$GF\left(2^M\right)$ MULTIPLIER IMPLEMENTATION ON A PARTIALLY RECONFIGURABLE FPGA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GIZEM KOCALAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

MAY 2015

Approval of the thesis:

$GF\left(2^M\right)$ MULTIPLIER IMPLEMENTATION ON A PARTIALLY RECONFIGURABLE FPGA

submitted by **GIZEM KOCALAR** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı
Supervisor, **Electrical Electronics Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Şenan Ece Güran Schmidt
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. İlkay Ulusoy Parnas
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Ali Ziya Alkar
Electrical and Electronics Engineering Dept., Hacettepe Uni. _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: GIZEM KOCALAR

Signature :

ABSTRACT

$GF\left(2^M\right)$ MULTIPLIER IMPLEMENTATION ON A PARTIALLY RECONFIGURABLE FPGA

Kocalar, Gizem
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı

May 2015, 88 pages

The thesis aims to design and implement a Galois field multiplier IP block that adapts to changing input traffic conditions, using partial reconfiguration feature of FPGAs. Two different multiplier blocks are used for high and low traffic rates. First, an area efficient multiplier block with small area utilization and low power consumption is designed by using splitting type multiplication algorithms. Second, a high performance multiplier IP block with higher resource consumption but better time performance is designed. At low traffic rates, area efficient multiplier block is used for better area and power utilization. However, when multiplication requests exceed a certain threshold, area efficient multiplier is not capable of serving the incoming requests within an acceptable time. In such cases, the high performance multiplier is activated by reconfiguring the FPGA using a partial bit file to be able to serve the multiplication requests faster. Although power consumption of high performance multiplier is high, on average it is balanced by use of area efficient multiplier when throughput requirement is variable.

Keywords: Galois Field, Karatsuba-Ofman multiplication, multiplication by splitting, FPGA, partial reconfiguration

ÖZ

KISMİ YENİDEN YAPILANDIRILABİLİR FPGA ÜZERİNDE $GF\left(2^M\right)$ ÇARPMA UYGULAMASI

Kocalar, Gizem
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Cüneyt Fehmi Bazlamaçcı

Mayıs 2015 , 88 sayfa

Bu tez, FPGA'lerin kısmi yeniden yapılandırılabilirlik özelliği kullanılarak, değişen giriş trafik koşullarına uyum sağlayabilen bir Galois alanı çarpım IP bloğu tasarımı ve uygulamasını amaçlamaktadır. Yüksek ve düşük trafik oranları için iki farklı çarpım bloğu kullanılmaktadır. İlk olarak, parçalı çarpma algoritmaları kullanılarak, küçük alan kullanımlı ve düşük güç tüketimli bir alan etkin çarpım bloğu tasarlanmıştır. İkinci olarak, daha yüksek kaynak tüketimli ancak daha iyi zaman performanslı bir yüksek performans çarpım IP bloğu tasarlanmıştır. Düşük trafik oranlarında, daha iyi alan ve güç kullanımı için alan etkin çarpım bloğu kullanılmaktadır. Ancak, çarpma istekleri belli bir eşiği aştığında, alan etkin çarpım gelen isteklere geçerli bir zamanda servis verememektedir. Böyle durumlarda çarpma isteklerine daha hızlı servis verebilmek için FPGA kısmi bit dosyası kullanılarak yeniden yapılandırılıp yüksek performans çarpım etkinleştirilmektedir. Yüksek performans çarpımın güç tüketimi yüksek olmakla birlikte işlem hacim gereksinimi değişken olduğunda alan etkin çarpımın kullanımıyla ortalamada dengelenmektedir.

Anahtar Kelimeler: Galois Alanı, Karatsuba-Ofman çarpımı, parçalı çarpma, FPGA, kısmi yeniden yapılandırılabilirlik

To my family

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my advisor Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı for his guidance and motivation throughout the study.

I sincerely thank Dr. Murat Cenk for his contributions to the mathematical background of this thesis.

I would like to thank my employer, ASELSAN, for supporting this study. I also would like to thank my colleagues Bengisu Tengilimoğlu, Berk Ülker, Emin Birey Soyer and Mehmet Ufuk Büyükşahin for their support.

Last but not least, I would like to thank my family for their love and support throughout my life.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xii

LIST OF FIGURES xiv

LIST OF ALGORITHMS xvi

LIST OF ABBREVIATIONS xvii

CHAPTERS

1 INTRODUCTION 1

1.1 AIM OF THE THESIS 3

1.2 CONTENT OF THE THESIS 4

2 BACKGROUND INFORMATION 5

2.1 CRYPTOGRAPHIC ALGORITHMS 5

2.1.1 SYMMETRIC KEY CRYPTOGRAPHY 7

2.1.2 PUBLIC KEY CRYPTOGRAPHY 8

2.1.3 ELLIPTIC KEY CRYPTOGRAPHY 10

2.1.4 HASH FUNCTIONS 10

2.2 GALOIS FIELD 10

2.3 IMPLEMENTATION PLATFORM 12

2.3.1 INTERNAL STRUCTURE OF FPGA 14

2.3.2 PARTIAL RECONFIGURATION 16

2.4 ARCHITECTURAL STRUCTURE 17

2.5 MULTIPLICATION ALGORITHMS 20

2.5.1 CLASSICAL MULTIPLICATION 20

2.5.2 MONTGOMERY MULTIPLICATION 21

2.5.3 KARATSUBA-OFMAN MULTIPLICATION . . . 23

3 LITERATURE WORK 27

3.1 HARDWARE IMPLEMENTATION OF KARATSUBA-OFMAN MULTIPLICATION 28

3.2 SPLITTING ALGORITHMS 29

3.3 APPLICATIONS OF GALOIS FIELD MULTIPLICATION . 34

3.4 FPGA DESIGNS USING PARTIAL RECONFIGURATION 38

4 AREA OPTIMIZED $GF\left(2^M\right)$ MULTIPLIER IMPLEMENTATION ON FPGA 39

4.1 IMPLEMENTATION RESULTS 41

4.2 COMPARISON OF RESULTS 49

4.3 REDUCTION BLOCK 52

5 PARTIALLY RECONFIGURABLE $GF(2^{128})$ MULTIPLIER DESIGN 55

5.1 MOTIVATION 55

5.2 DESIGN 56

5.3 SIMULATION AND IMPLEMENTATION 59

5.4 RESULTS 62

6 CONCLUSION 73

REFERENCES 77

APPENDICES

A AREA OPTIMIZATION ON VIRTEX 5 81

B AREA OPTIMIZATION ON KINTEX 7 85

LIST OF TABLES

TABLES

Table 2.1 Area and time performances of $GF(2^m)$ multiplier designs with different architectural structures 18

Table 3.1 KOM implementation results for FPGAs obtained in [16] 29

Table 3.2 Computational and delay complexities of splitting algorithms 30

Table 3.3 Cryptographic algorithms and security levels 34

Table 3.4 Time and area performances of multiplier blocks designed in [25] . . 36

Table 3.5 Throughput and area performances in [29] with and without partial reconfiguration 38

Table 4.1 Galois fields used for experiments with characteristic polynomials . 41

Table 4.2 LUT consumption of analyzed multiplier designs for selected fields on Virtex 5 50

Table 4.3 LUT consumption of analyzed multiplier designs for selected fields on Kintex 7 51

Table 5.1 Power consumption (in mW) of three multiplier types under different input traffic conditions 64

Table 5.2 Throughput of three multiplier types under different input traffic conditions in terms of packets/clock cycle 65

Table 5.3 Average delay (in ms) of three multiplier types under different input traffic conditions 66

Table 5.4 Maximum delay (in ms) of three multiplier types under different input traffic conditions 66

Table 5.5 Maximum buffer size requirement (in packets) of partially reconfigurable multiplier for different threshold values (in packets) under traffic conditions 69

Table 5.6 Effective area utilization of partially reconfigurable multiplier block under different traffic rates 70

Table A.1 Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5 81

Table A.1 Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5 (Continued) 82

Table A.1 Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5 (Continued) 83

Table A.1 Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5 (Continued) 84

Table B.1 Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7 85

Table B.1 Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7 (Continued) 86

Table B.1 Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7 (Continued) 87

Table B.1 Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7 (Continued) 88

LIST OF FIGURES

FIGURES

Figure 2.1 Interruption of flow of information 6

Figure 2.2 Interception of information 6

Figure 2.3 Modification of information 6

Figure 2.4 Fabrication of information 7

Figure 2.5 Symmetric key cryptography operational diagram 7

Figure 2.6 Public key cryptography operational diagram 9

Figure 2.7 Partial reconfiguration block diagram 17

Figure 2.8 Operation of pipelined architecture 19

Figure 2.9 Implementation of CM on hardware (Zhou et al., 2010, p. 1058) . . 21

Figure 2.10 Implementation of KOM in $GF\left(2^m\right)$ 25

Figure 3.1 AES-GCM operational block diagram 35

Figure 4.1 Algorithms selected for area optimized $GF\left(2^{113}\right)$ multiplier for Virtex 5 and Kintex 7 43

Figure 4.2 Algorithms selected for area optimized $GF\left(2^{128}\right)$ multiplier for Virtex 5 and Kintex 7 44

Figure 4.3 Algorithms selected for area optimized $GF\left(2^{163}\right)$ multiplier for Virtex 5 and Kintex 7 45

Figure 4.4 Algorithms selected for area optimized $GF\left(2^{193}\right)$ multiplier for Virtex 5 and Kintex 7 46

Figure 4.5 Algorithms selected for area optimized $GF\left(2^{233}\right)$ multiplier for Virtex 5 and Kintex 7 47

Figure 4.6 Algorithms selected for area optimized $GF\left(2^{283}\right)$ multiplier for Virtex 5 and Kintex 7 48

Figure 4.7 Area utilization of analyzed multiplier designs for selected fields on Virtex 5 50

Figure 4.8 Area utilization of analyzed multiplier designs for selected fields on Kintex 7 51

Figure 4.9 Reduction block 52

Figure 5.1 Operational block diagram for $GHASH_H\left(X_1||X_2||...||X_m\right) = Y_m$ 56

Figure 5.2 Partially reconfigurable multiplier block and queue structure 57

Figure 5.3 Implementation of partially reconfigurable multiplier on FPGA with high performance multiplier block implemented in the reconfigurable partition 60

Figure 5.4 Implementation of partially reconfigurable multiplier on FPGA with nonfunctional black box module implemented in the reconfigurable partition 60

Figure 5.5 Area utilization of three multiplier types in terms of number of utilized LUTs 63

Figure 5.6 Power consumption of three multiplier types under different traffic rates 64

Figure 5.7 Throughput of three multiplier types under different traffic rates . . 65

Figure 5.8 Average delay values of three multiplier types under different traffic rates 67

Figure 5.9 Maximum delay values of three multiplier types under different traffic rates 67

Figure 5.10 Effective area utilization of partially reconfigurable multiplier compared to area utilization of static area optimized and high performance multipliers under traffic rates $\lambda = 0.3$, 0.5 and 0.7 70

LIST OF ALGORITHMS

ALGORITHMS

Algorithm 2.1 Karatsuba-Ofman Multiplication algorithm 24

Algorithm 3.1 Bernstein's 3-way split algorithm 30

Algorithm 3.2 Karatsuba-like improved 3-way split algorithm 31

Algorithm 3.3 Bernstein 4-way split algorithm 32

Algorithm 3.4 Cenk improved 5-way split algorithm 33

LIST OF ABBREVIATIONS

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
BRAM	Block Random Access Memory
CBC	Cipher Block Chaining
CLB	Configurable Logic Block
CM	Classical Multiplication
DES	Data Encryption Standard
DH	Diffie-Hellman
DSA	Digital Signature Algorithm
DSP	Digital Signal Processing
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FPGA	Field Programmable Gate Array
GCM	Galois Counter Mode
GF	Galois Field
HDL	Hardware Description Language
ICAP	Internal Configuration Access Port
IDEA	International Data Encryption Algorithm
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
KOM	Karatsuba-Ofman Multiplication
LUT	Look Up Table
MAC	Message Authentication Code
MD5	Message Digest 5
NGE	Next Generation Encryption
NIST	National Institute of Standards and Technology
RAM	Random Access Memory

RC4	Rivest Cipher 4
RSA	Rivest, Shamir, Adleman
SHA	Secure Hash Algorithm

CHAPTER 1

INTRODUCTION

As the technology advances, communication via electronic devices becomes more widespread. Information exchange through communication networks using electronic devices such as smart phones, personal computers, integrated circuit cards get more common, bringing in the ease of rapid access to any information in our lives. However, sharing information through communication networks has also the disadvantage of threatening confidentiality. During data transmission between two devices via an insecure media such as the Internet, someone else may interfere with by eavesdropping, where information is undamaged but privacy is compromised, by tampering, where information is changed and then sent to the receiver node, or by impersonation, where the information is delivered to a node that pretends to be the intended recipient [1]. In such cases cryptographic applications are of critical importance to provide information security.

Main cryptographic algorithms used for Internet security are symmetric key cryptography and public key cryptography. Symmetric key cryptography uses the same key for encryption and decryption of data. Therefore, it requires transmission of the encryption key along with the encrypted data. On the other hand, public key cryptography encrypts and decrypts the data using two separate keys, public key and private key, which are mathematically connected. Although public key cryptography may provide higher level of security, symmetric key cryptography has better performance in authenticated encryption applications.

In implementation of cryptographic algorithms, efficiency is the main objective [2]. Two criteria considered in the implementation of a cryptographic application are

achieving high speed operation and efficient use of hardware resources. Achieving high speed operation is important in order to provide high speed data transmission between the communicating nodes. Since encryption and decryption of communication data is one of the most time consuming operations, fast implementation of cryptographic application is an essential constraint for the whole system. In addition to the timing constraints, since cryptographic applications are implemented mostly on portable devices with limited resources, area efficiency is also a critical goal to achieve. Resource constraint is not only for providing area efficiency, but it is also necessary for reducing power consumption.

Galois Field multiplication is an essential functional block in cryptographic algorithms. During encryption and decryption processes, arithmetic operations are handled in binary Galois fields. Although it is most popular for its use in cryptographic applications, there are many other areas that Galois Field arithmetic has a significant role. In coding theory, for almost all error correcting coding applications algebraic properties of Galois Field are used for achieving data transmission without errors. It is also used in digital signal processing applications in order to perform operations like convolution, filtering, discrete Fourier transform. Another area that Galois Field arithmetic has important functionality is computer algebra. In computer algebra systems, factorization of polynomials over Galois Fields is a fundamental mathematical block. Since Galois Field arithmetic applications have wide range of implementation areas, achieving high speed performance without wasting large amount of resources is an inevitable requirement.

In a $GF\left(2^m\right)$ arithmetic unit the basic operations are addition and multiplication. Other operations such as division, inversion and exponentiation are performed through several number of multiplications. Multiplication operation in $GF\left(2^m\right)$ is modular multiplication. In other words, in order to obtain the product of two numbers, first the multiplicands are multiplied and then modulo the characteristic polynomial of the result is computed. Complexity of addition in $GF\left(2^m\right)$ is very low compared to multiplication. Therefore, multiplier block in a $GF\left(2^m\right)$ arithmetic unit determines the overall performance.

1.1 AIM OF THE THESIS

A fast and compact $GF(2^m)$ multiplier block is an essential requirement in cryptographic applications. Area complexity and power consumption should be low in an effective design. Scalability is also an important point to consider. These requirements can be satisfied by designing a dedicated hardware block for $GF(2^m)$ multiplication. An FPGA based implementation of $GF(2^m)$ multiplier block can provide sufficient time, area and power performances for these requirements. Plus, the flexible and easily reconfigurable structure of FPGAs eases design and implementation processes.

In the first implementation part of this thesis, different algorithms for $GF(2^m)$ multiplication are evaluated to design an area efficient multiplier. Since cryptographic applications require multiplication of large numbers, the design is based on divide-and-conquer approach to reduce the problem of large number multiplication into several smaller number multiplications. This approach also reduces the complexity of operation by replacing large number multiplication by addition and some smaller number multiplications. Bit parallel architecture is preferred to provide higher operation speed.

In the second implementation part, a multiplication block design is proposed to be used in Advanced Encryption Standard - Galois Counter Mode (AES-GCM) core, which is the main Internet data security algorithm today. Galois Field multiplier is an essential functional block in an AES-GCM core. The two important criteria in implementation of Galois Field multiplier for AES-GCM applications are low delay and power efficiency. In order to achieve these goals, we included two different multiplier blocks in our Galois Field multiplication module. The area optimized multiplier block designed in the first implementation part is used for less resource utilization and lower power consumption. A high performance multiplier is also used in the design to meet low delay and high throughput requirements. Using the partial reconfiguration feature of FPGAs, high performance multiplier block is included in design at high traffic rates to provide better service to incoming data hence multiplication requests and excluded at low traffic rates in order to reduce area and power consumption. As a result, both time and area performances of our AES-GCM multiplier block

are improved by making real time adaptations to changing traffic conditions.

1.2 CONTENT OF THE THESIS

Rest of the thesis is organized in the following way.

First some background information about cryptographic algorithms, Galois Fields and arithmetic operations in $GF(2^m)$, and some concepts related to hardware block design are given in Chapter 2. Some commonly used algorithms for Galois field multiplier design are introduced.

Then, in Chapter 3, literature work about $GF(2^m)$ multiplication is included and the algorithms combined in the design are explained in detail. Some Galois field multiplication applications in cryptography are examined. Also some dynamically reconfigurable hardware designs are overviewed.

In Chapter 4, an area optimized $GF(2^m)$ multiplier block is implemented. Implementation results of this multiplier are analyzed and compared to similar designs that are already introduced in literature.

Chapter 5 includes a multiplier block design for cryptographic applications, which uses the area optimized multiplier block designed in the fourth chapter and a high performance multiplier block. Partial reconfiguration feature of FPGAs is utilized in this implementation. AES-GCM is chosen as the example cryptographic application and Galois field multiplication in GHASH function is simulated.

After the analysis of results, thesis is concluded with Chapter 6.

CHAPTER 2

BACKGROUND INFORMATION

2.1 CRYPTOGRAPHIC ALGORITHMS

Effective protection of information is possible by making a good organization of security requirements. In order to achieve this, six atomic elements of information, also known as the Parkerian hexad, are defined in [3] as follows:

- **Confidentiality:** Information is accessible only for authorized parties.
- **Possession:** Access to information resources is restricted. Possession is different from confidentiality. If resource of information is accessed by an unauthorized party this is a loss of possession, but there is not breach of confidentiality as long as the information is not reached.
- **Integrity:** Information is modified only by authorized parties.
- **Authenticity:** Origin or authorship of information is correctly identified.
- **Availability:** Accessibility of system resources by authorized parties whenever needed.
- **Utility:** Usefulness of information. If information is encrypted and then decryption key is lost, this is an example to breach of utility.

A security attack destroys one of the above information elements during flow of information. In [4], security attacks are characterized according to the destroyed function of data transmission as follows:

- **Interruption:** Attack on availability, shown in Figure 2.1. Some system resources are destroyed or they become unavailable or unusable.

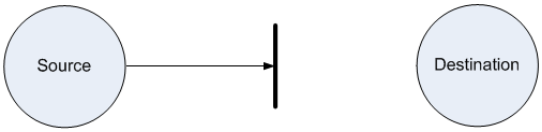


Figure 2.1: Interruption of flow of information

- **Interception:** Attack on confidentiality, shown in Figure 2.2. Information is accessed by an unauthorized party.

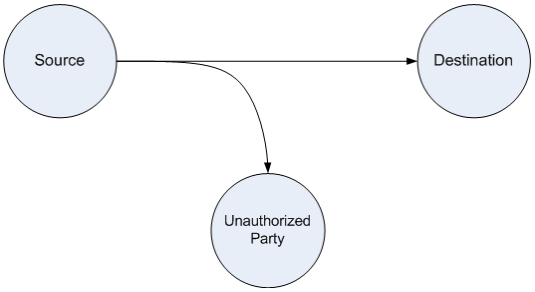


Figure 2.2: Interception of information

- **Modification:** Attack on integrity, shown in Figure 2.3. An unauthorized party tampers with the information.

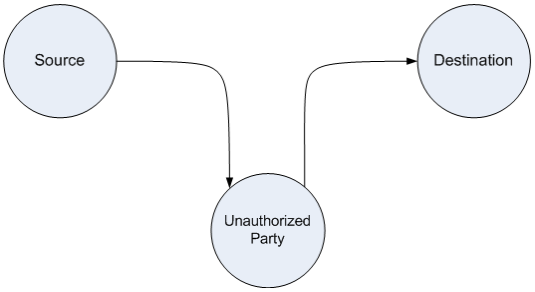


Figure 2.3: Modification of information

- **Fabrication:** Attack on authenticity, shown in Figure 2.4. Counterfeit information is inserted by an unauthorized party.

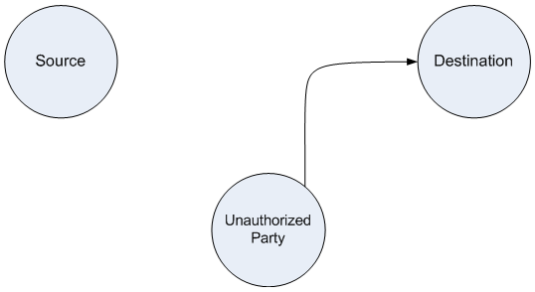


Figure 2.4: Fabrication of information

For transmission of data through an insecure media, such as the Internet, a cryptographic algorithm should be used in order to meet security needs. Numerous cryptographic algorithms are defined to achieve communication security. These algorithms can be mainly grouped as symmetric key cryptography, public key cryptography, elliptic key cryptography and hash functions. All types of cryptographic algorithms are used in Internet security applications for providing different levels of security.

2.1.1 SYMMETRIC KEY CRYPTOGRAPHY

In symmetric key cryptography, the same key is shared by transmitter and receiver nodes. The key is distributed among all nodes on the communication channel before any information is encrypted and sent or it is sent to the receiver node after encryption of data at the transmitter node along with the ciphertext. On reception of the ciphertext and the key, data is decrypted on the receiver side to obtain the plaintext. An operational diagram of symmetric key cryptography is shown in Figure 2.5.

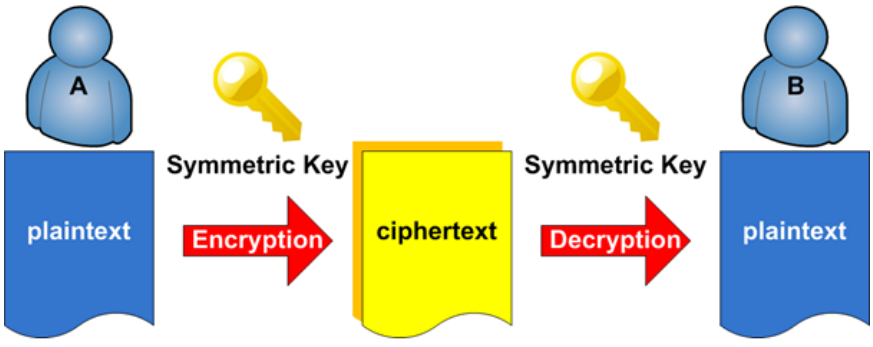


Figure 2.5: Symmetric key cryptography operational diagram

Symmetric key cryptography algorithms are simple to implement and fast in operation. However, they require transmission of the key between communicating nodes and key exchange should be done in a secure way to prevent unauthorized access. Numerous algorithms exist for key exchange between the nodes that provide secure transmission of the encryption key. Two types of symmetric key encryption are defined [5]. One is block cipher, which segments the plaintext into blocks of fixed length. Each block is encrypted using the symmetric key to obtain the corresponding ciphertext block. Ciphertext blocks have the same size as plaintext blocks. They are concatenated to form the ciphertext and sent to the receiver side. The other is stream cipher, which breaks the original plaintext into blocks with a fixed beginning but no fixed end. In this type of symmetric key ciphers, message block length is indefinite and thus modeled as an infinite length stream. However, in practice encryptor has finite memory and end of a message segment is defined by continuous observation.

Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are examples of symmetric key cryptography algorithms. Both DES and AES are block ciphers. DES uses 64-bit-long blocks with 56-bit-long key. Because of its small key size it is no longer considered to be secure. On the other hand, AES uses 128-bit-long blocks with 128-bit, 192-bit or 256-bit-long keys. Therefore it is more secure and still provides adequate protection when used with 128-bit key. For higher security levels longer keys are used [6].

2.1.2 PUBLIC KEY CRYPTOGRAPHY

Public key cryptography, or asymmetric key cryptography, is based on use of two separate keys for ciphering and deciphering the data. The reason this approach is called asymmetric is that the two keys used for encryption and decryption, namely public key and private key, are not the same. Since encryption and decryption keys are different, this approach is regarded to be more secure than symmetric cryptography [5].

Principle of operation in public key cryptography is described in [4] as follows: In a network where public key cryptography is used, every user is provided a public key and a private key. Public key of a user is contained in a public directory and available

to all other users in the network, whereas private key is only known by the user itself. Whenever a data transmission is to occur between two nodes, namely transmitter and receiver, first the data is encrypted at transmitter side using the public key of receiver. Then the encrypted data is sent through the insecure media. When receiver gets the encrypted data, it is decrypted using the private key, which is mathematically linked to the public key. The operational diagram in Figure 2.6 illustrates data transmission in an insecure media using public key cryptography algorithm.

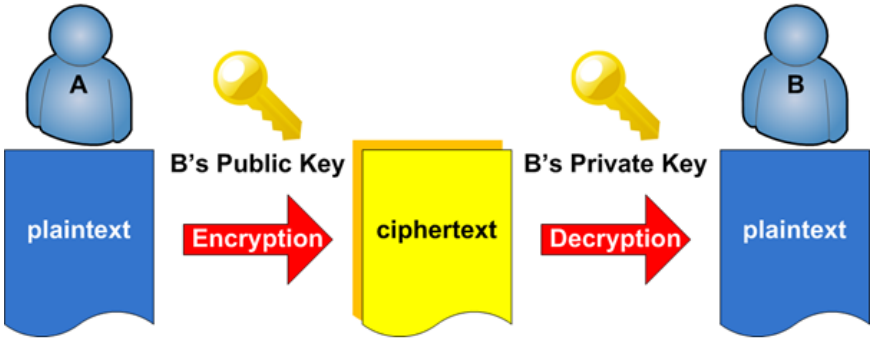


Figure 2.6: Public key cryptography operational diagram

Key generations in public key cryptography algorithms are based on difficult mathematical problems. What is meant by difficult is that computational complexity of solution of the problem is so high that it is not feasible to compute the private key using the public key. Integer factorization and discrete logarithm problems are examples of difficult problems, which are used for key generation in different public key cryptography algorithms.

Some examples of public key cryptography algorithms are Digital Signature Algorithm (DSA), Diffie-Hellman (DH) algorithm and RSA algorithm. However, these algorithms are not considered to be secure when used with small key sizes since subexponential attacks to break them exist. In order to provide a sufficient security level, their key sizes must be increased, which may not be an effective solution in practice.

2.1.3 ELLIPTIC KEY CRYPTOGRAPHY

Elliptic curve cryptography is an approach to public key cryptography, which is based on elliptic curves over finite fields. It uses Elliptic Curve Discrete Logarithm Problem to provide security, since it is a difficult problem to solve. The advantage of elliptic curve cryptography over public key cryptography algorithms is that it requires less hardware resources and consumes less power while providing high throughput and equal level of security [4]. Some examples of elliptic curve cryptography algorithms are Elliptic Curve Digital Signature Algorithm (ECDSA), Elliptic Curve Diffie-Hellman (ECDH) algorithm.

2.1.4 HASH FUNCTIONS

As defined in [5], a cryptographic hash function is a one-way function to output a digest from the input message. The input message may have any length but the hash function produces fixed length digests. Cryptographic hash functions are used for authentication in communication security applications. A cryptographic hash function is a deterministic function that always produces the same digest for a specific message. A good hash function is easy to compute and infeasible to invert. Another property a cryptographic hash function has that it is collision resistant, that is, it is hard to find two different messages for which the function produces the same digest. Some examples of cryptographic hash functions are Message Digest 5 (MD5), Secure Hash Algorithm 256 (SHA-256).

2.2 GALOIS FIELD

A field is defined as an abstraction of familiar number systems and their essential properties in [7]. A set of numbers, F, with addition and multiplication operations, which satisfies the usual arithmetic properties, namely additive identity, multiplicative identity and distributive law, is a field. Galois field, or finite field, is a field that contains finite number of elements in it.[1] It is represented using the notation $GF\,(p^n)$,

[1] More details on Galois field types and arithmetic operations can be found in the book [7] Chapter 2: Finite Field Arithmetic.

where GF represents Galois Field, p is the characteristic of the field and n is a positive integer. In a Galois field, the characteristic p is a prime number and defines the basis used in that field. In other words, elements of the field $GF\,(p^n)$ are represented using the numbers in the set $GF(p) = 0, 1, \ldots, p-1$. The power n gives the length of numbers in the field. A Galois field contains p^n elements, where p^n is called the order of the field. For example, the field $GF\,(2^3)$ has characteristic 2 and order 8, meaning that elements of the field are represented using basis 2, which is the binary form, and they are 3-digit-long. Order of the field is 8, that is, total number of elements in this field is 8.

In addition to characteristic and order, another element that defines a Galois field is the characteristic polynomial. Characteristic polynomial of $GF\,(p^n)$ is an irreducible polynomial of degree n. Irreducibility of a polynomial means that it is not possible to express that polynomial as a product of polynomials with smaller degrees [7]. As an example, characteristic polynomial of the field GF(2^3) can be $f(x) = x^3 + x + 1$.

Two operations are defined over a Galois field, being addition and multiplication. As an example, consider the field $GF\,(p^n)$. A and B are two elements of this field, where

$$A = \sum_{i=0}^{n-1} a_i x^i$$
$$B = \sum_{i=0}^{n-1} b_i x^i$$

for $(a, b)\,\epsilon\,\{0, 1, \ldots, p-1\}$. Characteristic polynomial of the field is $f(x)$. Arithmetic operations are defined as follows.

i. Addition

   Addition is the usual addition of polynomials with coefficient arithmetic performed modulo p. It is formulated as given in 2.1.

$$A + B = \sum_{i=0}^{n-1} (a_i + b_i) \mod p * x^i \tag{2.1}$$

ii. Multiplication

   Multiplication is performed modulo the characteristic polynomial. It is usually done in two steps, which are polynomial multiplication and modular reduction.

First a product is generated by usual polynomial multiplication. Then the actual result is obtained by dividing the product by the characteristic polynomial and taking the remainder. It is formulated as given in 2.2.

$$\begin{aligned} A * B &= \left(\sum_{i=0}^{n-1} a_i x^i \sum_{j=0}^{n-1} b_j x^j \right) \mod f(x) \\ &= \left(\sum_{k=0}^{2n-2} c_k x^k \right) \mod f(x) \\ &= \left(\sum_{i=0}^{n-1} d_i x^i \right) \end{aligned} \tag{2.2}$$

Subtraction in Galois fields is defined in terms of addition using additive inverse of the subtrahend. Similarly, division is defined in terms of multiplication using multiplicative inverse of the divisor.

Galois fields of characteristic 2 are called binary fields. Binary fields are preferred in many applications of Galois fields for their suitability to software and hardware implementations. Addition in binary fields is done by bitwise XOR operation. For bit multiplication, bitwise AND operation is done.

2.3 IMPLEMENTATION PLATFORM

In many cryptographic applications, basic arithmetic operations are performed over Galois Fields. Among these operations, addition and multiplication are the most essential ones, since the other operations such as inversion, exponentiation and division are carried out by using these two basic operations. In $GF\left(2^m\right)$ arithmetic, carryless addition is performed on binary numbers, which is represented by simple bitwise XOR operation. On the contrary, complexity of multiplication is much higher. Polynomial multiplication and modular reduction operations are performed consecutively to achieve a single multiplication in $GF\left(2^m\right)$. Exponentiation, inversion and division are performed by multiple multiplications. Hence, time and area complexities of multiplication operation determine the overall performance of the arithmetic unit. In order to provide better service to end user, multiplier block should be able to satisfy some time constraints. Plus, for portable devices power efficiency and area reduction

are of critical importance. This fact brings out the need for a fast multiplier with high throughput and low circuit complexity.

Delay performance and circuit complexity of the multiplier block in a $GF(2^m)$ arithmetic unit is important, since it is the most complex arithmetic block and has critical importance in determining the overall performance of the unit. There are numerous works related to $GF(2^m)$ multiplier design for satisfying these two constraints both in software and hardware. However, most of the more recent works focus on hardware implementation. The reason why software implementation of the multiplier block in a cryptographic application is not attracting much attention is its poor timing performance compared to hardware implementation [8]. Although many microprocessors have special instructions for Galois Field multiplication, such as Polynomial Multiplication Instruction PCLMULQDQ of Intel processors, none is able to provide better timing performance compared to a dedicated hardware implementation. Plus, such instructions are restricted to perform multiplication over 1-word-length multiplicands, which are 32-bit or 64-bit numbers; thus, long number multiplication is not possible using a single instruction. Such operations take more time and are not able to satisfy high speed requirements. Although fast multiplication algorithms for software implementation such as Comba and Karatsuba are used in some encryption systems [9], they are only able to satisfy basic security needs, where performance is not the primary concern, such as message encryption, file or folder encryption, as indicated in [8]. On the other hand, hardware solutions achieve high speed performance. In addition, they scale well to very large number multiplication requirements. Therefore, using a dedicated hardware block for $GF(2^m)$ multiplication seems to be a better solution for fast implementation with low resource consumption, especially in high performance security critical systems.

It is easy to implement $GF(2^m)$ arithmetic related operations on hardware platforms since operations are performed over binary fields anyway. As hardware implementation platform, two alternatives exist, which are Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA). A comparison of these two implementation platforms is made in [10] to state the advantageous and disadvantageous aspects of designing on either platform. ASIC technology is generally preferred for high density implementations, whereas FPGA based architectures are

preferred for smaller size designs. In terms of area and power consumption performances ASIC designs are far more advantageous compared to FPGAs. Plus, ASIC designs have better time performances with smaller delay and higher frequency operations. Another work in the literature related to performance comparison of ASIC and FPGA designs is [11], showing that ASIC designs outperform FPGA designs in terms of area, time and power consumption. In [11], it is stated that for designs using only logic clusters FPGA implementations are an average of 35 times larger than ASIC implementations and consume an average of 14 times more dynamic power, with an average of 3.4 times slower time performance. Despite these advantages ASIC designs offer, their design and implementation processes are costly. Moreover, it is not possible to make changes in the design after its production. Therefore, optimization in terms of logic gates is critical before the manufacturing process. On the other hand, FPGAs have the advantage of reprogrammability. Design on an FPGA can be changed as needed until the final specification is satisfied. With this flexibility provided by the reprogrammability feature of FPGA, it is possible to make even real time adaptions in hardware configuration, which is also the case for $GF\left(2^m\right)$ multiplier block implementation done in this thesis. Moreover, applications of $GF\left(2^m\right)$ multiplication involves small to medium size designs, for which optimization on ASIC platform is not a cost effective approach. As a result, an FPGA based implementation is regarded as a more suitable solution for $GF\left(2^m\right)$ multiplier design field, especially for real time reconfiguration ability.

2.3.1 INTERNAL STRUCTURE OF FPGA

Many multiplication methods are proposed for $GF\left(2^m\right)$ multiplier block implementation, claiming to have better time or area performances. Circuit complexity is theoretically calculated by number of mathematical operations. Number of bit additions and bit multiplications give the total number of gates used. Time performance has two elements, which are critical path and total circuit delay. Critical path is the part of circuit which has the longest delay. To compute total delay, one needs to know in how many steps total operation is done and what is the delay of each step. All these complexities obtained by theoretical computations give valid results for ASIC implementations. However, they are not meaningful for FPGA designs. Since FPGA

structure is not based on gates, theoretical time and area computations can not be used for computing FPGA implementation complexities.

FPGAs consist of five types of main blocks, which are Configurable Logic Blocks (CLB), dedicated blocks such as DSP slices, gigabit transceivers and block RAMs, input and output blocks providing interface to external devices, routing to provide connection between internal blocks and clocking resources. CLBs are the functional blocks where HDL code is implemented unless other block types are specified for implementation. Inside CLBs there are slices that contain several number of Look Up Tables (LUT). Any function defined by HDL code is implemented by the LUTs, which are the basic functional blocks of an FPGA. A LUT gets a certain number of input signals to produce one output. Number of LUTs consumed in a design is not determined by the circuit complexity, but it is determined by the number of input signals. When the same HDL code is implemented both on ASIC and FPGA, there is not one-to-one gate conversion due to their internal structure difference. Although theoretical gate count gives circuit complexity of ASIC implementation, it is not possible to state how many LUTs would be used exactly to implement a circuit on an FPGA, but it can only be estimated. Also for time complexity analysis, theoretical results are not meaningful. In an FPGA, all LUTs have the same delay and LUT delay is computed by the synthesis tool. Once the LUT delay is known total time of operation is calculated by using number of steps.[2]

Experimental work in this thesis is carried out on two different FPGA platforms, namely Virtex 5 and Kintex 7. Cited literature on the other hand has utilized Virtex 4 and Virtex 5 FPGAs. All three FPGA platforms (Virtex 4, 5 and Kintex 7) have different internal structures. Virtex 4 has 4-input 1-output LUT structure. Virtex 5 and Kintex 7 have 6-input 2-output LUT structures. For Virtex 5 and Kintex 7, 6-input 1-output or 5-input 2-output configurations can be used. Gate definitions in the HDL code are not directly implemented on an FPGA, but they are converted into logic functions to be implemented on LUTs. Resource consumption of LUT based implementations is not limited by the complexity of the circuit but by the number of inputs and outputs used [30]. Since internal structures of FPGAs are different and

[2] Detailed information about FPGA architecture and FPGA and ASIC comparison can be found on http://www.xilinx.com/

synthesis tool preferences have serious effects on resource utilization, it is not possible to tell exact logic resource consumption of an FPGA implementation. Although rough estimations can be made, resource utilization changes when different FPGA types and synthesis tools are used.

To sum up, depending on an FPGA's internal architecture, implementation of a system design on an FPGA may not be compliant with the expected resource consumption and timing results that are based on theoretical complexity computations. There is no way of exactly computing FPGA resource consumption but only a rough estimate can be done. It is possible to obtain the exact results by implementing the circuit in HDL and then synthesizing for an actual target FPGA chip.

2.3.2 PARTIAL RECONFIGURATION

Working on a flexible and dynamically configurable platform enables faster and easier design and implementation processes. In many cases, FPGA based designs are preferred for their reconfigurability. However, reconfiguration of FPGA generally does not take place at run time. While changing the functionality of an FPGA, the operation does not continue. As a solution to this fact, partial reconfiguration is introduced by FPGA vendors. For Xilinx FPGAs, partial reconfiguration is introduced for FPGA families Virtex-4, Virtex-5, Virtex-6, Kintex-7 and Artix-7 [12].

Partial reconfiguration is a technological development that allows run time modification of FPGA configuration. In a partially reconfigurable design, the design consists of a static part and a reconfigurable part. The static module is implemented once and it can not be changed during run time. The reconfigurable partition is the part than can be configured on the fly using a partial bit file for each different configuration. While static module continues its operation, reconfigurable partition can be changed using reconfigurable modules in order to make adaptations on the hardware designed. A partial bit file is included in the design through Internal Configuration Access Port (ICAP). During reconfiguration time, reconfigurable partition stops operation. A block diagram of a partially reconfigurable design is given in Figure 2.7.

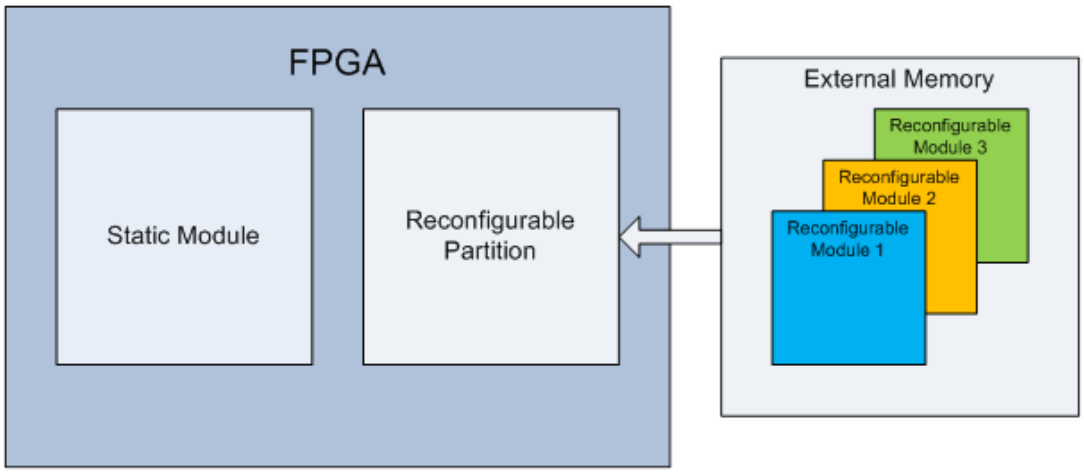


Figure 2.7: Partial reconfiguration block diagram

Partial reconfiguration enables more flexible use of hardware resources on an FPGA based design. Being able to dynamically change the hardware platform and make run time adaptations does not only reduce area and power consumption, but it also provides more functionality with restricted amount of resources. By using partial reconfiguration in an FPGA design it is possible to obtain more compact and more functional hardware platform that is able to dynamically adapt to changing environmental conditions, user preferences or performance metrics.

2.4 ARCHITECTURAL STRUCTURE

Various design styles are proposed for hardware implementation of Galois field multipliers, including designs that are based on different architectural structures. One classification of hardware architectures is based on data input structures. There are two main design styles in this classification, which are bit parallel and bit serial architectures [13]. Deciding on whether to design parallel or serial input hardware block is a common example of area-time trade-off paradigm. Bit parallel architectures receive input data at one clock cycle and start processing; therefore, they are faster. On the other hand, bit serial designs get input data bit by bit at each clock cycle. Although bit serial designs use less hardware resources, they are not suitable for high-speed implementations. Bit parallel designs are more advantageous in terms of time per-

Table 2.1: Area and time performances of $GF(2^m)$ multiplier designs with different architectural structures

Design	Number of AND gates	Number of XOR gates	Number of Registers	Latency
Bit parallel	m^2	m^2-1	$2m$	$(2+\lceil \log_2(m+1) \rceil)\, T_X + T_A$
Bit serial	m	$m+1$	$2m$	$(m+1)\,(T_X+T_A)$
Digit serial	$d*m$	$d*(2m+1)$	$2m$	$\lceil m/d \rceil *$ $(\lceil 1+log_2 d \rceil\, T_X + T_A)$

[1] T_A: delay of a 2-input AND gate.
[2] T_X: delay of a 2-input XOR gate.

formance. In addition, they offer high throughput when used in conjunction with pipelining techniques.

Both design styles are proposed in numerous works in the literature. In [14], a bit parallel architecture for Galois field multipliers is proposed to improve the time performance of the multiplier block. Another hardware block design for Galois field multiplier is proposed in [15] using bit serial approach. Resource consumption and time performance of both designs are given in Table 2.1. Gate counts of the designs show that, total number of gates in the parallel architecture is proportional to m^2, whereas it is proportional to m in the serial case. On the other hand, when their latencies are compared, bit parallel design is far more advantageous then bit serial one.

The trade-off between area and time is improved by digit-serial architectures. In a digit serial multiplier m bit polynomials are fragmented into d bit digits, so it takes $c = \lceil m/d \rceil$ cycles to get an input. Similarly, the result is generated as d bit digits and output in c cycles. In [15], also a digit serial architecture for Galois Field multiplier block is proposed. Comparison of area and time performances of this design is also included in Table 2.1.

In Table 2.1, resource consumption and delay performances of designs with different input/output structures are compared. The results show that serial input/output structures provide better area utilization whereas parallel input/output structures have better time performance. Digit serial architectures provide a trade-off between these

two criteria.

In order to improve the time performance of parallel architectures, a pipelined structure can be used. Pipelining allows the hardware block to start processing a new input data before the output of previous data is produced. An illustration of pipelining is shown in Figure 2.8. Without using the pipeline technique, a hardware block processes only one input at once and does not receive the next data before the output of the currently processed data is produced. On the contrary, a pipelined hardware architecture can get a new input at each cycle. A pipelined hardware block has a structure composed of registers, where these registers define boundaries of the pipeline stages. After the incoming data is processed in the first stage and passed to the second one, the first stage is ready to process the next data. As a result, a new process begins before the previous one is finished. The first valid output is generated after a certain amount of time, called the latency of the circuit. Then, the subsequent outputs are obtained at each cycle as long as input data is supplied. Although it is acceptable to wait until the end of the previous process when there is not frequent input data, at high traffic rates pipelined architectures are far more advantageous, since it enables effective use of hardware resources. It is important to clarify that pipelining does not shorten the whole processing time for an input. But it provides high throughput at high input traffic conditions and much better performance is obtained overall, especially in high-speed systems.

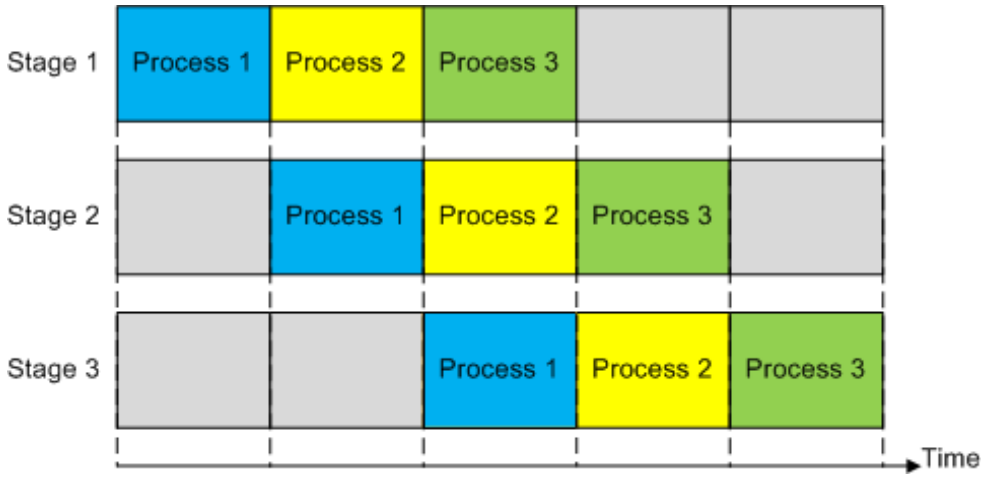


Figure 2.8: Operation of pipelined architecture

Galois field multiplier is the most time consuming block in an arithmetic unit. Considering the needs of Galois Field arithmetic applications, a fast multiplier block is a critical requirement. Therefore, a bit parallel design is considered as the most suitable architectural structure for $GF(2^m)$ multiplier block for the work conducted in this thesis.

2.5 MULTIPLICATION ALGORITHMS

Apart from the architectural structure, choosing the right multiplication algorithm is also an important criterion that affects the performance of multiplier block. Many hardware solutions for Galois Field multiplication with large numbers are proposed using different algorithms such as Karatsuba-Ofman, Mastrovito, Montgomery or classical school multiplication. Considering the application requirements, an algorithm is chosen in order to design high speed, low complexity or power efficient multiplier blocks. In this chapter, the most commonly used algorithms for hardware implementation of Galois Field multiplication are analyzed in further detail.

2.5.1 CLASSICAL MULTIPLICATION

One method of implementing Galois field multiplication block on hardware is using classical school multiplication algorithm (CM). CM is a common multiplication method used in hardware multiplier blocks. In this method, one multiplicand is first multiplied by each bit of the other multiplicand to get partial products. Each partial product is shifted by one and aligned according to the bit orders. Then the aligned partial products are summed up to obtain the product to be reduced.

Hardware implementation of this algorithm is illustrated in Figure 2.9 [16]. Each bit of two m-bit numbers are ANDed, resulting in m^2 2-bit AND operations. Then results of these AND operations are XORed according to their orders, requiring $(m-1)^2$ 2-bit XOR operations. As a result, $(2m-1)$-bit product is obtained, which is to be reduced using the characteristic polynomial of the field. Interpreting the gate counts, it is obvious that CM is not a suitable algorithm for large number multiplications since numbers of both AND and XOR gates increase dramatically as m gets larger.

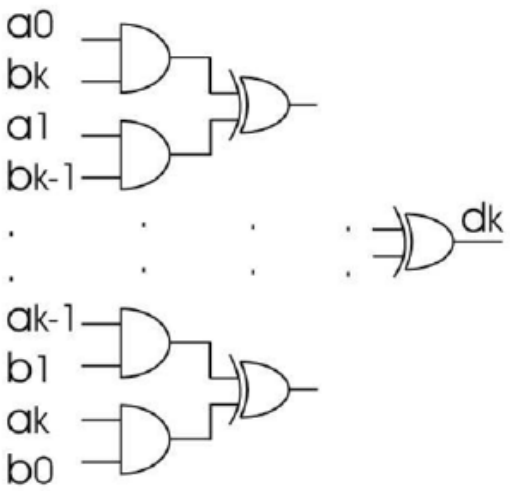


Figure 2.9: Implementation of CM on hardware (Zhou et al., 2010, p. 1058)

2.5.2 MONTGOMERY MULTIPLICATION

Another multiplication method is Montgomery multiplication algorithm. This algorithm is specifically used in $GF\left(2^m\right)$ arithmetic for cryptographic purposes where exponentiation is needed mostly [7]. It defines the Montgomery reduction algorithm to be used in modular multiplication operation. The algorithm suggests that multiplication is performed not using the actual inputs but their Montgomery representations so that no reduction operation is needed at the end of the multiplication operation.

Consider the operation $C\left(x\right) = A\left(x\right)B\left(x\right) mod\left(N\right)$. To calculate the product, C, this algorithm defines a constant, R, such that R satisfies the two conditions,

i. $R > N$

ii. $gcd(N, R) = 1$

Montgomery reduction produces $C'\left(x\right) * R^{-1} mod\left(N\right)$ for $C'\left(x\right) < RN$. As a result, reduction operation after the multiplication is eliminated.

Typically, $R = 2^k$ is chosen for simplicity, where k is any positive integer. Using the constant R, the numbers are first transformed into Montgomery representation as

follows, where A' and B' are Montgomery representations of A and B respectively.

$$A'(x) = A(x) * R$$
$$B'(x) = B(x) * R$$

Using these representations, the product in Montgomery representation, C', is calculated as follows, where $R^{-1} = 1/R \ mod(N)$.

$$\begin{aligned} C'(x) &= C(x) * R \, mod(N) \\ &= A(x) * B(x) * R \, mod(N) \\ &= (A(x) * R) * (B(x) * R) * R^{-1} \, mod(N) \\ &= A'(x) * B'(x) * R^{-1} \, mod(N) \end{aligned}$$

The obtained result can be used further in operations in Montgomery representation or can be inverse transformed to GF(2^m) representation as

$$C(x) = C'(x) * R^{-1} \, mod(N)$$

CM algorithm in $GF(2^m)$ depends on the idea of first multiplying the multiplicands and computing modulo characteristic polynomial afterwards. On the other hand, Montgomery multiplication method uses Montgomery reduction algorithm to handle the reduction operation during multiplication, through transformation to Montgomery representation and inverse transformation steps. As a result, significant reduction in area occupation is provided by eliminating reduction and performing two additional operations instead.

Once the multiplicands are transformed into Montgomery representation, performing multiple multiplication operations is possible before returning to $GF(2^m)$ representation. Therefore, this algorithm is mostly preferred for exponentiation instead of a single multiplication in order to make use of transformation step. This algorithm is very advantageous in large number exponentiation in $GF(2^m)$ arithmetic and widely used in many cryptographic applications. However, it is vulnerable to side channel attacks. It is possible to solve the internal working of the multiplier block by analyzing power consumption or some parameter changes [17].

2.5.3 KARATSUBA-OFMAN MULTIPLICATION

Karatsuba-Ofman multiplication (KOM) is another multiplication algorithm, which is the fastest among the algorithms introduced in this chapter and consumes less resources by reducing the computational complexity. In many cryptographic applications multiplication of large numbers is required, hence resulting in quite high resource consumption. KOM is a 2-way splitting multiplication algorithm. It is based on divide-and-conquer approach, that is, it breaks down the problem of large number multiplication into many smaller multiplications, which are simpler to calculate. Then, the results of sub-multiplications are combined to obtain the product of original multiplicands. This reduces the computational complexity of CM to $\mathcal{O}(n^{\log_2 3})$ [7].

KOM is not only used in $GF\left(2^m\right)$ arithmetic but is also applicable to other fields. To comprehend the idea in KOM algorithm consider the multiplication of two-term polynomials, $A\left(x\right) = a_1 x + a_0$ and $B\left(x\right) = b_1 x + b_0$, given in Equation 2.3.

$$\begin{aligned} D\left(x\right) &= A\left(x\right) * B\left(x\right) \\ &= \left(a_1 x + a_0\right) * \left(b_1 x + b_0\right) \\ &= \left(a_1 b_1\right) x^2 + \left(a_1 b_0 + a_0 b_1\right) x + \left(a_0 + b_0\right) \end{aligned} \tag{2.3}$$

The middle term in the above expression can be rewritten as follows.

$$a_1 b_0 + a_0 b_1 = \left(a_1 + a_0\right)\left(b_1 + b_0\right) - a_1 b_1 - a_0 b_0$$

The terms $a_1 b_1$ and $a_0 b_0$ in this expression are already computed. Therefore, the middle term in Equation 2.3 can be calculated by computing only $\left(a_1 + a_0\right)\left(b_1 + b_0\right)$. The advantage is that instead of performing four multiplications, number of sub-multiplications is reduced to three by performing two extra additions. Comparing their complexities, addition has better time and area performances than multiplication. So, it is more efficient to replace one multiplication by two additions. As a result, by applying KOM algorithm, one large multiplication operation is reduced to three smaller multiplications with a computational complexity smaller than the quadratic complexity of CM.

Algorithm 2.1: Karatsuba-Ofman Multiplication algorithm

Inputs : $A = A_0 + A_1 X^n$

$B = B_0 + B_1 X^n$

Output: D

1 $P_0 = A_0 B_0 = P_{0L} + P_{0H} X^n$

2 $P_1 = A_1 B_1 = P_{1L} + P_{1H} X^n$

3 $P_2 = (A_0 + A_1)(B_0 + B_1) = P_{2L} + P_{2H} X^n$

4 $R_0 = P_{0H} + P_{1L}$

5 $R_1 = P_{0L} + P_{2L}$

6 $R_2 = P_{1H} + P_{2H}$

7 $R_3 = R_0 + R_1$

8 $R_4 = R_0 + R_2$

9 $D = P_{0L} + R_3 X^n + R_4 X^{2n} + P_{1H} X^{3n}$

KOM algorithm is given in Algorithm 2.1. A visualization of KOM is given in Figure 2.10. In this figure, the hardware block is divided into three stages, i.e., splitting stage, sub-multiplication stage and recombination stage. Splitting stage is the first stage where multiplicands are partitioned into two and inputs to sub-multipliers are computed. Then comes the sub-multiplication stage, where the partitions are multiplied. In this stage, it is possible to use any multiplication algorithm. A recursive implementation can be done by using KOM in the sub-multiplication stage. After some number of multiplications, result of a sub-multiplication can be obtained using an algorithm like CM. Sub-multiplications can be done in parallel since they are independent of each other, so that all partial products are obtained at the same instant. After obtaining the partial products in the second stage, results are combined to obtain the result of the multiplication operation in the recombination stage.

In finite field arithmetic one more step is needed to compute the actual result. The product obtained by combination of partial products from sub-multipliers should be reduced by computing modulo the characteristic polynomial of the result.

Similar to KOM, various algorithms based on divide-and-conquer approach exist, which aim at splitting the large multiplicands into smaller partitions and combining them later, in order to reduce the computation complexity. Some of these algorithms,

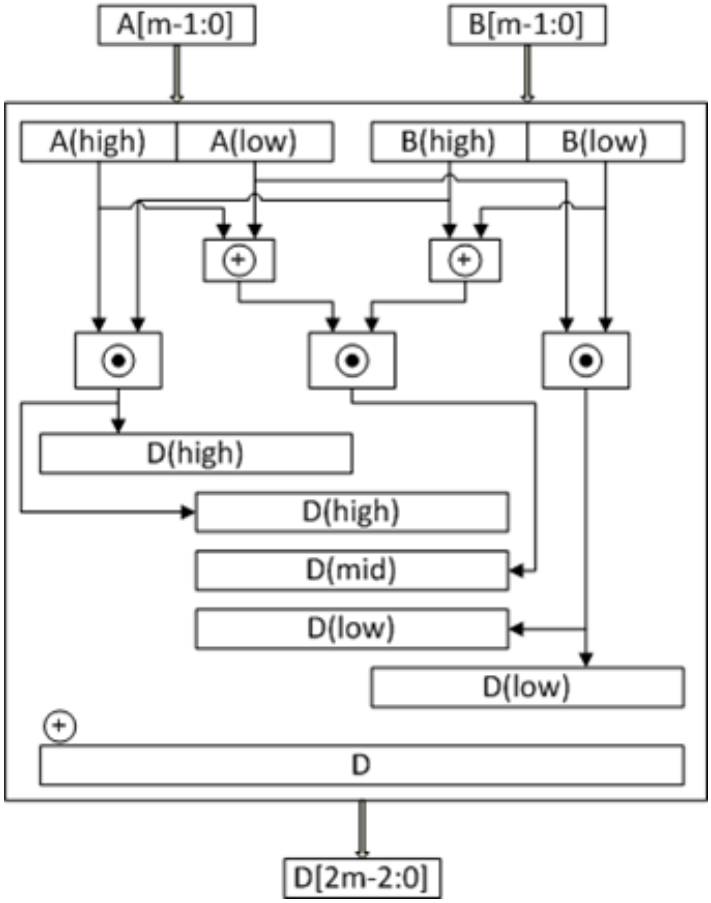


Figure 2.10: Implementation of KOM in $GF\left(2^m\right)$

which splits the multiplicands into 2, 3, 4 or 5 partitions, are used for the work conducted in this thesis to implement a low cost $GF\left(2^m\right)$ multiplier.

CHAPTER 3

LITERATURE WORK

Numerous works exists in the literature on the subject of design and implementation of Galois field multiplier for cryptographic applications. Many works concentrate on hardware implementation of Galois Field multiplier blocks for their ability to perform high-speed operation with low resource consumption, which is the most critical requirement of cryptographic applications. Various design styles and multiplication algorithms are proposed to achieve fast multiplication with low area complexity. Different design styles including systolic and non-systolic architectures, bit-parallel and bit-serial input/output structures are implemented on hardware platforms to provide better performance for Galois field multipliers [13]. Although design style is very effective on resource utilization and delay of hardware block, its performance is primarily determined by the implemented algorithm.

In this section, some recent works on Galois field multiplication are analyzed. Effects of preferred multiplication algorithms and their implementation are discussed in detail. Since both area and time performances are considered, algorithms that aim to reduce operation complexity with reasonable delay are studied, which are mainly splitting algorithms. Also Galois Field multiplier designs for specific applications are examined. In addition to $GF\left(2^m\right)$ multiplier designs and applications, use of partial reconfiguration on FPGAs in different designs is overviewed with analysis of recent works in literature.

3.1 HARDWARE IMPLEMENTATION OF KARATSUBA-OFMAN MULTIPLICATION

KOM is the first splitting algorithm introduced in the literature. By splitting the operands into two partitions and combining them after multiplication, it reduces the quadratic complexity of multiplication to $\mathcal{O}(n^{\log_2 3})$. Numerous works have been conducted in literature that aim to find an efficient way to implement a multiplier block using KOM algorithm for $GF\,(2^m)$ multiplication.

The most detailed implementation complexity analysis of KOM is done in [16]. In this work, recursive implementation of KOM on different hardware platforms is analyzed through experiments. Implementation platforms, on which the experiments are conducted, are ASIC, Virtex 4, a 4-input LUT based FPGA, and Virtex 5, a 6-input LUT based FPGA. The aim of the work is to compare area consumptions of multipliers that are implemented using KOM with different recursion depths implemented on three types of hardware platforms. On the last step of recursion, sub-multiplications are done using CM algorithm and recursion depth 0 implies that only one CM is performed.

Before getting the results for comparison, first the algorithm is optimized for odd-term and even-term polynomials. Binary fields given in the first column of Table 3.1, which are commonly used for cryptographic applications, are chosen for experiments. Then delay and area complexities are computed for all three implementation platforms at every recursion depth. Implementation results for two different FPGAs are obtained using two different synthesis tools, Xilinx ISE 10.1 and Synplify Pro 9.0.1, with enabling and disabling "Keep Hierarchy" option. The optimum results for area complexity on 4-input LUT based and 6-input LUT based FPGAs obtained in [16] using Xilinx ISE 10.1 tool are also included in Table 3.1.

The results obtained in [16] provide a guide for implementation of KOM on three different hardware platforms. The optimum solutions for $GF\,(2^m)$ implementations for commonly used m values are specified as a result of area and time complexity analysis. Effects of different hardware platforms, synthesis tools and coding styles are also examined using the complexity analysis results.

Table 3.1: KOM implementation results for FPGAs obtained in [16]

	Virtex 4		Virtex 5	
Galois Field	Recursion Depth	Number of LUTs	Recursion Depth	Number of LUTs
$GF\left(2^{113}\right)$	4	4613	4	3727
$GF\left(2^{128}\right)$	4	5520	4	4447
$GF\left(2^{163}\right)$	4	8624	4	6823
$GF\left(2^{193}\right)$	5	11060	4	8989
$GF\left(2^{233}\right)$	5	15154	5	12243
$GF\left(2^{283}\right)$	5	20321	5	16568

3.2 SPLITTING ALGORITHMS

Following introduction of 2-way splitting algorithm KOM to reduce the computational complexity of multiplication, several other algorithms based on splitting the multiplicands into partitions are proposed. 3-way, 4-way and 5-way splitting algorithms that appeared in literature are discussed in the present chapter. Their computational complexities and delay complexities of which are given in Table 3.2.

Examples of 3-way split algorithms for $GF\left(2^m\right)$ multiplication in literature are Bernstein's 3-way split algorithm proposed in [18] and Karatsuba-like improved 3-way split algorithm proposed in [19]. These algorithms are presented in Algorithm 3.1 and Algorithm 3.2, respectively. Bernstein's 3-way split algorithm reduces the computational complexity to $\mathcal{O}(n^{1.46})$. However, it requires performing a division operation to compute the product. So, this algorithm is not suitable for hardware implementation since division is performed by multiple multiplications and is not effective in terms of both area and time. On the other hand, Karatsuba-like improved 3-way split algorithm has a computational complexity of $\mathcal{O}(n^{1.63})$, which is higher compared to computational complexity of Bernstein's 3-way split algorithm, but still less than the quadratic complexity of CM. Plus, it has a logarithmic delay complexity that is far better than that of Bernstein's 3-way split algorithm. Therefore, Karatsuba-like improved 3-way split algorithm is chosen as the 3-way split algorithm to be implemented on hardware platform throughout the experiments conducted in this thesis.

Table 3.2: Computational and delay complexities of splitting algorithms

Algorithm	Computational Complexity	Delay Complexity
Bernstein 3-way split	$25.5n^{1.46} - 25.5n + 1$	$(1.5n + 8log_3(n))\, D_X + D_A$
Karatsuba 3-way split	$5.8n^{1.63} - 6n + 1.2$	$4log_3(n)\, D_X + D_A$
Bernstein 4-way split	$6.425n^{1.58} - 6.8n + 1.375$	$5log_4(n)\, D_X + D_A$
Cenk 5-way split	$6.5n^{1.5} - 7n + 1.5$	$9log_5(n)\, D_X + D_A$

In addition to 3-way split algorithms, a 4-way split algorithm, namely Bernstein's 4-way split algorithm, is proposed in [18] and a 5-way split algorithm, namely Cenk 5-way split algorithm, is proposed in [20]. These algorithms are presented in Algorithm 3.3 and Algorithm 3.4, respectively. Bernstein's 4-way split algorithm reduces the computational complexity of multiplication to $\mathcal{O}(n^{1.58})$. Computational complexity of Cenk 5-way split algorithm is $\mathcal{O}(n^{1.5})$. Both algorithms have subquadratic computational complexities and logarithmic delay complexities. Therefore, they are suitable for hardware implementation with efficient area utilization and reasonable delay values.

Algorithm 3.1: Bernstein's 3-way split algorithm

Inputs : $A = A_0 + A_1X^n + A_2X^{2n}$

$B = B_0 + B_1X^n + B_2X^{2n}$

Output: $C = AB$

1 $P_0 = A_0B_0$

2 $P_1 = (A_0 + A_1 + A_2)(B_0 + B_1 + B_2)$

3 $P_2 = (A_0 + A_1X + A_2X^2)(B_0 + B_1X + B_2X^2)$

4 $P_3 =$

$((A_0 + A_1 + A_2) + (A_1X + A_2X^2))((B_0 + B_1 + B_2) + (B_1X + B_2X^2))$

5 $P_4 = A_2B_2$

6 $U = P_0 + (P_0 + P_1)X^n$

7 $V = P_2 + (P_2 + P_3)(X^n + X)$

8 $C = U + P_4(X^{4n} + X^n) + \frac{(U+V+P_4(X^4+X))(X^{2n}+X^n)}{(X^{2n}+X)}$

Algorithm 3.2: Karatsuba-like improved 3-way split algorithm

Inputs : $A = A_0 + A_1X^n + A_2X^{2n}$

$B = B_0 + B_1X^n + B_2X^{2n}$

Output: $C = AB$

1 $P_0 = A_0B_0 = P_{0L} + P_{0H}X^n$

2 $P_1 = A_1B_1 = P_{1L} + P_{1H}X^n$

3 $P_2 = A_2B_2 = P_{2L} + P_{2H}X^n$

4 $P_3 = (A_1 + A_2)(B_1 + B_2) = P_{3L} + P_{3H}X^n$

5 $P_4 = (A_0 + A_1)(B_0 + B_1) = P_{4L} + P_{4H}X^n$

6 $P_5 = (A_0 + A_2)(B_0 + B_2) = P_{5L} + P_{5H}X^n$

7 $R_0 = P_{0H} + P_{1L}$

8 $R_1 = R_0 + P_{0L}$

9 $R_2 = R_1 + P_{4L}$

10 $R_3 = P_{1H} + P_{2L}$

11 $R_4 = R_1 + R_3$

12 $R_5 = P_{4H} + P_{5L}$

13 $R_6 = R_4 + R_5$

14 $R_7 = R_3 + P_{2H}$

15 $R_8 = R_7 + R_0$

16 $R_9 = R_8 + P_{3L}$

17 $R_{10} = R_9 + P_{5H}$

18 $R_{11} = R_7 + P_{3H}$

19 $C = P_{0L} + R_2X^n + R_6X^{2n} + R_{10}X^{3n} + R_{11}X^{4n} + P_{2H}X^{5n}$

The computational complexity and delay complexity values given in Table 3.2 are theoretical results obtained by mathematical calculations. In [21], numerous algorithms including the ones introduced in this chapter are analyzed theoretically to obtain computational and delay complexities. In order to obtain the minimum complexity, the algorithm giving the optimum result is used for each multiplication after splitting rather than using the same algorithm recursively. The results obtained in [21] give meaningful results for ASIC implementation since mathematical expressions are directly converted into logic gates. However, for FPGA implementation, these results are not valid. An experimental work to compute area and time complexities is needed that is similar to the analysis in [16] in order to determine the most efficient algorithms for different field sizes. The present thesis aims to conduct such an analysis and obtain an area optimized $GF\left(2^m\right)$ multiplier for FPGA. The results of this effort is presented in Chapter 4.

Algorithm 3.3: Bernstein 4-way split algorithm

Inputs : $A = A_0 + A_1X^n + A_2X^{2n} + A_3X^{3n}$

$B = B_0 + B_1X^n + B_2X^{2n} + B_3X^{3n}$

Output: $C = AB$

1 $P_0 = A_0B_0$

2 $P_1 = A_1B_1$

3 $P_2 = A_2B_2$

4 $P_3 = A_3B_3$

5 $P_4 = (A_0 + A_1)(B_0 + B_1)$

6 $P_5 = (A_2 + A_3)(B_2 + B_3)$

7 $P_6 = (A_0 + A_2 + (A_1 + A_3)X^n)(B_0 + B_2 + (B_1{+}_3)X^n)$

8 $C = P_0 + (P_0 + P_1 + P_4)X^n + (P_0 + P_1 + P_2 + P_6)X^{2n} +$
$(P_0 + P_1 + P_2 + P_3 + P_4 + P_5)X^{3n} + (P_1 + P_2 + P_3)X^{4n} +$
$(P_2 + P_3 + P_5)X^{5n}$

Algorithm 3.4: Cenk improved 5-way split algorithm

Inputs : $A = A_0 + A_1X^n + A_2X^{2n} + A_3X^{3n} + A_4X^{4n}$

$B = B_0 + B_1X^n + B_2X^{2n} + B_3X^{3n} + B_4X^{4n}$

Output: $C = AB$

1 $m_0 = A_0B_0 = p_1 + p_2X^n$

2 $m_1 = A_1B_1 = p_3 + p_4X^n$

3 $m_2 = A_2B_2 = p_5 + p_6X^n$

4 $m_3 = A_3B_3 = p_7 + p_8X^n$

5 $m_4 = A_4B_4 = p_9 + p_{10}X^n$

6 $m_5 = (A_0 + A_1)(B_0 + B_1) = p_{11} + p_{12}X^n$

7 $m_6 = (A_0 + A_2)(B_0 + B_2) = p_{13} + p_{14}X^n$

8 $m_7 = (A_2 + A_4)(B_2 + B_4) = p_{15} + p_{16}X^n$

9 $m_8 = (A_3 + A_4)(B_3 + B_4) = p_{17} + p_{18}X^n$

10 $m_9 = (A_0 + A_2 + A_3)(B_0 + B_2 + B_3) = p_{19} + p_{20}X^n$

11 $m_{10} = (A_1 + A_2 + A_4)(B_1 + B_2 + B_4) = p_{21} + p_{22}X^n$

12 $m_{11} = (A_0 + A_1 + A_3 + A_4)(B_0 + B_1 + B_3 + B_4) = p_{23} + p_{24}X^n$

13 $m_{12} = (A_0 + A_1 + A_2 + A_3 + A_4)(B_0 + B_1 + B_2 + B_3 + B_4) = p_{25} + p_{26}X^n$

14 $t_1 = p_1 + p_2$, $t_4 = p_4 + p_5$, $t_5 = p_{12} + p_{13}$, $t_9 = p_6 + p_7$, $t_{12} = p_{14} + p_{15}$, $t_{14} = p_{19} + p_{23}$, $t_{17} = p_8 + p_9$, $t_{22} = p_{16} + p_{17}$, $t_{25} = p_{20} + p_{21}$, $t_{26} = p_{25} + p_{26}$, $t_{27} = p_{19} + p_{24}$, $t_{33} = p_{22} + p_{23}$, $t_{36} = p_{22} + p_{24}$

15 $t_2 = t_1 + p_3$, $t_6 = t_4 + t_5$, $t_8 = t_1 + t_4$, $t_{15} = t_{14} + p_{25}$, $t_{18} = t_{17} + p_{10}$, $t_{28} = t_{25} + t_{26}$, $t_{37} = t_{12} + p_{26}$, $t_{38} = t_{36} + p_{10}$

16 $t_3 = t_2 + p_{11}$, $t_7 = t_2 + t_6$, $t_{10} = t_8 + t_9$, $t_{19} = t_{18} + p_{18}$, $t_{21} = t_{18} + t_{20}$, $t_{29} = t_{28} + t_{27}$, $t_{39} = t_{37} + t_{38}$

17 $t_{11} = t_{10} + p_9$, $t_{23} = t_{21} + t_{22}$, $t_{31} = t_7 + t_{19}$

18 $t_{13} = t_{11} + t_{12}$, $t_{24} = t_{23} + t_3$, $t_{32} = t_{28} + t_{31}$, $t_{35} = t_{11} + p_1$

19 $t_{16} = t_{13} + t_{15}$, $t_{30} = t_{29} + t_{24}$, $t_{34} = t_{32} + t_{33}$, $t_{40} = t_{35} + t_{39}$

20 $C = p_1 + t_3X^n + t_7X^{2n} + t_{16}X^{3n} + t_{30}X^{4n} + t_{34}X^{5n} + t_{40}X^{6n} + t_{23}X^{7n} + t_{19}X^{8n} + p_{10}X^{9n}$

Table 3.3: Cryptographic algorithms and security levels

Algorithm	Operation	Status
DES	Encryption	Avoid
3DES	Encryption	Legacy (Short key lifetime)
RC4	Encryption	Avoid
AES-CBC	Encryption	Acceptable
AES-GCM	Authenticated encryption	Next Generation Encryption
RSA-768	Encryption	Avoid
RSA-1024	Encryption	Avoid
RSA-2048	Encryption	Acceptable

3.3 APPLICATIONS OF GALOIS FIELD MULTIPLICATION

Galois field multiplication is most popular for its use in cryptographic applications. Almost all cryptographic algorithms used for Internet security today is based on Galois field arithmetic. In Table 3.3 encryption algorithms recommended for Internet security by Cisco [6] are given. In the table, status of algorithms are designated as "avoid" for algorithms that are not able to provide adequate security against threats and should not be used, "legacy" for algorithms that can be used only if no alternative exists since they do not provide a high level of security, "acceptable" for algorithms that provide adequate security and "next generation encryption" (NGE) for algorithms that provide high security level, which is expected to be adequate for next 20 years. The table indicates that, the most secure encryption algorithm for Internet security is AES-GCM, status of which is designated as NGE.

Galois Counter Mode (GCM) is defined as a mode of operation[1] that uses universal hashing over a binary field to provide authenticated encryption in [22]. GCM can be used as a stand-alone Message Authentication Code (MAC) to provide only authentication, as well as with a block cipher such as AES to provide authenticated encryption. It is capable of performing at speeds more than 10 Gbps when implemented on hardware, meeting the security requirements of high-speed systems. AES-GCM is defined as the default cipher suite in IEEE 802.1AE Media Access Control Security standard [23] and also used in 802.11AD Wireless Gigabit Alliance [24].

[1] A mode of operation is an algorithm that defines repeated application of a single block cipher in order to provide security and authenticity.

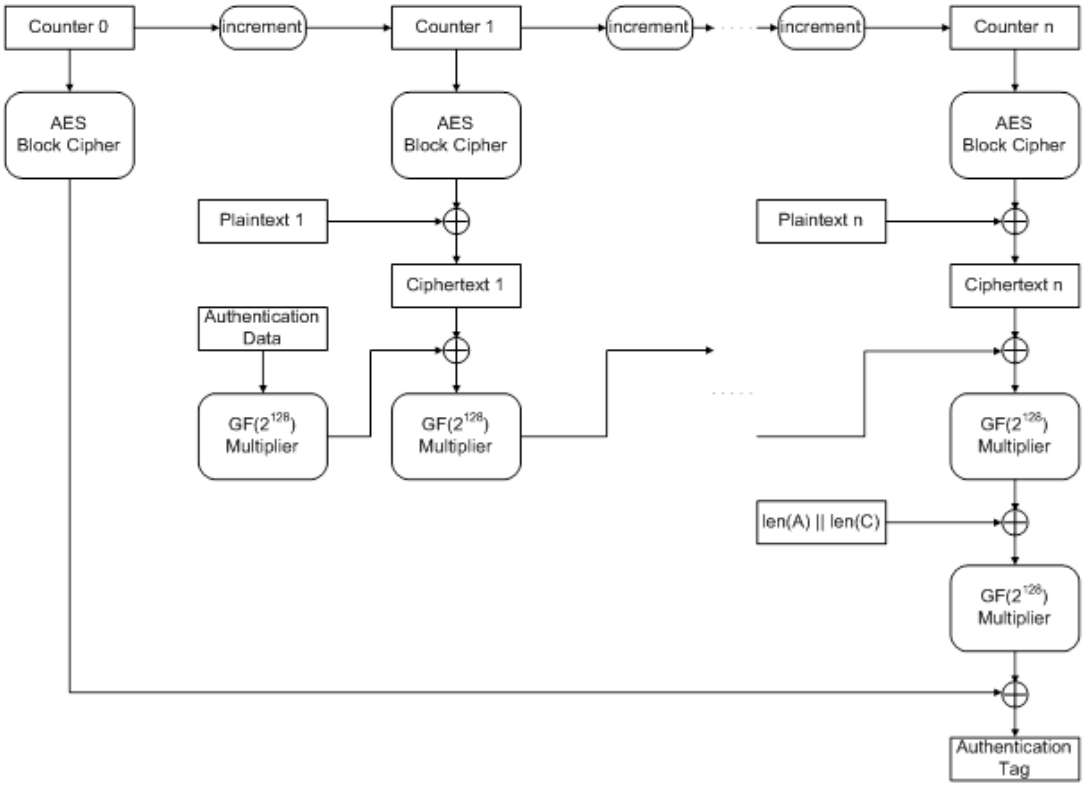


Figure 3.1: AES-GCM operational block diagram

AES-GCM core consists of two main blocks. The first one is the AES block, where encryption or decryption operation is performed. The second part is GHASH module, which operates as the authentication block. A block diagram of AES-GCM encryption module operation is shown in Figure 3.1.

The encryption module takes four inputs:

i. K, the encryption key,

ii. IV, the initialization vector,

iii. P, the plaintext, which is the data to be encrypted,

iv. A, the additional authenticated data, which is authenticated but not encrypted. This data generally includes network protocol related information like address, ports, protocol version, etc.

Outputs of the encryption module are:

i. C, the ciphertext,

ii. T, the authentication tag, which is obtained as the output of the GHASH function.

Operation of decryption module is similar. It takes five inputs, which are K, IV, C, A and T, to produce single output P. If there exists an authentication error, a failure signal is produced as the output of decryption operation instead of the plaintext.

Numerous works have been conducted to design and implement Galois field multiplier block for AES-GCM. In [25], a pipelined AES-GCM core optimized for FPGA implementation is designed. The main work done in [25] is the analysis of area and time complexities of Galois field multiplier block to be used in the AES-GCM core implemented on the selected FPGA, Virtex 4. Area complexity is given in terms of number of LUTs and delay is measured in terms of LUT delay. Improving both area and time performances is aimed in [25]. In order to design a speed efficient Galois field multiplier, CM is preferred as the multiplication algorithm. The multiplier block implemented using CM is named speed-efficient multiplier. In addition to the speed-efficient multiplier, an area-efficient multiplier is implemented. Recursive KOM is preferred as the multiplication algorithm for the area-efficient multiplier. In order to implement this multiplier block, area and time complexities at all recursion depths of KOM are compared and recursion depth 4 is chosen as the optimum result with an area utilization of 5523 LUTs. Then, both multipliers are implemented within AES-GCM core to obtain the results for total resource consumption and delay. Area and time performances of the multipliers designed in [25] on Virtex 4 FPGA are presented in Table 3.4.

Table 3.4: Time and area performances of multiplier blocks designed in [25]

Multiplier	**Resource consumption (LUT)**	**Delay (T_{LUT})**
Area-efficient (4-step KOM)	5523	8
Speed-efficient (CM)	10923	4

In [26], an AES-GCM core design on FPGA that supports 100 Gbps bandwidth is designed. Similar to the work conducted in [25], a multiplier design for hash function is implemented using recursive KOM. In this design, 4 independent multipliers operate in parallel to realize GHASH function. For every single multiplier block, although area complexity is not optimized, 2-step KOM is preferred as the multiplication algorithm in order to reduce the area complexity without increasing the total delay much. Pipelining technique is also included in multiplier design. KOM based multiplier is partitioned into 4 pipeline stages, at the splitting stages and recombination stages of two recursive multipliers. As a result, a multiplier design that can support 100Gbps Ethernet speed for an AES-GCM core is obtained. Area utilization is expressed in terms of number of occupied slices, which is equal to 14799 for Virtex 5 XC5VLX220 FPGA.

Another work that concentrates on design of Galois field multiplier for GHASH core on FPGA is [27]. Similar to [26], this design also includes 4 multipliers based on 2-step recursive KOM in order to realize GHASH function on Virtex 5 FPGA. The difference between [26] and [27] is the use of reduction block. [26] implements one reduction block per multiplier for all four multipliers used in GHASH function, whereas in [27], a single reduction block is used to reduce the results from all multipliers to obtain the actual product. Since the multipliers are used in a pipelined manner, this improvement introduced in [27] does not affect the speed of the whole design. But, it reduces area utilization by eliminating three reduction blocks and provides better area performance than the design in [26].

Many works on design and implementation of a $GF\left(2^{128}\right)$ multiplier block for AES-GCM core in literature use KOM as the multiplication algorithm for its area efficiency and reasonable delay. Another reason it is preferred in these designs is that applying pipelining technique with KOM is very straightforward due to its structure based on independent stages. So, it is possible to use KOM based multipliers in high speed systems to provide high speed operation, while reducing area utilization.

Table 3.5: Throughput and area performances in [29] with and without partial reconfiguration

Implementation	Throughput (Gbps)	Number of slices
Without partial reconfiguration	8.733	7215
With partial reconfiguration	24.922	3576

3.4 FPGA DESIGNS USING PARTIAL RECONFIGURATION

Numerous works exist in the literature that uses partially reconfigurable hardware designs in various areas including image processing, communication networks, cryptography, space applications and many others. Although a work on Galois filed multiplier design using partial reconfiguration does not exist in the literature, there are examples of its use in implementation of some cryptographic applications such as [28], which uses partial reconfiguration for implementation of AES and IDEA algorithms on FPGA, and [29], which focuses on AES implementation on FPGA with partial reconfiguration.

As an example of using partial reconfiguration in cryptography, in [29], AES algorithm is implemented on Virtex 2 FPGA, where key expansion phase is implemented as a partially reconfigurable module. In key expansion step of AES algorithm, a new round key block is generated to be used in each round of the algorithm. In [29], this phase is realized on software and resulting round keys are stored on FPGA by reconfiguration at run time. As a result area utilization of FPGA is reduced by about 50%. Using partial reconfiguration to store the round keys instead of calculating them on FPGA also improves the speed of the whole design and throughput is increased from 8.73 Gbps to 24.92 Gbps. The results obtained in this work are given in Table 3.5.

Many other works using partial reconfiguration exist in the literature in various application areas. Partial reconfiguration obviously provides efficiency in resource utilization. Plus, it can be included in designs in order to provide higher throughput as in the examples of [28] and [29], increase speed or add functionality to the hardware. By including this feature in FPGA based hardware designs, it is possible to make more flexible and efficient implementations.

CHAPTER 4

AREA OPTIMIZED $GF\left(2^M\right)$ MULTIPLIER IMPLEMENTATION ON FPGA

Two main objectives in design and implementation of a Galois field multiplier block on a hardware platform are achieving short delay and resource utilization efficiency. The most popular method for achieving fast multiplication is using look up tables. But, this method is inefficient for large number multiplications. Therefore, it is not useful for cryptographic applications like AES-GCM, where 128-bit numbers are multiplied. Another method for fast multiplication is using CM algorithm. However, it has quadratic computational complexity, thus results in high resource consumption. In order to overcome the problem of high resource consumption in fast multiplication methods, area optimized Galois field multipliers for fields commonly used in cryptographic applications are designed in this chapter. While reducing area usage, total delay of operation is also considered. Splitting algorithms are used in order to reduce resource utilization, without excessively increasing time complexity.

In many cryptographic applications Galois field multiplication units are designed using recursive KOM algorithm in order to find an optimum point for area-time trade off, since KOM divides the large number multiplication into smaller multiplications, thus reduces area complexity, by introducing a reasonable delay. Many works in literature concentrate on optimizing recursive KOM based multipliers on various hardware platforms, including different FPGA types. However, other splitting algorithms such as Karatsuba-like improved 3-way split algorithm, Bernstein 4-way split algorithm and Cenk improved 5-way split algorithm are analyzed only mathematically. The theoretical analysis for computational complexity provides valid results for ASIC

implementations, especially about area complexity, which is generally based on the number of gates used. Number of bit multiplications is equal to number of AND gates used in an ASIC design, and number of bit additions gives the number of XOR gates. Time complexity is also computed in a similar way. However, LUT based structure of FPGAs makes it impossible to compute area and time complexities exactly without experimental results. To the best of our knowledge, implementation and complexity analysis of 3-, 4- and 5-way splitting algorithms on FPGA does not exist in the literature. Therefore, a complexity analysis of these algorithms on selected FPGAs are made in order to design area optimized Galois field multiplier blocks for fields commonly used in cryptographic applications.

In [21], splitting algorithms are examined theoretically. Theoretical analysis results are obtained using seven different algorithms. But, as mentioned before, these results are not valid for FPGA implementation. In this thesis, four algorithms are selected out of seven algorithms included in [21], which are suitable for hardware implementation and provide low area complexity with subexponential delay. Area and time complexities are analyzed experimentally for selected hardware implementation platforms. The algorithms are KOM (Algorithm 2.1), Karatsuba-like improved 3-way split algorithm (Algorithm 3.2), Bernstein 4-way split algorithm (Algorithm 3.3) and Cenk 5-way split algorithm (Algorithm 3.4).

In [16], KOM is optimized for even and odd term multiplicands. In this thesis, all splitting algorithms used in experiments are optimized in a similar manner. For example, 3-way splitting algorithm is not applied only to $3n$-bit multiplicands. It is also applied to $(3n-1)-$ and $(3n-2)-$bit long polynomials. A method of applying this algorithm to $(3n-k)-$bit long polynomials, where k is equal to 1 or 2 in this case, is to extend the polynomial lengths to $3n$ bits by zero padding the most significant bits. But, instead of zero padding and increasing the length of the multiplicands, splitting the multiplicands into partitions of unequal lengths is a better method in terms of area complexity. In such a case, the most significant partition is shorter with a length of $(n-k)$ bits than the other two partitions, which are n bits long.

Selected splitting multiplication algorithms are implemented for the Galois fields examined in [16], which are given in Table 4.1. Method of implementation is as follows:

Table 4.1: Galois fields used for experiments with characteristic polynomials

Galois Field	**Characterstic Polynomial**
$GF\left(2^{113}\right)$	$f\left(x\right) = x^{113} + x^9 + 1$
$GF\left(2^{128}\right)$	$f\left(x\right) = x^{128} + x^8 + x^7 + x^2 + x + 1$
$GF\left(2^{163}\right)$	$f\left(x\right) = x^{163} + x^7 + x^6 + x^3 + 1$
$GF\left(2^{193}\right)$	$f\left(x\right) = x^{193} + x^{15} + 1$
$GF\left(2^{233}\right)$	$f\left(x\right) = x^{233} + x^{74} + 1$
$GF\left(2^{283}\right)$	$f\left(x\right) = x^{283} + x^{12} + x^7 + x^5 + 1$

For all selected fields, all possible field sizes for submultiplications using 2-, 3-, 4- and 5-way splitting are generated. For example, for the field size 113, 113-bit multiplicands are splitted into smaller partitions to find the field sizes of submultiplications needed for every splitting algorithm examined in this work. For KOM algorithm, which requires splitting the multiplicands into two, submultiplications for field sizes 57 and 56 are needed. For 3-way split algorithm, submultiplications for field sizes 38 and 37 are needed. For 4-way split algorithm, submultiplications for field sizes 58, 29 and 26 are needed. For 5-way split algorithm, submultiplications for field sizes 23 and 21 are needed. To find the most area efficient algorithm to implement these submultiplier blocks, the same splitting approach is applied to these resulting field sizes, until no more splitting is possible. Then, all selected multiplication algorithms including CM, which does not require any splitting, are implemented for these field sizes beginning from the smallest one. As the most area efficient algorithm for a field size is found, that submultiplier is implemented to be used in a larger field multiplication. For example, to implement KOM for field size 113, the most efficient multiplication algorithms for field sizes 57 and 56 should be known, and so on. So, multiplier blocks for field sizes 57 and 56 should be implemented in order to realize KOM for field size 113.

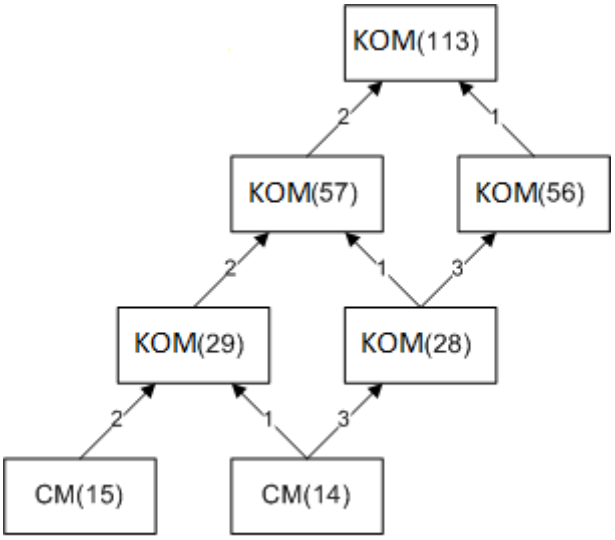
4.1 IMPLEMENTATION RESULTS

Galois fields selected for the experiments are given in Table 4.1. All multiplier blocks are implemented using Xilinx ISE Design Suite 14.7 with "Keep Hierarchy" option disabled. In Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6, algorithms to obtain minimum area

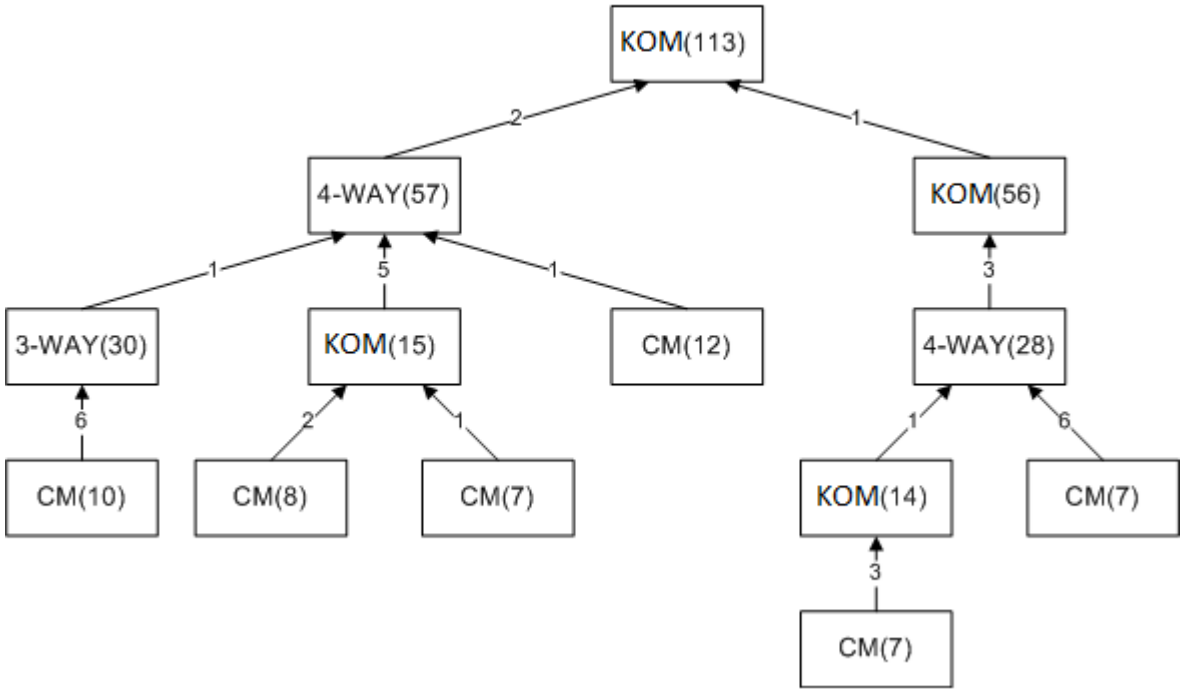
utilization for the fields in Table 4.1 are visualized for Virtex 5 and Kintex 7 FPGA platforms. Implementation results for all field sizes are included in the Appendix part.

In the Figures 4.1 - 4.6, each box represents a multiplication operation, indicating the algorithm preferred in that step and length of the multiplicands. The initial multiplication is given in the box at the top. For example, in Figure 4.1-(a), multiplication algorithms to implement multiplier block for field size 113 are given. Then it is partitioned into smaller multiplications according to the algorithm used at that step. For Figure 4.1, the most efficient multiplication algorithm for field size is indicated to be KOM, so it is partitioned into fields $GF\left(2^{57}\right)$ and $GF\left(2^{56}\right)$. The arrows connecting the boxes at each level indicates the resulting multiplications after splitting. The numbers on the arrows indicates how many multiplier blocks are needed at the next splitting stage. For the same figure,two $GF\left(2^{57}\right)$ multiplier blocks and one $GF\left(2^{56}\right)$ multiplier block are needed to realize one $GF\left(2^{113}\right)$ multiplier block using KOM algorithm.

The figures show that algorithms used for implementation of area optimized multiplier blocks on two different FPGAs are not the same. Moreover, the algorithms do not match with the ones that are theoretically the most efficient in terms of area utilization.

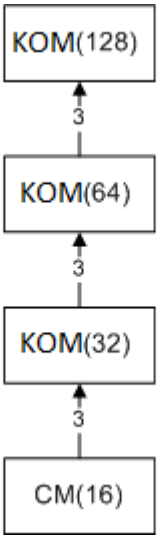


(a) Area optimized $GF\left(2^{113}\right)$ multiplier for Virtex 5

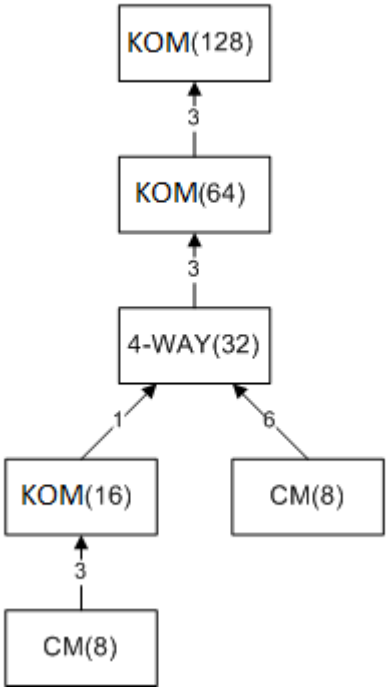


(b) Area optimized $GF\left(2^{113}\right)$ multiplier for Kintex 7

Figure 4.1: Algorithms selected for area optimized $GF\left(2^{113}\right)$ multiplier for Virtex 5 and Kintex 7

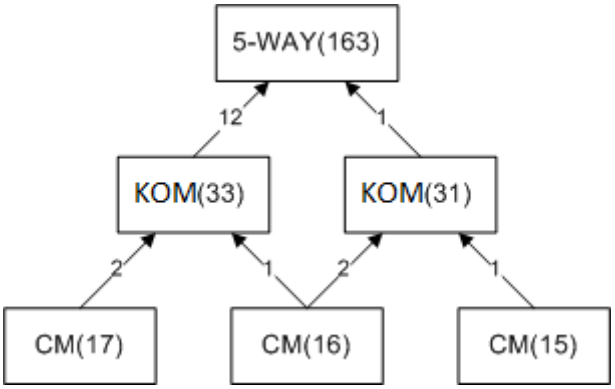


(a) Area optimized $GF\left(2^{128}\right)$ multiplier for Virtex 5

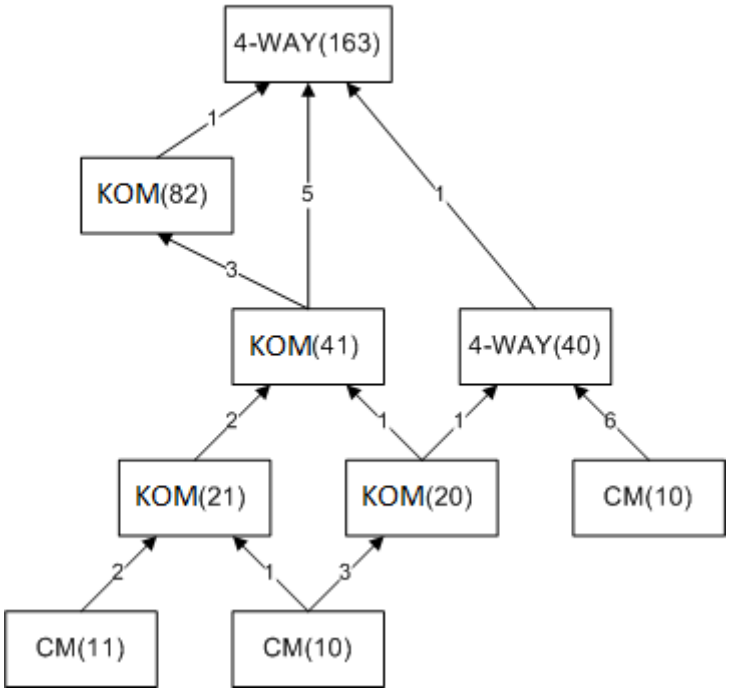


(b) Area optimized $GF\left(2^{128}\right)$ multiplier for Kintex 7

Figure 4.2: Algorithms selected for area optimized $GF\left(2^{128}\right)$ multiplier for Virtex 5 and Kintex 7

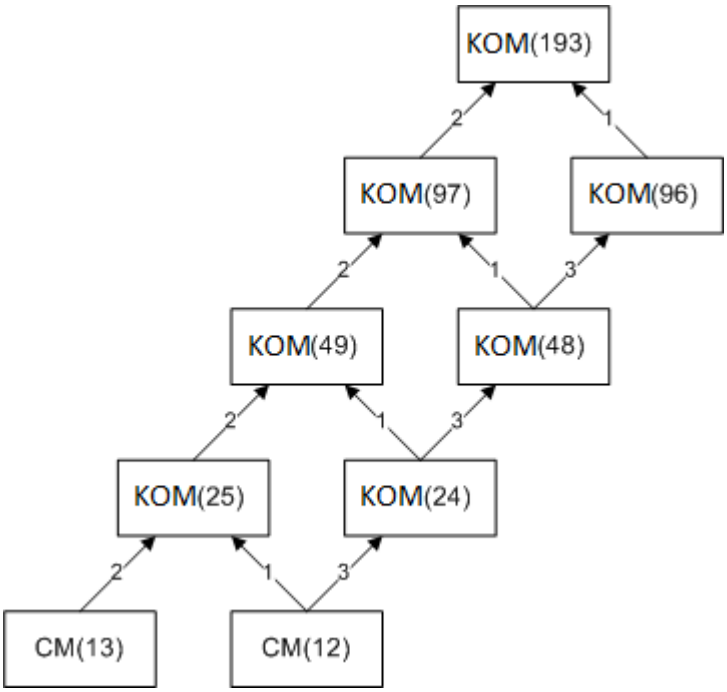


(a) Area optimized $GF\left(2^{163}\right)$ multiplier for Virtex 5

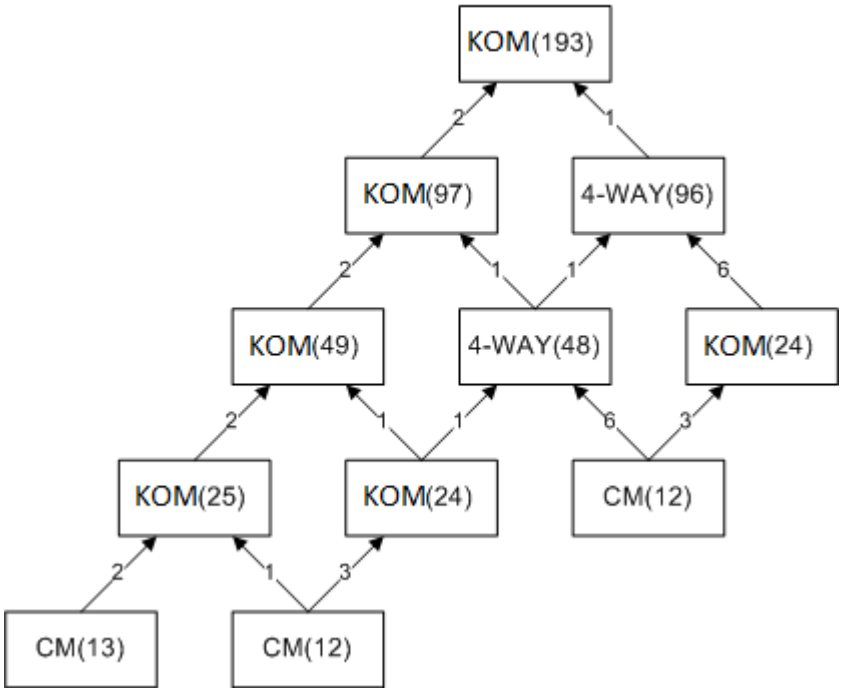


(b) Area optimized $GF\left(2^{163}\right)$ multiplier for Kintex 7

Figure 4.3: Algorithms selected for area optimized $GF\left(2^{163}\right)$ multiplier for Virtex 5 and Kintex 7

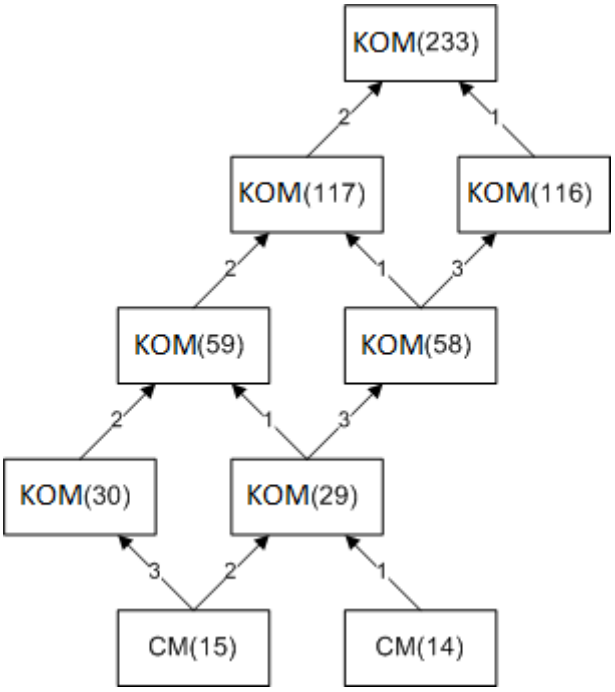


(a) Area optimized $GF\left(2^{193}\right)$ multiplier for Virtex 5

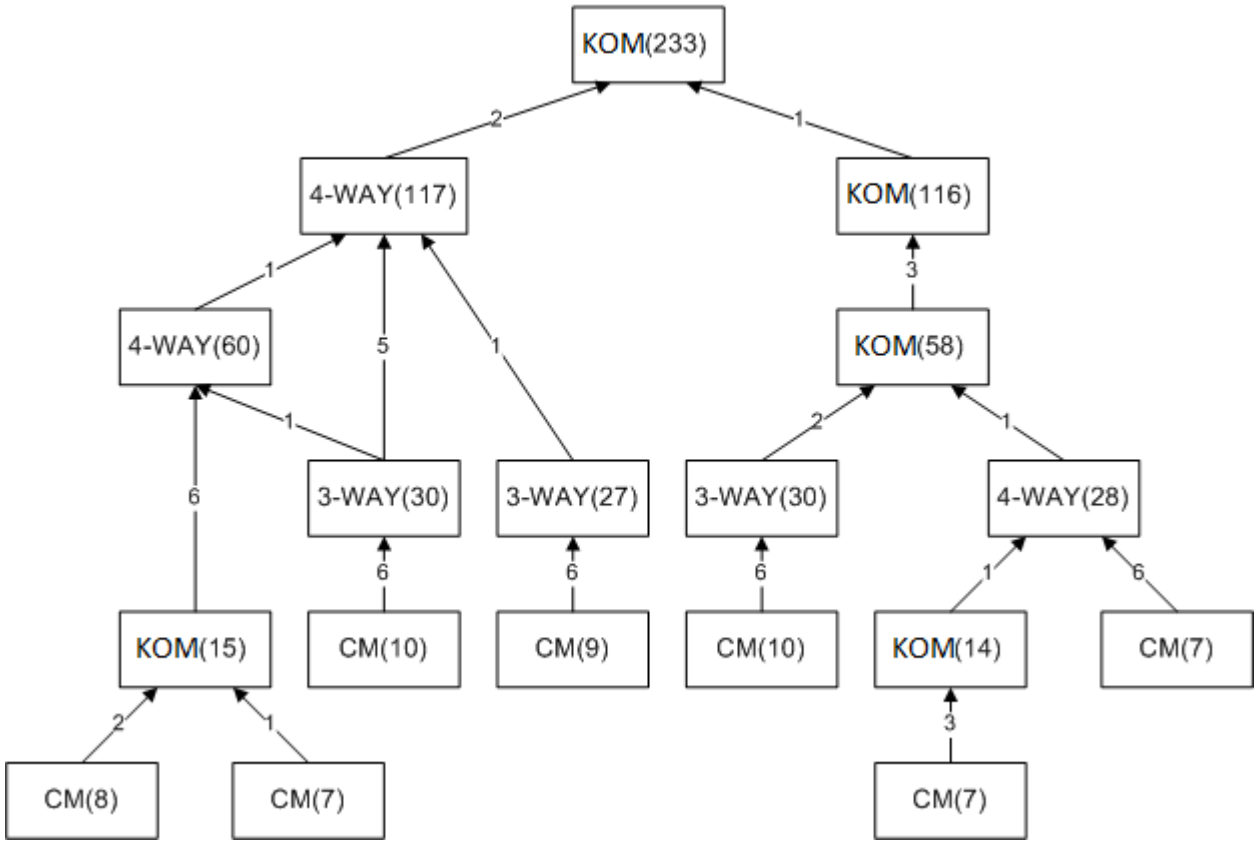


(b) Area optimized $GF\left(2^{193}\right)$ multiplier for Kintex 7

Figure 4.4: Algorithms selected for area optimized $GF\left(2^{193}\right)$ multiplier for Virtex 5 and Kintex 7

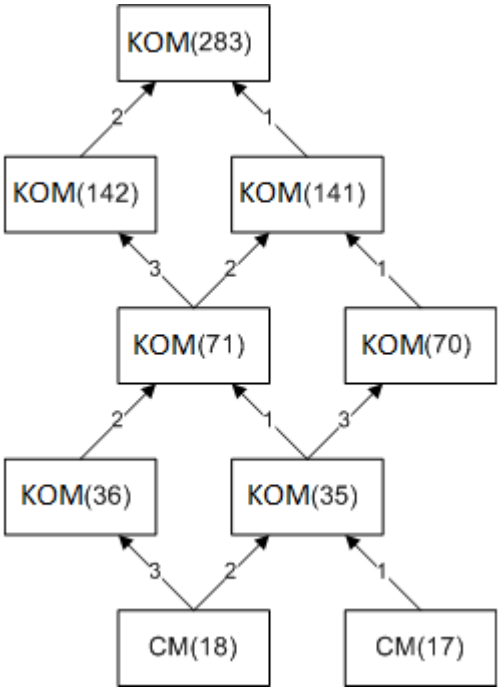


(a) Area optimized $GF\left(2^{233}\right)$ multiplier for Virtex 5

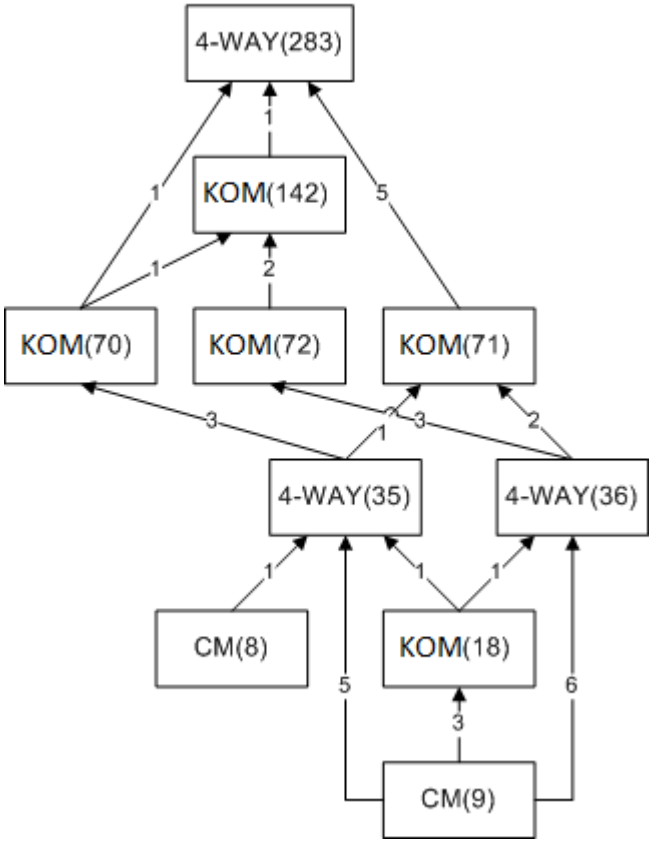


(b) Area optimized $GF\left(2^{233}\right)$ multiplier for Kintex 7

Figure 4.5: Algorithms selected for area optimized $GF\left(2^{233}\right)$ multiplier for Virtex 5 and Kintex 7



(a) Area optimized $GF\left(2^{283}\right)$ multiplier for Virtex 5



(b) Area optimized $GF\left(2^{283}\right)$ multiplier for Kintex 7

Figure 4.6: Algorithms selected for area optimized $GF\left(2^{283}\right)$ multiplier for Virtex 5 and Kintex 7

4.2 COMPARISON OF RESULTS

To compare the area utilization results of three multiplier designs all of them are implemented on the same platforms. Multipliers based on recursive KOM implementation in [16] are implemented again for Virtex 5 instead of directly using the results given in the work, since implementation tool difference has effects on implementation results. On Kintex 7, multiplier blocks with recursive KOM algorithm are implemented that have the least area utilization. Theoretically most efficient algorithms are also implemented on the selected FPGAs. Resource consumptions of theoretically recommended algorithms and recursive KOM implementations are compared to the area optimized multiplier block designs proposed in this work.

LUT consumption results are given in Table 4.2 for Virtex 5 and in Table 4.3 for Kintex 7. Plus, graphics that compare the results are given in Figure 4.7 for Virtex 5 and in Figure 4.8 for Kintex 7. For both FPGAs, algorithms recommended in [21] result in the worst area utilization. Due to LUT based internal structure of FPGA platform, theoretically lowest complexity algorithm recommended for each polynomial length gives the highest area complexity among all algorithms for the selected fields on both FPGAs. Recursive KOM implementation has better area utilization than implementation of algorithms recommended in [21]. The only exception is the multiplier block implementation for $GF\left(2^{113}\right)$ on Virtex 5, for which recommendation in [21] gives better area performance result. However, multiplier block designs proposed in this thesis provides the highest efficiency in terms of area utilization for all selected fields. As illustrated in graphics given in Figure 4.7 and Figure 4.8, for both FPGA platforms, area utilization is reduced by use of the splitting algorithm that is the most area efficient one at every step and minimum area multipliers are obtained.Only the recommended multiplier implementation for $GF\left(2^{193}\right)$ on Virtex 5 is the same in both [16] and this thesis.

The results show that it is possible to obtain smaller area complexities by using different splitting algorithms. Some of the optimal algorithms for different polynomial lengths are given in Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 for Virtex 5 and Kintex 7 implementations. All experimental results to choose the best algorithm for area minimization on the selected FPGAs are included in Appendix part.

Table 4.2: LUT consumption of analyzed multiplier designs for selected fields on Virtex 5

Field size	**113**	**128**	**163**	**193**	**233**	**283**
[16]	3573	4357	6555	8345	12359	16009
[21]	3541	10720	10827	13858	18346	31135
This work	3416	4130	6450	8345	11286	15495

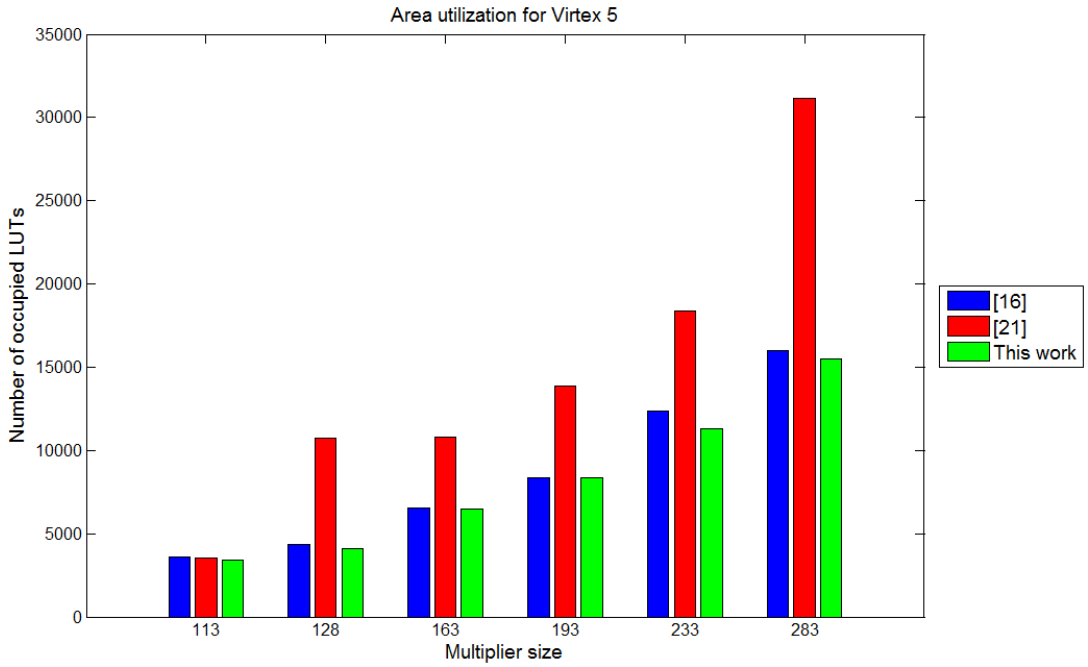


Figure 4.7: Area utilization of analyzed multiplier designs for selected fields on Virtex 5

Table 4.3: LUT consumption of analyzed multiplier designs for selected fields on Kintex 7

Field size	**113**	**128**	**163**	**193**	**233**	**283**
Recursive KOM	3208	3701	5520	7302	10390	13692
[21]	3506	7658	10627	12618	18044	27694
This work	3037	3690	5504	7288	10039	13655

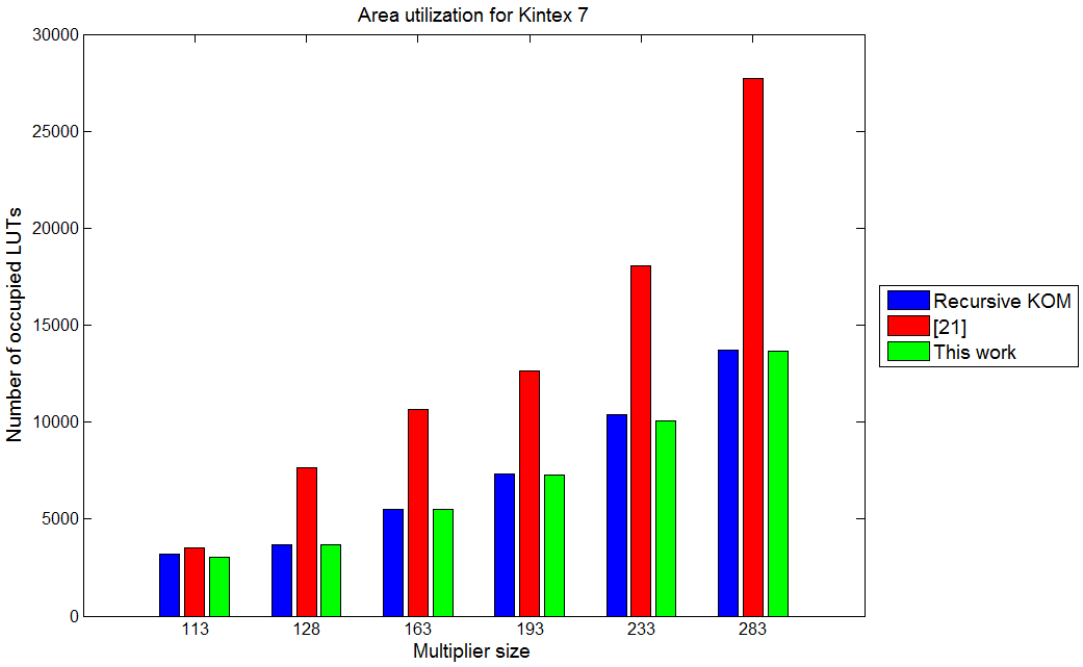


Figure 4.8: Area utilization of analyzed multiplier designs for selected fields on Kintex 7

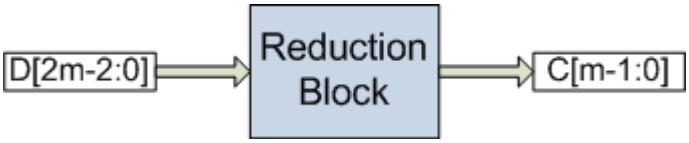


Figure 4.9: Reduction block

4.3 REDUCTION BLOCK

Area optimized multiplier blocks based on splitting algorithms does not produce the actual product of the multiplicands. The result should be reduced using the characteristic polynomial of the Galois field. This is achieved by computing modulo characteristic polynomial of the result, which requires performing multiple divisions. But, for binary fields, the product can be computed easily without any division operation.

As illustrated in Figure 4.9, the reduction block gets the $(2m-1)$-term multiplication result, D, as the input and produces the m-term product, C. Consider the binary field $GF\left(2^m\right)$ with characteristic polynomial, $f\left(x\right)$.

$$f\left(x\right) = x^m + x^k + 1$$

Bits of D with a degree less than m are already contained in the field. However, terms with a degree greater than or equal to m should be reduced. For the term x^m, result of the reduction is computed in Equation 4.1, where addition is performed modulo 2 for each term.

$$\begin{aligned} x^m &= x^m + f\left(x\right) \mod f\left(x\right) \\ &= x^m + x^m + x^k + 1 \mod f\left(x\right) \\ &= x^k + 1 \mod f\left(x\right) \end{aligned} \tag{4.1}$$

According to this result, all terms can be reduced as follows.

$$\begin{aligned} x^m &= x^k + 1 \mod f\left(x\right) \\ x^{m+1} &= x^{k+1} + x \mod f\left(x\right) \\ &\vdots \\ x^{2m-2} &= x^{m+k-2} + x^{m-2} \mod f\left(x\right) \\ &= x^{m-2} + x^{2k-2} + x^{k-2} \mod f\left(x\right), k < m/2 \end{aligned}$$

To implement this mathematical expression on a hardware platform, terms of the multiplication result, D, with degree greater than m are first reduced according to Equation 4.1 and then XORed with the terms that they have the same degree. Thus, reduction operation is achieved at 1 cycle on FPGA.

CHAPTER 5

PARTIALLY RECONFIGURABLE $GF\left(2^{128}\right)$ MULTIPLIER DESIGN

5.1 MOTIVATION

AES-GCM is one of the most commonly used cryptographic algorithms to provide information security on the Internet. It is the default cipher block defined in IEEE 802.1AE, which is the standard for Media Access Control security [23]. Another standard that mentions AES-GCM as a security protocol to provide data confidentiality, authentication and integrity is 802.11AD, namely Wireless LAN Medium Access Control and Physical Layer specifications [24]. It is also recommended as the next generation encryption algorithm, which means that the security level provided by AES-GCM is expected to be adequate for the next two decades, whereas other known encryption algorithms are not robust against subexponential attacks.

AES-GCM provides authenticated encryption. It has two functional blocks. One of them is AES block cipher, which encrypts the plain text. The other one is GHASH function. The main operation performed in the GHASH block is producing authentication tag, which is obtained as the product of a sequence of multiplications. The multiplications in GHASH function are in $GF\left(2^{128}\right)$. In Figure 5.1, block diagram of GHASH function is given, where Galois field multiplication by hash key is denoted by "$\cdot H$" symbol.

Galois field multiplication is the most essential operational block in GHASH function. To implement GHASH function on a hardware platform efficiently, optimizing

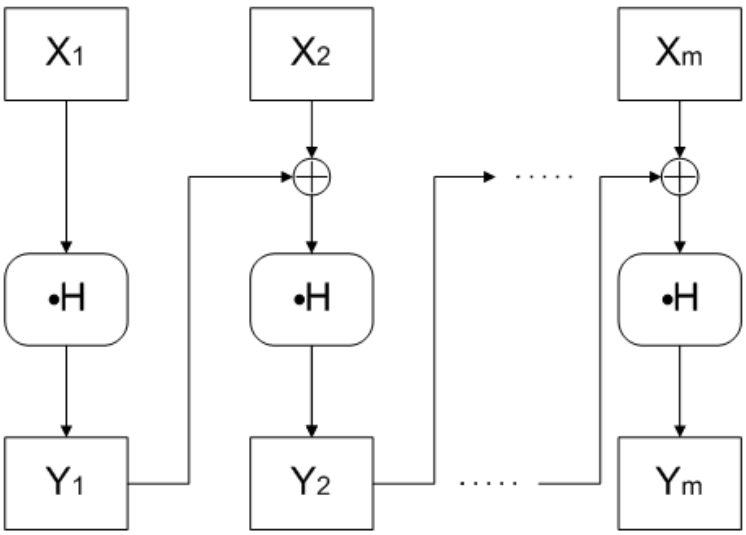


Figure 5.1: Operational block diagram for $GHASH_H\,(X_1||X_2||...||X_m) = Y_m$

resource utilization of Galois field multiplier is of critical importance. But, while minimizing the resource utilization, total delay of operation increases. In high speed communication systems, this may result in high transmission delay and even loss of data. To find an optimum point for area-time trade off, a multiplier design based on partial reconfiguration feature of FPGAs is introduced in this thesis.

5.2 DESIGN

A $GF\,(2^{128})$ multiplier block that can adapt to changing traffic conditions is designed. Flexibility of FPGAs with partial reconfiguration feature is used to implement an adaptable hardware block. Two different multiplier block types are used, which are area optimized and high performance multipliers. Area optimized multiplier block is the 128-bit multiplier block designed in Chapter 4. This block minimizes the area utilization and provides minimum power operation. However, output of a multiplication is obtained in 5 clock cycles. On the contrary, high performance multiplier gives the output of multiplication in 1 cycle using classical multiplication algorithm. But high resource utilization of this multiplier results in high power consumption.

Area optimized multiplier block is implemented to realize GHASH function by default. But, since its serving capacity, μ, is 0.2 packets/cycle, it is only able to serve the

incoming packets at low traffic rates. When the incoming traffic rate is higher than the service rate of area optimized multiplier, the data are not served within acceptable time. In such a case, high performance multiplier is activated to serve the packets in queue with a higher service rate. As a result, the packets that have been waiting to be served are multiplied by the hash key in high performance multiplier block, consuming more resource and power. In Figure 5.2 a visualization of the design is given.

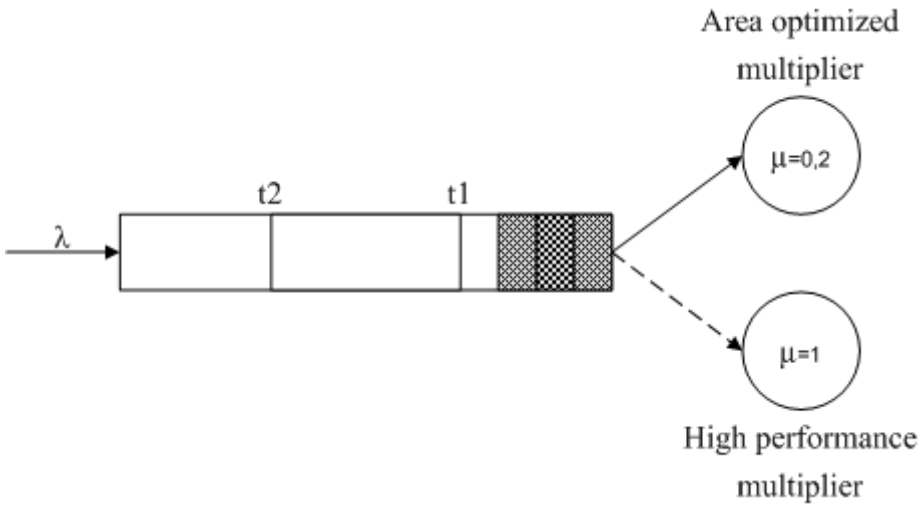


Figure 5.2: Partially reconfigurable multiplier block and queue structure

In this design, it is critical to determine at which point the actively used multiplication block should be changed. Though area optimized multiplier block is not capable of serving the incoming data at high traffic rate, it is advantageous to use it for better resource utilization when the traffic rate is low. On the contrary, high performance multiplier block consumes high amount of resources. But, in order to provide service to incoming data within an acceptable time so as not to cause delay in the whole system, it is essential to use this multiplier block at high traffic rates. The overall utilization of two multiplier blocks should provide a high throughput with a resource consumption that is not as high as that of high performance multiplier. The switching point between the two multipliers should be selected regarding this objective. To define the switching points on the packet queue, where the data to be multiplied by the hash key waits, a hysteretic buffer structure is used. In visualization of this structure in Figure 5.2, two threshold points, t1 and t2, are defined. When the number of data

packets waiting to be served exceeds the higher limit denoted as t2, high performance multiplier is activated. Multiplications are performed by high performance multiplier until queue length reduces beneath the lower threshold value denoted as t1.

Implementation of this design is realized on a partially reconfigurable FPGA. Implementing the design using different methods is possible. In one method of implementation, by defining both area optimized and high performance multipliers as reconfigurable modules, switching between the two multiplier blocks is achieved by partial reconfiguration. Reconfigurable partition is configured by the bit file of area optimized multiplier block by default. When the queue size exceeds the high threshold, t2, reconfigurable partition is configured by the bit file of high performance multiplier block. After the queue size is reduced below the low threshold, t1, it is configured by the bit file of area optimized multiplier again. The drawback of this design is that not any multiplication is performed during reconfiguration since both multipliers are defined as reconfigurable modules. In a real time application, the time spent during reconfiguration has critical effects. In order not to stop multiplication operation during reconfiguration process, another design method is considered. The default selection, which is the area optimized multiplier block, is implemented as a static module, whereas the high performance multiplier block is defined as a reconfigurable module. So, when the queue length exceeds the high threshold, the high performance multiplier block is inserted into the reconfigurable partition and multiplications are performed by this block. When the queue size is below the low threshold, the high performance multiplier block is removed. During installation and removal of the high performance multiplier block, the area optimized multiplier block continues operation. One advantage of this design is that multiplication operation is performed continuously during run time. Another advantage is that when high performance multiplier block is not included in the design, the reconfigurable partition can be used to implement another hardware block. This adds more functionality to hardware platform. Although dedicating the reconfigurable partition to high performance multiplier for some periods during the run time increases the area utilization of the multiplier, the effective area utilization is much less due to this functionality. Plus, power consumption of the adaptable multiplier is reduced without a critical change in time performance by implementing the multiplier module using this method.

5.3 SIMULATION AND IMPLEMENTATION

To evaluate the performance of partially reconfigurable multiplier design, simulations are done related to delay, throughput and required buffer size. To analyze area utilization and power consumption, the design is implemented on Xilinx Kintex 7 FPGA with part number XC7K325T-2FFG900C, using Xilinx ISE Design Suite 14.2 development tool.

Before evaluating performance of the partially reconfigurable design, first the area optimized and high performance multiplier blocks are analyzed for their area, power, throughput and delay behaviors. Area utilization of area optimized multiplier is 4568 LUTs for Kintex 7 FPGA, whereas high performance multiplier has an area utilization of 7041 LUTs. Then the implemented designs are analyzed using Xilinx Power Analyzer. The results obtained for different traffic conditions with λ values 0.3, 0.5 and 0.7 are given in Table 5.1. To analyze throughput and delay performances, Poisson distributed random traffics with user defined arrival rates are generated on C# software environment. Results of 1 second simulations for λ values 0.3, 0.5 and 0.7 show that area optimized multiplier has a throughput of 0.2 under all traffic conditions due to its serving capacity, whereas high performance multiplier provides the maximum throughput for all simulation conditions. For delay performances, area optimized multiplier has large average and maximum delay values, whereas high performance multiplier generates output without delay. In terms of buffer size, high performance multiplier does not need a buffer since packets are served without delay, whereas buffer requirement of area optimized multiplier is infinite under simulated traffic conditions with arrival rates larger than service rate of area optimized multiplier.

After the performance of area optimized and high performance multiplier blocks are evaluated, the same metrics are simulated also for the partially reconfigurable multiplier. Performance of the multiplier block depends on two variables, which are input traffic condition and threshold values. In order to observe the impact of these variables on multiplier performance, simulations are made under different traffic conditions and with different threshold values.

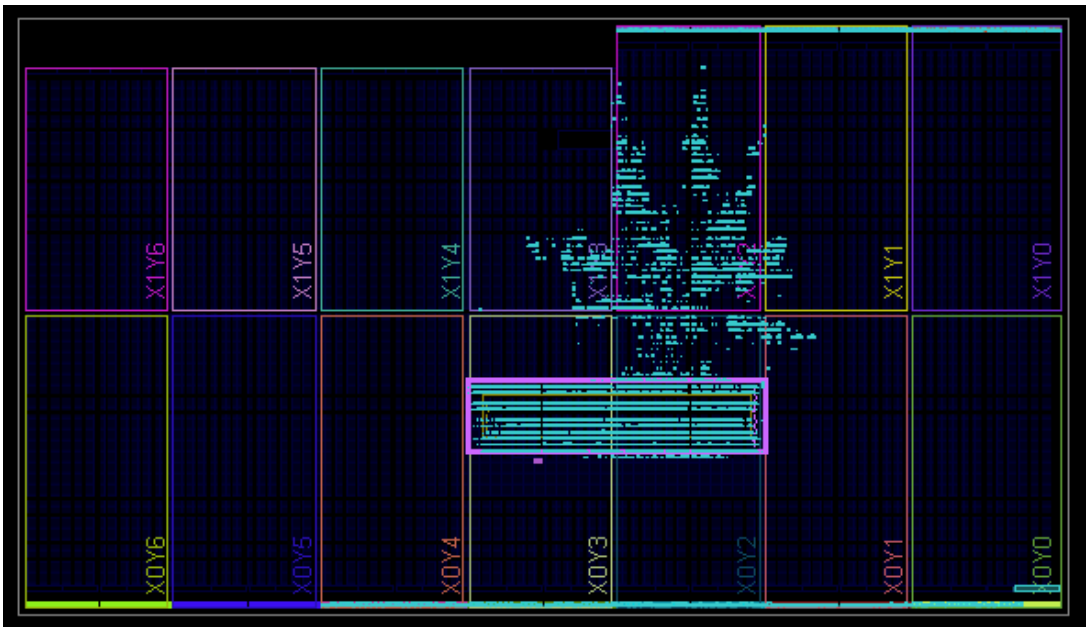


Figure 5.3: Implementation of partially reconfigurable multiplier on FPGA with high performance multiplier block implemented in the reconfigurable partition

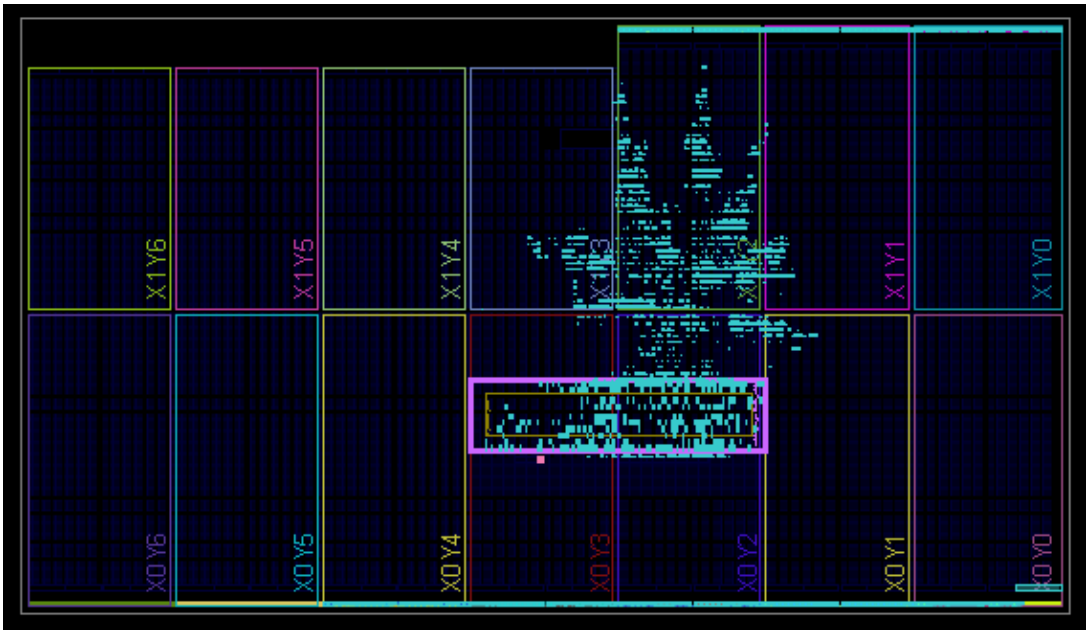


Figure 5.4: Implementation of partially reconfigurable multiplier on FPGA with non-functional black box module implemented in the reconfigurable partition

The same traffic conditions as the ones used in area optimized and high performance multiplier block simulations are generated on C# software environment. Simulation results for Poisson traffic with $\lambda = 0.3$, $\lambda = 0.5$ and $\lambda = 0.7$ are included in the next section to analyze the behavior of multiplier block under different input conditions.

The threshold values used in simulations are selected in accordance with the partial bit file size, which is generated after implementation on FPGA. Therefore, the design is implemented on the FPGA before simulations. The area optimized multiplier block is implemented in the static part whereas the high performance multiplier block is defined as a reconfigurable module. When the high performance multiplier block is not needed, it is replaced by a nonfunctional black box module. In Figure 5.3 and Figure 5.4, implementation results with and without high performance multiplier are given respectively. Reconfigurable partition is selected in order to meet the LUT requirement of high performance multiplier. An area consisting of 8000 LUTs is selected as reconfigurable partition. High performance multiplier block has 87% utilization within this partitions when implemented as a reconfigurable module. For this implementation, size of partial bit files for the reconfigurable partition is 432 KB. Since the ICAP interface has 400 MB/s bandwidth, required time to reconfigure the FPGA using a partial bit file is

$$\begin{aligned} \tau &= 432\,KB \div 400\,MB/s \\ &= 1054.6875\,\mu s \end{aligned}$$

For implementation of the design, 100 MHz clock is used. So, reconfiguration of FPGA is completed in approximately

$$\begin{aligned} c &= \lceil \tau\,\mu s * 100\,MHz \rceil \\ &= 105469\,clock\,cycles \end{aligned}$$

During this period, area optimized multiplier block continues serving the incoming data with a service rate of $\mu = 0.2$. Therefore, the maximum number of packets that may enter the queue is

$$\begin{aligned} N &= (1 - 0.2) * c \\ &= 84375\,packets \end{aligned}$$

While using the high performance multiplier block, when the number data packets in the queue drops below t1 threshold value, FPGA is reconfigured. Since high performance multiplier is not active during this period, new data packets start to enter queue. Whenever number of packets in the queue reaches t2 threshold value, FPGA needs to be reconfigured again to install the high performance multiplier. But, this is not possible if the previous reconfiguration process is not completed yet. Selecting the threshold values on hysteretic buffer such that the difference between t1 and t2 is not less than N guarantees that reconfiguration process is done before the number of packets waiting to be served in queue reaches the threshold. Plus, unless a packet arrives at every cycle, the reconfigurable partition can be used for different purposes by implementing a reconfigurable module with different functionality. By increasing the difference between t1 and t2, it is possible to use the reconfigurable modules with different functionalities for longer periods. However, this results in larger buffer size requirement for multiplier block since number of packets waiting in the queue increases in the meantime.

Time periods, during which high performance multiplier is used effectively, are also computed in simulations. This information is necessary to calculate the power consumption and effective area usage of partially reconfigurable multiplier block.

5.4 RESULTS

Partially reconfigurable multiplier block is analyzed under the traffic conditions mentioned before with buffer threshold values chosen as indicated in the previous section in order to find an optimum point between power consumption and delay and throughput performances.

Since the high performance multiplier block is implemented as a partial module for partially reconfigurable multiplier, area utilization is 8000 LUTs, which is greater than the area utilization of high performance multiplier block implemented as a static block. Plus, the area optimized multiplier is implemented as a static module and consumes certain amount of area during run time even if it is not active. Therefore, area consumption of the multiplier design is higher compared to area optimized and high

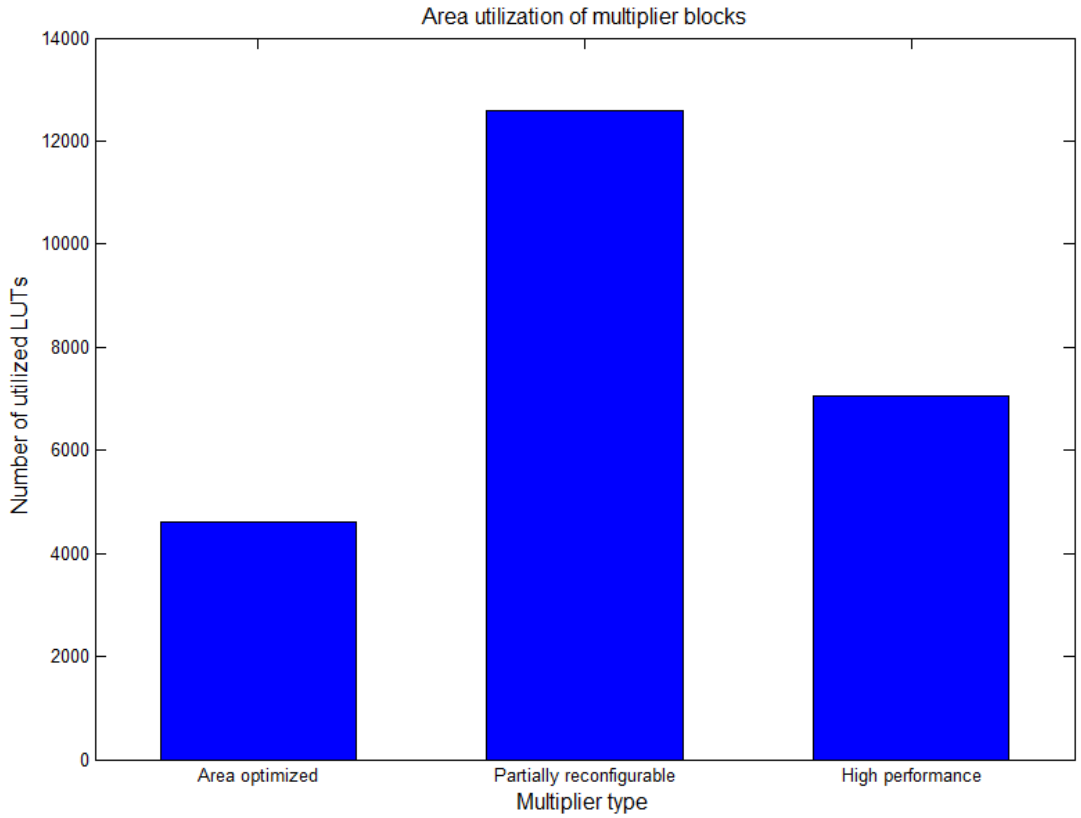


Figure 5.5: Area utilization of three multiplier types in terms of number of utilized LUTs

performance multiplier blocks alone. Area utilization of three types of multipliers are shown in Figure 5.5.

In contrast to area utilization, other performance metrics, which are power consumption, buffer requirement, throughput and delay, are affected by the traffic condition and threshold values. In order to observe the effect of these parameters, power, throughput and delay performances are simulated under several different conditions.

Power consumption of the multiplier block is directly affected by the input traffic rate. When the traffic rate is low, area optimized multiplier, whose power consumption is lower due to less amount of resource utilization, is active most of the operation time. In addition, input and output ports change their states less frequently at low traffic rates, resulting in less power consumption. Power consumption of three multiplier types under different traffic conditions are given in Table 5.1. They are also compared in Figure 5.6 for λ values 0.3, 0.5 and 0.7.

Table 5.1: Power consumption (in mW) of three multiplier types under different input traffic conditions

Traffic rate (λ)	0.3	0.5	0.7
Area optimized	480	480	480
Partially reconfigurable	490	560	651
High performance	512	609	684

The graphic illustrated in Figure 5.6 shows that partially reconfigurable multiplier design has a power consumption that is higher than that of area optimized multiplier but not as much as that of high performance multiplier under low, average and high traffic rates.

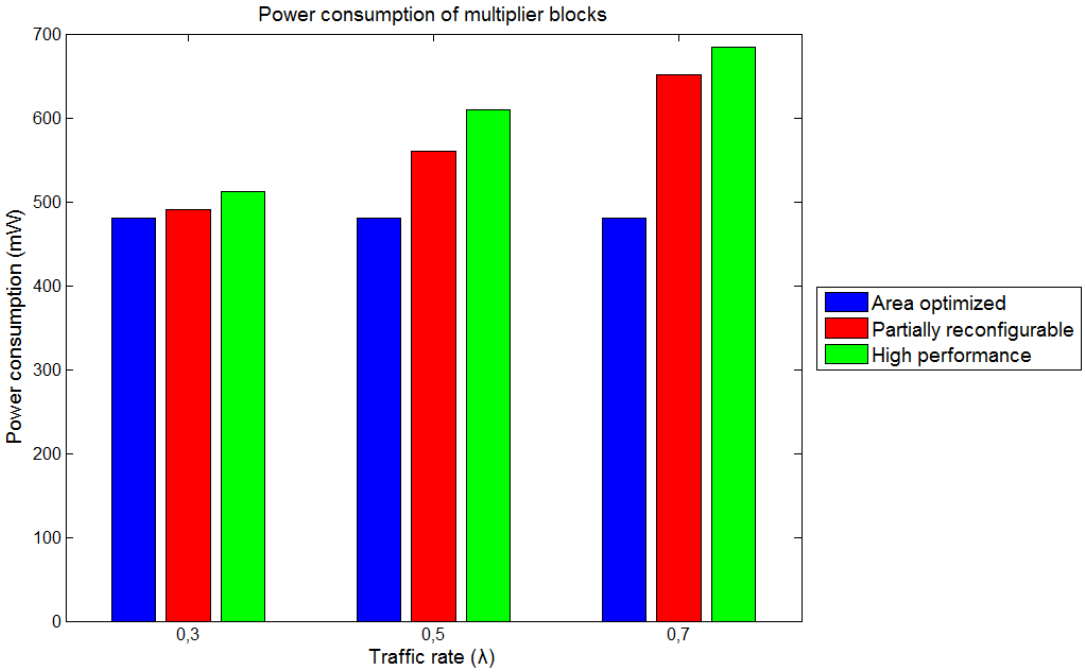


Figure 5.6: Power consumption of three multiplier types under different traffic rates

Throughput is another metric that is affected by the input traffic rate. For area optimized multiplier, since input traffic rates chosen for simulations are higher than the serving capacity of the multiplier, only limited number of packets are served. As a result its throughput is low. On the contrary, high performance multiplier is capable of serving any incoming data within 1 clock cycle. Therefore, all packets are served

Table 5.2: Throughput of three multiplier types under different input traffic conditions in terms of packets/clock cycle

Traffic rate (λ)	0.3	0.5	0.7
Area optimized	0.20	0.20	0.20
Partially reconfigurable	0.26	0.39	0.50
High performance	0.26	0.39	0.50

within the specified simulation time resulting in the maximum possible throughput rate for all traffic rates. For partially reconfigurable multiplier, throughput performance under different simulation conditions are illustrated in Figure 5.7 for λ values 0.3, 0.5 and 0.7. The results show that, partially reconfigurable multiplier provides the maximum possible throughput under all simulated traffic conditions, like high performance multiplier does. Throughput provided by each multiplier under different traffic conditions are also given in Table 5.2.

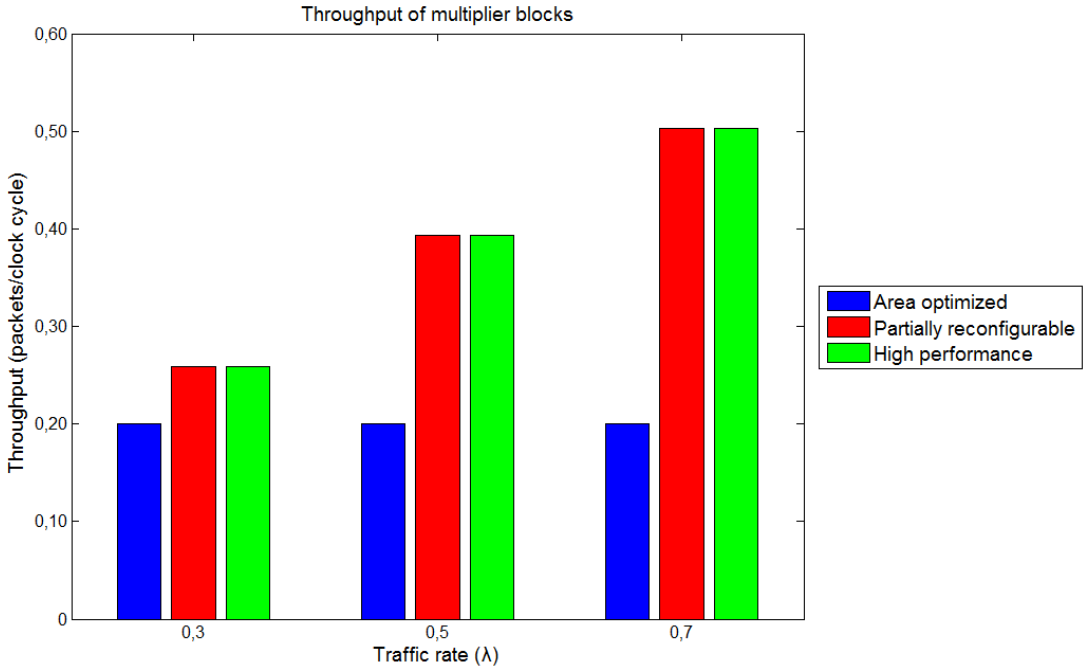


Figure 5.7: Throughput of three multiplier types under different traffic rates

Although throughput performance of partially reconfigurable multiplier block is as high as that of high performance, some delay is introduced while processing packets. High performance multiplier has 1 clock cycle delay since every multiplicand is multiplied by the key to obtain the product in 1 cycle. For 100 MHz clock operation this delay is negligible. On the other hand packet delay is very high for area optimized multiplier since its serving capacity is low. Under high traffic rates delay of incoming packets grow incrementally. This causes unbounded average and maximum delay values for area optimized multiplier. 1 second simulations under three different traffic loads shows that partially reconfigurable multiplier introduces a delay that is very low compared to that of area optimized multiplier. Exact values of average and maximum delays of three multiplier types for 1 second operation are given in Table 5.3 and Table 5.4 respectively for different traffic loads. In Figure 5.8 and Figure 5.9 average and maximum packet delays within 1 second operation are illustrated respectively for λ values 0.3, 0.5 and 0.7.

Table 5.3: Average delay (in ms) of three multiplier types under different input traffic conditions

Traffic rate (λ)	0.3	0.5	0.7
Area optimized	114.2	145.9	301.4
Partially reconfigurable	1.9	1.5	1.3
High performance	0	0	0

Table 5.4: Maximum delay (in ms) of three multiplier types under different input traffic conditions

Traffic rate (λ)	0.3	0.5	0.7
Area optimized	228.3	491.7	602.7
Partially reconfigurable	4.4	3.2	2.3
High performance	0	0	0

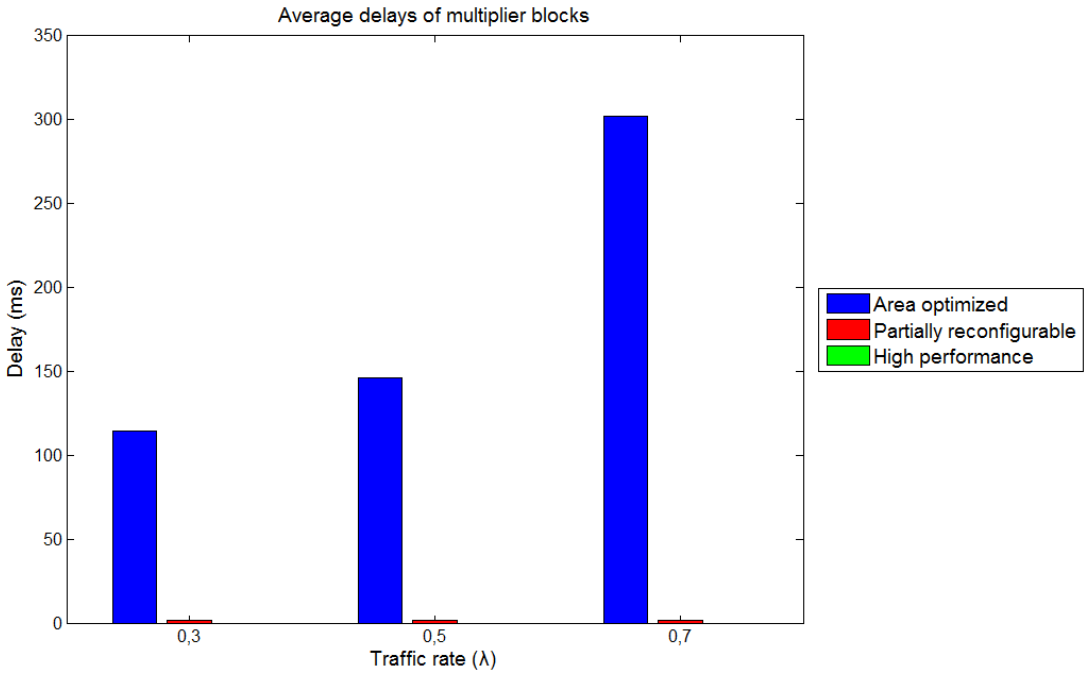


Figure 5.8: Average delay values of three multiplier types under different traffic rates

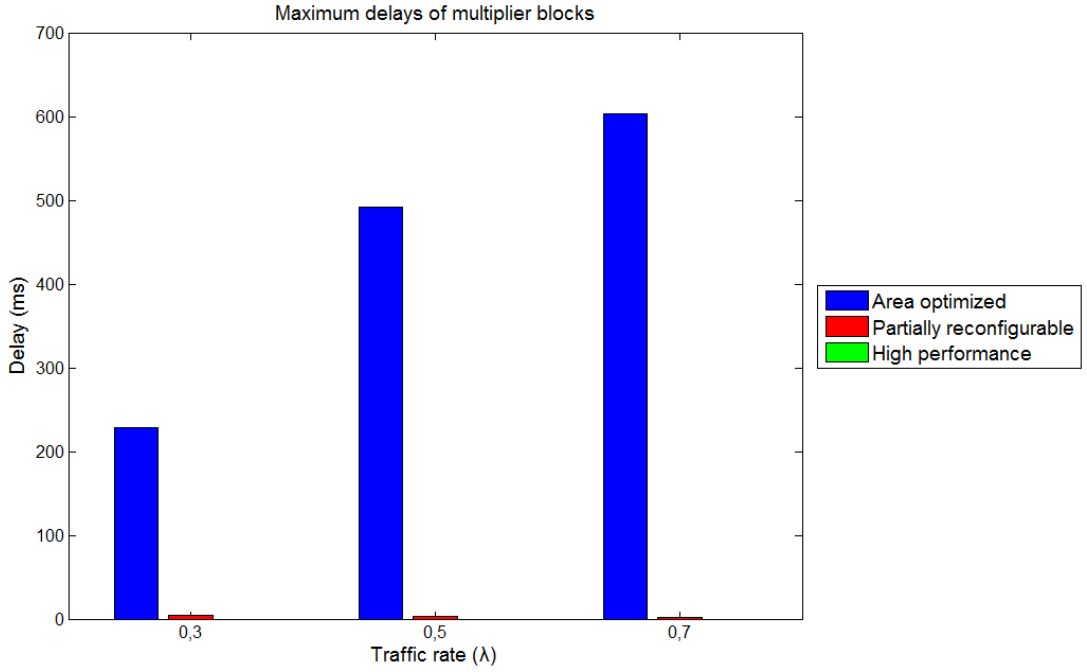


Figure 5.9: Maximum delay values of three multiplier types under different traffic rates

Buffer requirement is another metric that is affected by input traffic load and threshold values. High performance multiplier does not need a buffer since incoming data does not wait for multiplication. On the contrary, area optimized multiplier has a large buffer requirement, especially under high traffic loads. When traffic rate is higher than multiplier capacity, number of incoming data waiting in the queue increases. For partially reconfigurable multiplier, this is the case when high performance multiplier module is not inserted into the reconfigurable partition and area optimized multiplier module in the static part handles all the multiplication requirements. When the number of data waiting in the buffer exceeds t2 threshold, high performance multiplier module is inserted into the reconfigurable partition to meet the multiplication requirements waiting in the buffer. As t2 threshold value gets higher, buffer requirement of the partially reconfigurable multiplier increases. High performance multiplier continues operation until the number of multiplicands waiting in the buffer drops below t1 threshold. If t1 threshold value is selected small, high performance multiplier operates for longer period, causing higher power consumption. Therefore, selecting threshold values in accordance with the bit file size as mentioned in the previous section considering resource requirement and power consumption optimizes the buffer utilization. In the simulations, buffer size requirement of partially reconfigurable multiplier under different traffic conditions with λ values 0.3, 0.5 and 0.7 is examined. The results are given in Table 5.5.

FPGA used in this thesis has 16020 Kb BRAM resources. Since each data is 128 bits long, BRAM resources of the FPGA are sufficient to meet buffer requirement for upto 128160 packets. For the threshold values used for the simulations in this thesis, BRAM resources meet the buffer requirement of the partially reconfigurable multiplier for traffic rates $\lambda = 0.3$ and $\lambda = 0.5$. For higher traffic rates, buffer requirement can be reduced by changing threshold values, such that t1 and t2 values are closer. In such an implementation, buffer requirement will be met by BRAM resources. But flexibility of hardware platform will reduce in return, since reconfigurable partition will be used for multiplication for most of the operation time and some incoming data might be lost during reconfiguration process since there may not be enough space in the buffer.

Table 5.5: Maximum buffer size requirement (in packets) of partially reconfigurable multiplier for different threshold values (in packets) under traffic conditions

Thresholds		**Traffic rate**		
t1	**t2**	**0.3**	**0.5**	**0.7**
100	84475	112153	126322	137996
500	84875	112521	126755	138357
1000	85375	113093	127235	138878
2000	86375	114093	128245	139876

The results show that by using partially reconfigurable multiplier design it is possible to find an optimum point for power-delay trade off. Power consumption of partially reconfigurable multiplier is more than that of area optimized multiplier but less than that of high performance multiplier. Although high performance multiplier has negligible delay, little delay is introduced in partially reconfigurable multiplier. This delay is very low compared to that of area optimized multiplier, which is unbounded due to low service capacity. In addition, throughput performance that is as high as that of high performance multiplier is achieved. As a result, optimizing power and delay performances of a Galois field multiplier while achieving maximum possible throughput is possible.

The only drawback of this design is the high area utilization. Hardware resources utilized by area optimized multiplier module are in the static partition. Plus, a reconfigurable partition is defined where high performance multiplier module is implemented. Therefore, total area utilized by the multiplier modules is larger than the area utilization of both multiplier modules when implemented alone as a static block. However, high performance module is not included in the design the whole operation time. For time periods high performance multiplier is not required, reconfigurable partition can be used to implement another function, meaning that reconfigurable partition is an area that can be used for other purposes than multiplier block for periods during which Galois field multiplication is done by area optimized multiplier. So, it is possible to define an effective area utilization, which defines the area utilization dedicated to Galois field multiplication block during operation. To compute the effective area utilization of partially reconfigurable multiplier block, time periods when high performance multiplier is included during simulation time are recorded. Then, effective

area utilization of the multiplier is computed as in Equation 5.1.

$$A_{effective} = A_{static} + \frac{Time\ high\ performance\ multiplier\ is\ active}{Operation\ time} * A_{reconfigurable} \tag{5.1}$$

Using the definition given in Equation 5.1, effective area utilization of partially reconfigurable multiplier under traffic conditions with λ values 0.3, 0.5 and 0.7 is computed. Results are illustrated in graphic given in Figure 5.10. In Table 5.6, numeric values related to effective area utilization of partially reconfigurable multiplier block in terms of utilized LUT numbers are also given.

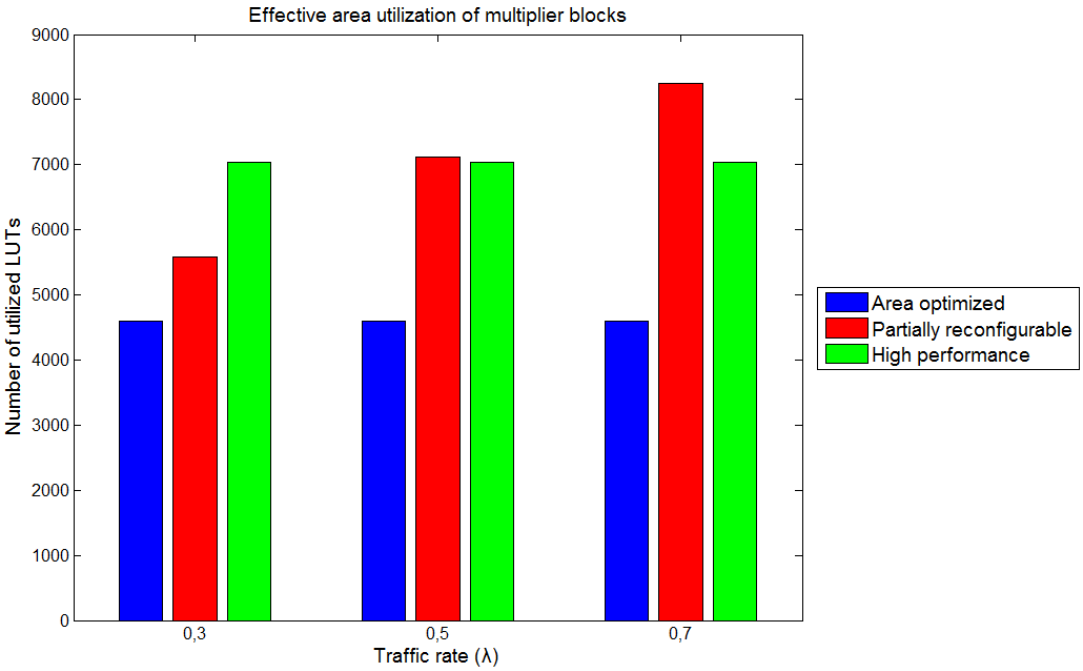


Figure 5.10: Effective area utilization of partially reconfigurable multiplier compared to area utilization of static area optimized and high performance multipliers under traffic rates $\lambda = 0.3$, 0.5 and 0.7

Table 5.6: Effective area utilization of partially reconfigurable multiplier block under different traffic rates

Traffic rate (λ)	Effective area (LUT)
0.3	5577
0.5	7111
0.7	8242

The results show that finding an optimum point between power consumption and delay performances is possible for $GF\,(2^{128})$ multiplier IP block by making real time hardware adaptations according to incoming data traffic. The partially reconfigurable multiplier IP block design provides low delay and low power operation while providing maximum possible throughput. Although area utilization seems to be high, effective area utilization is comparable to area utilization of a statically implemented multiplier block. With these results, partially reconfigurable multiplier block is a solution to power-delay trade off.

CHAPTER 6

CONCLUSION

Galois Field arithmetic has a wide range of applications. Cryptography is one of them, which is an indispensable part of our lives with increasing use of electronic communication devices. In most cryptographic algorithms, arithmetic operations are performed on Galois Fields, where $GF(2^m)$ multiplication is considered as one of the main functional blocks since other operations such as division, inversion, exponentiation depends on multiplication. For successful operation of a cryptographic application, it is important to have a $GF(2^m)$ multiplier block that provides high throughput. Low power consumption of a multiplier block is also important, since such applications are mainly used in portable devices with limited resources. Meeting both throughput and resource utilization requirements is hard, since there is a trade-off between time and power performances.

In this thesis, a method for meeting both time and power requirements of a $GF(2^m)$ multiplier block is proposed. The implementation platform for this work is FPGA, which is preferred for its high speed operation capability and implementation flexibility. In the design, two multiplier block types are included, namely the area optimized multiplier and the high speed multiplier. Area optimized multiplier has low area utilization and operates with low power consumption. However, its serving capacity is low, so it may not be sufficient under high traffic conditions. On the contrary, high performance multiplier is fast and provides high throughput. The main drawback of this multiplier is its high resource utilization. To improve the trade-off between time and power requirements, both multipliers are used for adapting to changing traffic rate. For this purpose, partial reconfiguration feature of FPGA is used in the design.

As the first step, an area optimized multiplier block is designed. For this purpose, splitting multiplication algorithms are used recursively. However, instead of using the same algorithm at every recursion step, the most suitable splitting algorithm is preferred for each polynomial length. This method is proposed in [21], but the theoretical results obtained in [21] are not valid for FPGA implementations. Therefore, the most efficient splitting algorithms for various polynomial lengths are verified experimentally for two FPGA types. The area optimized multiplier block that is used in the design proposed in this thesis is obtained using this approach. In addition, multiplier blocks for different polynomial lengths commonly used in cryptographic applications are obtained in these experiments for Virtex 5 and Kintex 7 FPGAs of Xilinx.

A high performance multiplier is also designed. For this purpose, a multiplier using CM algorithm is preferred for its low delay and high throughput capability. However, this multiplier consumes twice the area and power in comparison to the area optimized multiplier.

After obtaining two multiplier types, a new multiplier block is designed, using partial reconfiguration feature of FPGAs. The new multiplier block uses area optimized and high performance multiplier designs to respond to multiplication requests, according to a variable incoming traffic rate. A hysteretic buffer control is used to determine the active multiplier type. Better utilization of resources is achieved by defining optimum buffer thresholds for including and discarding the high performance multiplier.

Area optimized multiplier block is the default selection, so that power consumption of the multiplier block is low as long as the number of multiplication requests does not exceed the high threshold value. Whenever high threshold is exceeded, high performance multiplier is included as a partially reconfigurable module on FPGA. With the help of high performance multiplier, a high throughput operation is achieved and multiplication requests are met with a little delay. In order not to consume high power, partially reconfigurable high performance multiplier block is extracted from the design when number of multiplication requests drops below the low threshold value.

Performance of the partially reconfigurable multiplier IP block design is verified by

simulations. Power, throughput and delay performances are obtained for various input traffic conditions. Results obtained from these simulations show that an optimum point for power - delay trade off can be found by making real time hardware adaptations. Low power operation with low delay is possible while providing maximum throughput.

Although area utilization of the multiplier block is increased due to implementation of both area optimized and high performance multipliers on FPGA at the same time, effective area utilization is less since high performance multiplier block is not included in the design during the whole operation time. Plus, reconfigurable partition may be used to implement some other functional blocks, adding flexibility to hardware infrastructure.

One additional requirement of partially reconfigurable multiplier block is buffer implementation for incoming multiplication request. Implementing an additional buffer block results in more hardware resource consumption. But, implementing the buffer block using BRAM resources is possible in order not to increase LUT utilization of the Galois field multiplier IP block. The experimental works show that BRAM capacity of the selected FPGA, Kintex 7, is sufficient for the buffer size requirement for 128-bit Galois field multiplier block.

Reducing power consumption and providing low delay without reducing throughput is achieved by using our proposed partially reconfigurable multiplier block, in return for buffer space. The experimental work done in this thesis shows that partially reconfigurable multiplier design may provide upto 8% saving in power consumption and 21% saving in effective area utilization compared to high performance multiplier block while achieving the same throughput performance. In the meantime, very little delay is introduced, which is negligible compared to the unbounded delay characteristic of area optimized multiplier block. The partially reconfigurable multiplier block is most suitable for bursty traffic conditions where multiplication request rate is low on the average but gets very high from time to time.

To summarize, $GF(2^m)$ multiplication is an important functional block, and designing a high throughput multiplier with low power consumption is essential for cryptographic applications to provide fast security service with limited resources. The work

conducted in this thesis shows that it is possible to make such a design with real time hardware adaptations.

As a result of the work done in this thesis, new multiplier blocks for various polynomial lengths are introduced using splitting multiplication algorithms to make area optimization. The multiplication algorithms proposed in this thesis for Virtex 5 and Kintex 7 FPGAs are the most area efficient ones in the literature.

Apart from the area optimization work, a multiplier block design that is adaptable to changing traffic conditions is also realized. In this design, partially reconfigurable modules are used to improve power - delay trade off of hardware block design. It is proved by experimental work that it is possible to obtain a fast and high throughput $GF(2^m)$ multiplier block with low power consumption and effective area utilization, in return for extra buffer utilization. Changing the multiplier block characteristics in accordance with the incoming multiplication request rates provides better time and power performances, especially under bursty traffic conditions.

REFERENCES

[1] Mozilla Corporation (2005, September 26). *Introduction to Public-Key Cryptography*. Retrieved from `https://developer.mozilla.org/en/docs/Introduction_to_Public-Key_Cryptography` [last accessed on 5 May 2015]

[2] Savaş, E. and Koç, Ç. K. (2010). Finite Field Arithmetic for Cryptography. *IEEE Circuits and Systems Magazine, Second Quarter 2010,* 40-56.

[3] Parker, D. B. (2002). Toward a New Framework for Information Security. In Bosworth, S. and Kabay, M. E. (Eds.) *The Computer Security Handbook*. New York, NY: John Wiley & Sons.

[4] Stallings, W. (1999). *Cryptography and Network Security: Principles and Practice*. New Jersey: Prentice-Hall, Inc.

[5] Blahut, R. E. (2014). *Cryptography and Secure Communication*. New York: Cambridge University Press.

[6] Cisco (April 2012). *Next Generation Encryption*. Retrieved from `http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html` [last accessed on 5 May 2015]

[7] Hankerson, D., Menezes, A. and Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. New York: Springer-Verlag, Inc.

[8] Hietala, J. D. (2007). *Hardware versus Software: A Usability Comparison of Software-Based Encryption with Seagate Secure Hardware-Based Encryption* [White Paper]. Hardware versus software.

[9] Vladislav K. and Okhrimenko, A. (2012). Approaches for the Performance Increasing of Software Implementation of Integer Multiplication in Prime Fields. *IACR Cryptology ePrint Archive,* 170-170.

[10] Zuchowski, P. S., Reynolds, C. B., Grupp, R. J., Davis, S. G., Cremen, B. and Troxel, B. (2002). *A Hybrid ASIC and FPGA Architecture*. Paper presented at IEEE/ACM International Conference. doi: 10.1109/ICCAD.2002.1167533

[11] Kuon, I. and Rose, J. (2007). Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer Aided Design of Integrated Circutis and Systems, 26,* 203-215.

[12] Xilinx (2013). *Partial Reconfiguration User Guide* (UG702 v14.5).

[13] Meher, P. K. (2008). Systolic and Super-Systolic Multipliers for Finite Field GF(2^m) Based on Irreducible Trinomials. *IEEE Transactions on Circuits and Systems, 55,* 1031-1040.

[14] Imana, J. L., Sanchez, J. M. and Tirado, F. (2006). Bit-Parallel Finite Field Multipliers for Irreducible Trinomials. *IEEE Transactions on Computers, 55,* 520-533.

[15] Morales-Sandoval, M., Feregrino-Uribe, C. and Kitsos, P. (2011). Bit-serial and Digit-serial GF(2^m) Montgomery Multipliers Using Linear Feedback Shift Registers. *IET Computers & Digital Techniques, 5,* 86-94.

[16] Zhou, G., Michalik, H. and Hinsenkamp, L. (2010). Complexity Analysis and Efficient Implementations of Bit Parallel Finite Field Multipliers Based on Karatsuba-Ofman Algorithm on FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 18,* 1057-1066.

[17] Liu, Z., Großschadl, J. and Kizhvatov, I. (2010). Efficient and Side-Channel Resistant RSA Implementation for 8-Bit AVR Microcontrollers. Proceedings from SOCIOT 2010: *First International Workshop on the Security of the Internet of Things*. Tokyo, Japan.

[18] Bernstein, D. J. (2009). Batch Binary Edwards. *Advances In Cryptology - CRYPTO 2009, 5677,* 317-336.

[19] Cenk, M., Hasan, A. and Negre, C. (2012). *Efficient Subquadratic Space Complexity Binary Polynomial Multipliers Based On Block Recombination* (Research Report). Retrieved from `https://hal.archives-ouvertes.fr/hal-00712090v1/document` [last accessed on 5 May 2015]

[20] Cenk, M. and Özbudak, F. (2009). Improved Polynomial Multiplication Formulas over F_2 Using Chinese Remainder Theorem. *IEEE Transactions on Computers, 58,* 572-576.

[21] Cenk, M. and Hasan, M. A. (2015). *Some New Results on Binary Polynomial Multiplication* (Research Report). Retrieved from `http://eprint.iacr.org/2015/094` [last accessed on 5 May 2015]

[22] Viega, J. and McGrew, D. (2004). *The Galois/Counter Mode of Operation (GCM).* Submission to NIST. `http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf` [last accessed on 5 May 2015]

[23] IEEE Computer Society (2006). *IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security* (IEEE Std 802.1AE). New York ,USA: IEEE.

[24] IEEE Computer Society (2012). *IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements* (IEEE Std 802.11AD). New York, USA: IEEE.

[25] Zhou, G. and Hinsenkamp, L. (2007). *Efficient and High-Throughput Implementations of AES-GCM on FPGAs.* Paper presented at ICFPT 2007: International Conference on Field-Programmable Technology 2007, Kitakyushu, Japan. doi: 10.1109/FPT.2007.4439248

[26] Henzen, L. and Fichtner, W. (2010). *FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications.* Paper presented at Proceedings of the ESSCIRC 2010: European Solid State Circuits Conference 2010, Seville, Spain. doi: 10.1109/ESSCIRC.2010.5619894

[27] Abdellatif, K. M., Chotin-Avot, R. and Mehrez, H. (2012). *Efficient Parallel-Pipelined GHASH for Message Authentication.* Paper presented at ReConFig 2012: International Conference on Reconfigurable Computing and FPGAs 2012, Cancun, Mexico. doi: 10.1109/ReConFig.2012.6416742

[28] Granado, J. M., Vega-Rodriguez, M. A., Sanchez-Perez, J. M. and Gomez-Pulido, J. A. (2009). IDEA and AES, Two Cryptographic Algorithms Implemented Using Partial and Dynamic Reconfiguration. *Microelectronics Journal, 40,* 1032-1040.

[29] Granado-Criado, J. M., Vega-Rodriguez, M. A., Sanchez-Perez, J. M. and Gomez-Pulido, J. A. (2010). A New Methodology to Implement the AES Algorithm Using Partial and Dynamic Reconfiguration. *Integration, the VLSI Journal, 43,* 72-80.

[30] Xilinx (n.d.). FPGA and ASIC Technology Comparison [Powerpoint slides].

APPENDIX A

AREA OPTIMIZATION ON VIRTEX 5

Splitting multiplication algorithms preferred to implement the most area effective multiplier blocks for various field sizes on Virtex 5 are given in Table A.1 below.

Submultiplier sizes required for implementing the most efficient algorithm for the field size given in the first column are given in the third column. Splitting rule indicates the most efficient algorithm recommended for the corresponding field size. For example, for field size 283, splitting rule is "2n-1", which indicates that KOM algorithm, which splits the multiplicands into 2, should be used and since 283 cannot be splitted into two partitions of equal length there will be one partition with length "n=142" and one partition with length "n-1=141".

Table A.1: Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5

Multiplier size	**Splitting rule**	**Submultiplier size**
283	2n-1	142, 141
233	2n-1	117, 116
193	2n-1	97, 96
163	5n-2	33, 31
142	2n	71
141	2n-1	71, 70
128	2n	64
118	2n	59
117	2n-1	59, 58

Table A.1: Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5 (Continued)

Multiplier size	**Splitting rule**	**Submultiplier size**
116	2n	58
113	2n-1	57, 56
98	2n	49
97	2n-1	49, 48
96	2n	48
95	2n-1	48, 47
93	2n-1	47, 46
82	5n-3	17, 14
81	5n-4	17, 13
78	5n-2	16, 14
77	5n-3	16, 13
72	2n	36
71	2n-1	36, 35
70	2n	35
65	2n-1	33, 32
64	2n	32
63	2n-1	32, 31
60	2n	30
59	2n-1	30, 29
58	2n	29
57	2n-1	29, 28
56	2n	28
55	2n-1	28, 27
53	2n-1	27, 26
50	2n	25
49	2n-1	25, 24
48	2n	24
47	2n-1	24, 23

Table A.1: Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5 (Continued)

Multiplier size	**Splitting rule**	**Submultiplier size**
46	2n-2	24, 22
45	5n	9
43	2n-1	22, 21
42	2n-2	22, 20
41	2n-1	21, 20
40	2n	20
39	2n-1	20, 19
38	2n	19
37	2n-1	19, 18
36	2n	18
35	2n-1	18, 17
34	2n	17
33	2n-1	17, 16
32	2n	16
31	2n-1	16, 15
30	2n	15
29	2n-1	15, 14
28	2n	14
27	2n-1	14, 13
26	2n	13
25	2n-1	13, 12
24	2n	12
23	n	23
22	2n	11
21	n	21
20	n	20
19	n	19
18	n	18

Table A.1: Recommended splitting algorithms for Galois field multiplier implementation on Virtex 5 (Continued)

Multiplier size	**Splitting rule**	**Submultiplier size**
17	n	17
16	n	16
15	n	15
14	n	14
13	n	13
12	n	12
11	n	11
10	n	10
9	n	9
8	n	8
7	n	7
6	n	6
5	n	5
4	n	4
3	n	3
2	n	2

APPENDIX B

AREA OPTIMIZATION ON KINTEX 7

Splitting multiplication algorithms preferred to implement the most area effective multiplier blocks for various field sizes on Kintex 7 are given in Table B.1 below.

Submultiplier sizes required for implementing the most efficient algorithm for the field size given in the first column are given in the third column. Splitting rule indicates the most efficient algorithm recommended for the corresponding field size. For example, for field size 283, splitting rule is "4n-1", which indicates that 4-way splitting algorithm should be used and since 283 cannot be splitted into four partitions of equal length there will be three partitions with length "n=71" and one partition with length "n-1=70".

Table B.1: Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7

Multiplier size	**Splitting rule**	**Submultiplier size**
283	4n-1	142, 71, 70
233	2n-1	117, 116
193	2n-1	97, 96
163	4n-1	82, 41, 40
142	2n-2	72, 70
141	2n-1	71, 70
128	2n	64
118	4n-2	60, 30, 28
117	4n-3	60, 30, 27

Table B.1: Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7 (Continued)

Multiplier size	**Splitting rule**	**Submultiplier size**
116	2n	58
113	2n-1	57, 56
98	2n-2	50, 48
97	2n-1	49, 48
96	4n	48, 24
95	2n-1	48, 47
93	2n-1	47, 46
82	2n	41
81	4n-3	42, 21, 18
78	2n	39
77	2n-1	39, 36
72	2n	36
71	2n-1	36, 35
70	2n	35
65	2n-1	33, 32
64	2n	32
63	2n-1	32, 31
60	4n	30, 15
59	4n-1	30, 15, 14
58	2n-2	30, 28
57	4n-3	30, 15, 12
56	2n	28
55	2n-1	28, 27
53	2n-1	27, 26
50	2n	25
49	2n-1	25, 24
48	4n	24, 12
47	2n-1	24, 23

Table B.1: Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7 (Continued)

Multiplier size	**Splitting rule**	**Submultiplier size**
46	2n	23
45	2n-1	23, 22
43	4n-1	22, 11, 10
42	2n	21
41	2n-1	21, 20
40	4n	20, 10
39	4n-1	20, 10, 9
38	2n	19
37	2n-1	19, 18
36	4n	18, 9
35	4n-1	18, 9, 8
34	2n	17
33	2n-1	17, 16
32	4n	16, 8
31	2n-1	16, 15
30	3n	10
29	2n-1	15, 14
28	4n	14, 7
27	3n	9
26	2n	13
25	2n-1	13, 12
24	2n	12
23	2n-1	12, 11
22	2n	11
21	2n-1	11, 10
20	2n	10
19	2n-1	10, 9
18	2n	9

Table B.1: Recommended splitting algorithms for Galois field multiplier implementation on Kintex 7 (Continued)

Multiplier size	**Splitting rule**	**Submultiplier size**
17	2n-1	9, 8
16	2n	8
15	2n-1	8, 7
14	2n	7
13	n	13
12	n	12
11	n	11
10	n	10
9	n	9
8	n	8
7	n	7
6	n	6
5	n	5
4	n	4
3	n	3
2	n	2