Generic Encryptor/Decryptor Core Implementation of Tiny AES

Thesis submitted in the partial fulfillment of requirement for the award of degree of

Master of Technology

in

VLSI Design & CAD

Submitted By:

Vikas Pathak

Roll No: 601061028

Under the guidance of:

Dr. Sanjay Sharma

Associate Professor

Department of Electronics and Communication Engineering

Thapar University

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

June 2012

DECLARATION

I, **Vikas Pathak**, hereby certify that the work which is being presented in this thesis entitled **"Generic Encryptor/Decryptor Core Implementation of Tiny AES"** by me in partial fulfillment of the requirements for the award of degree of Master of Technology in VLSI Design from Thapar University (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Sanjay Sharma**, Associate Professor, ECED, Thapar University, Patiala.

The matter presented in this thesis has not been submitted in any other University / Institute for the award of any other degree.

Date: 02/07/12

Vikas

Vikas Pathak
Roll No. 601061028

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Date:

Dr. Sanjay Sharma
Associate Professor
ECED

Countersigned by:

Dr. Rajesh Khanna
Professor and Head ECED
Thapar University, Patiala
Date:

Dr. S. K. Mohapatra
Dean of Academic Affairs
Thapar University, Patiala
Date:

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to my supervisor **Dr. Sanjay Sharma, Associate Professor**, Electronics and Communication Engineering Department, for her invaluable guidance and advice throughout this report. I am greatly indebted to him for encouragement and support without which, it would not have been possible for me to complete this undertaking successfully. I am truly very fortunate to have the opportunity to work with him. His insightful comments and suggestions have continually helped me to improve my understanding.

My sincere thanks are due to **Dr. Rajesh Khanna**, **Head of the Department** of Electronics and Communication Engineering for providing the constant encouragement and providing the facilities in the department for the completion of my seminar work. I also express my gratitude towards **Dr. Kulbir Singh**, **PG Coordinator**, Department of Electronics and Communication Engineering**,** for her valuable guidance and encouragement.

I would like to thank entire faculty and staff of Electronics and Communication Engineering Department and then friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

I take pride of myself being son of ideal parents for their over lasting desire, sacrifice, affection blessings and help without which it would not have been possible for me to complete my studies.

Last but not least, I would like to thank God for all good deeds.

Vikas Pathak

ABSTRACT

This thesis work presents Designing and Verification of Generic, Single Encryptor/ Decryptor core Unit for area efficient hardware implementation of Tiny advanced encryption standard (AES) algorithm, which can not only implement all the three AES (AES128, AES196, and AES256) algorithms, but also do both the encryption and decryption processes in just single design unit. The AES can be implemented in either software or hardware. Hardware acceleration is the use of hardware to perform a task more efficiently than is possible in software. In order to achieve higher performance in today's heavily loaded communication networks, utilization of hardware accelerators for cryptography algorithms is more efficient. The design has been implemented on both Xilinx ISE tool for FPGA flow and Synopsis (Design Compiler) tool for ASIC flow and compared the results for various architectures of design for both the FPGA and ASIC flow in terms of Area, Timing and Power report. A unique feature of this design is that the round keys, which are consumed during different iterations of encryption, are generated in parallel with the encryption process, which can further increase the speed of design.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Background

The emergence of the Internet as a trusted medium for commerce and communication has made cryptography an essential component of modern information systems. Cryptography provides the mechanisms necessary to implement accountability, accuracy and confidentiality in communications [1]. As demands for secure communication bandwidth grow, efficient cryptographic processing will become increasingly vital to good system performance. To introduce cryptography, an understanding of issues related to information security in general is necessary. Information security manifests itself in many ways according to the situation and requirement. Regardless of who is involved, to one degree or another, all parties to a transaction must have confidence that certain objectives associated with information security have been met.

At contract age the signature evolves to take on a very integral part of the person's identity. This signature is intended to be unique to the individual and serve as a means to identify, authorize, and validate. With electronic information the concept of a signature needs to be redressed; it cannot simply be something unique to the signer and independent of the information signed. Electronic replication of it is so simple that appending a signature to a document not signed by the originator of the signature is almost a triviality.

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication [1]. Cryptography is not the only means of providing information security, but rather one set of techniques.

1.1.1 A Brief History of Cryptography and data Security

For over 4,500 years, cryptography has existed as a means of secretly communicating information. Egyptian hieroglyphics are the first example of the use of cryptography to hide information from those not "in the know". The use of cryptography ciphers is central to events surrounding historical figures such as Julius Caesar, Queen Elizabeth I, Mata Hari, and Alfred Dreyfus, while playing a significant role in the Allies victory over the Axis power during World War II, directly affecting the outcome of the Battle of Midway and other engagements.

Cryptography on its more contemporary form was fathered by Claude Shannon in 1949. Widely known for his work in electronic communications and digital computing, Shannon established the basic mathematical theory for cryptography and its counterpart, cryptanalysis. shanon's method relied on a unique shared secret, referred to as the key, that allowed two parties to communicate securely as long as this key was not compromised. This class of algorithms, known as private-key, secret-key, or symmetric wad the sole method of secure communication until 1976, when Whitefiel Diffie and Martin Hellman proposed a revolutionary key distribution methodology. This methodology led to the development of a new class of algorithms, termed public-key or asymmetric-key, where a pair of mathematically related keys are used and one of these keys is made public, obviating the need for a secret shared specifically between two parties. Today, information systems typically use a hybrid approach, combining the benefits of symmetric-key and public-key algorithms to form a system that is both fast and secure.

1.1.2 Cryptography and data Security in the Modern World

Cryptography currently plays a major role in many information technology applications. With more than 188 million Americans connected to the Internet, the use of cryptography to provide information security has become a top priority. many applications – electronic mail, electronic banking, medical databases and electronic commerce require the exchange of private information. For example, when engaging in electronic commerce, customers provide credit card numbers when purchasing products, if the connection is not secure, an attacker can easily obtain this sensitive data. In order to implement a comprehensive security plan for a given network to guarantee the security of a connection, the following services must be provided:

- Confidentiality: Information cannot be observed by an unauthorized parity.
- Data integrity: Transmitted data within a give communication session cannot be altered in transit due to error or an unauthorized party.
- Message Authentication: Parties within a given communication session must provide certifiable proof validating the authenticity of a message.
- Non-repudiation: Neither the sender nor the receiver of a message may deny transmission.
- Entity authentication: Establishing the identity of an entity, such as a person or device.
- Access Control: controlling access to data and resources.

A fundamental goal of cryptography is to adequately address these six areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

Cryptography, over the ages, has been an art practiced by many who have devised ad hoc techniques to meet some of the information security requirements. The last twenty years have been a period of transition as the discipline moved from an art to a science.

1.2 Motivation

Encryption is the translation of data to a secret code. Apart from its uses in Military and Government to facilitate secret communication, Encryption is used in protecting many kinds of civilian systems such as Internet e-commerce, Mobile networks, automatic teller machine transactions, copy protection (especially protection against reverse Engineering and Software piracy), and many more. The Encrypted data can only be deciphered if one has the password or the Key. The Encryption algorithm is a set of well-defined steps to transform data from a readable format to an encoded format using the Key. This set of well-defined steps is also called cipher. An encryption algorithm provides Confidentiality, Authentication, Integrity and Non-repudiation. Confidentiality ensures that the information is accessible to only authorized set of people. Authentication is the act of establishing that the algorithm is genuine. Integrity in general means completeness but in encryption it is adhering to some set of principles. It is based on consistency with some mathematical proof. Non- repudiation in cryptology means that it can be verified that the sender and the recipient were, in fact, the parties who claimed to send or receive the message, respectively.

In today's digital world, encryption is emerging as a disintegrable part of all communication networks and information processing systems, for protecting both stored and in transit data. Encryption is the transformation of plain data (known as plaintext) into unintelligible data (known as ciphertext) through an algorithm referred to as cipher. There are numerous encryption algorithms that are now commonly used in computation, but the U.S. government has adopted the Advanced Encryption Standard (AES) to be used by Federal departments and agencies for protecting sensitive information. The National Institute of Standards and Technology (NIST) has published the specifications of this encryption standard in the Federal Information Processing Standards (FIPS) Publication 197.

A low-power and low-cost advanced encryption standard (AES) coprocessor is proposed for embedded system-on-a-chip (SoC) design. The cost and power consumption of the proposed AES coprocessor are reduced considerably by optimizing the architectures of

SubBytes/InvSubBytes and MixColumns/InvMixedColumn, integrating the encryption and decryption procedures together by the method of resource sharing, and using the hierarchical power management strategy based on finite state machine (FSM) and clock gating (CG) technologies.

1.2.1 Importance of Hardware Implementation of AES

Any conventional symmetric cipher, such as AES, requires a single key for both encryption and decryption, which is independent of the plaintext and the cipher itself. It should be impractical to retrieve the plaintext solely based on the ciphertext and the encryption algorithm, without knowing the encryption key. Thus, the secrecy of the encryption key is of high importance in symmetric ciphers such as AES. Software implementation of encryption algorithms does not provide ultimate secrecy of the key since the operating system, on which the encryption software runs, is always vulnerable to attacks.

There are other important drawbacks in software implementation of any encryption algorithm, including lack of CPU instructions operating on very large operands, word size mismatch on different operating systems and less parallelism in software. In addition, software implementation does not fulfil the required speed for time critical encryption applications. Thus, hardware implementation of encryption algorithms is an important alternative, since it provides ultimate secrecy of the encryption key, faster speed and more efficiency through higher levels of parallelism.

1.2.2 Tiny AES

Tiny AES is the digital logic hardware implementation of AES algorithm, which utilizes the minimum chip area of silicon IC. This can be achieved by using minimum logic resources for area optimization.

At the present day, crypto devices are produced in order to be smaller and faster. So, AES chips should not only use very small area, but also have enough throughput. AES has been used widely in banking, communication and e-sign sectors. Therefore, electronics devices which are used in these sectors must be small and power effective for mobile applications. For instance, a mobile phone or radio frequency identification (RFID) device have to operate as long as possible with the same battery. Also, they should be smaller due to the on-going trend.

The AES has also become the default choice for various security services in many applications and standards. On the other hand, a straightforward hardware implementation of the AES may not always be satisfy the tight constraints imposed by resource limited devices such as radio frequency identification (RFID) tags and tiny sensor networks.

This report presents the designing and verification of generic single Encryptor/Decryptor core of the Advanced Encryption Standard cryptosystem. This design is capable of handling all possible combinations of standard bit lengths (128,192,256) of data and key. The fully rolled inner-pipelined architecture ensures lesser hardware complexity. S-boxes here are going to be implemented using combinational logic over composite field arithmetic which completely eliminates the need of any internal memory. During our literature survey various VLSI architectures have been studied and their results are compared for area optimized design and the architecture which best suited our less area requirement is considered.

1.3 Symmetric-key Cryptography and AES

Symmetric-key cryptography, also called secret key cryptography, is the most intuitive kind of cryptography. It involves the use of a secret key known only to the participants of the secure communication. Symmetric-key cryptography can be used to transmit information over an insecure channel, but it has also other uses, such as secure storage on insecure media or strong mutual authentication. In symmetric-key cryptography, the key must be shared by both the sender and the receiver. The sender applies the encryption function using the key to the plaintext to produce the ciphertext.

The ciphertext is sent to the receiver, who then applies the decryption function using the same shared key. Since the plaintext cannot be derived from the ciphertext without knowledge of the key, the ciphertext can be sent over public networks such as the Internet. Therefore, symmetric key cryptography is characterized by the use of a single key to perform both the encrypting and decrypting of data. Since the algorithms are public knowledge, security is determined by the level of protection afforded the key. If key is kept secret, both the secrecy and authentication services are provided. Secrecy is provided, because if the message is intercepted, the intruder cannot transform the ciphertext into its plaintext format. Assuming that only two users know the key, authentication is provided because only a user with the key can generate ciphertext that a recipient can transform into meaningful plaintext.

The United States standard for Symmetric-key cryptography, in which the same key is used for both encryption and decryption, is the Data Encryption Standard (DES) [2]. The mainstream cryptographic community has long held that DES's 56-bit key is too short to withstand a brute-force attack from modern computers. The current key size of 56 bits (plus 8 parity bits) of DES is now starting to seem small, but the use of larger keys with triple DES (3DES) can generate much greater security. If security is the only consideration, then 3DES will be an appropriate choice for a standardized encryption algorithm for decades to come.

However, the principle drawback of 3DES is that the algorithm is relatively sluggish in software. The original DES was designed for mid 1970s hardware implementation and does not produce efficient software code. 3DES, which has three times as many rounds as DES, is correspondingly slower. A secondary drawback is that both DES and 3DES use a 64- bit block size. For reasons of both efficiency and security, a larger block size is desirable.

Because of these drawbacks, 3DES is not a reasonable candidate for long term use. As a replacement, NIST (National Institute of Standards and Technology) of USA in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which would have security length equal to or better than 3DES and significantly, improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits. The AES algorithm was selected in October 2001 after a multi-year evaluation process led by NIST with submissions and review by an international community of cryptography experts and the Rijndael algorithm, invented by Joan Daemen and Vincent Rijmen, was selected as the standard which was published in November 2002. NIST's intent was to have a cipher that will remain secure well into the next century [1].

1.4 Literature Review

During the literature survey several combinations of area optimization options for the AES has been explored. Many of these options were considered separately in the literature but they were not combined. Many of the available options for efficient logical resource utilization have been discussed, so that less logical hardware can be used and hence minimum chip area is utilized and reached to an optimum result and that optimum result is used in this design.

There are various blocks in AES algorithm. Each block has been discussed separately by literature survey of various papers and finds an area optimized solution of each block. During the study it is found that SubBytes/InvSubBytes and MixedColumn/ InvMixedColumn blocks are most area consuming blocks. The composite Galois Field (GF) architecture has been proposed for SubBytes block, whose digital logical hardware cost is about 500 equivalent gates, as compare to 3000 equivalent gate of look-up table method [4]. Area can also be reduced in MixedColumn/ InvMixedColumn block by resource sharing of MixedColumn parts in InvMixedColumn implementation [9]. The row transformation is the rerouting of the data and thereby needs no additional logic cells.

In this thesis report, we have explored several combinations of area optimization options for the AES. Many of these options were considered separately in the literature but they were not

combined. In this report I have mentioned many of the available options for efficient logical resource utilization, so that less logical hardware can be used and hence minimum chip area is utilized and reached to an optimum result and that optimum result is used in this design. It should also be noted that while our final implementation has been realized using a FPGA, the considered area optimization alternatives may also be applied to ASIC designs.

The issues of generic, Single Encryption/ Decryption design unit for AES implementation have also been discussed. It was found that reference [10] implement the encryption and decryption unit in single block diagram, but it is not generic, on the other hand reference [6] design unit is generic but both the encryption and decryption processes were not implemented in that design. The reference paper [13] implements the AES regiendal algorithm, which is capable of encrypt/decrypt all the three combination (128,192,256 bits) of plain/cipher text and round keys. But according to main AES algorithm, all these three combinations of key lengths were accepted, but for plain/cipher text these three combinations of data length were eliminated and only single 128-bit data length was proposed. The implementation of all the three combinations of AES algorithms (AES-128, AES-192 and AES-256) and the single design unit for encryption-decryption processes are not discussed yet in any paper.

Different versions of AES algorithm exist today (AES128, AES196, and AES256) depending on the size of the encryption key, but none of them has optimized the area for single design unit for encryption/decryption, which can support AES128, AES196, and AES256 simultaneously. In this report, design and verification of area optimized hardware implementation of Generic, single encryption/decryption design unit for implementing the AES128, AES196, and AES256 algorithm simultaneously, is implemented.

Hence in this thesis work, the designing of main AES algorithm is going to be executed, which not only implement all the three combinations of AES algorithm (AES-128, AES-192 and AES-256) but also perform both the encryption-decryption process in a single design unit. Hence our thesis topic named as “Generic single Encryptor/Decryptor core of Tiny AES”. Here the word generic is referred because our design supports all three types of AES algo’s (AES-128, AES-192 and AES-256), the word tiny indicate that our proposed design should consume the less silicon area and the word single is used for its capability to do both the encryption and decryption process in a single design unit, so that single IC chip can be used for both purposes, which in turn also reduces the cost of our proposed fabricated chip.

1.5 Objectives

The following are the main objectives of this thesis work:

1. To propose an area efficient encryption/decryption scheme for tiny embedded systems.
2. To incorporate the support for all the three types AES algorithms (AES-128, AES-192 and AES-256).
3. To functionally simulate the design and to perform synthesis analysis of various proposed design options.

1.6 Organization of Thesis Report

Chapter 1 gives introduction and historical background of encryption. It also describes the motivation, literature survey and objectives of this thesis report.

Chapter 2 starts with the basic concepts of Encryption, it compares the symmetric and public key cryptography. It also discusses the various mathematics terms used in AES standard.

Chapter 3 starts with the introduction of AES standard, also discusses the functioning of all blocks of AES standard as describe in NIST, FIST AES standard.

Chapter 4 describes the various options of area optimized hardware implementation of various blocks of AES, available in literature and gives the implementation details of all the blocks of my Tiny AES design.

Chapter 5 shows the simulation (Verification) and synthesis results of implemented Tiny AES design, which are taken on Modelsim and Xilinx ISE 13.1, Synopsys Tools .

Chapter 6 concludes the present work of our thesis report. Scope of future work is also mentioned in this chapter.

Chapter 2

Fundamentals of Encryption

2.1 Introduction

Cryptography is generally understood to be the study of the principles and techniques by which information is converted into an encrypted version that is difficult (ideally impossible) for any unauthorized person to convert to the original information, while still allowing the intended reader to do so. In fact, cryptography covers rather more than merely encrypt in and decryption. It is, in practice, a specialized branch of information theory with substantial addition from other branches of mathematics. Cryptography is probably the most important aspect of communications security and is becoming increasingly important as a basic building block for computer security.

The increased use of computer and communications systems by the industry has increased the risk of theft of proprietary information. Although these threats may require a variety of countermeasures, cryptography is a primary method of protecting valuable electronic information. In data and telecommunications, cryptography is necessary when communicating over any unsecured medium, which includes just about any network, particularly the Internet. Within the context of any application-to-application communication, there are some specific security requirements, including:

- Authentication: The process of proving one's identity.
- 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, in general, two types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric or conventional) cryptography and public-key (or asymmetric) cryptography. In symmetric-key cryptography, an algorithm is used to scramble the message using a secret key in such a way that it becomes unusable to all except the ones that have access to that secret key. The most widely known symmetric cryptographic algorithm is DES, developed by IBM in the seventies. It uses a key of 56 bits and operates on chunks of 64 bits at a time. In public key cryptography, algorithms use two different keys: a private and a public one. A message encrypted with a private key can be decrypted with its public key (and vice versa). The owner of the key pair holds the private key, and may

distribute the public key to anyone. Someone who wants to send a secret message uses the public key of the intended receiver to encrypt it. Only the receiver holds the private key and can decrypt it.

The two basic building blocks of all encryption techniques are substitution and transposition. A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. Transposition technique is a different kind of mapping where mapping is achieved by performing some sort of permutation on the plaintext letter.

2.2 Symmetric-key Cryptosystem

2.2.1 Principles of Symmetric-key Cryptosystem

In Symmetric-key cryptography, a single key is used for both encryption and decryption. The sender uses the key (or some set of rules) to encrypt the plaintext and sends the cipher text to the receiver. The receiver applies the same key (or rule set) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption. With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Symmetric-key cryptography schemes are generally categorized as being either stream ciphers or block ciphers. Stream ciphers operate on a single bit (byte or computer word) at a time, and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same cipher text when using the same key in a block cipher whereas the same plaintext will encrypt to different cipher text in a stream cipher.

Stream ciphers come in several flavours but two are widely used. Self-synchronizing stream ciphers calculate each bit in the key stream as a function of the previous n bits in the key stream. It is termed "self-synchronizing" because the decryption process can stay synchronized with the encryption process merely by knowing how far into the n-bit key stream it is. One problem is error propagation; a garbled bit in transmission will result in garbled bits at the receiving side.

Synchronous stream ciphers generate the key stream in a fashion independent of the message stream but by using the same key stream generation function at sender and receiver. While stream ciphers do not propagate transmission errors, they are, by their nature, periodic so that the key stream will eventually repeat.

2.2.2 Model of Symmetric-key Cryptosystem

A symmetric or conventional encryption scheme has five ingredients (Figure 2.1):

(i) Plaintext/Message: This is the original intelligible message or data that is fed into the algorithm as input.

(ii) Encryption Algorithm: The encryption algorithm performs various substitution and transformation on the plaintext.

(iii) Secret Key: The secret key is also the input to the encryption algorithm. The key is a value independent of the plaintext. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.

(iv) Cipher text: This is the scrambled message produced as output. It depends on the plaintext and secret key. For a given message, two different keys will produce two different cipher texts.

(v) Decryption Algorithm: This is essentially the encryption algorithm run in reverse. It takes the cipher text and the secret key and produces the original plaintext.

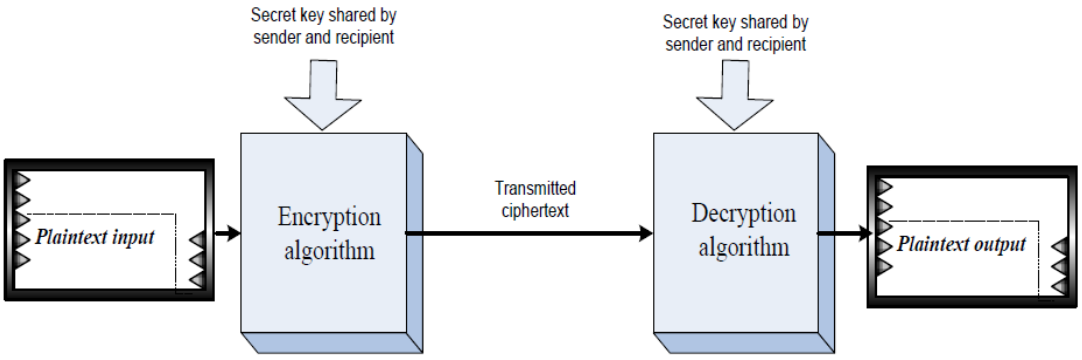


Figure 2.1: Model of Symmetric-key Cryptosystem

There are two requirements for secure use of symmetric-key encryption:

(a) A strong encryption algorithm: At a minimum, the algorithm to be such that an opponent who knows the algorithm and has access to one or more cipher texts would be unable to decipher the cipher text or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt cipher text or discover the key even if he or she is in possession of a number of cipher texts together with the plaintext that produced each cipher text.

(b) Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If some can discover the key and knows the algorithm, all communication using this key is readable.

In symmetric-key cryptosystem, a source produces a message in X = [X1, X2, X3, . . X_M] the plaintext, where M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. The binary alphabet {0,1} is also typically used. For encryption, a key of the form K = [K_1, K_2, K_3, … , K_J] is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination. With the message X and the encryption key K as input; the encryption algorithm forms the cipher text Y = [Y_1, Y_2, Y_3, … . . Y_N]. This process can be expressed using the following notation: $Y = E_K (Y)$.

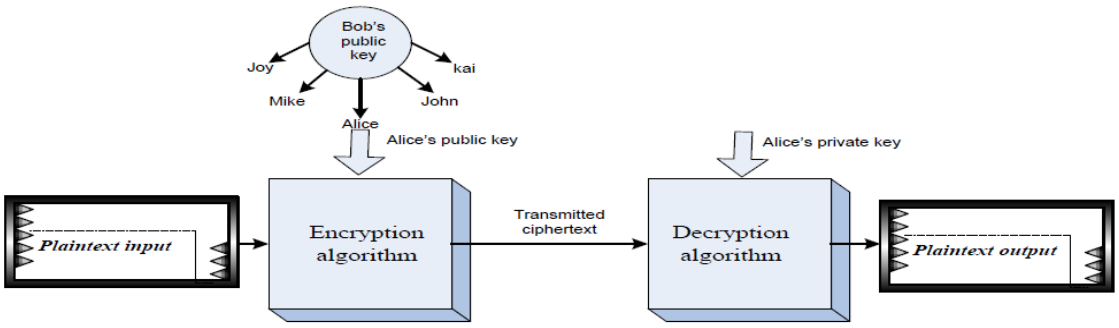
This notation indicates that Y is produced by using the encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key.

The intended receiver, in possession of the key, is able to invert the transformation:

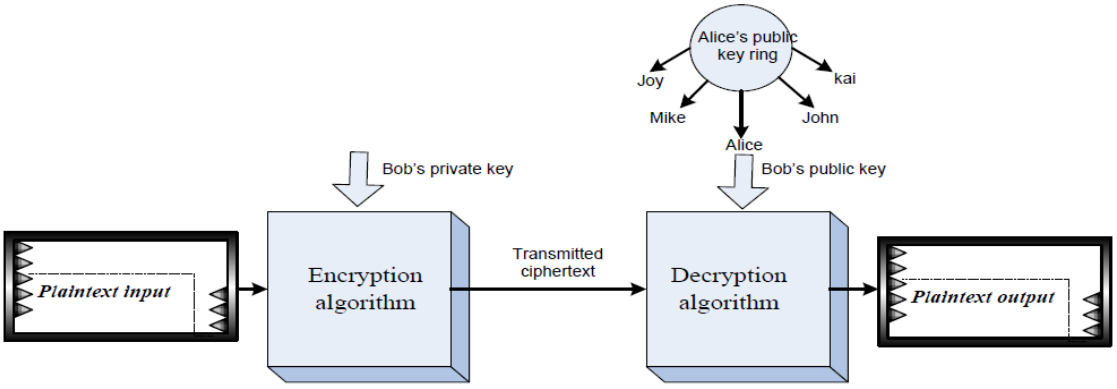
$$X = E_K (Y)$$

2.3 Public Key Cryptography

Symmetric-key cryptography is based on the sender and receiver of a message knowing and using the same secret key: the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. The main problem is getting the sender and receiver to agree on the secret key without anyone else finding out. If they are in separate physical locations, they must trust a courier, a phone system, or some other transmission medium to prevent the disclosure of the secret key being communicated.



(a)



(b)

Figure 2.2: Public-key Cryptography

2.4 Importance of Public-key and Symmetric key Cryptography

The primary advantage of public-key cryptography is increased security and convenience. Private keys never need to transmitted or revealed to anyone. In a symmetric key system, by contrast, the symmetric keys must be transmitted (either manually or through a communication channel), and there may be a chance that an enemy can discover the symmetric keys during their transmission.

Another major advantage of public-key systems is that they can provide a method for digital signatures. Authentication via symmetric-key systems requires the sharing of some symmetric keys and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming that the shared symmetric key was somehow compromised by one of the parties sharing the symmetric-key.

Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called non-repudiation.

A disadvantage of using public-key cryptography for encryption is speed; there are popular symmetric-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used with symmetric-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public and symmetric-key systems in order to get both the security advantages of public-key systems and the speed advantages of symmetric-key systems. The public-key system can be used to encrypt a symmetric- key which is used to encrypt the bulk of a file or message. Such a protocol is called a digital envelope.

In some situations, public-key cryptography is not necessary and symmetric-key cryptography alone is sufficient. This includes environments where secure symmetric key

agreement can take place, for example by users meeting in private. It also includes environments where a single authority knows and manages all the keys (e.g. closed banking system). Since the authority knows everyone's keys already, there is not much advantage for some to be public and others private. Also, public-key cryptography is usually not necessary in a single-user environment. In general, public-key cryptography is best suited for an open multi-user environment.

Public-key cryptography is not meant to replace symmetric-key cryptography, but rather to supplement it, to make it more secure. The first use of public-key techniques was for secure key exchange in an otherwise symmetric-key system, this is still one of its primary functions. Symmetric-key cryptography remains extremely important and is the subject of on-going study and research.

2.4.1 Advantages of symmetric-key cryptography

- Symmetric-key ciphers can be designed to have high rates of data throughput.
- Keys for symmetric-key ciphers are relatively short.
- Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions and computationally efficient digital signature schemes, to name just a few.
- Symmetric-key ciphers can be composed to produce stronger ciphers. Simple transformations which are easy to analyse, but on their own weak, can be used to construct strong product ciphers.

2.4.2 Disadvantages of symmetric-key cryptography

- In a two-party communication, the key must remain secret at both ends.
- In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted TTP.
- In a two-party communication between entities A and B, sound cryptographic practice dictates that the key be changed frequently and perhaps for each communication session.
- Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a TTP.

2.4.3 Advantages of public-key cryptography

- Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).
- Depending on the mode of usage, a private key/public key pair may remain unchanged for considerable periods of time, e.g., many sessions (even several years).
- Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.
- In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

2.4.4 Disadvantages of public-key encryption

- Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best-known symmetric-key schemes.
- Key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

2.4.5 Summary of comparison

- Public-key cryptography facilitates efficient signatures (particularly nonrepudiation) and key management, and
- Symmetric-key cryptography is efficient for encryption and some data integrity applications.

2.5 Basic Mathematics of Encryption

The byte value in AES [1] is represented as a set of bits (0 or 1) and is represented the collection of bits separated by comma as $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. These bytes are interpreted as finite field elements using polynomial representation as

$$b_7x^7 + b6x6 + b_5x^5 + b_4x4 + b_3x^3 + b_2x^2 + b_1x + b_0 \qquad (2.1)$$

All the operations performed in AES are modulo-2 operations. These Operations are not as the same operations used in general Number System. The basic operations on which the entire math of the AES algorithm is based are Addition, Multiplication. These operations are explained in the subsequent subsections.

2.5.1 Addition

In modulo-2 additions, two elements are added by adding the coefficients of the corresponding powers in the polynomial. The addition operation here is the XOR operation

denoted by +. Subtraction of the polynomials is exactly the same as addition. Addition of 2 bytes { $a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1 a_0$} and { $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1 b_0$} is {$c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1 c_0$} where each $c_i = a_i$ xor b_i. For example

if $A = (179)_{10}$ and $B = (90)_{10}$ then

$A+B = (179)_{10}$ xor $(90)_{10} = (233)_{10}$

$A-B = (233)_{10}$. (2.2)

This can be represented in different forms as shown below

$(x^7 + x^5 + x^4 + x + 1) \oplus (x^6 + x^4 + x^3 + x) = x^7 + x^6 + x^5 + x^3 + 1$ …. Polynomial notation

{10110011} ⊕ {01011010} = {11101001} Binary notation

{B3} ⊕ {5A} = {E9} Hexadecimal notation

2.5.2 Multiplication

Multiplication denoted by • is the multiplication of the polynomial over an irreducible polynomial of degree 8 [1]. A polynomial is irreducible, if the only divisors of the irreducible polynomial are one and the number itself. The irreducible polynomial is referred to as a Prime Number in Mathematics. For the AES algorithm this irreducible polynomial is

$m(x) = x^8 + x^4 + x^3 + x + 1$ (2.3)

Equation (2.3) can be also be written as $\{1\_00011011\}_2$ or $\{01\_1b\}_h$ in binary and Hexadecimal format respectively.

For example {72} • {18} = {dc}

$(x^6 + x^5 + x4 + x) . (x^4 + x^3) = x^{10} + x^9 + x^8 + x^5 + x^9 + x^8 + x^7 + x^4$

$= x^{10} + x^7 + x^5 + x^4$ 2.4

$x^{10} + x^7 + x^5 + x^4$ modulo $(x^8 + x^4 + x^3 + x + 1)$

$= x^7 + x^6 + x^4 + x^3 + x^2$ (2.5)

The modular multiplication over the polynomial m(x) ensures that the resultant product of all multiplication has the degree less than eight and can be represented by a byte. Unlike the addition the multiplication cannot be comprehended by a single operation.

2.5.3 Polynomials with coefficients in GF (2^8)

Four term polynomial are polynomial whose coefficients are finite field elements like $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ (2.6)

Where [a_3, a_2, a_1, a_0] will form a word. The coefficients of these polynomial i.e. a_3, a_2, a_1, a_0 are valid bytes in the Finite field. The addition of such polynomials is similar to that of our bit addition. To illustrate that let

$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ (2.7)

be the second four term polynomial. Addition is done by adding the coefficients of polynomial with like powers of x. This addition is the XOR operation between the corresponding BYTES in each polynomial.

Thus $a(x) + b(x) = (a_3 \text{ xor } b_3)\, x^3 + (a_2 \text{ xor } b_2)\, x^2 + (a_1 \text{ xor } b_1)\, x + (a_0 \text{ xor } b_0)$

similarly $a(x) \times b(x) = (a_3x^3 + a_2 x^2 + a_1 x + a_0) \times (b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0)$

$= a_3b_3 x^6 + (a_3 b_2 \text{ xor } a_2 b_3) x^5 + (a_3 b_1 \text{ xor } a_2 b_2 \text{ xor } a_1 b_3) x^4$

$+ (a_3 b_0 \text{ xor } a_2 b_1 \text{ xor } b_2 a_1 \text{ xor } b_3 a_0) x^3 + (a_2 b_0 \text{ xor } a_1 b_1 \text{ xor } b_2 a_0)$

$x^2 + (a_1 b_0 \text{ xor } b_1 a_0) x + a_0 b_0$ (2.8)

the resultant is divided by the irreducible polynomial of degree 4. The polynomial is x^4+1. Hence the resultant product is

$c(x) = a(x) \times b(x) \text{ modulo } x^4+ 1$

$= (a_3 b_0 \text{ xor } a_2 b_1 \text{ xor } b_2 a_1 \text{ xor } b_3 a_0) x^3+$

$(a_2 b_0 \text{ xor } a_1 b_1 \text{ xor } b_2 a_0 \text{ xor } a_3b_3) x^2 +$

$(a_1 b_0 \text{ xor } b_1 a_0 \text{ xor } a_3 b_2 \text{ xor } a_2 b_3) x +$

$(a_0 b_0 \text{ xor } a_3 b_1 \text{ xor } a_2 b_2 \text{ xor } a_1 b_3)$ (2.9)

When a (x) is fixed, the same operation can be written in matrix form as:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

As $x^4 +1$ is not an irreducible polynomial over GF (2^8), hence this multiplication is not invertible. But the AES Algorithm specifies a fixed four term polynomial that has an inverse

$a(x) = \{03\} x^3 + \{01\} x^2 + \{01\} x + \{02\}$ (2.10)

$a\text{-}1(x) = \{0b\} x^3 + \{0d\} x^2 + \{09\} x + \{0e\}$ (2.11)

The above polynomial is used in the COLUMN mixing for Encryption and the inverse polynomial is used in INVERSE COLUMN mixing for Decryption. Another polynomial used in the algorithm is

$r(x) = \{01\} x^3 + \{00\} x^2 + \{00\} x + \{00\}$ (2.12)

The polynomial r(x) is used in one of the steps in the AES algorithm which transforms a word like $[b_0,b_1,b_2,b_3]$ to $[b_1, b_2, b_3, b_0]$.

Chapter 3

ADVANCED ENCRYPTION STANDARD (AES)

3.1 Overview

This chapter is a summary of the Federal Information Processing Standards (FIPS) Publication 197 [1], issued by the National Institute of Standards and Technology (NIST) which specifies the Advanced Encryption Standard. Throughout the remainder of this chapter, the mathematical properties of the Advanced Encryption Standard (AES) are introduced using the information obtained from the AES specification.

The AES is a subset of a much larger encryption algorithm known as Rijndael, which was one of many proposals to the NIST competing for becoming a standard encryption algorithm. On October of 2000, the NIST announced the Rijndael algorithm as the winner due to the best overall score in security, performance, efficiency, implementation capability and simplicity.

The AES algorithm is a symmetric cipher. In symmetric ciphers, a single secret key is used for both the encryption and decryption, whereas in asymmetric ciphers, there are two sets of keys known as private and public keys. The plaintext is encrypted using the public key and can only be decrypted using the private key.

In addition, the AES algorithm is a block cipher as it operates on fixed-length groups of bits (blocks), whereas in stream ciphers, the plaintext bits are encrypted one at a time, and the set of transformations applied to successive bits may vary during the encryption process.

The AES algorithm operates on blocks of 128 bits, by using cipher keys with lengths of 128, 192 or 256 bits for the encryption process. Although the original Rijndael encryption algorithm was capable of processing different blocks sizes as well as using several other cipher key lengths, but the NIST did not adopt these additional features in the AES. [1]

3.2 Inputs, Outputs and the State

The plaintext input and ciphertext output for the AES algorithms are blocks of 128 bits. The cipher key input is a sequence of 128, 192 or 256 bits. In other words the length of the cipher key, N_k, is either 4, 6 or 8 words which represent the number of columns in the cipher key. The AES algorithm is categorized into three versions based on the cipher key length. The number of rounds of encryption for each AES version depends on the cipher key size.

In the AES algorithm, the number of rounds is represented by N_r, where $N_r = 10$ when $N_k = 4$, $N_r = 12$ when $N_k = 6$, and $N_r = 14$ when $N_k = 8$. The following Table 3.1 illustrated the variations of the AES algorithm. For the AES algorithm the block size (N_b), which represents the number of columns comprising the State is $N_b = 4$.

AES Version	Key Length (N_k words)	Block Size (N_b words)	Number of Rounds (N_r rounds)
AES128	4	4	10
AES192	6	4	12
AES256	8	4	14

Table 3.1: AES Variations

The basic processing unit for the AES algorithm is a byte. As a result, the plaintext, ciphertext and the cipher key are arranged and processed as arrays of bytes. For an input, an output or a cipher key denoted by *a*, the bytes in the resulting array are referenced as a_n, where *n* is in one of the following ranges:

Block length = 128 bits, $0 <= n < 16$

Key length = 128 bits, $0 <= n < 16$

Key length = 192 bits, $0 <= n < 24$

Key length = 256 bits, $0 <= n < 32$

All byte values in the AES algorithm are presented as the concatenation of their individual bit values between braces in the order $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. These bytes are interpreted as finite field elements using a polynomial representation:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x + b_1x + b_0x = \sum_{i=0}^{7} b_i x^i \qquad (3.1)$$

As an example, {10001001} (or {85} in hexadecimal) identifies the polynomial $x^7 + x^3 + 1$. The arrays of bytes in the AES algorithm are represented as $a_0a_1a_2...a_n$.

All the AES algorithm operations are performed on a two dimensional 4x4 array of bytes which is called the State, and any individual byte within the State is referred to as $s_{r,c}$, where letter '*r*' represent the row and letter '*c*' denotes the column. At the beginning of the encryption process, the State is populated with the plaintext. Then the cipher performs a set of substitutions and permutations on the State. After the cipher operations are conducted on

the State, the final value of the state is copied to the ciphertext output as is shown in the following figure.

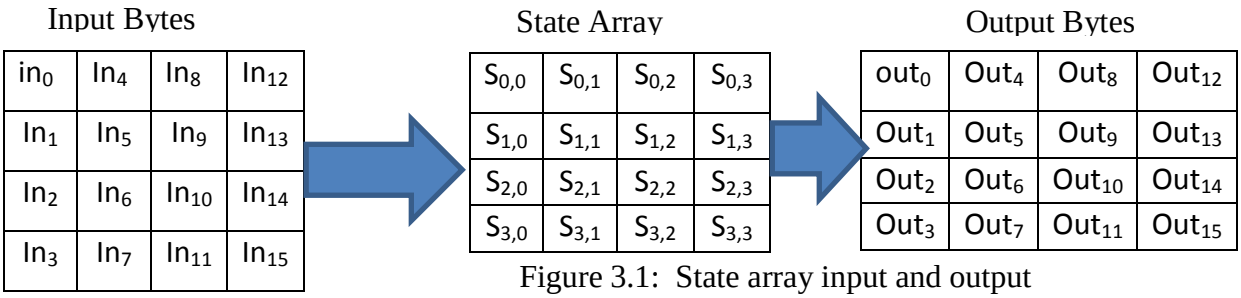


Figure 3.1: State array input and output

At the beginning of the cipher, the input array is copied into the *State* according the following scheme:

$$s[r,c] = in\,[r + 4c] \quad \text{for } 0 \le r < 4 \text{ and } 0 \le c < 4,$$

and at the end of the cipher the *State* is copied into the output array as shown below:

$$out[r+4c] = s[r,c] \quad \text{for } 0 \le r < 4 \text{ and } 0 \le c < 4$$

3.3 Cipher Transformations

The AES cipher either operates on individual bytes of the State or an entire row/column. At the start of the cipher, the input is copied into the State as described in Section 3.2. Then, an initial Round Key addition is performed on the State. Round keys are derived from the cipher key using the Key Expansion routine. The key expansion routine generates a series of round keys for each round of transformations that are performed on the State.

The transformations performed on the state are similar among all AES versions but the number of transformation rounds depends on the cipher key length. The final round in all AES versions differs slightly from the first N_r □1 rounds as it has one less transformation performed on the State. Each round of AES cipher (except the last one) consists of four byte oriented transformation. These transformations are:

a) Byte substitution using s box table(S-box)
b) Shifting rows of the state array using different offsets(Row transformation)
c) Mixing the data within each column of the state array.(Mixing columns)
d) Adding a round key to the state.(Add round key)

The AES cipher is described as a pseudo code in Figure 3.2. As shown in the pseudo code, all the N_r rounds are identical with the exception of the final round which does not include the MixColumns transformation. The array w[] represents the round keys that are generated by

the key expansion routine. In the following sections, individual transformations that are used in each encryption round are described.

The AES algorithm can also be described by the flow chart and encryption/decryption structure as shown in figure 3.3.

```
Cipher(byte PlainText[4*Nb], byte CipherText[4*Nb], word w[Nb*(Nr+1)])

Begin

   byte state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1

      SubBytes(state)

      ShiftRows(state)

      MixColumns(state)

      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])

   end for

SubBytes(state)

ShiftRows(state)

AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

out = state

end
```

Figure 3.2: Pseudo Code of AES Encryption Algorithm

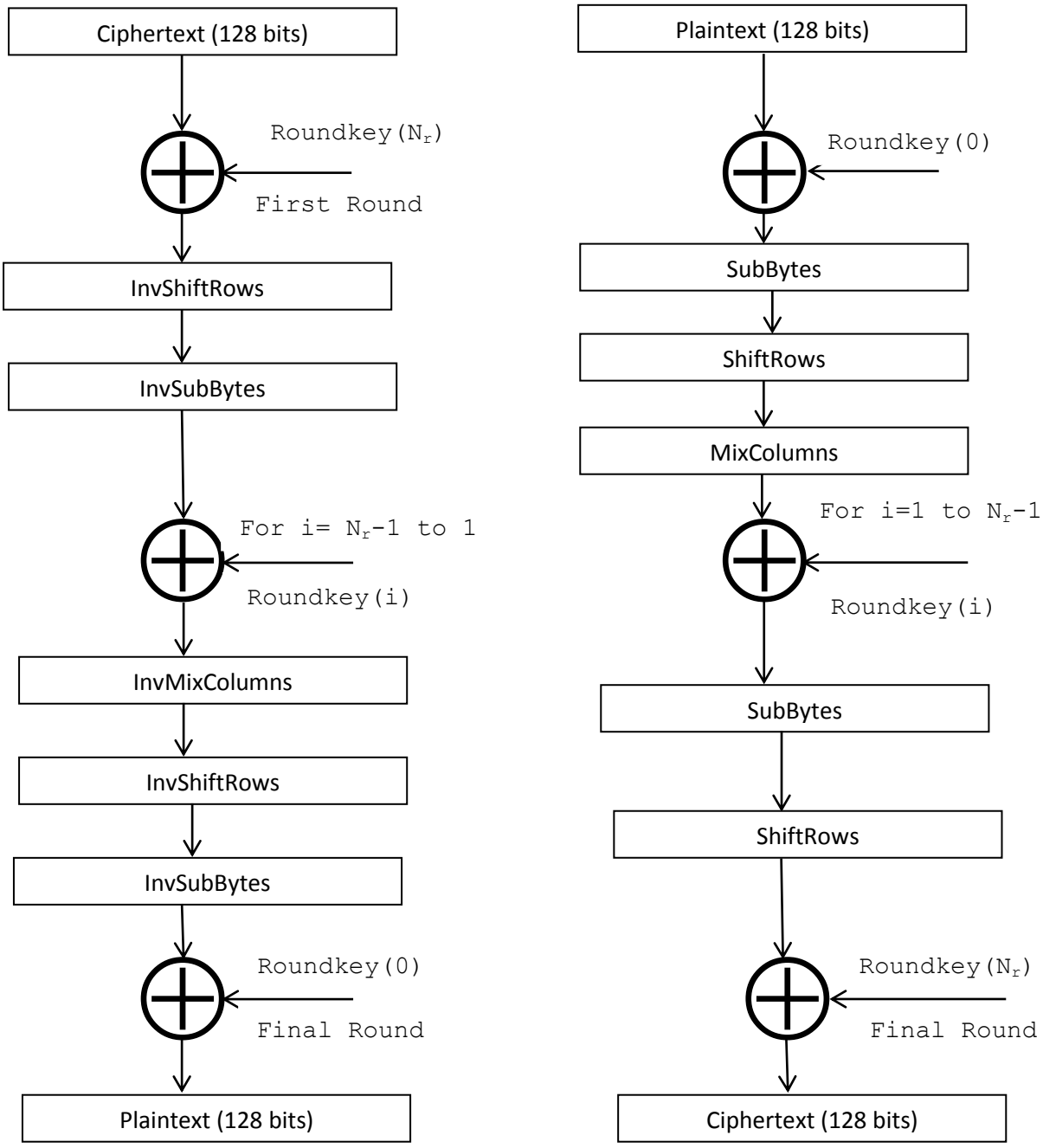


Figure 3.3: AES Decryption (a) and Encryption (b) structures

3.3.1 SubBytes ( ) Transformation

The *SubBytes* is a byte substitution operation performed on individual bytes of the State, as shown in Figure 3.4, using a substitution table called S-box.

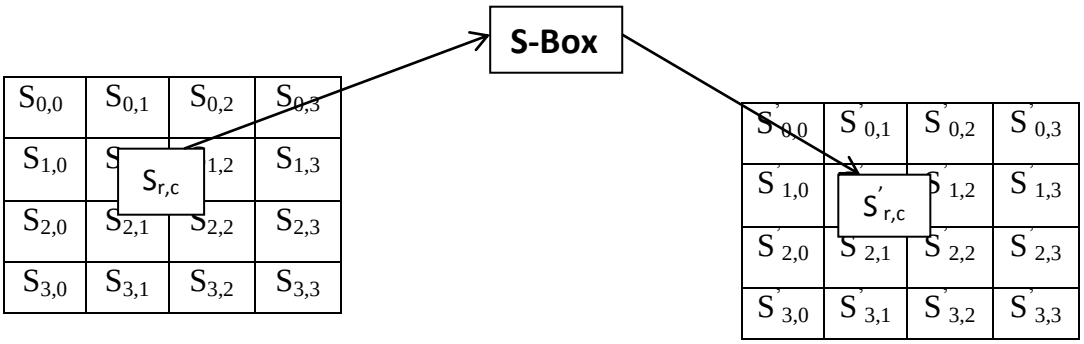


Figure 3.4 SubBytes Transformation

The invertible S-box table is constructed by performing the following transformation on each byte of the State.

- Take the multiplicative inverse in the finite field GF(2^8) of the byte.
- Apply the following transformation to the byte:

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \qquad (3.2)$$

The b_i is the i^{th} bit of the byte and c_i is the i^{th} bit of a constant byte with the value of {63}. The combination of the two transformations can be expressed in matrix form as shown below:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The S-box table shown in Table 2 is constructed by performing the two transformations described earlier for all possible values of a byte, ranging from {00} to {ff}. For example the substitution value for {53} would be determined by the intersection of the row with index '5' and the column with index '3'.

		Y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**B**	**C**	**d**	**e**	**F**
X	**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	89	0d	Bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 3.2: AES S-box

3.3.2 ShiftRows ( ) Transformation

This cyclically shifts the last three rows of the state by different offsets. The first row is left unchanged in this transformation. Each byte of the second, third and fourth rows are shifted one, two and three positions to the left respectively, which is illustrated in Figure 3.5.

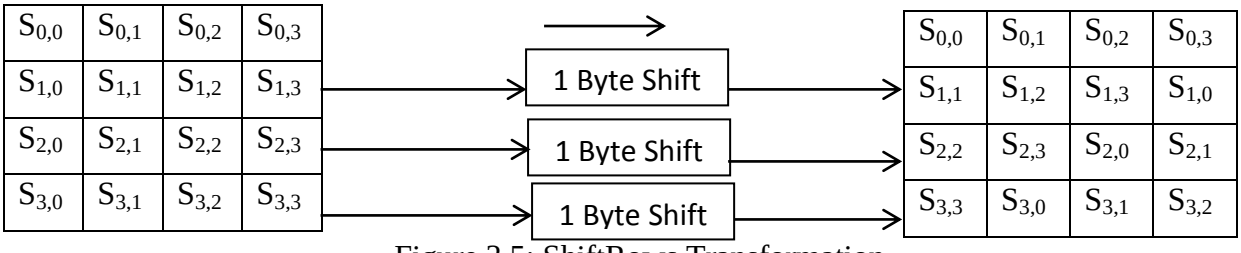


Figure 3.5: ShiftRows Transformation

3.3.3 MixColumns () Transformation

This transformation operates on the columns of the *State*, treating each columns as a four term polynomial the finite field GF(2^8). Each columns is multiplied modulo x^4+1 with a fixed four-term polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ over the GF(2^8). The MixColumns transformation can be expressed as a matrix multiplication as shown below:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{0,c} \\ s_{0,c} \\ s_{0,c} \end{bmatrix}$$

The MixColumns transformation replaces the four bytes of the processed column with the following values:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{0,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{03\} \bullet s_{0,c} \oplus s_{1,c}) \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \quad (3.3)$$

The "$\bullet$" corresponds to the multiplication of polynomials in GF(2^8) modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm the irreducible polynomial is:

$m(x) = x^8 + x^4 + x^3 + x + 1$.

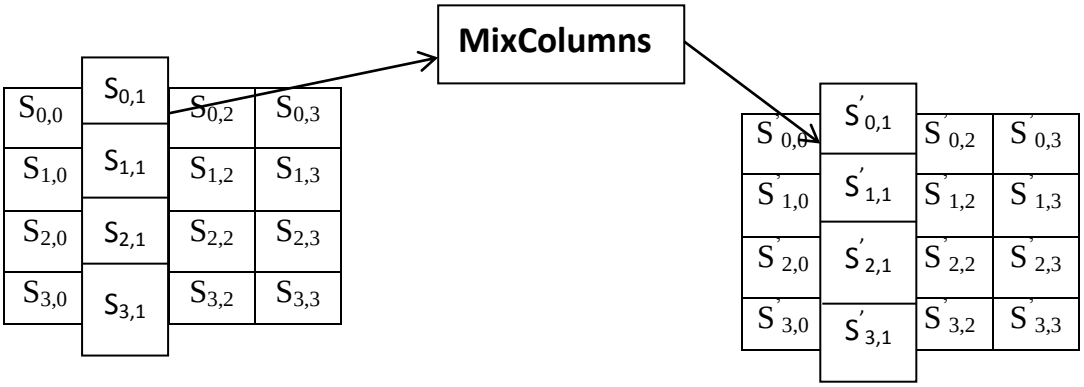


Figure 3.6 MixColumns Transformation

The MixColumns transformation is illustrated in Figure 5. This transformation together with ShiftRows, provide substantial diffusion in the cipher meaning that the result of the cipher

depends on the cipher inputs in a very complex way. In other words, in a cipher with a good diffusion, a single bit change in the plaintext will completely change the ciphertext in an unpredictable manner.

3.3.4 AddRoundKey ( ) Transformation

During the AddRoundKey transformation, the round key values are added to the State by means of a simple *Exclusive Or* (XOR) operation. Each round key consists of N_b words that are generated from the KeyExpansion routine. The round key values are added to the columns of the state in the following way:

$$\left[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}\right] = \left[s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}\right] \oplus \left[w_{round*Nb+c}\right] \quad \text{for } 0 \le c < N_b \qquad (3.4)$$

The *round* value is between $0 \le round \le N_r$. When round=0, the cipher key itself is used as the round key and it corresponds to the initial AddRoundKey transformation displayed in the pseudo code in Figure 3.3. The AddRoundKey transformation is illustrated in Figure 3.6.

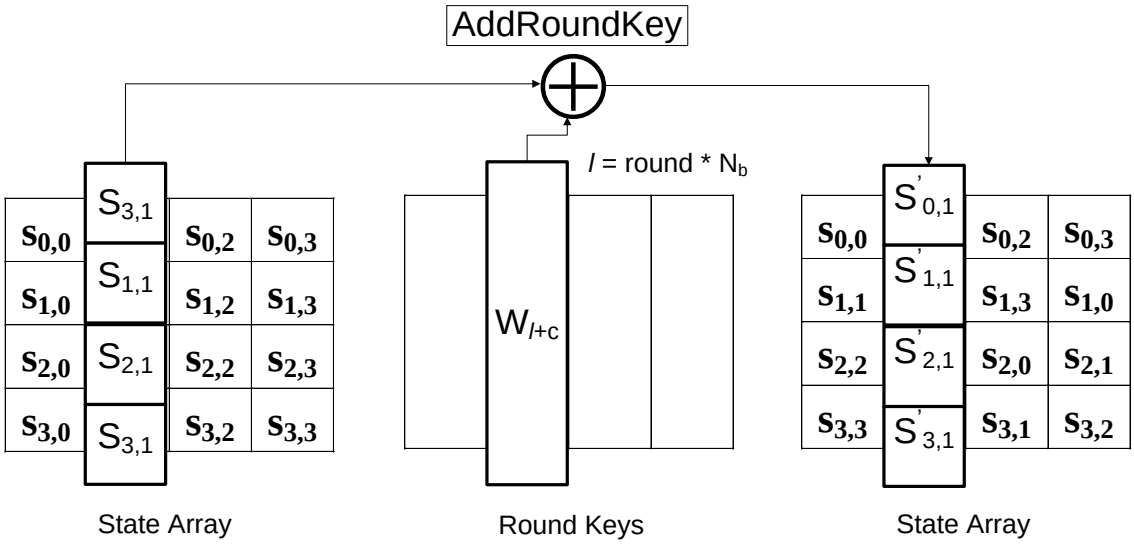


Figure 3.7: AddRoundKey Transformation

3.4 AES Key Expansion

The AES algorithm requires four words of round keys for each encryption round. That is total of 4*(N_r + 1) round keys considering the initial set of keys required for the first AddRoundKey transformation. All the round keys are derived from the cipher key itself.

The expansion of the cipher key into the round keys is performed by the KeyExpansion algorithm as shown in the pseudo code in Figure 3.8.

```
KeyExpansion(byte CipherKey[4*Nk], word w[Nb*(Nr+1)], Nk)

begin

word temp

i = 0

while (i < Nk)

w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])

i = i+1

end while

i = Nk

while (i < Nb * (Nr+1)]

temp = w[i-1]

if (i mod Nk = 0)

temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]

else if (Nk > 6 and i mod Nk = 4)

temp = SubWord(temp)

end if

w[i] = w[i-Nk] xor temp

i = i + 1

end while

end
```

Figure 3.8: Pseudo code for Key Expansion Algorithm

In the above pseudo code, the array w[] represents the round keys that are generated by the KeyExpansion routine and N_k represents the size of the cipher key. Depending on the version of the AES algorithm, N_k=4, 6 or 8. The first N_k words of the expanded key are filled with the cipher key. The Key Expansion transformation can also be described in a flowchart form as shown in figure 3.9.

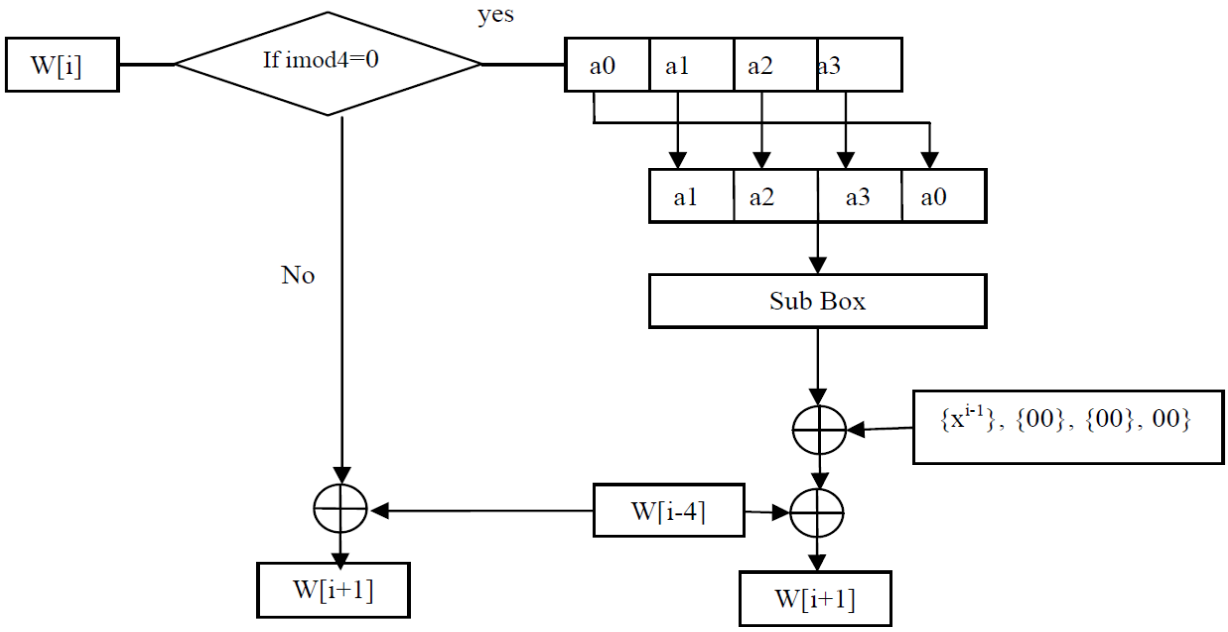


Figure 3.9: Key expansion flow chart

The SubWord ( ) function applies the same S-box substitution to each of the four bytes in the word. The RotWord( ) function takes a word [a0,a1,a2,a3] as input and perform a cyclic shift and returns the word [a1,a2,a3,a0]. The round constant word array, Rcon[i], contains a 32 bit value given by $[\{02\}^{i-1},\{00\},\{00\},\{00\}]$.

Every following round key , w[i], is equal to the XOR of the previous round key, w[i-1], and the word N_k positions earlier, w[i-N_k]. For words in positions that are a multiple of N_k, two transformations are initially applied to the previous round key, w[i-1]. These transformations are a cyclic shift of the bytes in the previous round key, followed by the application of the S-box table lookup to all four bytes of the word. Afterwards, an XOR with a round constant value, Rcon[i], is applied to the previous round key.

The KeyExpansion routine for the AES256 (N_k=8) is slightly different than the AES128 and AES192 ones, as an additional SubWord function is applied to the previous round key, w[i-1], prior to the XOR with w[i- N_k].

Chapter 4

Implementation of Generic Tiny AES Encryptor/Decryptor Core

4.1 Overview

In this chapter, the design of hardware model for implementing the AES algorithm is introduced. This chapter presents the designing of generic single Encryptor/Decryptor core of Tiny AES (Advanced Encryption Standard).The suggested design is capable of handling all possible combinations of standard bit lengths (128,192,256) of key. The fully rolled inner-pipelined architecture ensures lesser hardware complexity. S-boxes here are going to be implemented using combinational logic over composite field arithmetic which completely eliminates the need of any internal memory.[6]

In recent years, there has been an exponential growth in wireless technology and ever increasing need for throughput. However, there are certain segments that do not require high data transfer capacities, such as Zigbee, a new low-rate wireless network standard designed for automation and control network. Zigbee aims to be a low-cost and low-power solution for systems consisting of unsupervised groups of devices in houses, factories and offices.

AES is a complex computational algorithm that requires a huge number of resources and power consumption in applications. In recent literature, implementations of the AES algorithm in ASIC and field programmable gate array (FPGA) are reported. However, most of them focus on the performance of high throughput, which are not area-efficient and result in high power consumption. Whereas, in low-cost and low-rate environments, such as Zigbee system, power consumption and hardware cost are more important factors than data transfer capacities.

In this chapter, the basic building blocks used and implemented in my design are discussed, in which an area-optimized very large scale integration (VLSI) implementation of the AES coprocessor is executed. As a solution, the architectures of optimized SubBytes /InvSubBytes and MixColumns/InvMix-Columns are applied, which synthesize the encryption and decryption, and employ the various techniques of resource sharing [5]. My complete design has been implemented and verified using the Verilog HDL Language.

4.2 Top AES Block

This is our top most main module, in which all the blocks and FSM of our design are instantiated. In this section the black box, pin description, internal block diagram and it’s description is described.

4.2.1 Black Box

The Black box of top AES block is shown in figure 4.1.

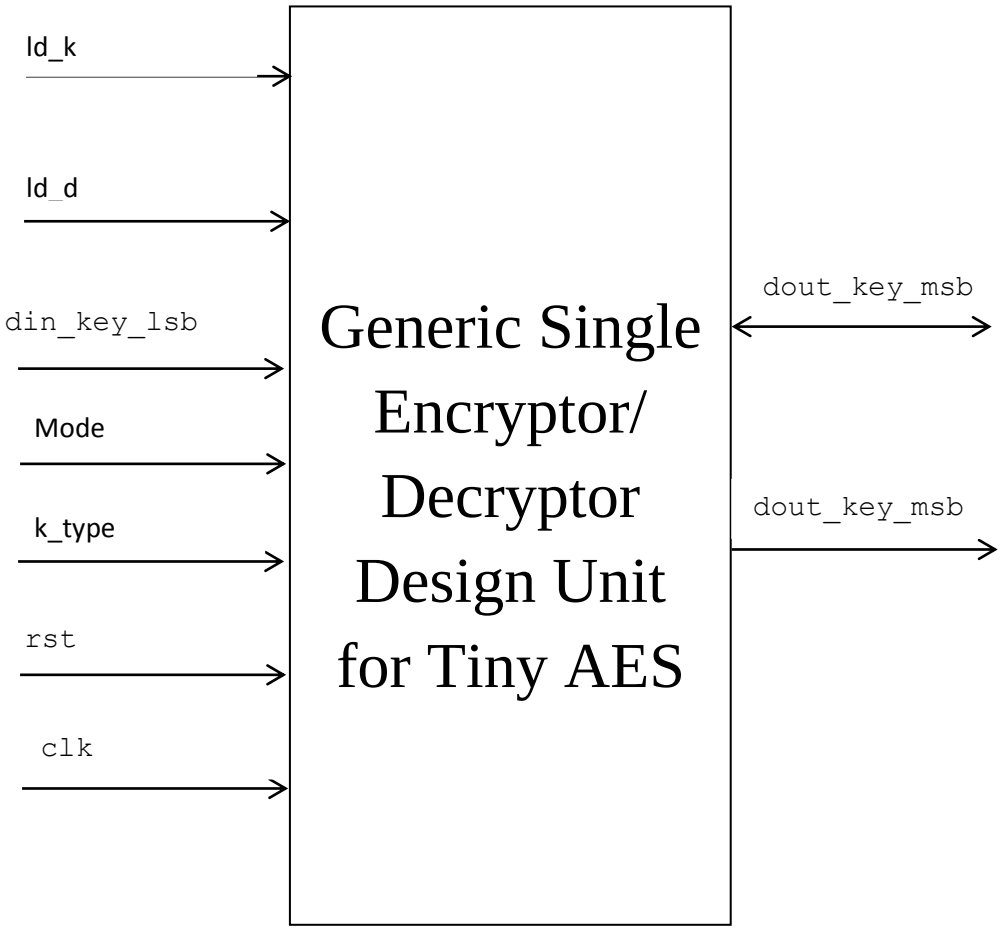


Figure 4.1 Black Box of top AES Block

4.2.2 Pin Description

1.	ld_k	1	I/P	Signal for showing that data present on bus is Valid 256-bit Key
2.	ld_d	1	I/P	Signal for showing that data present on bus is Valid 128-bit Plain/Cipher-Text
3.	din_key_lsb	128	I/P	multiplexed data bus showing i/p plain/cipher-text and LSB key input bits [127:0]
4.	Mode	1	I/P	signal for selecting either encryption/decryption mode
5.	k_type	2	I/P	signal for selecting the type of AES-key
6.	Rst	1	I/P	Power on synchronous reset signal
7.	Clk	2	I/P	global clock
8.	dout_key_msb	1	I/O	multiplexed bidirectional data bus showing o/p cipher/plain-text and key input bus signal for MSB bits [255:128]
9.	Done	1	O/P	signal showing of encryption /decryption has been completed and valid o/p cipher /plain-text is present on o/p data bus

Table 4.1 Pin Description of Top AES Block

4.2.3 Top level Internal Block Diagram

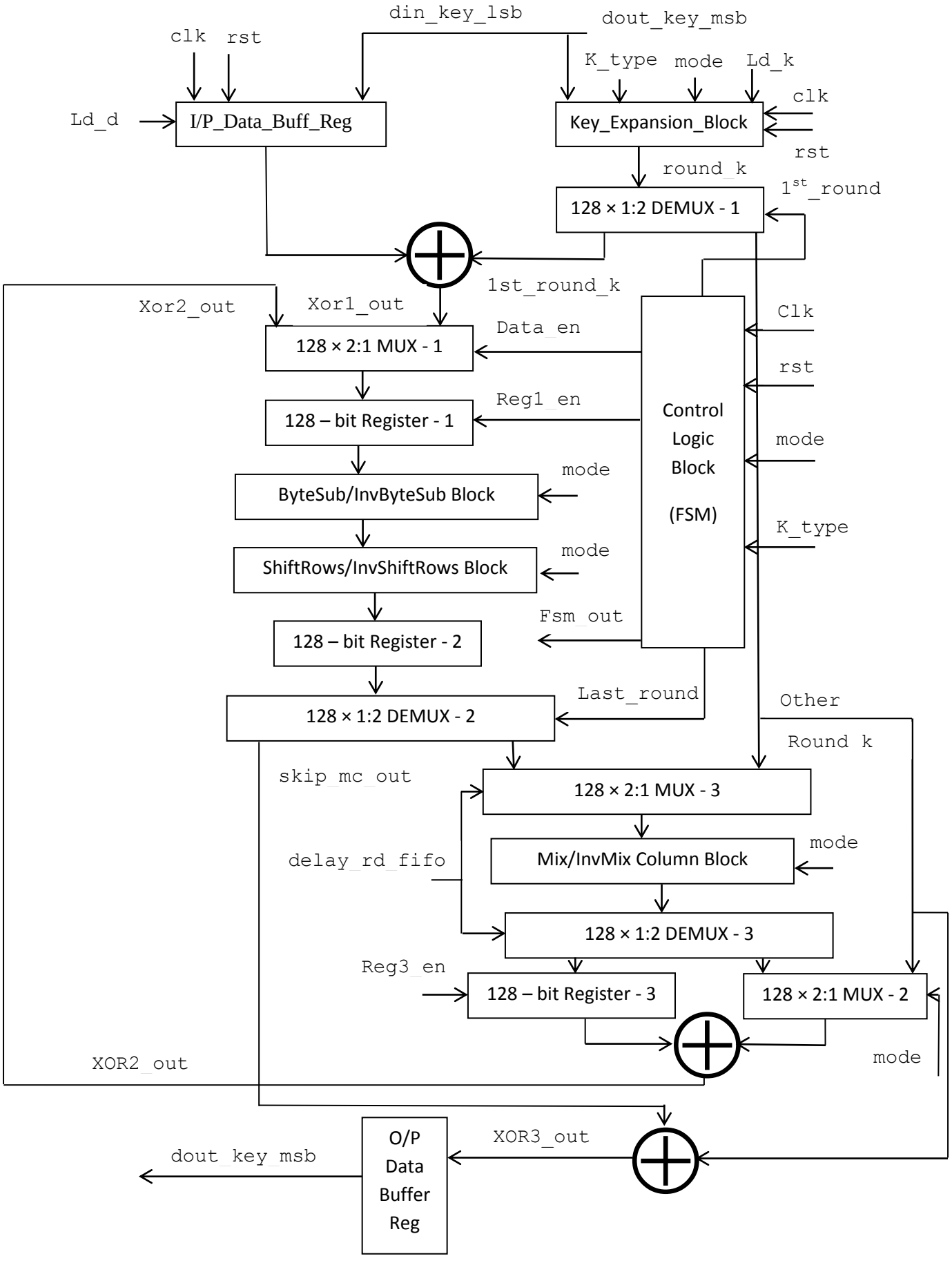


Figure 4.2: Internal Block Diagram of Top Tiny AES Block

4.2.4 Internal Block diagram description of Top AES Block

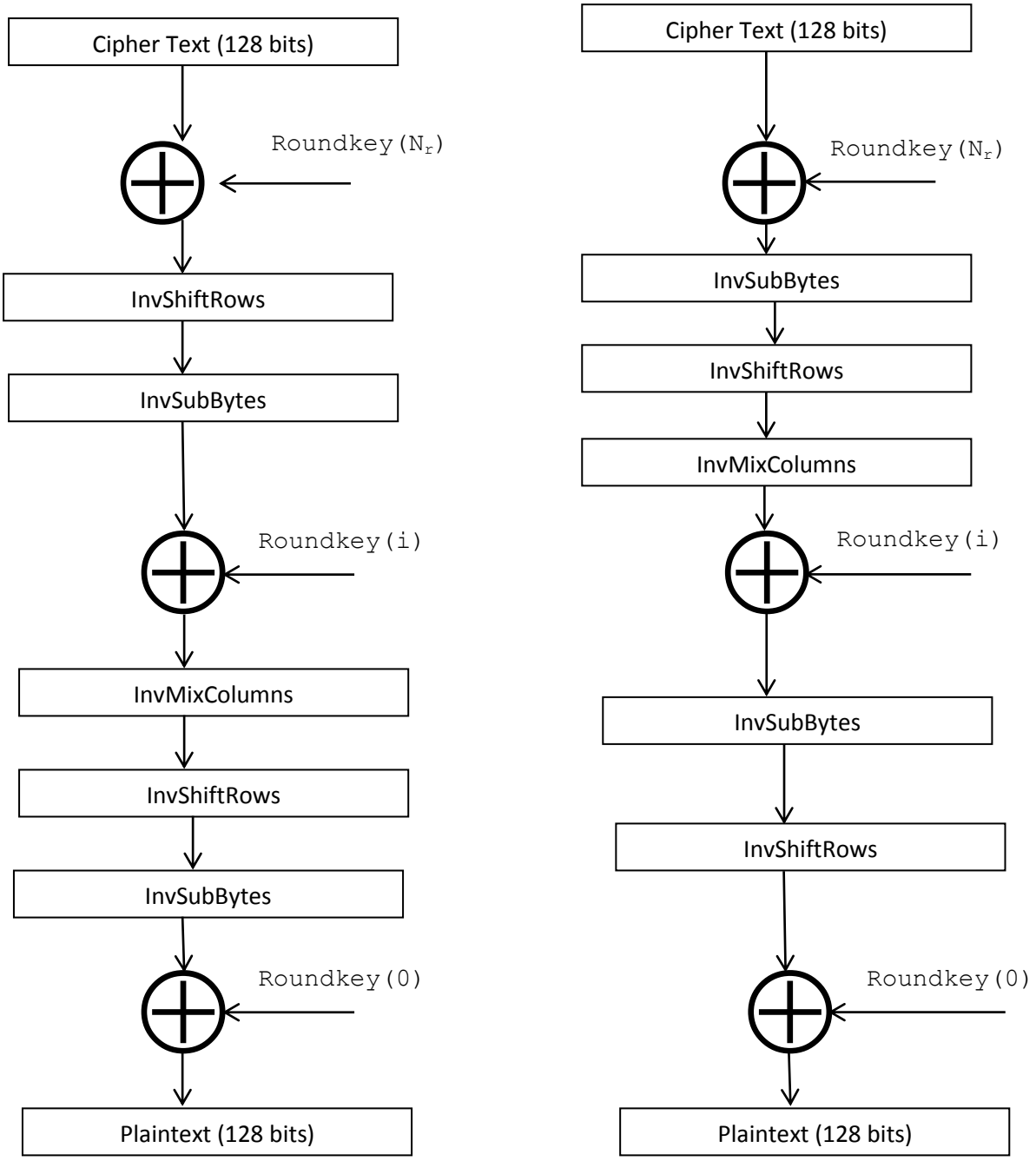


Figure 4.3: AES Straightforward (a) and modified (b) decryption structure

As shown in figure 4.3 (a) the decryption structure can be derived by inverting the encryption structure directly. However, the sequence of the sub-units operations will be different from that in encryption (see Fig. 1). This feature prohibits resource sharing between Encryptor and decryptor. To make maximum resource sharing we need some technique and some sort of re-arrangement of sub-units operations.

It can be observed from the operations involved in the decryption transformations that, the inverse ShiftRow (InvShiftRow) and the InvByteSub can be exchanged without affecting the decryption process. The InvMixcolumn can be moved before the AddRoundKey, provided that the InvMix-Column is applied to the roundkeys before it is added. Taking these into consideration, an equivalent decryption structure can be used as shown in Figure (b). In this figure, the MixRoundkey is the modified roundkey resulting from applying InvMixColumn to the AddRoundKey. The equivalent decryption structure has the same sequence of transformations as that in the encryption structure, and thus, resource sharing between encryptor and decryptor are possible. Figure 4.2 shows the block diagram of Encryption/Decryption core implementation of Tiny AES into a single chip. The operation either of encryption or decryption is selected by user defined control signal called Mode. In the Figure 4.2, *REG1 REG2* and *REG3* are used for providing the pipelining in AES block diagram to increase the clock frequency of our design. Hardware sharing of each sub-unit for Encryption/Decryption core implementation is shown in the respective sections.

Initially the 128-bit input plain/cipher-text is stored in data_buffer_register directly in control with the input ld_data signal. Simultaneously key expansion process is also started. There is a control logic block also, which will generate various control signals for main internal block diagram. Then we will wait until the first_round_key/last_round_key is to be generated from key_exp_block for encryption/decryption respectively. DEMUX-1 will switch the key expansion block output round_key to first_round_key/last_round_key or other round key for first/last round or other rounds, in control with first_round signal. This first/last round_key and input plain/cipher text is XORed to generate the 128-bit xor1_out. MUX-1 is used to select the xor1_out and xor2_out outputs to give its output to Reg1. Then reg1 output (in control with reg1_en) goes through various blocks like ByteSub/InvByteSub and ShiftRows/InvByteSub. In all these blocks encryption/decryption is selected by input signal called mode. This arrangement provide the various resource sharing in different blocks, which resulted in less area/digital logic resources. ShiftRows/InvByteSub output is given to Reg2 Register, which is used to provide the inner-pipelining.

Now according to AES algorithm, for last round Mix/InvMixColumn block has to be skipped for both main algorithm flow and round key flow. This is done by DEMUX-2, which will switch the Reg2 output directly or indirectly through Mix/InvMixColumn block for XORing in case of last or other round respevtively.

As it is mentioned before that for maximum resource sharing fig (b) is used, in which for decryption mode the MixRoundkey is the modified roundkey resulting from applying

InvMixColumn to the AddRoundKey. Initially I instantiate separate InvMixColumn module for this purpose. But later it was realized that InvMixColumn is also used in main algorithm flow and in both cases InvMixColumn is used at separate interval of time. So here also I have a better chance of resource sharing. So why should I miss this opportunity. MUX-3 and DEMUX-3 (in control with delay_rd_fifo signal) are used to provide this resource sharing of InvMixColumn between main algo flow and MixRoundkey. Main algorithm flow and MixRoundkey flow will be selected for delay_rd_fifo signal to be logic '0' or '1'. Here same Mix/InvMixColumn block is used for both cases, but for MixRoundkey *flow* only InvMixColumn process is required for both encryption and decryption mode. So when providing the control signal to Mix/InvMixColumn block we have to switch between mode signal and logic '0' for selecting the Mix/InvMixColumn process. Now MUX-2 is used to switch the MixRoundkey and AddRoundKey outputs for XORing1 in case of encryption and decryption respectively in control with mode input signal. The main purpose of Reg3 (in control with reg3_en signal) is not only to provide delay for synchronization of MixColumn and expanded round key outputs, but also to provide the inner pipelining for improving the clock frequency.

Now for other rounds (not last round) MUX-2 and Reg3 outputs are XORed to get the xor2_out output, which is again feedback to MUX-1 for next round. Here now as it is mentioned previously that for last round InvMixColumn block has to be skipped for round key flow also. So round key signal is directly fed to XOR-3. Hence finally in case of last round DEMUX-2 output signal called skip_mc_out and direct other_round_key signals are given to MUX-3 to provide the xor3_out signal, which is finally fed to output buffer register. Finally encrypted/decrypted cypher/plain-text output data signal is taken from output data buffer register.

4.3 Key Expansion Block

4.3.1 Black Box

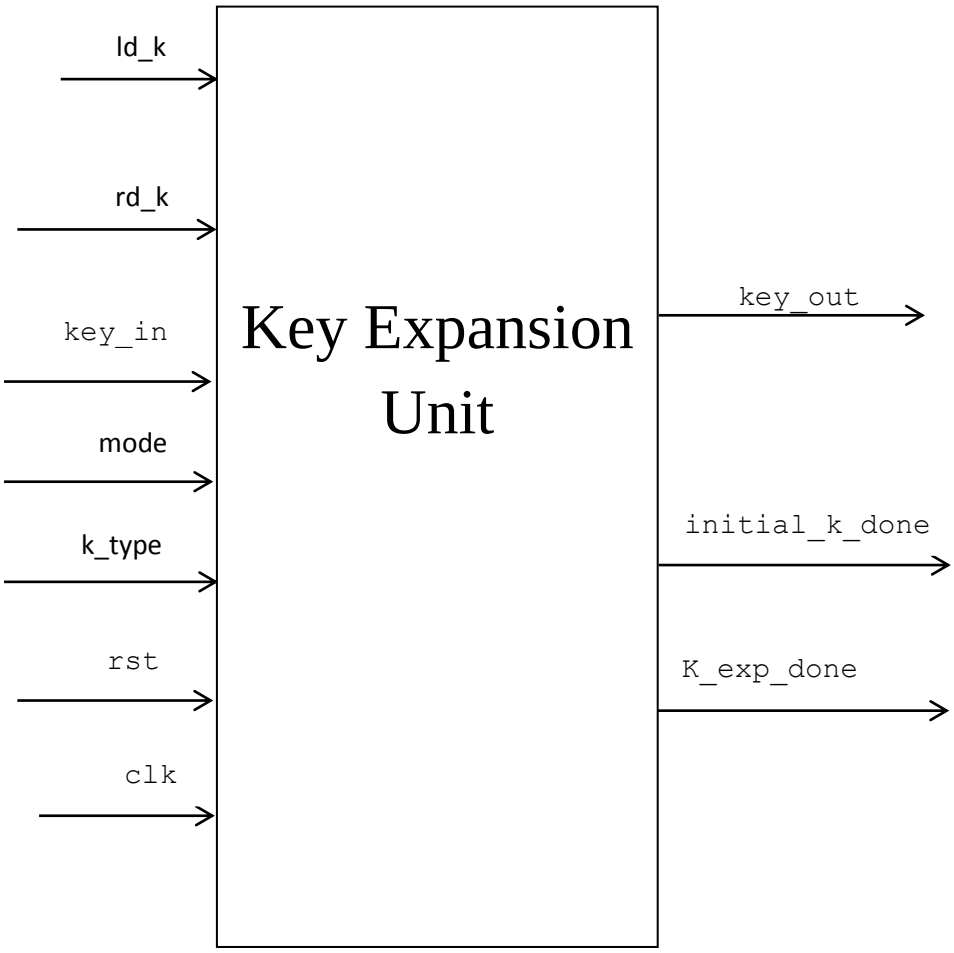


Figure 4.4 Black Box of Key expansion Block

4.3.2 Pin Description of Key Expansion module

S.No.	Name	Bits	Type	Description
1.	ld_k	1	I/P	Signal for showing that data present on din_key_lsb and dout_key_msb buses are Valid 256-bit Key
2.	rd_k	1	I/P	Signal for getting the expanded key from key expansion block
3.	key_in	256	I/P	key input bits
4.	Mode	1	I/P	signal for selecting either encryption/decryption mode
5.	k_type	2	I/P	signal for selecting the type of AES-key
6.	Rst	1	I/P	Power on synchronous reset signal
7.	Clk	2	I/P	global clock
8.	Key_out	1	I/O	Expanded O/P Key
9.	K_exp_done	1	O/P	signal showing that expansion process has been completed and valid key o/p is present on key_out bus

Table 4.2: Pin Description of Key Expansion Block

4.3.3 Internal Block Diagram of Key Expansion module

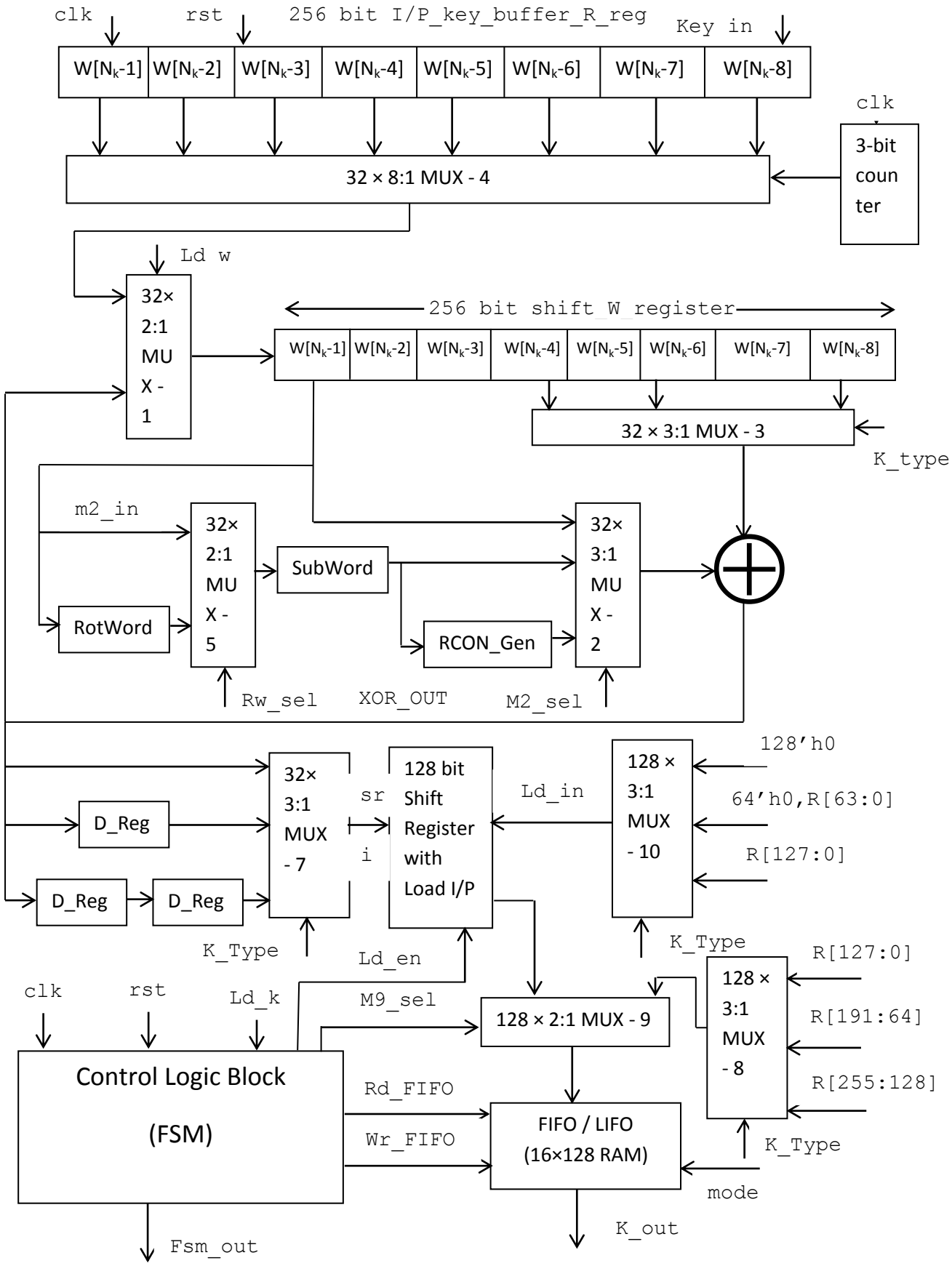


Figure 4.5: Internal Block Diagram of Key Expansion Block

4.3.4 Internal Block Diagram Description of Key Expansion module

The KeyExpansion unit is shown in Figure 4.5. The architecture is almost same as the KeyExpansion unit described in [3]. The basic principle of this block is that to generate the expanded key according to the type of key selected by k_type signal and according to key expansion algorithm and store the expanded key in a 16×128 bit FIFO/LIFO dual port RAM. FIFO/LIFO is used for reading the dual-port RAM from starting/end while encryption/ decryption respectively. For decryption operation the KeyScheduler should generate last RoundKey first using initial cipher key. To ensure this condition LIFO (last in first out) is also combined with FIFO (first in first out) to read the dual port RAM from end while decryption. R is a 256-bit register to store initial key. 256-bit key is shared by two data buses. Din_key_lsb and Dout_key_msb data buses contains the LSB and MSB-bits of 256-bit key respectively, which are also shared by data I/P and data O/P buses. Internally these two data buses are combined and initially the R-Register is loaded by the combined 256-bit key. For 128 or 192 bit key the LSB of R contains the actual 128 or 192 bit key and the remaining MSB-bits contains the 0's.

W is a 256-bit shift register, which take the 32-bit data at a time from MUX-1 and shift all the bits of register by 32-bits. At starting the W-Shift Register is loaded with initial key by continuously shifting the W-Register with the help of switching the MUX-4, which is selected by Mux-4 Counter. In this way LSB and MSB-bits of R and W-Shift-Registers are interchanged. Hence the LSB bits of W-Register contains the $W[N_k-4]$,$W[N_k-6]$,$W[N_k-8]$ words, which are alternatively selected by MUX-3 for different combination of key-types, while XORing the previous words with the new-one.

Now according to key expansion algorithm

$W[i] = W[i-N_k]$ XOR temp

Where, temp = SubWord(RotWord(W[i-1])) XOR RconGen[i/N_k] , for (i mod N_k = 0)

SubWord(W[i-1]), for (N_k>6 and i mod N_k = 4)

W[i-1], elsewhere

Here in our project (i mod N_k = 0 and i mod N_k = 4) is ensured by mod counter. The total number of rounds are counted by the key-round-counter. MUX-2 will select different combinations of temp signal according to the m2_sel signal in control with the control logic block. Various functions SubWord, RotWord and RconGen are executed by the respective blocks and modules, which are explained in separate sections of this report. MUX-5 is used

to decide, whether the RotWord Block is to be performed on W[i-1] word. It will also ensure to resource sharing of SubWord block between the above mentioned two cases.

Next the Outputs of MUX-2 and MUX-3, m2_out(i.e. temp) and m3_out(i.e. W[i-N_k]) are XORed to get the XOR_out word. Now this XOR_out word is not only passed to W-shift register via MUX-1 as a 32-bit serial input, which will work as a W[i-1] word on next round word of key expansion, but also sampled to form the 128-bit input for FIFO/LIFO Block with the help of fifo_shift_Register. A 2-bit fifo_wr_counter is also maintained to continuously generate the wr_fifo signal for writing the fifo_shift_register output to FIFO/LIFO Dual port Ram Block.

According to AES algorithm the 128-bit input cipher-text is XORed with the 128-bit input key for first round at starting. For 192-bit key the remaining 64-bits of keys are combined with the next words of expanded key words. MUX-8 and MUX-9 are used to write the initial first MSB 128-bits of input key (i.e. R[127:0], R[191:64], R[255:128]) into fifo according to k_type and m9_sel control signals. MUX-10 is used to load the fifo_shift_reg with the remaining bits of input key for the initial first value of key. MUX-7 is used for selecting xor_out directly or delayed output for synchronization with first time loading the fifo-shift_reg or fifo. The remaining LSB 64-bits of 192-AES and 128-bits of 256-AES are initially loaded into fifo_shift_reg at starting with the help of MUX-10, while serial shift input of this register is provided with various directly or delayed versions of XOR_out signal with the help of MUX-7 to ensure the safely storing of initial starting value of input key into the FIFO/LIFO dual port RAM Block.

All the control signals (like enable, reset and select signals of various counters and MUXes, ld_en signals of fifo_shift_reg, W and R-registers etc.) are generated by a control unit (FSM). For reading the FIFO/LIFO Dual port RAM, rd_fifo signal is given from top_aes_block control logic FSM. Reading and writing to fifo can be done simultaneously while encryption for fast execution of encryption process.

This architecture takes *4* clock cycles to generate single round key. Hence for N_r number of rounds, the total number of clock cycles required to generate complete round key is = $4\times N_r$ + 11 for both encryption and decryption. A maximum of $4\times14 + 11 = 67$ clock cycles are required to generate complete round key for 256 bits key.

4.3.5 Rcon Generation Block of Key Expansion module

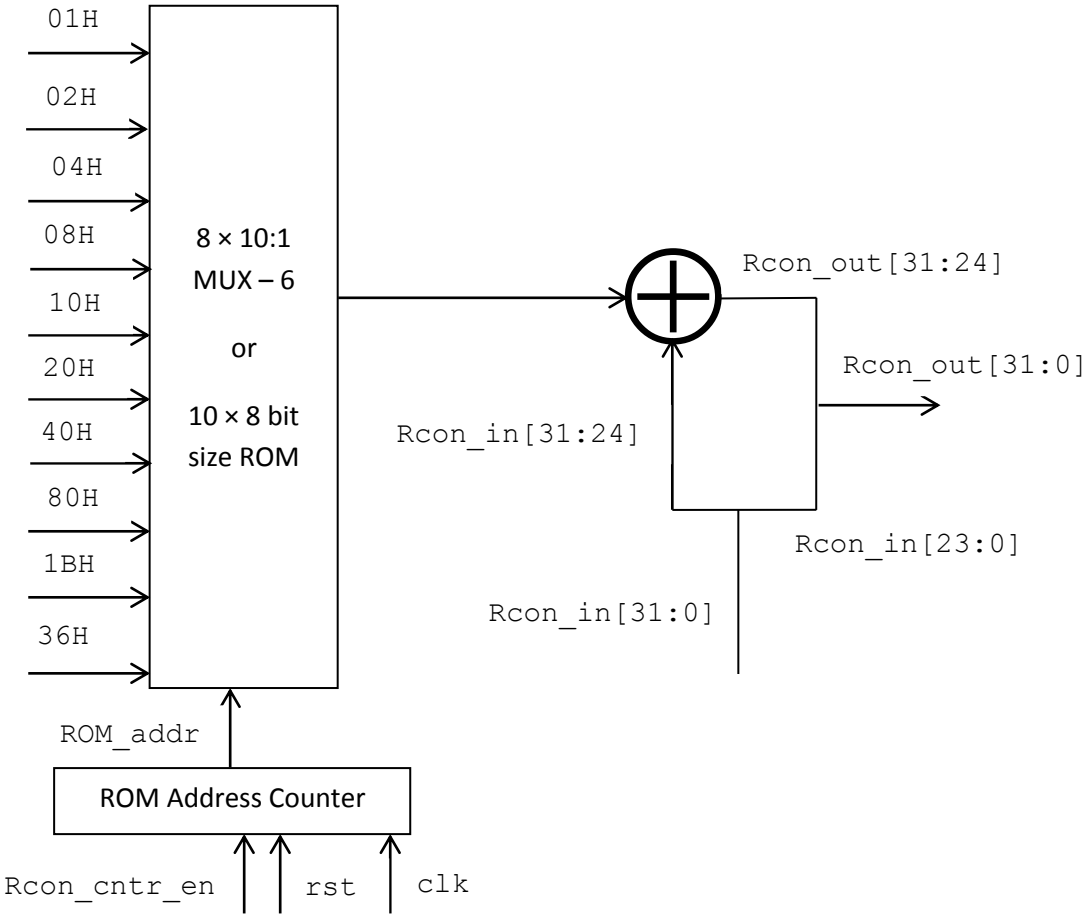


Figure 4.6 RconGeneration Block Diagram

RconGeneration Block diagram is shown in figure 4.6. As shown in figure 8 × 10:1 MUX–6 or (10 × 8 sized ROM) is used to select the particular value of Rcon[i] for various ranges of k_types. Various ROM addresses are generated with the help of ROM_address_counter, which is enabled/ disabled in control with FSM for various types of keys. The higher(MSB)-bits of 32-bits input Rcon_in signal are XORed with Rcon generated output of ROM and then combined with the lower 24 bits of Rcon_in to give the 32-bits Rcon_out output.

4.3.6 Various Counters used in Key Expansion Block

All counters are enabled and disabled or reset with the control of our control logic block (fsm).

(i) RCON_addr_cntr (4-bit) :- Used for generating the ROM Addresses for RconGeneration.

(ii) Mode checking counter (4-bit): - It is used for finding the mode of i

Where mode counter = 000-011, for $N_K = 4$
000-101, for $N_K = 6$
000-111, for $N_K = 8$

(iii) Key Round counter (3-bit) :- Used for counting the no. of rounds in key expansion block.

Where k_round_ counter = 0000-1010, for $N_K = 4$
0000-0111, for $N_K = 6$
0000-0110, for $N_K = 8$

(iv) Write FIFO counter (2-bit) :- Used for issuing a wr_fifo signal from FSM after every 4-clock cycles .

(v) MUX4 Counter (3-bit) : Used for initially load the W-Shift_Register with the help of switching the MUX-4 alternatively from R-register

4.4 SubBytes Block

In this section, we consider three different approaches for the Hardware realization of the SubBytes and InvSubBytes operations. In our design, only two s-boxes will be instantiated (one for SubBytes and one for InvSubBytes) and are shared between the encryption/ decryption and the key expansion modules. In our design we have implemented two methods (LUT and Galois Field methods) and compare the results for area optimization.

4.4.1 Lookup Tables (LUT) Method

In this method, the 265 byte values of s-box are stored in a ROM lookup table. Another 256 byte lookup table is required to store the inverse s-box. For FPGA implementations, this is usually a convenient option since block ROMs are already available on almost all FPGA chips. However, in applications where several systems are implemented on the same chip (as for ASIC designs), and the ROMs are utilized by the other system modules, the following two implementation options can be considered. In the proposed design, the cost and area are more important than speed, hence we employ the composite field calculation method to reduce hardware cost. [6]

4.4.2 Boolean functions optimization (BFO)

In this method, the s-box (similarly, the inverse s-box) coordinate functions are treated as 8-bit Boolean functions and they can be jointly optimized. In this implementation, the freely available Minilog program can be used that utilizes the Espresso heuristic logic minimizer to obtain an efficient two level Boolean function implementation of the SubBytes and InvSubBytes operations. Similar approach was proposed in [6].

4.4.3 Galois field (GF) evaluation of the multiplicative inverse

The ByteSub transformation is a non-linear byte substitution, also called S-box. The S-box is invertible and consists of the following two operations:

- Inversion in the GF(2^8) field, modulo the irreducible polynomial $m(x)=x^8+x^4+x^3+ x + 1$.
- Affine transformation defined as: $Y = AX^{-1} + B$, where A is an 8 × 8 fixed matrix and B is an 8 × 1 vector-matrix.

Using this approach, the composite field representation is used to reduce the computation cost of the s-box. Recall that SubByte operation corresponds to evaluating the multiplicative inverse of a Galois field element over GF(2^8) followed by an affine transformation (AT). Thus the major computation in this operation is the evaluation of the multiplicative inverse of an element in the finite field GF(2^8) [3]. In order to reduce the cost associated with this operation, several authors have designed AES s-boxes based on composite field techniques. The techniques use three-stage strategy :

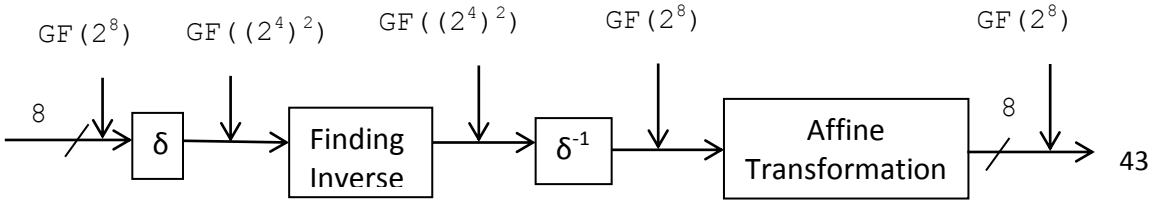
- Map the element $x \in GF(2^8)$ to a composite field $F \in GF((2^4)^2)$ using an isomorphic function δ.
- Compute the multiplicative inverse over the field $F \in GF((2^4)^2)$.
- Finally map the computation result back to the original field using the inverse function δ^{-1}.

We will implement the structure shown in Figure 4.7. where δ denotes an isomorphism that maps an element x in GF(2^8) to an element in GF($(2^4)^2$). The function δ is defined as δ(x) = T.x, where T denotes the following transformation matrix: [4]

$$T = \begin{vmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{vmatrix}$$

According to Fig. 4.7 the problem of calculating the inverse in GF(2^8) is now translated to GF(2^4), where the multiplication is performed modulo an irreducible polynomial with degree two in GF(2^8). Denoting the irreducible polynomial as $x^2 + Ax + B$, then the multiplicative inverse for arbitrary polynomial ax + b is given by [4]:

$$(ax + b)^{-1} = a(a^2B + abA + b^2)^{-1}x + (b + aA)(a^2B + abA + b^2)^{-1} \qquad (4.1)$$



X

Y

Figure 4.7: S-box Implementation Using GF Arithmetic

The irreducible polynomial $x^2+x+\lambda$ is used, i.e., we set A = 1, B = λ and $\lambda = (1100)_2$ [4]. After conversion from GF(2^8) to GF(2^4), the hardware design of Equation (4.1) is shown in Fig. 4.8 where x^2 is the square in GF(2^4), xλ is the multiplication with constant λ, x^{-1} is the inverse in GF(2^4) and can be calculates by the same method in GF(2^2), δ^{-1} is the inverse isomorphism mapping to GF(2^8), and × denotes the multiplication operation in GF(2^4).

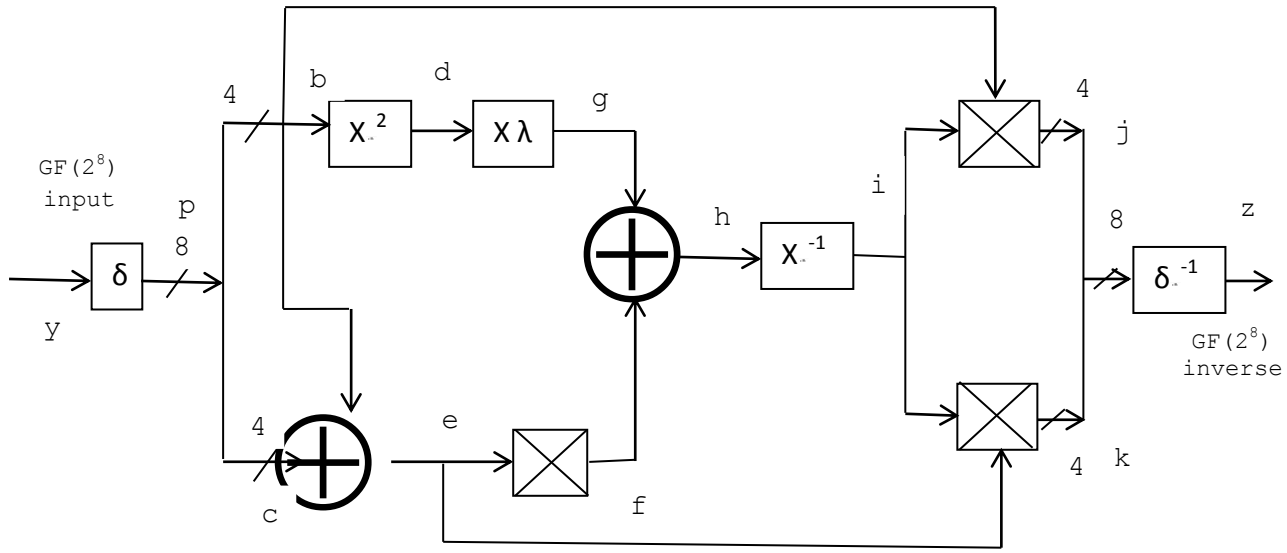


Figure 4.8: GF Multiplicative Inverse Module

In the Decryption process, the affine transformation is executed prior to the inverse. Fig. 4.9 shows this step.

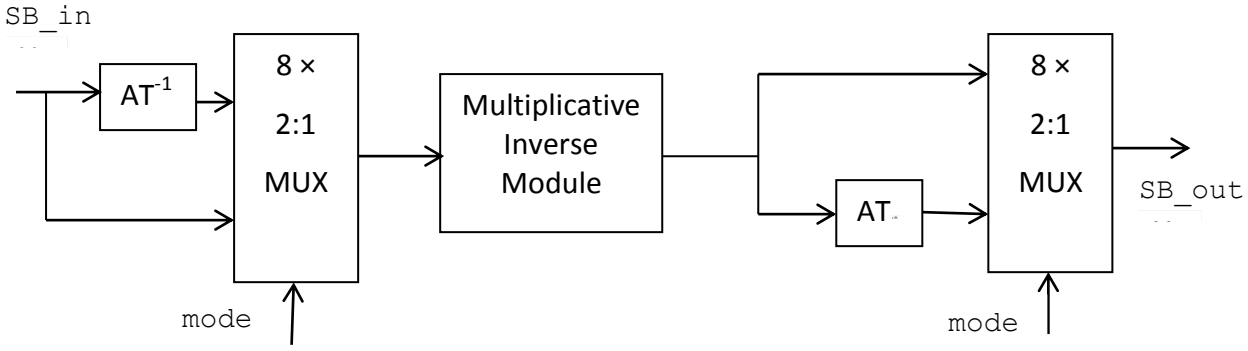


Figure 4.9 Resource Sharing for SubBytes and InvSubBytes

The multiplicative inverse of the polynomial is found using the architecture shown in Figure 4.8 [7]. The architecture consists of addition, multiplication and squaring which can be implemented separately as given in the following sections. Primarily all the operations are conducted in GF(2^4) resulting into lowering the complexity.

The upper nibble of f(x) is squared and then multiplied with λ Lower and upper nibble of f(x) are XORed and multiplied with lower nibble in GF(2). As shown in the figure 4.8 multiplicative inverse in GF(2^4) is computed and used to get upper and lower nibble of final multiplicative inverse in GF(2^8).

(a) Squaring in GF(2^4)

Squaring in GF(2^4) can be performed by converting the GF(2^4) elements into GF(2^2) with the help of irreducible polynomial, which yields to simple Boolean expression as given in equation (4.2).

$$d_3 = b_3$$
$$d_2 = b_3 \oplus b_2$$
$$d_1 = b_2 \oplus b_1$$
$$d_0 = b_3 \oplus b_1 \oplus b_0 \quad (4.2)$$

Where d is the element of GF(2^4).

(b) Multiplication with constant λ

The polynomial ’d’ obtained after squaring 'b' is to be multiplied with the constant λ where λ = {1 1 0 0}. The product is reduced using the polynomial $x^2 = x + \varphi$. This operation can be further simplified and 'g' could be achieved using simple Boolean expressions as given in equation (4.3).

$$g_3 = d_2 \oplus d_0$$
$$g_2 = d_3 \oplus d_2 \oplus d_1 \oplus d_0$$
$$g_1 = d_3$$
$$g_0 = d_2 \quad (4.3)$$

(c) Multiplication in GF(2^4)

The multiplication in GF(2^4) for f = e.c; j = e.i; and k = b.i where b, c, d, e, f, g, h, i, j and k are elements in GF(2^4). In order to simplify this step further, a sub block 'm' is used as shown in Figure 4.10, which involves actual multiplication in GF(2^2) using equations (4.4)

$$i_1 = h_1w_1 \oplus h_0w_1 \oplus h_1w_0, \; i_0 = h_1w_1 \oplus h_0w_0 \quad (4.4)$$

Multiplication with φ is done using Boolean expression of equation (4.5).

$$X_1 = y_1 \oplus y_0, \; X_0 = y_1 \quad (4.5)$$

Where x, y, i, h and w are elements in GF(2^2).

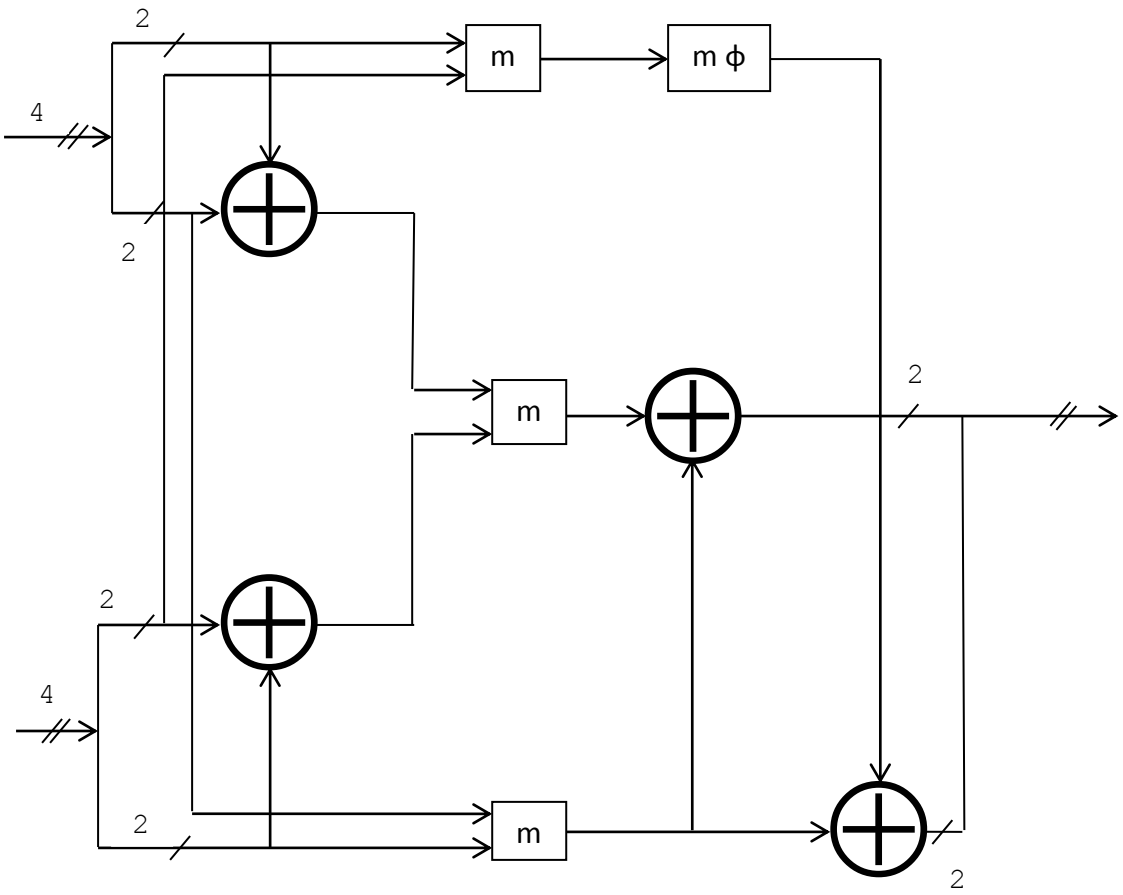


Figure 4.10: Multiplication in GF(2^2)

(d) Multiplicative inverse in GF(2^4)

Using the expressions from [7], the inverse of the individual bits can be computed from equations (4.6).

$$q_3^{-1} = q_3 \oplus q_3 \cdot q_2 \cdot q_1 \oplus q_3 . q_0 \oplus q_2$$

$$q_2^{-1} = q_3 . q_2 . q_1 \oplus q_3 \cdot q_2 \cdot q_0 \oplus q_3 . q_0 \oplus q_2 \oplus q_2 . q_1$$

$$q_1^{-1} = q_3 \oplus q_3 \cdot q_2 \cdot q_1 \oplus q_3 \cdot q_1 \cdot q_0 \oplus q_2 \oplus q_2 \cdot q_0 \oplus q_1$$

$$q_0^{-1} = q_3 . q_3 . q1 \oplus q_3 \cdot q_2 \cdot q_0 \oplus q_3 \cdot q_1 \oplus q_3 . q_1 . q_0 \oplus q_3 \cdot q_0 \oplus q_2 \oplus q_2 . q_1 \oplus q_2 . q_1 . q_0 \oplus q_1 \oplus q_0 \quad (6)$$

4.5 Implementation of MixColumns/InvMixColumns Block

In the AES algorithm, both the MixColumns and InvMixColumns are hardware demanding operations. Various architectures have been proposed for the implementations of the MixColumns and InvMixColumns transformations. By analysing the basic operations employed in MixColumns and InvMixColumns, it is found that the modular multiplier is the vital calculation module and that InvMixColumns is more complicated than MixColumns. To reduce design complexity, the InvMixColumns can be decomposed to share hardware

resources with MixColumns [5]. In our design, the method described in Ref. [13] is adopted that can save the area and reduce the complexity.

By this approach, as shown in Fig. 4.11(a), byte-level resource sharing is realized for both parallel and serial InvMixColumns decomposition. The architecture based on the serial InvMixColumns decomposition with byte-level resource sharing is the most area-efficient solution.

The MixColumn transformation multiplies each column of the State by polynomial c(X). The c(X) is defined as follows: $c(X) = \{03\}X^3 + X^2 + X + \{02\}$ (4.7)

The InvMixColumn transformation is the inverse of the MixColumn operation. InvMixColumn multiplies each column of the State by

$$d(X) = \{0B\}X^3 + \{0D\}X^2 + \{09\}X + \{0E\} \quad (4.8)$$

where $d(X) = c^{-1}(X)$

From (4.7) and (4.8), we can see that coefficients of d(X) are more complex than coefficients of c(X). As a result, hardware implementing AES decryption is larger and slower than for encryption. In order to reduce hardware cost, the InvMixColumn can be decomposed to share logic resources with MixColumn. Since both functions transform 32-bit words, we will call this decomposition the word-level resource sharing. There exist two possible decompositions of InvMixColumn: parallel and serial. In addition to word-level sharing, resources can be shared on a byte level and on a bit level as well. These approaches will be discussed in Sections 4.5.1 and 4.5.2.

4.5.1 Word-Level Resource Sharing

(i) Parallel InvMixColumn Decomposition: Parallel InvMix- Column decomposition was first proposed by J. Wolkerstorfer. It is based on the observation that d(X) can be expressed using c(X) in the following way:

$$d(X) = c(X) + e(X) \quad (4.9)$$

where e(X) is an extension polynomial defined as

$$e(X) = \{08\}X^3 + \{0C\}X^2 + \{08\}X + \{0C\} \quad (4.10)$$

(ii) Serial InvMixColumn Decomposition: The inverse d(X) of the polynomial c(X) is given by the formula

$$d(X) = c^{-1}(X) = c^3(X) \quad (4.11)$$

which suggests that the InvMixColumn operation can be realized by repeating MixColumn three times. For hardware implementations (4.11), can be expressed as

$$d(X) = c(X) \cdot f(X) \quad (4.12)$$

where, $f(X) = c^2(X) = \{04\}X^2 + \{05\}$ (4.13)

Hence, the InvMixColumn function can be implemented using the MixColumn function and the f(X) polynomial. Comparing c(X) and f(X) we see that f(X) has much simpler implementation than c(X) because two of its coefficients are zero.

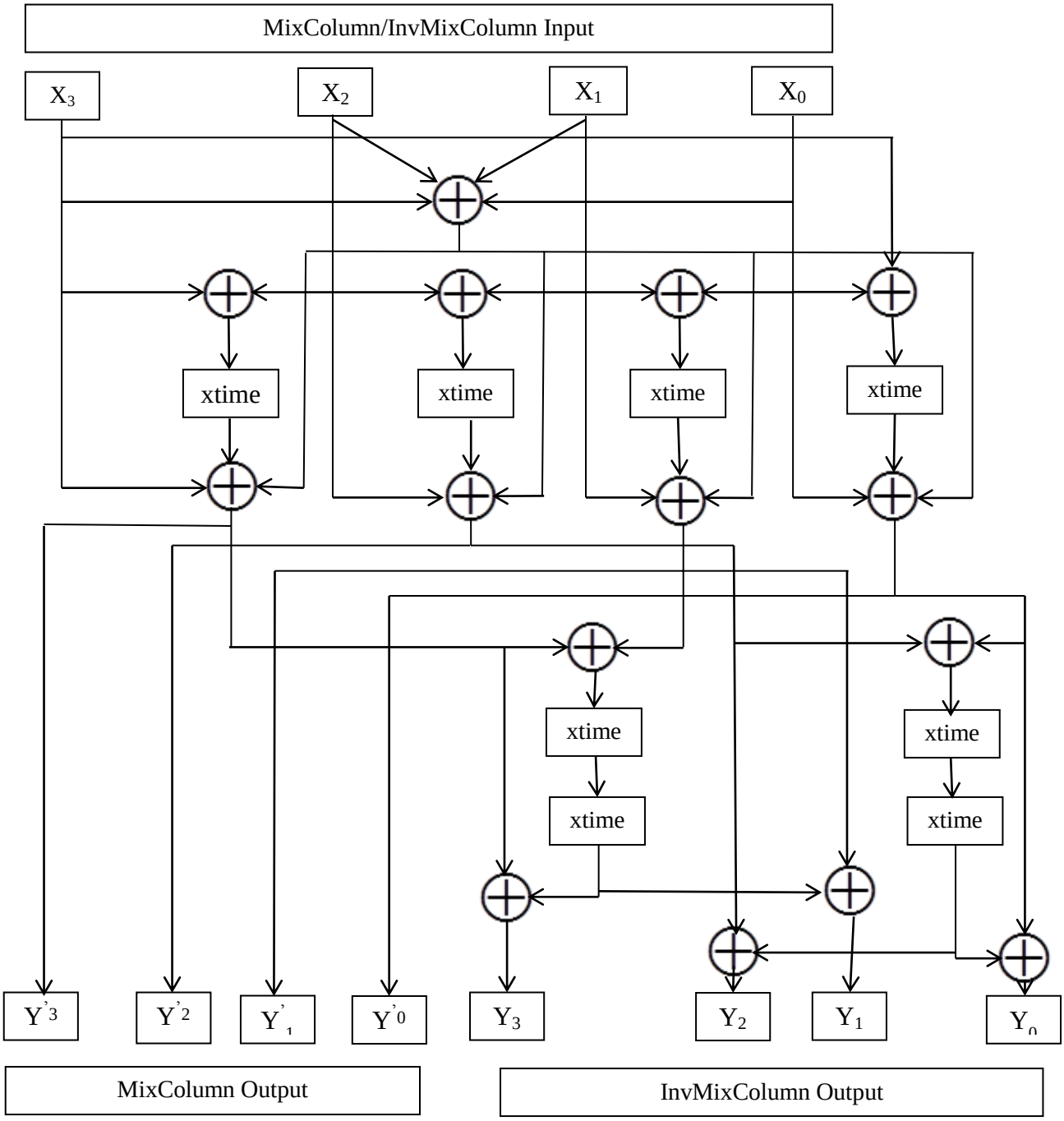


Figure 4.11 (a): MixColumn and InvMixColumn implementation based on InvMixColumn serial decomposition and byte-level resource sharing

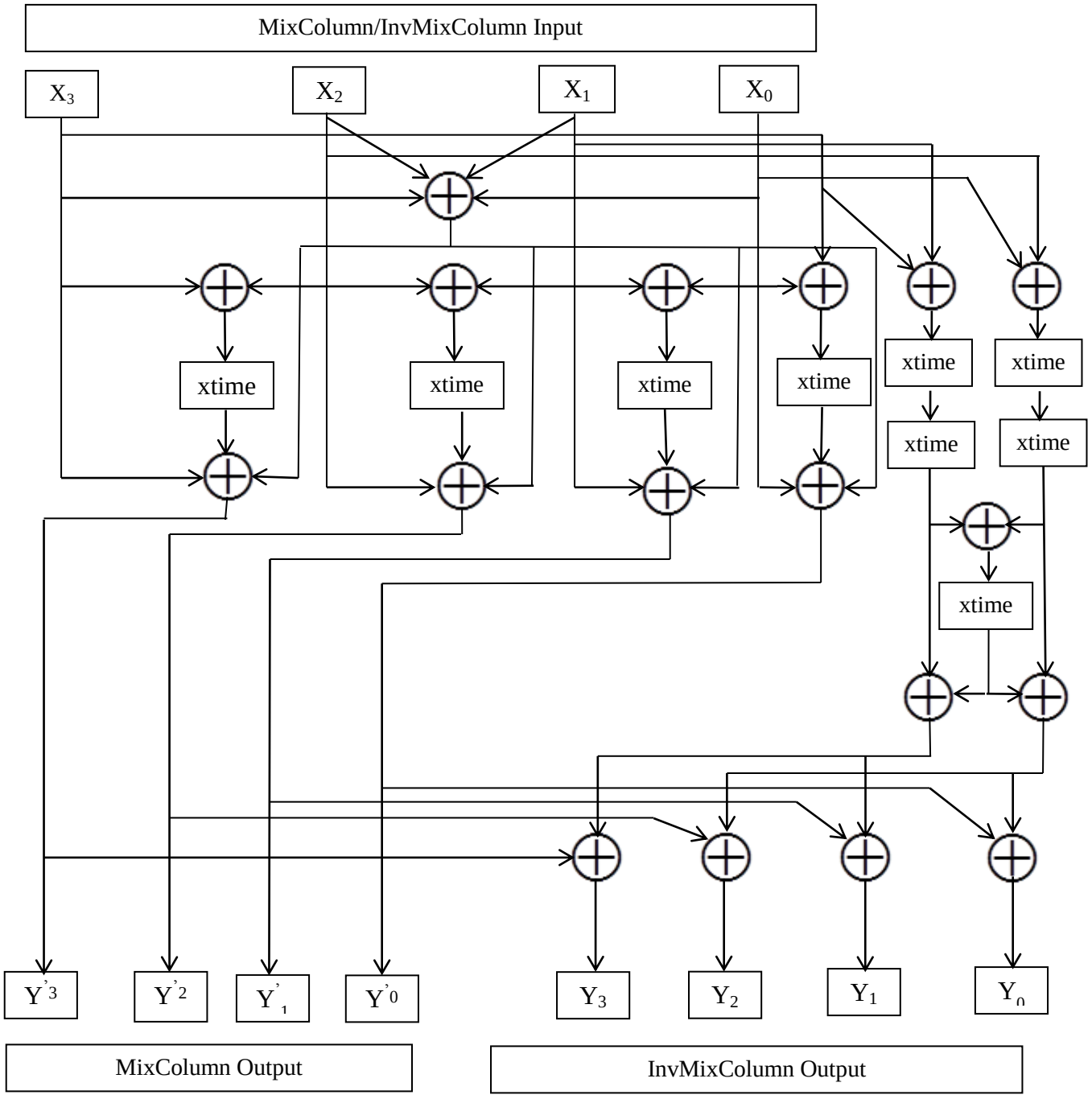


Figure 4.11 (b): MixColumn and InvMixColumn implementation based on InvMixColumn parallel decomposition and byte-level resource sharing

4.5.2 Byte-Level and Bit-Level Resource Sharing

Multiplication in algebraic fields is distributive over addition. This property enables byte-level resource sharing. Bit-level resource sharing can further improve logic reduction and hardware compilers usually do it efficiently.

(i) Byte-Level Resource Sharing in MixColumn Implementation:

The first byte of the MixColumn implementation based on byte-level resource sharing (see dashed rectangles in Fig. 4.6) is expressed as follows:

$$b_0 = \{02\}a_0 + \{03\}a_1 + a_2 + a_3$$
$$= (a_0 + a_1 + a_2 + a_3) + \{02\}(a_0 + a_1) + a_0 \quad (4.14)$$

The bold line in the dashed rectangles in Fig. 4.6 shows that the term $a_0 + a_1 + a_2 + a_3$ can be shared by all four bytes of the MixColumn function. Note, that both addition and subtraction operations in GF(2^8) are realized by the bit-wise addition modulo 2 (XOR). We use the symbol ($\oplus$) for this operation in Fig. 4.6.

(ii) Byte-Level Resource Sharing in InvMixColumn Implementation:

Byte-level resource sharing is possible in both serial and parallel InvMixColumn decompositions. In the serial decomposition, functions c(X) and f(X) have to be optimized separately. Since c(X) corresponds to the MixColumn function, it can be implemented in a way described in Section 4.3.2.1. For the f(X) function implementation,

based on the byte-level sharing, we propose expression of the first byte in the following way:

$$b0 = \{05\}b`_0 + \{04\}b`_2 = \{04\}(b`_0 + b`_{2)} + b`_0 \quad (4.15)$$

where $b`_0$ and $b`_2$ are the first and the third output bytes of the Mix-Column function. It can be seen [bold lines in the lower part of Fig. 4.11 (a)] that the first and the third byte of the f(X) function can share the term $\{04\}(b`_0 + b`_2)$ from the previous equation. The same approach can be used in the expression for the second and the fourth byte.

In the parallel InvMixColumn decomposition, the polynomial c(X) and the extension polynomial e(X) can be optimized either jointly or separately. The first byte of the InvMixColumn function is given as follows:

$$b_0 = \{0E\}a_0 + \{0B\}a_1 + \{0D\}a_2 + \{09\}a_3$$
$$= \{\{02\}(a_0 + a_1) + a_1 + a + a_3 + \{04\}(\{02\}(a_0+a_1) + \{02\}(a_2+a_3)+(a_0+a_2)) \quad (4.16)$$

where the term $\{02\}(a_0 + a_1) + a_1 + a_2 + a_3$ represents the first byte of the MixColumn function, and the term $\{04\}(\{02\}(a_0+a_1)+ \{02\}(a_2 + a_3) + (a_0 + a_2))$ represents the first byte of the extension function. Note, that the term $\{02\}(a_0+a_1)$ can be shared between the first output byte of the MixColumn function and the first byte and the third byte of the extension function.

In order to further reduce the area of the InvMixColumn function, we propose another solution based on the parallel InvMixColumn decomposition. In this case, the first byte of the InvMixColumn is expressed as follows:

$$b_0 = \{0E\}a_0 + \{0B\}a_1 + \{0D\}a_2 + \{09\}a_3$$

$$= (a_0 + a_1 + a_2 + a_3) + \{02\}(a_0 + a_1) + a_0$$
$$+ \{02\}(\{04\}(a_0 + a_2) + \{04\}(a_1 + a_3)) + \{04\}(a_0 + a_2) \quad (4.17)$$

where the term $(a_0+a_1+a_2+a_3)+\{02\}(a_0+a_1)+a_0$ represents the first byte of the MixColumn function, and the term $\{02\}(\{04\}(a_0 + a_2)+\{04\}(a_1+a_3))+\{04\}(a_0+a_2)$ represents the first byte of the extension function. Thus, this configuration enables concurrent Mix- Column and InvMixColumn implementation. It can be seen that all four output bytes of the extension function share the term $02\}(\{04\}(a_0+ a_2) + \{04\}(a_1 + a_3))$. Furthermore, the terms $\{04\}(a_0 + a_2)$ and $\{04\}(a_1 + a_3)$ are shared by the first and the third output bytes and by the second and the fourth output bytes, respectively, as shown in Fig. 4.11(b).

In Fig. 4.11, xtime is a function, which denotes the multiplication of incoming byte by {02} (i.e. multiplication of GF polynomial by x) and it can be achieved by left shift the incoming byte by 1 bit and the LSB is replaced by 0. Then the MSB is compared if it is (0), then the left shifted byte is the result. If it is (1) then the result is the shifted left value after XORed it with the reduction polynomial {1b} which is the finite field used in AES.

4.6 Implementation of ShiftRows/InvShiftRows Block

In the proposed design, no optimization has been performed on ShiftRows /InvShiftRows because it does not require any hardware logical resource, as it is only the re-scheduling or rearrangement of the bytes inside each row of the state with the sequence shown in the Fig. 4.12. The first row will not be shifted. The bytes in second row are cyclically shifted by one byte position to the lift, and so on. The inverse shift rows are similar as before except that the circulation will be in the right direction.

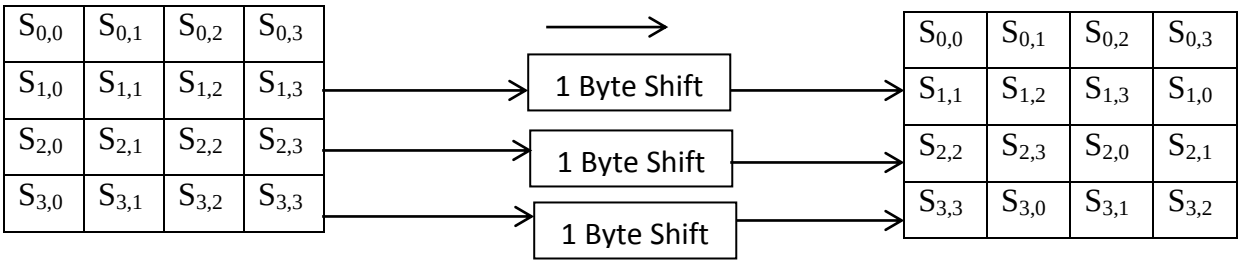


Figure 4.12: The shift row operation

Chapter 5

Simulation and Synthesis of Tiny AES

In this chapter, I will discuss the simulation and synthesis of different architectures and various blocks of the Tiny AES.

5.1 Verification Strategy

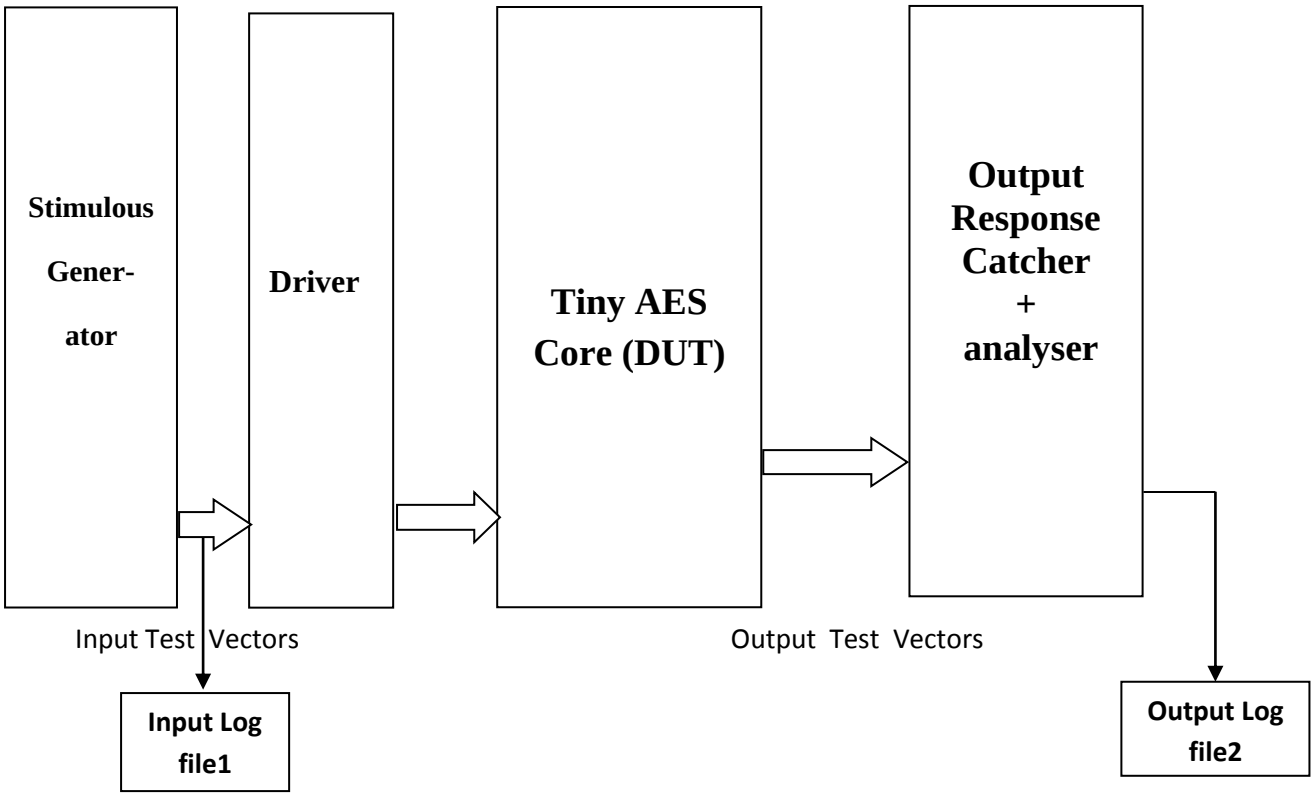


Figure 5.1: Block Diagram of Test Bench

As shown in figure above, Test-Bench have following blocks: -

- Stimulus Generator
- Driver
- Tiny AES Core (DUT)
- Output Response Catcher and Analyser
- Logfile1
- Logfile2

Description of blocks is as following: -

(i) Stimulus Generator:

This Block generates input vectors for Tiny AES core. This block generate clk, Reset, mode, k_type, 128-bit plain/cipher text (input data to be encrypted/decrypted) and 128, 192 or 256 bit key input. The input data is stored in log file1.

(ii) Driver:

The driver translates the stimuli produced by the generator into the actual inputs for the design under verification.

(iii) Output Response Catcher and Analyser:

This block receives the 128-bit encrypted/ decrypted cipher/plain text data from the DUT along with the control signal called 'done', which indicate that encryption process has been completed and the data present on bidirectional bus is the actual encrypted/decrypted output data.

(iv) Logfile2:

This file contains the received cipher/plain text data encrypted/ decrypted by our Generic Encryptor/Decryptor core of Tiny AES (DUT).

5.2 Various Test cases

Sr. No.	Test Scene	Test Case		Criteria	Implemented
1.	Reset (of all blocks including top block)		Power on reset	Pass	Yes
2.	ByteSub Block			Pass	Yes
3.	MixColumn Block			Pass	Yes
4.	ShiftRows Block			Pass	Yes
5.	Key Expansion Block Test cases	2.1	K_type = "00" (AES-128)	Pass	Yes
		2.2	K_type = "01" (AES-192)	Pass	Yes
		2.3	K_type = "10" (AES-256)	Pass	Yes
6.	Encryption Test cases (mode = '1')	2.1	K_type = "00" (AES-128)	Pass	Yes

		2.2	K_type = “01” (AES-192)	Pass	Yes
		2.3	K_type = “10” (AES-256)	Pass	Yes
7.	Decryption Test cases (mode = ‘0’)	3.1	K_type = “00” (AES-128)	Pass	Yes
		3.2	K_type = “01” (AES-192)	Pass	Yes
		3.3	K_type = “10” (AES-256)	Pass	Yes

Table 5.1: Various Test cases

5.3 Output Waveforms for various test cases

I have used Modelsim 10.1 Student edition for simulation. In this section the simulation results in turns of output waveforms for various test cases are shown.

(i) ByteSub Block test case

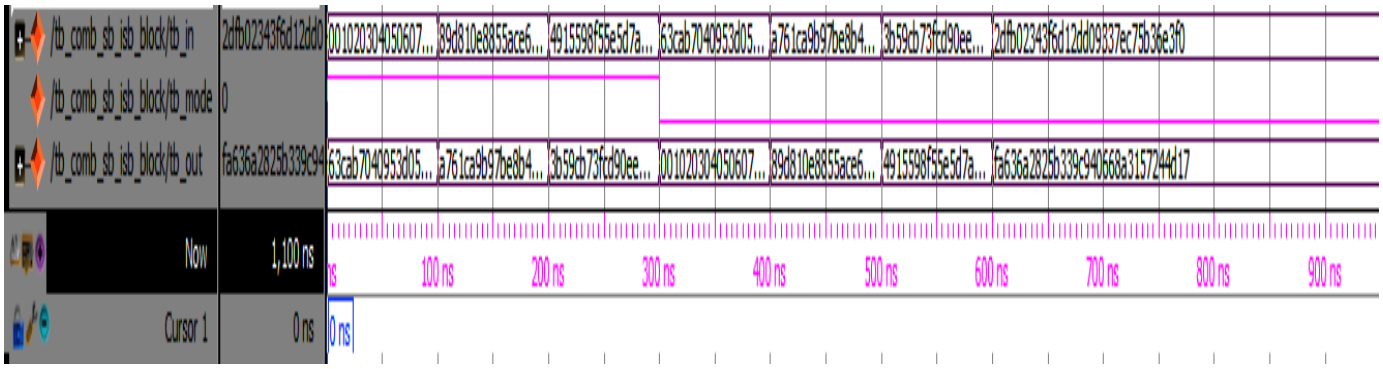


Figure 5.2: ByteSub Block test case

(ii) MixColumns Block test case

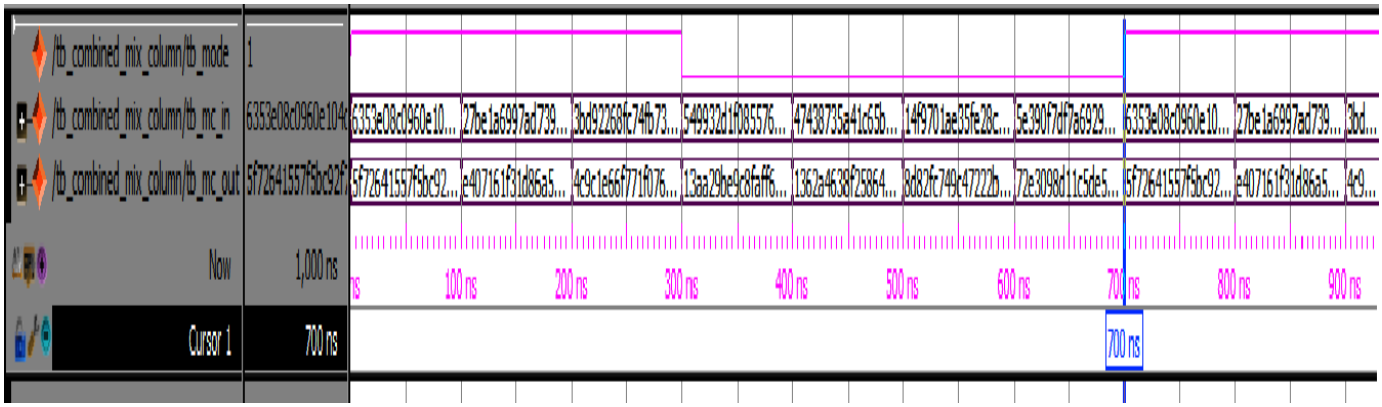


Figure 5.3: MixColumns Block test case

(iii) ShiftRows Block test case

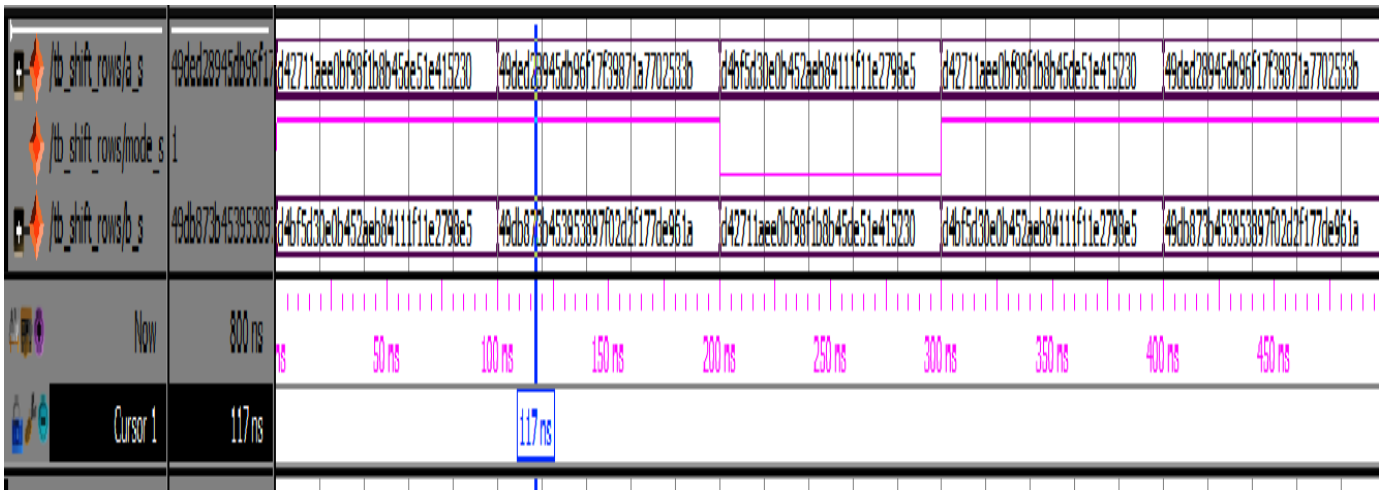


Figure 5.4: ShiftRows Block test case

(iv) Key Expansion test cases for all three types of keys

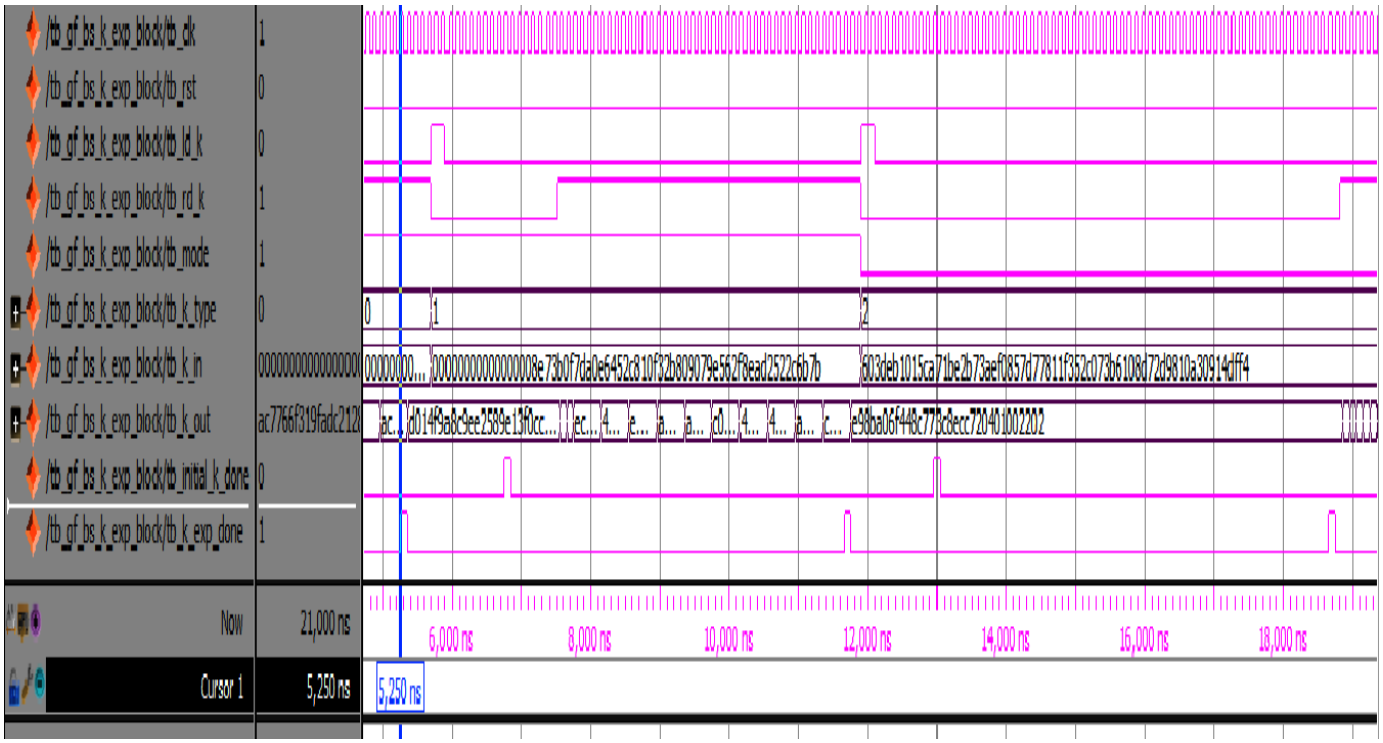


Figure 5.5: Key Expansion test cases for all three types of keys

(v) Encryption test cases of top module for all three types of keys

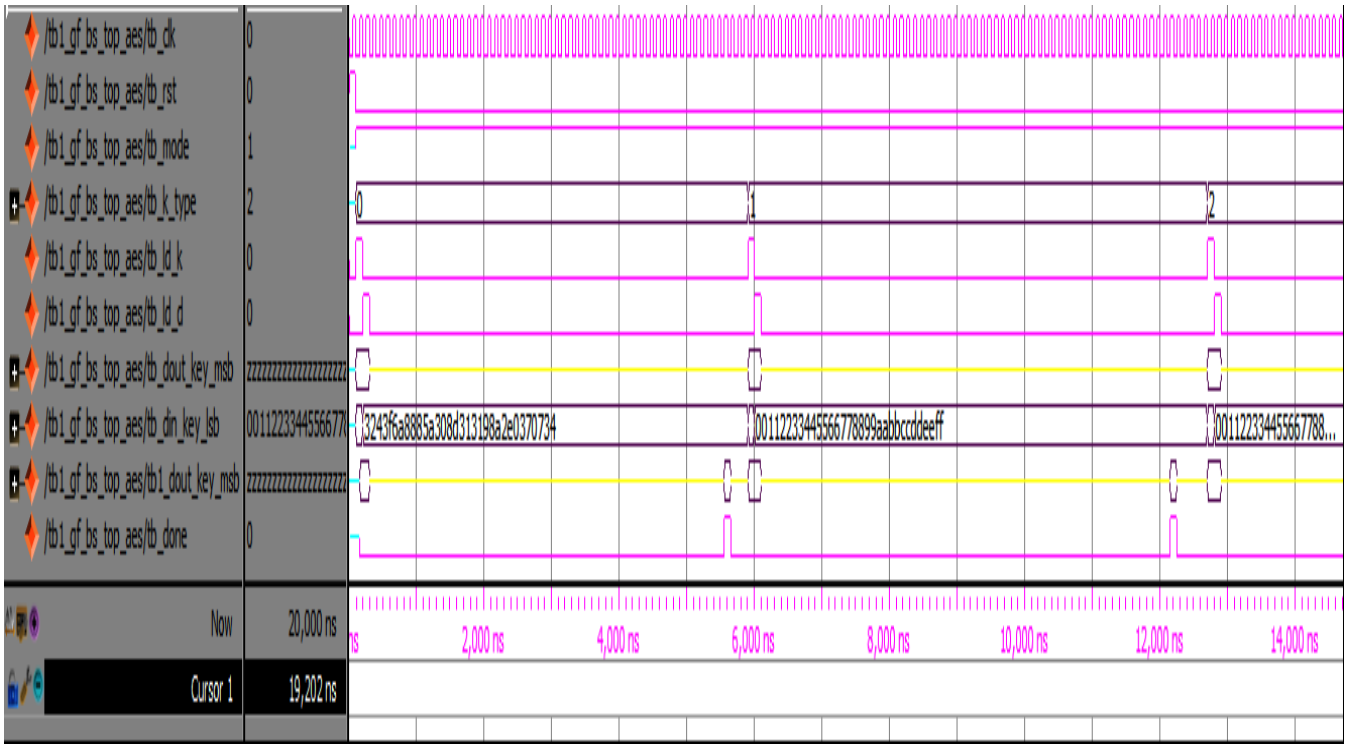


Figure 5.6: Encryption test cases of top module for all three types of keys

(vi) Decryption test cases of top module for all three types of keys

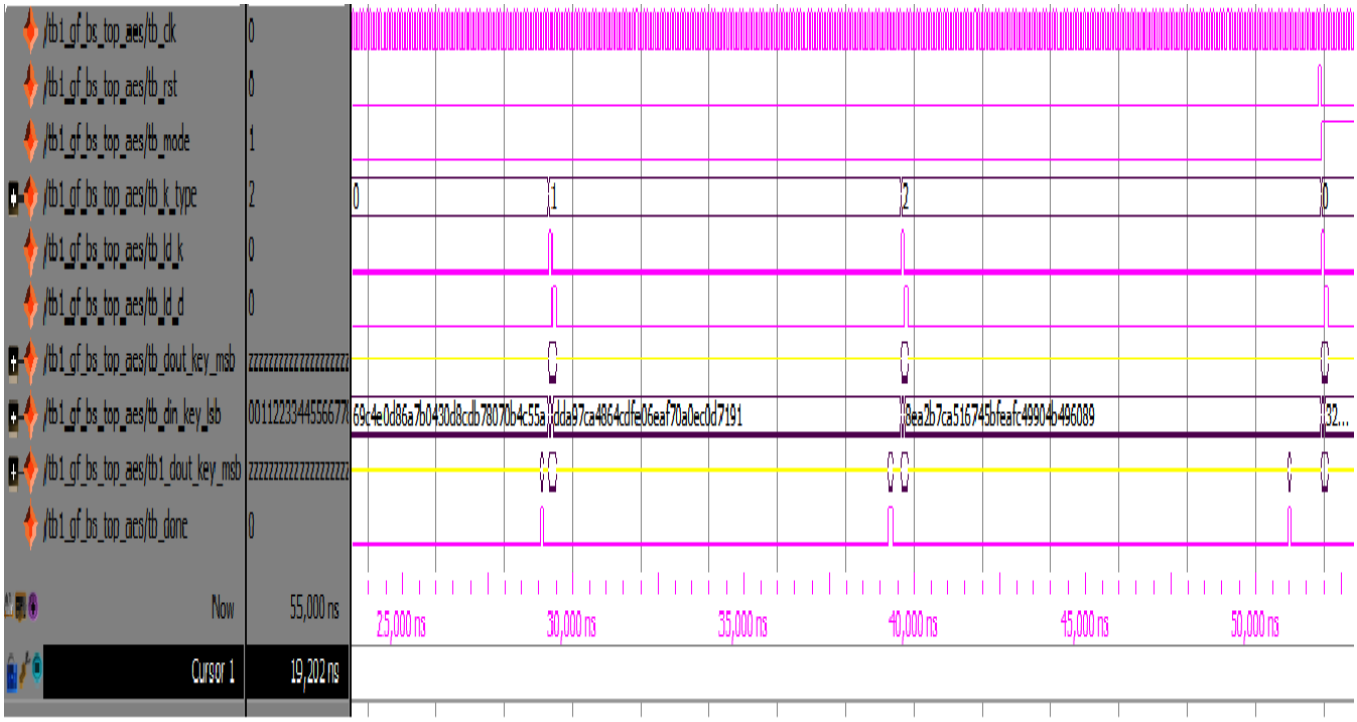


Figure 5.7: Decryption test cases of top module for all three types of keys

5.4 Analysis of Simulation Results

After analysing the simulation results, it was found that Key Expansion Block takes *4* clock cycles to generate single round key. Hence for N_r number of rounds, the total number of clock cycles required to generate complete round key is = $4 \times N_r + 11$ for both encryption and decryption. A maximum of $4 \times 14 + 11 = 67$ clock cycles are required to generate complete round key for 256 bits key.

Similarly the complete encryption process of top AES Block requires just 4-more clock cycles as compare to Key Expansion Block. In this way the total number of clock pulses required to complete the entire encryption process of this design can be formulated as

$$= 4 \times N_r + 15$$

As previously mentioned that the decryption process starts after the key expansion process has been completed. Hence the decryption process requires approximate double number of clock pulses as compare to the decryption process.

Total no. of clock pulses required for complete encryption/decryption process and key expansion process are shown in following table:-

	$N_k = 4$	$N_k = 6$	$N_k = 8$
Total no. of clock pulses required for complete encryption process	55	63	71
Total no. of clock pulses required for complete decryption process	86	100	114
Total no. of clock pulses required for complete key expansion process	51	59	67

Table 5.2: Comparison of Simulation Results

5.5 Synthesis Results

As mentioned previously and also clear from the name of Tiny AES that the main aim of my design is to minimize the area, which can be achieved by using the less digital logic hardware by applying various resource sharing options between encryption and decryption. The main area consuming block is SubByte Block. So my main effort was to reduce less logic hardware for this block. In this way I have designed my Tiny AES core at various stages. All these stages are described in following subsections:-

(1) AES using LUT method in SubByte Block

AS previously mentioned SubByte Block is the main area consuming block, so I start with minimizing the area of this block. I start by implementing this block with the help of LUT method, which utilizes the maximum digital logical hardware compare to other stages of

design. So this stage shows the implementation of tiny AES using LUT method in SubByte Block.

(2) Tiny AES using Galois Field method in SubByte Block without bidirectional bus

I further reduced the area of my design by using the Galois Field method in SubByte Block, which have less logical hardware as compare to previous version of design.

(3) Tiny AES using Galois Field method in SubByte Block with bidirectional bus

The main drawback of the previous level of design was that it was having more no of input/output pins (520), but the FPGA device which I selected was having only 320 I/O Buffers. So in the next level, I decided to use the bus sharing and bidirectional data bus concept among I/P, O/P data and encryption key signals.

I shared the 128-bit input bus signal din_k_lsb between input data and LSB bits of encryption key, which gives the input data and LSB bits of encryption key to the design at different intervals of time with the help of ld_k and ld_d signals. I made the dout_k_msb signal as bidirectional data bus to assign/take the MSB bits of encryption key and output data to/from the Tiny AES design respectively. In this way, I reduce the 256 number of I/O pins in this version of design, which use only 264 I/O Buffers as compare to 520 of previous stages of design and which fit in our selected FPGA device.

(4) Latch Free Tiny AES using Galois Field method in SubByte Block with bidirectional data bus

The demerit of all the above mentioned designs is that in all those designs so many unwanted latches are inferred, which utilizes the more logical hardware in terms of unwanted latches and which results in more silicon cell area of my design. As we know that latches are inferred in any design because of not including all the cases of if/else and case statement of our RTL design. Hence in this way I further reduce the silicon cell area of my design by removing all those unwanted latches from all the blocks of my design, by specifying all the conditions of if/else and case conditional statements.

I have implemented and verified all the blocks of all the stages of design using Verilog HDL language. I have synthesized all the above mentioned designs using both the synthesis tool Xilinx ISE 13.1 and Synopsys Design Compiler (Design Vision), who's various synthesis results of timing, area and power reports are compared in table 5.3 and 5.4 respectively.

5.5.1 Synthesis Results on FPGA

The Xilinx ISE is used for implementation of all the circuits on FPGA flow. The Working Environment for the design is:

- Target Device : 4vfx12ff668-10 (Virtex-4)
- Technology : 90 nm
- Tool Version : ISE 13.1
- FSM Style : LUT
- Optimization Goal : Area
- Design Strategy : Balanced
- Total Slices : 5472
- Total Slice Flip-Flops: 10944
- No. of 4 input LUTs: 10944

Table 5.3 shows the comparison of synthesis results for various stages of Design on Xilinx ISE 13.1. The RTL Schematic View of Top AES Block is shown in figure 5.8.

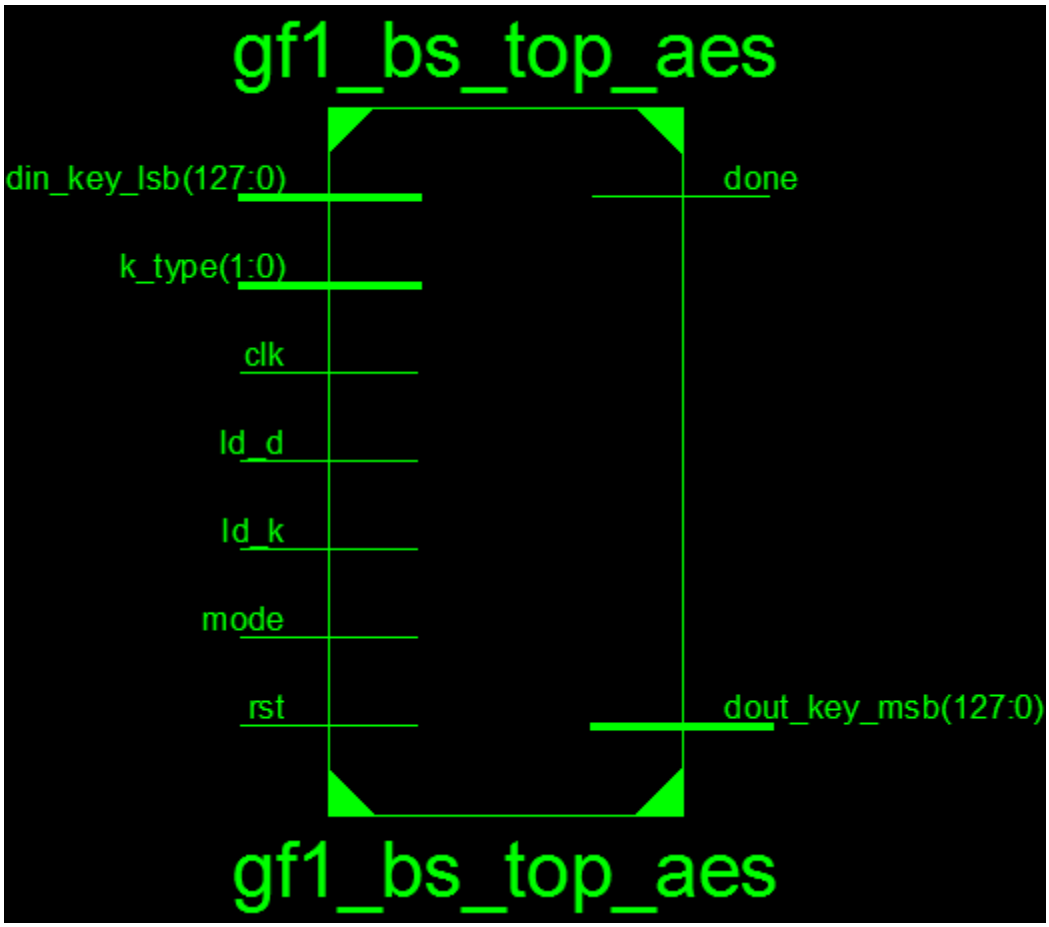


Figure: 5.8: RTL Schematic View of Top AES Block

	1			2			3			4		
	Used	**Available**	**% Utilization**	**Used**	**Available**	**% Utilization**	**Used**	**Available**	**% Utilization**	**Used**	**Available**	**% Utilization**
Number of Slices	4465	5472	81	2803	5472	51	2802	5472	51	2876	5472	52
Number of Slice Flip Flops	3246	10944	29	2475	10944	22	2473	10944	22	2251	10944	20
Number of 4 input LUTs	6765	10944	61	4668	10944	42	4667	10944	42	4878	10944	44
Number of bonded IOBs	520	320	162	520	320	162	264	320	82	264	320	82
Number of FIFO16/RAMB16s	4	36	11	4	36	11	4	36	11	4	36	11
Number of GCLKs	7	32	21	7	32	21	7	32	21	4	32	12
No. of Latches							11 (7,3 ,1 no. for 128, 32 8-bit latches)			6 no. of 128-bit latches		
Minimum Period (ns)	4.095			10.271			10.271			10.781		
Max Frq. (MHz)	244.219			97.357			97.357			92.755		
Minimum I/P arrival time before clock (ns)	8.826			12.977			14.213			12.511		
Maximum O/P arrival time after clock (ns)	4.694			4.694			6.353			6.353		

Table: 5.3 Comparison of Various stages of Design for Xilinx ISE Tool

5.5.2 Synthesis Results for ASIC Flow

Synopsys tool version D-2010.03-SP1 is used to implement and synthesize the design using ASIC flow. Design Compiler and Design Vision are the command mode and GUI mode of Synopsys tool respectively.

Table 5.3 shows the comparison of synthesis results for various stages of Design on Synopsys tool for ASIC flow.

	1	2	3	4
Number of ports	520	520	264	264
Number of nets	3662	3673	3640	3642
Number of cells	2755	2766	2982	2934
Number of references	33	34	47	46
Combinational area	366409.062500	216824.625000	247289.312500	211990.390625
Non-combinational area	234107.281250	234154.140625	237419.984375	229054.296875
Total cell area	600486.562500	450958.625000	484687.718750	441024.906250
data required time (ns)	3.61	4.94	2.94	4.97
library setup time (ns)	0.04	0.06	0.06	0.03
Minimum Time period (ns)	3.65	5.0	3.0	5.0
Maximum clock frequency (MHz)	273.97	200	333.33	200
Cell Internal Power (mw)	52.6591	56.0821	98.0889	53.4002
Net Switching Power (mw)	11.8009	18.4122	31.5400	17.8940
Total Dynamic Power (mw)	64.4600	74.4943	129.6289	71.2942
Cell Leakage Power (nw)	163.2306	134.7856	145.3295	130.7347

Table 5.4: Comparison of Various stages of Design for ASIC flow

5.5.3 Analysis of Synthesis Results

There are three main constraints (Time, Area and Power), which can be given to any synthesis tool while synthesizing the particular design. So that the particular design cab be optimized for any of the timing, area and power performance. Any design can be optimized for one constraint only at a time, which depends on the designer's goal for which performance he wants to work his design.

As previously mentioned in the objectives of this thesis work that main aim of this design is to use the area of silicon chip. So our design will be optimized for area. From the synthesis results of both Xilinx ISE and Synopsys Design Compiler tools, we can conclude that as we grow from various stages of this design, the combinational and sequential cell area (area constraints) is minimized, while maximum clock frequency (timing constraint) is decreased.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis report Designing and Verification of Generic, Single Encryptor/Decryptor core Unit for area efficient hardware implementation of Tiny AES has been discussed, which can not only implement all the three AES (AES128, AES196, and AES256) algorithms, but also do both the encryption and decryption processes in just single design unit. Different versions of AES algorithm exist today (AES128, AES196, and AES256) depending on the size of the encryption key, but none of them has optimized the area for single design unit for encryption/decryption, which can support all three types of encryption AES128, AES196, and AES256. For selecting particular type of AES, I have an extra pin called k_type in my implemented design.

It should also be noted that I have not only finally implemented and synthesized this design on Xilinx ISE tool for FPGA flow, but also on Synopsis tool for ASIC flow and compared the results for various stages of design for both the FPGA and ASIC flow in terms of Area, Timing and Power report. A unique feature of this design in this report is that the round keys, which are consumed during different iterations of encryption, are generated in parallel with the encryption process, which can further increase the speed of design.

The issues of generic, Single Encryption/ Decryption design unit for AES implementation have been discussed. During the literature survey, it was found that reference [10] implement the encryption and decryption unit in single block diagram, but it is not generic, on the other hand reference [6] design unit is generic but both the encryption and decryption processes were not implemented in that design. I also found a paper as shown in reference [13], which do the desired purpose but that implement the AES-Rijndael algorithm and I have to implement the main AES algorithm. Hence it can be concluded that there is not any single design unit in literature for implementation of main AES algorithm , which can do both encryption and decryption processes as well as support all three types of AES Algo's (AES128, AES192, AES256). In this report I have provided a solution for above mentioned problem and designed a generic single Encryptor/Decryptor core for Tiny AES algorithm, which can work in all three types of AES's conditions. For this I have used the architecture of

[13] and make it possible to implement the main AES-algorithm by slightly change its structure for minimum area.

5.2 Future Work

Future work includes addition of inner pipelined architecture in the one stage and the four staged pipelined AES so that the throughput and speed are increased and comparable to the fully pipelined loop unrolled architectures. Further optimization or new architectures for S-box and Inverse Mix Columns can also improve the performance. In the present architecture the data transformation does not start until the 128 bits of the Input are accumulated for the first set of data. Proper usage of this idle time can also help in increasing the performance.

In this design the 128-bit data is processed through various blocks of AES algorithm in parallel way, which required separate 16-times instantiation of SubBytes and MixedColumn blocks because each of this separate block process 1-byte of data at a time, which results in more digital logical hardware and more silicon area. In the near future design this 128-bit of data can be initially stored in memory state table and processed in serial way by serially taking the 1-byte data at a time from memory and apply it to SubBytes and MixedColumn blocks one by one and then after processing particular function on the 1-byte data, write back it to memory state table. This proposed design will definitely consume less silicon area in terms of 15-lower no of SubBytes and MixedColumn digital logic blocks as compare to my current design, but that proposed design will utilizes the more no of clock cycles (i.e. it's processing time will be more) as compare to the current design, which required just maximum 71 and 110 no of clock pulses for encryption and decryption respectively.

REFERENCES

[1] Federal Information Processing Standards Publication (FIPS 197), Advanced Encryption Standard (AES), Nov. 26, 2001.

[2] Federal Information Processing Standards Publication (FIPS 46),The Data Encryption Standard (DES), 46 on Nov. 23, 1977.

[3] K.V. Dalmisli, B Ors, " Design of new tiny circuits for AES encryption Algorithm" in 3rd international conference on Signals,Circuits and Systems (SCS), pp. 1-5, 2009.

[4] A.A. Kamal, A.M. Yousef, "An Area Optimized Implementation of the Advanced Encryption", in International Conference on Microelectronics, pp. 159-162, 2008.

[5] LI Zhen-rong, ZHUANG Yi-qi, ZHANG Chao, JIN Gang, "Low-power and area-optimized VLSI implementation of AES coprocessor for Zigbee system", in the science direct a journal of China Universities of Posts and Telecommunications, pp. 89-94, June 2009.

[6] M Alam, S Ray, D Mukhopadhayay, S Ghosh, D Roy Chowdhury, I Sengupta, "An Area Optimized Reconfigurable Encryptor for AES-Rijndael**",** in Design**,** Automation & Test in Europe Conference and Exhibition**,** pp. 1-6, 2007.

[7] P.V.S Shastry, A. Agnihotri, D. Kachhwaha, J. Singh, M.S. Sutaone, "A combinational implemnatation of S-box of AES", 2011 IEE 54th international Midwest Symposium on Circuits and Sytems, pp. 1-4, 2011.

[8] S.K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S.K. Hsu, H. Kaul, M.A. Anders, R.K. Krishnamurthy, "53 Gbps Native GF$(2^4)^2$ Composite-Field AES-Encrypt/ Decrypt Accelerator for Content-Protection in 45 nm High-Performance Microprocessors", IEEE Journal of Solid-State Circuits,Volume 46, pp. 767 – 776.

[9] V. Fischer, M. Drutarovsky, P. Chodowiec, F Gramain, "InvMixColumn decomposition and multilevel resource sharing in AES implementations", IEEE Transactions on VLSI Systems, Volume 13, pp. 989-992, 2005.

[10] A. Rady, E. El Sehely, A.M. El Hennawy, Design and implementation of area optimized AES algorithm on reconfigurable FPGA, International conference on Microelcetronics, pp. 35-38, 2007.

[11] A.J. Elbirt, C. Paar, Efficient Implementation of Galois Field Fixed Field Constant Multiplication, Third International Conference on Information Technology: New Generations, 2006, pp. 172-177, 2006.

[12] Jia Zhao, Xiaoyang Zeng, Jun Han, Jun Chen, Very Low-cost VLSI implementation of AES Algorithm, IEEE Asian Solid-State Circuits Conference, pp. 223-226, 2006.

[13] Monjur Alam, Santosh Ghosh, Dipanwita RoyChowdhury, Indranil Sengupta, Department of Computer Science and Engineering Indian Institute of Technology, Kharagpur, Single Chip Encryptor/Decryptor Core Implementation of AES Algorithm, 21st International conference on VLSI Design.

[14] J.Wolkerstorfer, "An ASIC implementation of the AES MixColumn operation," in *Proc. Austrochip 2001*, Vienna, Austria, Oct. 12, 2001, pp. 129–132.

[15] C.C. Lu and S.-Y. Tseng, "Integrated design of AES (advanced encryption standard) encrypter and decrypter," in *Proc. IEEE Int. Conf. Application*-Specific Systems, Architectures and Processors (ASAP'02), 2002, pp. 277–285.

[16] H. Samiee, R.E. Atani, H. Amindavar, "A novel area-throughput optimized architecture for the AES algorithm", 2011 International Conference on Electronic Devices, Systems and Applications (ICEDSA), pp. 29-32, 2011.

[17] Renhai Chen, Zhiping Jia, Yibin Li, Hui Xia, Xin Li, The application specific instruction processor for AES, 3rd International Conference on Electronics Computer Technology (ICECT), Volume 4 pp. 394–396, 2011.

[18] Ai-Wen Luo, Qing-Ming Yi, Min Shi, "Design and implementation of area-optimized AES based on FPGA", International Conference on Business Management and Electronic Information (BMEI), Volume 1, pp 743 – 746, 2011.

[19] Cong Chen, Xiangyu Li, Liji Wu, Xiangmin Zhang, Design and implementation of a Differential Power Analysis System for cryptographic devices, 10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp.1967 – 1969, 2010.

[20] Kuo-Huang Chang, Yi-Cheng Chen, Chung-Cheng Hsieh, Chi-Wu Huang, Chi-Jeng Chang, Embedded a low area 32-bit AES for image encryption/decryption application, IEEE International Symposium on Circuits and Systems, pp. 1922–1925, 2009.

[21] S. Ahuja, S.T. Gurumani, C. Spackman, S.K. Shukla, "Hardware Coprocessor Synthesis from an ANSI C Specification", IEEE Design & Test of Computers, Volume: 26, pp. 58-67, 2009.

[22] M.M. Wong, M.L.D. Wong, I Hijazin, A.K. Nandi, Composite field GF(((2^2)^2)^2) AES S-Box with direct computation in GF(2^4) inversion, 7th International Conference on Information Technology in Asia (CITA 11), pp. 1–6, 2011.

[23] N. Ahmad, R. Hasan, W.M. Jubadi, Design of AES S-box using combinational logic optimization, IEEE Symposium on Industrial Electronics & Applications (ISIEA), pp. 696-699, 2010.

[24] M. Mozaffari-Kermani, A. Reyhani-Masoleh, A low-cost S-box for the Advanced Encryption Standard using normal basis, IEEE International Conference on Electro/Information Technology, pp. 52 – 55, 2009.