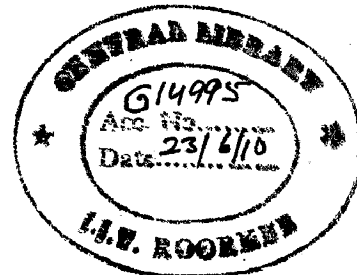
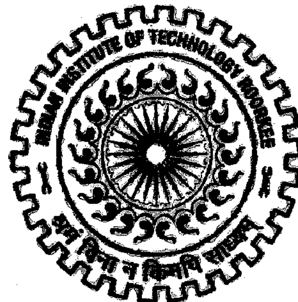
HIGH PERFORMANCE ADVANCE ENCRYPTION STANDARD IMPLEMENTATION ON FPGA

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
ELECTRONICS AND COMPUTER ENGINEERING
(With Specialization in Semiconductor Devices & VLSI Technology)

By
VISHWANATH PATEL





DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)

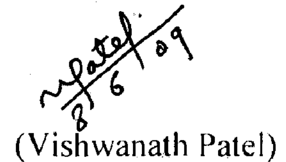
JUNE, 2009

CANDIDATE'S DECLARATION

I hereby declare that the work, which is presented in this thesis, entitled "**HIGH PERFORMANCE ADVANCE ENCRYPTION STANDARD IMPLEMENTATION ON FPGA**", being submitted in partial fulfillment of the requirements for the award of the degree of MASTER OF TECHNOLOGY with specialization in SEMICONDUCTOR DEVICES AND VLSI TECHNOLOGY in department of Electronics & Computer Engineering, Indian Institute of Technology, Roorkee is an authentic record of my won work carried out from July 2008 to June 2009, under the guidance and supervision of **Dr. R. C. Joshi** and **Dr. A. K. Saxena**, professors, Department of Electronics & Computer Engineering, Indian Institute of Technology, Roorkee, INDIA.

The result embodied in this dissertation, have not submitted for the award of any other Degree or Diploma.

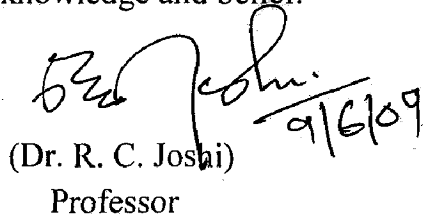
Date: 08/06/09
Place: Roorkee



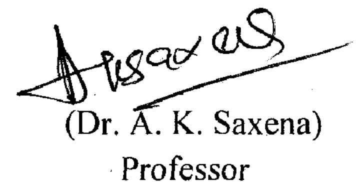
(Vishwanath Patel)

CERTIFICATE

This is to certify that the statement made by the candidate is correct to best my knowledge and belief.



(Dr. R. C. Joshi)
Professor



(Dr. A. K. Saxena)
Professor

Department of Electronics & Computer Engineering,
Indian Institute of Technology Roorkee,
Roorkee-247667, INDIA

Date: 09/06/09
Place: Roorkee

ACKNOWLEDGEMENT

It is my privilege and pleasure to express my profound sense of respect, gratitude and indebtedness to my guides, **Dr. R. C. Joshi** and **Dr. A. K. Saxena**, Professors, department of electronics and computer engineering, Indian Institute of Technology Roorkee, for their inspiration, guidance, constructive criticisms and encouragement throughout this dissertation work. The valuable hours of discussion and suggestions that I had with them have undoubtedly helped in supplementing my thoughts in the right direction for attaining the desired objective. I consider myself extremely fortunate for having got the opportunity to learn and work under their able supervision over the entire period of my association with them.

Thanks are due to lab staff of VLSI Design and Sponsored Project Laboratory, Department of Electronics & Computer Engineering, Indian Institute of Technology Roorkee for providing necessary facilities and support.

I am greatly indebted to all my friends, who have graciously applied themselves to the task of helping me with ample moral supports and valuable suggestions. Finally, I would like to extend my gratitude to all those persons who directly or indirectly helped me in the process and contributed towards this work.

ABSTRACT

In today's world most of the communication is done using electronic media. Data Security plays a vital role in such communication. In October 2000, the National Institute of Standards and Technology (NIST) selected the Rijndael as the Advanced Encryption Standard (AES) algorithm to replace the old Data Encryption Standard (DES).Till then four modes has been proposed by NIST.

A fourth and recent mode of operation of AES proposed by NIST in November 2006, SP800-38D, Galois/Counter Mode of Operation (GCM), that provide not only data security through encryption but also massage authentication.

Before GCM, SP800-38A only provided confidentiality and SP800-38B provided authentication. SP800-38C provided confidentiality using the counter mode and authentication. However the authentication technique in SP800-38C was not parallelizable and slowed down the throughput of the cipher. Hence, none of these three recommendations were suitable for high speed network and computer system applications.

This work includes, demonstration and analysis of FPGA architectures for, SP800-38A (AES-ECB) and SP800-38D (AES-GCM) modes of AES algorithm with the view of enhancing their performance. AES-GCM is a complex unit, AES-ECB (Electronic Code-Book) is used as one of its internal component; so this thesis first presents efficient iterative and fully pipelined based hardware architectures for AES-ECB mode and then finally presents fully pipelined and parallelized hardware architecture for AES-GCM.

Area optimization in above stated designs has been approached through implementing Sboxes of AES by Composite Field Arithmetic (CFA) technique and their comparison is made with respective LUTs (Look-Up tables) based designs.

Since modular multiplier is a very important unit of AES-GCM, which not only very crucial to determine speed of design but also covers 50% of overall area of the design, there are two multipliers has been analyzed and used in final AES-GCM design.

In this thesis, all the designs are implemented on multi-core Xilinx's virtex-4 FPGA platform.

CONTENTS

Title	Page No.
Candidate's Declaration	i
Certificate	i
Acknowledgement	ii
Abstract	iii
Contents	iv
List of Tables	viii
List of Figures	ix
Chapter 1 Introduction and Statement of the Problem	1
1.1 Introduction	1
1.2 Statement of the problem	3
1.4 Organization of Thesis	3
Chapter 2 Historical Review and General Consideration	6
2.1 Historical Review	6
2.2 Mathematical Background	7
2.2.1 Finite Fields	7
2.2.2 Operations over Binary Finite Fields GF (2^m)	8
2.2.3 Composite Field Arithmetic	11
2.3 Field Programmable Gate Arrays (FPGA)	13
2.3.1 Advantages of FPGA in Cryptographic Applications	14
2.3.2 Virtex-4	15
Chapter 3 Security Standard	17
3.1 Advanced Encryption Standard (AES)	17
3.1.1 AES Cipher	19
3.1.2 Byte Substitution (SubBytes)	19
3.1.3 Shift Rows	20

3.1.4 Mix Columns 21
3.1.5 Key Schedule 22
3.2 Confidentiality Mode of Operation Background 23
3.2.1 Electronic Codebook Mode (ECB) 23
3.2.2 Counter Mode (CTR) 24
3.3 Galois/Counter Mode (GCM) 26
3.3.1 Block Cipher 26
3.3.2 Input and Output Data 26
3.3.3 Types of Applications of GCM 27
3.3.4 GHASH Function 28
3.3.5 GCTR Function 29
3.3.6 GCM Specification 31

Chapter 4 Parallel Multiplier Designs for GCM 35
4.1 Mastrovito Multiplier 35
4.1.1 Matrix Vector Product 35
4.1.2 Mastrovito Multiplier Design using MVP 36
4.2 Karatsuba Algorithm Sub-quadratic Multiplier 37
4.2.1 KA Multiplier Formulation 38
4.2.2 Modulo Reduction 39
4.2.3 KA Multiplier Design for GCM 40
4.3 FPGA Implementation Results 42

Chapter 5 FPGA Implementation of AES-ECB Architectures 43
5.1 Compact Single Round AES Design 43
5.1.1 Single Round Compact Design 43
5.1.2 Composite Field Arithmetic based SubBytes function 46
5.1.3 Implementation Results 47
5.1.4 Performance Comparison with other Designs 47
5.2 High Speed Sub-pipelined Design 49
5.2.1 The AES Algorithm And Its Subpipelined Architecture 49

5.2.2 Detailed Hardware Implementation Architectures.................................. 50
5.2.3 Implementation Result and Comparison.. 57
Chapter 6 FPGA Implementation of AES-GCM Architecture.............................. 60
6.1 Modules Design... 60
6.1.1 AES Module.. 60
6.1.2 GHASH Module.. 62
6.2 High Speed Hardware Implementation of AES-GCM.................................... 64
6.2.1 Format of Data Packet of IPsec ESP... 64
6.2.2 Data Flow in GCM.. 65
6.2.3 Hardware Implementation Bidirectional GCM... 67
6.3 Verification of AES-GCM Functionality.. 72
6.3.1 IPsec Signal Generator.. 72
6.3.2 Verifying Both AES-GCM-AE and AES-GCM-AD on FPGA........................... 73

Chapter 7 Conclusion and Future Work... 75
7.1 Conclusion.. 75
7.2 Future Work... 76

References... 77
Papers Published.. 81
Appendices... 82
Appendix A : Test-Vectors for AES-GCM.. 82
Appendix B : Simulation Result of Implemented Designs..................................... 84

LIST OF TABLES

Table No.		Page No.
Table 2.1	: Resources of Virtex-4 FPGA Family Members	15
Table 4.1	: Area of KA Multiplier with varied ending conditions	41
Table 4.2	: Multiplier's Place and Route Results Summary	42
Table 5.1	: Synthesis and Place & Route results of compact AES designs	48
Table 5.2	: The logic and routing delay of compact AES designs	48
Table 5.3	: Performance comparison of compact AES designs	48
Table 5.4	: Gate counts and critical paths functional blocks in the SubBytes Transformation [17]	52
Table 5.5	: Comparison of FPGA implementation of the AES algorithm	58
Table 5.6	: Power consumed	58
Table 6.1	: Comparison between Iterative and Pipelined AES	61
Table 6.2	: Comparison between different GHASH architectures	63
Table 6.3	: Place and Route Results Summary of other important units of AES-GCM	70
Table 6.4	: Full AES-GCM's Place and Route Results Summary	70
Table 6.5	: Power analysis of the designs	70

LIST OF FIGURES

Figure No.		Page No.
Figure 2.1	: Simplified virtex-4 CLB	16
Figure 3.1	: (a) AES Encryption, (b) AES Decryption algorithm	18
Figure 3.2	: AES Round State Array Transformation	18
Figure 3.3	: (a) Visual diagram, (b) Block diagram of composite Sbox	20
Figure 3.4	: AES Shift Rows	21
Figure 3.5	: AES 128 bit Key Schedule Round	22
Figure 3.6	: ECB Encryption and ECB Decryption. [1]	24
Figure 3.7	: CTR Encryption and CTR Decryption [1]	25
Figure 3.8	: $GHASH_H(X_1 \parallel X_2 \parallel ... \parallel X_m) = Y_m$.[4]	29
Figure 3.9	: $GCTR_K(ICB, X_1 \parallel X_2 \parallel ... \parallel X_n^*) = Y_1 \parallel Y_2 \parallel ... \parallel Y_n^*$.[4]	30
Figure 3.10	: AES-GCM-AE_K(IV, P, A) = (C, T).[4]	32
Figure 3.11	: AES-GCM-AD_K(IV, C, A, T) = P or FAIL.[4]	34
Figure 4.1	: Mastrovito Multiplier for GCM	37
Figure 4.3	: (a) Abstract view, and (b) Full view of the Karatsuba Multiplier.	41
Figure 4.4	: Multiplier performance comparison	42
Figure 5.1	: Compact single round AES FPGA architecture	44
Figure 5.2	: Round unit	44
Figure 5.3	: Key Scheduler Unit	45
Figure 5.4	: State diagram	46
Figure 5.5	: The architecture of Subpipelining	50
Figure 5.6	: Implementation of the subBytes Transformation	51
Figure 5.7	: Implementations of individual blocks: (a) Multiplier in $GF(2^4)$; (b) Multiplier in $GF(2^2)$; (c) Squarer in $GF(2^4)$; (d) Constant multiplier ($\times \lambda$); and (e) Constant multiplier ($\times \phi$)	52
Figure 5.8	: Efficient implementation of the MixColumns (red dashed rectangle only) and InvMixColumns transformation	55

Figure 5.9 : Different cutest of Round and Subkey unit for sub-pipelined architecture........ 56
Figure 5.10 : Scatter graphs for comparison of (a) Power, (b) Throughput, (c) No. of slices and (d) both (b)&(c), of subpipelined AES designs........... 59
Figure 6.1 : AES CTR over ECB Mode Cipher Structure........ 62
Figure 6.2 : GHASH Hardware Architecture........ 63
Figure 6.3 : The Use of GCM in IPsec ESP [31]........ 65
Figure 6.4 : (a) The Data Flow of GCM Encryption (b) The Data Flow of GCM Decryption........ 66
Figure 6.5 : AES-GCM Encryption Architecture........ 67
Figure 6.6 : AES-GCM Decryption Architecture. 69
Figure 6.7 : Comparison of various units of Full AES-GCM........ 71
Figure 6.8 : Area and power comparison of two type of AES-GCM........ 71
Figure 6.9 : Throughput and throughput per slice comparison of two type of AES-GCM........ 71
Figure 6.10 : 16-bit LFSR for IPsec ESP Signal Generator........ 72
Figure 6.11 : AES-GCM Verification System. 73
Figure 6.12 : AES-GCM Hierarchical HDL Codes Design........ 74

CHAPTER 1

INTRODUCTION AND STATEMENT OF THE PROBLEM

1.1 Introduction

In the past traditional communications were based on letters, payments were done using checks or cash, and secret documents were saved in sealed boxes. Today everything is changed, and is changing quickly. Everyday more people buy cell phones, the number of e-mail users goes up, and more people pay their payments over the internet. Paperless office strategies save and process documents in electronic format. These trends are going to make the life easier but at the same time produce security risks. The rapid development of electronic communication systems requires a secure infrastructure, too. Cryptography is the mathematical tool which is used by security engineers to secure data against unauthorized access or manipulation.

Like every other useful service, security will not be achieved for free. Implementing cryptography tasks costs time, money (chip area), and energy. To meets these constraints of upcoming modern applications, intensive work is required in this field.

Implementing cryptographic algorithms on reconfigurable hardware provides major benefits over ASIC (application-specific integrated circuit) and software platforms, since they offer high speed similar to ASIC and high flexibility similar to software. ASIC implementations are fast but must be designed all the way from behavioral description to the physical layout. They have to follow an expensive and time consuming fabrication process. Software implementations offer high flexibility but they are not fast enough for the applications where time factor is vital.

In nutshell, reconfigurable devices are attractive, since the time and costs of VLSI design and fabrication can be reduced. Moreover, they offer high potential for reprogramming and experimenting on multiple architectures or several revisions of the same architecture, which enhance robustness of security system.

The AES algorithm is a private-key encryption algorithm. In January 1997, the National Institute of Standards and Technology (NIST) invited proposals for new algorithms for

the Advanced Encryption Standard (AES) to replace the old Data Encryption Standard (DES). After two rounds of evaluation on the 15 candidate algorithms, NIST selected the Rijndael as the AES algorithm [1] in October 2000.

Since then, the NIST has published a total of four recommendations for Block Cipher AES Modes of Operation, specifically SP800-38A [1], SP800-38B [2], SP800-38C [3], and SP800-38D [4]. A block cipher mode of operation is an algorithm that uses a symmetric key block cipher to provide confidentiality, authentication or both for information security.

In SP800-38A, NIST recommends five confidentiality modes of operation for use with an underlying symmetric key block cipher algorithm: Electronic Codebook (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher Feedback (CFB) mode, Output Feedback (OFB) mode, and Counter (CTR) mode. These five modes can be separated into two groups: one is a non-feedback mode group, including ECB and CTR; one is a feedback mode group, including CBC, CFB and OFB. In the feedback modes, the current computation/execution step depends on the result of the previous step. Therefore, to implement these kinds of modes in hardware, an iterative architecture is typically adapted for low throughput requirements rather than a pipelined architecture. In contrast, the use of ECB or CTR mode, or non-feedback modes, supports pipelined or parallelized architecture designs for processing high-speed data flows.

As the forth security standard of Block Cipher Mode of Operation, SP800-38D, *Galois/Counter Mode of Operation (GCM)*, fills the need above. GCM features the use of an approved symmetric key block cipher with a block size of 128 bits and a universal hash function that is defined over a binary Galois field. The most recently approved symmetric key block cipher with a block size of 128 bits is the Advanced Encryption Standard (AES) algorithm that is specified in Federal Information Processing Standard (FIPS) Pub.197 [3]. The specified universal hash function in GCM is defined over a binary Galois field (GF) and is a 128-bit polynomial multiplier over GF (2^{128}), called GHASH. GHASH can provide a secure, parallelizable, and efficient authentication mechanism. For the confidentiality mechanism of GCM, the CTR mode embedded by ECB mode, called GCTR, is adopted using an underlying block cipher. GCM, i.e. SP800-

38D, was officially published in November 2006. However except one or two, there are no known high performance FPGA (field programmable gate array) architectures or implementations of this standard.

1.2 Statement of the problem

The objective of this thesis is to demonstrate, analyze and implement FPGA architectures for, SP800-38A (ECB) and SP800-38D (GCM) modes of AES algorithm with the view of enhancing their performance.

In above stated two modes, GCM comparatively a big and complex design. It includes AES engine (SP800-38A), GHASH (modular multiplier), and Key-expanded modules. So to achieve our objectives, the problem can be subdivided as follows:

1. To implement iterative and pipelined architectures of AES-ECB (Electronic Code-Book) mode on FPGA and investigate their performance.
2. To investigate different type of modular multipliers used in GHASH and analyze their performance by implementing on FPGA.
3. To integrated various GCM modules together, along with control logic to implement the highly parallel, pipelined and entire new security standard, AES-GCM.
4. To optimize the AES designs in term of area, CFA (Composite Field Arithmetic) technique analyzed for making Sbox (sub unit of AES).
5. To verify the feasibility, efficiency and cost of each hardware module of AES. the architectural designs synthesized, timing simulated, and downloaded to the FPGA virtex-4 platform.

1.3 Organization of Thesis

This thesis is organized as follows: **Chapter 2** provides historical review of different implemented architectures in this thesis. An overview of the mathematical definitions over GF and composite field arithmetic is provided as mathematical background. Introduction to the FPGA device structures and its advantage in security systems also

provided in this chapter. **Chapter 3** presents security standards, AES, other confidential modes of AES and GCM. **Chapter 4** describe two type of parallel multiplier used in GCM and their implementation result. In **chapter 5**, the proposed hardware architectures of AES are presented. The proposed hardware architectures of AES-GCM are presented in **chapter 6**. The bit parallel multiplier over GF, and the pipelined AES discussed in chapter 4 and 5 are chosen as the modules to build AES-GCM. A methodology, to verify the AES-GCM hardware implementation is also discussed during this chapter. Finally, **chapter 7** provides conclusion and future work.

CHAPTER 2

HISTORICAL REVIEW AND GENERAL CONSIDRATION

This chapter provides the review and concepts necessary in order to understand implemented architectures: Section 2.1 provides historical review. Section 2.2 introduces to the concepts of finite fields, and Composite Field Arithmetic (CFA); an area reduction technique. Section 2.3 gives quick overview of FPGA and describes the advantages of FPGA technology for cryptographic system.

2.1 Historical Review

In November 2001, after a 5-year standardization process in which fifteen competing designs were presented and evaluated, Rijndael [32] (developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen) was selected as the most suitable for Advance Encryption Standard. Details of [32] given in section 3.1.

Since then, the NIST has published a total of four recommendations for Block Cipher AES Modes of Operation, specifically SP800-38A [1], SP800-38B [2], SP800-38C [3], and SP800-38D [4].

Out of five confidentiality sub-modes of SP800-38A; Electronic Codebook (ECB) mode is focused in this work. FPGA based architecture point of view, all important research of this mode can be divided into two fields; small iterative design and high speed pipelined design.

There are very few number of designs proposed for small design in literature, [18], [19], [20], [21] and [22] are some important one. Initially, memory (RAM) based non-parallelization Sbox implementation is used, but it is not area efficient. Although, parallelization of Sbox [19] helped in area reduction along with some speed improvement, but there not any specific area reduction technique adopted. Internal pipelining of single round is also employed in [18] which is effective for speed but not for area. In spite of that their speed is again limited by memory based Sbox. Some efforts of area reduction are also spoiled because of their complex control unit, since control unit can covers lots of

area if not properly designed. In 2005, CAF approached adopted for cryptographic systems, which is very efficient in area reduction, but as per of author knowledge up to now not any design based on this approach claimed in small AES design.

High speed pipelined AES architectures ([16], [17], [21], [26], [27], [28] and [29]) is quite famous among the researcher because of its requirement in modern application. Initially, parallelized outer 10 stage pipelined [27] or inner sub-pipelined ([26], [29]) are generally employed for speed improvement but their speed is limited by memory based Sbox. This problem is solved in [17], by making highly sub-pipelined CFA based Sbox implementation and achieved highest 21.5 Gbps throughput, but design is tested for 3 and 7 pipeline stages only.

In November 2006, fourth mode of AES proposed; SP800-38D or GCM. Research on hardware architectures or implementations of GCM is fairly small. This is likely due to the new mode of operation. As per knowledge of author there is not any design found on FPGA. Although design [33] demonstrates AES-GCM ASIC based architecture, using 0.18 um CMOS standard cell library, but it can be a good design to implement on FPGA.

From the above historical review, it can be concluded that, the following major research gaps still exist.

i. Not any particular architecture technique of area optimization has been used till now in small (iterative) AES design. So CFA area reduction technique can be proved efficient for area optimization by implementing Sbox of small (iterative) AES design.

ii. In high speed pipelined architecture, maximally 7 stage pipelined has been employed till now. But still speed improvement can be possible by further exploration of pipelined architecture to more stages.

iii. AES-GCM being a recent mode, has been implemented only on ASIC, but not over FPGA. So detailed analysis and implementation of GCM mode on FPGA can be a good work to be carried out.

Thus, this dissertation work is to effectively fill above stated research gaps.

2.2 Mathematical Background

The fundamentals of AES and GHASH (Multiplying unit of GCM) are based on operations over the finite field. This section provides an introduction to these operations. The concepts and methods have been gathered from [6] and [7].

2.2.1 Finite Fields

A field can be considered as a set whose elements form a group G under two operations: multiplication indicated by symbol "·" and addition indicated by symbol "+". These operations obey the basic algebraic properties. The relative finite field concepts are list as follows:

Concept 1. (F, +, ·) is a field if the following properties hold:

- The elements of F form a group under addition.
- The non-zero elements of F form a group under multiplication.
- The addition and multiplication operations are commutative, i.e. $x + y = y + x$ and $xy = yx$ for all $x, y \in F$.
- The multiplication operation can be distributed through the addition operation, i.e. $x\cdot (y + x) = x\cdot y + x\cdot z$ for all x, y, and $z \in F$.

Concept 2. A field F with a finite number of elements is a finite field.

Concept 3. A non-zero element of a finite field F is said to be a primitive element or generator of F if its powers cover all nonzero field elements.

Concept 4. A unique finite field exists for every prime number. These fields are denoted $GF(p^m)$ where p is prime and m is a positive integer. One kind of field which is commonly used in cryptography applications is the binary finite fields $GF(2^m)$ where m is a large integer.

Concept 5. A basis for $GF(2^m)$ over GF(2) is a set of m linearly independent elements of $GF(2^m)$. Any element of $GF(2^m)$ can be represented as an algebraic sum of the basis elements.

The binary field $GF(2^m)$ contains 2^m elements. Each element is represented by the selected basis. The most common representation is based on polynomial basis. With the polynomial basis $\alpha = \{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$, the elements of $GF(2^m)$ can be represented as polynomial of degree m-1 as follows:

$$GF(2^m) = \{A | A = a_0 + a_1\alpha + \cdots + a_{m-1}\alpha^{m-1} \text{ where } a_j \in GF(2), 0 \le j \le m-1\}$$

where α is the root of an irreducible polynomial F(x) of degree m over GF(2).

Let

$$F(x) = 1 + f_1 x + f_2 x^2 + \cdots + f_{m-1}x^{m-1} + x^m$$

where $f_i \in GF(2), 0 \le i \le m-1$. The irreducible polynomial F(x) is often referred to as the field polynomial. The arithmetic in AES-GCM is based on polynomial basis and uses the polynomial $F(x) = 1 + x + x^2 + x^7 + x^{128}$ as field polynomial.

2.2.2 Operations over Binary Finite Fields GF (2^m)

Both operations, field addition and field multiplication, map a pair of field elements A and B onto another field element C, all A, B, and $C \in GF(2^m)$. The following introduction on field addition and multiplication is based on polynomial basis. The field elements A, B, and C are the following polynomials, respectively:

$$A(\alpha) = a_0 + a_1\alpha + \cdots + a_{m-1}\alpha^{m-1}$$
$$B(\alpha) = b_0 + b_1\alpha + \cdots + b_{m-1}\alpha^{m-1}$$
$$C(\alpha) = c_0 + c_1\alpha + \cdots + c_{m-1}\alpha^{m-1}.$$

2.2.2.1 Field Addition

Over a finite field $GF(2^m)$, a field addition of two elements A and B consists of adding the two polynomials together. Because the coefficients in A and B are over $GF(2)$ and each pair of coefficients are added independently, their sum C is written as

$$C(\alpha) = A(\alpha) + B(\alpha) = \sum_{i=0}^{m-1} (a_i + b_i)\alpha_i \qquad (2.1)$$

The pair of coefficients addition $a_i + b_i$ in Eq.(2.1) is performed modulo 2 and translated to an exclusive- OR (XOR) operation in FPGA technology. That is to say that the field addition in Eq.(2.1) is computed by an m-bit XOR operation and does not require a carry chain.

2.2.2.2 Field Multiplication

- **Bit Serial Multiplier**

Field multiplication over a finite field $GF(2^m)$, is executed by straightforward multiplying two polynomials $A(\alpha)$ and $B(\alpha)$, then dividing the resulting 2m-bit polynomial by F(α); the m-bit remainder is the result C(α). The product C of field elements A and B is expressed as

$$C(\alpha) = A(\alpha) \times B(\alpha) mod\ F(\alpha) = \sum_{i=0}^{m-1} c_i\alpha_i = \sum_{i=0}^{m-1}\sum_{i=0}^{m-1} a_ib_j\alpha^{i+j}\ mod\ F(\alpha) \quad (2.2)$$

A simple method of computing this involves the use of a linear feedback shift register (LFSR). The pseudo code for this multiplier given below simply loops through the summation in Eq.(2.2) and accumulates a modulo reduced answer. The LFSR contains one of the operands A, and depending on its most significant bit, the field polynomial is XORed to the LFSR at each step. The result of the multiplication is generated in the register C by the end of m iterations. This register adds the value of A at each step depending on the coefficients of the other multiplicand H. This design is called a serial multiplier design, totally m iteration are needed for calculating a multiplication over $GF(2^m)$ if $A(\alpha)$ can be loaded in parallel. Other multiplier designs exist such as the parallel multiplier that is able to compute $C(\alpha)$ in a single iteration. More details will be provided in next section.

Algorithm 2.1: $GF(2^m)$ multiplier

Input: $A, H \in GF(2^m)$; $F(x)$ *Field Polynomial.*
Output: $C(\alpha)$
$C = 0$
$for\ i = 0\ to\ m\ do$
  $if\ H_i = 1\ then$
    $C \leftarrow C \oplus A$
  $end\ if$
  $if\ A_{127} = 0\ then$
    $A \leftarrow rightshift(A)$
  $else$
    $A \leftarrow rightshift(A) \oplus F(\alpha)$
  $end\ if$
$end\ for$
$return\ C$

- **Bit Parallel Multiplier**

Compared to the bit serial multiplier which needs m clock cycles to complete a multiplication over $GF(2^m)$, a bit parallel multiplier can complete computation in only 1 clock cycle over the same GF. (Because the circuit delays are very different between the bit serial multiplier and the bit parallel multiplier, the minimum clock period of clock for parallel multiplier is much larger than the minimum one for serial multiplier. i.e., 1 clock cycle computation time for parallel multiplier should be roughly equal to several or tens clock cycles computation time for serial multiplier.)

A dedicated polynomial basis finite field bit parallel multiplier has been proposed in [7], called the Mastrovito multiplier. This multiplier is adapted to a fixed field polynomial $F(\alpha)$. The implementation procedure of the Mastrovito multiplier and Karatsuba multiplier are described in chapter 4.

2.2.3 Composite Field Arithmetic

The non-LUT-based implementations of the AES algorithm are able to exploit the advantage of subpipelining further. Nevertheless, these approaches may have high hardware complexities. Although two *Galois Fields* of the same order are isomorphic, the complexity of the field operations may heavily depend on the representations of the field elements. Composite field arithmetic can be employed to reduce the hardware complexity. We call two pairs $\{GF(2^n), Q(y) = y^n + \sum_{i=0}^{n-1} q_i y^i, q_i \in GF(2)\}$ and $\{GF((2^n)^m), p(y) = x^m + \sum_{i=0}^{m-1} p_i x^i, p_i \in GF(2^n)\}$ a composite field [12] if

- $GF(2^n)$ is constructed from $GF(2)$ by $Q(y)$,
- $GF((2^n)^m)$ is constructed from $GF(2^n)$ by $p(x)$.

Composite fields will be denoted by $GF((2^n)^m)$, and a composite field $GF((2^n)^m)$ is isomorphic to the field $GF(2^k)$ for $k = nm$. Additionally, composite fields can be built iteratively from lower order fields. For example, the composite field of can be built iteratively from using the following irreducible polynomials [7]:

$$\begin{cases} GF(2) \Rightarrow GF(2^2): & P_0(x) = x^2 + x + 1 \\ GF(2^2) \Rightarrow GF((2^2)^2): & P_1(x) = x^2 + x + \phi \\ GF((2^2)^2) \Rightarrow GF(((2^2)^2)^2): & P_2(x) = x^2 + x + \lambda \end{cases} \tag{2.3}$$

where $\phi = \{01\}_2$ and $\lambda = \{1100\}_2$. Meanwhile, an isomorphic mapping function $f(x) = \delta \times x$ and its inverse need to be applied to map the representation of an element in $GF(2^8)$ to its composite field and vice versa. The 8×8 binary matrix are decided by the field polynomials of $GF(2^8)$ and its composite fields. Such a matrix can be found by the exhaustive-searchbased algorithm in [12]. The δ matrix corresponding to $P(x) = x^8 + x^4 + x^3 + x + 1$ and the field polynomials in Eq.(2.3) can be found as below:

$$\delta = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Taking the isomorphic mapping into consideration, not all the transformations in the AES algorithm are suitable to be implemented in the composite field. In order to facilitate substructure sharing, the constant multiplications in the MixColumns/InvMixColumns transformation are implemented by first computing $\{02\}_{16}S_{i,j}$, $\{04\}_{16}S_{i,j}$ and $\{08\}_{16}S_{i,j}$, then adding those terms corresponding to the nonzero bits in the constants. For example, the constant multiplication of $\{0b\}_{16} = \{00001011\}_2$ can be computed by adding $S_{i,j}$, $\{02\}_{16}S_{i,j}$ and $\{08\}_{16}S_{i,j}$. In this approach, the $\{02\}_{16}S_{i,j}$, $\{04\}_{16}S_{i,j}$ and $\{08\}_{16}S_{i,j}$, can be computed once and shared by all the constant multiplications. Meanwhile, the number of terms, which need to be added is determined by the number of nonzero bits in the constants. Using the δ matrix defined in (8), the constant multiplications of $\{02\}_{16}$ and $\{03\}_{16}$ in $GF(2^8)$ in the MixColumns are mapped to constant multiplications of $\{5f\}_{16}$ and $\{5e\}_{16}$ in the composite field, respectively. Although the hardware overhead of the mapping of constants can be eliminated by computing the mapping beforehand, the composite field representations of $\{02\}_{16}$ and $\{03\}_{16}$ have more nonzero bits, which makes the constant multiplications more expensive. The same argument also holds for the constant multiplications used in the InvMixColumns transformation, where $\{09\}_{16}$, $\{0b\}_{16}$ and $\{0c\}_{16}$ are mapped to $\{75\}_{16}$, $\{2a\}_{16}$ and $\{57\}_{16}$ in the composite field, respectively. The only exception is that the composite field representation of $\{0d\}_{16}$, which is $\{09\}_{16}$, has one less nonzero bit, but this is offset by the larger number of nonzero bits in the composite field representations of the other three constants.

Furthermore, $\{10\}_{16}S_{i,j}$, $\{20\}_{16}S_{i,j}$ and $\{40\}_{16}S_{i,j}$ also need to be computed as a result of the higher-weight nonzero bits in $\{75\}_{16}$, $\{2a\}_{16}$ and $\{57\}_{16}$, which adds more complexity to the hardware implementations. Therefore, it is more efficient to implement

the MixColumns/InvMixColumns in the original field $GF(2^8)$. The ShiftRows/InvShiftRows is a trivial transformation, only cyclical shifting is involved, and thus its implementation does not depend on the representation of *Galois Field* elements. Meanwhile, the field addition, which is simply XOR operation, has the same complexity in the composite field and the original field. Additionally, the affine/inverse affine transformation can be combined with the inverse isomorphic/isomorphic mapping. Based on the above observations, it is more efficient to carry out only the multiplicative inversion in the SubBytes/InvSubBytes in the composite field, while keep the rest of the transformations in the original field $GF(2^8)$.

2.3 Field Programmable Gate Arrays (FPGA)

The thesis presents the architecture of FPGA implementation of AES security algorithms. The common implementation approaches are corresponding to three different technologies. They are:

- Application Specific Integrated Circuits (ASICs)
- Software-Programmed General Purpose CPU (SPGPC)
- Field Programmable Gate Arrays (FPGAs)

ASICs are specifically designed for a fixed solution, and are thus very efficient. However, the circuit cannot be changed after fabrication. This requires a redesign of the chip if any modification needs to be done.

SPGPCs are a flexible solution. CPUs execute a set of instructions to perform an algorithm. By changing the software code, the functionality of the system is altered without touching the hardware. But the SPCGPC's efficiency is much lower than that of an ASIC.

FPGAs offer a compromise between the ASIC and the SPGPC, achieving higher performance than software, while maintaining a higher level of flexibility than hardware.

2.3.1 Advantages of FPGA in Cryptographic Applications

The following attributes of the FPGA technology are particularly advantageous for cryptographic applications [8].

Algorithm Agility: More and more security applications intend to be algorithm independent and allow switching encryption algorithms on the flying. The encryption algorithm can be chosen through the negotiation made by two communication parties.

Algorithm Upload: From a cryptographic point of view, algorithm upload can be necessary because a current algorithm is out of date or broken; a new algorithm is created. The security designer of the corresponding security company can upload the new bit streams of security standard to reconfigure FPGA device through the networks.

Throughput: Although FPGA implementations are typically slower than ASIC implementations, FPGA implementations are obviously faster than software implementations. In a cryptosystem, if a software solution is chosen for clients, then, a FPGA implementation should be adapted for servers in high-speed backbones.

Cost Efficiency: The production costs of an ASIC are often too high for a small number of servers in security systems. Thus, the use of FPGAs is a common alternative. Furthermore, this is the one of reasons why the FPGA is chosen for security research in institutes and universities.

Therefore, it is often best to choose an FPGA to implement cipher, such as AES-GCM standard. The CMC-FPGA-prototype-platform was chosen in this thesis for prototyping since it represents a generalized multi-core platform, appropriate for security applications. This FPGA platform will be discussed next.

2.3.2 Vertex-4

A traditional FPGA is usually an integrated circuit consisting of

- Configurable Logic Blocks (CLBs),
- Input/Output Blocks (IOBs) and
- Programmable routing resources.

More specifically, Table 2.1 shows all the main resources of the Virtex-4 xc2vp100 targeted in this thesis.

Table 2.1: Resources of Virtex-4 FPGA Family Members.

Devices	No. of slices (1CLB=4 Slices)	No. of 4 input LUTs	No. of RAM Block (18K)	Max User I/Os	Max. operating frequency (MHz)
XC4VLX25	10,752	21504	72	448	500
XC4VLX100	49,152	98,304	240	960	500
XC4VLX200	89,088	178,176	360	960	500

- **Configurable Logic Blocks (CLBs)**

The CLBs in the Virtex-4 are comprised of both combinational and sequential logic. The combinational logic can be configured to become possible Boolean functions. Flip-Flops are provided to support sequential logic and can be utilized or bypassed depending on the configuration.

One CLB has four slices. Each slice is identical and contains:

- Two function generators F and G
- Two storage elements
- Arithmetic logic gates
- Multiplexers
- Fast carry look-ahead chain
- Horizontal cascade chain

A general slice structure of Virtex-4 is shown in Figure 2.1. The function generators F and G can be configured as 4-input look-up tables (LUTs), as 16-bit shift registers, or as

16-bit distributed SelectRAM+ memory. The multiplexers, MUXF5 and MUXFX can provide any function of five, six, seven, or eight inputs when combined with function generators. The two storage elements can be configured either edge-triggered flip-flops or level-sensitive latches. Each CLB has internal fast interconnect and connects to a switch matrix to access general routing resources.

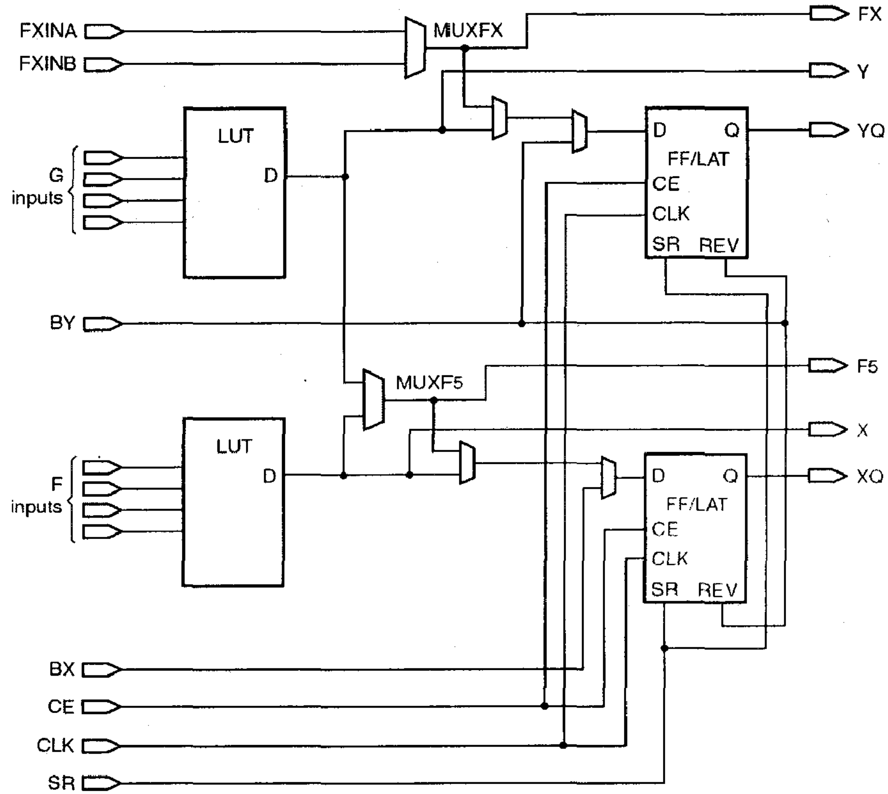


Figure 2.1: Simplified virtex-4 CLB.

CHAPTER 3

SECURITY STANDARD

This chapter provides the details of various modes of AES, which are implemented in this thesis: Section 3.1 introduces to Advance Encryption Standard security algorithm, Section 3.2 provides background of confidentiality mode of operation and Section 3.3 gives detail of GCM mode of AES.

3.1 Advanced Encryption Standard (AES)

The Advanced Encryption standard is a 128 bit block cipher that has been widely used since its acceptance in 2001 [5]. The design of AES was intended to be a more secure replacement of DES (Data Encryption Standard). Many efficient hardware and software designs have been documented, taking into consideration various tradeoffs of speed and area resources. The following sections will provide a general functional description of AES with an increased focus on the hardware design of AES components.

3.1.1 AES Cipher

Figure 3.1 showing schematic of AES encryption and decryption. Different hardware datapaths can be created from these modular round structures. An iterative design can be made by simply adding a 128 bit data register at the end of the round structure. After a maximum of 14 cycles the AES encryption result can be obtained. This iterative design can be unrolled to create a pipelined implementation that has registers placed between round blocks. This is an outer pipelined AES design and a 128 bit output can be generated at each clock cycle with a full pipeline. There is enough flexibility, however, in choosing locations of the pipelined registers. Within each of the round components, additional pipelined stages can be added within the Sub-bytes operation which will be described in Section 3.1.2. This is labeled as an inner pipelined AES design, and although a higher latency and area is present, higher throughputs are possible.

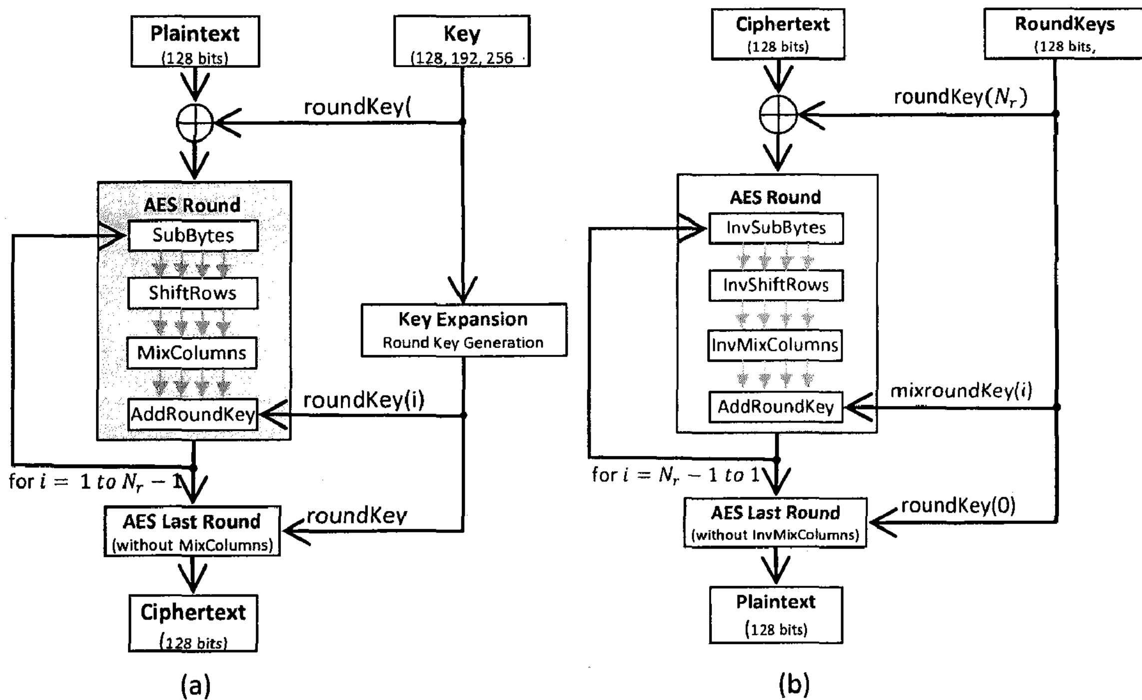


Figure 3.1: (a) AES Encryption, (b) AES Decryption algorithm.

The 128 bit plain text input is mapped into a state array which is a 4x4 block of 8 bit words that is manipulated in each round. For the following sections the state array block will be used to describe the different round operations so it is important to understand how the input is transformed into the state array. Figure 3.2 shows this transformation, by filling bytes of data into the state array by columns. After the AES encryption round, the last state array outputted is transformed back into a 128 bit stream.

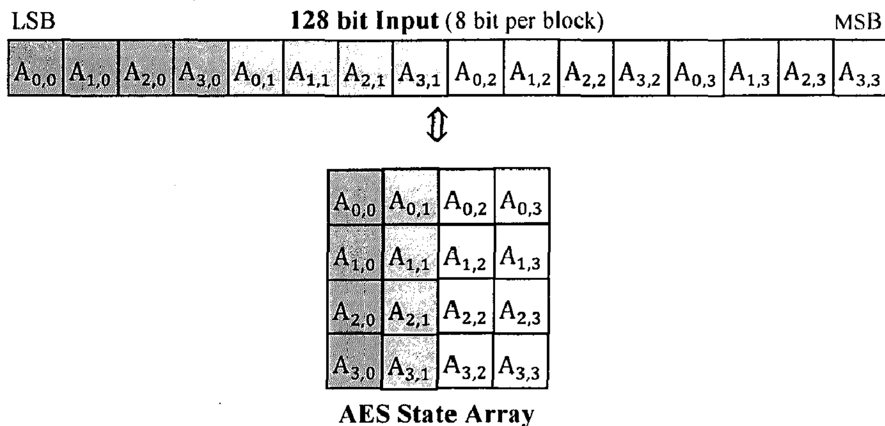


Figure 3.2: AES Round State Array Transformation.

3.1.2 Byte Substitution (Subbytes)

The subbytes operation uses multiple substitution box components (Sbox) each of which performs an 8 bit substitution. Each 8-bit word of data in the state array, is substituted using the Sbox. This results in 16 Sbox components used for each round block, and is the most hardware area consuming part of an AES round. The Sbox computation is essentially a multiplicative inverse in $GF(2^8)$ followed by an affine transformation which is a linear mapping from one vector space to another [9]. A lookup table of 2^8 values can be used to implement the Sbox component, but it can also be mathematically computed using logic gates.

- **Sbox Designs**

Rijimen, one of the creators of AES showed in [10] a method of computing the Sbox by breaking operations in $GF(2^8)$ down to a composite field $GF((2^4)^2)$ resulting in significant hardware area savings which would otherwise not be possible using look-up table implementations. The inverse formula for the Sbox is given in its reduced version, in Eq.(3.1), where λ is $(1100)_2$. The addition, multiplication, and inverse operations are computed in $GF((2^4)^2)$, and can be further broken down to the smaller composite fields, $GF((2^2)^2)$ and $GF(2^2)$, using the divide and conquer method.

$$a'x + b' = (ax + b)^{-1} = a(a^2\lambda + b(a + b))^{-1}x + (b + a)(a^2\lambda + b(a + b))^{-1} \quad (3.1)$$

Figure 3.3 shows a visual diagram of the composite Sbox. The isomorphic mapping to the composite field, $(GF(2^8) \rightarrow GF((2^4)^2)$ can be implemented using a matrix vector multiplication. The affine transformation consists of a linear transformation followed by a translation which can be achieved by a matrix vector multiplication and vector addition respectively. The isomorphic mapping and affine transformation both use fixed matrices that are sparse so the computation costs of these operations are minimal [9].

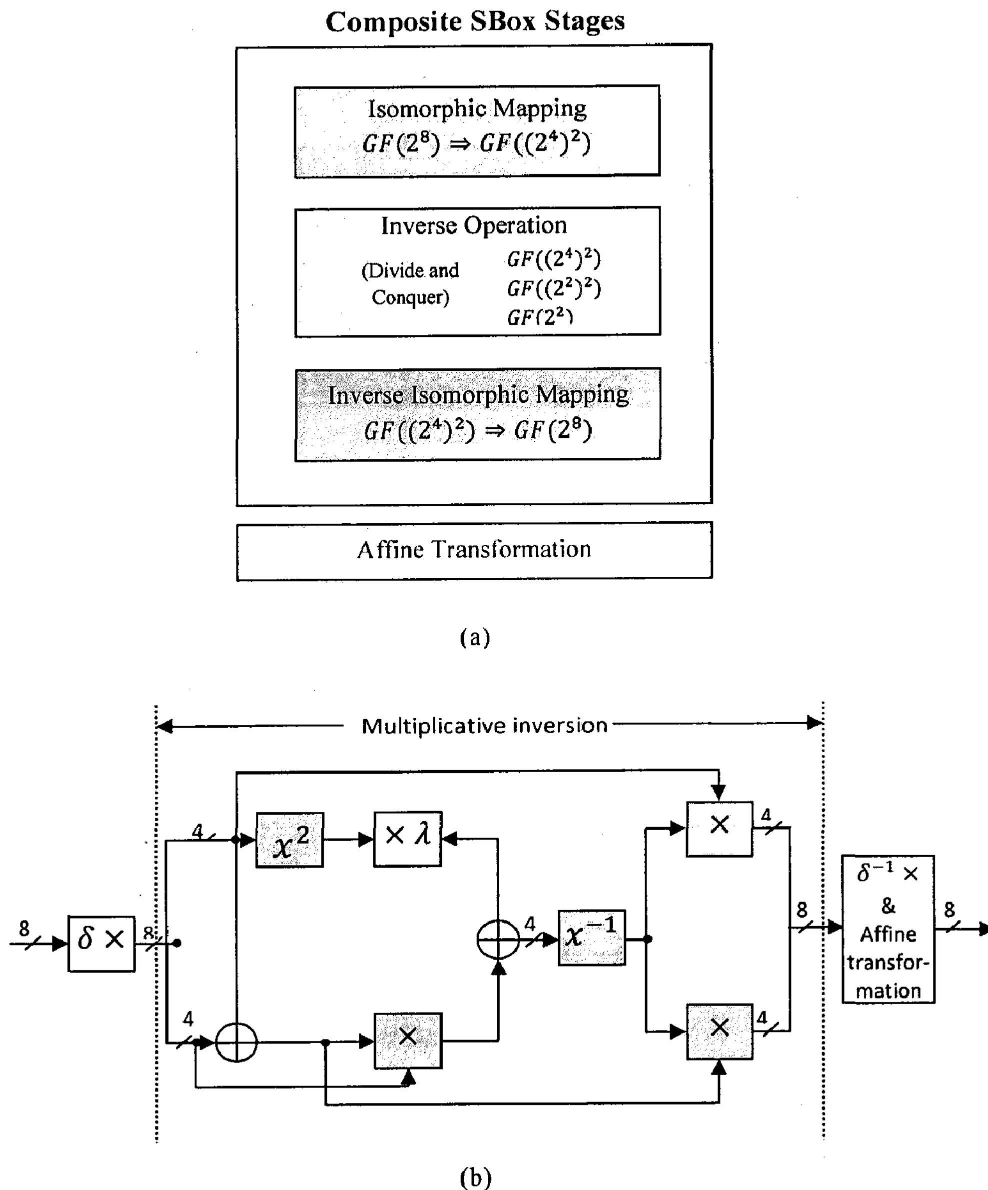


Figure 3.3: (a) Visual diagram, (b) Block diagram of composite Sbox.

3.1.3 Shift Rows

The Shift Rows operations consists of cyclically moving elements around in all but the first row of the 128 bit input block. The rows are left shifted by 1, 2, and 3 times respectively for rows 2, 3 and 4. The following mapping illustrates this process. In

hardware no logic is required for this step and simple wire connections are used for this step to route the input to the output.

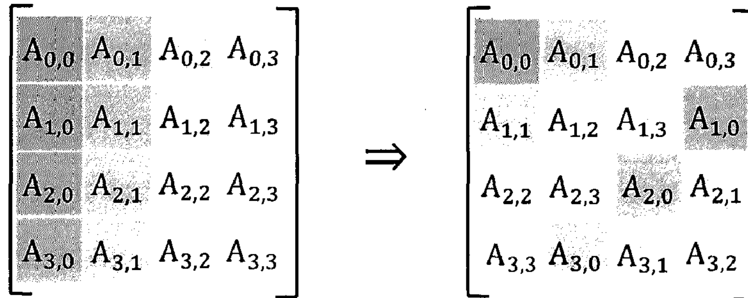


Figure 3.4: AES Shift Rows

3.1.4 Mix Columns

The mix columns operation consists of a multiplication and reduction operation over $GF(2^8)$. Each column of the state array is multiplied by the polynomial $3x^3 + x^2 + x + 2$ and reduced modulo the field generating polynomial $x^4 + 1$. This operation is generally optimized into a single matrix vector product. The four column blocks are used as the vectors, while a constant 4x4 matrix is used that combines the modulo operation. The result vector is stored in the next state array at the same location as the original column vector. All elements are 8 bits in width and the multiplication and addition operations are performed over $GF(2^8)$.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (3.2)$$

Since the elements of the matrix are of low degree the multiplications are simplified. A multiplication with 2 in $GF(2^8)$ consists of a shift operation along with a modulo reduction if an overflow occurs. This operation can be reused with multiplying by 3, but an extra addition is required since $3 \cdot a_i = (2 \cdot a_i) \oplus a_i$.

3.1.5 Key Schedule

Round keys are XORed at the end of every round and are generated using a Key Schedule. These keys can be pre-computed or generated at each round. The Sbox components used in the subbytes section are also used here for the round key generation. For each inputted key length, the method of generating keys is slightly different, but they contain similar logic components.

The 128 bit key has a Sbox operation done on the last column of the cipher key state array after the column bytes are rotated. This is followed by a $rcon$ value XOR addition. The $rcon$ value is generated based on the exponentiation formula $rcon(i) = x^{254+1} \; mod \; x^8 + x^4 + x^3 + x + 1$ performed over $GF(2^8)$. These values are usually pre-computed and once the $rcon$ value is added, there is an XOR chain on the columns of the state array that creates the next 128 bit round key. Figure 3.5 shows a single round key computation. This process is repeated by using the round key as a cipher for generating the next 128 bits of key material. The $rcon\ i$ value starts at 1 and increments for each round key.

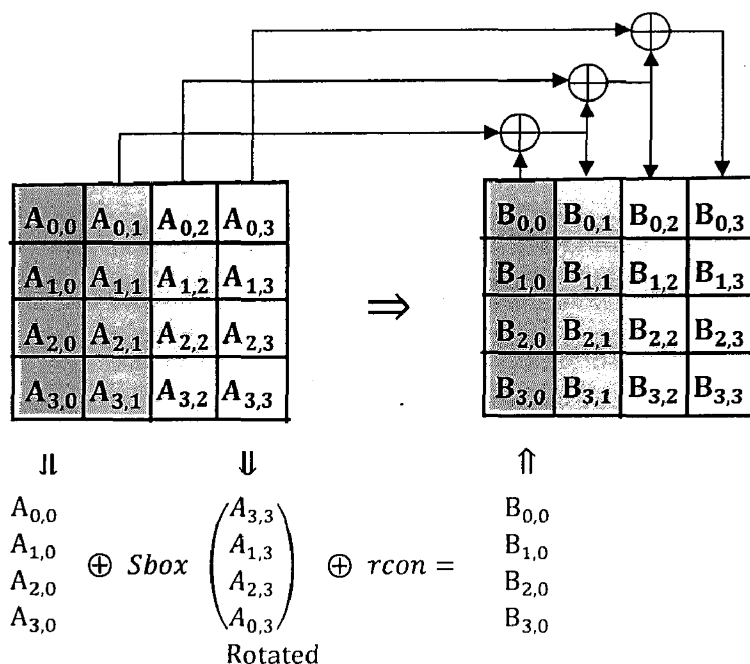


Figure 3.5: AES 128 bit Key Schedule Round.

In order to compute the key schedule operation in hardware, most designs generally pre-compute roundkeys before starting data encryption or decryption. Computing the key schedule on the fly, while rounds are being computed is possible for encryption, and has been implemented for iterative AES [11]. There is added complexity when supporting all keys primarily because of the overlap occurring in operations. In this thesis, 128 bit key employed for various design, so there is no need to explain 192 and 256 bit key schedules.

3.2 Confidentiality Mode of Operation Background

Two modes of operation for Symmetric Key Block Ciphers, ECB and CTR, are selected to create the confidentiality in AES-GCM because they can admit pipelined, parallelized implementations and have minimal computational latency for high data rates. These modes are introduced below and more details can be obtained from [1].

3.2.1 Electronic Codebook Mode (ECB)

The ECB mode is defined as follows and shown in Figure 3.6:

ECB Encryption: $C_j = CIPH_K(P_j) for\ j = 1, \dots, n.$

ECB Decryption: $P_j = CIPH_K^{-1}(C_j) for\ j = 1, \dots, n.$

where, $CIPH_K(P_j)$ is the forward cipher function of the block cipher algorithm, such as AES, under the key K applied to the plaintext P_j; $CIPH^{-1}_K(C_j)$ is the inverse cipher function of the block cipher algorithm under the key K applied to the ciphertext C_j.

In ECB encryption and ECB decryption, multiple forward cipher functions and inverse cipher functions can be computed in parallel or pipeline. In the GCTR module of AES-GCM, ECB encryption block is embedded into a CTR block (see Figure 6.1).

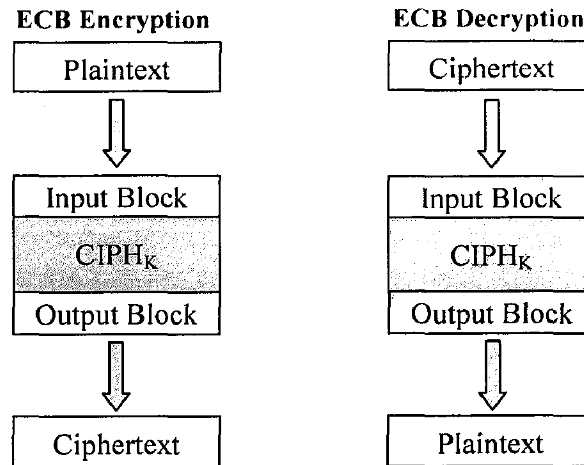


Figure 3.6: ECB Encryption and ECB Decryption. [1]

3.2.2 Counter Mode (CTR)

The CTR mode is a confidentiality mode also that features the application of the block cipher to a set of input data groups, called counters, to produce a set of keystreams that are XORed with the plaintext to produce the ciphertext, and vice versa. The CTR mode is defined as follows and shown in Figure 3.7.

CTR Encryption:

$$O_j = CIPH_K(T_j) for\ j = 1, \dots, n,$$
$$C_j = P_j\ XOR\ O_j\ for\ j = 1, \dots, n-1,$$
$$C_n^* = P_n^*\ XOR\ MSB_u(O_n).$$

CTR Decryption:

$$O_j = CIPH_K(T_j) for\ j = 1, \dots, n,$$
$$P_j = C_j\ XOR\ O_j\ for\ j = 1, \dots, n-1,$$
$$P_n^* = C_n^*\ XOR\ MSB_u(O_n).$$

The symbols used in the CTR encryption and decryption are :

T_j : the counters for the jth input data group,

O_j : the key stream for the jth input data group,

P_j : the jth plaintext group,

C_j : the jth ciphertext group.

C^*_n : the last group of the ciphertext, which may be a partial group.

P^*_n : the last group of the plaintext, which may be a partial group.

$MSB_u(O_n)$: the bit string consisting of the u most significant bits of the bit string O_n.

In CTR encryption and CTR decryption, only the forward cipher function is invoked on each counter group, no inverse cipher function. The resulting key streams are XORed with the corresponding plaintext or ciphertext blocks to produce the ciphertext or plaintext blocks. For the last group, which may be a partial group of u bits, the most significant u bits of the last output group are used for the XOR operation; the remaining bits of the last output group are discarded. The forward cipher functions can be performed in parallel and pipelined.

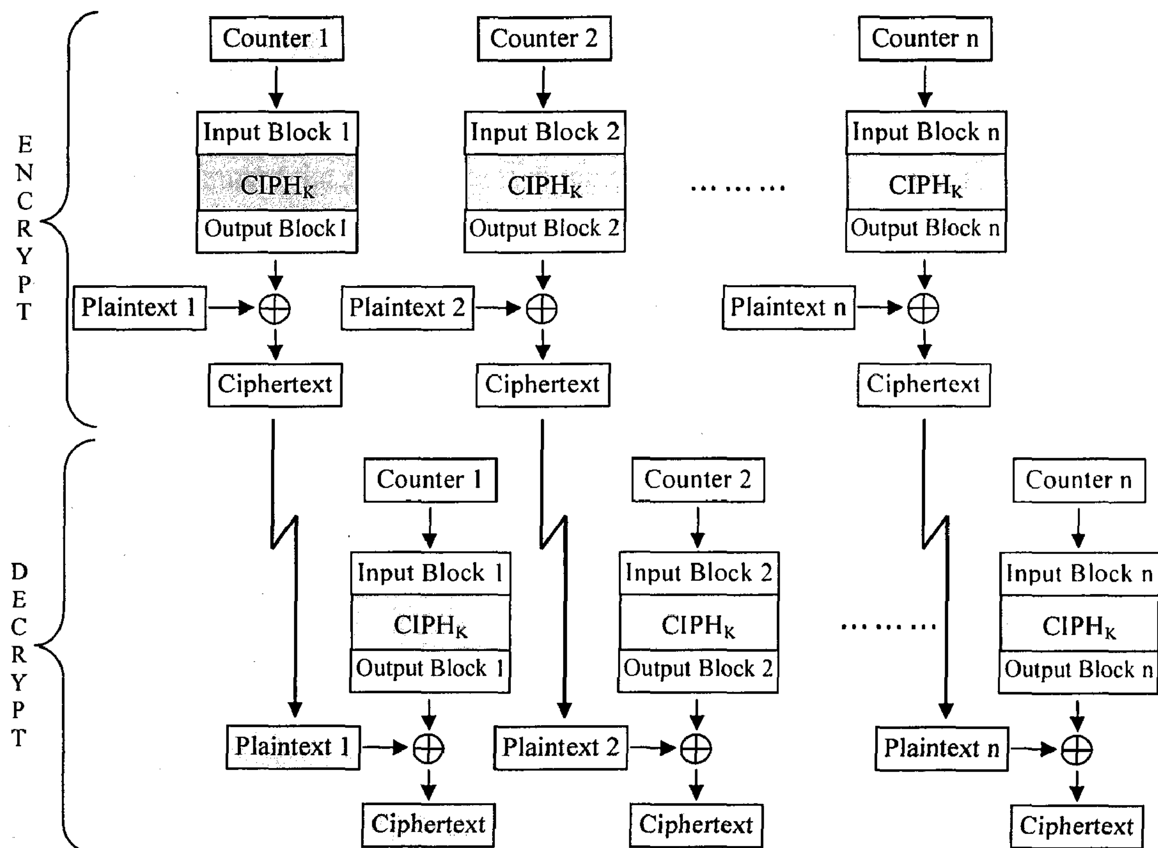


Figure 3.7: CTR Encryption and CTR Decryption [1].

Both CTR encryption and CTR decryption are invoked in AES-GCM encryption and AES-GCM decryption, respectively.

3.3 Galois/Counter Mode (GCM)

The elements of GCM and the associated notation and requirements are introduced in the three sections below. The block cipher and key are discussed in Sec. 3.3.1. The data elements of the authenticated encryption and authenticated decryption functions of GCM are discussed in Sec. 3.3.2. The types of application of GCM supposed in [4] are summarized in Sec. 3.3.3. The GHASH function, GCTR function and GCM specification are described in section 3.3.4, 3.3.5 and 3.3.6, respectively.

3.3.1 Block Cipher

The AES-GCM standard depends on the symmetric key block cipher AES. The AES-GCM key is the block cipher key. The key shall be generated uniformly at random, or close to uniformly at random. The key should be established secretly among the parties to communicate. AES-GCM designates the encryption function of the block cipher AES as the forward cipher function denoted $CIPH_K$ which actually is AES in ECB mode (see Figure 3.1). GCM does not employ the inverse cipher function.

3.3.2 Input and Output Data

GCM consists of the two functions that are called authenticated encryption and authenticated decryption. The requirements and notation for the input and output data of these functions are introduced in Section 3.2.2.1 and 3.2.2.2.

3.3.2.1 Authenticated Encryption

There are three input bit streams to the authenticated encryption operation:

- A plaintext, denoted as P that can have up to2^{39} bits;
- Additional authenticated data (AAD), denoted as A that can have up to 2^{64} bits;

- An initialization vector denoted, as IV that can have up to 2^{64} bits.

In this thesis, a 96-bit IV is adopted for efficiency following the suggestion in [13]. GCM verifies the authenticity of both P and AAD; GCM also protects the confidentiality of P, while the AAD is transmitted in the clear. The IV is a nonce that is associated with the data to be against related attack.

The following two bit strings comprise the output data of the authenticated encryption function:

- A ciphertext, denoted by C, with the same bit length as that of the plaintext.
- An authentication tag, denoted T that have up to 128 bits. The T's bit length is denoted as t.

3.3.2.2 Authenticated Decryption

The inputs to the authenticated decryption function are values for IV, A, C, and T, as described in Sec. 3.2.2.1 above. The output is one of the following:

- The plaintext P that corresponds to the ciphertext C, or
- An indication that the inputs are not authentic, denoted as FAIL.

GCM authenticated decryption computes the authentication tag T' based on received data, and compares it with the received authentication tag T. If the two tags T and T' are equal, then P will be the output of the authenticated decryption function. Otherwise, FAIL will be the output.

3.3.3 Types of Applications of GCM

There are four types of applications of GCM that are recommended in SP800-380D. They are

a. GCM with an arbitrary length IV,
b. GCM with the default IV, i.e. the length of the IV is restricted to exactly 96 bits.
c. GMAC, i.e. the algorithm generates a stand-alone authentication tag T on the AAD with the arbitrary length IV. The plaintext P is the empty string.

d. GMAC with the default IV.

In the thesis, GCM with the default IV is chosen with size shown in Appendix A.

3.3.4 GHASH Function

The authentication mechanism within GCM is based on the hash function, GHASH, that features multiplication by a fixed hash subkey, over a binary Galois field GF(2^{128}). The hash subkey, denoted as H, is generated by applying the block cipher to the 128-bit "0" string. GHASH is a keyed hash function. Algorithm 3 below specifies the function that will be invoked within the AES-GCM authenticated encryption and authenticated decryption functions:

Algorithm 3.1: $\text{GHASH}_H(X)$

Input: **1.** *Bit string* X *with length* $len(X) = 128 \cdot m$ *for some integer* m.

2. *The hash subkey* H.

Output: *Block* Y_m.

Steps:

1. *Let* $X_1, X_2, \ldots, X_{m-1}, X_m$ *represents the unique sequence of blocks such that* $X = X_1 || X_2 || \ldots || X_{m-1} || X_m$.
2. *Let* Y_0 *be the "zero block," which means* Y_0 *is a bit string comprised by 128 binary* 0.
3. *For* $i = 1, \ldots, m$, *let* $Y_i = (Y_i - 1 \oplus X_i) \cdot H$.

   where "

   " indicates multiplication over finite field as discussed in chapter 2.
4. *Return* Y_m.

The GHASH function is illustrated in Figure 3.8 below.

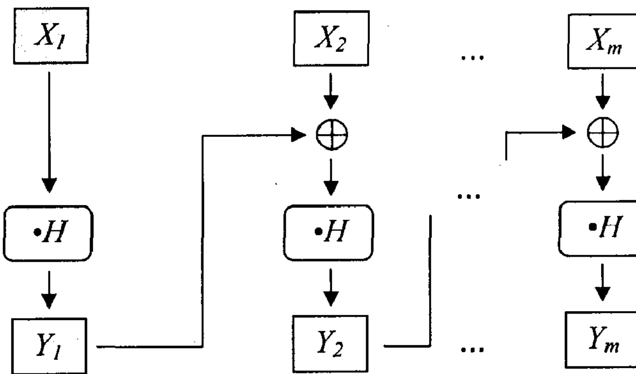


Figure 3.8: $GHASH_H(X_1 \| X_2 \| ... \| X_m) = Y_m$.[4]

3.3.5 GCTR Function

The mechanism for the confidentiality of GCM is a variation of the CTR mode (see section 3.2.2.), called GCTR, with a particular incrementing function, denoted inc, for generating the necessary sequence of counter blocks. The first counter block for the plaintext encryption is generated by incrementing a block that is derived from IV.

Algorithm 3.2 below specifies the GCTR function that will be invoked within the algorithms for the GCM authenticated encryption and authenticated decryption functions:

Algorithm 3.2: $GCTR_K$ (ICB, X)

***Input*:** 1. *Bit string X, of arbitrary length;*

2. *Initial counter block ICB, i.e. IV or some value generated from IV;*

3. *Approved block cipher CIPH (such as AES) with a* 128 − *bit block size;*

4. *Key K;*

***Output*:** *Bit string Y of bit length len(X).*

***Steps*:**

1. $Let\ n = \lceil Len(X)/128 \rceil$

2. $Let\ X_1, X_2, ..., X_{n-1}, X_n^*$ *represents the unique sequence of blocks such that* $X = X_1 \| X_2 \| ... \| X_{n-1} \| X_n^*$.

3. $Let\ CB_1 = ICB.$

4. $For\ i = 2\ to\ n, let\ CB_i = inc(CB_i - 1).$

5. $For\ i = 1\ to\ n - 1, let\ Y_i = X_i \oplus CIPH_K(CB_i).$

6. $Let\ Y_n^* = X_n^* \oplus MSB_{len}(X * n)(CIPH_K(CB_i)).$

7. $Let\ Y = Y_1 || Y_2 || \ldots || Y_{n-1} || Y_n^*.$

8. $Return\ Y.$

Note:

1. Len(X) indicates the bit length of the bit string X.
2. $X_i || X_{i+1}$ indicates the concatenation of two bit strings X_i and X_{i+1}.
3. LSBs (X) indicates the bit string consisting of the s right-most bits of the bit string X.
4. MSBs (X) indicates the bit string consisting of the s left-most bits of the bit string X.
5. Int(X) indicates the integer for which the bit string X is a binary representation.
6. Inc(X) indicates the output of the GCM incrementing function applied to the block X, the more specifically, $inc(X)=MSB_{96}(X) || [(int(LSB_{32}(X))+1) \bmod 2^{32}]_{32}$.

Figure 3.9 below illustrates the GCTR function.

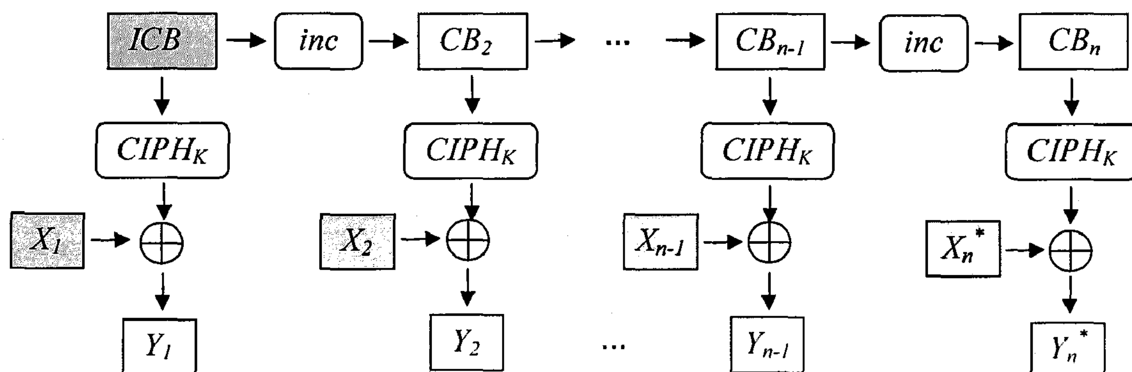


Figure 3.9: $GCTR_K(ICB, X_1 || X_2 || \ldots || X_n^*) = Y_1 || Y_2 || \ldots || Y_n^*$.[4]

3.3.6 GCM Specification

Algorithms for the authenticated encryption and authenticated decryption functions of GCM are specified in Section 3.2.6.1, and 3.2.6.2 below. The block cipher is AES (see section 3.1).

3.3.6.1 Authenticated Encryption

Algorithm 3.3 below performs the authenticated encryption function.

Algorithm 3.3: AES-GCM-AE_K (IV, P, A)

Input: $1. Block\ cipher\ CIPH\ (i.e. AES)\ with\ a\ 128 - bit\ block\ size;$

$2. Key\ K;$

$3. Tag\ length\ t.$

$4. Initialization\ vector\ IV;$

$5. Plaintext\ P;$

$6. Additional\ authenticated\ data\ A.$

Output: $1. Cipher\ text\ C;$

$2. Authentication\ tag\ T.$

Steps:

$1. Let\ H\ =\ CIPH_K(0^{128})$

$2. Define\ a\ block, J_0, as\ follows: J_0 = IV||0^{31}1, i.e. J_0\ is\ a\ 128 - bit\ string$ $consisted\ of\ 96 - bit\ IV, 31\ '0'\ bits, and 1\ '1'\ bit.$

$3. Let\ C\ =\ GCTR_K(inc(J_0), P).$

$4. Let\ u\ =\ 128 \cdot \lceil len(C)/128 \rceil - len(C), and\ let\ v\ = 128 \cdot \lceil len(A)/128 \rceil -$ $len(A)$

$5. Define\ a\ block, S, as\ follows:$

$$S = GHASH_H(A||0^v||C||0^u||[len(A)]_{64}||[len(C)]_{64}).$$

6. $Let\ T = MSB_t(GCTR_K(J_0, S)).$

7. $Return\ (C, T).$

Note:

1. $[x]_s$ indicates the binary representation of the non-negative integer x as a string of s bits, where $x < 2s$.
2. 0^s denotes the string that consists of s '0' bits, e.g. $0^5 = B00000$.

The authenticated encryption function is illustrated in Figure 3.10 below.

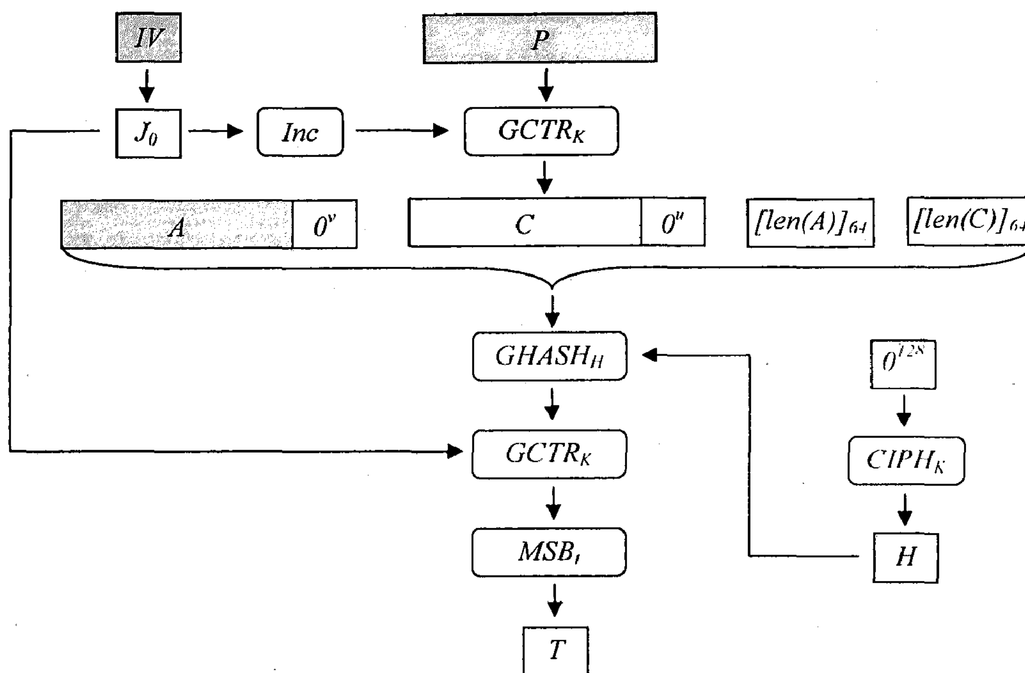


Figure 3.10: AES-GCM-AE_K (IV, P, A) = (C, T).[4]

3.3.6.2 Authenticated Decryption

Algorithm 3.4 below performs the authenticated decryption function.

Algorithm 3.4: AES-GCM-AD$_K$ (IV, C, A, T)

Input: 1. $Block\ cipher\ CIPH\ (i.e.AES)\ with\ a\ 128 - bit\ block\ size;$

2. $Key\ K;$
3. $Tag\ length\ t.$
4. $Initialization\ vector\ IV;$
5. $Cipher\ text\ C;$
6. $Additional\ authenticated\ data\ A.$
7. $Authentication\ tag\ T.$

Output: $Plaintext\ P\ or\ indication\ of\ inauthenticity\ FAIL;$

Steps:

1. $Let\ H = CIPH_K(0^{128})$
2. $Define\ a\ block, J_0, as\ follows: J_0 = IV||0^{31}1. i.e. J_0\ is\ a\ 128 - bit\ string\ consisted\ of\ 96 - bit\ IV, 31\ `0'\ bits, and 1\ `1'\ bit.$
3. $Let\ P = GCTR_K(inc(J_0), C).$
4. $Let\ u = 128 \cdot \lceil len(C)/128 \rceil - len(C), and\ let\ v = 128 \cdot \left\lceil \frac{len(A)}{128} \right\rceil - len(A)$
5. $Define\ a\ block, S, as\ follows:$
   $S = GHASH_H(A||0^v||C||0^u||[len(A)]_{64}||[len(C)]_{64}).$
6. $Let\ T' = MSB_t(GCTR_K(J_0, S)).$
7. $If\ T = T', then\ return\ P;\ else\ return\ FAIL.$

The authenticated decryption function is illustrated in Figure 3.11 below.

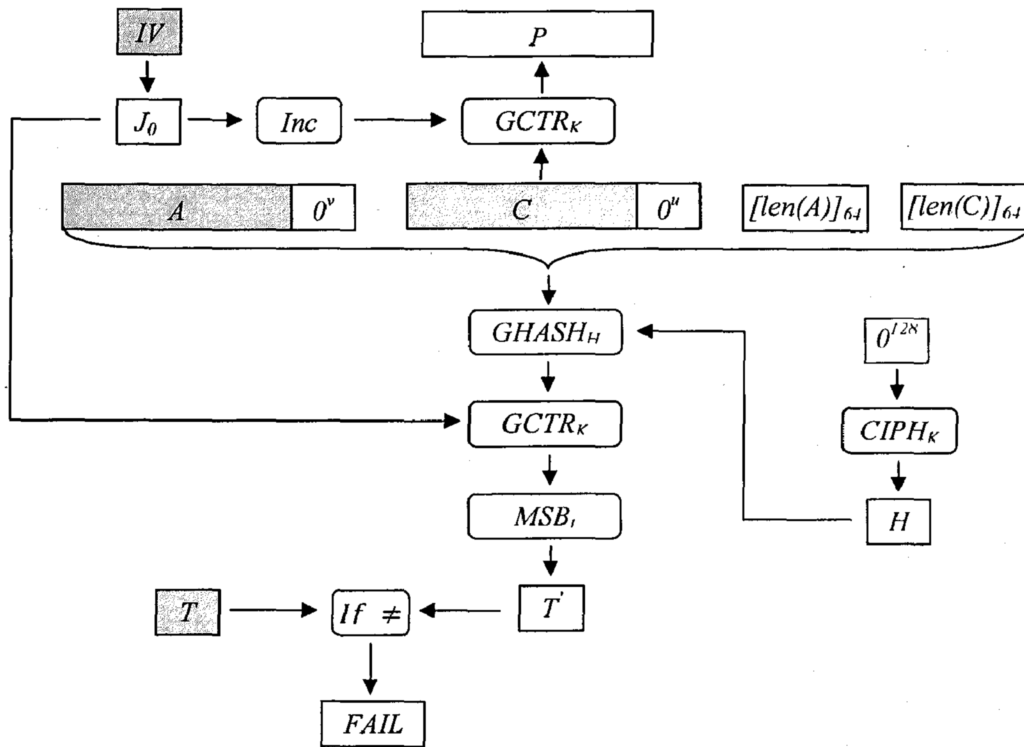


Figure 3.11: AES-GCM-AD$_K$ (IV, C, A, T) = P or FAIL.[4]

CHAPTER 4

PARALLEL MULTIPLIER DESIGNS FOR GCM

Due to the feedback chaining present for the Galois multiplication operation in the GCM, pipelined designs have generally chosen parallel multipliers to complete the multiplication step in a single clock cycle. There are two type of multiplier fulfilling these criteria. First is, Mastrovito multiplier, it has been a prime choice for its low critical path but it unfortunately has a quadratic space complexity. Second is, A popular Sub-quadratic multiplier based on the Karatsuba multiplication algorithm (KA).

A comparison of these parallel multipliers, with FPGA implementation results will be provided toward the end of the chapter. The multipliers are designed specifically for the GCM operation but may be generalized for other applications as well.

4.1 Mastrovito Multiplier

The Mastrovito multiplier uses a matrix vector product (MVP) which can compute modulo reduced result in a single step. The matrix used in the operation is constructed from the field defining polynomial, so this method is applicable when a field polynomial or a set of polynomials is known ahead of time which is the case for GCM. . The MVP approach is first described before going into Mastrovito multiplier for GCM.

4.1.1 Matrix Vector Product

The original GF multiplication operation given in Eq.(4.1) can be modified to formulate the matrix vector product and the rearranged equation is provided below. The polynomial matrix P is computed using the coefficients of $A(\alpha)$ while the vector portion is simply the transposed coefficients of $B(\alpha)$. The matrix vector product shown here computes the multiplication and reduction operations in a single step.

$$C(\alpha) \equiv A(\alpha) \cdot B(\alpha) mod\ F(\alpha)$$

$$C(\alpha) \equiv \sum_{i=0}^{m-1} (\alpha^i \cdot a\ mod\ F(\alpha)) \cdot b_i \tag{4.1}$$

$$C = P \cdot b^T$$

$$P = \{a^{(0)}, a^{(1)}, a^{(2)}, \dots \dots, a^{(m-1)}\}$$

In Eq.(4.1), C is the column vector corresponding to the polynomial $C(\alpha)$. An expansion of the polynomial matrix P is given in Eq.(4.2). The $a^{(i)}$ coefficients are essentially column vectors that are modulo reduced versions of $x^i a\ mod\ F(\alpha)$.

$$\begin{aligned} a &\equiv a\alpha^0\ mod\ F(\alpha) \\ a^{(1)} &\equiv a\alpha^1\ mod\ F(\alpha) \\ a^{(2)} &\equiv a\alpha^2\ mod\ F(\alpha) \equiv a^{(1)}\alpha\ mod\ F(\alpha) \\ a^{(3)} &\equiv a\alpha^3\ mod\ F(\alpha) \equiv a^{(2)}\alpha\ mod\ F(\alpha) \\ &\dots \\ a^{(i)} &\equiv a^{(i-1)}\alpha\ mod\ F(\alpha) \end{aligned} \tag{4.2}$$

The first column of P, $a^{(0)}$ has the coefficients of $A(\alpha)$ while each subsequent column is the previous column multiplied by α and modulo reduced by $F(\alpha)$. When this matrix is multiplied by the coefficients of $B(\alpha)$, the result $C(\alpha)$ is achieved.

4.1.2 Mastrovito Multiplier Design using MVP

The Mastrovito multiplier is a widely used parallel multiplier with a quadratic space complexity [7]. The design is essentially a brute force multiplier in the sense that the MVP is computed like traditional matrix multiplication. It does optimize the operation since the repeated values that are present in the polynomial matrix can be computed once and then reused as signals in hardware for the brute force multiplication portion. Hardware resources are saved to some extent in this way. Elements in P are in GF(2), so

AND and XOR gates are used for element wise multiplication and addition respectively. Since the Mastrovito multiplier uses the brute force approach, after computing elements in P, the Mastrovito design has a single layer of AND gates for element multiplication followed by layers of XOR gates to compute the final result. The simplicity of the Mastrovito design is evident in Figure 4.1 which provides an overview of the multiplier. The design is easy to code into a low level design using any hardware description language such as VHDL.

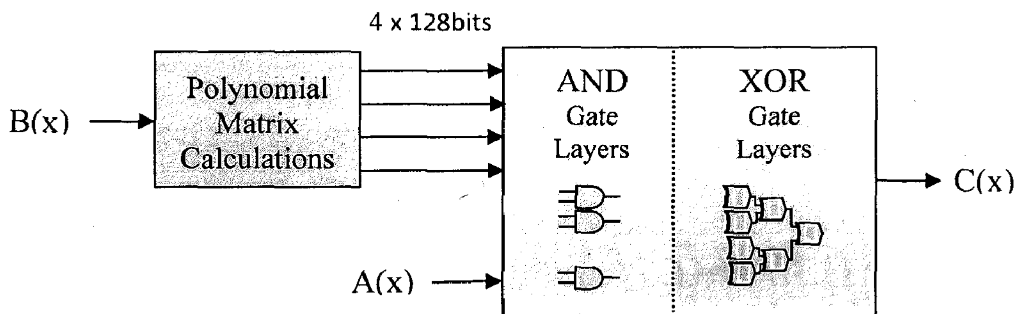


Figure 4.1: Mastrovito Multiplier for GCM.

The area complexity of the Mastrovito multiplier design for the brute force portion is m^2AND gates while the number of XOR gates is $m^2 - m$. The XOR gate count for the polynomial matrix computations will vary based on the field polynomial, and is computed using the Hamming weight of the matrix. For the GCM this is equal to 784 XOR gates.

The time complexity can be summarized as $T_A + (\lceil log_2 m \rceil + \lceil log_2 \theta + 1 \rceil)T_X$, where T_A and T_X is the AND gate and XOR gate delays respectively. The θ constant is the maximum Hamming weight from all the columns of the polynomial matrix.

4.2 Karatsuba Algorithm Sub-quadratic Multiplier

The Karatsuba Algorithm (KA) was originally used to compute digit multiplication [12], and was mapped to polynomials by [14]. It has a Sub-quadratic area complexity but with a larger delay in comparison with the Mastrovito multiplier. Sub-quadratic multipliers such as KA generally decrease the number of multiplication operations while increasing the number of addition computations. Since the cost of adding GF elements is low and

equivalent to XORing bit streams in hardware, the KA is a suitable approach for GF multiplication. Using divide and conquer techniques the multiplication operation is divided up into smaller and smaller operations followed by an expansion to get the final product. This reduction and subsequent expansion is constructed by levels of XOR operations and as a result causes the delay of the multiplier to increase.

4.2.1 KA Multiplier Formulation

The elements $A(\alpha), B(\alpha) \in GF(2^m)$ are first each split into two polynomials of max degree $\frac{m}{2} - 1$. A_h and B_h represents the upper polynomial coefficients while A_l and B_l represents the lower coefficients of the elements. The following equations show $A(\alpha)$ split into two smaller polynomial elements, A_h and A_l.

$$
\begin{aligned}
&A(\alpha) = \alpha^{m/2} A_h + A_l \\
&A_h = (a_{m-1}, a_{m-2}, \dots, a_{\frac{m}{2}+2}, a_{\frac{m}{2}+1}) \\
&A_l = (a_{m/2}, a_{m/2-1}, \dots, a_1, a_0)
\end{aligned} \tag{4.3}
$$

The multiplication of the two elements in $GF(2^m)$ is first computed to get a polynomial of max degree $2m - 2$ ($C'(\alpha)$). The $\oplus$ operation represents XORing bit streams in Eq.(4.4) and multiplication operations shown are with sub-polynomials. The original multiplication is divided into three lower degree polynomial multiplications and this can be further split recursively. The $C'(\alpha)$ element is obtained once the recursion unrolls, and this is then modulo reduced separately to get the final $C(\alpha)$ element.

$$
\begin{aligned}
&D_0, D_1, D_2 \; have\; max\; degree\; \frac{m}{2} - 1 \\
&D_0 = A_l B_l \\
&D_1 = (A_h \oplus B_l)(A_l \oplus B_h) \\
&D_2 = A_h B_h \\
&C'(\alpha) = \alpha^m D_2 \oplus \alpha^{\frac{m}{2}} (D_1 \oplus D_0 \oplus D_2) \oplus D_0 \\
&C(\alpha) = C'(\alpha) \bmod F(\alpha)
\end{aligned} \tag{4.4}
$$

4.2.2 Modulo Reduction

Modulo reduction of $C'(\alpha)$ using the field polynomial can be performed by a multiplication with a fixed reduction matrix. Using the GCM field polynomial as an example, the higher order coefficients of $C'(\alpha)$ can be modulo reduced based on the following equations.

$$
\begin{aligned}
0 &\equiv \alpha^{128} + \alpha^{7} + \alpha^{2} + \alpha + 1 \; mod \; F(\alpha) \\
\alpha^{128} &\equiv \alpha^{7} + \alpha^{2} + \alpha + 1 \; mod \; F(\alpha) \\
\alpha^{129} &\equiv \alpha^{8} + \alpha^{3} + \alpha^{2} + \alpha \;\; mod \; F(\alpha)
\end{aligned}
\tag{4.5}
$$

The reduction matrix has $2m - 2$ columns and m rows. The matrix essentially maps $C'(\alpha)$ to $C(\alpha)$ and is shown in Figure 4.2 for the GCM. The first m columns of the matrix form an identity matrix since elements of degree $1 \; to \; m - 1$ do not need to be reduced. Using Eq.(4.5), all elements of degree m to $2m - 2$ can be modulo reduced and then used in creating the remaining $m - 2$ columns of the reduction matrix.

$$
\underset{C(\alpha)}{\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ \vdots \\ C_{126} \\ C_{127} \end{bmatrix}}
=
\underset{Reduction\;matrix}{\begin{bmatrix}
1 & 0 & 0 & \cdots & | & 1 & 0 & & \\
0 & 1 & 0 & \cdots & | & 1 & 1 & & \\
0 & 0 & 1 & \cdots & | & 1 & 1 & \cdots & \\
\vdots & \vdots & \vdots & \ddots & | & 0 & 1 & & \\
 & & & & | & 0 & 0 & \cdots & \\
 & & & & | & 0 & 0 & & \\
 & & & & | & 1 & 0 & & \\
 & Identity & & & | & 0 & 1 & & \\
 & & & & | & \vdots & \vdots & \ddots & \\
\downarrow & \downarrow & \downarrow & & & \downarrow & \downarrow & & \downarrow \\
1 & \alpha^{1} & \alpha^{2} & \cdots & & \alpha^{128} & \alpha^{129} & \cdots & \alpha^{254}
\end{bmatrix}}
\cdot
\underset{C'(\alpha)}{\begin{bmatrix} C'_0 \\ C'_1 \\ C'_2 \\ C'_3 \\ C'_4 \\ \vdots \\ \vdots \\ C'_{253} \\ C'_{254} \end{bmatrix}}
$$

Figure 4.2: Reduction matrix for GCM.

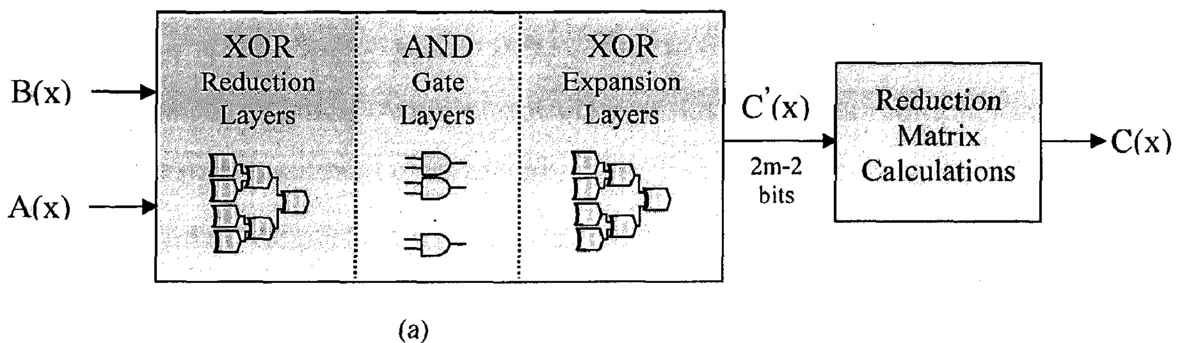
The cost of this operation in relation to the KA multiplication is small and is dependent on the field polynomial. The Hamming weight of the reduction matrix for the GCM shows that this operation requires 527 XOR gates. Having low order terms within

the field polynomial helps reduce the cost of the operation since higher order terms have additional feedback terms which increase the cost of the operation. For a field polynomial such as $\alpha^{128} + \alpha^{40} + \alpha^2 + \alpha + 1$ the cost of the operation is 623 XOR gates. The delay for the reduction operation can be computed by $(\lceil log_2\theta + 1 \rceil)T_X$, where θ is the largest Hamming weight computed by row of the reduction matrix. For the GCM reduction matrix this delay is computed to be $3T_X$.

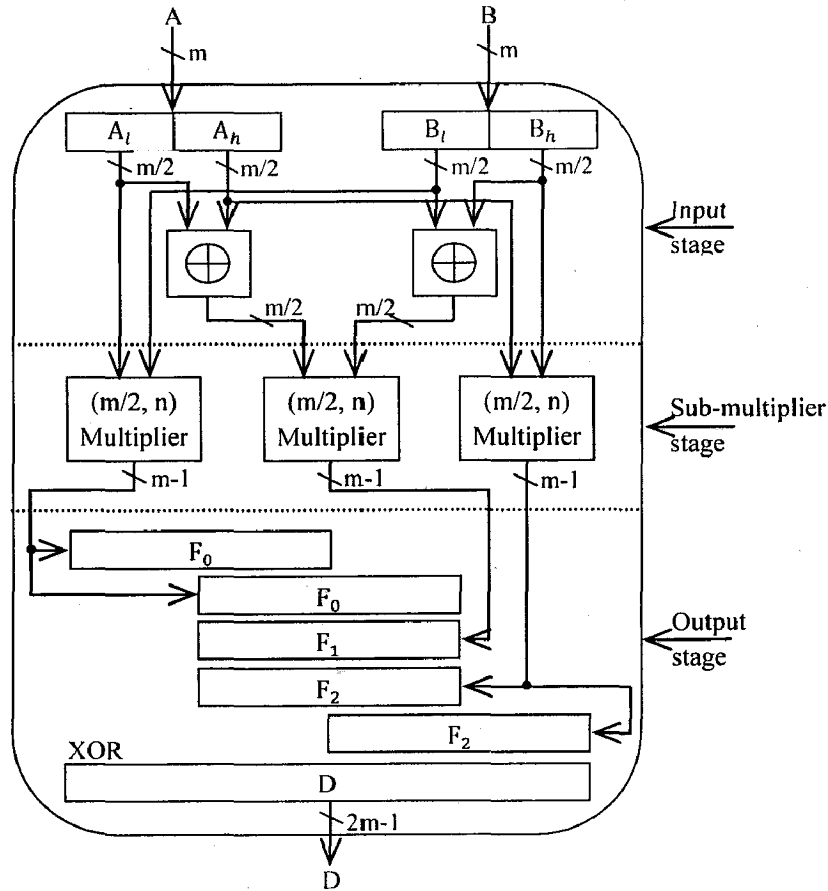
4.2.3 KA Multiplier Design for GCM

The Karatsuba algorithm generally works best with elements of even degree since each step in the recursion splits polynomials equally. The input element size for the GCM Galois operation is 128 bits, a power of 2, so the KA multiplier can be easily applied without any changes required. A high level view of the Karatsuba multiplier is provided here with all the major components required.

The polynomial elements can be conveniently split down to single element multiplications but this is not always desirable in terms of area efficiency. When the ending condition of the recursion is changed and brute force multiplication per- formed instead, this leads to some savings in terms of AND and XOR gates. The following table shows the number of gates required for halting at different polynomial sizes. The gate counts do not include the reduction operation which has a fixed number of gates and a fixed delay of $3T_X$. The ending condition delays are based on the brute force multiplication delay which is $T_A + log_2(n)T_X$, where n is the halting value.



(a)



(b)

Figure 4.3: (a) Abstract view, and (b) Full view of the Karatsuba Multiplier.

Table 4.1: Area of KA Multiplier with varied ending conditions.

Halt	XOR gates	AND gates	Total gates	NAND gates	Delay
2	9913	2916	12829	45484	$T_A + 19T_x$
4	8455	3888	12343	41596	$T_A + 17T_x$
8	7969	5184	13153	42244	$T_A + 15T_x$
16	8455	6912	15367	47644	$T_A + 13T_x$
32	9913	9216	19129	58084	$T_A + 11T_x$
64	12415	12288	24703	74236	$T_A + 9T_x$

We can see from Table 4.1 that it is worthwhile halting the KA when the polynomial size is 4 since it provides the lowest area and delay complexity. Since the cost of XOR gates in hardware is usually larger than that of AND gates, in order to get more accurate area estimates for ASIC implementations, the NAND gate count is included. The area

cost of 1 XOR gate is bounded by the area of 4 NAND gates while one AND gate is bounded by the area of 2 NAND gates. When taking the NAND gate count into consideration the results still showed halting at 4 as the optimal choice in terms of area.

4.3 FPGA Implementation Results

Table 4.2 showing FPGA implementation results and Figure 4.4 showing performance comparison of above discussed multipliers. On analyzing the result we find that, Karatsuba multiplier used 58% (Approx.) less area as compared to Mastrovito because of sub-quadratic complexity nature of former, but cost for small area of Karatsuba have to paid in term of speed, its throughput is 32% (Approx.) less than that of Mastrovito.

The preferences of multiplier mostly depend on type of application and desired critical parameters, otherwise in overall performance Karatsuba proof better than Mastrovito.

Table 4.2: Multiplier's Place and Route Results Summary.

Multiplier	Delay (ns)	Frequency (MHz)	Throughput (Gbps)	Slices	gates	Kbps/Slice	Power (mW)
Mastrovito	10.260	97.465	12.476	8,229	85,161	1516.10	990
Karatsuba	14.705	68.004	8.706	3,486	40,497	2497.42	1438

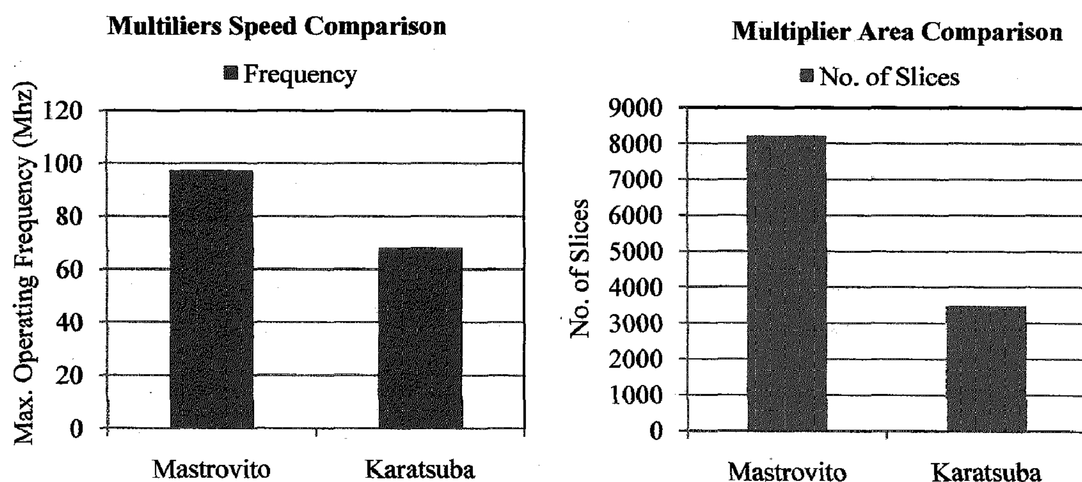


Figure 4.4: Multiplier performance comparison.

CHAPTER 5

FPGA IMPLEMENTATION OF AES-ECB ARCHITECTURES

This chapter includes details of proposed architectures of AES in ECB mode and discussion on their implementation result followed by comparison with previous claim. Section 5.1 cover iterative compact single round AES design, which is optimize for small area and section 5.2 describe pipeline based high speed architecture of AES.

5.1 Compact Single Round AES Design

This section presents high-performance and compact architecture for single round Advance Encryption Standard (AES) security algorithm using feedback mode. There are two design based on stated architecture has been implemented on virtex-4 Field Programming Gate Array (FPGA) device. These two designs differ in method used for sub-bytes function implementation, in first design Look-Up Tables (LUTs) and in second design fully combinational gates using Composite Field Arithmetic (CFA) has been employed for sub-byte function implementation.

5.1.1 Single Round AES Architecture

The working of proposed architecture is straight forward, as we can see in Figure 5.1. There are three main units i.e. Round unit, Key Scheduler unit and control unit. We will discuss one by one later part of this section.

The multiplexer named as MUX direct particular input by using 2bit selective line data_reg_mux_sel to the input of Data Register at proper clock cycle and that particular 128 bits data store in 128 bits register, for being used by Round unit as a input. Since round 0 is just a XORing between 128 bits data and 128 bits original user-key, performed externally using 128 bits 2 input XOR as shown in Figure 5.1.

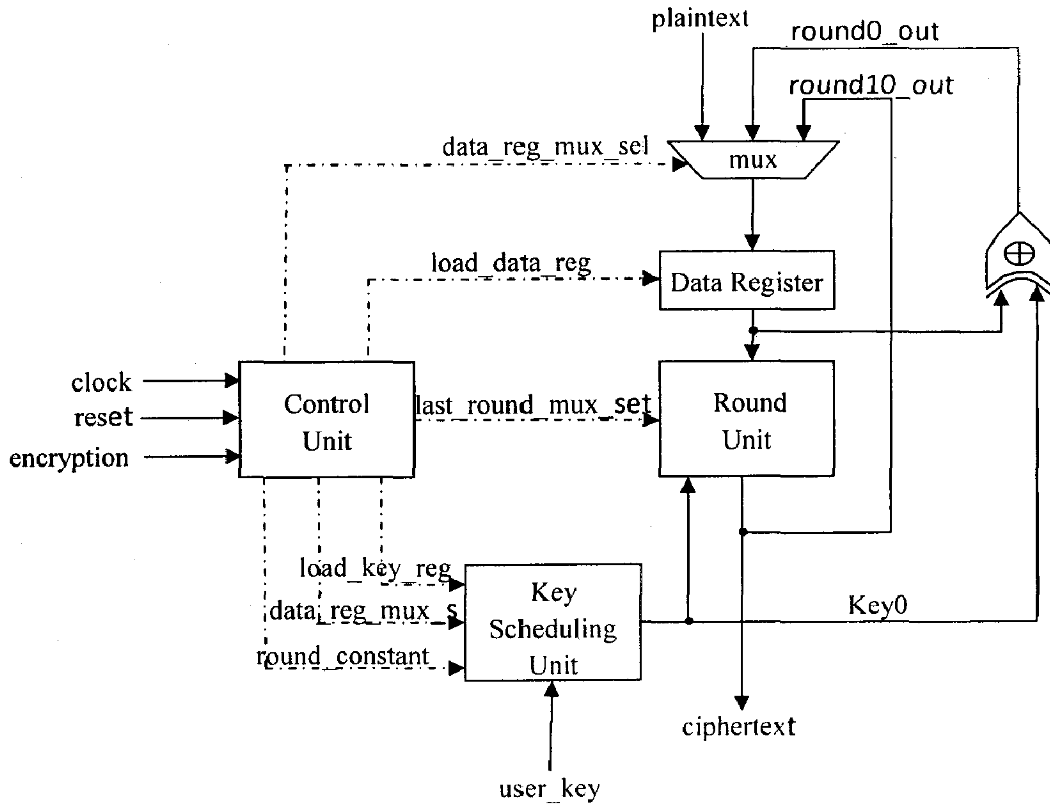


Figure 5.1: Compact single round AES FPGA architecture.

5.1.1.1 Round Unit

As we have discuss earlier that round 1 to round 9 are the combination of four functions i.e. sub SubBytes, ShiftRows, MixColumns and AddRounKey, round 10 combination of

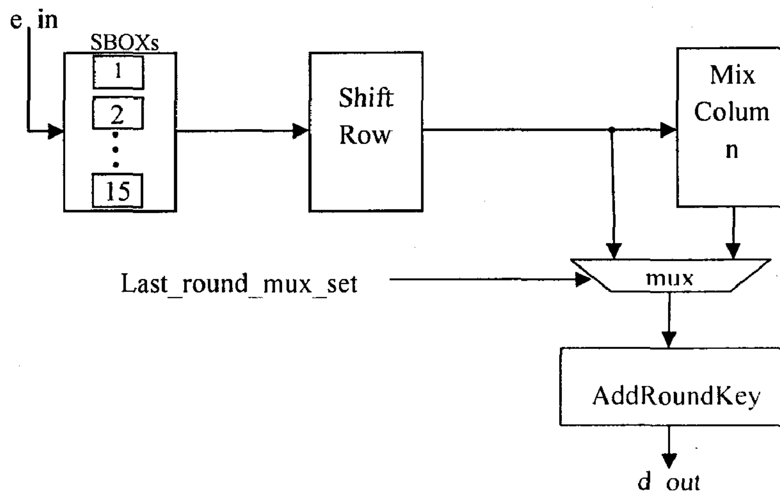


Figure 5.2: Round unit.

three function except MixColumns. Figure 5.2 show round unit that also take care of round 10 using MUX that controlled by Last_round_mux_set 1bit selection line. The concept of parallel processing has been used in this architecture for calculating the sub bytes. Instead of calculating the sub bytes sequentially, which consumes a lot of time, the 16 sub bytes are generated simultaneously using 16 S Boxes [15] and that can be realized by16 look-up tables with 8bit-input/output or by CFA. ShiftRows is a bit shuffling function, requiring no hardware. For MixColumns and AddRoundKey realization general procedure adopted as describe in [5].

5.1.1.2 Key Scheduler Unit

Figure 5.3 is a Key Scheduler (Expansion) Unit [16] used to generate round keys on the fly in the encryption process. The hardware required to generate one set of round key is implemented and re-used it for calculating the rest of the round keys. This results in reduction in space used for storing the sub keys values and also improves the speed of operation since round key is generated simultaneously while the sub bytes, shift rows and mix columns take place.

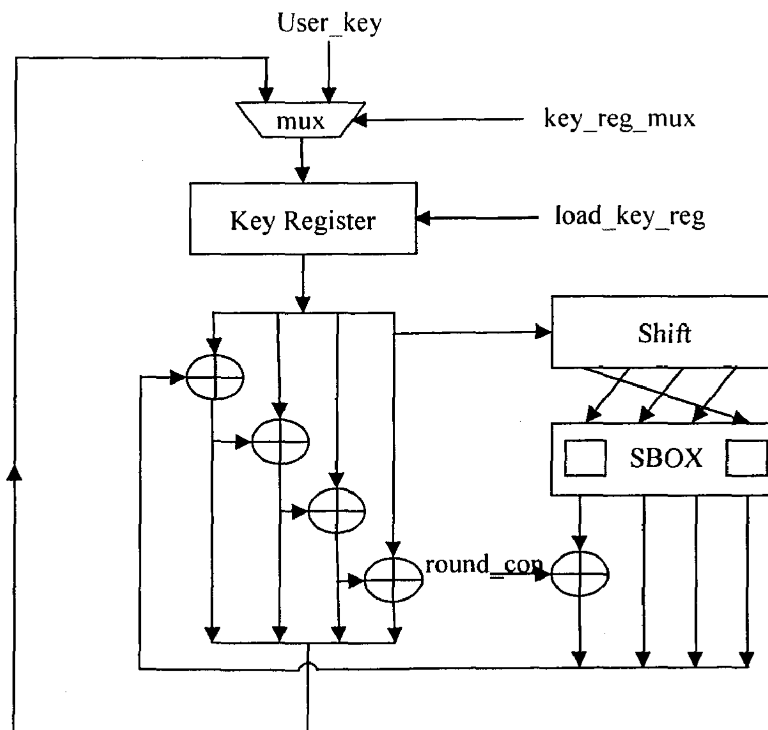


Figure 5.3: Key Scheduler Unit.

5.1.1.3 Control Unit

Control unit is brain of the system, which used to control and maintains proper synchronization between different components of design. As shown in Figure 5.4, Control unit realized using Finite State Machine (FSM) having thirteen states, one for initiation, one for load input and remaining eleven for eleven round of AES. Each state defined by their specific value of state variable (control signal).

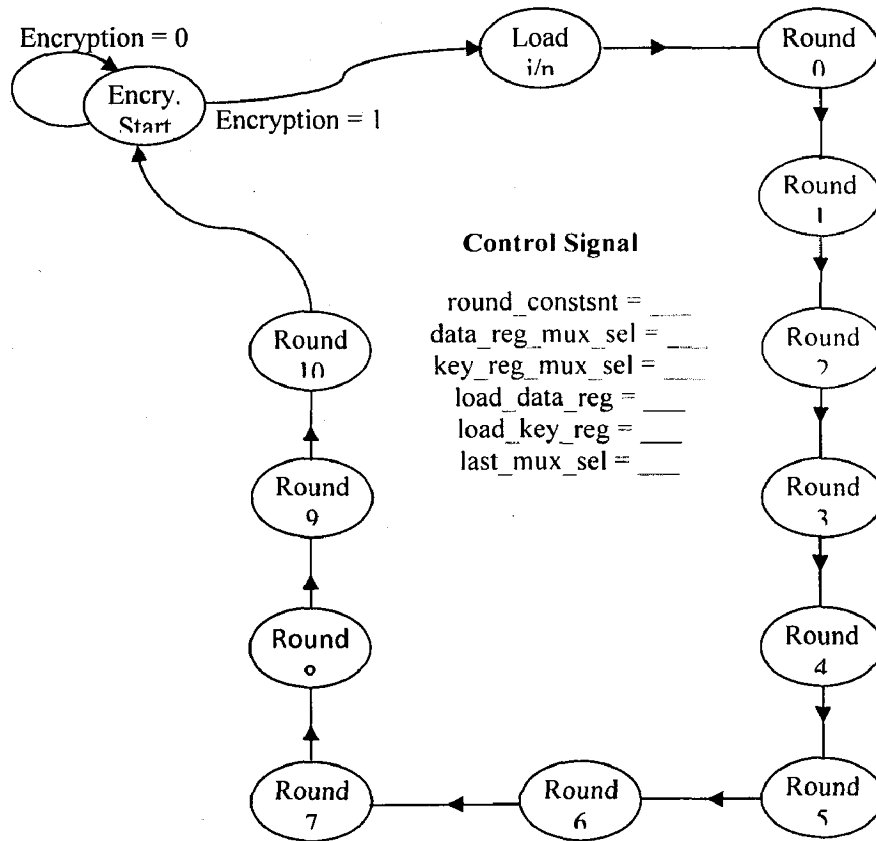


Figure 5.4: State diagram.

5.1.2 Composite Field Arithmetic based SubBytes function

The composite field approach reduces not only the hardware complexity but also exhibits the advantages of inner round pipelining, but here we are not interested in pipelining. The Galois field F1: $GF(2^8)$ is mapped into composite field F2: $GF((2^4)^2)$ or sometimes $GF(((2^2)^2)^2)$ [17].

In this paper, we construct the isomorphic composite field by using the fields defined in Eq.(5.1) [16]. The field conversion matrix δ is given in Eq.(5.2).

$$\begin{cases} GF(2) \Rightarrow GF(2^4) : p(x) = x^4 + x + 1 \\ GF(2^4) \Rightarrow GF((2^4)^2) : q(y) = y^4 + y + \beta, \\ where\ \beta = \{0b1000\}_2\ or\ x^3 \end{cases} \quad (5.1)$$

$$\delta = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (5.2)$$

Figure 3.1.2 describes the optimized design of SubBytes transformation over composite field.

5.1.3 Implementation Results

Based on presented architecture, two designs have been implemented on FPGA virtex-4 XC4VLX100-12ff1148 package kit, using Xilinx Foundation Series f 9.2i as synthesis and Modelsim 6.3f as simulation tool. The design was coded using VHDL language.

On comparison, we find that LUT based design is 19.18% faster than CFA based design and on the other hand CFA based design proved more cost effective, since it required 35.55% smaller area than LUT based design. But as whole CFA based design is More efficient than LUT based because it gives higher throughput to slices ratio than LUT based design, as shown in Table 5.1.

5.1.4 Performance Comparison with other Designs

There are very few number of designs proposed for small AES design in literature. After intensive search we encountered few single round AES designs, as shown in Table 5.3

Table 5.1. Synthesis and Place & Route results of compact AES designs.

Design	LUT (Syn.) #	Slices (Syn.) #	Period (Syn.) ns	Slices (PAR) #	Blocks RAMs (PAR) #	Usage (PAR) %	Period (PAR) ns	Freq. (PAR) Mhz	Thr. (PAR) Mbits/s	Thr./sl (PAR) Mbit/s /slice
LUT Based	4296	2222	5.294	2571	20	05%	10.161	98.416	1049.7	0.408
CFA Based	3230	1668	7.285	1657	00	03%	12.110	82.576	880.81	0.532

Table 5.2. The logic and routing delay of compact AES designs.

Design	Period ns	Logic ns	Percent %	Routing ns	Percent %
LUT Based	10.161	7.966	78.4%	2.194	21.6%
CFA Based	12.110	3.826	31.6%	8.283	68.4%

with their performance. Wide trade-off is possible in area and speed of design, so throughput to slices ratio has been taken as comparison parameter for various designs.

As we can see Table 5.3, throughput to slices ratio design [18] is 0.747, highest among all design but it takes four clock cycles to complete one round and also required three block RAMs and above all very small throughput of 166 Mbps. On the other hand, our composite field arithmetic based design is pure memory less design having 0.532 throughputs to slices ratio, which is second highest among all and first in all design of one clock cycles per round, in addition of that it shows a high throughput up to 880.81 Mbps.

Table 5.3. Performance comparison of compact AES designs.

Design		Area		Throughput [Mbps]	Thr./Slices [Mbits/s/slices]	clock cycles per round
		CLB Slices	Block RAMs			
Pawe_l Chodowiec and Kris Gaj [18]		222	03	166	0.7470	4
P. Chodowiec et al. [19]		~1230	18	577	0.4690	1
A. Dandalis et al. [20]		5673	00	353	0.0620	1
A.J. Elbirt et al. [21]		3528	00	294.2	0.0834	1
K. Gaj et al. [22]		2902	00	331.5	0.1142	1
Proposed Designs	LUT Based	2571	20	1049.7	0.4080	1
	CFA Based	1657	00	880.81	0.5320	1

5.2 High Speed Subpipelined AES Design

This section presents high-speed architectures for the hardware implementation of the Advanced Encryption Standard (AES) algorithm by dividing each round unit into substages with equal delays, named as subpiplining. Composite field Arithmetic is used to implement the SubBytes and InvSubBytes transformations of the AES algorithm, which makes it a fully memory less combinational logic design. Also as a direct consequence, the unbreakable delay introduces by look-up tables in the conventional approaches is eliminated and the advantage of subpipelining can be further explored.

5.2.1 The AES Algorithm And Its Subpipelined Architecture

5.2.1.1 The Subpipeline Architecture

The pipelined architecture is realized by inserting rows of registers between each round unit. Similar to the pipelining, subpipelining also inserts rows of registers among combinational logic, but registers are inserted both between and inside each round unit, as shown in Fig. 5.5. In subpipelining, more blocks of data can processed simultaneously. It can be observed that the more substages with equal delay each round unit can be divided into, the larger speedup the subpipelining can achieve.

However, dividing each round unit into arbitrary number of substages does not always bring speedup. Since the minimum clock period is determined by the indivisible component with the longest delay, dividing the rest of the round unit into more substages with shorter delay does not reduce the minimum clock period. Although more blocks of data are being processed simultaneously, the average number of clock cycles to process one block of data does not change. Therefore, the overall speed does not improve despite increased area caused by the additional registers. In a LUT-based implementation, it can be observed that nearly half the delay of a round unit is attributed to the LUTs, and thus, each round unit can be divided into only two substages to achieve some speedup without wasting any area. On the contrary, the longest unbreakable delay in the non-LUT-based approaches is the delay of individual logic gates. Accordingly, each round unit can be divided into multiple substages with approximately equal delay.

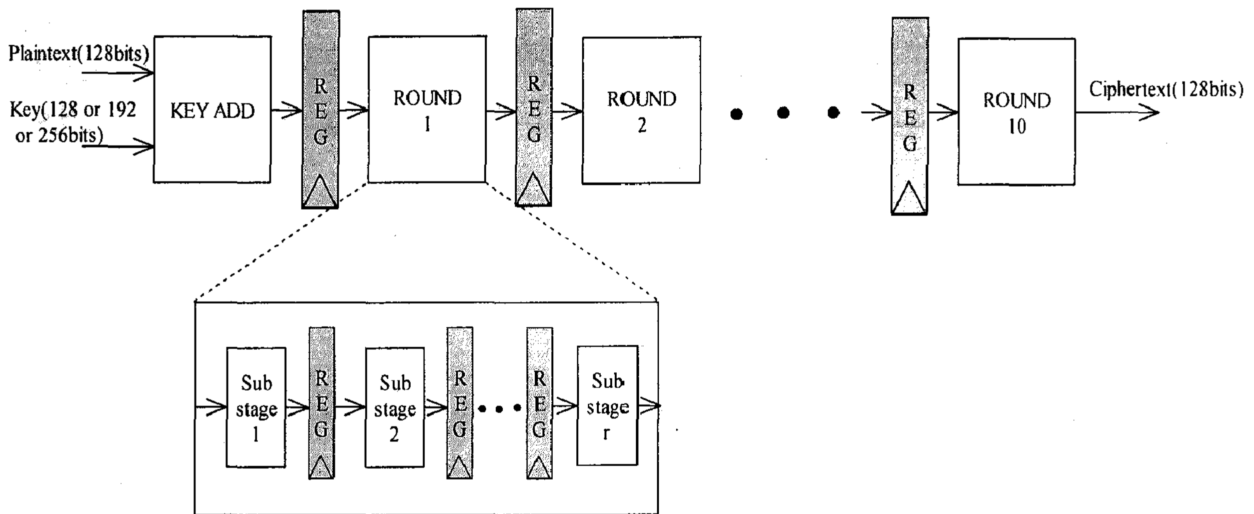


Figure 5.5: The architecture of Subpipelining.

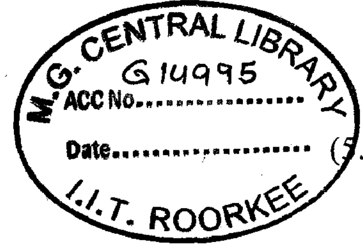
5.2.2 Detailed Hardware Implementation Architectures

In this section, we present detailed architectures for each transformation of the AES algorithm. The implementation of each transformation is optimized to reduce area and increase speed. Meanwhile, an efficient key expansion architecture suitable for subpipelined round units is presented. Based on the analysis on the gate counts in the critical path of the round units and the key expansion, optimized subpipelining architectures of the AES algorithm are present.

5.2.2.1 Implementations of the SubBytes/InvSubBytes Transformation

The multiplicative inversion in involved in the Sub- Bytes/InvSubBytes is a hardware demanding operation, it takes at least 620 gates to implement by repeat multiplications in $GF(2^8)$ [23]. However, the gate count can be reduced greatly by using composite field arithmetic. In the SubBytes transformation, using substructure sharing, the isomorphic mapping function can be implemented by 12 XOR gates with 4 XOR gates in the critical path. Meanwhile, the combined inverse isomorphic mapping and the affine transformation can be implemented by 19 XOR gates, and the critical path consists of 4 XOR gates also. In the composite field $GF((2^4)^2)$, an element can be expressed as

$s_h x + s_l$, where $s_h, s_l \in GF(2^4)$ and x is a root of $P_2(x)$. Using Extended Euclidean algorithm, the multiplicative inverse of $s_h x + s_l$ modulo $P_2(x)$ can be computed as in Eq.(5.3)


M.G. CENTRAL LIBRARY
G14995
ACC No.
Date
I.I.T. ROORKEE

$$(s_h x + s_l)^{-1} = s_h \Theta x + (s_h + s_l)\Theta \quad (5.3)$$

where $\Theta = (s_h^2 \lambda + s_h s_l + s_l^2)^{-1}$. According to Eq.(5.3), the multiplicative inversion in $GF(2^8)$ can be carried out in $GF((2^4)^2)$ by the architecture illustrated in Figure 5.6 (chapter4). The multipliers in $GF(2^4)$ can be further decomposed into multipliers in $GF(2^4)$ and then to $GF(2)$, in which a multiplication is simply an AND operation. Figure 5.8 illustrates this decomposition, together with the other blocks used in Figure 5.6 except the inversion in $GF(2^4)$ block. As can be observed from Figure 5.7, a multiplier in $GF(2^4)$ can be implemented by 21 XOR gates and 9 AND gates.

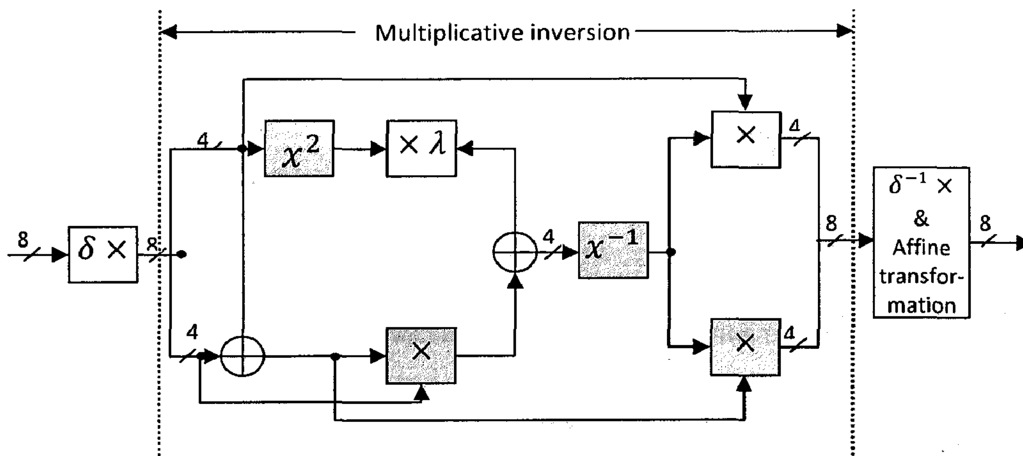


Figure 5.6: Implementation of the subBytes Transformation.

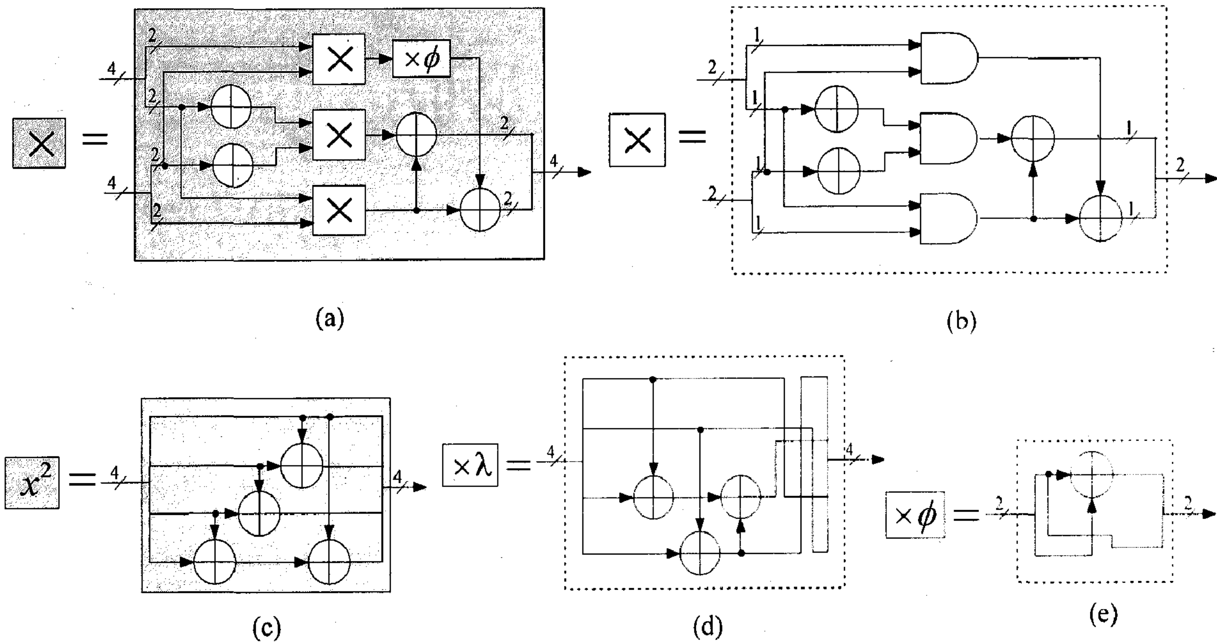


Figure 5.7: Implementations of individual blocks: (a) multiplier in $GF(2^4)$; (b) multiplier in $GF(2^2)$; (c) squarer in $GF(2^4)$; (d) constant multiplier ($\times \lambda$); and (e) constant multiplier ($\times \phi$).

with 4 XOR gates and 1 AND gate in the critical path. Table 5.4 summarizes the gate count and critical path of each block in the SubBytes except the block of inversion in $GF(2^4)$ in Figure 5.6.

Table 5.4: Gate counts and critical paths functional blocks in the SubBytes Transformation [17].

Block	Total no. of gates	Critical path
$\times \phi$	1 XOR	1 XOR
$\times \lambda$	3 XOR	2 XOR
x^2	4 XOR	2 XOR
Multiplier in $GF(2^2)$	4 XOR+3 AND	2 XOR+1 AND
Multiplier in $GF(2^4)$	21 XOR+9 AND	4 XOR+1 AND

The inversion in $GF(2^4)$ can be implemented by further decomposed by applying formulas similar to Eq.(5.3) iteratively. Composite field decomposition can reduce the hardware complexity significantly when the order of the field involved is large.

However, for small fields, such as $GF(2^4)$, further decomposition may not be the optimum approach. So we adopt direct implementation approach. Taking the four bits of $x \in GF(2^4)$ as $\{x_3, x_2, x_1, x_0\}$, it can be derived that each bit in $x^{-1} = \{x_3^{-1}, x_2^{-1}, x_1^{-1}, x_0^{-1}\}$ can be computed by the following equations:

$$\begin{cases} x_3^{-1} = x_3 + x_3x_2x_1 + x_3x_0 + x_2 \\ x_2^{-1} = x_3x_2x_1 + x_3x_2x_0 + x_3x_0 + x_2 + x_2x_1 \\ x_1^{-1} = x_3 + x_3x_2x_1 + x_3x_2x_0 + x_2 + x_2x_0 + x_1 \\ x_0^{-1} = x_3x_2x_1 + x_3x_2x_0 + x_3x_1 + x_3x_1x_0 + x_3x_0 \\ \qquad +x_2 + x_2x_1 + x_2x_1x_0 + x_1 + x_0 \end{cases} \tag{5.4}$$

direct implementation of the derived equation, Eq.(5.4), has the smallest gate count (14 XOR and 9 AND) and the shortest critical path (3 XOR and 2 AND) [25].

5.2.2.2 Implementations of the MixColumns/InvMixColumns Transformation

Various architectures have been proposed for the implementation of the MixColumns/InvMixColumns transformation [9], [24], [25]. Applying substructure sharing both to the computation of a byte and between the computation of the four bytes in a column of the State, an efficient MixColumns implementation architecture can be derived.

$$\begin{cases} S'_{0,c} = \{02\}_{16}(S_{0,c} + S_{1,c}) + (S_{2,c} + S_{3,c}) + S_{1,c} \\ S'_{1,c} = \{02\}_{16}(S_{1,c} + S_{2,c}) + (S_{3,c} + S_{0,c}) + S_{2,c} \\ S'_{2,c} = \{02\}_{16}(S_{2,c} + S_{3,c}) + (S_{0,c} + S_{1,c}) + S_{3,c} \\ S'_{3,c} = \{02\}_{16}(S_{3,c} + S_{0,c}) + (S_{1,c} + S_{2,c}) + S_{0,c} \end{cases} \tag{5.5}$$

According to Eq.(5.5), the MixColumns transformation can be implemented by the architecture shown in Figure 5.8. The function of the block "XTime" is to compute constant multiplication by $\{02\}_{16}$. An element of $GF(2^8)$ can be expressed in polynomial form as $S = s_7x^7 + s_6x^6 + s_5x^5 + s_4x^4 + s_3x^3 + s_2x^2 + s_1x + s_0$, where $s_1, s_2, \ldots\ldots s_7 \in GF(2)$, and x is a root of the field poly nomial $p(x)$. then

$$\{02\}_{16}S = xS = s_7x^8 + s_6x^7 + s_5x^6 + s_4x^5 + s_3x^4 + s_2x^3 + s_1x^2 + s_0x \ mod \ p(x)$$
$$= s_6x^7 + s_5x^6 + s_4x^5 + (s_3 + s_7)x^4 + (s_2 + s_7)x^3 + s_1x^2 + (s_0 + s_7)x + s_7.$$

Therefore, the "XTime" block can be implemented by 3 XOR gates with only one XOR gate in the critical path. As illustrated in Figure 5.8, the total number of XOR gates for computing one column of the State is 108, and the critical path is 3 XOR gates. Similarly, in the InvMixColumns transformation, Eq.(5.2) can be rewritten as

$$\begin{cases} S'_{0,c} = (\{02\}_{16}(S_{0,c} + S_{1,c}) + (S_{2,c} + S_{3,c}) + S_{1,c}) \\ +(\{02\}_{16}\left(\{04\}_{16}(S_{0,c} + S_{2,c}) + \{04\}_{16}(S_{1,c} + S_{3,c})\right) \\ +\{04\}_{16}(S_{0,c} + S_{2,c})) \\ S'_{0,c} = (\{02\}_{16}(S_{0,c} + S_{1,c}) + (S_{2,c} + S_{3,c}) + S_{1,c}) \\ +(\{02\}_{16}\left(\{04\}_{16}(S_{0,c} + S_{2,c}) + \{04\}_{16}(S_{1,c} + S_{3,c})\right) \\ +\{04\}_{16}(S_{1,c} + S_{3,c})) \\ S'_{2,c} = (\{02\}_{16}(S_{2,c} + S_{3,c}) + (S_{0,c} + S_{1,c}) + S_{3,c}) \\ +(\{02\}_{16}\left(\{04\}_{16}(S_{0,c} + S_{2,c}) + \{04\}_{16}(S_{1,c} + S_{3,c})\right) \\ +\{04\}_{16}(S_{0,c} + S_{2,c})) \\ S'_{3,c} = (\{02\}_{16}(S_{3,c} + S_{0,c}) + (S_{1,c} + S_{2,c}) + S_{0,c}) \\ +(\{02\}_{16}\left(\{04\}_{16}(S_{0,c} + S_{2,c}) + \{04\}_{16}(S_{1,c} + S_{3,c})\right) \\ +\{04\}_{16}(S_{1,c} + S_{3,c})) \end{cases} \quad (5.6)$$

Using substructure sharing, Eq.(5.6) can be implemented by the architecture illustrated in Figure 5.9. The "X4Time" block, which computes the constant multiplication of $\{04\}_{16}$, can be implemented by two serially concatenated "XTime" block. Alternatively, it can also be implemented according to the equation derived below

$$\{04\}_{16}S = x^2S = s_7x^9 + s_6x^8 + s_5x^7 + s_4x^6 + s_3x^5 + s_2x^4 + s_1x^3 + s_0x^2 \ mod \ p(x)$$
$$= s_5x^7 + s_4x^6 + (s_3 + s_7)x^5 + (s_2 + (s_6 + s_7))x^4 + (s_1 + s_6)x^3$$
$$+(s_0 + s_7)x^2 + (s_6 + s_7)x + s_6.$$

Sharing $s_6 + s_7$, the "X4Time" block can be implemented by 5 XOR gates with 2 XOR gates in the critical path. It follows that the architecture in Figure 5.9 can be implemented by 193 XOR gates with 7 XOR gates in the critical path. Meanwhile, the upper half in Figure 5.9 work as architecture for the implementation of the MixColumns. Therefore in a joint encryptor/decryptor implementation, only the architecture in Figure 5.9 needs to be implemented for both the MixColumns and the InvMixColumns transformations.

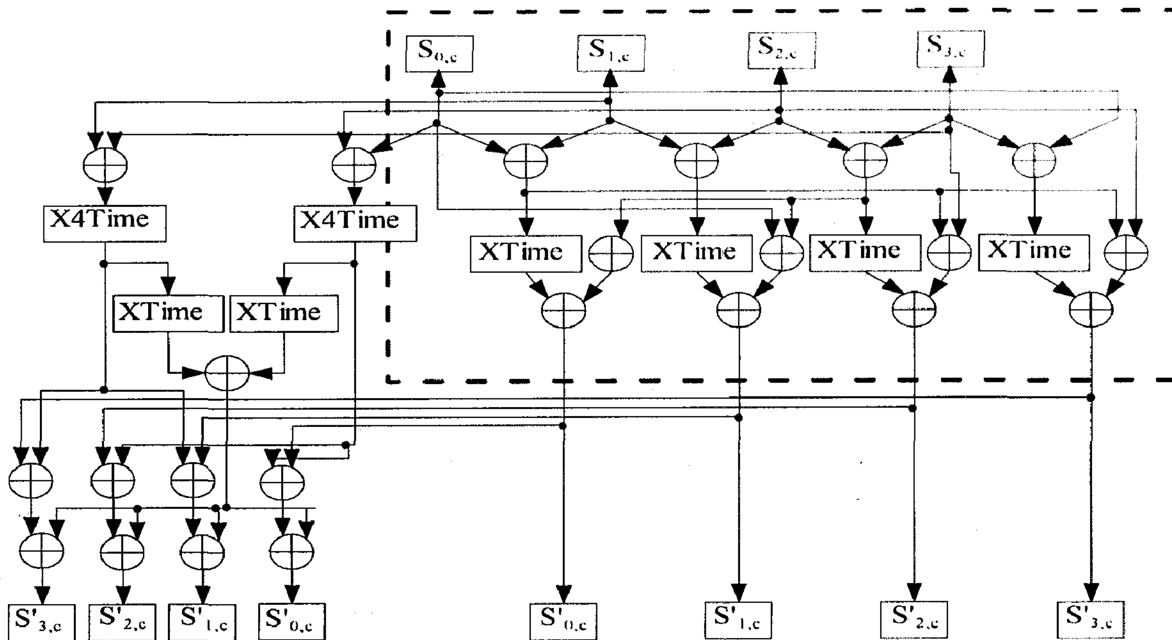


Figure 5.8: Efficient implementation of the MixColumns (red dashed rectangle only) and InvMixColumns transformation.

5.2.2.3 Implementation of Round And Key Expansion unit

Figure 5.9 showing one round unit along with corresponding roundkey generating unit. Both unit working in parallel way, so in subpipline based architecture proper synchronization has to be maintain between data and key within whole unit. Roundkeys can be either generated beforehand and stored in memory or generated on the fly. In the former approach, roundkeys can be read out from memory using appropriate addresses, and there is no extra delay for decryption.

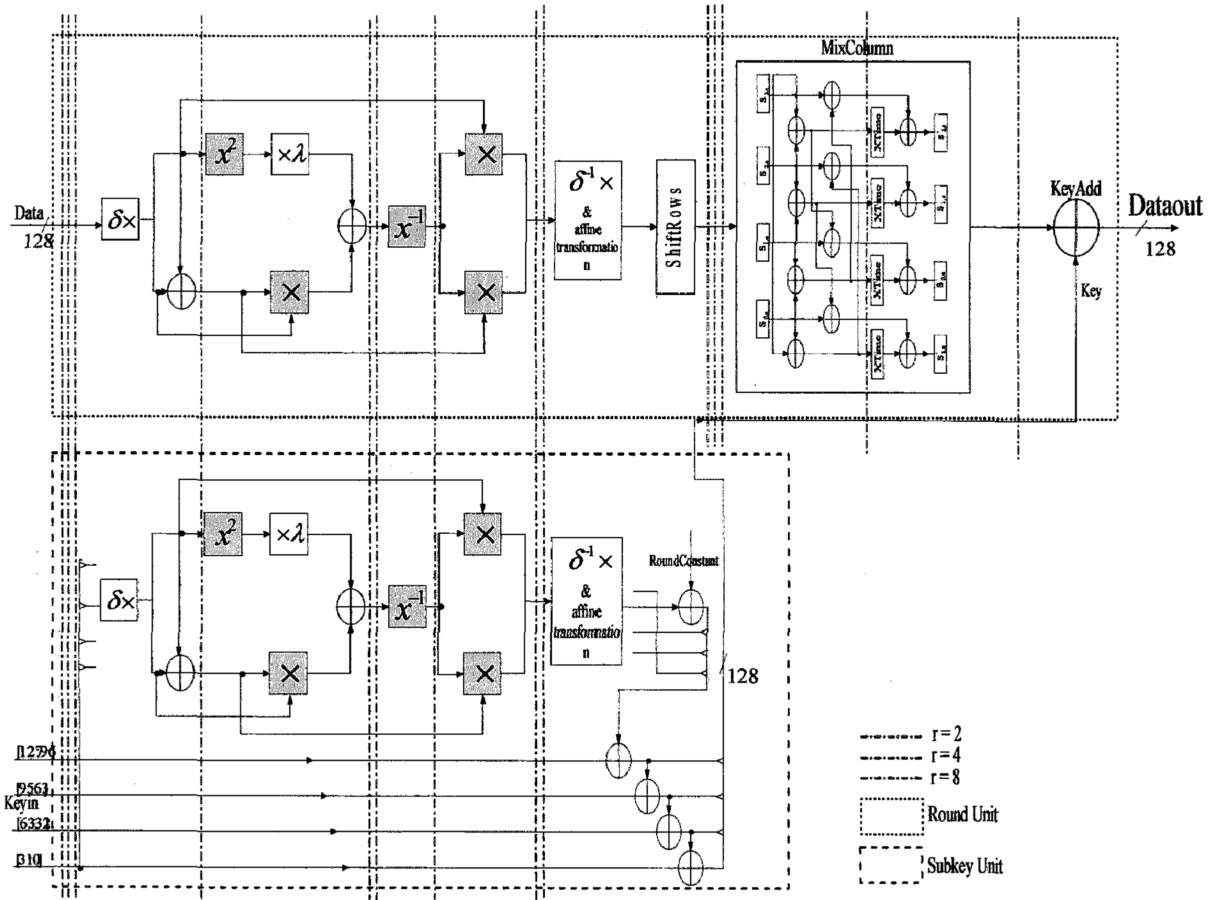


Figure 5.9: Different cutest of Round and Subkey unit for subpipelined architecture.

However, this approach is not suitable for the applications where the key changes constantly. Meanwhile, the delay of memory access is unbreakable, which may offset the speedup achieved by subpipelining the round units. Therefore it is more advantageous to generate roundkeys on the fly in a subpipelined architecture. The subpipelined architecture can achieve maximum speedup if each round unit can be divided into substages with equal delay. Based on the analysis of the gate count in the critical path of each component, cutsets as illustrated in Figure 5.9 can be added to divide the encryption round unit into r = 2, 4 and 8 substages with approximately equal delay. Since the roundkeys are generated on the fly, we need to divide the key expansion unit into the same number of substages with the same maximum delay as in the round unit to avoid extra buffers and delay. Assuming the same subpipelined SubBytes transformation is used in the key expansion unit.

5.2.3 Implementation Result and Comparison

Based on above presented subpipelining concept, we have implemented six designs, in first two design sbox is implemented with LUT and each round having 2 and 3 stage, respectively and in other four design sbox is implemented with composite field arithmetic with each round having 2, 3, 4 and 8 stage. Post-placement timing report shows a fully subpipelined encryptors of 128-bit key with respective substages in each round unit can operate at a throughput of 16.542 Gbps, 13.561Gbps, 15.564 Gbps, 12.971 Gbps , 26.479 Gbps and 31.449 Gbps respectively, on a Xilinx XCV xc4vlx100-12 ff1513 device in non-feedback modes with Xilinx ISE9.1 i is used to synthesize the design and provide post-placement timing results. Detail of each design shown in Table 5.5. We have given each unit a specific name, i.e. (8,CFA,4) showing that each round of design have 8 substages and sbox is implemented with Composite Field Arithmetic concept instead of LUT and there are 4 out of 8 substages present in sbox. The main motto of six designs implementation is to analysis Area-Throughput trade-offs and finally getting highly efficient subpipelined design.

As can be observed Figure 5.10. (2,CFA,0) has high speed as compared to (3,CFA,0) because in both design sbox is in critical path, so increasing substages in remaining round unit shows no improvement in speed, rather reduction of speed due to large area placement complexity. Same reason for high speed of (2,LUT) than (3,LUT).If we compared (2,CFA,0) and (2,LUT) or (3,CFA,0) and (3,LUT), we find that design having sbox implemented using LUT showing 50% more area and 22% speed enhancement than design having CFA based sbox. But, the unbreakable delay introduces by look-up tables restrict substages to r = 2 and so speed.

As can be observed from Table 5.5, our architecture can achieve higher speed than all prior FPGA implementations known to the authors, and more efficient than the previous fastest design [17] in terms of equivalent throughput/slice. In the computation of throughput/slice, one BlockRAM (BRAM) is equivalent to 128 slices [26]. Further speedup can be achieved by dividing each round unit into more substages with equal delay. In this aspect, it has advantages over the designs utilizing BRAMs on Xilinx

FPGAs to implement SubBytes/InvSubBytes. Since the minimum clock period is decided by the unbreakable delay of BRAMs, a fully subpipelined implementation using BRAMs can not achieve higher speed even if larger FPGA devices are available.

Table 5.5: Comparison of FPGA implementation of the AES algorithm.

Design		Device	Frequency (Mhz)	Throughput (Gbps)	Slices	BRAMs	$\frac{Mbps}{Slice}$
Elbirt el al [21]		XCV1000-4	31.8	1.938	10992	0	0.176
Mcloone el. al. [27] (pre-placement time)		XCV812e-8	93.9	12.020	2000	244	0.367
Jarvinen el al [16]		XCV1000e-8	129.2	16.500	11719	0	1.408
Saggese el al [26]		XCV2000e-8	158	20.300	5810	100	1.091
Standaert el [28]		XCV3200e-8	145	18.500	15112	0	1.228
K. Gaj, P. Chodowiec [29]		XCV1000e-8	131	12.20	12600	80	0.97
X. Zhang, K.K. Parhi [17]	(r=3)	XCV812e-6	93.5	16.032	9406	0	1.272
	(r=7)	XCV1000e-8	168.4	21.556	11022	0	1.956
Proposed design	**(2,LUT)**	XCV1000-12	129.232	16.542	16800	200	0.9846
	(3,LUT)	XCV1000-12	121.595	15.564	21571	200	0.7215
	(2,CFA,0)	XCV1000-12	105.932	13.561	11128	0	1.2186
	(3,CFA,0)	XCV1000-12	101.338	12.971	11871	0	1.0926
	(4,CFA,2)	XCV1000-12	206.868	26.479	12726	0	2.08
	(8,CFA,4)	XCV1000-12	245.700	31.449	16478	0	1.9086

Table 5.6: Power consumed.

	Design	Slices	BRAMs	$\frac{Mbps}{Slice}$	Power Consumed (mW)
Proposed design	**(2,LUT)**	16800	200	0.9846	1080
	(3,LUT)	21571	200	0.7215	1150
	(2,CFA,0)	11128	0	1.2186	1058
	(3,CFA,0)	11871	0	1.0926	1101
	(4,CFA,2)	12726	0	2.08	1136
	(8,CFA,4)	16478	0	1.9086	1189

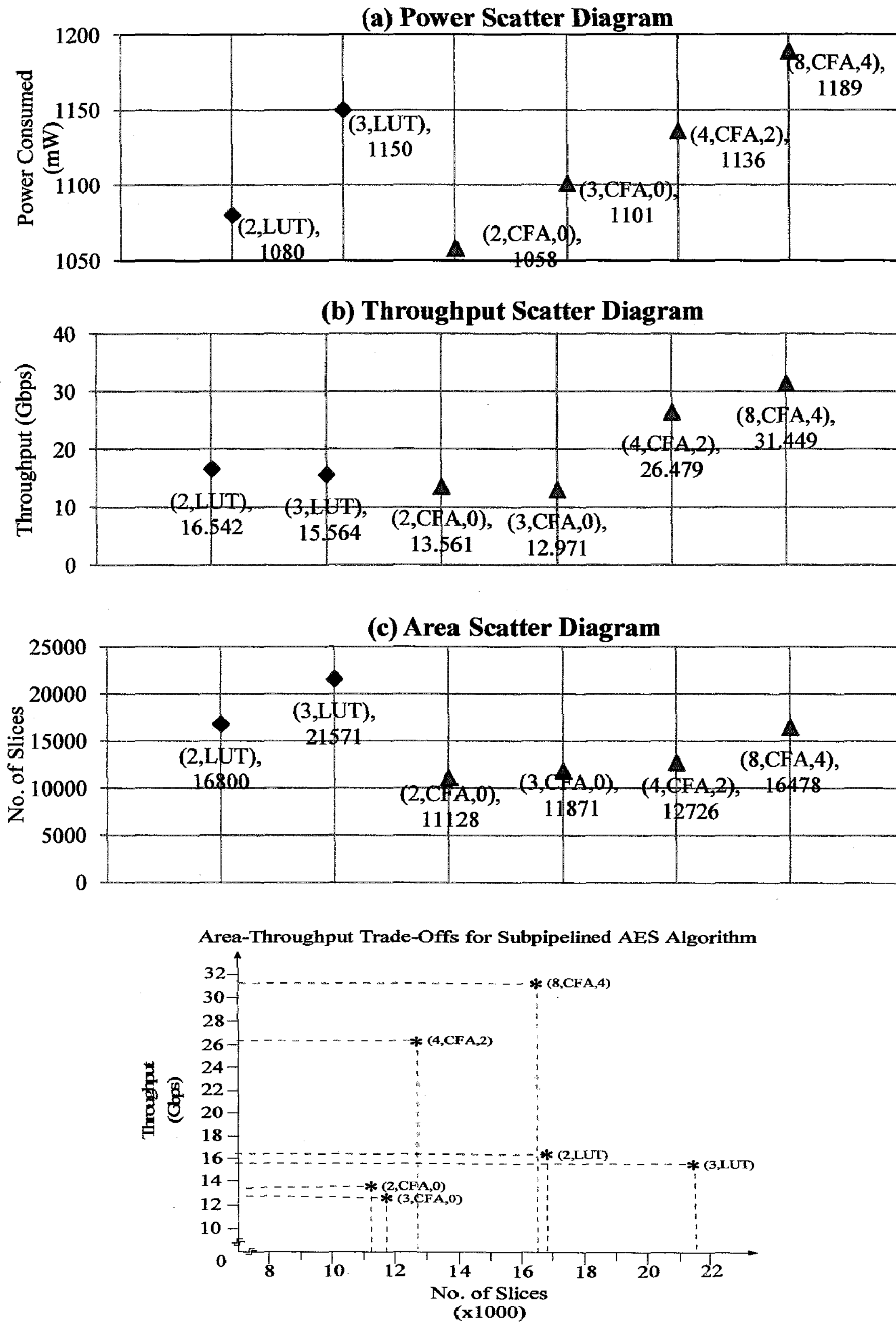


Figure 5.10: Scatter graphs for comparison of (a) power, (b) Throughput, (c) No. of slices, (d) both(b)&(c), of subpipelined AES designs.

CHAPTER 6

FPGA IMPLEMENTATION OF AES-GCM ARCHITECTURE

This chapter describes the AES-GCM implementation on the FPGA Platform. Section 6.1 discusses the architectures of the modules of AES and GHASH, section 6.2 discusses the architectures of the AES-GCM, including IPsec data packet, and GCM data flow. Section 6.3 discusses how to verify the AES-GCM functionality.

6.1 Modules Design

In AES-GCM encryption and AES-GCM decryption, AES and GHASH are the basic modules which are responsible for confidentiality and authentication, respectively. In section 6.1.1, an iterative AES and fully pipelined AES are presented; in section 6.1.2, a bit serial GHASH and a bit parallel GHASH are presented. The pipelined AES and parallel-bit GHASH modules are selected for designing a high speed AES-GCM architecture discussed in section 6.2.

6.1.1 AES Module

For the 128-bit key size, the AES algorithm requires calculating 10 round transformations, and each round contains four phases: SubBytes, ShiftRows, MixColumns, and AddRoundKey (see section 3.1). This allows implementing AES algorithm in either iterative method (see section 5.1) or pipelined method (see section 5.2). In an iterative AES design, the round transformation is instantiated only once. This round transformation block of hardware is used 10 times in 10 computation clock cycles while the intermediate value is stored in a Data register and used as input for the next time. A pipelined AES design can calculate all 10 rounds transformations in one clock cycle by duplicated a single round 10 times (see Figure 6.1). A pipelined AES architecture can be achieved by placing 128-bit registers between each round as we already achieved in section 5.2. In large FPGAs, registers are almost free; a pipelined structure can take advantage of this feature of FPGAs.

The control logic of both the iterative and pipelined AES architectures is implemented by using a finite state machine (FSM). Table 6.1 shows a rough comparison between these two approaches on throughput and cost.

Table 6.1: Comparison between Iterative and Pipelined AES.

Architectures		Num of unroll round in hardware	Num of ciphertext per 10 clock cycle	Cost in Virtex-4			
				No. of slices	Throughput (Gbps)	Throughput/ no. of slices (Mbps/slices)	Power (mW)
Iterative AES	LUT	1	1	2571	1.04977	0.408	1019
	CFA	1	1	1452	0.80736	0.556	1438
Pipelined AES	LUT	10	10	19013	11.94701	0.628	1112
	CFA	10	10	10686	10.61004	0.993	1045

In AES-GCM algorithm, the AES block is implemented in a pipelined architecture. This AES block works as a core in a hybrid from ECB and CTR mode. This hybrid actually is GCTR (see section 3.2.5). Figure 6.1 shows the structure how the AES block is embedded into an ECB module and how the ECB module is embedded into a CTR module. In ECB module, AES block encrypts an input which actually is a continuously increasing counter value in CTR module, and produces the output as keystream. In CTR module, the keystream XORs a plaintext to produce output, i.e. ciphertext. After the first 10 clock cycles, the pipeline is fully filled so that the AES module can output a new 128-bit keystream every clock cycle.

The iterative AES can also be adopted in the AES-GCM algorithm, specifically for generating the hash subkey H. This calculation can be done in advance if the 128-bit key is known because H is nothing but the output of the iterative AES module. Therefore, it needs 10 clock cycles to generate H after inputting 128 bits '0' string into the iterative AES module.

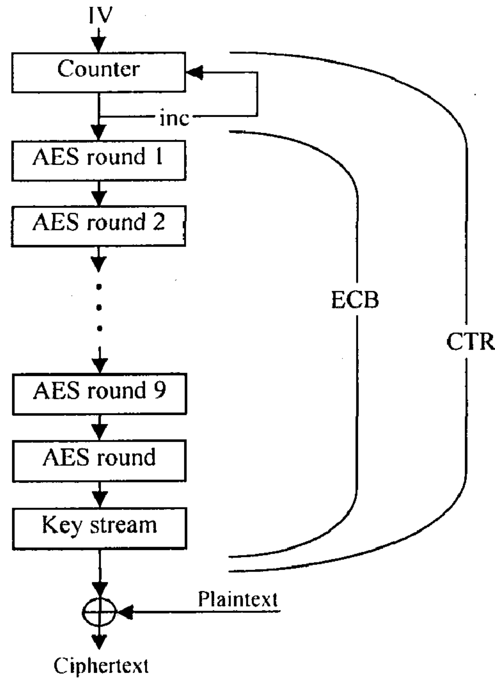


Figure 6.1: AES CTR over ECB Mode Cipher Structure.

As briefly mentioned in section 3.1.1, except for the SubBytes operation in round transformation, the ShiftRows, MixColumns, and AddRoundKey are all directly designed using CLBs in FPGA. SubByte which actually is a LUT operation can be designed either using CLBs or signal-port block select RAM (see section 3.1.2). Table 6.4 in section 6.2.3 shows the differences in performance and cost between AES-GCM implementations which are purely using CLBs (CFA) and those using both CLBs and Block RAMs (LUT).

6.1.2 GHASH Module

A 128-bit multiplier over $GF(2^{128})$ is the core of the GHASH architecture. In AES-GCM. the $GF(2^{128})$ multiplier multiplies two 128-bit operands modulo the field polynomial $F(x) = 1 + x + x^2 + x^7 + x^{128}$ to generate a 128-bit output. The GHASH architecture is shown in Figure 6.2. One operand of the GF multiplier is the hash subkey H which can be treated as a fixed 128-bit constant for it will not change if the 128-bit key does not change. The Register Y whose initial value is zero holds the intermediate hash value for next step authentication computation.

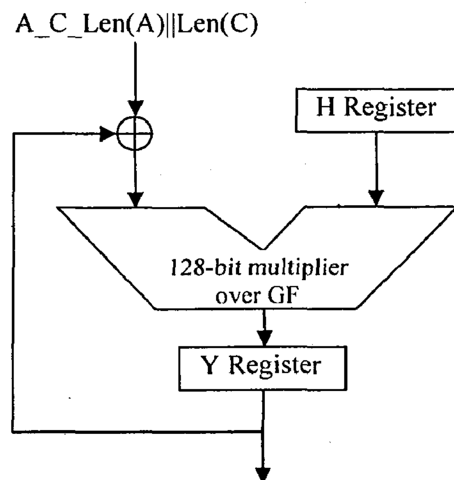


Figure 6.2: GHASH Hardware Architecture.

The architecture shown in Figure 6.2 is based on an iteration operation. Suppose all input data and output data are satisfied with the definitions in section 3.3. In the first m clock cycles, the 128-bit additional authenticated data block sequence (AAD) A_1, A_2, ..., A_m are hashed to the GHASH through one of two inputs of XOR gates as described by algorithm 3.1. In the next n clock cycles, the 128-bit ciphertext block sequence C_1, C_2, ..., C_{n-1}, C_n are hashed to the same input of XOR gates following AAD. In the last clock cycle, 128-bit word length (A)||length(C) is hashed. Meanwhile, the intermediate hash value Y_i (see Figure 3.8) is fed back to another input of XOR gates to generate the another operand for the GF multiplier.

It takes m + n + 1 cycles to compute the hash value for bit parallel multiplier, and 128*(m + n + 1) cycles for Bit Serial multiplier. There is a rough comparison listed in Table 6.2 between GHASH architectures using these two kinds of multipliers.

Table 6.2: Comparison between different GHASH architectures.

GHASH architecture	Latency (clock cycle)	Hardware complexity (k = 128)
Using Bit Serial Multiplier	128*(m+n+1)	O(k)
Using Parallel Multiplier	m+n+1	$O(k^2)$

In order to match pipelined AES module, the GHASH module is implemented using a Bit Parallel GF multiplier (Mastrovito and Karatsuba multiplier). From a whole Mastrovito Bit Parallel $GF(2^{128})$ multiplier point of view, 128^2 two-input AND gates and $O(128^2)$ two-input XOR gates were used for implementation. The delay from this architecture is one AND gate and 7 XOR gates. This is the critical path in the entire AES-GCM circuit design. Although a Bit Parallel multiplier over GF can be pipelined for high data rate [30], this is not the case for GHASH because the GHASH is a kind of feedback mode as mentioned in section 3.3.4.

6.2 High Speed Hardware Implementation of AES-GCM

This section describes the AES-GCM implementation. It begins by a brief introduction on the data packet structure of IPsec ESP [31] in section 6.2.1, then follows with a top level data flow description of the pipelined AES and bit parallel GHASH modules in section 6.2.2. Finally the details of the AES-GCM implementation are presented in section 6.2.3.

6.2.1 Format of Data Packet of IPsec ESP

The IPsec Encapsulating Security Payload (ESP) Packet Format is to arrange input/output data in proper format as shown in Figure 6.3.

The document in [31] clearly explains how to use AES-GCM as an IPsec ESP mechanism to provide confidentiality and data origin authentication.
Information with respect to the format of data packet of IPsec ESP is provided in RFC4106[31]. The Use of Galois-Counter Mode in IPsec ESP is shown in Figure 6.3.

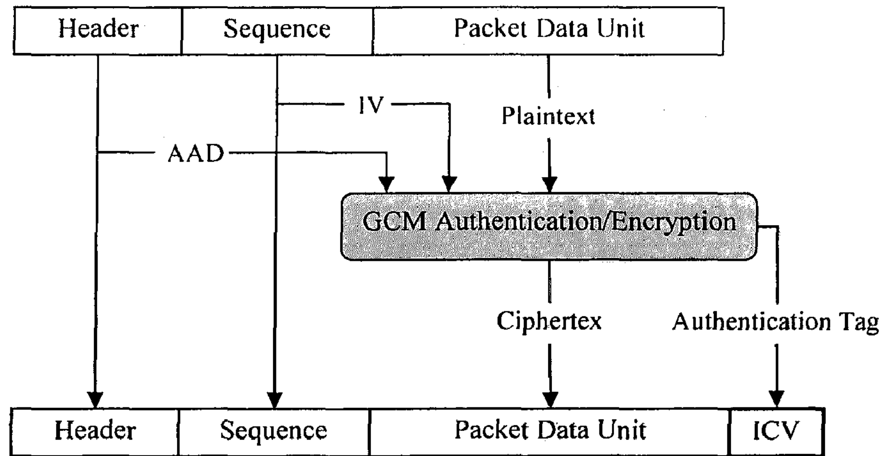


Figure 6.3: The Use of GCM in IPsec ESP [31].

6.2.2 Data Flow in GCM

If the AES module is implemented in the pipelined architecture, the GHASH module is implemented by choosing a parallel-bit multiplier as its core, and the hash subkey H can be calculated out ahead in an iterative AES module based on a known key by each communication party. The data flow in GCM Encryption is shown in Figure 6.4(a); and the data flow in GCM Decryption is shown in Figure 6.4(b). For GCM encryption, AES-GCM starts to compute intermediate hash value Y_i when it receives additional authenticated data. It takes m clock cycles to generate Y_m. Then the GHASH has to be idle for 11 clock cycles until the first ciphertext block C_1 is generated by the GCTR which is created by using a pipelined AES module. For GCM with default IV, the IV is always 96 bits long, and J_0 can be created instantly by concatenation of bit strings.

The key streams for GCM encryption are created after the 10th clock cycle when J_0 is input into the pipeline of GCTR. At the 11th clock cycle, cipher block C_1 is generated and input to GHASH. GHASH begins to hash data again. At the m+11+n+1 clock cycle, Y_{m+n+1} is generated and XORed with K_0 (i.e. $CIPH_K(J_0)$) to create authentication tag T.

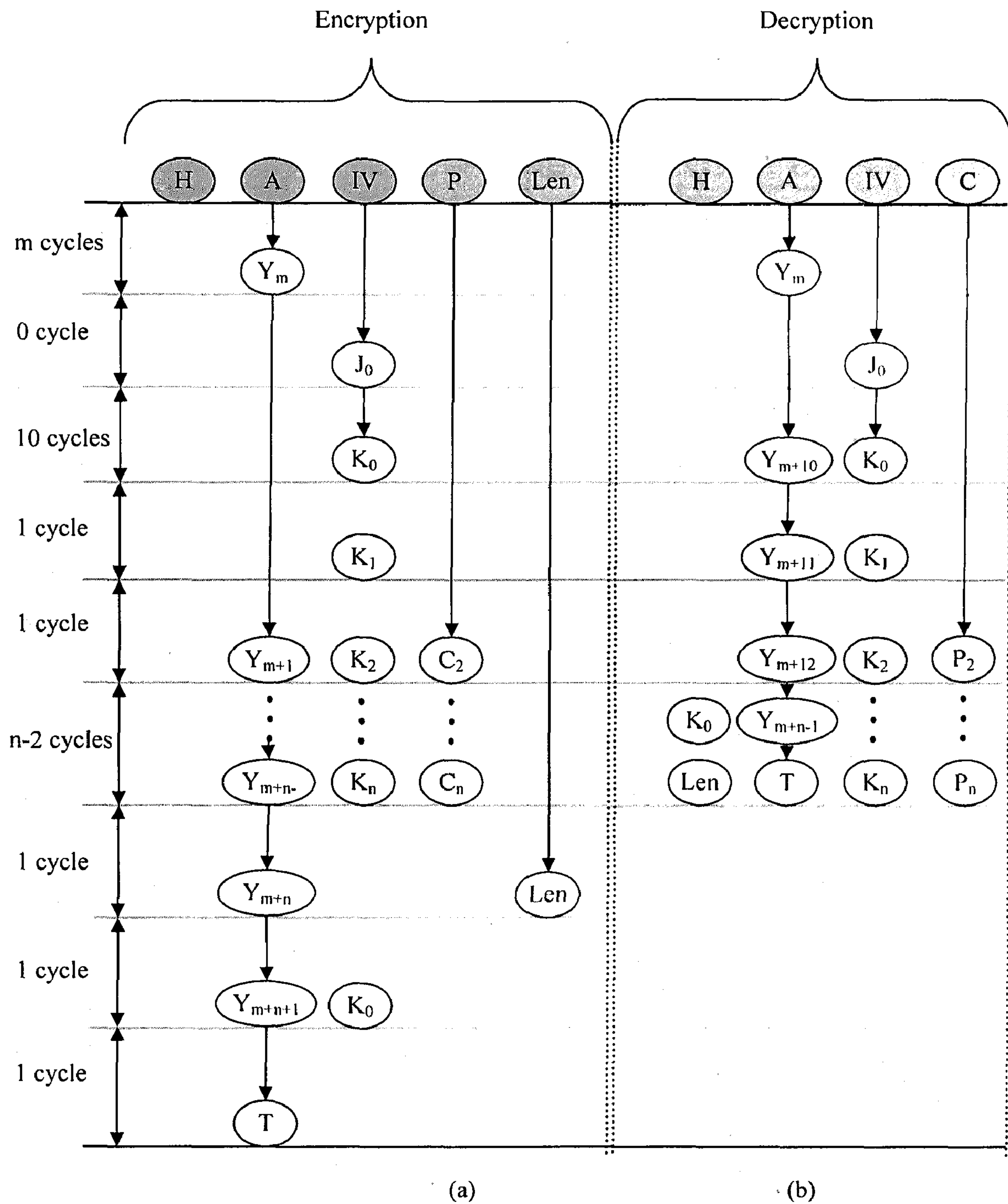


Figure 6.4: (a) The Data Flow of GCM Encryption (b) The Data Flow of GCM Decryption.

For GCM decryption, GHASH can directly compute the authentication tag T' based on AAD and ciphertext C from the input of GCM Decryption. Therefore, the max 11 clock cycles are saved compared with data flow in GCM Encryption.

6.2.3 Hardware Implementation Bidirectional GCM

Based on the data flow analysis in section 6.2.2, a bidirectional AES-GCM hardware module is built. The "bidirectional" means: the AES-GCM module can work not only as GCM encryption but also as GCM decryption depending on the logic value of the control signal Encryption. If the Encryption signal is high, then AES-GCM works in GCM encryption mode, otherwise, it works in GCM decryption mode. The schematic of both designs are same as shown in Figure 6.5 and 6.6, except tag comparison circuit in decryption mode.

If Encryption signal is high, then AES-GCM works in GCM encryption mode i.e. AES-GCM-AE (see section 3.3.6.1). In Figure 6.5, the data paths are 128-bit wide. The control signals do not show up except signal Encryption. They all are driven by a finite sate machine (FSM) module which is designed according to IPsec ESP packet format.

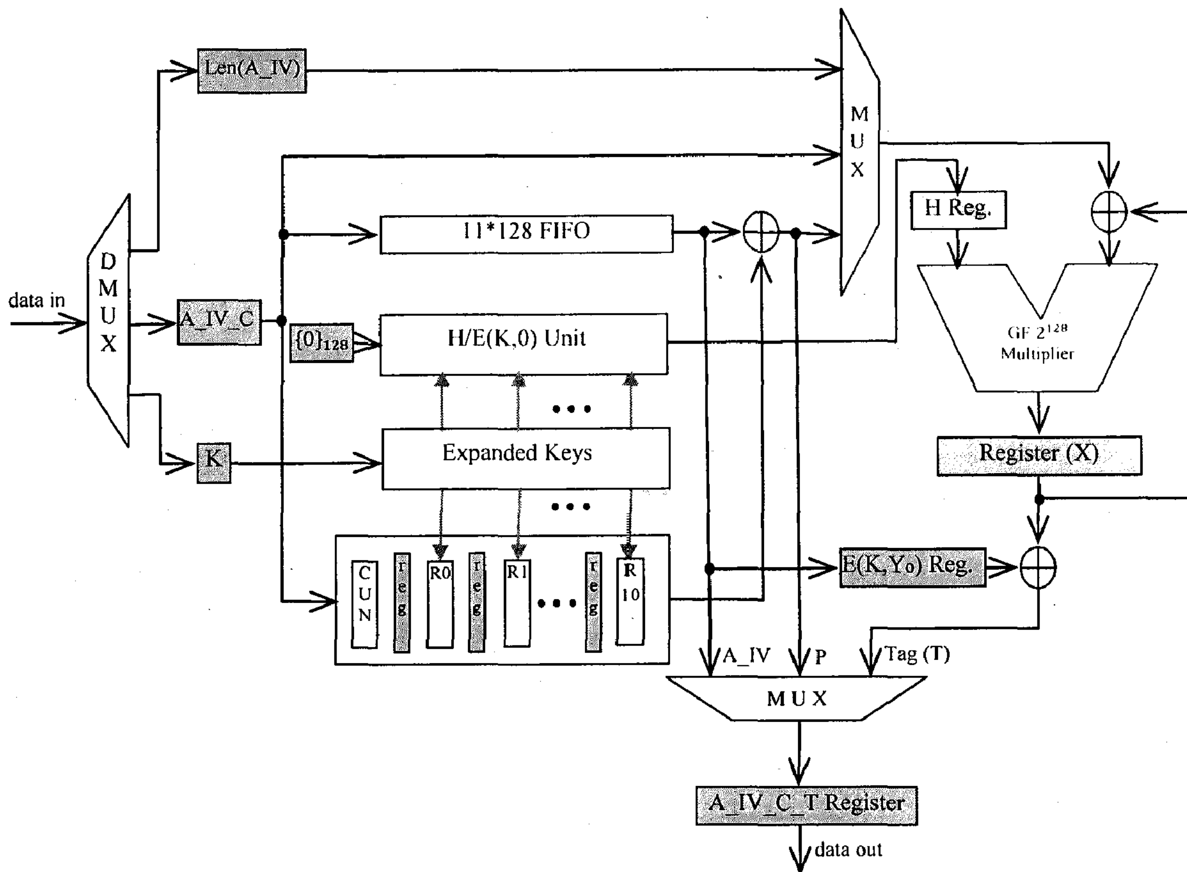


Figure 6.5: AES-GCM Encryption Architecture.

The 44 32-bit round-key words are stored in a look up table instead of generated in real time. The hash subkey H is generated by an iterative AES module from the key K in advance.

A 3-to-1 multiplexer MUX-I is used whose output connects to one of the input ports of XOR gates in GHASH module. The three inputs of MUX-I are additional authenticated data AAD, ciphertext C and length information length (A)||length(P). As discussed in section 3.3, in the first m clock cycles, the output of MUX-I is the additional authenticated data A. After 11 clock cycles, in the next n clock cycles, the output of MUX-I switches to the ciphertext C. The Final output of MUX-I is length (A)||length(P). The first 128-bit key stream which is produced by GCTR, from the initial value IV of GCM, is stored in the $AES_K(J_0)$ Register. This $AES_K(J_0)$ Register is later used to generate the authentication tag T. Since the IV of GCM is followed by plaintext P, and the first 128-bit keystream is generated by GCTR after a delay of 10 clock cycles. Therefore, the plaintext P is delayed by 11 clock cycles in order to be encrypted by the corresponding key streams. A 11* 128-bit FIFO meets this requirement. In the first 11 clock cycles, the data flow AAD, IV and payload data P are input to the FIFO. From the 12th clock cycle onwards, the FIFO remains in a dynamic full status by reading data out and writing new data in simultaneously until reaching the end of the IPsec ESP packet. Suffering 11 clock cycles delay through the FIFO, AAD and IV connect directly to one of the inputs of the 3-to-1 MUX-II; delayed payload data P exclusive-ORs with GCTR output, key stream, to produce ciphertext which is connected to one input of MUX-I and MUX-II. The left input of MUX-II is the authentication tag T which is the result of GHASH final output Y_{m+n+1} XORing value in $AES_K(J_0)$ Register. MUX-II output connects to register Output. The final output of AES-GCM-AE from register Output is data flow A_IV_C_T corresponding to the input data flow A_IV_P.

As mentioned in section 6.1.2, the critical path of this design is determined by the GHASH module. The delay of all other paths in Figure 6.5 is smaller than the delay produced by GHASH module.

If Encryption is low, then AES-GCM works as GCM decryption as shown in Figure 6.6 i.e. AES-GCM-AD (see section 3.2.6.2). AES-GCM-AD is similar to AES-GCM-AE.

Compared with Figure 6.5, one difference is that two 2-to-1 multiplexers, also named MUX-I and MUX-II are used instead of two 3-to-1 multiplexers in the Figure 6.5. The reason that 2-to-1 multiplexers are used is that the authentication tag T' is computed directly from A and C of the original input A_IV_C_T and it does not need to be input into register Output either. Another difference is that a 128-bit comparator is used to generate the FAIL signal depending on the comparison between T and T'. The delay of the comparator is 1 XOR gate plus 7 OR gates which is still smaller than the delay of the 128-bit bit parallel multiplier over GF(2^{128}) which is 1 AND gate plus 7 XOR gates.

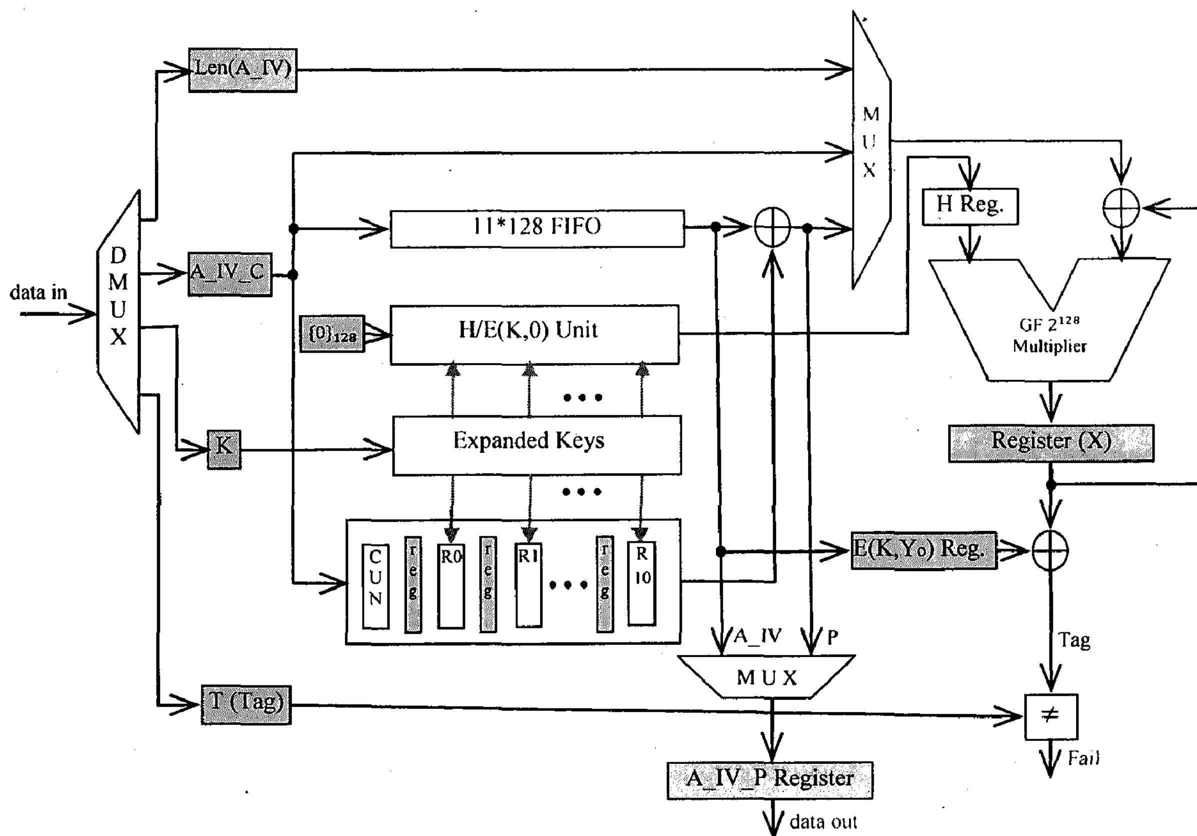


Figure 6.6: AES-GCM Decryption Architecture.

Like the S-box of AES module in section 6.1.1, the 11*128-bit FIFO can also be implemented by using dual-port Block SelectRAM+ or dual-port Distributed SelectRAM+. Therefore, AES-GCM can be implemented either by purely using CLBs (named as CFA) or using CLBs and block RAM (named as LUT). Table 6.3 and 6.4 lists the performance and cost comparison of these designs. Table 6.3 shows resources

utilization by AES datapath and Key expansion unit. Table 6.4 shows full AES-GCM unit's resources utilization in Virtex-4 xc4vlx200-11-ff1513. For the LUT based scheme (1), 58.3% ((130*128+12039)/49152=0.583) slices & 54% (130/240=0.369) BRAM

Table 6.3: Place and Route Results Summary of other important units of AES-GCM.

AES-GCM Design	Units	Delay (ns)	Frequency (MHz)	Throughput (Gbps)	No. of Slices	Kbps / Slices
LUT Based	Key Expansion	3.722	268.670	34.389	3465	9924.67
	AES Data Path	9.878	101.232	12.957	8564	1513.04
CFA Based	Key Expansion	9.137	109.445	14.008	2750	5093.81
	AES Data Path	11.637	85.932	10.999	7625	1442.49

blocks are used to implement AES-GCM with Mastrovito multiplier and 48.7% ((130*128+7304)/49152=0.487) slices & 54% (130/240=0.369) BRAM blocks are used with Karatsuba multiplier; for the CFA based scheme (2), 40.6% (19957/49152=0.406) slices are used to implement AES-GCM with Mastrovito multiplier and 41.3% (20320/49152=0.413) slices are used with Karatsuba multiplier.

Table 6.4: Full AES-GCM's Place and Route Results Summary.

AES-GCM Design	With Multiplier	Delay (ns)	Frequency (MHz)	Throughput (Gbps)	RAM Blocks	No. of Slices	Kbps / Slices
LUT Based	Mastrovito	9.613	104.026	13.315	130	128*130+12039 =28679	464.28
	Karatsuba	12.785	78.217	10.012	160	128*160+7,304 =27784	360.35
CFA Based	Mastrovito	10.442	95.767	12.258	0	19957	614.22
	Karatsuba	11.575	86.393	11.058	0	20320	544.21

Table 6.5: Power analysis of the designs.

AES-GCM Design	With Multiplier	No. of gates	Power (mW)
LUT Based	Mastrovito	8,735,739	1284
	Karatsuba	10,610,177	1804
CFA Based	Mastrovito	299,411	1136
	Karatsuba	301,131	1594

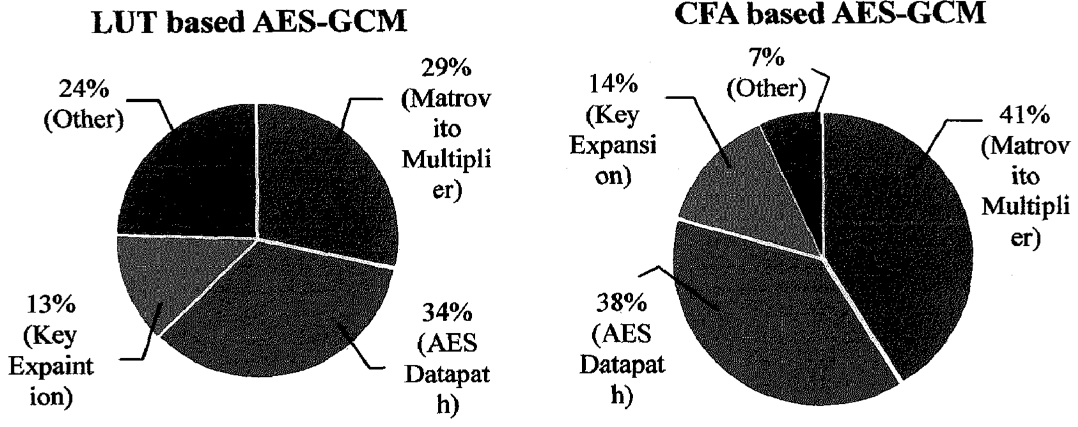


Figure 6.7: Area comparison of various units of Full AES-GCM.

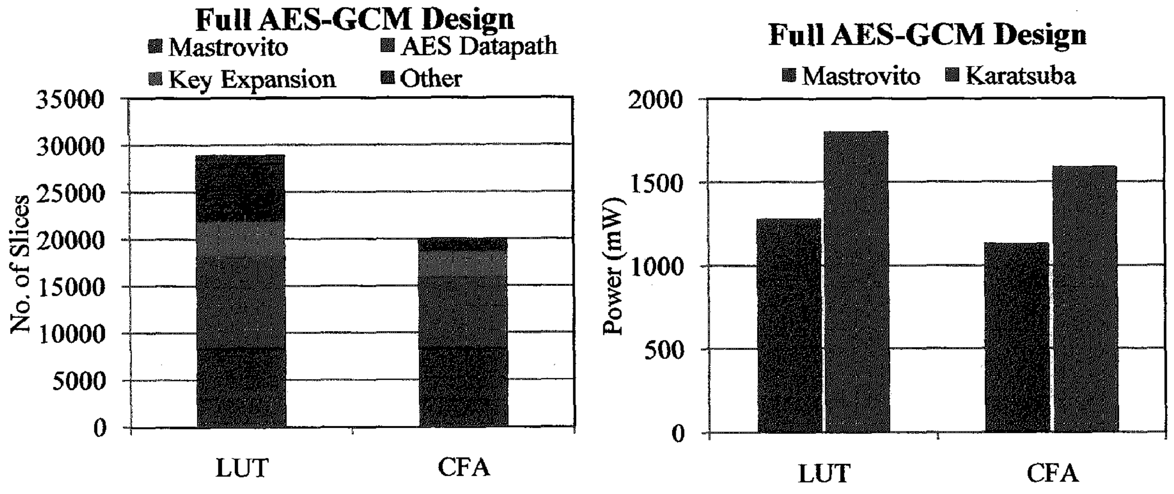


Figure 6.8: Area and power comparison of two type of AES-GCM.

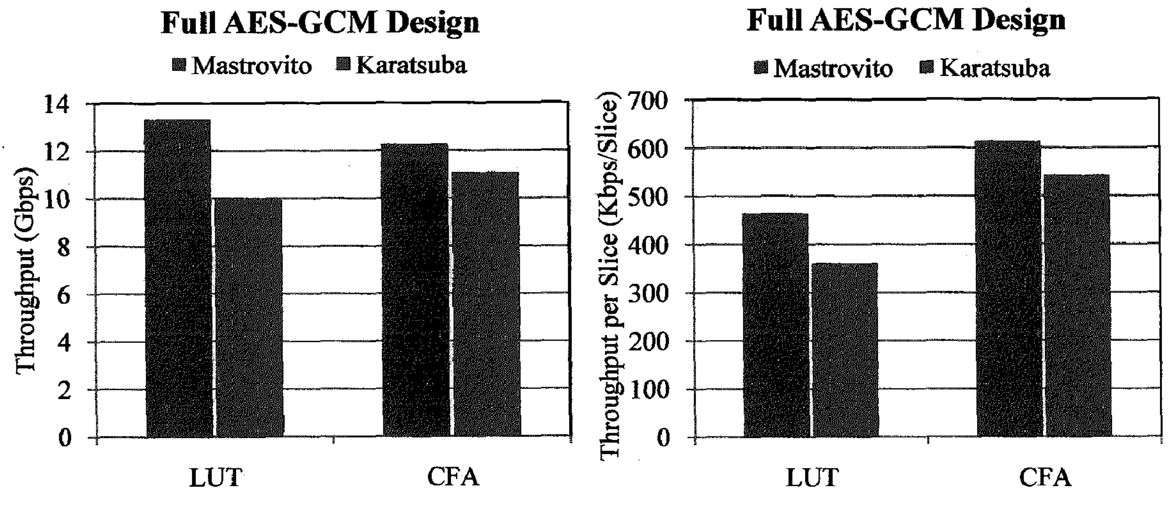


Figure 6.9: Throughput and throughput per slice comparison of two type of AES-GCM.

Figure 6.7, 6.8, 6.9 providing various important comparison columns bar diagrams, made by using information given in above tables, which make their interpretation easy.

6.3 Verification of AES-GCM Functionality

This section describes how the modules were verified in the realistic environment CMC-prototype-platform. All of them including AES, GHASH, AES-GCM-AE, and AES-GCM-AD were verified on this platform. They were also although designed in VHDL and timing simulated using Modelsim, respectively. The results are compared with other researches on hardware implementations of AES-GCM.

6.3.1 IPsec Signal Generator

In order to perform verification, an IPsec ESP signal Generator had to be built based on the IPsec ESP data packet format discussed in section 6.2.1. Figure 6.10 shows a 16-bit LFSR which generates 2^{16}-1 bit stream sequence periodically based on a primitive polynomial $f(x)= 1+x+x^3+x^{12}+X^{16}$ for building the IPsec signal generator which consisted of 8 16-bit LFSRs. The primitive polynomial with degree 16 was chosen since the maximum length of payload data of IPsec data packet is 2^{16} bit long. At the beginning, the control signal start_LFSR asserts for m clock cycles, the signal generator generates m blocks of parallel 128-bit data as AAD; at the next clock cycle, start_LFSR desserts for generating IV-GCM; sequentially, start_LFSR asserts again for n clock cycles in order to generate n blocks of parallel 128-bit data as payload data P. The values of m and n are controlled by one input of the signal generator, in other words, it is adjustable to meet the test requirement.

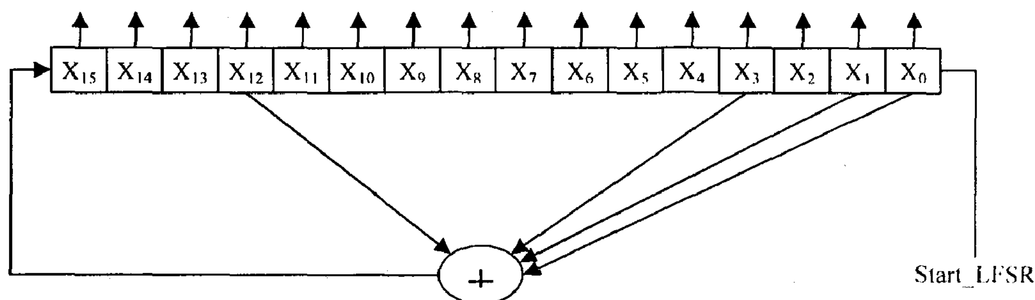


Figure 6.10: 16-bit LFSR for IPsec ESP Signal Generator.

6.3.2 Verifying Both AES-GCM-AE and AES-GCM-AD on FPGA

In Figure 6.11, two AES-GCM modules are used, one working as AES-GCM-AE by connecting Encryption to the power, one working as AES-GCM-AD by connecting Encryption to the ground. The mimic IPsec data packets A_IV_P from the signal generator go through the AES-GCM-AE and the AES-GCM-AD consecutively, and then go to the comparison module in which there is another identical IPsec signal generator for checking the recovered data P validity. If each node in Figure 6.8 works correctly, then the plaintext P will be recovered from the AES-GCM-AD without any bit-errors, the signal Verifying_GCM will go to high to indicate the AES-GCM-AE and the AES-GCM-AD have been verified successfully. The comparison between T and T' is handled in the AES-GCM-AD module. If T is not equal to T', then the signal T_Verification (corresponding the output Fail in algorithm 3.4) will be set to high to indicate that the inputs are not authentic. The signal Verifying_GCM and T_Verification physically are connected to two LEDs on the FPGA-platform in order to observe the verification results.

After choosing 40MHz clock input as the global clock of FPGA, downloading the bitstream of the described architecture in Figure 6.7 to the Virtex-4 xc4vlx200-11-ff1513, the LED Verifying_GCM turns on and LED T_Verification remains off. Hence the module with AES-GCM functionality is implemented successfully on FPGA platform.

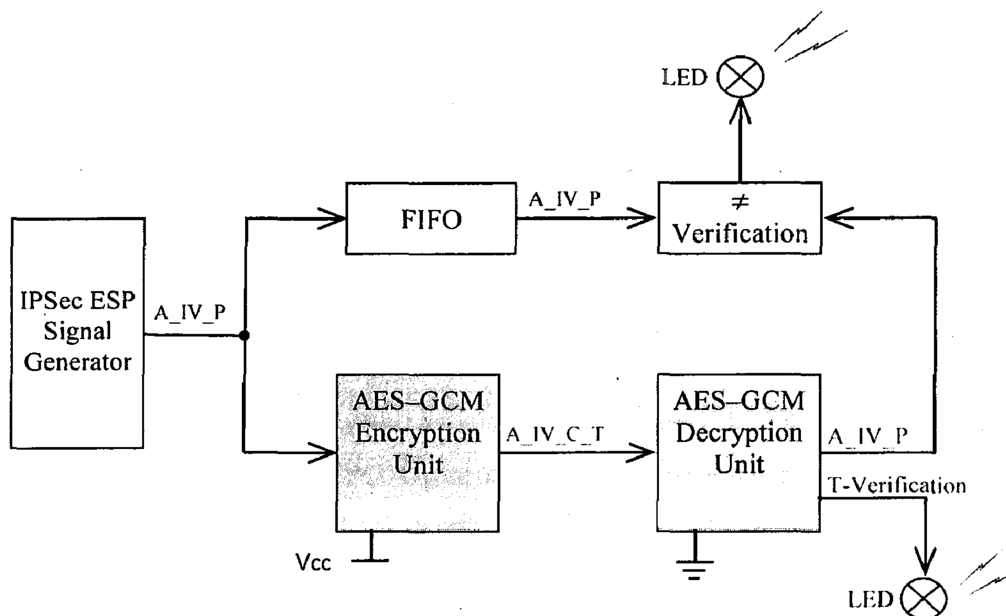


Figure 6.11: AES-GCM Verification System.

In addition, in the appendix A of [13], the designers of GCM provides several cases of test vectors for testing AES-GCM implementation designs with different AES key sizes. The Test Case 3 and Test Case 4 are chosen to verify the work in this thesis. More specifically, first, using the 128-bit secret key K provided in Test Case 3 or Test Case 4 generates not only 44 32-bit expanded key words for AES round-transformations but also hash subkey H for GHASH hash operations; second, using the additional authentication data A, the initial vector IV, and the plaintext data P provided in Test Case 3 or Test Case 4 as parameters builds a test-bench which works as a stimulus to output data flow A_IV_P into AES-GCM module for timing simulation; Finally, comparing the results A_IV_C_T of the timing simulation of AES-GCM with the A'_IV'_C'_T' provided in Test Case 3 and Test Case 4 and make sure they are identical (see the dash-line part of Figure 6.11).

All the VHDL codes for generating AES-GCM, test benches, and test vectors are printed out and listed in Appendixes. The hierarchical HDL code designs are shown in Figure 6.12.

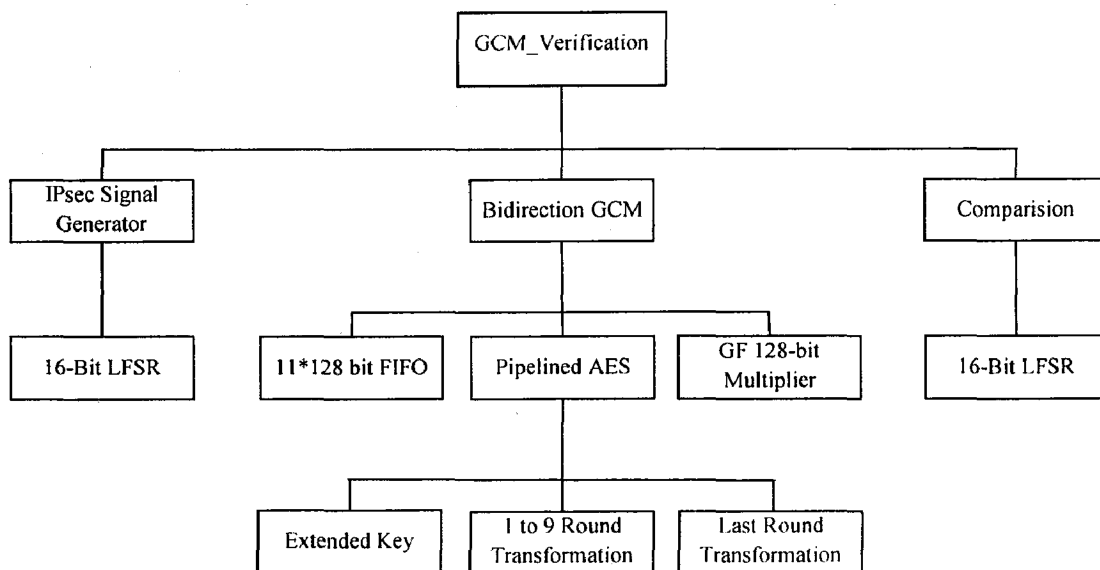


Figure 6.12: AES-GCM Hierarchical HDL Codes Design.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this dissertation, the development of a complete architecture for two mode of the AES security standard has been presented and implemented on modern Xilinx virtex-4 FPGA platform.

Initially, two architecture of AES-ECB mode; compact single round (iterative) and pipelined architecture presented, and implemented on FPGA for both CAF and LUT based schemes.

Then, using above design, a highly pipelined and parallelized architecture of AES-GCM mode integrated and implemented on FPGA for both CFA and LUT based schemes, while GHASH implementation scheme is discussed in bit-parallel methods. and two well known parallel modular multipliers; Mastrovito and Karatsuba multiplier has been implemented on FPGA. Finally, feasibility of the AES-GCM architecture has been verified through verifying circuit.

On compared with previous researches, presented FPGA architectures of AES modes are robust and achieve a good throughput.

The Contributions achieved by this work are as follows:

- Presented compact single round (iterative) architecture based on AES-ECB mode for FPGA.
- Detail study of pipelining and subpipelining architectures based on AES-ECB made and implemented on FPGA.
- Implementation of the AES-GCM security standard has been performed in a FPGA platform.
- AES-GCM module can work in bidirectional, either GCM encryption or GCM decryption, mode.

- All the above designs are implemented both for LUT and CAF, and performance comparison performed for each design.
- Power used by various design has been also analyzed along with throughput and area.

7.2 Future Work

As in this work, all the design discussed and implemented based on 128 bit key. So all these design can be explore for 192 and 256 bit key.

Speed and area optimization is main focus during this work, although power has been calculated but there is not any specific method used for its optimization, so it could be good area to work.

Dynamic reconfigurable system for specific application and its real time implementation can be made using one of the designs implemented in the work.

A new public-key cryptographic scheme; Elliptical curve cryptography is recently quite famous for their high security features as compared to presented AES scheme. But its limited speed is big obstacle for their commercialization in modern high speed application. So finding solution of this problem can be a good future research field.

REFERENCES

[1] NIST Special Publication 800-38A, "*Recommendation for Block Cipher Modes of Operation—Methods and Techniques*", December 2001. http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[2] NIST Special Publication 800-38B, "*Recommendation for Block Cipher Modes of Operation: the CMAC Authentication Mode*", U.S. DoC/NIST, October 2003. http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[3] NIST Special Publication 800-38C, "*Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*", U.S. DoC/NIST, May 2004. http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf

[4] NIST Special Publication 800-38D Draft, "*Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) for Confidentiality and Authentication*", April 2006. http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

[5] FIPS Publication 197, "*The Advanced Encryption Standard (AES)*", U.S. DoC/NIST, November, 2001. http://www.securitytechnet.com/resource/crypto/standard/fips/fips-197.pdf

[6] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, "*Handbook of Applied Cryptography*", CRC Press LLC, 1997.

[7] Leilei Song; Parhi, K.K., "*Low-complexity modified Mastrovito multipliers over finite fields $GF(2^M)$*", Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on , vol.1, no., pp.508-512 vol.1, Jul 1999

[8] T. Wollinger and C. Paar., "*How secure are FPGAs in cryptographic applications?*", In Proc. of the 13th Int'l Conference on Field-Programmable Logic and its Applications (FPL), pages 91–100, 2003.

[9] A. Satoh, S. Morioka, K. Takano, and S. Munetoh., "*A Compact Rijndael Hardware Architecture with S-Box Optimization*", Advances in Cryptology- ASIACRYPT, pages 239-254, 2001. http://www.springerlink.com/index/bc7dvd7ymadu3j8l.pdf

[10] V. Rijmen., "*Efficient Implementation of the Rijndael S-box*". http://www.comms.scitech.susx.ac.uk/fft/crypto/rijndael-sbox.pdf

[11] M. Alam, S. Ghosh, D. RoyChowdhury, and I. Sengupta., "*Single Chip Encryptor/Decryptor Core Implementation of AES Algorithm*", 21st International Conference on VLSI Design, 2008. VLSID 2208., pages 693-698, 2008.

[12] M.Machhout, M.Zeghid, W.El hadj youssef, B.Bouallegue, A.Baganne, R.Tourki, "*Efficient Large Numbers Karatsuba-Ofman Multiplier Designs for Embedded Systems*", International Journal of Electronics, Circuits and Systems, 2009.

[13] D. McGrew, J. Viega, "*The Galois/Counter Mode of Operation (GCM)*", May 31, 2005. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes

[14] C. Paar., "*A new architecture for a parallel finite field multiplier with low complexity based on composite fields*", IEEE Transactions on Computers, 45(7):856-861, 1996.

[15] Henry Kuo and Ingrid Verbauwhede, "*Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijindael Algorithm*", in 3rd international workshop cryptographic Hardware and embedded systems (CHES 2001), LNCS2162, Paris, May 2001,pp 51-64.

[16] Kimmo U. Järvinen, Matti T. Tommiska, and Jorma O. Skyttä., "*A fully pipelined memoryless 17.8 Gbps AES-128 encryptor*", In Proceedings of the 11th ACM International Symposium on Field-Programmable Gate Arrays, FPGA 2003, pages 207–215.

[17] X. Zhang, K.K. Parhi, "*High-speed VLSI architectures for the AES algorithm*", IEEE Transactions on Very Large Scale Integration (VLSI) Systems 12 (9) (2004) 957–967.

[18] P. Chodowiec and K. Gaj., "*Very Compact FPGA Implementation of the AES Algorithm*", In Cryptographic Hardware and Embedded Systems, CHES 2003, pages 319–333. Springer, Sept. 2003.

[19] Chodowiec P., Gaj K., Bellows P., Schott B., "*Experimental Testing of the Gigabit IPSec Compliant Implementations of Rijndael and Triple DES Using SLAAC-1*

VFPGA Accelerator Board", Information Security Conference (ISC 2001), Malaga, Spain, 2001.

[20] Dandalis A., Prasanna V.K., Rolim J.D., "*A Comparative Study of Performance of AES Final Candidates Using FPGAs*", Cryptographic Hardware and Embedded Systems Workshop (CHES 2000), Worcester, Massachusetts, 2000.

[21] Elbirt A.J., Yip W., Chetwynd B., Paar C., "*An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists*", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 9 Issue: 4, August 2001.

[22] Gaj K. and Chodowiec P., "*Comparison of the hardware performance of the AES candidates using reconfigurable hardware*", Third Advanced Encryption Standard (AES3) Candidate Conference, New York, 2000.

[23] M. H. Jing, Y. H. Chen, Y. T. Chang, and C. H. Hsu, "*The design of a fast inverse module in AES*", in Proc. Int. Conf. Info-Tech and Info-Net, vol. 3, Beijing, China, Nov. 2001, pp. 298–303.

[24] C. C. Lu and S. Y. Tseng, "*Integrated design of AES (advanced encryption standard) encrypter and decrypter*", in Proc. IEEE Int. Conf. Application Specific Systems, Architectures Processors, 2002, pp. 277–285.

[25] X. Zhang and K. K. Parhi, "*Implementation approaches for the advanced encryption standard algorithm*", IEEE Circuits Syst. Mag., vol. 2, no. 4, pp. 24–46, 2002.

[26] G. P. Saggese, A. Mazzeo, N. Mazocca, and A. G. M. Strollo, "*An FPGA based performance analysis of the unrolling, tiling and pipelining of the AES algorithm*", in Proc. FPL 2003, Portugal, Sept. 2003.

[27] M. McLoone and J. V. McCanny, "*Rijndael FPGA implementation utilizing look-up tables*", in IEEE Workshop on Signal Processing Systems, Sept. 2001, pp. 349–360.

[28] F. Standaert, G. Rouvroy, J. Quisquater, and J. Legat, "*Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements & design tradeoffs*", in Proc. CHES 2003, Cologne, Germany, Sept. 2003.

[29] K. Gaj, P. Chodowiec, "*Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays*", CT-RSA 2001, LNCS 2020, 2001, pp. 84–99.

[30] G. Ahlquist, B. Nelson, and M. Rice, "*Optimal Finite Field Multipliers for FPGAs, International Workshop on Field Programmable Logic and Applications*", pp. 51-60, August, 1999.

[31] J. Viega, D. McGrew, "*The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)*", Internet proposed standard RFC 4106, June 2005.http://tools.ietf.org/html/rfc4106

[32] Joan Daemen, Vincent Rijmen, "*AES Proposal: Rijndael*", September 2000. http://csrc.nist.gov/encryption/aes/round2/AESAlgs/Rijndael/Rijndael.pdf

[33] Bo Yang, Sambit Mishra, and Ramesh Karri., "*High Speed Architecture for Galois/Counter Mode of Operation (GCM)*", Cryptology ePrint Archive, Report 2005-156, May 2005. http://eprint.iacr.org/2005/146

PAPERS PUBLISHED

1. **Vishwanath Patel**, R. C. Joshi, A. K. Saxena, "*FPGA Implementation of DES Using Pipeline Concept with Skew Core Key Scheduling*" Journal of Theoretical and Applied Information Technology, Vol 5. No3. March-2009.
2. **Vishwanath Patel**, R. C. Joshi, A. K. Saxena, "*High-Performance FPGA Implementation of Compact Single Round AES Design*", International Conference on Computer and Network Technology (ICCNT 2009).*(in press)*
3. **Vishwanath Patel**, R. C. Joshi, A. K. Saxena, "*Efficient Composite Field Arithmetic Based Subpipelined VLSI Architectures for the AES Algorithm*", International Journal of Educational Technology (IJET), Inderscience Publishers. *(under review)*
4. **Vishwanath Patel**, R. C. Joshi, A. K. Saxena, "*FPGA Implementation of Advance Encryption Standard-Galois Counter Mode*". *(under writing)*

APPENDIX A

TEST-VECTORS FOR AES-GCM [13]

GCM Test Case #03 (AES-128)

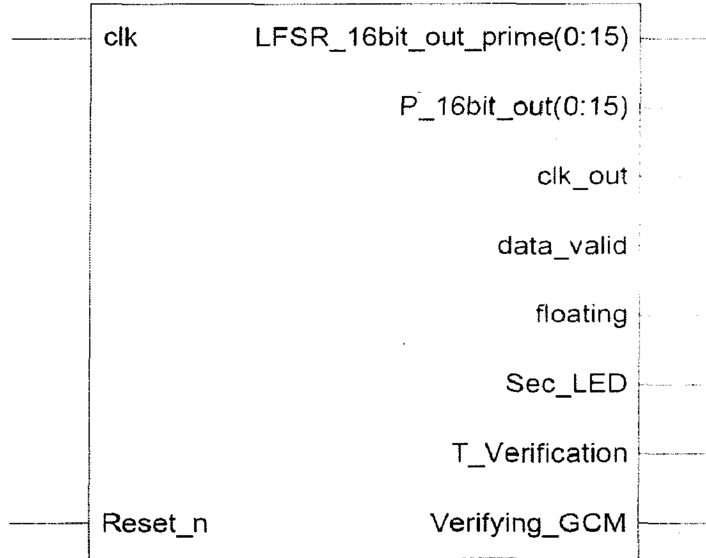
Variable	Value
K	: feffe9928665731c6d6a8f9467308308
P	: d9313225f88406e5a55909c5aff5269a
	: 86a7a9531534f7da2e4c303d8a318a72
	: 1c3c0c95956809532fcf0e2449a6b525
	: b16aedf5aa0de657ba637b391aafd255
IV	: cafebabefacedbaddecaf888
H	: b83b533708bf535d0aa6e52980d53b78
Y_0	: cafebabefacedbaddecaf88800000001
E(K,Y_0)	: 3247184b3c4f69a44dbcd22887bbb418
Y_1	: cafebabefacedbaddecaf88800000002
E(K,Y_1)	: 9bb22ce7d9f372c1ee2b28722b25f206
Y_2	: cafebabefacedbaddecaf88800000003
E(K,Y_2)	: 650d887c3936533a1b8d4e1ea39d2b5c
Y_3	: cafebabefacedbaddecaf88800000004
E(K,Y_3)	: 3de91827c10e9a4f5240647ee5221f20
Y_4	: cafebabefacedbaddecaf88800000005
E(K,Y_4)	: aac9e6ccc0074ac0873b9ba85d908bd0
X_1	: 59ed3f2bb1a0aaa07c9f56c6a504647b
X_2	: b714c9048389afd9f9bc5c1d4378e052
X_3	: 47400c6577b1ee8d8f40b2721e86ff10
X_4	: 4796cf49464704b5dd91f159bb1b7f95
len(A)||len(C)	: 00000000000000000000000000000200
GHASH(H,A,C)	: 7f1b32b81b820d02614f8895ac1d4eac
C	: 42831ec2217774244b7221b784d0d49c
	: e3aa212f2c02a4e035c17e2329aca12e
	: 21d514b25466931c7d8f6a5aac84aa05
	: 1ba30b396a0aac973d58e091473f5985
T	: 4d5c2af327cd64a62cf35abd2ba6fab4

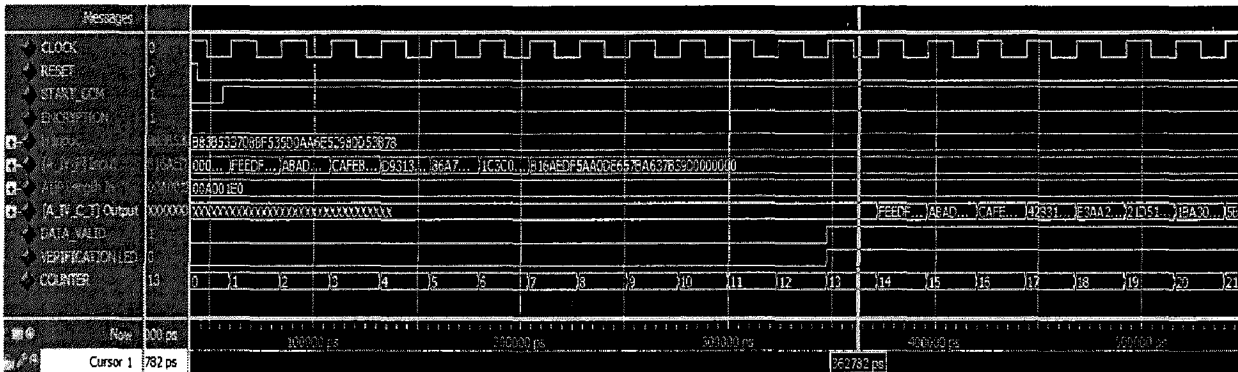
```
GCM Test Case #04 (AES-128)
Variable Value
--------------------------------------------------------------------
K               : feffe9928665731c6d6a8f9467308308
P               : d9313225f88406e5a55909c5aff5269a
                : 86a7a9531534f7da2e4c303d8a318a72
                : 1c3c0c95956809532fcf0e2449a6b525
                : b16aedf5aa0de657ba637b39
A               : feedfacedeadbeeffeedfacedeadbeef
                : abaddad2
IV              : cafebabefacedbaddecaf888
H               : b83b533708bf535d0aa6e52980d53b78
Y_0             : cafebabefacedbaddecaf88800000001
E(K,Y_0)        : 3247184b3c4f69a44dbcd22887bbb418
X_1             : ed56aaf8a72d67049fdb9228edba1322
X_2             : cd47221ccef0554ee4bb044c88150352
Y_1             : cafebabefacedbaddecaf88800000002
E(K,Y_1)        : 9bb22ce7d9f372c1ee2b28722b25f206
Y_2             : cafebabefacedbaddecaf88800000003
E(K,Y_2)        : 650d887c3936533a1b8d4e1ea39d2b5c
Y_3             : cafebabefacedbaddecaf88800000004
E(K,Y_3)        : 3de91827c10e9a4f5240647ee5221f20
Y_4             : cafebabefacedbaddecaf88800000005
E(K,Y_4)        : aac9e6ccc0074ac0873b9ba85d908bd0
X_3             : 54f5e1b2b5a8f9525c23924751a3ca51
X_4             : 324f585c6ffc1359ab371565d6c45f93
X_5             : ca7dd446af4aa70cc3c0cd5abba6aa1c
X_6             : 1590df9b2eb6768289e57d56274c8570
len(A)||len(C)  : 00000000000000a000000000000001e0
GHASH(H,A,C)    : 698e57f70e6ecc7fd9463b7260a9ae5f
C               : 42831ec2217774244b7221b784d0d49c
                : e3aa212f2c02a4e035c17e2329aca12e
                : 21d514b25466931c7d8f6a5aac84aa05
                : 1ba30b396a0aac973d58e091
T               : 5bc94fbc3221a5db94fae95ae7121a47
```

APPENDIX B

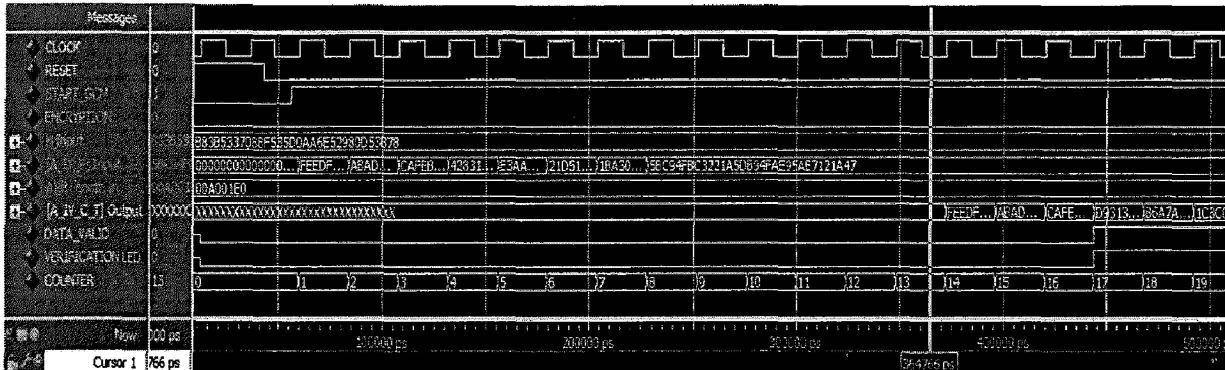
SIMULATION RESULTS OF IMPLEMENTED DESIGNS

1. AES-GCM Designs:-





(a)



(b)

1.1 LUT based AES-GCM with Mastrovito Multiplier

```
Timing summary:
---------------
Timing errors: 0 Score: 0
Constraints cover 369867 paths, 0 nets, and 109174 connections
Design statistics:
Minimum period: 9.613ns (Maximum frequency:     104.026MHz)
Minimum input required time before clock:       10.362ns
Minimum output required time after clock:       10.550ns
Analysis completed Sun May 24 04:26:36 2009
----------------------------------------------------------------------
```

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	3,174	98,304	3%
Number of 4 input LUTs	22,237	98,304	22%
Logic Distribution			
Number of occupied Slices	12,039	49,152	24%
Number of Slices containing only related logic	12,039	12,039	100%
Number of Slices containing unrelated logic	0	12,039	0%
Total Number of 4 input LUTs	22,730	98,304	23%
Number used as logic	22,237		
Number used as a route-thru	77		
Number used as Shift registers	416		
Number of bonded IOBs	40	960	4%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Number of FIFO16/RAMB16s	130	240	54%
Number used as FIFO16s	0		
Number used as RAMB16s	130		
Total equivalent gate count for design	8,735,739		
Additional JTAG gate count for IOBs	1,920		

```
Power summary:                                   I(mA)          P(mW)
----------------------------------------------------------------------
Total estimated power consumption:                               1284
---
Vccint 1.20V:                                    572             687
Vccaux 2.50V:                                    234             585
Vcco25 2.50V:                                    5               13
---
Clocks:                                          51              61
Inputs:                                          3               3
Logic:                                           166             199
Outputs:
Vcco25                                           5               13
Signals:                                         0               0
---
Quiescent Vccint 1.20V:                          353             424
Quiescent Vccaux 2.50V:                          234             584
----------------------------------------------------------------------
```

1.2 LUT based AES-GCM with Karatsuba Multiplier

Timing summary:

Timing errors: 0 Score: 0

Constraints cover 860812 paths, 0 nets, and 50973 connections

Design statistics:

Minimum period: 12.785ns (Maximum frequency: 78.217MHz)

Minimum input required time before clock: 12.573ns

Minimum output required time after clock: 12.305ns

Analysis completed Sat May 23 23:29:32 2009

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	2,788	178,176	1%
Number of 4 input LUTs	12,280	178,176	6%
Logic Distribution			
Number of occupied Slices	7,304	89,088	8%
Number of Slices containing only related logic	7,304	7,304	100%
Number of Slices containing unrelated logic	0	7,304	0%
Total Number of 4 input LUTs	12,773	178,176	7%
Number used as logic	12,280		
Number used as a route-thru	77		
Number used as Shift registers	416		
Number of bonded IOBs	40	960	4%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Number of FIFO16/RAMB16s	160	336	47%
Number used as FIFO16s	0		
Number used as RAMB16s	160		
Total equivalent gate count for design	10,610,177		
Additional JTAG gate count for IOBs	1,920		

Power summary:	**I (mA)**	**P (mW)**
Total estimated power consumption:		1804

Vccint 1.20V:	891	1069
Vccaux 2.50V:	289	722
Vcco25 2.50V:	5	13

Clocks:	49	58
Inputs:	3	3
Logic:	211	253
Outputs:		
Vcco25	5	13
Signals:	0	0

Quiescent Vccint 1.20V:	628	754
Quiescent Vccaux 2.50V:	289	722

1.3 CAF based AES-GCM with Mastrovito Multiplier

```
Timing summary:
---------------
Timing errors: 0 Score: 0
Constraints cover 55489067 paths, 0 nets, and 144085 connections
Design statistics:
Minimum period: 10.227ns (Maximum frequency:        97.780MHz)
Minimum input required time before clock:           12.936ns
Minimum output required time after clock:           9.952ns
Analysis completed Sun May 24 03:32:59 2009
------------------------------------------------------------------
```

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	5,100	98,304	5%
Number of 4 input LUTs	37,587	98,304	38%
Logic Distribution			
Number of occupied Slices	19,957	49,152	40%
Number of Slices containing only related logic	19,957	19,957	100%
Number of Slices containing unrelated logic	0	19,957	0%
Total Number of 4 input LUTs	38,080	98,304	38%
Number used as logic	37,587		
Number used as a route-thru	77		
Number used as Shift registers	416		
Number of bonded IOBs	40	960	4%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Total equivalent gate count for design	299,411		
Additional JTAG gate count for IOBs	1,920		

```
Power summary:                                  I(mA)     P(mW)
------------------------------------------------------------------
Total estimated power consumption:                        1136
---
Vccint 1.20V:                                   449       538
Vccaux 2.50V:                                   234       585
Vcco25 2.50V:                                   5         13
---
Clocks:                                         71        85
Inputs:                                         3         3
Logic:                                          30        36
Outputs:
Vcco25                                          5         13
Signals:                                        0         0
---
Quiescent Vccint 1.20V:                         345       415
Quiescent Vccaux 2.50V:                         234       584
------------------------------------------------------------------
```

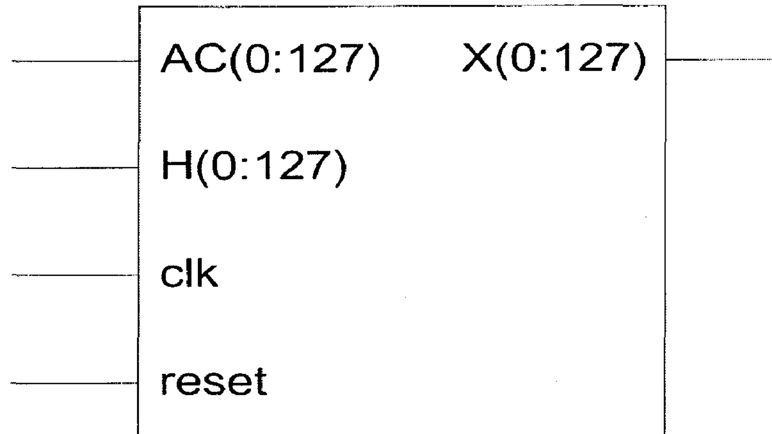
1.4 CAF based AES-GCM with Karatsuba Multiplier

```
Timing summary:
---------------
Timing errors: 0 Score: 0
Constraints cover 54831791 paths, 0 nets, and 144977 connections
Design statistics:
Minimum period: 11.575ns (Maximum frequency: 86.393MHz)
Minimum input required time before clock: 16.767ns
Minimum output required time after clock: 12.759ns
Analysis completed Sun May 24 01:13:31 2009
-----------------------------------------------------------------------
```

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	5,105	178,176	2%
Number of 4 input LUTs	38,190	178,176	21%
Logic Distribution			
Number of occupied Slices	20,320	89,088	22%
Number of Slices containing only related logic	20,320	20,320	100%
Number of Slices containing unrelated logic	0	20,320	0%
Total Number of 4 input LUTs	38,684	178,176	21%
Number used as logic	38,190		
Number used as a route-thru	78		
Number used as Shift registers	416		
Number of bonded IOBs	40	960	4%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Total equivalent gate count for design	301,131		
Additional JTAG gate count for IOBs	1,920		

```
Power summary:                                    I(mA)          P(mW)
-----------------------------------------------------------------------
Total estimated power consumption:                               1594
---
Vccint 1.20V:                                     715            859
Vccaux 2.50V:                                     289            722
Vcco25 2.50V:                                     5              13
---
Clocks:                                           73             88
Inputs:                                           3              3
Logic:                                            30             36
Outputs:
Vcco25                                            5              13
Signals:                                          0              0
---
Quiescent Vccint 1.20V:                           610            732
Quiescent Vccaux 2.50V:                           289            722
-----------------------------------------------------------------------
```

2. Multipliers:-



2.1 Mastrovito Multiplier

```
Timing Summary:
---------------
Speed Grade: -12
Minimum period: No path found
Minimum input arrival time before clock:  No path found
Maximum output required time after clock: No path found
Maximum combinational path delay:         10.260ns
------------------------------------------------------------------
```

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	14,193	98,304	14%
Logic Distribution			
Number of occupied Slices	8,229	49,152	16%
Number of Slices containing only related logic	8,229	8,229	100%
Number of Slices containing unrelated logic	0	8,229	0%
Total Number of 4 input LUTs	14,193	98,304	14%
Number of bonded IOBs	384	960	40%
Total equivalent gate count for design	85,161		
Additional JTAG gate count for IOBs	18,432		

```
Power summary:                                      I(mA)     P(mW)
------------------------------------------------------------------
Total estimated power consumption:                            990
---
Vccint 1.20V:                                       338       406
Vccaux 2.50V:                                       234       584
Vcco25 2.50V:                                       0         0
---
Inputs:                                             0         0
Logic:                                              0         0
Outputs:
Vcco25                                              0         0
Signals:                                            0         0
---
Quiescent Vccint 1.20V:                             338       406
Quiescent Vccaux 2.50V:                             234       584
```

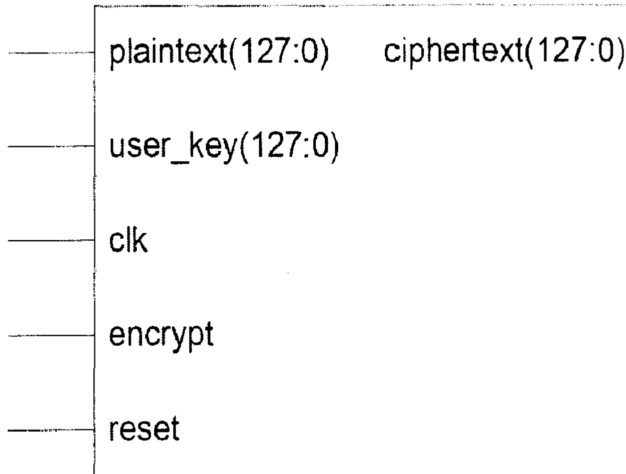
2.2 Kratsuba Multiplier

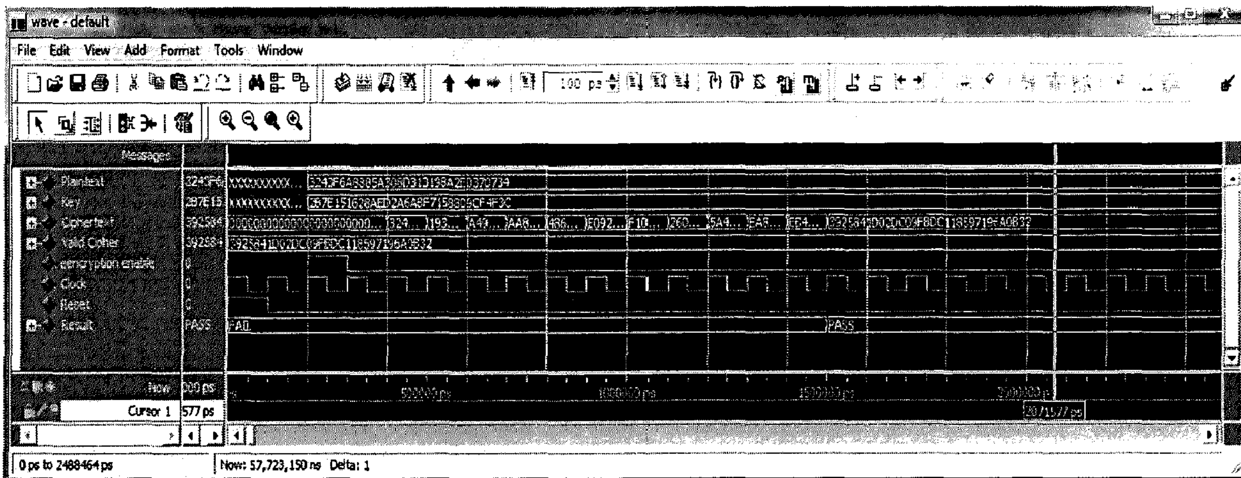
```
Timing Summary:
---------------
Speed Grade: -11
Minimum period: No path found
Minimum input arrival time before clock:  No path found
Maximum output required time after clock: No path found
Maximum combinational path delay:         15.397ns
-----------------------------------------------------------------------
```

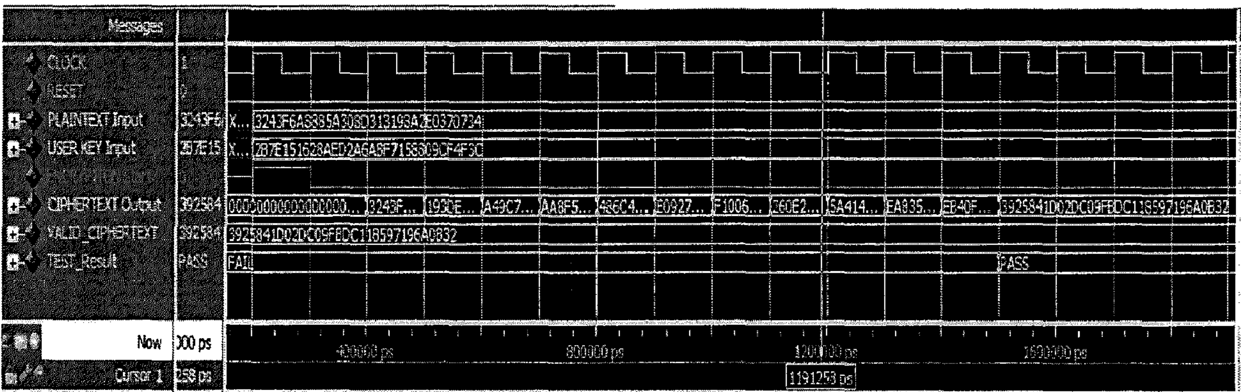
Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	7,542	178,176	4%
Logic Distribution			
Number of occupied Slices	3,890	89,088	4%
Number of Slices containing only related logic	3,890	3,890	100%
Number of Slices containing unrelated logic	0	3,890	0%
Total Number of 4 input LUTs	7,542	178,176	4%
Number of bonded IOBs	384	960	40%
Total equivalent gate count for design	46,290		
Additional JTAG gate count for IOBs	18,432		

```
Power summary:                                   I(mA)          P(mW)
-----------------------------------------------------------------------
Total estimated power consumption:                              1438
---
Vccint 1.20V:                                    597            717
Vccaux 2.50V:                                    289            722
Vcco25 2.50V:                                    0              0
---
Inputs:                                          0              0
Logic:                                           0              0
Outputs:
Vcco25                                           0              0
Signals:                                         0              0
---
Quiescent Vccint 1.20V:                          597            717
Quiescent Vccaux 2.50V:                          289            722
-----------------------------------------------------------------------
```

3. Compact Single Round AES-ECB Design:-







3.1 LUT based Compact Single Round AES-ECB Design

```
Timing summary:
---------------
Timing errors: 0 Score: 0
Constraints cover 728546 paths, 0 nets, and 23466 connections
Design statistics:
Minimum period: 10.161ns (Maximum frequency:    98.416MHz)
Minimum input required time before clock:       0.870ns
Minimum output required time after clock:       13.623ns
Analysis completed Wed Dec 31 12:18:29 2008
--------------------------------------------------------------------------
```

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	625	98,304	1%
Number of 4 input LUTs	4,297	98,304	4%
Logic Distribution			
Number of occupied Slices	2,571	49,152	5%
Number of Slices containing only related logic	2,571	2,571	100%
Number of Slices containing unrelated logic	0	2,571	0%
Total Number of 4 input LUTs	4,297	98,304	4%
Number of bonded IOBs	387	768	50%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Total equivalent gate count for design	39,005		
Additional JTAG gate count for IOBs	18,576		

```
Power summary:                                   I(mA)      P(mW)
--------------------------------------------------------------------------
Total estimated power consumption:                          1019
---
Vccint 1.20V:                                    362        434
Vccaux 2.50V:                                    234        584
Vcco25 2.50V:                                    0          0
---
Clocks:                                          17         21
Inputs:                                          3          3
Logic:                                           0          0
Outputs:
Vcco25                                           0          0
Signals:                                         0          0
---
Quiescent Vccint 1.20V:                          342        410
Quiescent Vccaux 2.50V:                          234        584
--------------------------------------------------------------------------
```

3.2 CFA based Compact Single Round AES-ECB Design

```
Timing summary:
---------------
Timing errors: 0 Score: 0
Constraints cover 3073369 paths, 0 nets, and 11089 connections
Design statistics:
Minimum period: 13.177ns (Maximum frequency:     75.890MHz)
Minimum output required time after clock:        14.706ns
Analysis completed Thu May 21 03:24:36 2009
----------------------------------------------------------------------
```

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	260	178,176	1%
Number of 4 input LUTs	2,819	178,176	1%
Logic Distribution			
Number of occupied Slices	1,452	89,088	1%
Number of Slices containing only related logic	1,452	1,452	100%
Number of Slices containing unrelated logic	0	1,452	0%
Total Number of 4 input LUTs	2,819	178,176	1%
Number of bonded IOBs	387	960	40%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Total equivalent gate count for design	19,306		
Additional JTAG gate count for IOBs	18,576		

```
Power summary:                                   I(mA)       P(mW)
----------------------------------------------------------------------
Total estimated power consumption:                           1438
---
Vccint 1.20V:                                    597         717
Vccaux 2.50V:                                    289         722
Vcco25 2.50V:                                    0           0
---
Clocks:                                          0           0
Inputs:                                          0           0
Logic:                                           0           0
Outputs:
Vcco25                                           0           0
Signals:                                         0           0
---
Quiescent Vccint 1.20V: 597 717
Quiescent Vccaux 2.50V: 289 722
----------------------------------------------------------------------
```

4. High Speed Pipelined AES Design:-

Key(127:0)
Ciphertext(127:0)
Plaintext(127:0)
clk
reset

Current Simulation Time: 11000 ns		10000	10200	10400	10600	10800	11000
clk	1						
reset	0						
load	0						
data_in[127:0]	1...	128'h11111111111111111111111111111111					
key_in[127:0]	1	128'h11111111111111111111111111111111					
done	0						
e_data[127:0]	1...	128'hE5DB5AAB5B8B8E38FJ52A1798E3EB9D					

---xXx---