HIGH THROUGHPUT AES CRYPTO CO-PROCESSOR

Submitted in partial fulfillment of the requirements for the degree of

Master of Science

by
Kassaye Tafesse

Under the Guidance of

Ato Getachew Teshome

Department of Electrical and Computer Engineering

Addis Ababa Institute Of Technology (Addis Ababa University)

Feb, 2013

To
My Parents

Approval Sheet

Thesis entitled : HIGH THROUGHPUT AES CRYPTO CO-PROCESSOR
by : Kassaye Tafesse

is approved for the degree of Master of Science(Microelectronics Eng'g).

Examiners :

Ato Bisrat Derebessa

(Internal Examiner)

Dr. Yalemzewd Negash

(External Examiner)

Supervisor :
Ato Getachew Teshome

Chairman :
Prof. Kwon Ho Yeol

Date :

Place : Addis Ababa

Declaration

I hereby declare that this thesis is my original work, has not been presented for a degree in this and any other college or university to the best of my knowledge and that all sources of materials and researches used for it have been fully acknowledged.

(Kassaye Tafesse)

Dept of Electrical and Computer Engineering
Addis Ababa Institute of Technology
Addis Ababa.
Feb,2013.

Countersigned by

(Ato Getachew Teshome)
Dept of Electrical and Computer Engineering
Addis Ababa Institute of Technology
Addis Ababa.
Feb,2013.

Acknowledgements

This thesis would not have been possible without the guidance and the help of several individuals who contributed and extended their valuable assistance in the preparation and completion of this study.

First,I would like to take this opportunity to extend my deepest gratitude to my advisor,Ato Getachew Teshome, Lecturer at Addis Ababa University for his continuous encouragement and active guidance.Next my gratitude goes to my twin sister Yetinayet Tafesse ,to my best friend Andualem Sileshi and to all my parents and friends who have been there for me in one way or another during the research work.

I also thank Information Network Security Agency,for giving me the opprotunity to study this Masters program.

Special thanks goes to Mr.Alem Hagos and Mr.Solomon Guadie for thier cooperation to create a good enviroment for me during the research work.

On top of all I praise the almighty God, who let this to happen.

Abstract

In 2001, National Institute of Standards and Technology (NIST) approved Rijndael as Advanced Encryption Standard(AES).Since its approval, AES is being used widely in different security applications for its good security, simple design and hardware suitability.AES can be used to encrypt any digital information including videos, images and texts.Since the algorithm involves several iterations, there is still a challenge for high performance implementation.There are some cryptographic applications which demand a higher data throughput implementation of the algorithm.Many researches were done on improving its performance for specific application.In this thesis ,we presented a higher data throughput implementation of AES 128 bit encryption module for FPGA technology than the previous works in the litrature.We started by exploring different available architectures for high throughput design.Then ,FPGA specific features were studied and incorporated in the implementation to improve the throughput of the algorithm.A fully unrolled and pipelined architecture together with pre calculated and stored key values for each rounds is used.We have used VHDL as a hardware description language.The software used for this work is Xilinx ISE design suite 12.3.This tool is used for writing, debugging and also for Synthesis Place and Route. Simulation and checking the performance results were done using the simulation tool ISim Simulator available with the software.From the synthesis result we obtained ,the system runs at 450.532MHz and has a throughput of 57.668 Gbps.

Contents

Certificate i

Declaration ii

Acknowledgements iii

Abstract iv

List of Symbols xi

1 INTRODUCTION 1

1.1 Introduction 1

1.2 Thesis objective 2

1.2.1 Specific Objectives 2

1.3 Methodology 2

1.4 Thesis Outline 3

1.5 Notation and Conventions 4

1.5.1 Bytes 4

1.5.2 Byte Arrays 4

1.5.3 State 4

1.5.4 The State as an Array of Columns 5

1.6 Mathematical Background 6

1.6.1 Addition 6

1.6.2 Multiplication 7
1.6.3 Multiplication by y 7

2 ADVANCED ENCRYPTION STANDARD 9

2.1 Cryptography 9
2.2 Encryption 9
2.3 Rijndael 11
2.3.1 Substitution of Bytes Transformation 12
2.3.2 ShiftRows Transformation 13
2.3.3 Mixing of Columns Transformation 13
2.3.4 Addition of Round Key Transformation 13
2.3.5 Key Expansion Algorithm 14
2.3.6 Round Key selection 16
2.3.7 Summary 16
2.3.8 Modes of Operations of AES 16

3 AES HW Implementation Approaches 19

3.1 HW architectures of symmetric block ciphers 19
3.2 Architectural Optimization 20
3.2.1 Pipelining 20
3.2.2 Sub-Pipelining 20
3.2.3 Loop Unrolling 21
3.3 Algorithmic Optimization 21
3.3.1 SubBytes implementation(S-Box) 22
3.3.2 Implementation of Mix Columns 24
3.3.3 T- Box Approach 24

4 Hardware Design 26

4.1 AES in FPGA 26

4.2 Hardware Overview 26
4.3 Design choices and assumptions 27
4.3.1 Encryption module 28
4.3.2 Modes of Operation 28
4.3.3 Key scheduler 28
4.4 Design Overview 29
4.5 Design Choices 29
4.6 Overall System Architecture 31
4.7 Outer round architecture 31

5 RESULTS AND EVALUATION **33**
5.1 Reference Implementation 33
5.2 Results 34
5.3 Performance Evaluation 35
5.4 Simulation results 37
5.5 Comparisons of Implementation 37

6 CONCLUSION AND RECOMMENDATIONS **38**
6.1 Conclusion 38
6.2 Recommendations 38

7 Appendix **40**

References **52**

List of Figures

2.1 Affine transformation 12

2.2 Byte substitution transformation 13

2.3 Shifting of rows 13

2.4 Mixcolumn transformation 14

2.5 Addition of Round Key transformation 14

2.6 Key Scheduling process 15

2.7 Encryption and decryption in ECB mode 17

2.8 Encryption and decryption in CBC mode 17

2.9 Encryption and decryption in CFB mode 17

2.10 Encryption and decryption in OFB mode 18

2.11 Encryption and decryption in CTR mode 18

3.1 Basic iterative 19

3.2 Partial outer-round pipelining 20

3.3 Full outer-round pipelining 20

3.4 Inner round pipelining 21

3.5 Partial inner and outer-round pipelining 21

3.6 Full inner and outer-round pipelining 21

3.7 Partial loop unrolling 22

3.8 Full loop unrolling 22

3.9 Affine Transformation 22

3.10 Inverse Affine Transformation 23

3.11 Combined architecture of subbyte/Inv subbyte transformation 23
3.12 Mix column transformation with matrix form 24
3.13 Block diagram of XTime 24
3.14 Block diagram for substructure sharing implementation of MC 24
3.15 Block diagram for straight forward implementation of MC transformation . . . 25
3.16 T-Box tables for Encryption 25
3.17 Resultant matrix 25

4.1 AES Algorithm 27
4.2 Overall System Architecture 28
4.3 Inner round module architecture 32

List of Tables

2.1 Shift offsets for different block sizes 13

2.2 Relation of keys and number of rounds 14

3.1 S-Box based on Galois Field $GF(2^8)$ 23

5.1 Resource utilization summary for iterative design 35

5.2 Resource utilization summary for fully unrolled design 35

5.3 Performance analysis of hardware optimizations 36

5.4 Summary of related works 37

List of Symbols

AES Advanced Encryption Standard
ASIC Application Specific Integrated Circuit
AT Affine Transform
CBC Cipher Block Chaining
CFB Cipher Feedback
CLB Configurable Logic Block
CLK Clock
CTR Counter
DES Data Encryption Standard
DSS Digital Signature System
ECB Electronic Codebook
FIP Federal Information Processing
FPGA Field Programmable Gate Array
GF Galio Field
HW Hardware
HDL Hardware Description Language
IV Initialization Vector
LUT Look Up Table
MAC Message Authentication Code
MC Mix Column
NIST National Institute of Standards and Technology
OFB Output Feedback
PKC Public Key Cryptography
PRBG Pseudo Random Bit Generator
PRF Pseudo Random Function
RSA Rivest,Shamir and Adelman
SBOX Substitution Box
SKC Secret Key Cryptography
SW Software

Chapter 1

INTRODUCTION

1.1 Introduction

Information communication has specific security requirements, which includes Authentication: the process of proving one's identity, Privacy/confidentiality: ensuring that no one can read the message except the intended receiver, Integrity: assuring the receiver that the received message has not been altered in any way from the original, Non-repudiation: a mechanism to prove that the sender really sent this message. There are different cryptographic algorithms available to achieve these requirements but most of them are computationally intensive, either deals with huge numbers and complex mathematics or involves several iterations. In general, there are three types of cryptographic schemes that are used to accomplish these goals. Secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions. Secret Key Cryptography (SKC): uses a single key for both encryption and decryption. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption. Hash Functions: Uses a mathematical transformation to irreversibly encrypt information. The Advanced Encryption Standard (AES) is a cryptographic algorithm that was approved by the Federal Information Processing Standard (FIPS) for having the best quality among 15 candidates.Since its acceptance, the AES is being used in different applications to protect digital information. The AES algorithm is in the category of symmetric block cipher. It is used to encrypt and decrypt information. Encryption converts data to an unintelligible form called cipher text while decrypting the cipher text converts the data back into its original form, called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt a digital data. Throughout this paper, the initial unencrypted data is referred to as plaintext whereas, the encrypted data is called cipher text, which will in turn be decrypted into usable plaintext.

1.2 Thesis objective

Always, the demand for higher performance digital processor does not stop. Cryptographic processors are not in exception to this fact. So, there are some security applications, like multimedia communication, which require a higher throughput implementation of the AES algorithm than existing implementations. To this end, the general goal of this thesis is to design hardware architecture for AES crypto co-processor using FPGAs which has a higher data throughput as compared to related works in the literature.

1.2.1 Specific Objectives

1. To study and understand the AES algorithm so that we can implement it in hardware.
2. To design the AES transformation modules at architectural and structural level.
3. To simulate the designed architecture and test functionality of the system
4. To synthesize the hardware on FPGA.
5. To measure the system performance in terms of HW cost (area), and throughput.
6. To compare the system with other standard implementations.

1.3 Methodology

For the realization of this thesis the methodology we followed is as follows. First, we deeply studied the AES algorithm and its previous implementations based on SW as well as HW. We started by exploring different available techniques. Next, the best approach for our optimization was selected by studding different available hardware optimization techniques. Lastly we combined these and other optimization techniques and come up with the improved design. The design was created with HDL description for Xilinx FPGA families and simulation was carried out to make sure that the system functionality is correct. Lastly hardware synthesis of the design was carried out and then evaluated the performance and cost of the system. In general,to achieve the thesis objective, we have followed the top down approach as explained below[9]. The top-down Methodology with its different abstraction levels keeps the complexity of each abstraction level within limits. Using a top-down approach to design a complex digital system will automatically subdivide the overall problem into a number of smaller problems. The decomposed problems are smaller and their solutions added up to solve the overall problem. The levels of abstraction that we follow are ,the system level, the algorithmic level, the architectural level and the register-transfer level.

System Level: The system level defines the functionality of a digital system and boundary conditions under which the circuit should work.

Algorithmic Level: The algorithmic level explores different alternatives to implement desired functionality of a circuit.

Architectural Level: The architectural level of a circuit defines a circuit by its modules and their sub modules. These modules implement functionalities defined in the high-level model.

Register-Transfer Level: The register-transfer level refines the architectural level by defining the modules and sub-modules more precisely. The register-transfer level defines the functionality more precisely by fixing which operations are executed in which clock cycle.

1.4 Thesis Outline

Notations and conventions that are used throughout this thesis and mathematical backgrounds are discussed in chapter 1.Chapter 2 gives an introduction to the Advanced Encryption Standard algorithm and underlying theory. Theories concerning different hardware implementation options for AES algorithm are presented in Chapter 3.Architectural and algorithmic optimization techniques are also presented in this Chapter. The theory part of this thesis will be concluded with Chapter 4 by giving a short introduction to FPGAs. Chapter 4 also discusses the hardware design along with the design choices and assumptions taken. Simulation results and performance evaluation of the design will be given in Chapter 5. The thesis ends with conclusions and suggestions for future work in Chapter 6.

1.5 Notation and Conventions

1.5.1 Bytes

A byte is a unit of data in a computer system which is eight bits long. Each bit in a byte has a value of either zero or one. The AES algorithm uses a byte as a basic processing unit. All the input, output and key are composed of a specified number of bytes called byte arrays. In computer system a byte array is referred as a[i] or ai where i is the index used to refer a single byte from the array and lies between 0 and n-1, where n is the maximum length of the array. In AES cipher key n is the length of the key and i range from 0 to n-1. For example ,for a cipher key of length 128 bits which is 16 bytes ,n lies in the range 0 to 15.Whereas, for key sizes 192 and 256 n lies in the range 0 to 23 and 0 to 31 respectively. AES algorithm represents a byte as a collection of bits by a brace like $(b^7, b^6, b^5, b^4, b^3, b^2, b^1, b^0)$.This notation is used for polynomial representation as a finite field element. For example, a finite field element
$x^5 + x^4 + x^2 + 1$

can be represented as (00110101) in binary notation. Hence the element 00110101 can be represented as 35 in hexadecimal notation.In this notation each four digits are represented by one character. I.e. 0011 is represented by 3 while 5 represents the four bit nibble 0101. Some finite field operations involve one additional bit b^8 to the left of an 8-bit byte. When the b^8 bit is present, it appears as 01 immediately preceding the 8-bit byte. For example, a 9-bit sequence is presented as (01)(1b).

1.5.2 Byte Arrays

A collection of bytes form a byte array. The length of the byte array for the AES input ,output and cipher keys are dependent on the key sizes used. For example, for AES 128 bit the array contains 16 bytes.Hence this array is referred as a0 to a15.For 192 and 256 bits key version of AES the length of the byte array ranges from 0 to 23 and 0 to 31 respectively. This byte array can be represented as $a_n = (a_0, a_1, a_2 a_n)$ where n is the length of the array.

1.5.3 State

The inputs and outputs of the AES algorithm transformations are two dimensional arrays and are called state denoted by S. For AES 128 bit key size ,this state data is a four by four matrix with four rows and four columns of bytes. One row of this state data contains 4 bytes.

Each byte in the state array is referred by two indices .The first index refers the row number and ranges from 0 to 3 while the second index is the column number c and lies in the range 0 to Nb-1. where Nb is the number of blocks. A byte from the state data can be referred as $S_{r,c} or S[r,c]$.The cipher process starts by coping a specified number of bytes to the state array. For the case of AES 128 bits,16 bytes of data are copied to the 4x4 matrix of the state. Any further encryption process uses this state array. After the encryption or decryption process is done, the output data is copied from the state matrix to the output array. This output array will have equal length to the state matrix. At the start of the encryption or decryption, the input array is copied to the State array with

$S_{(r,c)} = in(r + 4c)$

where

$0 <= r <= 3$ and $1 <= c <= Nb$

At the end of the encryption and decryption the State is copied to the output array with

$out(r + 4c) = S_{(r,c)}$

where

$0 <= r <= 3$ and $1 <= c <= Nb$

1.5.4 The State as an Array of Columns

Each column of a state data is made up of a word of four bytes. The four bytes in this column word are referenced by using the row number as an index. In general, for the AES 128 bits the state data contains four words and is denoted as $W_0, W_1, W2, W_3$.A state can be taken as a linear array of four words of column data and the column number c used as an index. These arrays of four words can be denoted as

$W_0 = S_{0,0}S_{1,0}S_{2,0}S_{3,0}$,
$W_1 = S_{0,1}S_{1,1}S_{2,1}S_{3,1}$,
$W_2 = S_{0,2}S_{1,2}S_{2,2}S_{3,2}$
and
$W_3 = S_{0,3}S_{1,3}S_{2,3}S_{3,3}$

1.6 Mathematical Background

Using the previous notations, the AES algorithm defines the byte as a finite field element which can be added and multiplied using the finite field rules. These rules are different from the rules we normally use for numbers. In the following sections we will see the rules of addition and multiplication in the finite field.

1.6.1 Addition

In the finite field the addition of two elements is performed by adding the coefficients of the corresponding powers in the polynomials.XOR operation is used for the addition of two elements in finite fields. Such addition is performed modulo-2. In modulo-2 addition
1 xor 1 = 0 ,
1 xor 0 = 1 ,
0 xor 1 = 1
and
0 xor 0 =0.

For two bytes represented as finite field elements, the addition is performed by modulo 2 additions of corresponding bits. The addition of two bytes can be done as follows

$$(\mathrm{a}^7, a^6, a^5, a^4, a^3, a^2, a^1, a^0) \text{ XOR } (b^7, b^6, b^5, b^4, b^3, b^2, b^1, b^0) = (c^7, c^6, c^5, c^4, c^3, c^2, c^1, c^0), \quad to \tag{1.2}$$

where each
ci = ai xor bi and i represents corresponding bits.

For example, the following expressions are equivalent to one another.
$(x^5 + x^3 + x^2 + x + 1)$ + $(x^6 + x + 1) = x^6 + x^5 + x^3 + x^2$ (Polynomial notation)
(00101111)xor (01000011) = (01101100) (Binary notation)
(2f)xor(43) = (6c) (Hexadecimal notation)

A similar operation is performed while subtracting two polynomials.

1.6.2 Multiplication

In Galiofield $GF(2^8)$,the multiplication of two polynomial representations is computed by multiplying them and taking modulo of the product with degree eight irreducible polynomial. An irreducible polynomial has only one divisor. This irreducible polynomial for the AES algorithm is given as follows[1].

m(x) = $x^8 + x^4 + x^3 + x + 1$
For example, (57).(83) = (c1) because
$(x^6+x^4+x^2+x+1)(x^7+x+1) = x^{13}+x^{11}+x^9+x^8+x^7 + x^7+x^5+x^3+x^2+x+x^6+x^4+x^2+x+1$
$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$
$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ Modulo $(x^8 + x^4 + x^3 + x + 1)$
$= x^7 + x^6 + 1.$

The result of the polynomial multiplications should not exceed degree of 8 and this can be ensured by taking the modular reduction of the result by m(x). Unlike addition, there is no simple operation at the byte level that corresponds to this multiplication. The multiplication of two polynomials associative and the element 01 is the multiplicative identity.

The multiplicative inverse of any non zero polynomial a(x) can be given by $a^{-1}(x)$ and is found by using the the extended Euclidean algorithm. To compute polynomials a(x) and c(x) such that
$b(x) \cdot a(x) + m(x) \cdot c(x) = 1$
Hence, $a(x) \cdot b(x) mod m(x) = 1,$
which means
$b \cdot 1(x) = a(x) mod m(x)$
Moreover, for any a(x), b(x) and c(x) in the field, it holds that
$a(x) \cdot (b(x) + c(x)) = a(x) \cdot b(x) + a(x) \cdot c(x)$

It follows that for the set of 256 possible byte values, with XOR used as addition and multiplication defined as above, has the structure of the finite field $GF(2^8)$.

1.6.3 Multiplication by y

When multiplying a binary polynomial a(x) with the polynomial y ,the result y.a(x) can be computed by using modulo reduction with an irreducible polynomial m(x).If the last bit of the result or a^7 has a value of 1 then modulo reduction of the result is performed by subtracting polynomial m(x) but if a^7 is zero the result is already in reduced form. The multiplication by 2 which is denoted as (00000010) or (02),is performed by left shift and a bit wise XOR

with 1b.This bit wise XOR may or may not be computed depending on the value of the last bit a^7.This operation is called xtime() operation. For multiplying polynomial of higher power of x ,we can use the xtime() operation repeatedly and add the intermediate results[2].

For example,

$(57) \cdot (13) = (fe)$

because

$(57) \cdot (02)$ = xtime$((57)) = (ae)$

$(57) \cdot (04)$= xtime$(ae) = (47)$

$(57) \cdot (08)$ = xtime$(47) = (8e)$

$(57) \cdot (10)$ = xtime$(8e) = (07)$,

Thus,

$(57) \cdot (13) = (57) \cdot ((01) \cdot (02) \cdot (10))$

$=(57) \cdot (ae) \cdot (07)$

$=(fe)$

Chapter 2

ADVANCED ENCRYPTION STANDARD

2.1 Cryptography

The word cryptography is derived from two Greek words kryptos meaning hidden and graphic meaning writing. It deals with the principles and methods of encryption, which plays an important role in the security of data transmission.The process of encryption converts original information, called plaintext, in to transformed information, called cipher text.For this purpose, it uses a key, symmetric or asymmetric .The encrypted data looks like a random combination of characters which makes it difficult to be read by an attacker without the knowledge of the encryption key.So in today's information communication, cryptography has become the main tool to ensure data security.In its modern definition, cryptography is a discipline that embodies principles, means, and methods for the transformation of data in order to hide its information content, prevent its undetected modification and prevent its unauthorized use[27].

2.2 Encryption

Encryption is the process by which an electronic data is converted to an unintelligible form which is difficult to be read by the intruder or attacker.This encrypted data can only be decrypted by using the key.In case of symmetric encryption algorithm this key is the key which was used to encrypt the data .Whereas for the public key algorithms like in RSA, there are two different keys, one for encryption and the other one for decrypting the data. Eventhough these two keys look quite different ,there is a mathematical relationship between them which makes it almost impossible to the attacker to get the content of the message using different key sets. Encryption converts data to an unreadable form called cipher text while decryption of the cipher text converts the data back to its original form, called plaintext. Throughout this paper, the initial unencrypted data is referred to as plaintext ,whereas the encrypted data is called cipher

text, which will in turn be decrypted into usable plaintext.Depending on the type of key used, encryption algorithms are divides in to three types as follows.

1. Unkeyed
2. Secret key
3. Public key

1. Unkeyed algorithms:These types of algorithms do not use a secret key .Examples include One-Way functions, cryptographic hash functions and random bit generators.
2. Secret key algorithms: In case of secret/symmetric key algorithms, one key is used for both encrypting and decrypting the data.For this type of encryption a special key exchange mechanism has to be used between the sender and receiver of the message in order to prevent the key from falling in the hands of unauthorized users.Example of Symmetric encryption systems include Message Authentication Code (MAC),Pseudo Random Bit Generators (PRBG, and Pseudo Random Functions (PRF).Based on the way they process the plaintext data,symmetric ciphers can be classified in to stream ciphers and block ciphers.

   Stream Ciphers Stream ciphers are an important class of symmetric key encryption algorithms.In this scheme of ciphers, each character of the plaintext data is processed separately by using a transformation which varies with time.When implemented in hardware they are faster and consume lesser hardware resources as compared to the block cipher algorithms.They are also more important for some application like communication, where small data is buffered at a time or when individual characters are required to be processed separately.In addition to this stream cipher algorithms are more important for applications where there is high probability of error because they have limited or no error propagation.

   Block Ciphers: In these encryption methods ,a fixed length of data block is processed as a single entity.Symmetric key block ciphers are the most prominent and important elements in many cryptographic systems.The input and output of the encryption algorithm are the same length block data with length denoted by n.When processing a larger plaintext data,the iput data is divided in to blocks of same length and is processed separately block by block.To encrypt messages longer than the block size, the entire plaintext will be broken into blocks and each block will be processed individually in different mode of operation.Block cipher modes of operations will be discussed separatly in the modes of operations section. Block ciphers can individually provide confidentiality as a fundamental building block, their versatility allows construction of pseudo random number generators, stream ciphers, MAC and hash functions.

3. Public key algorithms: In these types of encryption algorithms ,there are two different keys ,called public and private keys.The public key is made public to all intended users and used for encrypting the data while the private key is made private and is only known by the receiver of the data.The private key is used for decrypting the data back to its original form.Since asymmetric key algorithms are less efficient than the symmetric algorithms ,they are mainly used for authentication and key management.Examples of asymmetric key encryption system include, Digital Signature System (DSS), Cryptographic protocol for key agreement.

2.3 Rijndael

Different algorithms were presented in the 1997 National Institute for Standards and Technology (NIST) contest for the new Advanced Encryption Standard (AES). Although many of them showed a similar characteristics, NIST announced that the Rijndael Algorithm was the winner of the contest on October 2^{nd} 2000.The main criteria that made the Rijndael algorithm a winner of the contest were , its strength in security, performance, efficiency, implementation ability and flexibility.This algorithm was developed by Joan Daemen of Proton World International and Vincent Fijmen of Katholieke University at Leuven.The AES (Rijndael) algorithm is suitable for both hardware and software applications.Rijndael is in a category of symmetric block cipher algorithms which means that it converts plaintext blocks to ciphertext blocks and that one key is used for both encryption and decryption.The length of the input and output blocks in AES is 128 bits or 16 bytes.In the process of encryption and decryption ,the algorithm scrambles the data during several rounds of different basic operations.The basic transformation operations in AES algotithm are ,subByte transformation,shiftrows transformation,mixcolumn transformation and addround key transformation.There is also a key scheduler transformation that generates a round key for each round using the key expansion algorithm.The round transformation can be shown by the following pseudocode[1].

```
Round(State,RoundKey)
{
ByteSub(State);
ShiftRow(State);
MixColumn(State);
AddRoundKey(State,RoundKey);

}
LastRound(State,RoundKey)
```

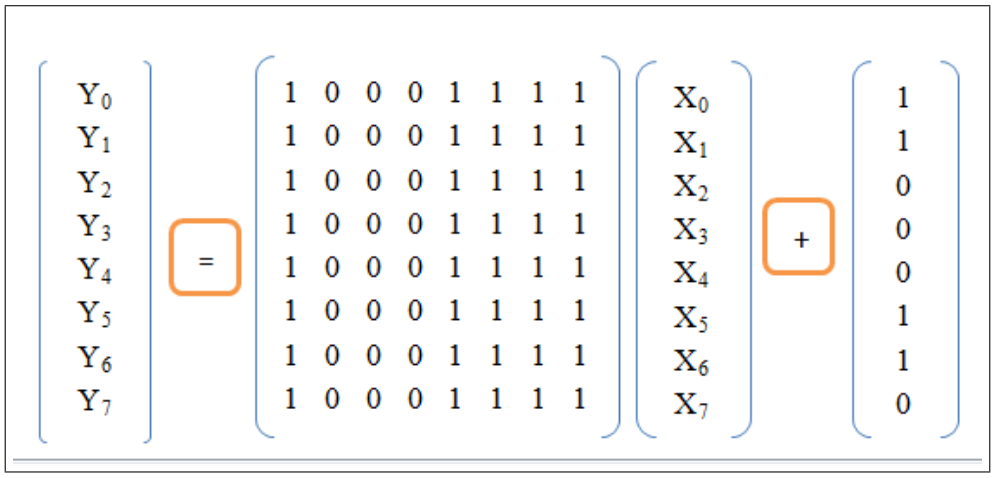

Figure 2.1: Affine transformation

{

ByteSub(State) ;
ShiftRow(State) ;
AddRoundKey(State,RoundKey);

}

The actual encryption process starts with a simple addition of the plaintext with the first round key.Then, Byte substitution, Row shifting, Column mixing and Round key addition are performed for specified number of iteration (e.g. 10 rounds for AES-128 bits).The number of iterations/rounds depends on the length of the key used for the encryption process.The final round is a little different in that ,there is no mixcolumn transformation.Now lets have a close look at each round transformations.

2.3.1 Substitution of Bytes Transformation

In this transformation ,each byte of the state data is substituted by another byte.This substitution process is non linear. The byte substitution is processed as follows

(a) Find the multiplicative inverse in the finite field $GF(2^8)$

(b) compute the affine transformation (over GF (2))

The affine transformation can be expressed in matrix form as shown below in figure 2.1.Figure 2.2 illustrates the byte substitution process. Basically there are two ways of implementing this transformation in hardware.The first option is on the fly calculation where a dedicated hardware module is used for the calculation of the above steps or a substitution box or S-box can be constructed and used as a look up table. So the implementation depends on specific performance requirement.The S-box used in the Sub Bytes

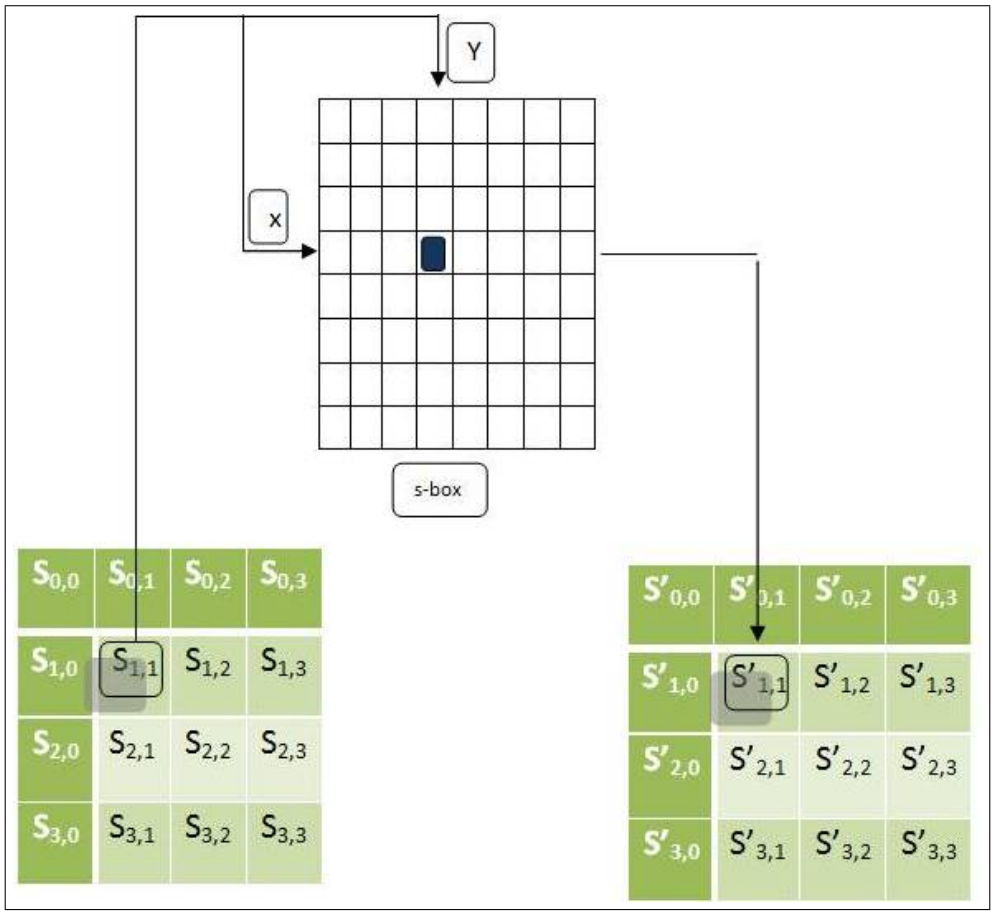

Figure 2.2: Byte substitution transformation

transformation is presented in hexadecimal form. For example, if a byte from the state array has hexadecimal value 64, then the substitution value would be determined by the intersection of the row with index 6 and the column with index 4 in the SBOX table. This would result in hexadecimal value of 43.

2.3.2 ShiftRows Transformation

This transformation processes the state data row wise.The last three rows of the state data are shifted cyclically to the left by a fixed number called offset. From figure 2.3, we can see that the first row will not be shifted.The second row is shifted by one cyclically to the left while the third and the fourth rows are shifted by two and three offset values respectively.Table 2.1 gives the shift offset values.

2.3.3 Mixing of Columns Transformation

This transformation is based on Galois Field multiplication.Unlike the shiftrow transformation,mixcolumn transformation processes the state data column wise.Each column data is considered as a four-term polynomial over $GF(2^8)$ and multiplied modulo $x^4 + 1$

Table 2.1: Shift offsets for different block sizes

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

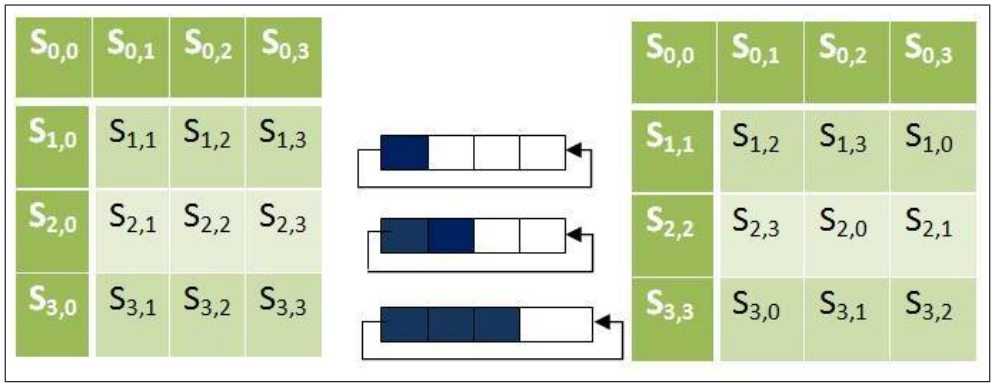

Figure 2.3: Shifting of rows

Table 2.2: Relation of keys and number of rounds

Nr	Nb=4	Nb=6	Nb=8
Nk=4	10	12	14
Nk=6	12	12	14
Nk=8	14	14	14

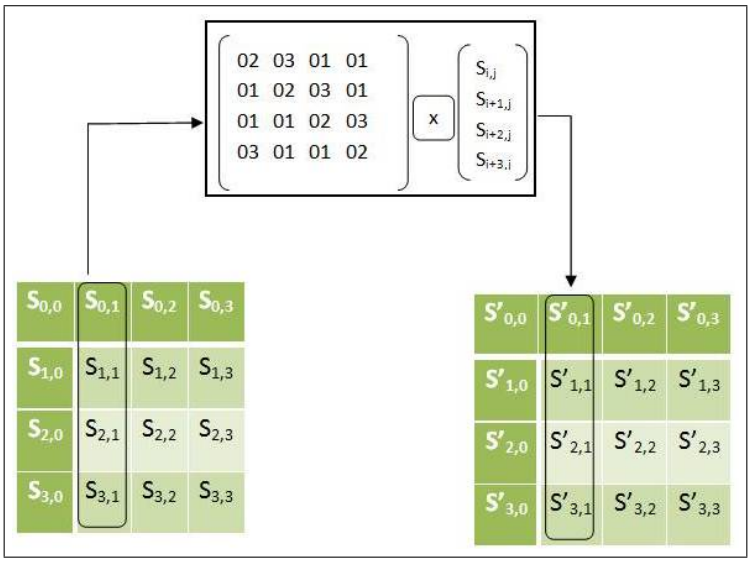

Figure 2.4: Mixcolumn transformation

with a fixed polynomial a(x), represented as follows. $a(x) = 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02$ Every byte of a column data is replaced with another byte that is a function of all four bytes in the given column.The whole mix column transformation can be reduced as a matrix multiplication as shown in figure 2.4.

2.3.4 Addition of Round Key Transformation

AddRoundKey transformation is the process by which a round key is added to the state data.This process is done by using bitwise XOR operation.A round key has Nb*4 bytes where Nb is the number of word in the key data for each rounds.So the output column data from the mixcolumn transformation will be added to these Nb words of the round key buffer. I.e the first column of the state matrix from mixcolumn output will be added with the corresponding column in the round key matrix and the process continues until all the state matix datas are added with the round key.According to the rule specified in the algorithm the first round is only addition of round key.The application of the addRound-Key transformation to the rounds of the encryption occurs for Nr-1 rounds starting from 1.The action of this transformation is illustrated in figure 2.5.

2.3.5 Key Expansion Algorithm

As we have discussed earlier, the addition of round key transformation addes the round key to the state matrix.These round keys are calculated from the original input key by using the key expansion algorithm.This key scheduler module generates round keys sufficient for the whole rounds. Each key has Nb*4 bytes where Nb varies for different key size.The key scheduling process is shown in figure 2.6.In the process of key expansion the first Nk words is the original key while the rest words are obtained by using the expansion

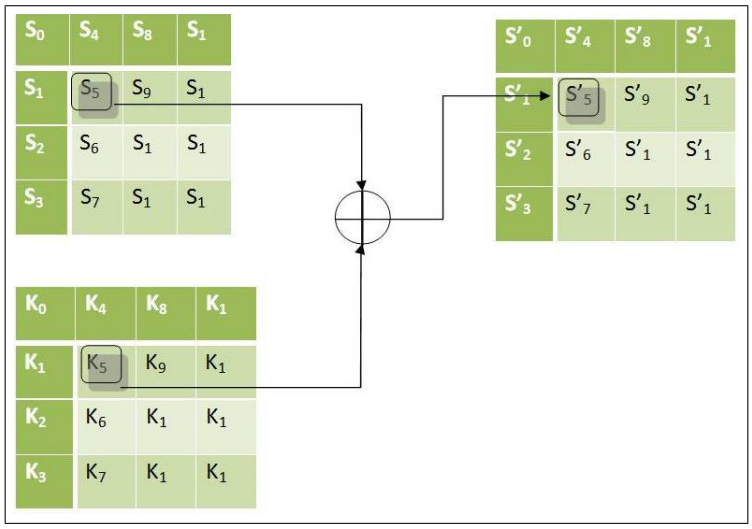

Figure 2.5: Addition of Round Key transformation

algorithm.The key expansion function depends on the value of Nk.There is a version for Nk equal to or below 6

For $Nk <= 6$, we have
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])

{

for($i = 0; i < Nk; i++$)
W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3])
$for(i = Nk; i < Nb * (Nr + 1); i++)$

{

temp = W[i - 1]

if (i % Nk == 0)

temp = SubByte(RotByte(temp)) $^{R}con[i/Nk]$
$W[i] = W[i - Nk]^{t}emp$

}

The function 'g'consists of the following subfunctions.

1. SubWord () This function takes a word data and processes it according to the algorithm and gives a word data as an output.This function is the same as the byte substitution transformation discussed previously which can be implemented by using a dedicated module or table look up approach using SBOXes.

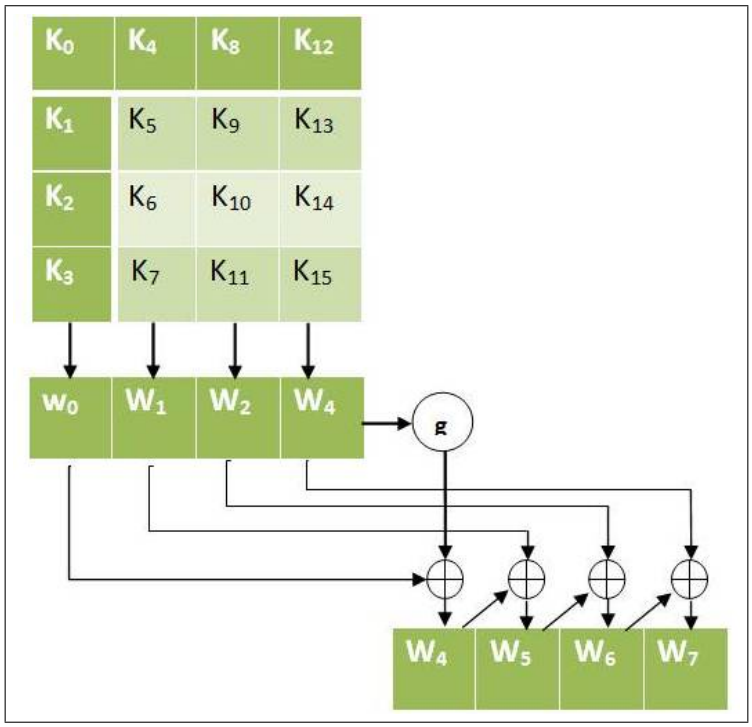

Figure 2.6: Key Scheduling process

2. RotWord ( ) this function also takes a word data and does a cyclic permutation of the four bytes such that the input word (a,b,c,d) gives the output word (b,c,d,a).The first Nk words contain the original Key.Every next word W[k] is equal to the EXOR of the previous word W[i-1] and the word Nk positions earlier W[i-Nk].For words in positions that are a multiple of Nk,a transformation is applied to W[i-1] prior to the XOR and a round constant is XORed.This transformation consists of a cyclic shift of the bytes in a word (RotByte), followed by the byte substitution transformation to all four bytes of the word
For $Nk > 6$, we have
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])

{

for $(i = 0; i < Nk; i++)$
$W[i] = (key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3])$
for $(i = Nk; i < Nb * (Nr + 1)i++)$

{

temp = W[i - 1];

if (i % Nk == 0)

temp = SubByte(RotByte(temp)) $^{R}con[i/Nk]$

temp = SubByte(temp);
W[i] = W[i - Nk] ^{t}emp

}

2.3.6 Round Key selection

Round key i is given by the Round Key buffer words W[Nb*i] to W[Nb*(i+1)].

Schedule Generation Each round key is a 4-word (128-bit) array generated as a product of the previous round key, a constant that changes each round, and a series of S-Box lookups for each 32-bit word of the key.The first round key is the same as the original key.Each byte (w0 - w3) of initial key is XORed with a constant that depends on the current round, and the result of the S-Box lookup for wi, to form the next round key. The number of rounds required for three different key lengths is presented in Table 2.2. The Key schedule Expansion generates a total of Nb(Nr + 1) words.The algorithm requires an initial set of Nb words, and each of the Nr rounds requires Nb words of key data.The resulting key schedule consists of a linear array of 4-byte words, denoted [wi], with i in the range $0 <= i < Nb(Nr + 1)$ [1].

2.3.7 Summary

Rijndael is a symmetric block cipher of variable key and block size.Since it has won the advanced encryption standard compilation by NIST, it is called as AES.The standard Rijndael has 128 bit block size and 128, 192 and 256 bit key.The AES encryption algorithm has four round functions, AddRoundKey(), SubByte(), ShittRow() and Mix Column (Omitted in the last round). The decryption also has the same number of rounds with reverse transformation, and order of round function is slightly different.i.e.InvShiftRow ( ), InvSubBtte( ), AddRoundKey ( ), and InvMixColumn ( ) (Still omitted in the last round).The key expansion algorithm generates 128 bit key for each round and one more key for the initial AddRoundKey ( ) function.The same expanded key is used for both encryption and decryption except the latter reads the round keys in reverse order.Figer summarizes the AES algorithm pictorially.Finally,like other block ciphers AES works in different modes of operations.NIST recommended 5 modes of operations, which can be used depending on the type of application.

2.3.8 Modes of Operations of AES

Generally block cipher algorithms have a weakness of exposing the pattern in the original

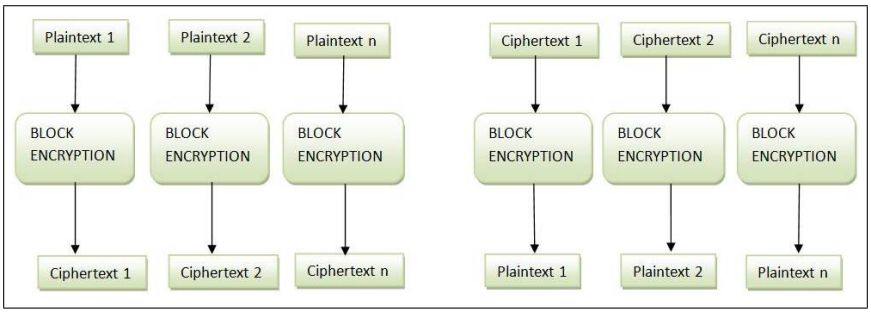

Figure 2.7: Encryption and decryption in ECB mode

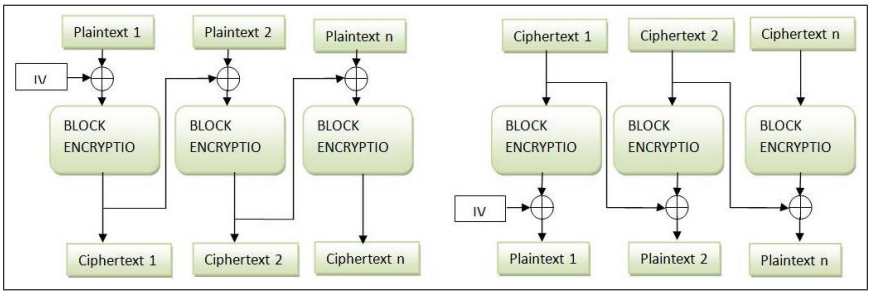

Figure 2.8: Encryption and decryption in CBC mode

input text. I.e. two equal plaintext blocks give similar cipher text blocks.As a remedy to this problem, different modes of operation can be used.NIST recommends five different modes to be used with AES.These are Electronic Code Book, Cipher Block Chaining, Cipher Feedback, Output Feedback and Counter mode of operation.

Electronic Code Book, ECB out of the five modes ECB mode is simple.It ciphers the 16 byte input block data at a time.Implementation will be easy and fewer operations are required when using this mode than the other modes.It is illustrated in figure 2.7.

Cipher Block Chaining, CBC In this cipher mode, the previous cipher text block is XOR'ed with the current plaintext block as shown in figure 2.8.An initialization vector is used in the first round for XOR'ing with the plaintext.In CBC decryption mode, to get the plaintext, the output from the block Cipher decryption has to be XOR'ed with the previous cipher text.

Cipher Feedback, CFB This mode converts the AES block cipher into a stream cipher.I.e. the cipher text is obtained by combining the plaintext with a pseudorandom string using an XOR operation, which is the case in CFB. XOR'ing the output cipher text with the same string produces the input plaintext.Figure 2.9 shows how the AES block cipher is used in CFB mode. Notice that block cipher encryption is used both in encryption and decryption.

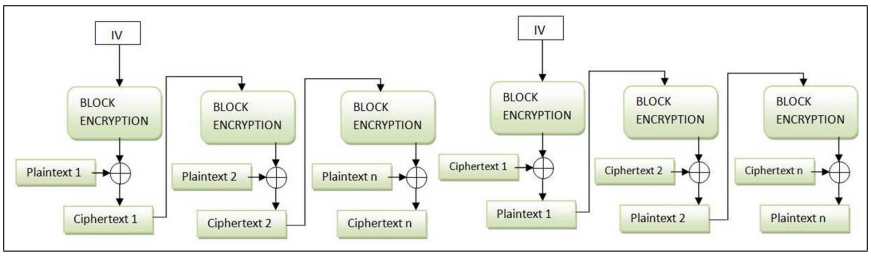

Figure 2.9: Encryption and decryption in CFB mode

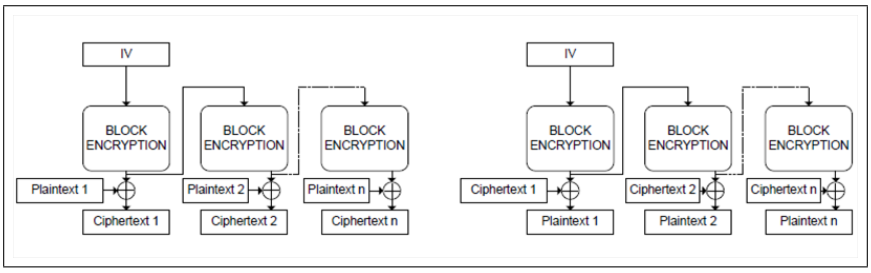

Figure 2.10: Encryption and decryption in OFB mode

Output Feedback, OFB OFB like CFB, OFB converts AES into stream cipher.The difference from CFB is that input to an encryption is the output from the previous block encryption, not the previous cipher text.Figure 2.10 shows how OFB is performed.In this mode also, only block cipher encryption is used.

Counter, CTR This mode uses a counter to produce the cipher stream.Encryption or decryption is performed by XOR'ing the plaintext/cipher text with the cipher stream.Figure 2.11 shows AES in CTR mode.Like in CFB- and OFB mode, CTR mode only used for block cipher encryption.

In the modes CBC, CFB and OFB, initialization vectors, IV, are used as input.The initialization vectors are 128 bit vectors which can be computed using different strategies.One strategy recommended is ,to apply the forward block cipher to a nonce(a data vector not expected to recur) using the same key as used in encryption.The nonce should be a unique data

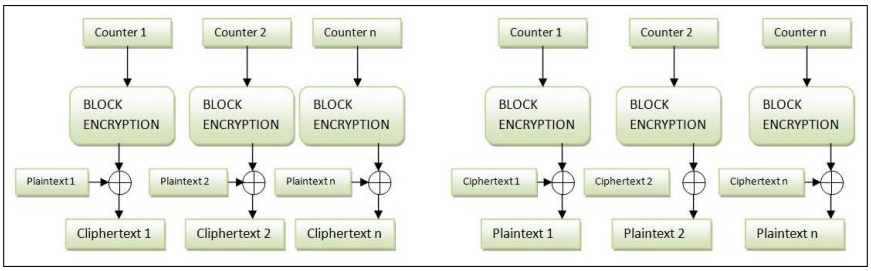

Figure 2.11: Encryption and decryption in CTR mode

block for each execution of the encryption process.Except ECB ,the other four modes described in the preceding sections ensures that two equal plaintext blocks does not produce the same ciphertext blocks.This prohibits the ciphertext from revealing patterns in the plaintext.

Chapter 3

AES HW Implementation Approaches

Generally there are two optimization techniques for the implementation of AES.These are Architectural optimization and Algorithmic optimization. Algorithmic optimization deals with inner round strength, where as architectural optimization deals with loop unrolling ,pipelining and sub-pipelining.We will see architectural optimization techniques in detail but now let us have a look at some symmetric block ciphers hardware architectures.

3.1 HW architectures of symmetric block ciphers

As we have discussed in chapter two of this thesis, there are five modes AES encryption algorithm.In general these modes of operations are divided as Feedback and Non-Feedback modes of operations.

1. Non-feedback modes, such as Electronic Code Book mode (ECB) and counter mode (CTR).
2. Feedback modes, such as Cipher Block Chaining mode (CBC), Cipher Feedback Mode (CFB), and Output Feedback Mode (OFB).

In feedback mode of operation the encryption process of the current data block is dependent on the previous data block.Because of this data dependency, parallel encryption is not possible .So all blocks must be encrypted sequentially.For non feedback mode of operations ,there is no data dependency between the previous and the next data blocks. Because of this feature, the non feedback modes are suitable for hardware implementation since they allows different optimization techniques.The feedback modes are more secure than the non feedback but have less speed because they does not allow parallel processing[24].

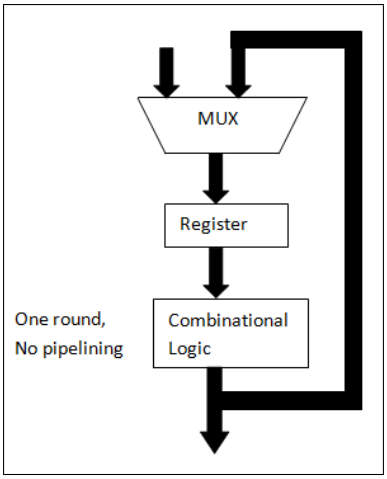

Figure 3.1: Basic iterative

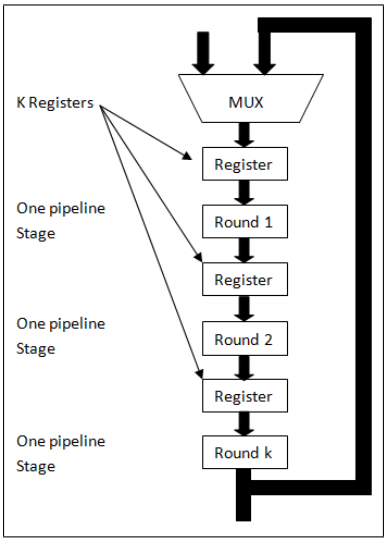

Figure 3.2: Partial outer-round pipelining

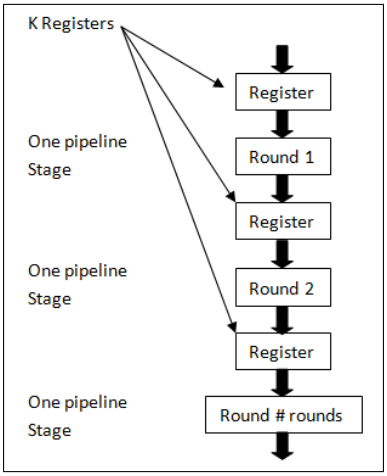

Figure 3.3: Full outer-round pipelining

3.2 Architectural Optimization

Hardware implementation of AES or in general block cipher algorithms can make use of different architectural optimization techniques for implementing each round. Among these Pipelining, Sub-pipelining and loop unrolling are discussed in detail in this section.

3.2.1 Pipelining

Pipelining is one of architectural optimization techniques that allow simultaneous execution of different data blocks which increases the speed of cryptographic algorithms.In the following figure 3.1,we can see the basic architecture is constructed from combinational logics and registers inside them. This architecture can only process one data block at a time where as in the partial outer pipelined architecture ,a number of rounds are unrolled and implemented by a dedicated hardware module.The number of unrolled loops is dependent on the available hardware resources and the intended speedup.If all rounds are unrolled and implemented separately, then the architecture is called fully outer-round pipelined architecture.This architecture is shown in figure 3.3.In fully pipelined architecture a data block can move to the next stage leaving its current stage for the next data block .Hence, after some clock cycles, equal to the number of round stages, all round modules can process data blocks in parallel.

3.2.2 Sub-Pipelining

Pipelining is also possible inside each round modules.This can be achieved by inserting as many pipeline register as required within a round.This architecture is shown in figure 3.4.This internally pipelined module can be repeated for a number of times less than or equal to the total rounds.The choice of the number of round module which run in parallel is basically dependent on the available hardware resources or the required throughput.So to achieve maximum throughput all round can be implemented to run in parallel.This architecture is called fully inner and outer round pipelining and is shown in figure 3.6.If the number of round modules are less than the total rounds ,then the architecture is called partial inner and outer pipelining and is shown in figure 3.5.

3.2.3 Loop Unrolling

An architecture with partial loop unrolling is shown in figure 3.7.In the partial loop unrolling the combinational part of the circuit implements K rounds of the cipher,instead of a single round.Once again K must be a divisor of the total number of rounds, Nr.The number of clock cycles necessary to encrypt a single block of data decreases by a factor of K.At the same time the minimum clock period increases by a factor slightly smaller than K,leading to

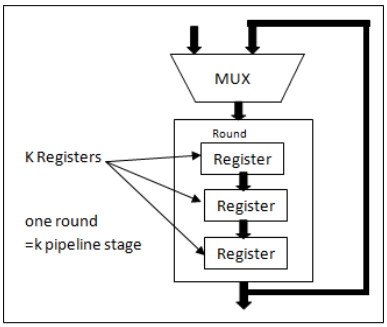

Figure 3.4: Inner round pipelining

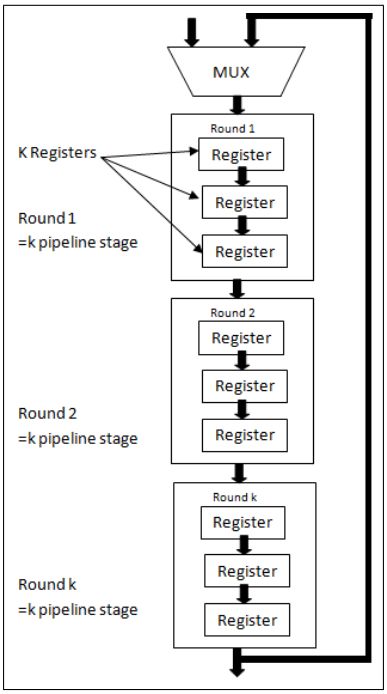

Figure 3.5: Partial inner and outer-round pipelining

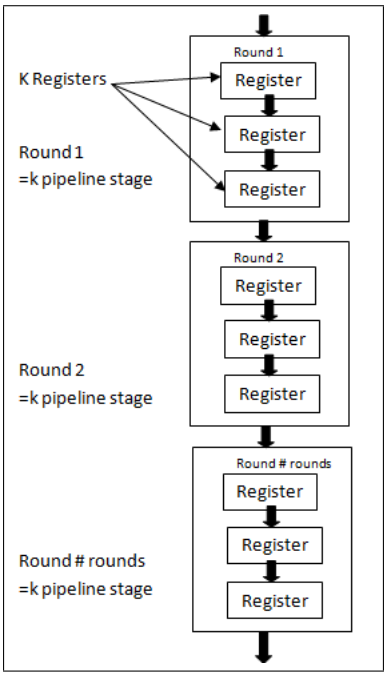

Figure 3.6: Full inner and outer-round pipelining

an overall relatively small increase in the encryption throughput,and decrease in the encryption latency.The combinational part of the circuit accounts for most of the circuit area.Consequently the total area of the encryption or decryption unit increases proportionally to unrolled rounds K.Architecture with full loop unrolling is shown in figure 3.8.The input multiplexer and the feedback loop become unnecessary, causing a small increase in the cipher speed and decrease in the circuit area compared to the partial loop unrolling.In summary, loop unrolling enables increasing the circuit speed in both feedback and non-feedback operating modes.The increase however is minimal and benefits limited since it comes with a large increase in area.As a result, choice of this architecture is best suited only for feedback cipher modes,where no other architecture offers speed greater than the basic iterative architecture,and only for implementations where large increase in the circuit area can be tolerated.

3.3 Algorithmic Optimization

Optimization is also possible within the round ,which is called algorithmic optimization.As we have discussed in chapter two of this paper,AES has four round transformations.These are Shift Rows, Add Round Key, Sub-Bytes and Mix columns transformations.In the following sessions we will discuss the available options for implementing these transformations.

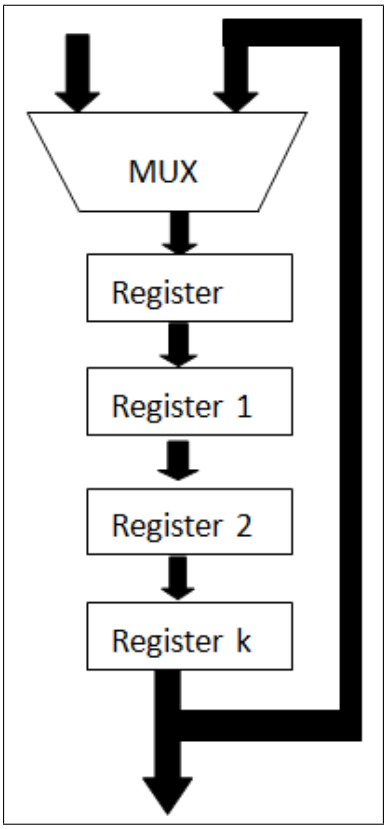

Figure 3.7: Partial loop unrolling

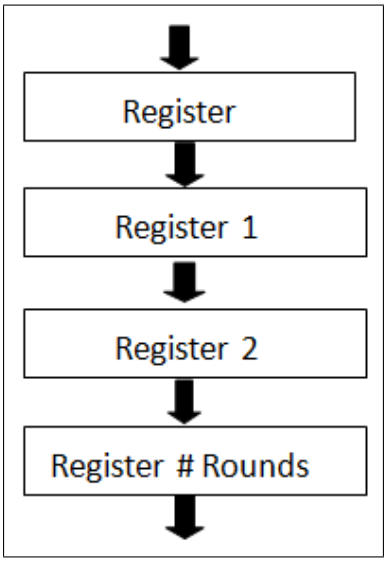

Figure 3.8: Full loop unrolling

Table 3.1: S-Box based on Galois Field $GF(2^8)$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1 8	9	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

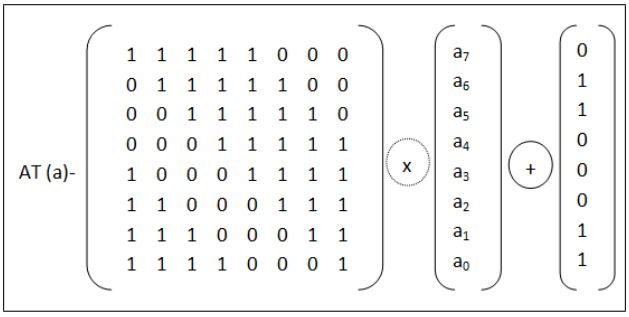

Figure 3.9: Affine Transformation

3.3.1 SubBytes implementation(S-Box)

Subbytes transformation can be implemented by using on the fly calculations or by using SBOXes.Generally the implementation is dependent on the required speedup or hardware cost.Usually the look up table based implementation is preferred.So each byte of the state data is substituted by the corresponding byte from the SBOX.The SBOXes are constructed according to the algorithm.I.e. involve computation of multiplicative inverse of each element in GF, followed by an affine transformation shown in figure 3.9 where AT is the affine transformation while the vector 'a' is the multiplicative inverse of the input byte from the state array.This SBOX contains the substitution values of all possible bytes. I.e. any byte ranges from 00 to ff .This means one SBOX contains 256 possible values.Table 3.1 gives S-Box values.

S^{-1} **- Box** The inverse of byte substitution transformation is called inverse subByte transformation and is used in decryption.Just like the subByte, we can make inverse SBOX tables.The inverse SBOX table is constructed by using inverse of the affine transformation followed by calculation of multiplicative inverse in GF and in figure 3.10 Where $AT^{-1}$is the affine transfor-

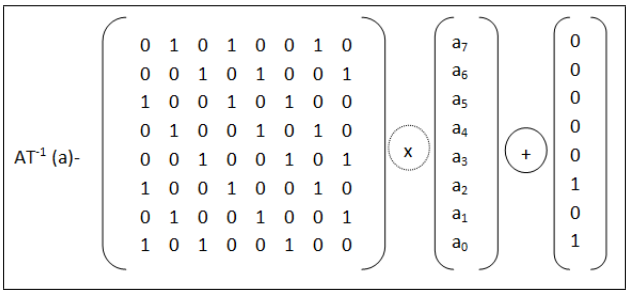

Figure 3.10: Inverse Affine Transformation

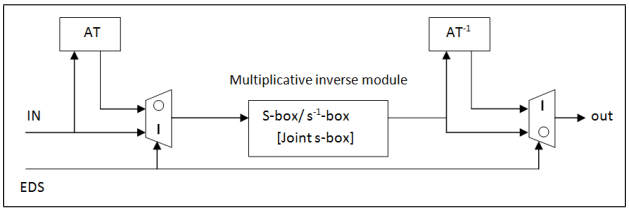

Figure 3.11: Combined architecture of subbyte/Inv subbyte transformation

mation while the vector 'a' is the multiplicative inverse of the input byte from the state array.

Joint S - Box For architectures which implements both encryption and decryption modules as a single hardware,resource sharing is possible between the subByte and inverse subByte modules since both modules implement the same multiplicative inverse module.This architecture is called combined architecture and is shown in figure 3.11.In this combined architecture encryption and decryption can be performed by proper signal selection of Subbyte or Inverse subbyte.

S-Box Construction Methodology using composite field arithmetic For resource constrained applications ,AES SBOX can be computed by mapping the arithmetic operations on $GF(2^8)$ to isomorphic field $GF((2^4)^2)$.Eventhough this architecture has longer delay ,the SBOX construction will need less hardware resources.In a $GF(2^8)$ element, each bit can be considered to be the coefficient to corresponding power term in $GF(2^8)$ polynomial. bx+c represents any arbitrary polynomial ,given an irreducible polynomial $x^2 + Ax + B$, b referring to most significant nibble while c stands for the least significant nibble.

3.3.2 Implementation of Mix Columns

Mix column (MC) The Mix Columns (MC) transformations operate column-by-column on the 4*4 state arrays. The matrix expression of the MC on the Cth column can be expressed as

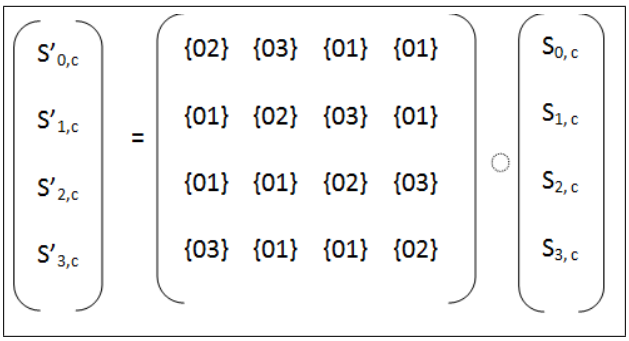

Figure 3.12: Mix column transformation with matrix form

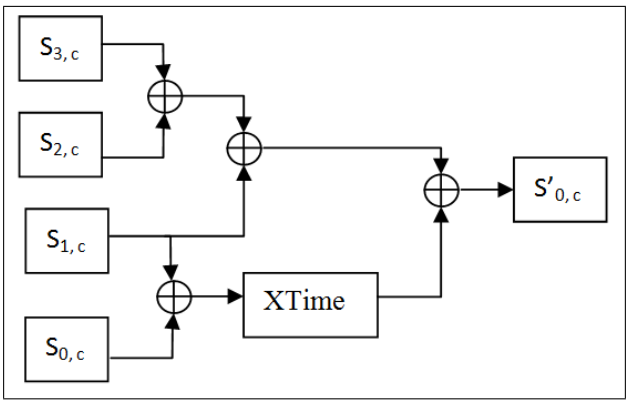

Figure 3.13: Block diagram of XTime

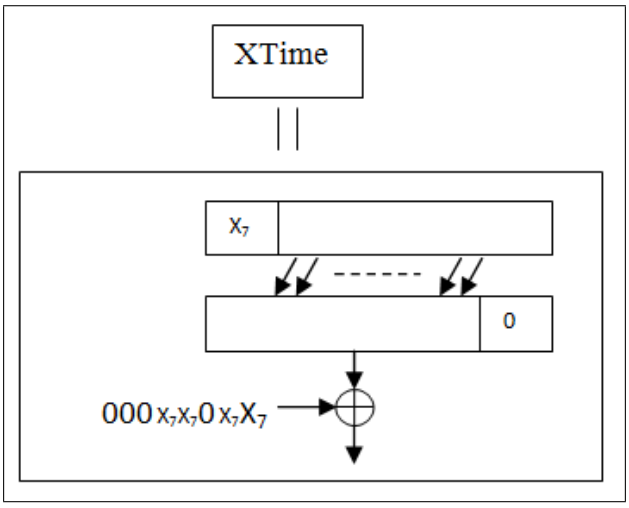

Figure 3.14: Block diagram for substructure sharing implementation of MC

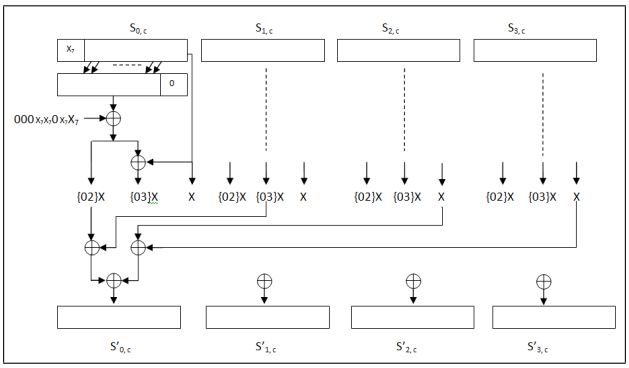

Figure 3.15: Block diagram for straight forward implementation of MC transformation

In the Mix Columns transformation, which is implemented by constant multiplication of 02 and 03 in GF. Suppose X is a byte in the State,(02)X can be implemented operations, and 03X can be computed by (02X) If X is expressed in binary form as
$(x^7, x^6, x^5, x^4, x^3, x^2, x^1, x^0)$,
(02)X can be calculated by
(02)X = $(x^7, x^6, x^5, x^4, x^3, x^2, x^1).02$

3.3.3 T- Box Approach

T-Box The subbytes, shiftrows and mixcolumns transformations are combined and can be implemented as a whole.This is referred as T-box approach .Starting from state array after the MixColumns transformation can be expressed as shown in figure 3.16, for $0 < c < Nb$

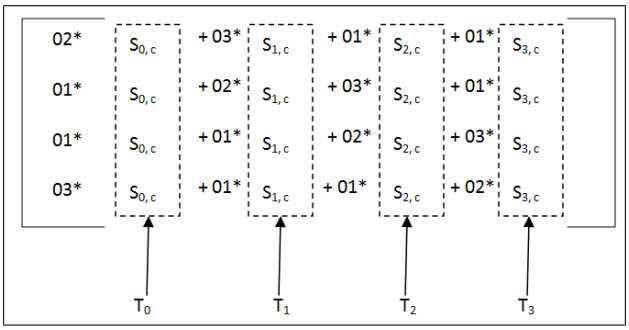

Figure 3.16: T-Box tables for Encryption

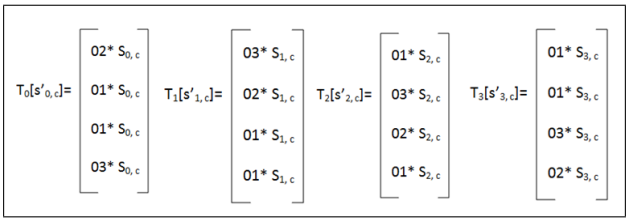

Figure 3.17: Resultant matrix

T-Box Tables for Encryption: The resultant matrix shown in figure 3.17 has each column with fixed constants multiplied by state inputs[24].These state inputs are updated with the next row elements for the next iteration.That is, first row elements of the state inputs are always multiplied with the first column constants of the multiplication matrix.Similarly,the second row elements of the state inputs are always multiplied with the second column constants of the multiplication matrix and so on. The first row of the state input uses table T0,the second row uses the table T1,the third row uses the table T2 and fourth row uses the table T3.Finally, the combination of subbytes, shiftrows and mixcolumns transformations can be implemented by XORing the output's of T-boxes.In the last round of encryption, there is no mixcolumn transformation .So S-box is used instead of T-box.

Chapter 4

Hardware Design

4.1 AES in FPGA

These days FPGAs are getting more and more attention because of their ability to be programmed by the customers or designers after manufacturing.By changing the function of each CLBs and connections among them using reconfiguration ,a new circuit with new functionality can be achieved.FPGAs are more suitable for implementation of cryptographic algorithms.Compared to ASIC,FPGAs provide the following additional advantages.

1. For device prototyping they take relatively shorter design cycle.
2. The design tools are relatively cheaper.
3. When large numbers of architectures are to be tested FPGAs are better options.
4. Higher accuracy of comparison: in the absence of the physical design and fabrication, ASIC designs are compared based on inaccurate pre-layout simulations.
5. FPGA designs are compared based on very accurate post-layout simulations and experimental testing.

From several FPGA families available in the market, in this paper we have chosen vertex 6 family from Xilinx, Inc. for implementing AES algorithm.

4.2 Hardware Overview

Nowadays FPGAs are getting more and more attention than their competitors.One of the reasons is their ability to be re-programmed.FPGAs have a wide range of applications in digital electronics.These include digital signal processing, software defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection

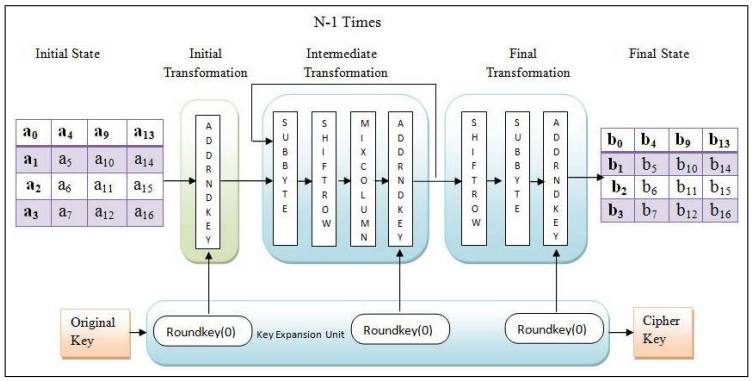

Figure 4.1: AES Algorithm

and a growing range of other areas.In this thesis we have designed AES encryption module for FPGA implementation and improved the design to be suitable for those applications where the throughput of execution is required to be very high.To achieve high data throughput implementation it is obvious that the hardware resource utilization will be larger than the basic implementation.But as we are targeting those applications which don't have hardware restriction,the extra increase in resource utilization can be tolerated.Applications such as multimedia communication require high level of security and at the same time high data throughput. So this can be one of the areas where our design can be applicable.As we have clearly stated in the objective of this thesis work, our target is to design the co-processor assuming the data that is needed to be encrypted will come from the main processor and the encrypted data will be sent to main processor. Figure 4.2 shows the overall architecture of the system and Figure 4.3 presents the inner round module architecture. From the figure 4.2 ,the input line takes 128 bits plaintext data to be encrypted and the cipher data is put on the output data line. Reset and clock signals are added for controlling of the system operation.Since we have stored the pre-calculated values of the key required for the whole round ,we do not need a dedicated key scheduler module.In the following sections we will present the detailed explanation of the system including the assumptions and design choices considered in the design.

4.3 Design choices and assumptions

Generally, designing optimum AES Architecture depends on the following main factors

1. AES can be designed for specific parameter optimization.These include minimum area, minimum power consumption, maximum throughput, maximum throughput to area ratio, etc.
2. As we have seen earlier in chapter two,AES supports five modes of operations which can be broadly classified as Feedback and Non feedback modes. So the implementation can support either of the two modes or both.
3. The AES algorithm has encryption and decryption part which can be implemented sepa-

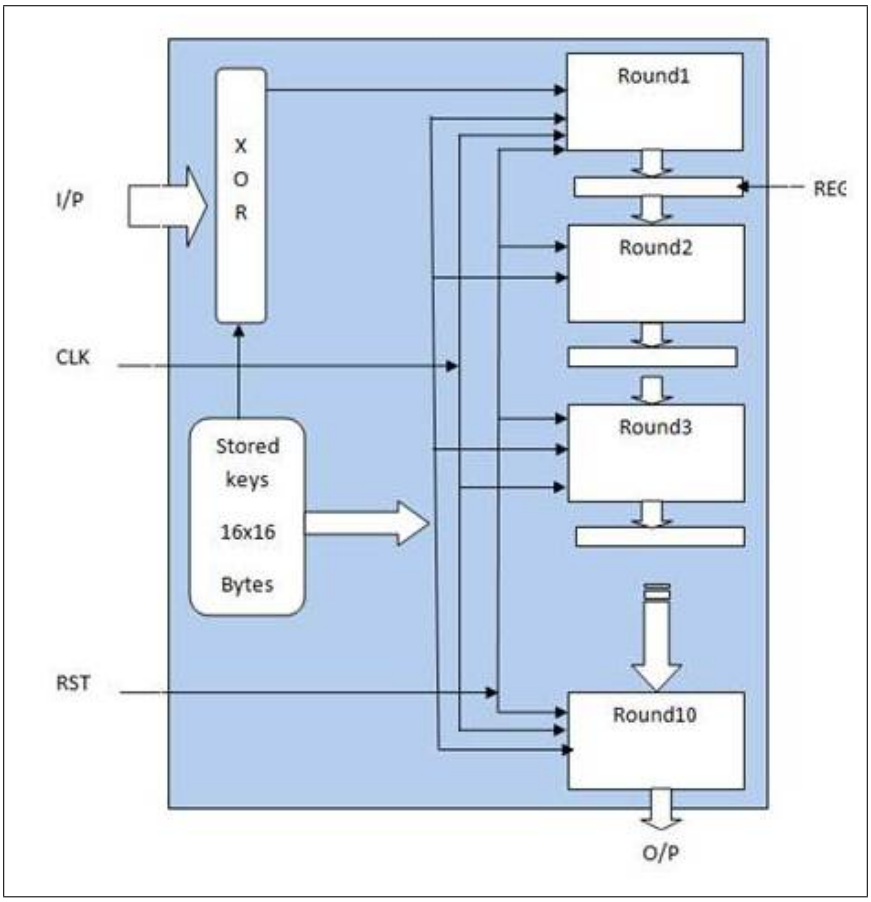

Figure 4.2: Overall System Architecture

rately or can be designed as a single module.So the implementation is dependent on the performance requirement and the available resource.

4. There are also different technologies to implement cryptographic algorithm in hardware such as such as ASIC or FPGA.
5. AES can be designed to be resistant to different attacks like Differential Power Analysis, Timing Analysis.

Since the choice of optimization criteria and the assumption taken into consideration can greatly affect the result of the Hardware Design, we would like to discuss the design choices and assumption that are taken in this thesis briefly.

4.3.1 Encryption module

In this thesis we only targeted the encryption module.

4.3.2 Modes of Operation

As we have discussed in the second chapter,AES can be used in feedback and non-feedback modes of operation.Since the feedback modes do not give us much flexibility to use the available hardware optimization techniques,in our design we have decided to use the non-feedback mode.Actually this implementation can also be used in feedback mode of operation.But the optimization advantages which we are talking can only be achieved when the algorithm is used in non-feedback mode of operation.

4.3.3 Key scheduler

The key scheduler module can be implemented in different ways.One option is to use on the fly calculation which is not hardware friendly.The other option is using pre calculated values of key in the round transformation.The choice of key scheduling module implementation depends on the application which we are targeting.If the key does not change too often, we can use pre-calculated key values implemented as a look up table.So in this thesis work we have selected the second option .I.e. we have stored the key values in a key table and used for encryption during the process. Whenever a key change is required, the mode of the encryption module will be changed to key load and the calculated values of key will be loaded to the system.For this purpose we did not dedicated 128 bits input ports .Rather we set a mode to the module to handle this.Normal mode and Key load mode.In the normal mode, the module encrypts the data using the stored key, while in the key load mode, a pre calculated values of key will be loaded from external hardware or the main processor to the system using the same data lines that were used to accept the plaintext data.

4.4 Design Overview

In the following sections, we will discuss the methods used to increase the throughput of AES hardware implementation. Since we have chosen the non-feedback mode of operation, we have more flexibilities to improve the throughput in the cost of resources. Here are the techniques we used to achieve our goal .As we have seen in Chapter 2 of this thesis, the role of the shift rows module is to shift the rows of the state data with an offset values of 0,1,2 and 3 starting from the first row to the fourth row respectively. The next transformation module as specified by the algorithm is the sub Bytes module. But practically, the shift rows transformation can be done either after or prior to the sub Bytes transfprmation without any problem. This is because, the content of the state array after performing the two transformations is same in both

cases. At the start of encryption, the input plaintext block is XORed with the original key and put in the state array. Then, these state data bytes are send to the shift rows module, which shifts the bytes according to the rule and put them back to the state array. But this can be achieved by connecting the state arrays registers to the appropriate data lines of the input without the need of the shifter module. So with this method, we end up in three stage pipelined architecture of the AES round module .The three sub modules of round function can be executed in one cycle but we preferred them to execute in three cycles by inserting three pipeline registers between each modules. I.e. when the sub byte sub module finishes the output will be stored in the state array registers and another data substitution continues. So, basically after three clock cycles all the three sub modules can run in parallel which in turn increase the throughput of the implementation. The mix column operation is implemented by using shift operation and comparison operations. We have shared the 128 input line for accepting keys and the input plaintext by adding a mode of operation to the implementation. In this design, all cipher keys required for the whole round are calculated in advance and stored in the memory .The key memory will be referred when processing the addRoundkey transformation for each round. To achieve a higher data throughput all ten rounds of AES 128 bits are unrolled and implemented separately with a dedicated hardware module. To allow parallel processing of these round modules, we have inserted 128 bits pipeline register between each round modules. Generally, the architecture we followed in this design is a fully unrolled and pipelined architecture.

4.5 Design Choices

In general we have followed a fully unrolled and pipelined design approach where all the round transformations are implemented as a separate module to increase the data throughput of the implementation.In the objective of this thesis work it has been stated that our aim is to optimize the throughput of AES implementation.For this purpose we have selected the fully pipelined design approaches .i.e. the inner round of AES transformations are implemented as a separate modules.

SubBytes Module For the purpose of high data through put we have decided to implement the AES subbyte module using the sbox approaches.I.e. the pre calculated values of all possible sub byte transformations are store in the ROM and each byte is substituted from these values.Since we have 128 bit or 16 byte data to be substituted we used 16 instances of s-box for parallel substitution of all bytes .So in this method we can substitute 16 byte of data in one clock cycle .

ShiftRows Module The shiftrow module is used to shift the substituted values of the state data with a fixed number depending on the block size.But this module can be merged or this transformation can be done together with the subbyte transformation by reordering the data values to be substituted.By doing so , the length of clock cycle will increase but we will achieve in reducing one clock cycle per each round .That means eventually we can achieve 10 clock

cycle improvement .Hence, this will increases the throughput of execution.

MixColumn Module Regarding this transformation function we used on the fly calculation by using shift and comparission operations. To calculate the mixcolumn values of the 16 state datas we have used 4 instances of the same module.

AddRoundKey Module we implemented this transformation by using XOR operation.

KeyScheduler As we have seen in chapter 2 of this thesis, we have two approaches to implement this module. The first one is on the fly calculation of the key values using the key scheduler module but this approach increases computational overhead to the overall AES module.So we have decided to pre calculate and store the values of the key which is sufficient to all rounds in the ROM.In our case we need 11key values each 128 bit size.To increase the module flexibility to use different key for different data encryption ,we have assumed the key values to calculate in the main processor and send it to the co-processor.For this purpose we do not have a dedicated input line for the key values .Rather we decided to share resources from the input data lines. i.e. we send the key values by using the data input lines byte adding a mode of operation to the design. Byte substitution is implemented by using table look up approache rather than on the fly calculation. In the objective of this thesis work it has been stated that our aim is to improve the throughput of AES implementation. For this purpose we have selected the fully pipelined design approaches i.e. the inner round of AES transformations are implemented as a separate modules. Basically one round module consists of there modules,the subByte,Mixcolumn and the AddRoundKey.

4.6 Overall System Architecture

In general ,for designing our throughput improved hardware architecture ,we have followed the fully unrolled design approach.In this design, all of the AES rounds are unrolled and implemented with a separate hardware. So, for the total of ten rounds in AES 128 cipher we have 10 round modules.Between these round modules we have inserted 9 pipeline registers of size 128 bit each.We have 128 bit length input data lines for the plaintext and the same length output lines for the ciphertext.We have also the reset and CLK signals as the input.Firstly, the plaintext is XORed with the initial key and will be sent to the first round module.Since we have put all the round modules sequentially, each starts execution after the previous module has finished its task.From the diagram we can clearly see that after 10 clock cycles all the round modules will run in parallel, which is the reason why we have higher data throughput.This is achieved only because we have decided to use the non-feedback mode of operation in the very beginning

of this paper and the output of each round module will be given to the the next round module.Finally the cipher output will be put on the output register.The XOR module that we see in the design is required for calculation of the initial round key addition.

4.7 Outer round architecture

Encryption is performed by using ten instances of the AES round transformation module discussed above.As the AES algorithm specifies ,the first and the last modules are a little different in that, in the first round there is only add round key transformation where as in the last case there will not be the mix column modules. So we have decided to implement this rounds differently.As it is clear in figure 4.2, we have inserted 9 pipeline registers of size 128 bit each to enable pipelining in the outer round too.In addition to this, we have stored the pre-calculated values of the key in the register.For this purpose we need 11 register of 128 bit size.By doing so, we have reduced the computational overhead of the system. As it shown in figure 4.3, each round of the AES module has three sub modules, subByte mixColumn,addround key. We have already discussed the exclusion of AES shift row module. Generally to execute one block of data i.e. 128 bit in our case, we need a total of 33 Clock cycles .The length of clock cycle will be discussed in the next chapter. Once the subByte module finishes its computation it will give its output to the mixcolumn module via the pipeline register (128 bit length)and a new state data will be fed to it.By doing so, basically after three clock cycles these three sub modules will run in parallel, this in turn increases the data throughput.The input and output of this sub module are also of length 128 bit.Synchronous clock and reset signals are also given to this module for controlling purposes.

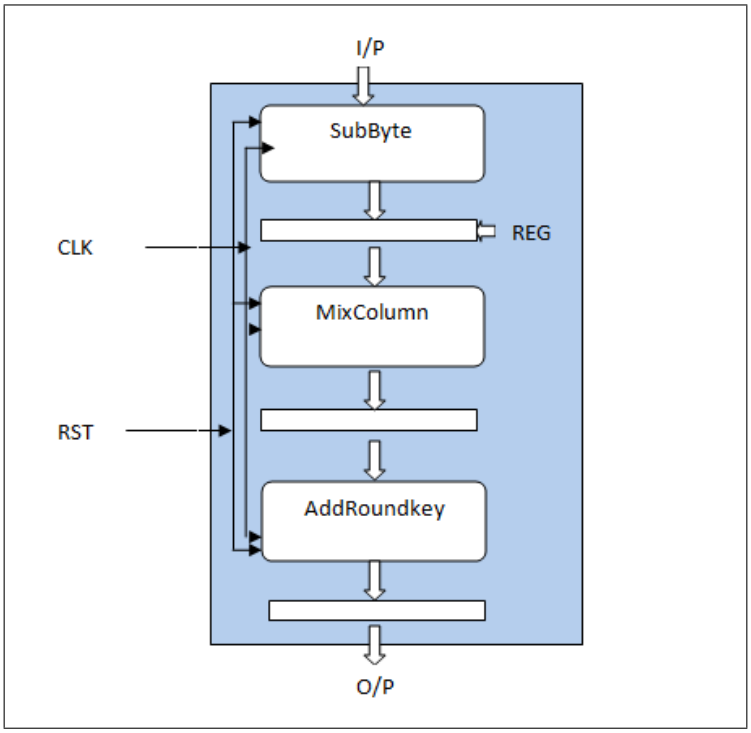

Figure 4.3: Inner round module architecture

Chapter 5

RESULTS AND EVALUATION

5.1 Reference Implementation

We have taken the basic AES architecture as a benchmark to our optimized design.So first we implemented the basic iterative architecture of AES 128 bits for encryption module and then ,we evaluated the performance in terms of throughput and resource utilization.Our final implementation is a fully pipelined and unrolled architecture for larger data throughput for non-feedback modes of operations. The target technology is Virtix6 FPGA. In the following section we will describe the benchmark implementation and our improved implementation. In both cases the AES 128 bits key version is implemented

Iterative design In this design only one round of AES algorithm is implemented and encryption is done by rotating through this round module for a number of times. The number of iteration is dependent on the key size. For the AES 128 bits key size, the algorithm iterates ten times through this round module in order to encrypt one data block. This implementation encrypts one AES 128-bit block of plaintext data at a time. The Input to Output latency for this implementation is 41 clock cycles.This iterative approach allows the computation of one AES round with 4 clock cycles.So for the total of 10 rounds we need 40 cycles .The additional one clock cycle comes from the fact that there is an initial round key addition step as stated in the algorithm. Besides the architecture for the data unit, the key unit implements the key scheduling. Here also one round key is generated per 4 clock cycles. The architecture of the data unit and the key unit are describe in detail in the following sections.

Data Unit The data unit of the AES module contains combinational logic to calculate the AES transformations SubBytes, ShiftRows, MixColumns, AddRoundKey. Sixteen instances of the S-Box circuit are used in this architecture. The MixColumns Operation is implemented

using shift and comparison operations. In this implementation there are four instances of the same mixcolumn module for the 128 byte mixing since each unit mixes 32 bit of data at a time.

Key Unit The key unit performs the key-scheduling algorithm. Starting with the original key as input ,it delivers one round key per 4 clock cycle. In each step of the key schedule ,the first 32-bit word is rotated and the S-Box lookups are applied .Therefore four instances of the S-Box are required. A constant called Rcon is also added to the output of the S-Box circuit with an XOR gate. This result is combined with the last 32-bit word with an XOR. The other three new 32-bit words are calculated from the old values and an XOR operation with the other inputs. A 128 bits register is used to store the round key used in each round.

Fully Unrolled and pipelined implementation The highest data throughput for AES encryption can be reached when all ten rounds are unrolled and pipelining registers are introduced between each round[24] . Because of this unrolled architecture only non-feedback modes of operation are useful. According to the architecture in figure 4.2, it consists of an initial round which is an XOR gate of the original key with the data input followed by ten instances of the round transformation. The last round instance does not include the MixColumns circuit because the operation is not required in the last round. In this implementation, we have excluded the shiftrow module and all the round keys were generated in advance and stored. In this implementation, data blocks are accepted every clock cycle. After an initial delay of 30 clock cycles, the corresponding cipher text blocks appear at the output every clock cycle.For this architecture the input to output latency is 30 clock cycles. This approach allows the computation of one AES round with 3 clock cycles since the shift row module has been elliminated. So generally, for the total of 10 rounds we need 30 cycles.

Implementations of Key Scheduling Round keys can be either generated beforehand and stored in memory or generated on the fly. We used the former approach,where all round keys are read out from memory using appropriate addresses. This approach is suitable for the applications where the key does not changes constantly.

5.2 Results

Because of its flexibility to exchange among environments, we have used VHDL as a hardware description language. The software used for this work is Xilinx ISE design suite 12.3.This tool is used for writing, debugging ,optimizing and also for Synthesis ,Place and Route For simulation and checking the performance of the design, we used the simulation tool ISim Simulator available with the xilinx software. The following results are based on ISim

Table 5.1: Resource utilization summary for iterative design

Logic Utilization	Used	Available	Percentage Utilization
Number of Slice Registers	837	393600	0
Number of Slice LUTs expenses	696	196800	0
Number of fully used LUT-FF pairs	410	4626	36
Number of bonded IOBs	391	600	65
Number of Block RAM/FIFO	5	704	0

Table 5.2: Resource utilization summary for fully unrolled design

Logic Utilization	Used	Available	Percentage Utilization
Number of Slice Registers	4466	393600	1
Number of Slice LUTs expenses	2160	196800	1
Number of fully used LUT-FF pairs	2000	4626	43
Number of bonded IOBs	258	600	43
Number of Block RAM/FIFO	40	704	5

Simulation tools using Timing Analyzer and Waveform Generator. All the four round transformations of AES are implemented and tested separatly. The simulation is done by using test vectors from NIST for vertex 6 FPGA families and for specific device XC6VSX315T.A fully unrolled methods of design is implemented in order to improve the throughput of execution. We have said that this architecture is designed particularly for those applications where there is no area or resource and power constraint but we just like to show the resource utilization of the AES encryption module inorder to be sure that it does not exceed the available resource capacity of the targeted device.

5.3 Performance Evaluation

Before discussing the details of the timing results we would like to mention that each modules in the design was simulated using the Xilinx ISE simulator .As we have discussed in the previous chapter, we need 3 clock cycles for each round and we have ten modules in the implementation.So we need a total of 30 clock cycle for encryption of one data block, 128 bit or 16 byte in our case. Based on the data we obtained from the simulation tool, the overall system runs at 450.532MHz.The resource utilization for iterative design is give in Table5.1 and Table 5.2 shows the consumed resources for fully unrolled design. The calculation of the throughput of an AES implementation depends on various arguments. Equation(5.1)[9] explains that the throughput of a circuit depends on the block size and the latency. The time T_{clk} is the clock period of the circuit and depends on the longest critical path through the circuit. The value No.of Rounds per block is the number of rounds required to calculate one block of data. No.of Pipeline stages presents the number of pipeline stages in the architecture and No.of Utilized stages is the number of pipeline stages which can be used in parallel.

Table 5.3: Performance analysis of hardware optimizations

Architecture	Iterative	IP	Sub-unrolling	Sub-unrld piped	FU	FU piped
Unrolled rounds	1	1	k	k	10	10
Inner round stages	0	n	0	n	0	n
pipelined stages	1	n	k	k.n	10	10.n
round per clock	10	10	$10/k$	$10/k$	1	1
T_clk	T_{ref}	T_{ref}/n	T_{ref}	T_{ref}/n	T_{ref}	T_{ref}/n
utilized stages	1	n	k	k.n	10	10.n
Latency	$10.T_{ref}$	$10T_{ref}/n$	$10T_{ref}/k$	$10T_{ref}/k.n$	T_{ref}	T_{ref}/n
Throughput ECB	Th_{ref}	$n.Th_{ref}$	$k.Th_{ref}$	$k.n.Th_{ref}$	$10Th_{ref}$	$10.n.Th_{ref}$
Throughput CBC	Th_{ref}	Th_{ref}	Th_{ref}	Th_{ref}	Th_{ref}	Th_{ref}
Resources	A_{ref}	$A_{ref} + n.R$	kA_{ref}	$k(A_{ref} + n.R)$	$10A_{ref}$	$10(A_{ref} + n.R)$

$$Throughput = Blocksize/Latency.....(5.1)$$

$$Latency = (T_{clk} \cdot No.of Roundsperblock \cdot No.of Pipelinestages)/No.of Utilizedstages....(5.2)$$

Table 5.3 shows the performance analysis of different architectures of the AES algorithm[9]. In this table,IP stands for iterative pipelined and FU stands for fully unrolled. The first column contains the reference implementation where an iterative approach without pipelining is used. The last two lines show the data throughput rate related to the reference data throughput of the basic architecture. In non-feedback modes like ECB the more pipeline stages are included the higher is the performance. For feedback modes like CBC the data throughput can not be increased when only a single data stream is encrypted. For multiple independent blocks of data the same throughput is possible for all modes of operation. The hardware resources are listed in the last row. Thereby, all values are related to A_{ref} which are the requirements of the iterative reference implementation. Additional 128-bit register for pipelining are indicated with R. So using the above formulas we can calculate the performance of iterative design and our implementation as follows.

For the iterative method Timing
Minimum period: 1.943ns (Maximum Frequency: 514.593MHz)
Throughput= 6.5867 Gbps
Resource utilization

For fully unrolled method Timing
Minimum period: 2.220ns (Maximum Frequency: 450.532MHz)
Throughput=57.668 Gbps
Resource utilization

Table 5.4: Summary of related works

Name	Year	Module	Resources	Freq/Th	Comments
Nazar A.saqibet.al	2003	Enc	2744 CLB	20.192 MHz/258.5Mbits/s	Sequential
Nazar A.saqibet.al	2003	Enc	2136 CLB	22.41MHz/2868Mbits/s	Pipelined
DeenKotturi et.al	2005	Enc	5408 Slices	232.6MHz/29.77Gbits/s	Fully pipelined
Tim Good et.al	2005	Enc/Dec	16693 Slices	184.8MHz/23.654Mbits/s	FP loop unrolled
chih peng Fan	2008	Enc/Dec	139357 Slices	222.2MHz/28.4Gbits/s	Fully pipelined
Banraplangjyrwa	2009	Enc/Dec	6211 Slices	142.5MHz/1458Mbits/s	Iterative
Yulin Zhang	2010	Enc	2389 Slices	271.15MHz/34.7Gbits/s	Pipelined
Gurmail Singh	2011	Enc	6352 Slices	347.6MHz/44.5Gbits/s	Fully pipelined

5.4 Simulation results

For the purpose of simulating our design we have used ISim Simulator from Xilinx corporation and NIST text vector to check the proper functionality of the designs. Here we have performed the simulation using NIST test vector So by comparing the output of the simulation and the NIST test vectors output we have seen that the system works perfectly as expected and has a throughput of 57.668 Gbit/s.

5.5 Comparisons of Implementation

The architecture of an AES implementation mainly defines the required hardware resources on an FPGA. Additionally, the used synthesis tool and the target device influences this result. In this section the design is compared with some related work. Depending on the desired target technology for FPGAs or standard cell technology the results are measured in LUTs, Block RAMs or in gate equivalents. The comparison is therefore only possible for the same target technology. Table 5.4 summarizes the results obtained in some previous implementations[24]. Direct comparison among various FPGA implementations of the AES algorithms is difficult, since FPGA target devices are usually different.So the values have to be seen as a relative comparison of resource requirements and data throughput.

Chapter 6

CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

In this thesis we have improved the execution of AES Encryption module to achieve high data throughput.We have achieved 57.668Gbps .As it is known, the result of optimization tipically depends on the choice of the design parameters.In this paper, we have selected the AES 128 bit encryption module since it is the most widely used version.We have also used the pre calculated key values but we have added a means to change the key values by the main processor.For this purpose we have added a mode of operation to our module to share the data line for inserting the key values.By doing so, we eliminated the need to dedicate 128 bits data line for insertion of the key values ,hence, can be used for other purposes later in the implenetation.

6.2 Recommendations

As a recommendation to future works, we suggest the following.

1. The algorithm has three key size (128,192,256 bits) and we have shown the throughput improvment for AES 128 bits.But one can try to optimize the other key sizes by using similar or different methods.
2. As we have said we improved the encryption module so others can try to optimize the decyrtion module using the same or other techniques.
3. In this paper, we have optimized AES implementation for high data throughput but others can look for other optimization parameters like optimizing for area,power and cost.

Chapter 7

Appendix

Test Vectors

AES−128 (Nk=4, Nr=10)
PLAINTEXT: 00112233445566778899aabbccddeeff
KEY: 000102030405060708090a0b0c0d0e0f
CIPHER (ENCRYPT):

round[0].input 00112233445566778899aabbccddeeff
round[0].k_sch 000102030405060708090a0b0c0d0e0f
round[1].start 00102030405060708090a0b0c0d0e0f0
round[1].s_box 63cab7040953d051cd60e0e7ba70e18c
round[1].s_row 6353e08c0960e104cd70b751bacad0e7
round[1].m_col 5f72641557f5bc92f7be3b291db9f91a
round[1].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[2].start 89d810e8855ace682d1843d8cb128fe4
round[2].s_box a761ca9b97be8b45d8ad1a611fc97369
round[2].s_row a7be1a6997ad739bd8c9ca451f618b61
round[2].m_col ff87968431d86a51645151fa773ad009
round[2].k_sch b692cf0b643dbdf1be9bc5006830b3fe
round[3].start 4915598f55e5d7a0daca94fa1f0a63f7
round[3].s_box 3b59cb73fcd90ee05774222dc067fb68
round[3].s_row 3bd92268fc74fb735767cbe0c0590e2d
round[3].m_col 4c9c1e66f771f0762c3f868e534df256
round[3].k_sch b6ff744ed2c2c9bf6c590cbf0469bf41
round[4].start fa636a2825b339c940668a3157244d17
round[4].s_box 2dfb02343f6d12dd09337ec75b36e3f0
round[4].s_row 2d6d7ef03f33e334093602dd5bfb12c7

```
round[ 4].m_col 6385b79ffc538df997be478e7547d691
round[ 4].k_sch 47f7f7bc95353e03f96c32bcfd058dfd
round[ 5].start 247240236966b3fa6ed2753288425b6c
round[ 5].s_box 36400926f9336d2d9fb59d23c42c3950
round[ 5].s_row 36339d50f9b539269f2c092dc4406d23
round[ 5].m_col f4bcd45432e554d075f1d6c51dd03b3c
round[ 5].k_sch 3caaa3e8a99f9deb50f3af57adf622aa
round[ 6].start c81677bc9b7ac93b25027992b0261996
round[ 6].s_box e847f56514dadde23f77b64fe7f7d490
round[ 6].s_row e8dab6901477d4653ff7f5e2e747dd4f
round[ 6].m_col 9816ee7400f87f556b2c049c8e5ad036
round[ 6].k_sch 5e390f7df7a69296a7553dc10aa31f6b
round[ 7].start c62fe109f75eedc3cc79395d84f9cf5d
round[ 7].s_box b415f8016858552e4bb6124c5f998a4c
round[ 7].s_row b458124c68b68a014b99f82e5f15554c
round[ 7].m_col c57e1c159a9bd286f05f4be098c63439
round[ 7].k_sch 14f9701ae35fe28c440adf4d4ea9c026
round[ 8].start d1876c0f79c4300ab45594add66ff41f
round[ 8].s_box 3e175076b61c04678dfc2295f6a8bfc0
round[ 8].s_row 3e1c22c0b6fcbf768da85067f6170495
round[ 8].m_col baa03de7a1f9b56ed5512cba5f414d23
round[ 8].k_sch 47438735a41c65b9e016baf4aebf7ad2
round[ 9].start fde3bad205e5d0d73547964ef1fe37f1
round[ 9].s_box 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 9].s_row 54d990a16ba09ab596bbf40ea111702f
round[ 9].m_col e9f74eec023020f61bf2ccf2353c21c7
round[ 9].k_sch 549932d1f08557681093ed9cbe2c974e
round[10].start bd6e7c3df2b5779e0b61216e8b10b689
round[10].s_box 7a9f102789d5f50b2beffd9f3dca4ea7
round[10].s_row 7ad5fda789ef4e272bca100b3d9ff59f
round[10].k_sch 13111d7fe3944a17f307a78b4d2b30c5
round[10].output 69c4e0d86a7b0430d8cdb78070b4c55a
INVERSE CIPHER (DECRYPT):
round[ 0].iinput 69c4e0d86a7b0430d8cdb78070b4c55a
round[ 0].ik_sch 13111d7fe3944a17f307a78b4d2b30c5
round[ 1].istart 7ad5fda789ef4e272bca100b3d9ff59f
round[ 1].is_row 7a9f102789d5f50b2beffd9f3dca4ea7
round[ 1].is_box bd6e7c3df2b5779e0b61216e8b10b689
round[ 1].ik_sch 549932d1f08557681093ed9cbe2c974e
round[ 1].ik_add e9f74eec023020f61bf2ccf2353c21c7
```

```
round[ 2].istart  54d990a16ba09ab596bbf40ea111702f
round[ 2].is_row  5411f4b56bd9700e96a0902fa1bb9aa1
round[ 2].is_box  fde3bad205e5d0d73547964ef1fe37f1
round[ 2].ik_sch  47438735a41c65b9e016baf4aebf7ad2
round[ 2].ik_add  baa03de7a1f9b56ed5512cba5f414d23
round[ 3].istart  3e1c22c0b6fcbf768da85067f6170495
round[ 3].is_row  3e175076b61c04678dfc2295f6a8bfc0
round[ 3].is_box  d1876c0f79c4300ab45594add66ff41f
round[ 3].ik_sch  14f9701ae35fe28c440adf4d4ea9c026
round[ 3].ik_add  c57e1c159a9bd286f05f4be098c63439
round[ 4].istart  b458124c68b68a014b99f82e5f15554c
round[ 4].is_row  b415f8016858552e4bb6124c5f998a4c
round[ 4].is_box  c62fe109f75eedc3cc79395d84f9cf5d
round[ 4].ik_sch  5e390f7df7a69296a7553dc10aa31f6b
round[ 4].ik_add  9816ee7400f87f556b2c049c8e5ad036
round[ 5].istart  e8dab6901477d4653ff7f5e2e747dd4f
round[ 5].is_row  e847f56514dadde23f77b64fe7f7d490
round[ 5].is_box  c81677bc9b7ac93b25027992b0261996
round[ 5].ik_sch  3caaa3e8a99f9deb50f3af57adf622aa
round[ 5].ik_add  f4bcd45432e554d075f1d6c51dd03b3c
round[ 6].istart  36339d50f9b539269f2c092dc4406d23
round[ 6].is_row  36400926f9336d2d9fb59d23c42c3950
round[ 6].is_box  247240236966b3fa6ed2753288425b6c
round[ 6].ik_sch  47f7f7bc95353e03f96c32bcfd058dfd
round[ 6].ik_add  6385b79ffc538df997be478e7547d691
round[ 7].istart  2d6d7ef03f33e334093602dd5bfb12c7
round[ 7].is_row  2dfb02343f6d12dd09337ec75b36e3f0
round[ 7].is_box  fa636a2825b339c940668a3157244d17
round[ 7].ik_sch  b6ff744ed2c2c9bf6c590cbf0469bf41
round[ 7].ik_add  4c9c1e66f771f0762c3f868e534df256
round[ 8].istart  3bd92268fc74fb735767cbe0c0590e2d
round[ 8].is_row  3b59cb73fcd90ee05774222dc067fb68
round[ 8].is_box  4915598f55e5d7a0daca94fa1f0a63f7
round[ 8].ik_sch  b692cf0b643dbdf1be9bc5006830b3fe
round[ 8].ik_add  ff87968431d86a51645151fa773ad009
round[ 9].istart  a7be1a6997ad739bd8c9ca451f618b61
round[ 9].is_row  a761ca9b97be8b45d8ad1a611fc97369
round[ 9].is_box  89d810e8855ace682d1843d8cb128fe4
round[ 9].ik_sch  d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 9].ik_add  5f72641557f5bc92f7be3b291db9f91a
```

```
round[10].istart  6353e08c0960e104cd70b751bacad0e7
round[10].is_row  63cab7040953d051cd60e0e7ba70e18c
round[10].is_box  00102030405060708090a0b0c0d0e0f0
round[10].ik_sch  000102030405060708090a0b0c0d0e0f
round[10].ioutput 00112233445566778899aabbccddeeff
```

```
VHDL Code for the Implementation
use IEEE.STD_LOGIC_1164.ALL;
use work.AES_CONSTANTS.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity AES_128_UNROLLED is
Port (  SYS_CLK,RST,START : in STD_LOGIC;
                            PLAINTEXT_IN : in  STD_LOGIC_VECTOR (127 d

                            CIPHERTEXT_OUT : out  STD_LOGIC_VECTOR (12

end AES_128_UNROLLED;

architecture Behavioral of AES_128_UNROLLED is
 COMPONENT AES_ROUND IS
 PORT (          SYS_CLK,RST : in STD_LOGIC;
                            ROUND_DATA_IN : in  STD_LOGIC_VECTOR (127
                            ROUND_KEY_IN : in  STD_LOGIC_VECTOR (127 d
                            ROUND_DATA_OUT : out  STD_LOGIC_VECTOR (12
        END COMPONENT;
        COMPONENT LAST_ROUND IS
 PORT (          SYS_CLK,RST : in STD_LOGIC;
                            ROUND_DATA_IN : in  STD_LOGIC_VECTOR (127
                            ROUND_KEY_IN : in  STD_LOGIC_VECTOR (127 d
                            ROUND_DATA_OUT : out  STD_LOGIC_VECTOR (12
        END COMPONENT;

type KEY_ARRAY  is ARRAY (0 to 10) of  STD_LOGIC_VECTOR(127 downto 0);
SIGNAL KEY_BUF : KEY_ARRAY;
SIGNAL RST_BUF : STD_LOGIC := '0';

SIGNAL ROUND_DATA_IN1 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_IN2 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
```

```
SIGNAL ROUND_DATA_IN3 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_IN4 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_IN5 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_IN6 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_IN7 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_IN8 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_IN9 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL LAST_ROUND_DATA_IN : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS =>


SIGNAL ROUND_DATA_OUT1 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL ROUND_DATA_OUT2 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL ROUND_DATA_OUT3 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL ROUND_DATA_OUT4 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL ROUND_DATA_OUT5 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL ROUND_DATA_OUT6 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL ROUND_DATA_OUT7: STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL ROUND_DATA_OUT8 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL ROUND_DATA_OUT9 : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL LAST_ROUND_DATA_OUT : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS =>


begin


ROUND1: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN1,
                ROUND_KEY_IN => KEY_BUF(1),
                ROUND_DATA_OUT=>ROUND_DATA_OUT1,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
ROUND2: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN2,
                ROUND_KEY_IN => KEY_BUF(2),
                ROUND_DATA_OUT=>ROUND_DATA_OUT2,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
```

```
ROUND3: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN3,
                ROUND_KEY_IN => KEY_BUF(3),
                ROUND_DATA_OUT=>ROUND_DATA_OUT3,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
ROUND4: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN4,
                ROUND_KEY_IN => KEY_BUF(4),
                ROUND_DATA_OUT=>ROUND_DATA_OUT4,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
ROUND5: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN5,
                ROUND_KEY_IN => KEY_BUF(5),
                ROUND_DATA_OUT=>ROUND_DATA_OUT5,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
ROUND6: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN6,
                ROUND_KEY_IN => KEY_BUF(6),
                ROUND_DATA_OUT=>ROUND_DATA_OUT6,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
ROUND7: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN7,
                ROUND_KEY_IN => KEY_BUF(7),
                ROUND_DATA_OUT=>ROUND_DATA_OUT7,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
ROUND8: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN8,
                ROUND_KEY_IN => KEY_BUF(8),
                ROUND_DATA_OUT=>ROUND_DATA_OUT8,
                SYS_CLK => SYS_CLK,
```

```
                RST => RST_BUF
        );
ROUND9: AES_ROUND PORT MAP(
                ROUND_DATA_IN => ROUND_DATA_IN9,
                ROUND_KEY_IN => KEY_BUF(9),
                ROUND_DATA_OUT=>ROUND_DATA_OUT9,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
ROUND10: LAST_ROUND PORT MAP(
                ROUND_DATA_IN => LAST_ROUND_DATA_IN,
                ROUND_KEY_IN => KEY_BUF(10),
                ROUND_DATA_OUT=>LAST_ROUND_DATA_OUT,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
RST_BUF<=RST;
--      INTIAL_ROUND : PROCESS(SYS_CLK)
--      BEGIN
--              IF (SYS_CLK'event AND SYS_CLK ='1') then
--                      IF START = '1' then
--
--
--
--
--      END IF;
--      END IF;
--      END PROCESS;

ROUND_DATA_IN1 <= PLAINTEXT_IN XOR KEY(0);
        ROUND_DATA_IN2<=ROUND_DATA_OUT1;
        ROUND_DATA_IN3<=ROUND_DATA_OUT2;
        ROUND_DATA_IN4<=ROUND_DATA_OUT3;
        ROUND_DATA_IN5<=ROUND_DATA_OUT4;
        ROUND_DATA_IN6<=ROUND_DATA_OUT5;
        ROUND_DATA_IN7<=ROUND_DATA_OUT6;
        ROUND_DATA_IN8<=ROUND_DATA_OUT7;
        ROUND_DATA_IN9<=ROUND_DATA_OUT8;
        LAST_ROUND_DATA_IN<=ROUND_DATA_OUT9;
        CIPHERTEXT_OUT<=LAST_ROUND_DATA_OUT;
```

```
   KEY_DATA :  For i in 0 to 10 generate
      begin
                        KEY_BUF(i)<= KEY(i);
                end generate;

end Behavioral;
//////////////////////////////////////////////////////////////////////////////////
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:05:54 09/22/2012
-- Design Name:
-- Module Name:    AES_ROUND - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity AES_ROUND is
```

```
 Port (          SYS_CLK,RST : in STD_LOGIC;
                                ROUND_DATA_IN : in  STD_LOGIC_VECTOR (127
                                ROUND_KEY_IN : in  STD_LOGIC_VECTOR (127 d
                                ROUND_DATA_OUT : out  STD_LOGIC_VECTOR (12

end AES_ROUND;

architecture Behavioral of AES_ROUND is


COMPONENT SubBytes
        PORT(
                SubBytes_IN : IN std_logic_vector(127 downto 0);
                SYS_CLK : IN std_logic;
                RST : IN std_logic;
                SubBytes_OUT : OUT std_logic_vector(127 downto 0)
                );
        END COMPONENT;

COMPONENT MixColumns
        PORT(
                SYS_CLK : IN std_logic;
                RST : IN std_logic;
                DATA_IN : IN std_logic_vector(127 downto 0);
                DATA_OUT : OUT std_logic_vector(127 downto 0)
                );
        END COMPONENT;

COMPONENT AddRoundKey
        PORT(
                Data_IN : IN std_logic_vector(127 downto 0);
                Key_IN : IN std_logic_vector(127 downto 0);
                SYS_CLK : IN std_logic;
                RST : IN std_logic;
                Data_OUT : OUT std_logic_vector(127 downto 0)
                );
        END COMPONENT;
SIGNAL SubBytes_IN_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0'
SIGNAL SubBytes_OUT_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0
SIGNAL MixColumns_IN_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '(
```

```
SIGNAL MixColumns_OUT_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS =>
SIGNAL AddRoundKey_IN_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS =>
SIGNAL AddRoundKey_OUT_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS =>

SIGNAL ROUND_DATA_OUT_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS =>

SIGNAL KEY_BUF : STD_LOGIC_VECTOR(127 downto 0) := (OTHERS => '0');
SIGNAL RST_BUF : STD_LOGIC := '0';
begin

KEY_BUF<=ROUND_KEY_IN;
RST_BUF<=RST;
INST_SubBytes: SubBytes PORT MAP(
                SubBytes_IN => SubBytes_IN_BUF,
                SubBytes_OUT => SubBytes_OUT_BUF,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );


INST_MixColumns: MixColumns PORT MAP(
                SYS_CLK => SYS_CLK,
                RST => RST_BUF,
                DATA_IN => MixColumns_IN_BUF,
                DATA_OUT => MixColumns_OUT_BUF
        );

INST_AddRoundKey: AddRoundKey PORT MAP(
                Data_IN => AddRoundKey_IN_BUF,
                Key_IN => KEY_BUF,
                Data_OUT => AddRoundKey_OUT_BUF,
                SYS_CLK => SYS_CLK,
                RST => RST_BUF
        );
        ROUND_COMPUTATION : PROCESS(SYS_CLK)
begin
        IF (SYS_CLK'event AND SYS_CLK ='1') then
                IF RST = '1' then
                        ROUND_DATA_OUT <= (OTHERS => '0');
                ELSE
```

```
                    ROUND_DATA_OUT <= ROUND_DATA_OUT_BUF;
                END IF;
        END IF;
END PROCESS;
SubBytes_IN_BUF         <= ROUND_DATA_IN;
MixColumns_IN_BUF       <= SubBytes_OUT_BUF;
AddRoundKey_IN_BUF      <= MixColumns_OUT_BUF;
ROUND_DATA_OUT_BUF      <= AddRoundKey_OUT_BUF;

end Behavioral;
```

References

[1] Singh, S. N., "Nonlinear predictive control of Feedback Linearizable Systems and Flight Control System Design," *Journal of Guidance, Control, and Dynamics*, Vol. 18, No. 5, 1995, pp. 1023–1028.

[2] Babu, K. R., Sarma, I. G., and Swamy, K. N., "Switched Bias Proportional Navigation for Homing Guidance Against Highly Maneuvering Targets," *Journal of Guidance, Control, and Dynamics*, Vol. 17, No. 6, 1994, pp. 1357–1363.

[3] Khalil, H. K., *Nonlinear Systems*, Prentice Hall, New Jersey, USA, 2002.

[4] Nijmeijer, H. and Schaft, A. V. D., *Nonlinear Dynamical Control Systems*, Springer-Verlag, New York, USA, 1990.

[5] Singh, S. N., "Nonlinear predictive control of Feedback Linearizable Systems and Flight Control System Design," *Journal of Guidance, Control, and Dynamics*, Vol. 18, No. 5, 1995, pp. 1023–1028.