while, multiplication is performed using the logical AND operation. However, multiplication in GF(2^n) is carried out by multiplying two polynomials and then reducing the result modulo an irreducible polynomial to ensure that the result lies within the field.

Definition 1: Irreducible Polynomial over GF(2)

For every $m \geq 1$, there exists at least one irreducible polynomial of degree m. Trinomials ($x^m + x^k + 1$) and pentanomials ($x^m + x^{k_1} + x^{k_2} + x^{k_3} + 1$) are preferred in cryptography due to their sparse structure, which minimizes XOR operations during modular reduction (Lidl and Niederreiter, 1994). To define the finite field GF(2^m), we begin with a special polynomial known as the irreducible polynomial, denoted here as $P(z)$.

$$P(z) = z^m + p_{m-1}z^{m-1} + \cdots + p_1 z + p_0, \tag{2.1}$$

where each coefficient p_i belongs to GF(2), i.e., $p_i \in \{0,1\}$. This polynomial cannot be factored into polynomials of lower degree over GF(2), which makes it irreducible. It acts as the modulus for the field, ensuring that polynomial operations wrap around and remain within GF(2^n).

Definition 2: Polynomial Basis and Field Generator

Let θ be a root of the irreducible polynomial $P(z)$ in some extension field, (Lidl and Niederreiter, 1994). Then the powers of θ form a set called the polynomial basis for GF(2^n):

$$\mathcal{B} = \{1, \theta, \theta^2, \ldots, \theta^{n-1}\}. \tag{2.2}$$

Every element in GF(2^m) can be expressed uniquely as a linear combination of these basis elements, using coefficients from GF(2). The value θ essentially serves as the building block or generator of the field under the polynomial basis.

Definition 3: Element Representation in GF(2^m)

Using the polynomial basis $\mathcal{B}$, any element $E(z)$ in GF(2^n) can be written as a binary polynomial of degree at most $n-1$:

$$E(z) = e_0 + e_1 z + e_2 z^2 + \cdots + e_{n-1} z^{n-1}, \tag{2.3}$$

where each coefficient e_i is either 0 or 1 (i.e., $e_i \in \mathrm{GF}(2)$). These polynomials represent the actual data elements in the field, and are manipulated using polynomial arithmetic.

Definition 4: Binary Vector Representation

The polynomial $E(z)$ can also be directly mapped to a binary vector:

$$E = (e_{n-1}, e_{n-2}, \ldots, e_1, e_0), \tag{2.4}$$

Where e_{n-1} is the most significant bit (MSB) and e_0 is the least significant bit (LSB). This binary representation is commonly used in digital systems and hardware implementations because of its simplicity. This provides the framework for implementing efficient arithmetic operations in $GF(2^n)$ and forms the basis of the sequential polynomial basis multiplier described in the following section.

2.2 Hardware Implementation of GF(2)

The simplest finite field, GF(2), consists of just two elements: 0 and 1. Despite its simplicity, GF(2) forms the foundation for a wide range of digital systems, error-correcting codes (Peterson and Weldon, 1972), and cryptographic algorithms. In hardware, the operations of GF(2) are particularly attractive because they map directly onto basic digital logic gates, enabling highly efficient circuit implementations. The simplest finite field is GF(2). Its arithmetic operations are easily summarized:

Addition / Subtraction

$+/-$	0	1
0	0	1
1	1	0

Multiplication

$\times$	0	1
0	0	0
1	0	1

Hardware Implementation

Addition: $a \oplus b$

Subtraction: $a \oplus b$

Multiplication: $a \wedge b$

2.3 Modular Arithmetic Operations

The finite field GF(2^3) is constructed using the irreducible polynomial $f(x) = x^3 + x + 1$ over GF(2). Elements of this field are binary polynomials of degree at most 2, represented as 3-bit strings (a_2, a_1, a_0), where $a_i \in \{0, 1\}$ correspond to coefficients of x^2, x, and the constant term, respectively. For example: 001 $\leftrightarrow$ 1, 010 $\leftrightarrow x$, 100 $\leftrightarrow x^2$, 111 $\leftrightarrow x^2+x+1$, and so on, enumerates products of all element pairs in GF(2^3). For $k \geq 3$, where is the degree:

$$x^3 \equiv x+1 \pmod{f(x)} \tag{2.5}$$

$$x^k = x^{k-3} \cdot (x+1), k \geq 3 \tag{2.6}$$

Table 2.1: Table for $a(x) \times b(x) \pmod{f(x)}$ in $GF(2^3)$

×	000	001	010	011	100	101	110	111
000	0	0	0	0	0	0	0	0
001	0	1	2	3	4	5	6	7
010	0	2	4	6	3	1	7	5
011	0	3	6	5	7	4	1	2
100	0	4	3	7	6	2	5	1
101	0	5	1	4	2	7	3	6
110	0	6	7	1	5	3	2	4
111	0	7	5	2	1	6	4	3

For example, $101 \times 100 \equiv (x^2+1) \times (x^2) \pmod{f(x)} = x^4+x^2 \equiv x^2+x+x^2 = x$, which is 010 in binary representation (decimal number 2 in the table). Cryptographic primitives like block ciphers (like AES) (Lai, 1992) and error-correcting codes (like Reed-Solomon) (Peterson and Weldon, 1972) are based on finite field arithmetic over $GF(2^n)$, such as the $GF(2^3)$ operations discussed above. Effective hardware and software implementation is made possible by the simplified representation of field operations provided by table such as Table 2.1.

2.4 Conventional Algorithm

The conventional algorithm (CA) polynomial multiplication is the most straightforward method used for multiplying two polynomials (Knuth, 1997). In this approach, each coefficient of one polynomial is multiplied by every coefficient of the other polynomial, and the partial products are summed accordingly. This section provides an overview of CA used for

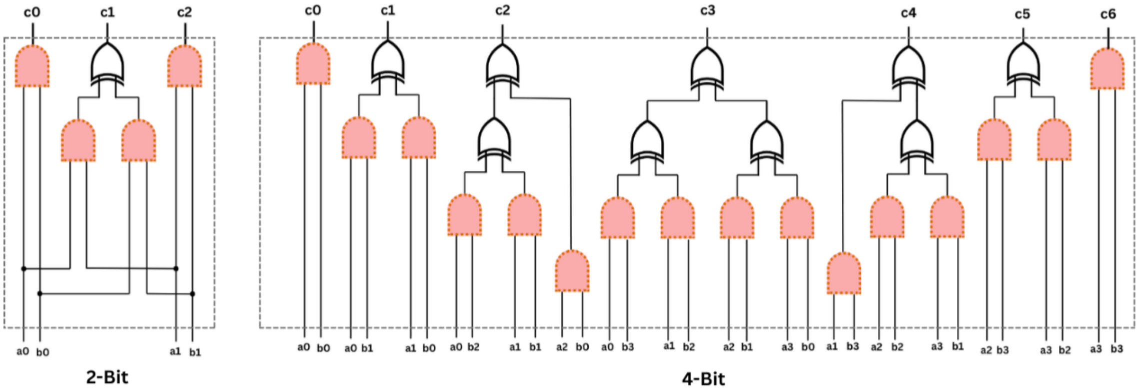

Figure 2.1: DFG for hardware implementation of (a) 2-bit and (b) 4-bit conventional binary polynomial multiplier, modeled after Fig. 1 from (Heidarpur and Mirhassani, 2021).

multiplying binary polynomial multiplications. To begin, consider two binary polynomials of degree one given by:

$$A(x) = a_1x + a_0, \quad B(x) = b_1x + b_0$$

where $a_1, a_0, b_1, b_0 \in \mathrm{GF}(2)$. Using the Conventional Algorithm (CA), the product $C(x) = A(x) \cdot B(x)$ is obtained by multiplying each term of $A(x)$ with each term of $B(x)$ and summing the results modulo 2:

$$\begin{aligned} C(x) &= (a_1x + a_0)(b_1x + b_0) \\ &= a_1b_1x^2 + a_1b_0x + a_0b_1x + a_0b_0 \\ &= a_1b_1x^2 + (a_1b_0 \oplus a_0b_1)x + a_0b_0 \end{aligned}$$

where $\oplus$ denotes addition in GF(2), implemented as an XOR gate, and multiplication in GF(2) corresponds to the AND operation. As an example, a 2-bit binary polynomial multiplier can be justified using four AND gates and one XOR gate. In general, to multiply two binary polynomials of n bits using the CA, the total number of logic gates required is given by: $CA_{XOR}(n) = (n-1)^2$, and $CA_{AND}(n) = n^2$. The propagation delay of the CA for polynomial multiplication in $\mathrm{GF}(2^m)$ can be analyzed by examining the critical path through the combinational logic. The delay profile of the CA is more complicated for the broader situation of n-bit multiplication. Typically, the delays for these gates are denoted as T_A for a 2-input AND gate and T_X for a 2-input XOR gate. For the subsequent summation, where the number of XOR stages scales with a logarithmic depth adder tree structure is required as shown in Figure 2.1. The logarithmic term $\lceil \log_2 n \rceil$ arises because the n partial products, generated in parallel using AND gates (delay T_A), are summed using a *binary tree* of XOR gates. In GF(2), addition is performed as a bitwise XOR, and at each level of the tree the number of terms is halved until only one result remains. This requires $\lceil \log_2 n \rceil$ levels, with each level adding a delay of T_X along the critical path. Therefore, the overall propagation delay is:

$$T_\mathrm{CA}(n) = T_\mathrm{A} + \lceil \log_2 n \rceil \cdot T_\mathrm{X}$$

2.5 Karatsuba Algorithm

The Karatsuba Algorithm (KA) represents a significant advancement over conventional polynomial multiplication techniques by employing a divide-and-conquer approach that reduces computational complexity, (Karatsuba, 1963). As illustrated in Figure 2.2 – 2.3, the algorithm decomposes the multiplication problem into smaller subproblems, particularly effective for binary polynomial multiplication in $GF(2^n)$.

For two n-term polynomials, $A(x) = \sum_{i=0}^{n-1} a_i x^i$ and $B(x) = \sum_{i=0}^{n-1} b_i x^i$, where $n = 2m$, the KA begins by partitioning each polynomial into higher and lower segments: $A(x) = A_H x^n + A_L$, and $B(x) = B_H x^n + B_L$. This partitioning strategy enables the computation of the product through three carefully designed sub-multiplications rather than the four required by conventional methods. The fundamental KA identity expresses the product as:

$$\begin{aligned} C(x) = A(x).B(x) &= \left(A_H x^n + A_L\right)\left(B_H x^n + B_L\right) \\ &= \underbrace{A_H B_H}_{P_2} x^{2m} + \underbrace{\left((A_H + A_L)(B_H + B_L)\right)}_{\text{cross term}} x^m + \underbrace{A_L B_L}_{P_0}. \end{aligned} \tag{2.7}$$

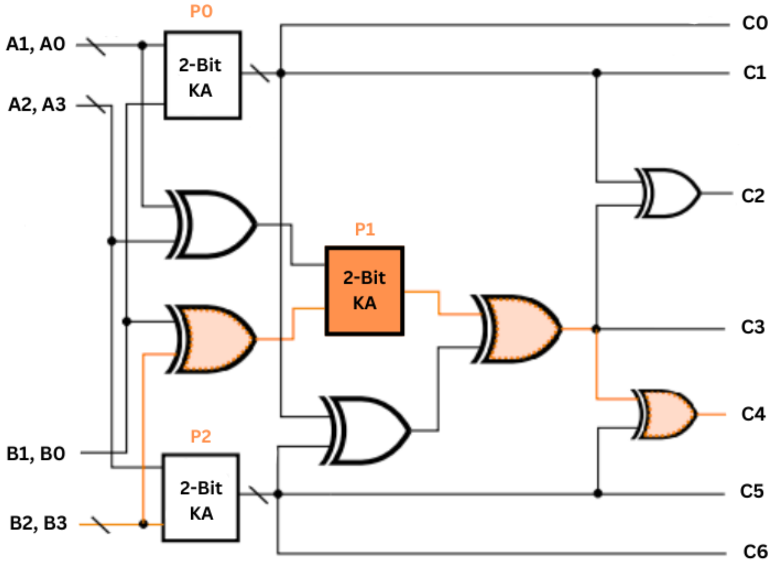

Figure 2.2: Three submultipliers of OFKA are used in the DFG hardware implementation of a 4-bit Karatsuba binary polynomial multiplier, modeled after Fig. 2 in (Heidarpur and Mirhassani, 2021).

where the three critical subproducts are:

$$P_2 = A_H B_H \tag{2.8}$$

$$P_1 = (A_H + A_L)(B_H + B_L) \tag{2.9}$$

$$P_0 = A_L B_L \tag{2.10}$$

Thus, if we could obtain the cross term P_1 directly, three m-bit polynomial multiplications would suffice: one for $A_H B_H$, one for $A_L B_L$, and one for the cross term. The computational advantage of KA becomes evident in its complexity analysis. The algorithm achieves a sub-quadratic growth rate, requiring only $n^{\log_2 3}$ AND gates compared to the n^2 gates needed by the CA. The XOR gate count follows a more complex relationship given by $6n^{\log_2 3} - 8n + 2$, reflecting the overhead from polynomial additions and subtractions. The propagation delay, characterized by $T_{KA}(n) = T_A + (3\log_2 n - 1)T_X$, demonstrates a logarithmic scaling with operand size, though with a larger constant factor than the conventional approach due to the additional recombination steps.

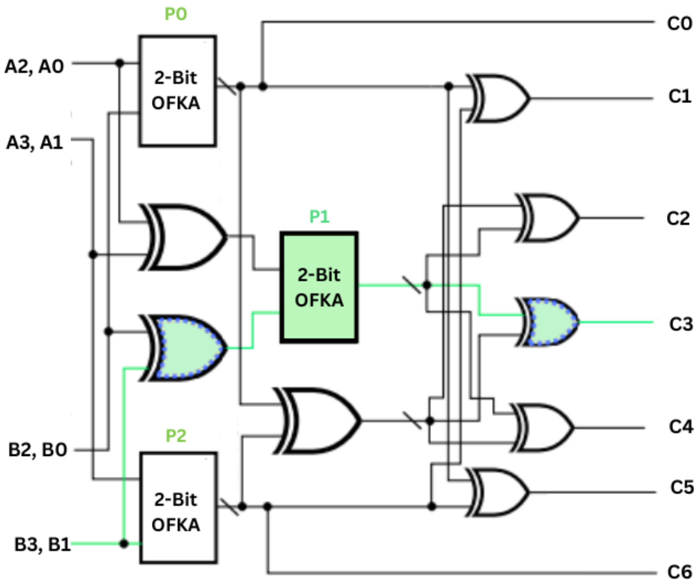

Figure 2.3: Three submultipliers of OFKA are used in the DFG hardware implementation of a 4-bit overlap-free Karatsuba binary polynomial multiplier, modeled after Fig. 3 in (Heidarpur and Mirhassani, 2021).

2.6 Overlap-Free Karatsuba Algorithm

The Overlap-Free Karatsuba Algorithm (OKA) is an optimized variant of the classical Karatsuba multiplication method, specifically designed to minimize critical path delays in hardware implementations. In the standard Karatsuba recombination (2.7), *overlap* refers to the event that, after applying the shifts by x^n or x^{2m}, coefficients from different subproducts contribute to the same power of x in $C(x)$ and therefore must be *summed* (XORed) together. Overlap-free Karatsuba arranges the subproducts so these collisions do not occur, reducing recombination XOR depth and improving latency while preserving the three-submultiplier structure.

Unlike the standard Karatsuba approach, which partitions polynomials into higher and lower segments, OKA employs an innovative decomposition strategy based on grouping odd and even coefficients. For two polynomials $A(x)$ and $B(x)$ in $GF(2^n)$ with $n = 2m$

where $y = x^2$, OKA expresses them as:

$$A(y) = \sum_{i=0}^{m-1} a_{2i} y^i + x \sum_{i=0}^{m-1} a_{2i+1} y^i = A_e(y) + x A_o(y)$$

$$B(y) = \sum_{i=0}^{m-1} b_{2i} y^i + x \sum_{i=0}^{m-1} b_{2i+1} y^i = B_e(y) + x B_o(y)$$

Three sub-multipliers are utilized, as in the conventional Karatsuba method. In OKA, the product $P_0(y) = A_e(y)B_e(y)$ contains only even powers of x, where the three critical subproducts are as follows:

$$P_2(y) = A_o(y)B_o(y) \tag{2.11}$$

$$P_1(y) = (A_e(y) + A_o(y))(B_e(y) + B_o(y)) \tag{2.12}$$

$$P_0(y) = A_e(y)B_e(y) \tag{2.13}$$

This decomposition reduces overlap and optimizes the critical path in hardware implementations. This separation eliminates any overlap between these terms, allowing the removal of one XOR gate from the critical path compared to the standard Karatsuba implementation. The product becomes:

$$A(x)B(x) = P_2(y)y + [P_1(y) - P_2(y) - P_0(y)]x + P_0(y) \tag{2.14}$$

An important characteristic of OKA is that the term $P_0(y) + yP_2(y)$ contains only odd exponents (x^{2n+1}), whereas $P_1(y)$ is composed entirely of even exponents (x^{2n}). Because there is no overlap between these two sets of terms, one XOR operation can be eliminated from the critical path compared to the conventional Karatsuba multiplier. This overlap-free structure results in a more efficient hardware implementation. For instance, in a 4-bit OKA multiplier, the critical path is shortened by one XOR gate delay relative to its Karatsuba counterpart 2.1. While the DFG of a first-order OKA multiplier closely mirrors that of

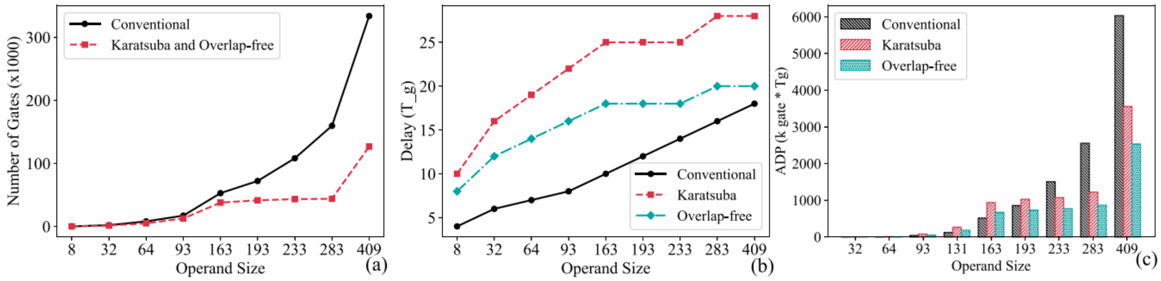

Figure 2.4: Comparison of hardware implementations for Conventional, Karatsuba , and Overlap-Free Karatsuba algorithms in (Heidarpur and Mirhassani, 2021) © IEEE.

the original Karatsuba, this key difference in the recombination stage delivers measurable performance gains.

The computational complexity of OKA can be analyzed in terms of logic gate usage and propagation delay. In hardware-oriented implementations, the two most relevant logic components are XOR and AND gates. The XOR gate complexity of OKA for an n-bit operand is given by $OKA_{XOR}(n) = 6n^{\log_2(3)} - 8n + 2$. and $OKA_{AND}(n) = n^{\log_2(3)}$. The overhead counts for XOR gate: forming $A_e{+}A_o$ and $B_e{+}B_o$: n XORs, combining $P_1 \oplus P_2 \oplus P_0$: $(n-1)+(n-1) = 2n-2$ XORs, and alignment/placement without collisions (overlap-free): $n-2$ XORs. Thus the per-level XOR overhead is

$$L(n) = n + (2n-2) + (n-2) = 4n-4.$$

Letting (n) denote total XORs,

$$(n) = 3\left(\frac{n}{2}\right) + (4n-4), \qquad (1) = 0. \tag{2.15}$$

This expression indicates that the number of XOR gates grows asymptotically with $n\log_2(3)$, which is approximately $1.585n$, but the constant terms demonstrate a notable reduction in gate usage when compared to the conventional karatsuba algorithm. The optimization comes from eliminating overlapping XOR combinations that would otherwise be needed to merge intermediate results. The number of AND gates required by the OKA algorithm remains the same as in the traditional Karatsuba method: $OKA_{AND}(n) = n^{\log_2(3)}$. This is because the core multiplication operations still involve the same number of multiplicative terms; the reduction in complexity is purely due to optimized recombination of the partial results using XOR operations. In terms of delay, assuming that the propagation delay of an AND gate is T_a and the delay of an XOR gate is T_x, the total delay for OKA is modeled as:

$$T_{OKA}(n) = T_A + (2\log_2(n) - 1)T_X \tag{2.16}$$

This reflects a key advantage over Karatsuba's delay model:

$$T_{KA}(n) = T_A + (3\log_2(n) - 1)T_X \tag{2.17}$$

Let $n = 2^k$, k times:

$$T(2^k) = 3k\,T_X + T(1) = 3k\,T_X + T_A = T_A + \big(3\log_2 n\big)T_X.$$

Accounting that the very last XOR layer counted above is *absent* at the base (where there is no recombination, just the terminal AND), we subtract one T_X:

$$\boxed{T(n) = T_A + \big(3\log_2 n - 1\big)T_X}.$$

Analogously for OKA:

$$T(2^k) = 2k\,T_X + T(1) = 2k\,T_X + T_A \;\Rightarrow\; \boxed{T(n) = T_A + \big(2\log_2 n - 1\big)T_X}.$$

The OKA represents a significant advancement in efficient binary polynomial multiplication. By eliminating overlapping terms in the recombination phase of the recursive multiplication process, OKA reduces the number of XOR operations while maintaining the same number of AND operations as classical karatsuba. Figure 2.4 it can be noticed that as the operand size increases, the gate count for the conventional approach grows significantly faster compared to the karatsuba and OKA methods. Although the karatsuba algorithm exhibits higher latency growth than both the conventional and OKA methods, its delay becomes increasingly comparable to the other two as operand size grows. This confirms that OKA offers reduced delay compared to karatsuba by minimizing overlap and thereby optimizing the critical path. Thus, a total of three XOR gate delays, i.e., $3T_X$, are required besides the cost of recursively computing the three partial products.

Chapter 3

Background

This chapter provides the theoretical foundations required to understand the cryptographic techniques and hardware architectures discussed in this thesis. It begins with an overview of cryptography and its importance in securing data over insecure communication channels. The chapter then introduces fundamental concepts of finite fields, polynomial arithmetic, and finite-field multiplication algorithms, along with their relevance to elliptic curve cryptography (ECC) for resource-constrained IoT devices. Different multiplication algorithms, hardware architectures, and implementation techniques are outlined, highlighting their performance trade-offs. The discussion sets the stage for the proposed design and implementation in later chapters.

Cryptography ensures data security by protecting information from unauthorized access or leaks during transmission over insecure channels. Heidarpur and Mirhassani (Heidarpur and Mirhassani, 2021) introduced a finite-field multiplier optimized for FPGA platforms, achieving a 25% better area-delay product (ADP) than cellular automata (CA), 30% over Karatsuba, and 25% over overlap-free Karatsuba (OKA). The intuition behind the focusing ADP is the combined efficiency metric that balances two important (and often competing) goals in hardware design: Area shows how much hardware resource the circuit consumes on FPGA. Delay represents how long it takes for the circuit to produce an output (the critical-path latency). Thirumoorthi and Madhan (Thirumoorthi et al., 2023) developed a hybrid design combining low-time CA and low-space Karatsuba techniques, improving ADP by 15% over traditional Karatsuba-Ofman multipliers (KOM) while reducing delay by 25%,

Table 3.1: Comprehensive Overview of Finite Field Multiplication Techniques

Category	Subcategory	Characteristics	Performance Tradeoffs	Ref.
Algorithm	Karatsuba-Ofman	Divide-and-conquer approach with complexity: $O(n^{1.585})$	40% faster than conventional approach but can cause recursive overhead	(Karatsuba, 1963)
	Montgomery	Division-free modular reduction and residue number system	Efficient for RSA/ECC but can lead to conversion overhead	(Morales-Sandoval et al., 2011)
	Cantor	Polynomial interpolation leads to optimal for extension fields	Pairing-friendly with complex control logic	(Cantor, 1989)
	FFT-based	Frequency domain conversion, which requires root of unity	$O(nlogn)$ speed but complex and large prime requirements	(Von zur Gathen and Gerhard, 1996)
Architecture	Systolic	Parallel processing units	High speed with high complexity	(Meher, 2008) (Guo and Wang, 1998)
	Non-Systolic	Single processing unit	Low complexity with slow speed	(Meher, 2009) (Wu and Hasan, 1998)
	Semi-Systolic	Hybrid structure	Moderate speed with moderate complexity	(Tsai and Wang, 2000) (Fournaris and Koufopavlou, 2008b)

although with an increase in resource usage 11%. Meher's (Meher, 2008) proposed a multiplication method for $GF(2^n)$ using irreducible AOP, which simplifies modular reduction via cyclic-left-shift operations, minimizing redundant logic. In contrast, Pillutla and Boppana (Pillutla and Boppana, 2019) designed a serial digit multiplier tailored for trinomials, allowing simultaneous input processing digit by digit to increase throughput.

3.1 Finite Field Multiplication Algorithms & Architectures

Table 3.1 outlines many methodologies employed for finite field multiplication, classified into principal categories: Algorithms and Architectures. Each category breaks down key subcategories, including their essential characteristics, performance compromises, and corresponding references. The Algorithms category covers computational approaches such as Karatsuba-Ofman, Montgomery, Mastrovito, Cantor, and FFT-based methods, highlighting their complexity, suitability for certain cryptographic schemes (e.g., RSA/ECC), and their efficiency in specific contexts. The Architecture category differs between processing unit topologies (Systolic, Non-Systolic, and Semi-Systolic), focusing on trade-offs between performance and complexity. Finally, Table 3.2 illustrates the different input/output fashions (Bit-Serial, Bit-Parallel, and Digit-Serial), considering the balance between area needs and processing speed which is further discussed below.

3.2 Hardware Implementation Techniques

A Bit-serial architecture sequentially input/output one bit of the operand at a time (per clock cycle). Due to its high area efficiency and low hardware resource requirements, this method is ideal for low-cost or low-power devices like RFID tags or embedded systems with tight space constraints. This advantage, however, comes at the expense of performance because bit-serial multiplication requires a number of clock cycles, which is equivalent to the operand size. For instance, it would take eight cycles to multiply two 8-bit values in $GF(2^8)$, with each cycle handling a single-bit partial product. Bit-parallel architecture, on the other hand, processes every bit concurrently in a single cycle to accomplish multiplication. Since it uses full parallelism, this approach is the quickest of the three, but it comes at a far larger cost in terms of size and hardware complexity. Full-width multiplication logic is implemented using bit-parallel multipliers, which enable nearly instantaneous computation of the result. High-throughput cryptographic processors and applications requiring the highest speed, such as FPGA-based high-speed networking devices, are ideal candidates for such architectures.

Between these two extremes lies the digit-serial architecture, which processes many bits, or digits, every clock cycle. Depending on the design, this can be 2, 4, or more bits. By using less space than bit-parallel implementations and providing faster speed than bit-serial solutions, this architecture offers a balanced trade-off. It works especially well for systems that require a trade-off between hardware resources and performance, like embedded cryptography applications with modest space and performance needs. In contrast to bit-parallel designs, a digit-serial multiplier with a digit size of 2 in $GF(2^8)$ would accomplish a multiplication in four cycles, increasing speed while preserving a smaller footprint. In conclusion, bit-serial designs focus minimal area at the sacrifice of speed, bit-parallel architectures emphasize maximum speed with substantial hardware consumption, whereas digit-serial systems provide a pragmatic compromise, balancing resource utilization and performance. The selection among these is primarily contingent upon the application's limitations and performance objectives.

3.3 Complexity Analysis of Different Multiplier Types

Performance in GF(2^n) multipliers is commonly evaluated using two main criteria: space complexity and time complexity. Space complexity refers to the total number of 2-input XOR and AND gates required for the implementation, while time complexity refers to the longest delay a signal experiences through these XOR and AND gates.

A Bit-parallel multipliers for GF(2^n) can be broadly categorized into three groups based on their asymptotic space complexity: quadratic, subquadratic, and hybrid multipliers (Grabbe et al., 2003) . Quadratic multipliers are the most straightforward form, based on conventional algorithms, further discussed in Section 2.4. In these designs, the mul-

Table 3.2: Overview of Hardware Architecture Finite Field Multiplication Techniques

Category	**Subcategory**	**Characteristics**	**Performance Tradeoffs**	**Ref.**
	Bit-Serial	Sequential bit processing	Small area with slow speed	(Fournaris and Koufopavlou, 2008a), (Kitsos et al., 2003)
Hardware Architecture	Bit-Parallel	Full-bit parallelism	Fast speed with large area	(Reyhani-Masoleh and Hasan, 2002), (Imaña et al., 2006), (Meher, 2009)
	Digit-Serial	Digit-level processing	Balanced speed and moderate area	(Meher, 2009), (Meher, 2008), (Guo and Wang, 1998)

tiplication of two n-bit operands is done by generating all possible partial products using AND gates and summing them with XOR gates. This direct approach results in a time and space complexity of $O(n^2)$, which makes it simple and practical for small field sizes, typically up to 64 bits. The reduction step, where the product is reduced modulo an irreducible polynomial, is also hard-wired into the circuit, often using a matrix-based approach like the Mastrovito multiplier (Halbutogullari and Koç, 2000). Beth and Gollmann (Beth and Gollmann, 1989) examined methods for implementing public-key algorithms based on modular integer arithmetic based on Bit-Level Serial-In-Parallel-Out. Their work emphasized architectures suitable for VLSI implementations, focusing on optimizing performance and efficiency in hardware designs for cryptographic applications.

To address the inefficiencies of quadratic designs at larger operand sizes, subquadratic multipliers employ more advanced algorithms that reduce the number of partial multiplications needed. The most well-known among these is the Karatsuba-Ofman algorithm, (Karatsuba, 1963) which is explained in Section 2.5, which recursively splits operands into halves and performs only three multiplications (instead of four) for each recursive level. This reduces the total complexity of multiplication to approximately $O(n^{1.585})$.

Hybrid multipliers, attempt to combine the strengths of both quadratic and subquadratic methods. These architectures typically apply a few levels of recursive subquadratic multiplication to break down the operands into smaller chunks. Once the size of these sub-operands becomes manageable, typically in the range of 8 to 16 bits, a base-case quadratic multiplier (such as a Mastrovito's structure) is used to complete the multiplication. This hybrid approach allows designers to fine-tune the trade-offs between hardware area, speed, and design complexity.

Quadratic multipliers, extensively studied in the literature, employ various basis representations of elements in $\mathrm{GF}(2^n)$, including polynomial, shifted polynomial, normal, dual, weakly dual, and triangular bases (Lee and Lim, 1999). While quadratic multipliers typically exhibit lower time complexities, subquadratic multipliers excel in having reduced asymptotic space complexity, making them especially suitable for VLSI implementations when dealing with large operand sizes. However, for smaller operand sizes, such as 32 bits, space complexity might not be as critical. In these scenarios, computational speed tends to dominate design considerations. Consequently, a sub-quadratic or hybrid approach is frequently adopted to achieve optimal performance in practical cryptographic processors (Von Zur Gathen and Gerhard, 2003), (von zur Gathen and Shokrollahi, 2005).

The choice of basis for representing elements in $\mathrm{GF}(2^n)$, where two important bases are the *normal basis* and the *dual basis* (Cohen and Niederreiter, 1996; Menezes, 1993). A *normal basis* is formed by taking an element $\alpha \in \mathbb{GF}(2^n)$ and its powers:

$$\{\alpha, \alpha^q, \alpha^{q^2}, \dots, \alpha^{q^{n-1}}\}.$$

This representation makes operations like squaring and exponentiation very efficient, which is helpful for cryptographic applications.

A *dual basis* is another basis related to the original one, which allows easy extraction of coefficients and supports efficient multiplication. It is defined using the field trace function, and is particularly useful for designing fast hardware circuits. Using normal and dual bases can significantly improve the speed and efficiency of cryptographic algorithms, such as those used in elliptic curve cryptography and key exchange protocols.

3.4 Cryptographic Relevance and NIST Recommendations

The NIST plays a critical role in standardizing cryptographic operations, particularly polynomial multiplication over GF(2^n), to ensure both security and computational efficiency. For elliptic curve cryptography, NIST recommends using irreducible trinomials of the form $x^n + x^k + 1$, where the middle term x^k is chosen with the smallest possible degree k. This minimizes computational complexity during field arithmetic. For example, the trinomial $x^{233} + x^{73} + 1$ is standardized for GF(2^{233}), with $k < n/2$ to optimize efficiency (PUB, 2000). Such trinomials are not only prevalent but also account for over 50% of practical implementations (Meher, 2008).

When selecting optimal polynomials, the underlying processor architecture must be considered to maximize hardware performance. This raises questions about the necessity of standardizing specific irreducible polynomials, as alternatives may offer better compatibility with modern computational systems (Scott, 2007). In particular, there are no irreducible trinomials for certain values of n. In such cases, the next best choice is an irreducible pentanomial (a polynomial with five non-zero terms). Despite their slightly higher complexity, pentanomials still enable efficient modular reduction operations, ensuring practicality in cryptographic implementations.

Chapter 4

Architectures

This chapter presents a detailed exploration of multiple modular polynomial multiplier architectures, emphasizing their structural designs, computational strategies, and practical hardware implications. These architectures adopt bit-level serial input techniques to minimize I/O complexity while leveraging parallel computation pathways for output generation. By introducing modular reduction mechanisms and incorporating shift register-based processing elements, these designs achieve compactness without compromising correctness. Furthermore, this chapter explores techniques such as the Karatsuba-Ofman algorithm integrated into a reconfigurable framework. This combination significantly reduces the clock cycle $O(n)$ to approximately $O(n/2)$, allowing high-performance polynomial arithmetic suitable for large field sizes used in modern cryptography. This reduction in number of clock cycles is due to its parallelism nature instead of sequential, resulting in operations being performed simultaneously. Gate-level optimizations, including NAND substitutions, are also introduced to further reduce transistor count, critical path delay, and overall silicon area. The proposed architecture is evaluated through a combination of RTL design, gate-level synthesis, and theoretical analysis using standard logic gate delay models. Two architectures—including a Reconfigurable, and Karatsuba are implemented and compared using performance metrics such as gate count, delay, and area-delay product.

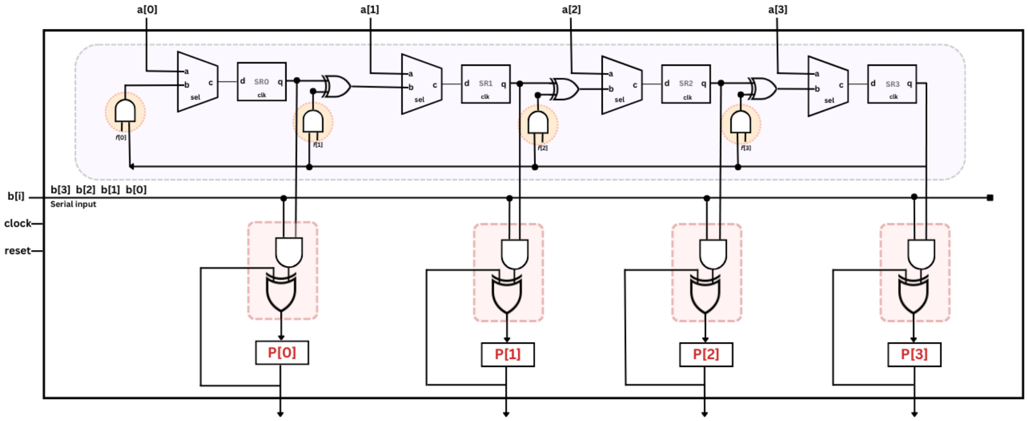

Figure 4.1: Proposed 4-bit Modular Polynomial Multiplier for $a(x).b(x) \bmod f(x)$

4.1 Reconfigurable BL-SIPO

A serial-in parallel-output LSB-first modular polynomial multiplier was proposed in the paper of (Beth and Gollmann, 1989); however, it was limited to fixed modulus polynomials. We extend the multiplier to propose a reconfigurable modular polynomial multiplier in which the modulus polynomial is one of the input parameters and it can be easily changed without the hardware modification. The proposed low-cost and reconfigurable modular polynomial multiplier is optimized for hardware efficiency in terms of area and flexibility, making it particularly suitable for resource-constrained environments, such as security services in IoT applications. The multiplier consists of n processing elements (PEs), each equipped with a 2-input AND gate and a 2-input XOR gate. Additionally, the design includes $2n$ registers, n XOR gates, n AND gates, and n 2-to-1 multiplexers. To minimize I/O overhead, one of the input polynomials is fed serially into the architecture. After n clock cycles, the modular product is produced in parallel. The modulus polynomial is provided as an input parameter, allowing for easy reconfiguration without any hardware modification.

4.1.1 Architecture

The proposed modular polynomial multiplier is illustrated in Figure 4.1 (for $n = 4$). The partial product is stored in the registers $p[0]$ to $p[3]$, with the coefficients of $b(x)$ serially inputting the least significant bit $b[0]$, while the coefficients of $b(x)$ and $f(x)$ are loaded in parallel. Multiplexers select between their two inputs based on a clock counter signal. During the first clock cycle, the coefficients of $a(x)$ (denoted $a_{n-1:0}$) are loaded into the shift registers SR, and the output of the AND gate is selected, resulting in SR $= a(x)$. In subsequent clock cycles, the shift registers are iteratively updated as SR $\leftarrow a(x) \cdot x \mod f(x)$, where modular reduction is computed using the coefficients of $f(x)$. The corresponding DFG for the proposed architecture can be found in Figure 4.2 which shows the data pathway through different modules.

In the depicted diagram, the data flow begins with the input polynomial $a[3:0]$ and the serial input b. The right side of the diagram features a modular reduction pipeline, where each bit of a is processed through a series of multiplexers (MUX0–MUX3) and logic gates (AND, XOR) that implement the modular reduction with respect to the irreducible polynomial $f(x)$. The outputs of these multiplexers are loaded into a set of shift registers (SR0–SR3), which store the intermediate reduced values of the polynomial. These values are then fed into the left side of the diagram, where each shift register output ($SR[0]$ to $SR[3]$) is connected to a corresponding processing unit (PE0–PE3). Each processing unit multiplies its input bit by the serial input b and combines it with the previous accumulated result using XOR logic, effectively performing the modular multiplication. The outputs of the processing units are then stored in another set of shift registers (R0–R3), which accumulate the final product bits $p[3:0]$.

Throughout this process, the control logic (counter and select signal) orchestrates the timing and selection of data paths, ensuring that the correct values are processed and stored at each clock cycle. This pipelined structure allows for efficient, sequential modular multiplication and reduction, with data flowing from the input registers, through the reduction and multiplication stages, and finally to the output registers. The processing element (PE) calculates $p[i] = p[i] + a[i] \cdot x^i \cdot b(x) \mod f(x)$ at each clock cycle, and the final result $P(x) = A(x)B(x) \mod f(x)$ is obtained after n clock cycles. A similar algorithm was proposed in (Fournaris and Koufopavlou, 2008b); however, it was limited to fixed modulus

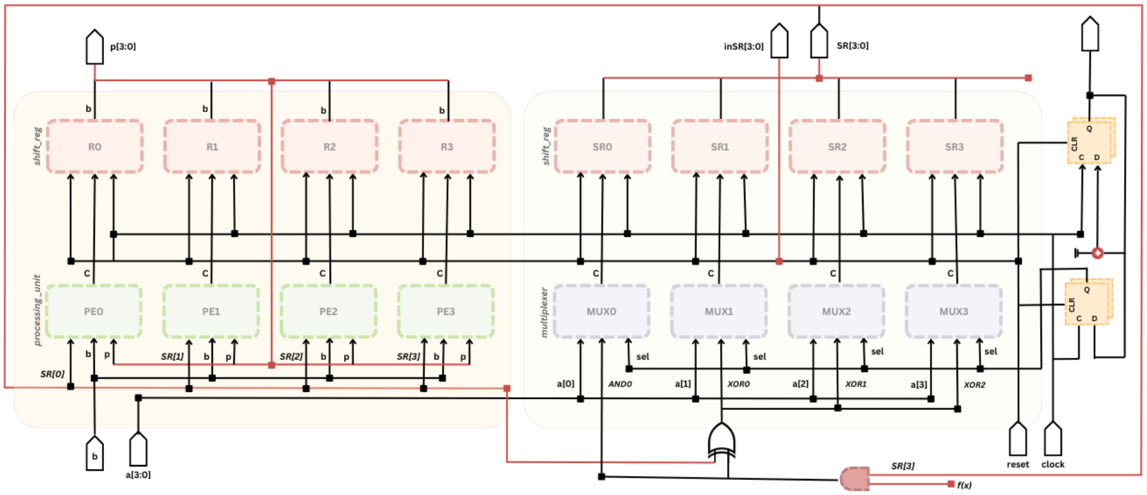

Figure 4.2: DFG of Reconfigurable Modular Polynomial Multiplier

polynomials and lacked critical implementation details such as shift register initialization and multiplexer control logic. The algorithm 1 describes the detailed operations for the proposed reconfigurable modular polynomial multiplier.

4.1.2 Summary

The Reconfigurable BL-SIPO modular polynomial multiplier presents a notable enhancement over the conventional SIPO architecture by introducing runtime reconfigurability of the modulus polynomial $f(x)$, thereby improving flexibility in cryptographic applications. Unlike the fixed-modulus SIPO, which is limited to specific irreducible polynomials and lacks adaptability, the reconfigurable variant allows dynamic selection of modulus without altering the hardware, supporting a broader range of field operations in $GF(2^n)$. Furthermore, by incorporating parallel output and modular control via multiplexers and clock-driven shift registers, it maintains hardware efficiency while enabling more generalized modular arithmetic.

However, despite its advantages, this architecture still retains the bit-serial input structure, which imposes a linear latency proportional to the operand size n, making it less suitable for high-throughput or real-time cryptographic operations. Moreover, although the modular reduction logic is adaptable, the architecture processes polynomial multiplica-

Algorithm 1 GF-based Reconfigurable Modular Polynomial Multiplier

```
Input: Polynomial $A(x)$ coefficients $a[0:n-1]$
Polynomial $B(x)$: coefficients $b[0:n-1]$
Modulus $f(x) = x^n + x^m + 1$
Output: Polynomial $P(x)$ coefficients $p[0:n-1] = A(x)B(x) \bmod f(x)$

  Load $a[0:n-1]$ into shift registers $SR[0:n-1]$
  for $(i = 0; i \leq n-1; i++)$ {
    $p[i] \leftarrow b[0] \wedge a[i]$
  }
  for $(j = 1; j \leq n-1; j++)$ {
    $SR[0] \leftarrow SR[n-1] \wedge f[0]$
    for $(i = 1; i \leq n-1; i++)$ {
      $SR[i] \leftarrow SR[i-1] \oplus (SR[n-1] \wedge f[i])$
    }
    Right Shift $SR$
    for $(i = 0; i \leq n-1; i++)$ {
      $p[i] \leftarrow p[i] \oplus (b[j] \wedge SR[i])$
    }
  }
```

tions in full width, which may result in increased time complexity for larger operand sizes. These limitations are addressed in the subsequent algorithm, which introduces overlap-free decomposition of polynomial inputs (based on the Karatsuba-Ofman method) to reduce computational complexity and hardware overhead, enabling faster and more scalable polynomial multiplication while preserving reconfigurability.

4.2 Reconfigurable Karatsuba Modular Polynomial Multiplier

The Reconfigurable Karatsuba multiplier extends traditional BL-SIPO polynomial multiplication technique by integrating the Karatsuba-Ofman algorithm, significantly enhancing performance through complexity reduction. The Reconfigurable Karatsuba architecture can also be known as K-Method. Traditionally, polynomial multipliers operate with quadratic complexity $O(n^2)$, which quickly becomes impractical for larger field sizes required for modern cryptographic systems. In this design, the polynomial $A(x)$ is partitioned into two subsets—odd-indexed coefficients and even-indexed coefficients—represented as

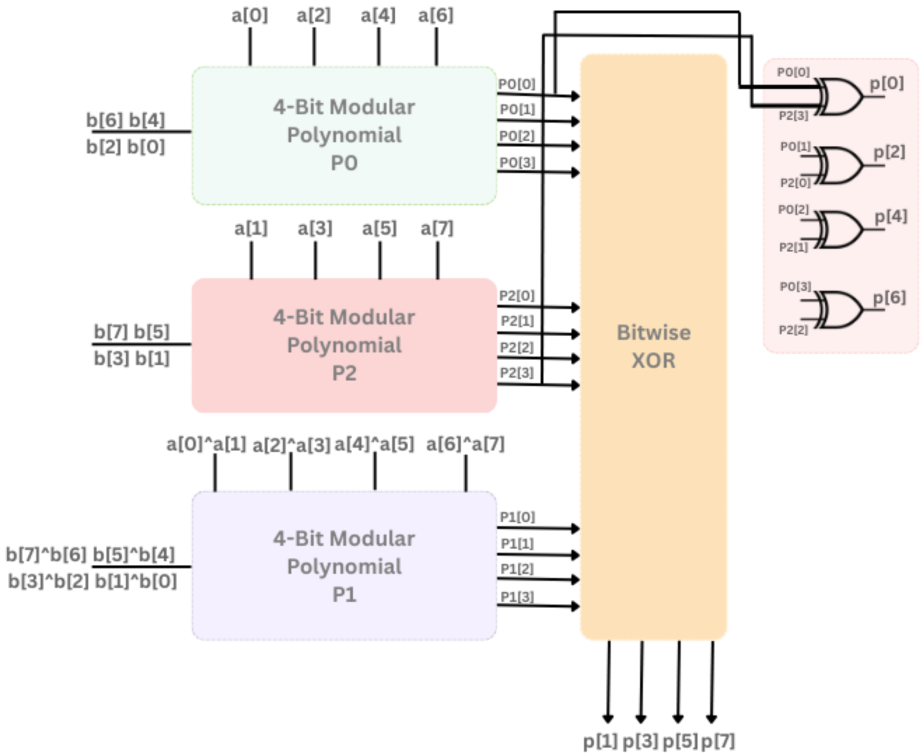

Figure 4.3: Architecture of K-Method Reconfigurable BL-SIPO for 8-bit coefficients.

$A_{even}(x)$ and $A_{odd}(x)$. These subsets undergo independent multiplication with corresponding subsets of polynomials $B(x)$, denoted similarly as $B_{even}(x)$ and $B_{odd}(x)$. Specifically, the product is computed by decomposing it into three sub-multiplications.

4.2.1 Architecture

Figure 4.3 illustrates the proposed Karatsuba polynomial multiplier for n=8 with $f(x) = x^8 + 1$. The top module implements an 8-bit modular polynomial multiplier over the binary field $GF(2^4)$, based on the Karatsuba-Ofman algorithm. The design accepts an 8-bit input polynomial $A(x)$, and two 1-bit serial inputs U_b and V_b corresponding to bits from another polynomial $B(x)$ each consisting of 4-bit. The input polynomial $A(x)$ is first partitioned into two subsets: evenA comprising the even-indexed coefficients $\{A[6], A[4], A[2], A[0]\}$, and oddA comprising the odd-indexed coefficients $\{A[7], A[5], A[3], A[1]\}$. An intermediate sum, Asum, is calculated as the bitwise XOR of evenA and oddA. These components serve as the basis for the three required sub-multiplications in the Karatsuba scheme: $evenA \times evenB$, $oddA \times oddB$, and $Asum \times Bsum$. It specifically highlights the flow of individual coefficient bits through various hardware blocks, including shift registers, processing elements, and multiplexers. This modular structure not only supports reconfigurability for arbitrary modulus polynomials but also efficiently manages the bit-level computation of partial products and their modular reduction. The coefficients of b are passed serially into the system, undergo parallel processing within dedicated units, and are finally recombined to produce the modular polynomial product. Three 4-bit modular multiplier modules are instantiated to compute the partial products where $f(x) = x^4 + 1$ is the irreducible polynomial used for modular reduction.

This architecture introduces a hierarchical and overlap-free computation scheme that effectively reduces latency by decomposing the n-bit modular multiplication into 2 concurrent $\frac{n}{2}$-bit operations. This strategy enables parallel processing and significant delay reduction. For example, for an 8-bit multiplier, three 4-bit multiplications are performed in parallel, reducing the critical path latency from n to $\frac{n}{2}$ clock cycles.

4.2.2 Summary

The Reconfigurable Karatsuba architecture offers a significant improvement over both the conventional BL-SIPO and the second proposed architecture on reconfigurable modular multipliers by combining Karatsuba decomposition with modular reconfigurability. Unlike the standard BL-SIPO design, which suffers from linear latency growth and rigid modular reduction logic, the K-Method structure efficiently decomposes polynomial multiplications into smaller, parallelizable sub-multiplications using even and odd coefficient groupings. This results in a reduced computational complexity of approximately $O(n^{1.585})$, compared to the quadratic complexity $O(n^2)$ of traditional approaches. Additionally, while the standard reconfigurable BL-SIPO supports flexible modulus configuration, it still relies on full-width multipliers and high latency. The K-Method mitigates this limitation by leveraging smaller 4-bit modular polynomial submodules, which recursively handle polynomial products and modular reductions with lower area and clock cycle. As a result, this architecture achieves a superior balance between hardware efficiency, modular adaptability, and computational speed, making it ideal for resource-constrained cryptographic systems that require both flexibility and performance. Table 4.1 presents a comparative analysis of area and time complexity for various modular polynomial multipliers over $GF(2^n)$. The table lists key hardware metrics, including the number of AND, XOR, flip-flop (FF), and multiplexer (Mux) components, along with the latency (in clock cycles) and the critical path delay (based on AND, XOR, and multiplexer delays).

In contrast, designs such as the (Pillutla and Boppana, 2020) and (Imaña, 2010) multipliers have quadratic growth in AND and XOR counts, which increases area consumption significantly for large n. The *Reconfigurable GFPM* offers a balance between flexibility and area usage, although it requires additional multiplexers for configurability.

In terms of time performance, the (Selimis et al., 2009) architecture achieves a low delay of $2T_X + T_{\mathrm{MUX}}$ due to its efficient parallel processing, while maintaining moderate area requirements. The *K-Method* demonstrates competitive latency ($n/2$ cycles) but introduces higher multiplexer usage, potentially impacting routing complexity. The *Reconfigurable GFPM* share base delay of $T_A + T_X + T_{MUX}$, making it attractive for design prioritizing predictable timing. The optimal choice depends on the target application's constraints—low-power IoT devices benefit from small-area designs, while high-throughput cryptographic processors favor low-latency multipliers.

4.3 Performance Comparison

This section examines the performance of the proposed multiplier and compares it with other studies in this field. Theoretical metrics such as area (number of gates, FF, Mux), delay, and ADP (Area x Delay) are used to assess algorithm efficiency. FPGA synthesization is done on various bit sizes from 4-bit to 233-bit multipliers and was created using

Table 4.1: Area and Time Complexity Comparison of $GF(2^n)$ Polynomial Multipliers

Design	**#AND**	**#XOR**	**#FF**	**#Mux**	**Latency**	**Delay**
(El and Reyhani-M., 2015)	$2n-1$	$2n+N_\alpha^2-1$	$3n-2$	0	n	$\max\{T_\alpha^2+T_X,\ T_A+2T_X\}$
(Pillutla and Boppana, 2020)	n^2	n^2-1	$1.5n^2+n$	0	$\frac{n}{2}+2$	T_A+T_X
(Imaña, 2010)	$\frac{n^2+n}{2}$	$\frac{n^2+n}{2}$	$5n-1$	$4n$	$2k_t+1$	$T_A+\lfloor\log_2 n\rfloor T_X+2T_{MUX}$
(Selimis et al., 2009)	$3n$	$2n$	$4n+2$	$2n$	$0.664n$	$2T_X+T_{MUX}$
Reconfigurable GFPM	$2n$	$2n$	$2n$	n	n	$T_A+T_X+T_{MUX}$
K-Method	$3n$	$4n$	$3n$	$1.5n$	$\frac{n}{2}$	$T_A+T_X+T_{MUX}$

Note: k: digit size; t: max exponent of nonzero terms; Latency = number of clock cycles; Area and time complexities are assumed equal for XOR and XNOR gates. PB = Polynomial Basis.

irreducible trinomials to verify this methodology. The work in (El and Reyhani-M., 2015) introduced a most-significant-bit (MSB)-first fully bit level serial-in parallel-out polynomial basis (PB) multiplier based on a definition of the field elements. Pillutla et al. designed a systolic polynomial basis finite field multiplier based on a class of trinomials (Pillutla and Boppana, 2020). A parallel-in and parallel-out sequential polynomial basis multiplier was proposed in (Imaña, 2010). Our proposed architecture for reconfigurable and K-method accepts any modulus polynomial $f(x)$ as input. This eliminates hardware redesign costs for different standards, making it ideal for WSN devices that require cryptographic agility. Design (El and Reyhani-M., 2015) employs $2n-1$ AND gates and a significantly higher number of XOR gates, expressed as $2n+N_\alpha^2-1$, along with $3n-2$ flip-flops. It uses no multiplexers, indicating a purely combinational implementation. However, the delay, given as $\max\{T_\alpha^2+T_X, T_A+2T_X\}$, may become substantial depending on the complexity of the α^2 module. In contrast, the K-method design offers better balance, with lower FF and XOR counts and a predictable delay of $T_A+T_X+T_{\text{MUX}}$. Pillutla et al. utilized n^2 AND gates and n^2-1 XOR gates, with a high storage requirement of $1.5n^2+n$ flip-flops, (Pillutla and Boppana, 2020). Although it has no MUX overhead and maintains a moderate latency of $\frac{n}{2}+2$, its quadratic hardware complexity limits scalability. The proposed Karatsuba architecture improves upon this by reducing all components to linear growth with respect to n, making it more hardware-efficient. Similarly (Imaña, 2010), features a hardware cost of $\frac{n^2+n}{2}$ for both AND and XOR gates and uses $5n-1$ flip-flops and $4n$ multiplexers. Its latency, $2k_t+1$, is the highest among all designs due to shift and addition cycles, and the delay is relatively complex: $T_A + \lfloor \log_2 n \rfloor T_X + 2T_{\text{MUX}}$. While it introduces logarithmic efficiency in delay, its significant resource demands and high latency make it less suitable for efficient implementations compared to the proposed design. The work in (Selimis et al., 2009) is relatively efficient, with $3n$ AND and $2n$ XOR gates, and uses $4n+2$ flip-flops and $2n$ multiplexers. It achieves a low latency of $0.664n$ and a straightforward delay of $2T_X + T_{\text{MUX}}$. The proposed design is comparable in terms of gate usage and delay but uses fewer flip-flops and multiplexers, improving area efficiency. In conclusion, the proposed design achieves a well-balanced trade-off between hardware complexity and performance with $3n$ AND gates, $4n$ XOR gates, $3n$ flip-flops, and $1.5n$ multiplexers, it delivers reduced latency ($\frac{n}{2}$) and consistent delay ($T_A + T_X + T_{\text{MUX}}$).

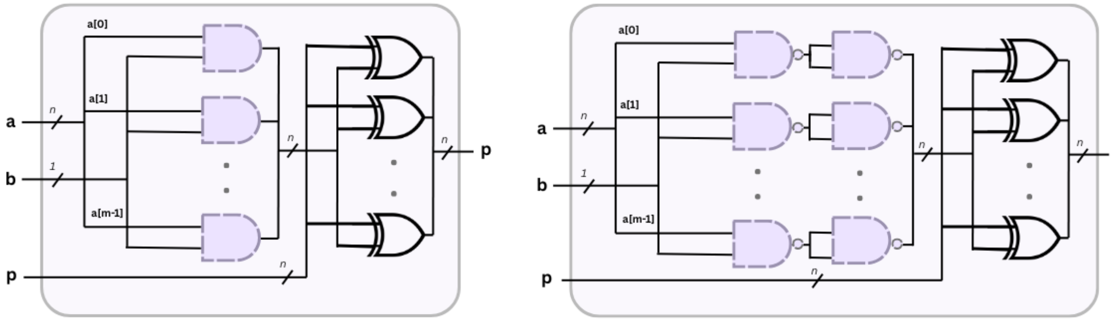

Figure 4.4: AND and NAND gate substitution gate level modelling for module processing elements (PEs)

4.3.1 Gate-level Substitution

The modification of digital algorithms to employ NAND and XNOR gates instead of conventional AND and XOR logic gates achieves significant reductions in area and delay complexities due to inherent efficiencies in transistor-level design and algorithmic restructuring. A primary advantage stems from the transistor efficiency of NAND gates compared to AND gates. A CMOS NAND gate requires only 4 transistors (with two NMOS in series and two PMOS in parallel), whereas an AND gate is typically implemented as a NAND gate followed by an inverter, totaling 6 transistors. Similarly, while XOR and XNOR gates have comparable transistor counts (around 6 transistors in static CMOS), strategic use of XNOR logic in arithmetic operations (e.g., partial product accumulation or carry propagation) can simplify interconnects and reduce the need for additional inverters or logic stages, further optimizing area. Figure 4.4 illustrates the use of NAND gates for gate-level modeling within the processing elements (PEs), demonstrating how conventional logic can be optimized for improved area and delay efficiency.

In summary, the proposed reconfigurable designs outperform existing approaches by a wide margin in terms of area-time efficiency. By leveraging gate-level optimization strategies such as NAND substitution and logic restructuring, these architectures achieve more improvement in ADP, making them highly suitable for high-performance and area-constrained cryptographic hardware implementations.

4.4 Theortical Anaylsis

This thesis employs realistic standardized delay values of logic gates from common ICs produced by STMicroelectronics (e.g., M74HC08, M74HC86). Measure real-world performance accurately: This gives realistic delay estimates for hardware implementations that are feasible. By employing a uniform baseline for every kind of logic gate (AND, XOR, OR, etc.), it guarantees a balanced comparison. The theoretical delay in the critical path at the gate level of the design was calculated using discrete component delays for the 2-input AND gates ($T_{AND} = 6ns$), the XOR gates ($T_{XOR} = 12ns$), and the 2:1 MUXes ($T_{MUX} = 11ns$). To estimate the quantity of CMOS transistors utilized, conventional counts have been employed: 6 transistors for an AND gate, 6 for an XOR gate, 6 for an OR gate, 6 for a 2:1 multiplexer, 8 for SR-latch, and 6 for a NAND gate.

The proposed architectures demonstrate substantial improvements in both performance and hardware efficiency, particularly in terms of transistor count, delay, and the ADP. The *Reconfigurable GFPM* architecture utilizes 466 AND gates, 466 XOR gates, 466 flip-flops, and 233 multiplexers, resulting in a total of 10,718 transistors. It completes computation in 233 clock cycles with a delay of 29 ns per cycle, yielding a total execution time of 6,757 ns. This leads to an ADP of 72.421×10^6. The another proposed architecture, *K-Method* is optimized through a logic structure using 699 AND gates, 932 XOR gates, 699 flipflops, and 350 multiplexers. Despite a higher transistor count of 17,475, it completes computation in just 117 clock cycles with a 29 ns delay, resulting in a total time of 3,393 ns. The ADP for this architecture is 59×10^6, which is the lowest among the proposed designs and when compared with state-of-the-art multiplier designs.

The proposed design for K-method and reconfigurable described in Section 4.1– 4.2 is similarly divided into two main stages: The modular reduction stage, which consists of an AND gate, an XOR gate and a 2:1 multiplexer, resulting in a total gate-level delay of $T_{\text{AND}} + T_{\text{XOR}} + T_{\text{MUX}} = 6 + 12 + 11 = 29\,\text{ns}$; and the product accumulation stage (PE), which includes an AND gate followed by an XOR gate, leading to a delay of $T_{\text{AND}} + T_{\text{XOR}} = 6 + 12 = 18\,\text{ns}$. Since the modular reduction stage involves an additional multiplexer, it defines the critical path of the design. However, by employing gate-level substitution techniques discussed in 4.3.1, specifically replaces AND gates with faster and more efficient NAND gates—the critical path delay can be reduced to 27 ns, offering improved performance with reduced hardware overhead. Table 4.2 presents the overall transistor count and latency of the proposed multiplier compared with existing designs for NIST $GF(2^{233})$. Compared to existing designs to demonstrates substantial improvements in key metrics such as area, delay, and latency, where ADP is cacaluated with base 10^6. In particular, the proposed design of K-method reduces total ADP count by 24.04% compared to El and Reyhani-M. (2015), 97.87% compared to Pillutla and Boppana (2020), 99.06% compared to Imaña (2010), and 36.66% compared to Selimis et al. (2009).

Table 4.2: Analyzing the transistor count and delay of the proposed multiplier and comparing it with related studies for NIST $GF(2^{233})$.

Design	**#AND**	**#XOR**	**#FF**	**#Mux**	**#Clocks**	**Delay**	**Trans.**	**ADP**	**% Reduction**
(El and Reyhani-M., 2015)	465	467	697	0	233	30	11,168	78	24.04
(Pillutla and Boppana, 2020)	54,289	54,288	81,666	0	119	18	1,304,794	2,794	97.87
(Imaña, 2010)	27,261	27,261	1,164	932	149	124	342,036	6,319	99.06
(Selimis et al., 2009)	699	466	934	466	155	35	17,258	93	36.66
Reconfigurable GFPM	466	466	466	233	233	29	10,718	72.42	18.53
K-Method	699	932	699	350	117	29	17,475	59	—

Note: All values are reported for field size $GF(2^{233})$. Delay is per cycle. Time = number of clocks × delay. Area × Time = number of transistors × time.

4.5 Summary

In summary, the proposed modular polynomial multiplier architectures—ranging from the basic BL-SIPO to the reconfigurable and Karatsuba-based designs—demonstrate significant improvements in hardware efficiency, flexibility, and computational speed for cryptographic applications. By leveraging bit-serial input, modular reduction pipelines, and advanced decomposition techniques such as the Karatsuba-Ofman algorithm, these architectures achieve lower area, reduced latency, and improved area-delay product (ADP) compared to conventional designs. Gate-level optimizations, including the substitution of NAND gates, further minimize critical path delay, making the designs highly suitable for resource-constrained environments. Theoretical analysis, supported by standardized gate delay values and transistor counts, confirms that the proposed multipliers outperform existing approaches in both area and time complexity, particularly for large field sizes such as NIST $GF(2^{233})$. These results highlight the practicality and scalability of the proposed architectures for modern cryptographic hardware, where efficiency, adaptability, and performance are paramount.

Chapter 5

Comparison

5.1 Tools and Hardware Setup

This section outlines the hardware platforms and software tools used for the implementation, simulation, and evaluation of the modular polynomial multipliers. The selection of tools was guided by their compatibility with the target devices and their ability to support synthesis and implementation within the available resources.

5.1.1 FPGA Development Platforms

The modular polynomial multipliers were synthesized and tested on the following FPGA development boards:

- **Xilinx Artix-7 (Basys 3 Board)**
- **Xilinx Zynq-7000 SoC (ZedBoard)**
- **Spartan-7 SP701 Evaluation Platform (xc7s100fgga676-2)**
- **Artix-7 AC701 Evaluation Platform**
- **Zynq UltraScale+ ZCU104 Evaluation Board (xczu7ev-ffvc1156-2-e)**
- **Zynq UltraScale+ ZCU106 Evaluation Platform (xczu7ev-ffvc1156-2-e)**

5.1.2 Software Tools

- **AMD Xilinx Vivado Design Suite (Version 2023.1)**

  Used as the primary environment for RTL design, simulation, synthesis, implementation, and bitstream generation. Vivado's IP catalog was optionally employed for integrating reusable components.

- **ModelSim (Optional)**

  Used for waveform-level simulation and behavioral verification of Verilog modules prior to hardware deployment.

- **Operating System:**

  Development was conducted on **Windows 10 Pro (64-bit)**, supporting both Vivado and ModelSim in a GUI-based environment.

5.1.3 Summary of Usage

The development process adhered to a conventional FPGA design workflow. Verilog was used for hardware description, and simulations were carried out in Vivado to verify correctness. Functional correctness was further validated by comparing simulation outputs against equivalent C models. After successful verification, the designs were synthesized and deployed on various FPGA boards, using multiple operand sizes to evaluate performance, area, and timing metrics. The proposed Karatsuba 8-bit modular polynomial multiplier was implemented and synthesized on the Xilinx Spartan-7 SP701 Evaluation Platform. Figure 5.1 illustrates the post-synthesis logic and module-level visualization of the design as mapped onto the FPGA device. The colored regions in the figure correspond to distinct functional modules, including the core sequential logic and the parallel submodules responsible for partial product computation. This visualization demonstrates the modular and hierarchical structure of the architecture, as well as the efficient utilization of FPGA resources. The use of the Spartan-7 platform validates the practicality and scalability of the proposed design for real-world cryptographic applications.

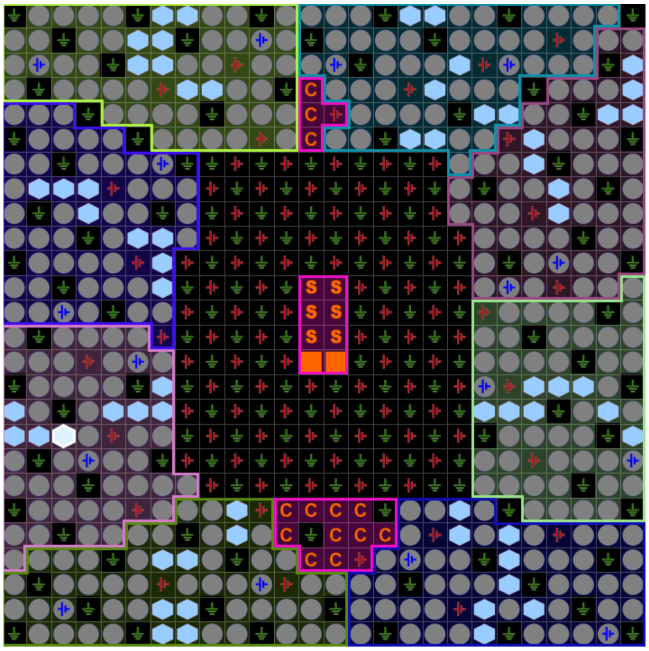

Figure 5.1: Module-level visualization of the Karatsuba 8-bit multiplier on a Spartan-7 SP701 FPGA.

5.2 Synthesization and Comparison Metrics

The comparison of various multiplier designs across different FPGA platforms reveals key insights into their efficiency, flexibility, and resource consumption. The K-method consistently demonstrates the best performance in terms of ADP. This design also supports modularity and reconfigurability, making it highly suitable for applications that require adaptability and optimized performance. In the case of the algorithm from (Heidarpur and Mirhassani, 2021), modular reduction was not applied for operand sizes 8 and 24, as it is generally not recommended for operand sizes below 113 bits. In contrast, both the Reconfigurable GFPM and the K-method incorporate built-in modular reduction as part of their design for flexibility purposes.

The Reconfigurable GFPM also offers reconfigurability and modularity but often at the expense of increased delay and area usage, resulting in the highest ADP on Artix-7 (171,428.818). Reconfigurable GFPM is further optimized for the K-method. Platform-wise, the Spartan-7 platform is associated with lower resource usage and faster perfor-

Table 5.1: Comparison of Different Multiplier Designs by FPGA Implementation

Algorithm	**Size**	**LUTs**	**Slices**	**Total Delay**	**ADP**	**Modular**	**% Reduction**	**Device & Tool**
(Samanta et al., 2014)	8	62	36	13.95	1,367	N/A	74.16	Spartan-7
(Heidarpur and Mirhassani, 2021)		46	24	11.01	770	N/A	54.12	Spartan-7
Reconfigurable GFPM		19	21	28.5	1,140	Yes	69.01	Spartan-7
K-Method Reconfigurable		27	45	4.906	353.232	Yes	–	Spartan-7
(Arish and Sharma, 2015)	24	1,018	972	13.00	25,870	N/A	91.52	Virtex 4
(Heidarpur and Mirhassani, 2021)		360	184	12.51	6,805	N/A	67.78	Virtex 4
Reconfigurable GFPM		51	54	36.984	3,883.32	Yes	43.55	Zynq UltraScale+ MPSoCs
K-Method Reconfigurable		79	130	10.488	2,191.992	Yes	–	Zynq UltraScale+ MPSoCs
(Imana, 2017)	113	5,501	2,354	20.56	1,61,498	Yes	74.71	Artix-7
(Heidarpur and Mirhassani, 2021)		3792	1084	10.52	51,295	Yes	20.40	Artix-7
Reconfigurable GFPM		232	234	397.873	171,428.818	Yes	76.18	Artix-7
K-Method Reconfigurable		355	585	43.434	40,827.96	Yes	–	Artix-7

mance, whereas Artix-7 supports more complex designs but with significantly higher resource demands and latency. The Zynq UltraScale+ MPSoCs represent a balanced middle ground in terms of both performance and resource usage. Overall, the K-method reconfigurable multiplier emerges as the most efficient and flexible design.

5.3 Summary

This chapter presented a detailed comparison of multiple modular polynomial multiplier architectures synthesized and tested on a range of FPGA platforms. The evaluation focused on key performance metrics such as LUT and slice utilization, total delay, and area-delay product (ADP), while also considering the modularity and reconfigurability of each design. A key feature of the table is the ”% Reduction” column in Table 5.1, which quantifies the improvement in ADP achieved by the K-method relative to other referenced architectures. Specifically, this percentage represents how much the K-method reduces the ADP compared to each baseline design, highlighting its superior efficiency. For instance, on the Artix-7 platform, the K-method achieves up to 76.18% ADP reduction compared to the Reconfigurable GFPM, and similar substantial reductions are observed across other platforms and reference designs. The results demonstrate that the K-method consistently delivers the best trade-off between area and speed, owing to its optimized architecture and support for modular reduction. In contrast, traditional and literature-based designs, such as those of (Heidarpur and Mirhassani, 2021) and (Samanta et al., 2014), generally exhibit a higher ADP and lack the modularity or reconfigurability of proposed solutions. The chapter concludes that careful architectural choices and platform-aware optimization are essential for achieving high-performance, scalable, and resource-efficient cryptographic hardware, with the K-method emerging as the most effective solution for a wide range of practical applications.

Chapter 6

Conclusion

This work presents a modular multiplier architecture tailored for efficient polynomial arithmetic in $GF(2^n)$. The proposed design integrates lightweight processing elements, serial shift registers, and fixed modular reduction logic to perform recursive multiplication with high hardware efficiency. The design also supports modularity and reconfigurability, making it highly suitable for applications that require adaptability and optimized performance. This reinforces the effectiveness of our design, especially in space and power constrained environments. These results confirm that the proposed architecture not only meets the requirements of compact and low-power cryptographic hardware but also provides scalable and high-throughput performance suitable for practical deployment. Its subquadratic growth with respect to operand size, combined with modular structure and low-latency characteristics, makes it highly suitable for secure hardware in resource-constrained environments such as embedded systems, WSN devices, and hardware security modules.

The results demonstrate that the proposed multiplier achieves an effective balance between hardware efficiency and performance. Compared to prior works, it significantly reduces area and latency while maintaining competitive delay. The architecture also supports arbitrary modulus polynomials, enabling cryptographic agility without hardware redesign—ideal for resource-constrained environments like WSNs. Table 5.1 illustrates the simulation performance with different FPGA boards by AMD Vivado Design Tool and compares it with other work. It can be seen that our proposed polynomial multiplier reduces ADP and logic usage greatly.

Future work on $GF(2^n)$ multiplier architectures can focus on several promising directions. One approach is the adoption of hierarchical designs that employ recursive block structures to reduce latency and enable logic reuse, thereby improving scalability and maintainability. Another area is gate-level optimization, where refining logic depth can shorten the critical path delay. Furthermore, extending multiplier architectures to support additional cryptographic primitives—such as AES, hash functions, and post-quantum cryptography—would enhance their versatility and ensure long-term applicability in evolving security environments.

Bibliography

S Arish and RK Sharma. An efficient floating point multiplier design for high speed applications using karatsuba algorithm and urdhva-tiryagbhyam algorithm. In *2015 international conference on signal processing and communication (ICSC)*, pages 303–308. IEEE, 2015.

Thomas Beth and Dieter Gollmann. Algorithm engineering for public key algorithms. *IEEE Journal on selected areas in communications*, 7(4):458–466, 1989.

David G Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50(2):285–300, 1989.

Stephen Cohen and Harald Niederreiter. *Finite fields and applications: proceedings of the third international conference, Glasgow, July 1995*, volume 233. Cambridge University Press, 1996.

Whitfield Diffie and Martin E Hellman. New directions in cryptography. In *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pages 365–390. 2022.

Hayssam El and Arash Reyhani-M. New bit-level serial GF (2^m) multiplication using polynomial basis. In *2015 IEEE 22nd Symposium on Computer Arithmetic*, pages 129–136. IEEE, 2015.

Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

Apostolos P Fournaris and O Koufopavlou. Versatile multiplier architectures in GF (2^k)

fields using the montgomery multiplication algorithm. *Integration*, 41(3):371–384, 2008a.

Apostolos P Fournaris and O Koufopavlou. Versatile multiplier architectures in GF (2^k) fields using the montgomery multiplication algorithm. *Integration*, 41(3):371–384, 2008b.

Cornelia Grabbe, Marcus Bednara, J"urgen Teich, Joachim von zur Gathen, and Jamshid Shokrollahi. FPGA designs of parallel high performance GF (2^{233}) multipliers. In *ISCAS (2)*, pages 268–271, 2003.

J-H Guo and C-L Wang. Digit-serial systolic multiplier for finite fields GF (2^m). *IEE Proceedings-Computers and Digital Techniques*, 145(2):143–148, 1998.

Alper Halbutogullari and Cetin K Koç. Mastrovito multiplier for general irreducible polynomials. *IEEE Transactions on Computers*, 49(5):503–518, 2000.

Moslem Heidarpur and Mitra Mirhassani. An efficient and high-speed overlap-free karatsuba-based finite-field multiplier for FPGA implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(4):667–676, 2021.

Jose L Imana. Fast bit-parallel binary multipliers based on type-i pentanomials. *IEEE Transactions on Computers*, 67(6):898–904, 2017.

José Luis Imaña. Low latency GF (2^m) polynomial basis multiplier. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(5):935–946, 2010.

José Luis Imaña, Juan Manuel Sanchez, and Francisco Tirado. Bit-parallel finite field multipliers for irreducible trinomials. *IEEE Transactions on Computers*, 55(5):520–533, 2006.

Anatolii Karatsuba. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, pages 595–596, 1963.

Paris Kitsos, George Theodoridis, and Odysseas Koufopavlou. An efficient reconfigurable multiplier architecture for galois field GF (2^m). *Microelectronics Journal*, 34(10):975–980, 2003.

Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

Xuejia Lai. *On the design and security of block ciphers*. PhD thesis, ETH Zurich, 1992.

Chang-Hyi Lee and Jong-In Lim. A new aspect of dual basis for efficient field arithmetic. In *International Workshop on Public Key Cryptography*, pages 12–28. Springer, 1999.

Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge university press, 1994.

Pramod Kumar 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*, 55(4):1031–1040, 2008.

Pramod Kumar Meher. Systolic and non-systolic scalable modular designs of finite field multipliers for reed–solomon codec. *IEEE transactions on very large scale integration (VLSI) systems*, 17(6):747–757, 2009.

AJ Menezes. Applications of finite fields. *Kluwer Academic Publisher google schola*, 2: 379–423, 1993.

Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

Miguel Morales-Sandoval, C Feregrino-Uribe, and Paraskevas Kitsos. Bit-serial and digit-serial GF (2^m) montgomery multipliers using linear feedback shift registers. *IET Computers & Digital Techniques*, 5(2):86–94, 2011.

Andrew M Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 224–314. Springer, 1984.

William Wesley Peterson and Edward J Weldon. *Error-correcting codes*. MIT press, 1972.

Siva Ramakrishna Pillutla and Lakshmi Boppana. 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)*, pages 920–924. IEEE, 2019.

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

FIPS PUB. Digital signature standard (dss). *Fips pub*, pages 186–192, 2000.

Arash Reyhani-Masoleh and M Anwarul Hasan. A new construction of massey-omura parallel multiplier over GF(2^k). *IEEE Transactions on Computers*, 51(5):511–520, 2002.

R Rivest. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978.

Jagannath Samanta, Razia Sultana, and Jaydeb Bhaumik. FPGA based modified karatsuba multiplier. In *Proc. Int. Conf. VLSI Signal Process.(ICVSP)*, volume 10, page 12, 2014.

B Schneier. Applied cryptography, 2nd edn, j ohn wiley and sons, 1996.

Michael Scott. Optimal irreducible polynomials for GF (2^m) arithmetic. *Cryptology EPrint Archive*, 2007.

George N Selimis, Apostolos P Fournaris, Harris E Michail, and Odysseas Koufopavlou. Improved throughput bit-serial multiplier for GF (2^m) fields. *Integration*, 42(2):217–226, 2009.

Madhan Thirumoorthi, Alexander J Leigh, Moslem Heidarpur, Mohammed Khalid, and Mitra Mirhassani. Novel formulations of m-term overlap-free karatsuba binary polynomial multipliers and their hardware implementations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(10):1509–1522, 2023.

WC Tsai and Sheng-Jyh Wang. Two systolic architectures for multiplication in GF (2^m). *IEE Proceedings-Computers and Digital Techniques*, 147(6):375–382, 2000.

Joachim Von zur Gathen and Jürgen Gerhard. Arithmetic and factorization of polynomial over F2. In *Proceedings of the 1996 international symposium on Symbolic and algebraic computation*, pages 1–9, 1996.

Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2003.

Joachim von zur Gathen and Jamshid Shokrollahi. Efficient FPGA-based karatsuba multipliers for polynomials over. In *International Workshop on Selected Areas in Cryptography*, pages 359–369. Springer, 2005.

Huapeng Wu and M. Anwarul Hasan. Low complexity bit-parallel multipliers for a class of finite fields. *IEEE Transactions on Computers*, 47(8):883–887, 1998.