

# FPGA IMPLEMENTATION OF ADVANCED ENCRYPTION STANDARD

*A Thesis Submitted in Partial Fulfillment of  
the Requirements for the Degree of*

**Bachelor of Technology  
In  
Electronics and COMMUNICATION Engineering**

BY

**VIKASH KUMAR  
110EC0169**

**ASHUTOSH TIBREWAL  
110EC0183**



**Department of Electronics & Communication Engineering  
National Institute of Technology, Rourkela  
2014**

# FPGA IMPLEMENTATION OF ADVANCED ENCRYPTION STANDARD

*A Thesis Submitted in Partial Fulfillment of  
the Requirements for the Degree of*

**Bachelor of Technology  
In  
Electronics and COMMUNICATION Engineering**

**BY  
VIKASH KUMAR  
110EC0169**

**ASHUTOSH TIBREWAL  
110EC0183**

**Under the guidance  
of  
Prof. A K SWAIN**



**Department of Electronics & Communication Engineering  
National Institute of Technology, Rourkela  
2014**



**NATIONAL INSTITUTE OF TECHNOLOGY**  
**ROURKELA**

**CERTIFICATE**

This is to certify that the project work titled “**FPGA Implementation of Advanced Encryption Standard**” submitted by *Vikash Kumar (110EC0169)* and *Ashutosh Tibrewal (110EC0183)* in the partial fulfilment of the requirements for the degree of **Bachelor of Technology in Electronics and Communication Engineering** during the session 2013-2014 at National Institute of Technology, Rourkela is an authentic and bona fide work carried out by them under my supervision.

DATE:

**Prof. A K Swain**, Assistant Professor

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

## ACKNOWLEDGEMENT

We would be keen to express our gratefulness to our project guide **Prof A. K. Swain** for his guidance and consistent encouragement throughout the time-span of this Project work.

We would also like to express thanks to **Prof K.K. Mahapatra, Prof D. P. Acharya, Prof S. K. Patra, and Prof S.K. Das** for taking up the courses of Digital VLSI Design, Embedded Systems, Microprocessors, and Digital Electronics respectively which inspired us to take up this project which comes under the category VLSI and Embedded system.

We would like to thank the VLSI lab staffs for providing us the adequate materials, kits and software which were essential for developing our project.

We would like to thank the Institute for giving us the opportunity to study here and enrich our knowledge and skills and providing us the state-of-art infrastructure.

Lastly, we would like to thank our Parents for making us eligible for what we are today and we are doing to fulfill our wishes.

*Vikash Kumar*

*110EC0169*

*Ashutosh Tibrewal*

*110EC0183*

*Dedicated To Our Family*  
*And*  
*Our Teachers*

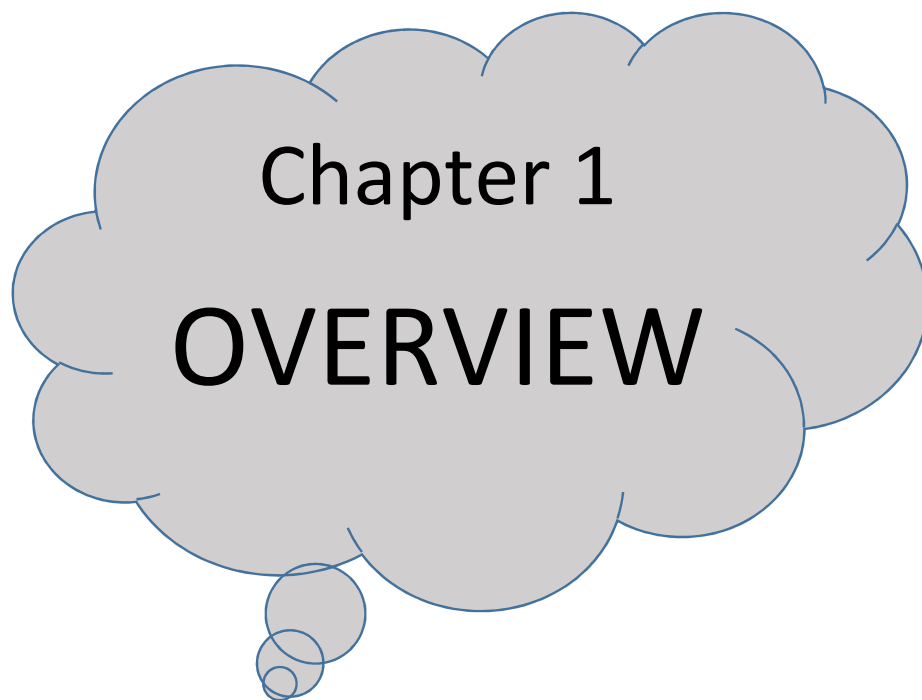
# ABSTRACT

*Security is a crucial parameter to be recognized with the improvement of electronic communication. Today most research in the field of electronic communication includes look into on security concern of communication. At present most by and large consumed and recognized standard for encryption of data is the Advanced Encryption Standard. AES was transformed to supplant the developing Data Encryption Standard. The AES calculation is fit for handling cryptographic keys which are of 256, 128, & 192 bits to encode & unscramble data in squares of 128 bits. The center of the calculation is made up of four key parts, which manage 8 bit data pieces. The whole 128 bit data to the calculation is dealt with into a 4 x 4 grid termed a state, to obtain the 8 bit square.*

*Considering the complex nature of advance encryption standard (AES) algorithm, it requires a huge amount of hardware resources for its practical implementation. The extreme amount of hardware requirement makes its hardware implementation very burdensome. During this research, a FPGA scheme is introduced which is highly efficient in terms of resource utilization. In this scheme implementation of AES algorithm is done as a finite state machine (FSM). VHDL is used as a programming language for the purpose of design. Data path and control unit are designed for both cipher and decipher block, after that respective data path and control unit are integrated using structural modeling style of VHDL. Xilinx\_ISE\_14.2 software is being used for the purpose of simulating and optimizing the synthesizable VHDL code. The working of the implemented algorithm is tested using VHDL test bench wave form of Xilinx ISE simulator and resource utilization is also presented for a targeted Spartan3e XC3s500e FPGA.*

# CONTENTS

|  |    |
|--|----|
| <b>CERTIFICATE</b> .....                         | 3  |
| <b>ACKNOWLEDGEMENT</b> .....                     | 4  |
| <b>ABSTRACT</b> .....                            | 6  |
| <b>1. OVERVIEW</b> .....                         | 9  |
| 1.1. MOTIVATION .....                            | 9  |
| 1.2. RESEARCH OBJECTIVE.....                     | 10 |
| 1.3. LITERATURE SURVEY.....                      | 10 |
| 1.4. DESIGN TOOLS .....                          | 10 |
| 1.5. ORGANISATION.....                           | 11 |
| <b>2. INTRODUCTION</b> .....                     | 13 |
| 2.1. WHY CRYPTOGRAPHY?.....                      | 13 |
| 2.2. WHAT IS CRYPTOGRAPHY?.....                  | 13 |
| 2.3. TYPES OF CRYPTOGRAPHIC ALGORITHM.....       | 14 |
| 2.4. GALOIS FIELD .....                          | 18 |
| 2.5. DATA ENCRYPTION STANDARD .....              | 18 |
| <b>3. THE ADVANCED ENCRYPTION STANDARD</b> ..... | 21 |
| 3.1. INTRODUCTION .....                          | 21 |
| 3.2. HISTORY .....                               | 21 |
| 3.3. AES CIPHER AND DECIPHER .....               | 22 |
| 3.4. STAGES IN CIPHER AND DECIPHER .....         | 26 |
| <b>4. AES ARCHITECTURE</b> .....                 | 32 |
| 4.1. CIPHER .....                                | 34 |
| 4.2. DECIPHER.....                               | 39 |
| 4.3. TRANSFORMATION BLOCKS .....                 | 44 |
| <b>5. RESULTS</b> .....                          | 51 |
| 5.1. MATLAB GUI IMPLEMENTATION .....             | 51 |
| 5.2. VHDL SIMULATION RESULTS .....               | 52 |
| <b>6. CONCLUSION</b> .....                       | 60 |
| <b>7. REFERENCE</b> .....                        | 60 |



Chapter 1

**OVERVIEW**



# 1. OVERVIEW

## 1.1. MOTIVATION

With overall communication of private and secret information over the machine systems then again the Internet, there is dependably a plausibility of risk to information privacy, information honesty and, likewise information accessibility. Information encryption keeps up information secrecy, trustworthiness and validation. Data has happened to the most imperative stakes in developing interest of need to store each and every significance of occasions in regular life. Messages need to be secured from unapproved gathering. Encipherment is one of the security systems to secure data from community. Encryption shrouds the first substance of a message in order to make it mixed up to anybody, with the exception of the individual who has the extraordinary information to peruse it.

In the past cryptography implies just encryption and decoding utilizing mystery keys, these days it is characterized in diverse components like topsy-turvy-key encipherment (public-key cryptography) and symmetric-key encipherment (called as privet-key cryptography). The general population key calculation is intricate and has high reckoning time. Private Key calculations include stand out key, both for encryption and unscrambling while, open key calculations include two keys, one for encryption and an alternate for decoding. There were numerous cryptographic algorithms proposed, for example, Data Encryption Standard (DES), 2-DES, 3-DES, the Advanced Encryption Standard, Elliptic Curve Cryptography, and different calculations. Numerous examiners and programmers are continually attempting to break these calculations utilizing beast constrain and side channel assaults. A few strike were effective as it was the situation for the Data Encryption Standard in 1993.

AES, is the well-accepted cryptographic algorithm which could be utilized to ensure security towards electronic information. This thesis gives an AES algorithm respect to FPGA and VHDL this proposes a strategy to incorporate the AES coder and the AES decoder. This strategy can be of a small-intricacy structural planning, particularly in sparing the fittings asset in executing the AES (Inv) Sub Bytes module and (Inv) Mix column module and so on. Most composed modules could be utilized for both AES encryption and decoding. Additionally, the construction modeling can at present convey a bulk information rate in both encryption/decoding procedures. The suggested building design is suited for equipment-discriminating requisitions, for example, shrewd card, PDA, and cellular telephone, and so on.

Design optimization is being done by using Finite State Machine. Data path and control unit are designed for both cipher and decipher block, after that respective data path and control unit are integrated using structural modeling style of VHDL. Xilinx\_ISE\_14.2 software is being used for the purpose of simulating and optimizing the synthesizable VHDL code.

## 1.2. RESEARCH OBJECTIVE

In the light of optimized FPGA implementation of Advance Encryption Standard (AES) algorithm, the main objective of our research are:

1. Designing of Finite State Machine (FSM) using minimum number of state for the purpose of FPGA implementation of AES algorithm.
2. Designing of Hardware efficient data path for Encryption and decryption.
3. Designing of Hardware efficient control unit path for Encryption and decryption.
4. FPGA resource optimization.
5. VHDL Simulation of AES Algorithm.

## 1.3. LITERATURE SURVEY

- FPGA schemes for minimizing the power-throughput trade-off in executing the Advanced Encryption Standard algorithm, Journal of Systems Architecture 56 (2010) 116–123.(Jason Van Dyken, José G. Delgado-Frias)
- ADVANCED ENCRYPTION STANDARD, Federal Information Processing Standards Publication 197, November 26, 2001.

## 1.4. DESIGN TOOLS

Several developmental tools were used for the implementation of our project. This includes generating Test-bench waveform, RTL simulations etc. and design summary.

We used Xilinx ISE (integrated software environment) 14.2 software for designing out circuit using VHDL code and Developing the Test-bench and schematics of the modules.

This software allows us to take our design from design entry through Xilinx device programming. The ISE project navigator processes our design through various steps in the ISE design flow.

The following are the steps used

- Design Entry
- Synthesis
- Implementation
- Simulation and verification
- Device Configuration

The Test-bench waveform containing the signals can be used to simulate the modules used in our project in the Xilinx ISE simulator.

This provides a powerful and highly advanced self-contained development platform for designs targeting the Spartan 3e FPGA from Xilinx. Features like Xilinx Platform Flash, USB end, JTAG parallel programming interfaces are also found on this board.

## 1.5. ORGANISATION

This thesis is organized as follows:

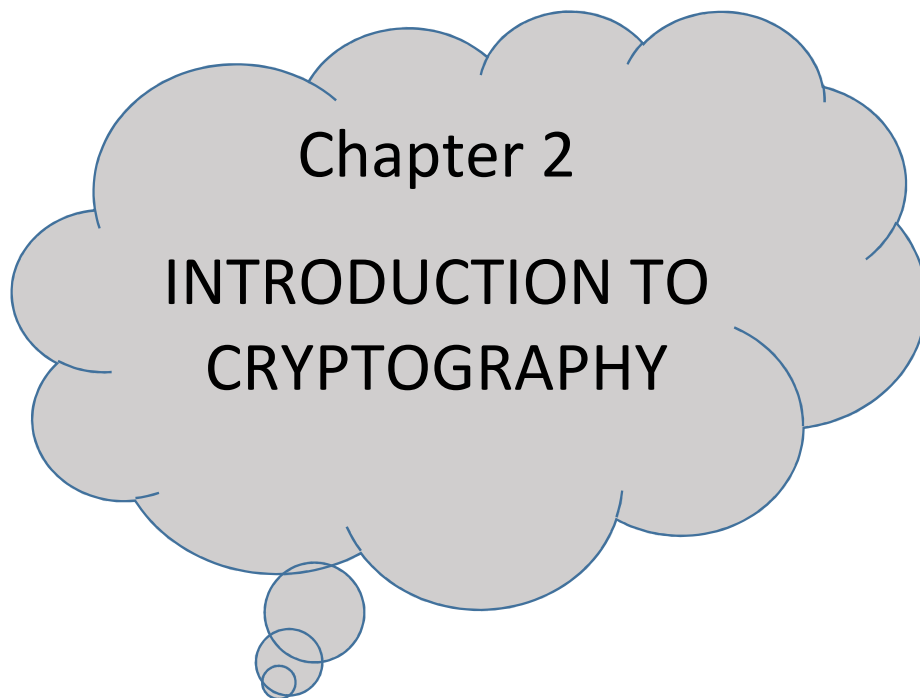
Chapter 2 describe history and requirement of cryptography, concept of Galois field and about data encryption standard (DES) algorithms, which was used earlier.

Chapter 3 describes the AES algorithm in details. The four encryption stages are presented: Byte Substitution, Shift Rows, Mix Column and lastly Add Round Key and inverse part of all four blocks. It also describes the details of Cipher and Decipher block.

In Chapter 4, a proposed architecture of AES algorithm is presented. In which, we have described the detailed architecture of designed data path and control unit for both cipher and decipher.

In Chapter 5, simulation and results are presented in this chapter, with the test bench wave form and block architecture of each block used in the AES along with complete AES block.

Finally, the conclusion and future work are presented in Chapter 6.



Chapter 2  
INTRODUCTION TO  
CRYPTOGRAPHY

## 2. INTRODUCTION

### 2.1. WHY CRYPTOGRAPHY?

Does expanded security give solace to distrustful individuals? Then again does security give some extremely essential insurances that we are guileless to accept that we needn't bother with? Throughout this period when the World Wide Web gives crucial correspondence between countless individuals and is constantly progressively utilized as an apparatus for trade, security turns into an enormously essential issue to manage.

There are numerous angles to security and numerous provisions, extending from safe trade and installments to private correspondences and ensuring passwords. One vital perspective for safe interchanges is that of cryptography, which the fundamental center of this subject is. At the same time it is paramount to notice that when cryptography is fundamental for safe interchanges, it is not independent from anyone else sufficient. The onlooker is exhorted, then, that the themes secured in this part just portray the first of numerous steps important for important security in any count of situations.

### 2.2. WHAT IS CRYPTOGRAPHY?

Cryptography is an art of composing in mystery symbols and is an antiquated craft; the initially reported utilization of cryptography in composing goes once again to circa-1900 B.C. at the point when an Egyptian copyist utilized non-standard symbolic representations in an engraving. A few masters contend that cryptography showed up spontaneously at some point in the wake of composing was imagined, with requisitions running from strategic messages to war-time fight tactics. It is not at all astonishment, then, that new types of cryptography came not long after the across the board improvement of machine interchanges. In information and telecommunications, cryptography is fundamental when conveying over any non-trusted medium, which incorporates pretty much any system, especially the WWW.

Inside the connection of any provision-to-requisition communication, there are some particular security prerequisites, including:

- *Authentication*: The procedure of demonstrating one's character. (The essential types of host-to-host validation on the WWW today are name-based or location-based, both of which are famously feeble.)
- *Privacy/confidentiality*: Guaranteeing that nobody can read the message with the exception of the proposed receiver.
- *Integrity*: Guaranteeing the receiver that the received message has not been compromised in any possible way from the initial.
- *Non-repudiation*: A procedure to demonstrate that the messenger really sent the message. [3]

Cryptography, then ensures information from theft or change, as well as be utilized for client confirmation. There are, when all is said in done, three sorts of cryptographic plans ordinarily used to achieve these objectives: mystery key (or symmetric) cryptography, open-key (or unbalanced) cryptography, and hash works, each of which is depicted beneath. In all instances, the introductory decoded information is alluded to as plain-text. It is encoded into figure content, which will thus (ordinarily) be decoded into utilizable plain-text.

### 2.3. TYPES OF CRYPTOGRAPHIC ALGORITHM

There are numerous ways of categorizing cryptographic algorithms. For commitments to this thesis, they will be classified based on the number of keys that are engaged for encryption and decryption, and further demarcated by their application and use. The three kinds of algorithms that is conferred are given below in fig 2.3.

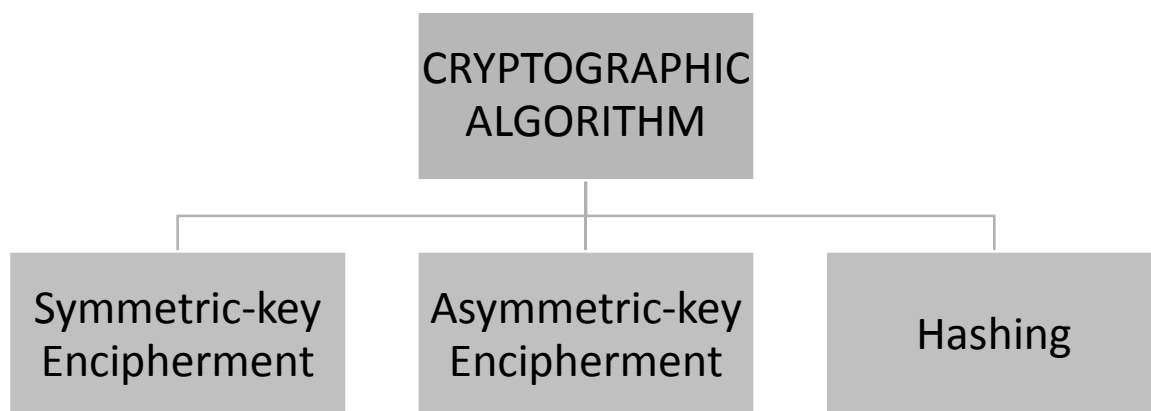


Fig 2.3: Types of Cryptographic Algorithm based on number of keys

### 2.3.1. Symmetric-key Encipherment or Secret key Cryptography



Fig 2.3.1(a): Block Diagram of Symmetric key Encipherment

In symmetric-key encipherment a substance say Viku, can make an impression on an alternate element, say Ashu, over an unstable channel with the presumption that a foe, say Eve, can't comprehend the substance of the message by basically listening stealthily over the channel. Viku scrambles the message utilizing an encryption calculation; Ashu unscrambles the message utilizing an unscrambling calculation. Symmetric-key encipherment utilizes a solitary mystery key for both encryption and unscrambling. Encryption/decoding might be considered electronic locking. In this, Viku puts the message in a crate and locks the container utilizing the imparted mystery key; Ashu opens the case with the same key and takes out the message.

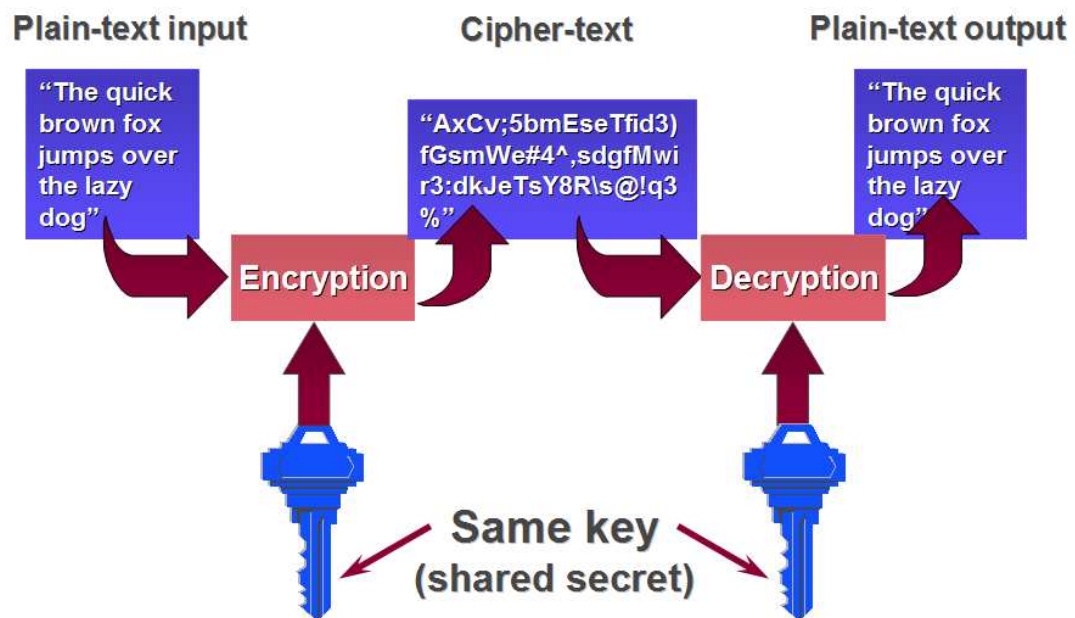


Fig 2.3.1(b): Example for Symmetric Key Encipherment [2]

### 2.3.2. Asymmetric-key Encipherment or Public key Cryptography

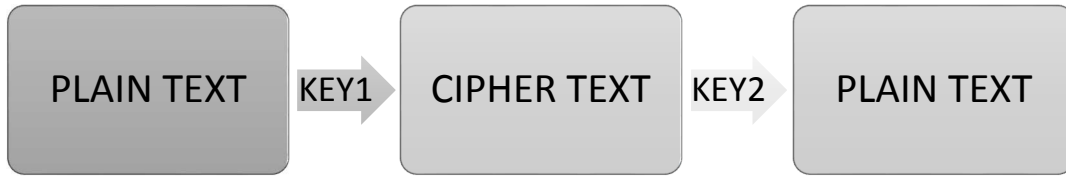


Fig 2.3.2(a): Block Diagram of Asymmetric key Encipherment

In asymmetric-key encipherment, we have the same circumstance as the symmetric-key encipherment, with a couple of exemptions. Initially, there are two keys rather than one: one open key and one private key. To send a secured message to Ashu, Viku first encodes the message utilizing Ashu's open key. To unscramble the message, Ashu utilizes his own particular private key.

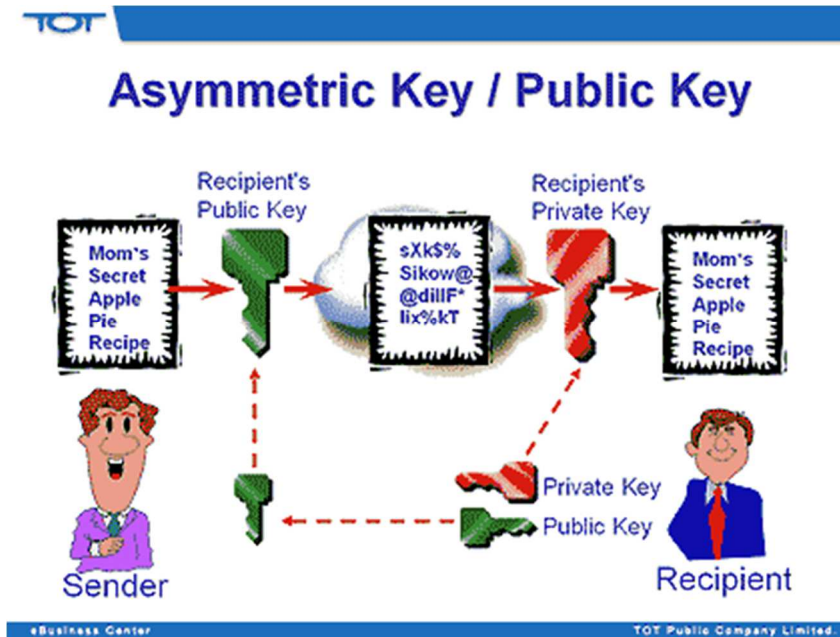


Fig 2.3.2(b): Example for Asymmetric Key Encipherment [2]



### 2.3.3. HASHING



Fig 2.3.3(a): Block Diagram of Hashing

In hashing, an altered-length message condensation is made out of a variable-length message. The condensation is typically much more modest than the message. To be valuable, both the message and the review be sent to Ashu. Hashing is utilized to give check values, which were examined prior in connection to give information respectability.

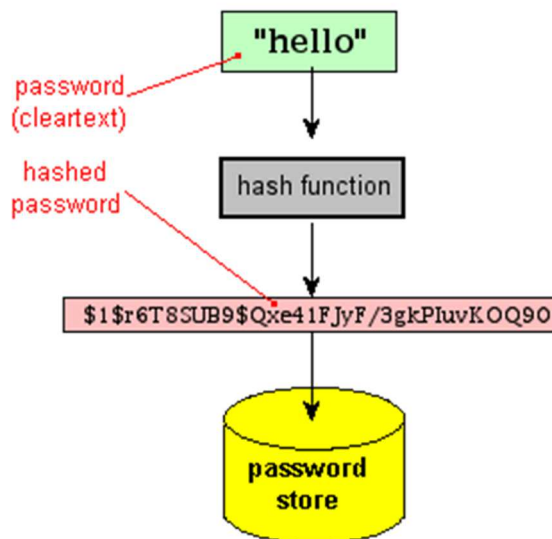


Fig 2.3.2(b): Example for Hashing [2]

## 2.4. GALOIS FIELD

Galois Field, named after Evariste Galois, otherwise called finite field, alludes to a field in which there exists finitely numerous components. It is especially valuable in translating machine information as they are represented in binary structures. That is, computer information comprise of two numbers, 0 and 1, which are the segments in Galois field whose number of element is two. Representing to information as a vector in a Galois Field permits scientific operations to scramble information effectively and effectively.

There are many cryptographic algorithms using GF among them, the AES algorithm uses the GF ( $2^8$ ). The data byte can be characterized using a polynomial representation of GF ( $2^8$ ).

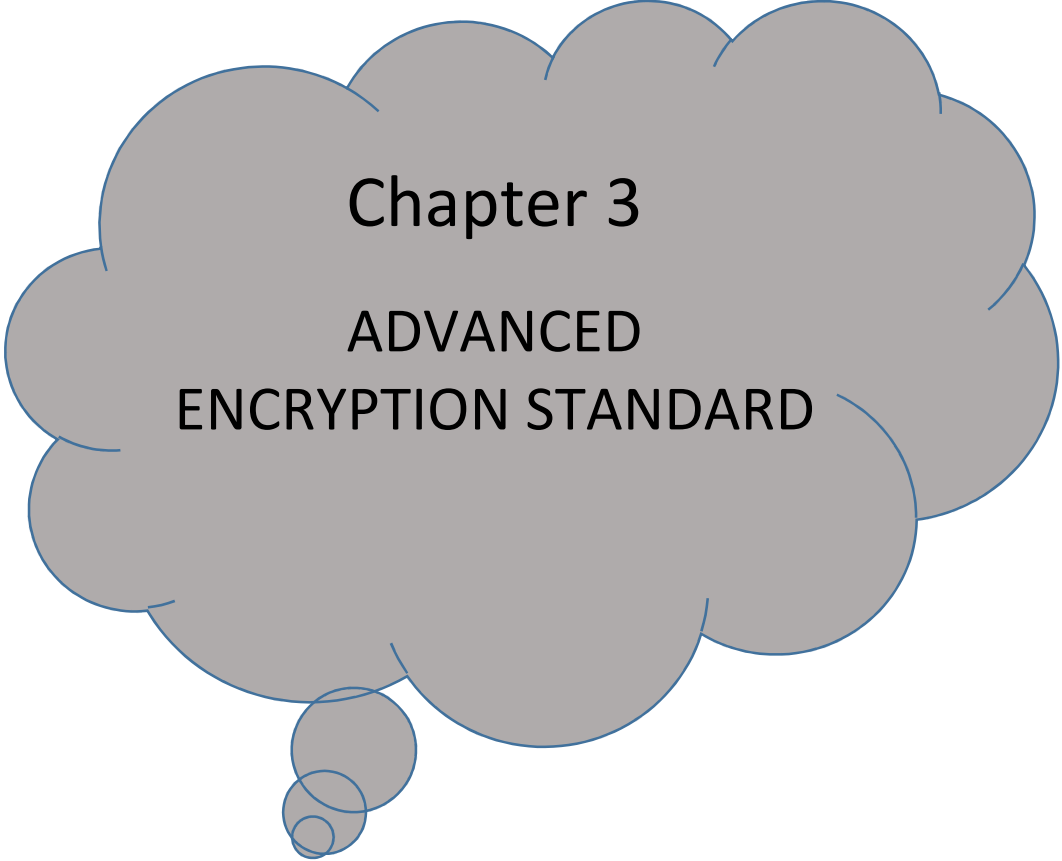
Arithmetic operation is completely not quite the same as typical arithmetic algebra, an addition can be discovered utilizing bit-wise XOR operation. In Galois field, the multiplication product of polynomials will be modulo an irreducible polynomial so final answer can be within the used finite field. The polynomial which cannot be factorized of two or more than two is called as irreducible polynomial. In Galois field GF ( $2^8$ ) addition/subtraction is same as XOR operation and multiplication/division is same as the AND operation. The binary representation of irreducible polynomial used in GF ( $2^8$ ) is  $p=100011011$ . [4]

## 2.5. DATA ENCRYPTION STANDARD

Up to this point, the primary standard for encryption of the information remained a symmetric algorithm called as the DES (Data Encryption Standard). Notwithstanding, this must now been supplanted by another standard called by way of the AES (Advanced Encryption Standard) which we shall take a gander in future. DES is a 64 bit piece figure which implies that it encrypting information 64 bits at once. This is differentiated to a stream cipher in which stand out bit at once (or frequently little gatherings of bits, for example, a byte) is scrambled.

DES was the fruit of a research project performed by International Business Machines (IBM) Corporation in the later parts of 1960's which give rise to a cipher called as LUCIFER. In the earlier parts of 1970's it was decided to commercialize LUCIFER and a quantity of significant

modifications were added. IBM wasn't alone on this ship of modifications as they asked technical help from the National Security Agency (NSA) (other outside experts were aboard but it is probable that, from a technical point of view, the NSA was the chief backer). The changed variety of LUCIFER was presented as a suggestion for the novel national encryption standard demanded by the National Bureau of Standards (NBS). It was lastly accepted in 1977 as the Data Encryption Standard –(DES) (FIPS PUB 46). [1]



**Chapter 3**  
**ADVANCED**  
**ENCRYPTION STANDARD**

## 3. THE ADVANCED ENCRYPTION STANDARD

### 3.1. INTRODUCTION

The **Advanced Encryption Standard** is a determination for the purpose of encryption of automated information built by the **National Institute of Standards and Technology** of U.S. in 2001. AES is focused around the **Rijndael** figure created by **Joan Daemen** and **Vincent Rijmen** (two Belgian cryptographers), who proposed a suggestion to NIST throughout the AES determination process. Rijndael is a group of figures with distinctive key and piece sizes. For AES, NIST chose three parts of the Rijndael family, each with a piece size of 128 bits, yet three distinctive key lengths: 128, 192 and 256 bits. AES has been received by the U.S. government and is currently utilized around the world. It succeeds the **Data Encryption Standard (DES)**, which was distributed in 1977. The algorithm depicted by AES is a symmetric-key calculation, importance the same key is utilized for the purpose of encryption and decryption of the information.

### 3.2. HISTORY

The prior ciphers might be broken without hardly lifting a finger on advanced processing frameworks. The DES calculation was softened up 1998 utilizing a framework that cost about \$250,000. It was additionally unreasonably moderate in programming for it was made for middle-1970's equipment and doesn't process effective programming code. Then again, Triple DES has three times the same number of iterations as DES and is relatively sluggish. And also, the 64 bit square size of triple DES besides DES isn't extremely effective also is faulty concerning security.

What was obliged was a fresh out of the box new encryption algorithm that might be impervious to the majority of the identified attacks. NIST needed someone to assist the making of another algorithm. Nonetheless, in view of the discussion that ran with the DES standard, and the ages of a few limbs of the U.S. government having a go at all that they could to upset sending of protected cryptography this was liable to increase solid distrust. The issue remained was, NIST would really have liked to help make another fantastic encryption standard yet they couldn't get included specifically.

Tragically they were truly the main ones with the specialized notoriety and assets to the lead the exertion. As opposed to outlining or serving to outline a figure, what they did rather was to

announce a challenge in which anybody on the planet could join in. The challenge was affirmed on 2<sup>nd</sup> January, 1997 and the thought existed to create another encryption algorithm which might be utilized for securing delicate, non-characterized, U.S. government data. The figures needed to encounter a great deal of prerequisites and the entire configuration must be completely archived (not at all like the DES figure). Once the hopeful algorithms had been deposited, a few years of examination as cryptographic meetings occurred. In the first adjust of the opposition fifteen algorithms were acknowledged and this remained contracted to five in the 2<sup>nd</sup> adjust. The algorithms remained tried for productivity and safety both by a percentage of the world's finest freely eminent cryptographers and NIST themselves.

Later this examination NIST at last picked an algorithm introduced as Rijndael. Rijndael was titled after the name of, who created and deposited it - Dr. Joan Daemen from Proton World International & Dr. Vincent Rijmen, a postdoctoral scientist in the Electrical Engineering from Department of Katholieke Universiteit Leuven (two Belgian cryptographers). On November the 26<sup>th</sup> of 2001, AES (that is an institutionalized rendition of Rijndael) turned into a FIPS standard (FIPS 197). [1]

### 3.3. AES CIPHER AND DECIPHER

Similar to DES, AES is a symmetric block cryptograph, which implies it utilizes the identical key for the purpose of encryption and decryption. Notwithstanding, AES is truly not the same as DES in various means. The algorithm Rijndael takes into consideration a mixture of block and key sizes and not only the 56 and the 64 bits of the DES block and the key size. The block & the key can indeed be picked freely from 160, 196, 128, 256, and 224 bits and doesn't need to be identical. Then again, the AES standard shapes that algorithm can just acknowledge the block size of 128 bits and the decision of 3 keys - 192, 128, & 256 bits. Contingent upon which form is utilized, the designation of the standard is changed to AES-192, AES-256 or AES- 128 separately. And these contrasts AES varies from DES in which isn't a feistel structure. Review that in a feistel structure, a large portion of the information block is utilized to change the other 50% of the information block and afterward the parts are exchanged. For this situation the whole information block is prepared in parallel throughout each one round utilizing replacements and stages.

Various AES factors rely on upon the key length. For instance, if the key size utilized is 128 then the amount of iterations is ten while it is fourteen and twelve for 256 & 192 bits separately. At current the utmost widely recognized key size liable to be utilized is 128 bit key. This portrayal of AES algorithm consequently depicts this specific execution.

Rijndael was planned to have the subsequent features:

- Battle in contradiction of every recognized attacks.
- Rapidity and code firmness on a widespread range of platforms.
- Blueprint Easiness.

The complete assembly of AES can be grasped through fig: 3.3. The input is just a single data of 128 bit for the purpose of decryption & encryption and is recognized as the **in** dimension. This data is imitated into a **state** dimension which is adapted at every phase of the algorithm and later imitated to an output dimension (see figure). Both the plain text and key are portrayed as a 128 bit square dimension of bytes. This key is later expanded into a dimension of key schedule words (32 bits) (the **w** matrix). It need to be noted that the ordering of bytes within the **in** matrix is by column. The same is applicable to the **w** dimension. [1]

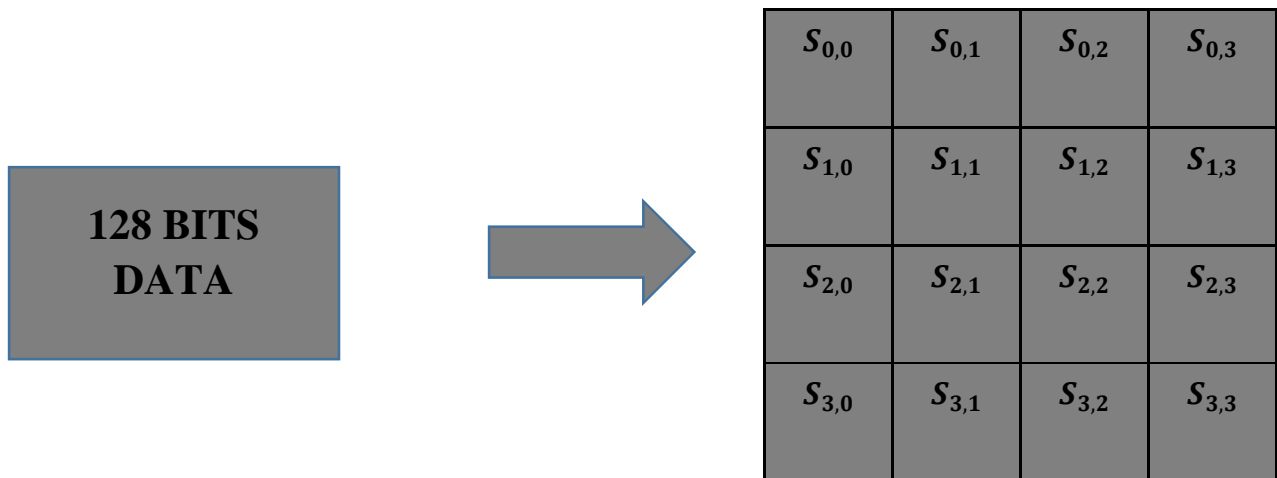


Fig 3.3: Conversion of 128 bits of data to State matrix

The above matrix in fig 3.3 is the state matrix who's each element is of one byte data.

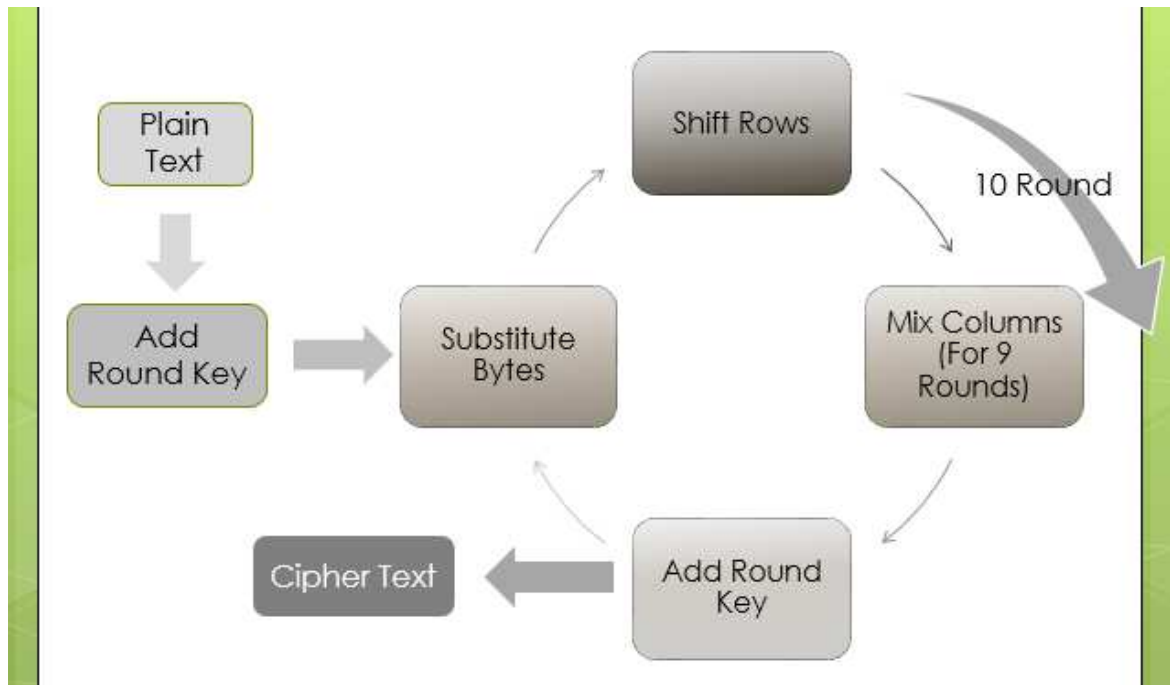


Figure 3.3(a): Block Diagram of Encryption

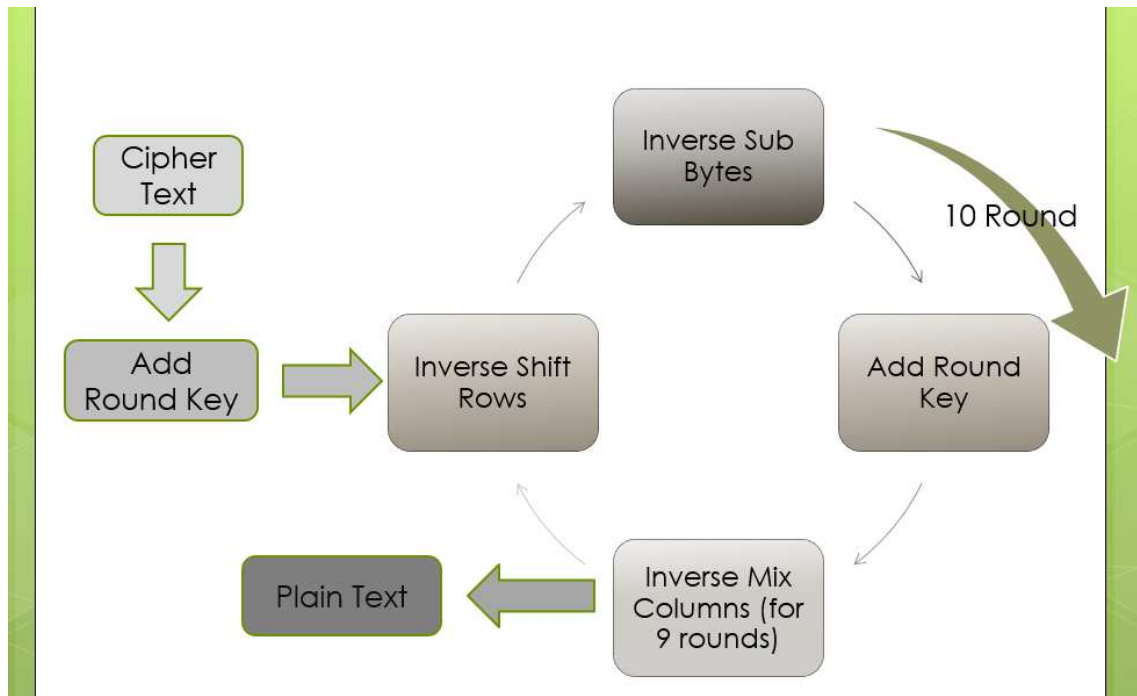


Figure 3.3(b) Block diagram of Decryption



### 3.3.1. INNER WORKING OF ROUNDS

The algorithm initiates from Add round key process followed by total of 9 iterations of four processes and the 10th iteration of 3 processes. This is applicable for both encryption and decryption including a special case that each process of an iteration the decryption algorithm is the reverse of its corresponding process from encryption algorithm. 4 processes used are as given below in fig 3.3.1(a). [1]

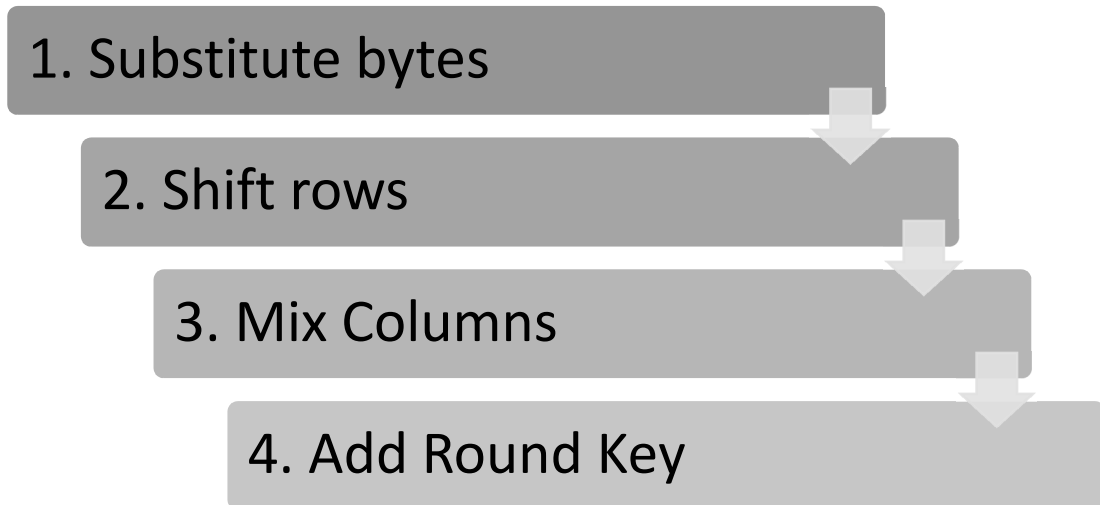


Fig 3.3.1(a): Four Stages of Encryption

The 10th iteration just doesn't use the **Mix Columns transformation**. The decryption algorithm initiates from an Add round key process trailed by 9 iterations of decryption process which comprises of the subsequent processes shown in fig 3.3.1(b).

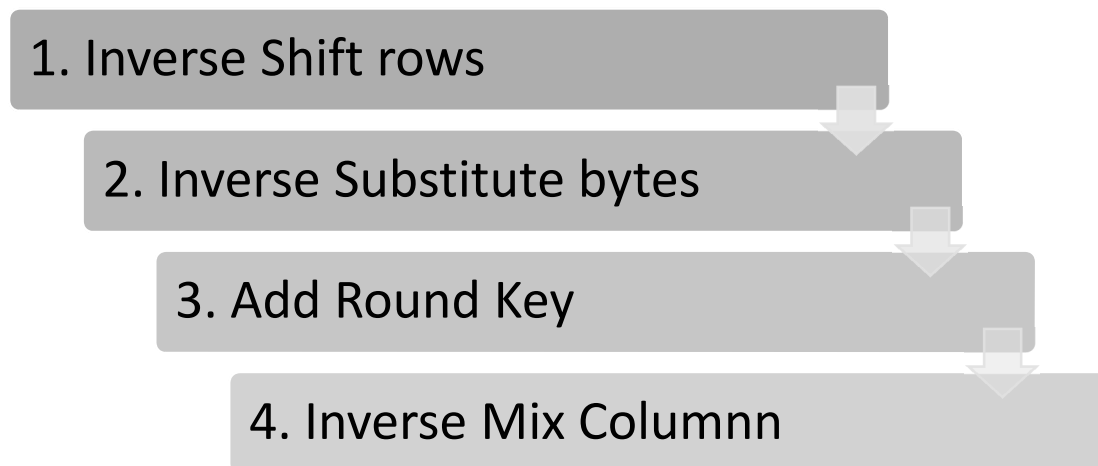


Fig 3.3.1(b): Four Stages of Decryption

Yet again, the 10th iteration just ignores the **Inverse Mix Columns transformation** process. For each of these processes will now be well-thought-out in more detail.

### 3.4. STAGES IN CIPHER AND DECIPHER

#### 3.4.1. BYTE SUBSTITUTION

**Byte Substitution** is basically a lookup table utilizing a  $16 \times 16$  double dimension of byte values known as **s-box**. This dimension comprises of every conceivable combos of 8 bit sequence ( $2^8 = 16 \times 16 = 256$ ). Nonetheless, the s-box isn't only an arbitrary stage of these qualities and there is an overall-characterized technique for making the s-box matrix. The architects of Rijndael demonstrated how this was carried out dissimilar to the s-box DES for which not at all justification was given. We won't be excessively interested here how the s-boxes are created and can basically take them as lookup tables. Again the dimension that gets worked upon all around the encryption is called state-matrix.

We need to be interested with how this framework is influenced in every one iteration. For this specific adjust every byte is linked into another 8 bits in the accompanying way: the left-hand side 4 bits of the half word is utilized to determine a specific row of the s-box and the right-hand side 4 bits tags a specific column. For instance, the 8 bits {95} (wavy sections speak to hex values in FIPS PUB 197) chooses line nine segment five that eventually comes out to hold the quantity {2a}, which is utilized to modify the **state** matrix.

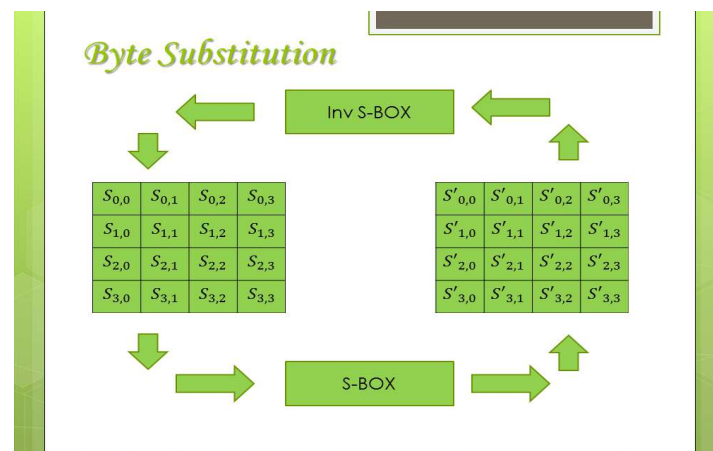


Figure 3.4.1: Byte Substitution

The Inverse byte substitution change makes utilization of an Inverse s-box. For this situation what is wanted is to choose the quality {2a} and get the worth {95}. The s-box is intended to be impervious to called cryptanalytic ambushes. Particularly, the Rijndael engineers looked for a plan that has a low connection between data bits and yield bits, and the property that the yield can't be depicted as a straightforward numerical capacity of the information. Also, the s-box has no altered focuses (s-box (a) = an) and no inverse settled focuses (s-box (a) = -a) where (an) is the bitwise compliment of a. The s-box must be invertible if decrypting is to be conceivable (Is-box[s-box (a)] = a) be that as it may it ought not to be its counter directionally toward oneself i.e. s-box (a) 6 = Is-box.

### 3.4.2. SHIFT ROW TRANSFORMATION

Shift Row Transformation is as displayed in figure 3.4.2. This is a humble permutation and nothing more. It works as below:

- The very initial row (i.e. row 0<sup>th</sup>) of the **state matrix** isn't modified.
- The 2<sup>nd</sup> row is left shifted by 1 byte in a round path.
- The 3<sup>rd</sup> row is left shifted by 2 bytes in a round path.
- The fourth row is left shifted by 3 bytes in a round path.

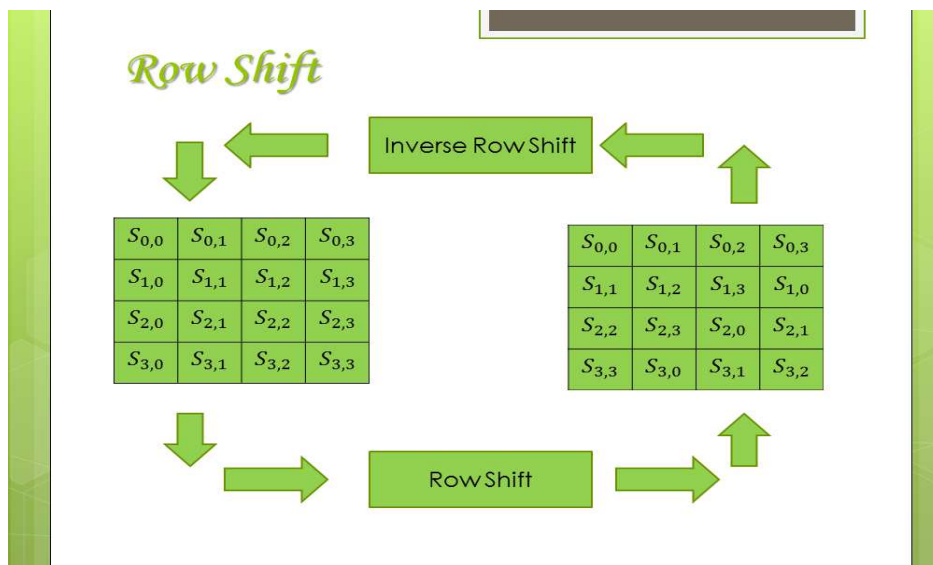


Figure 3.4.2: Row Shift Transformation

The Inverse Shift Rows conversion (called as Inv Shift Rows) performs these round movements in the inverse heading for each of the last three columns (the first column was unchanged in any situation). This process may not seem to do abundant yet in the event that you contemplate how the bytes are requested inside state then it could be seen to have significantly a greater amount of an effect. Keep in mind that state is dealt with as a cluster of four byte sections, i.e. the main section really speaks to bytes 1, 2, 3 and 4. A one byte movement is in this way a direct separation of 4 bytes. The conversion additionally guarantees that the 4 bytes of 1 segment are extent out to 4 separate segments.

### 3.4.3. MIX COLUMN TRANSFORMATION

MIX COLUMN TRANSFORMATION is essentially a substitution yet it makes utilization of math of GF ( $2^8$ ). Every segment is worked on separately. Every byte of a segment is charted into another esteem that is a capacity of each of the 4 bytes in the section. The conversion might be dictated by the accompanying grid increase on state demonstrated in fig 3.4.3

Every component of the item framework is the entirety of results of components of one line and one segment. For this situation the unique augmentations & multiplication are achieved in GF ( $2^8$ ). The Mix Columns change of a solitary segment  $j$  ( $0 \leq j \leq 3$ ) of state could be communicated as: Where  $x$  means multiplication over the finite field GF ( $2^8$ ).

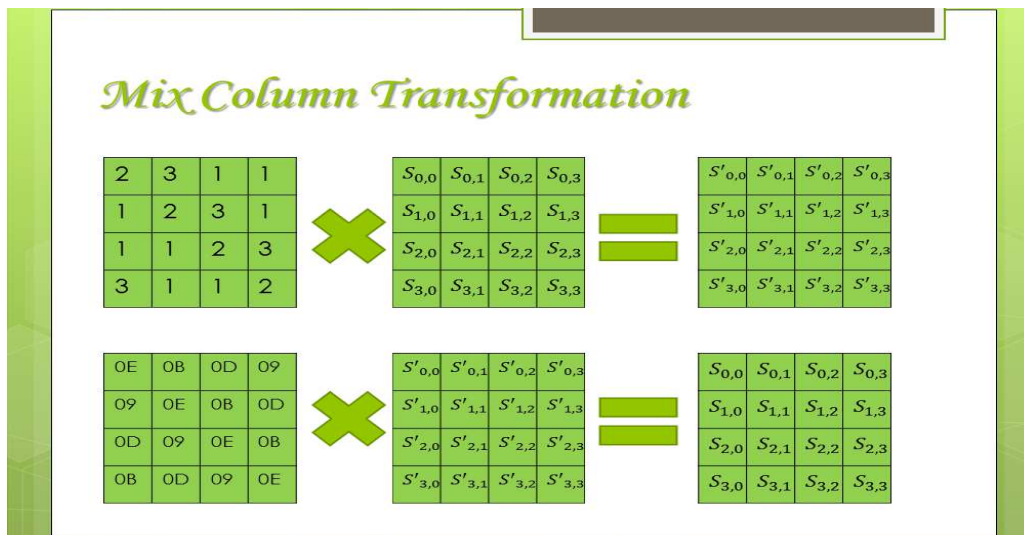
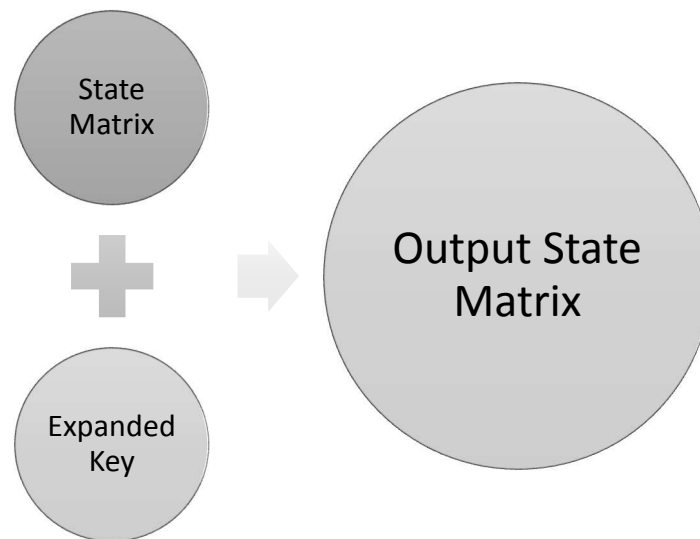


Figure 3.4.3: Mix column Transformation

### 3.4.4. ADD ROUND KEY

In this process (called as Add Round Key) the 128 bits of state are bitwise Xored through the 128 bits of the round key. The procedure is seen as a column wise process between the word of a state column and one WORD of the round key. This conversion is as basic as would be prudent which benefits in effectiveness yet it additionally influences all of state.



### 3.4.5. KEY EXPANSION UNIT

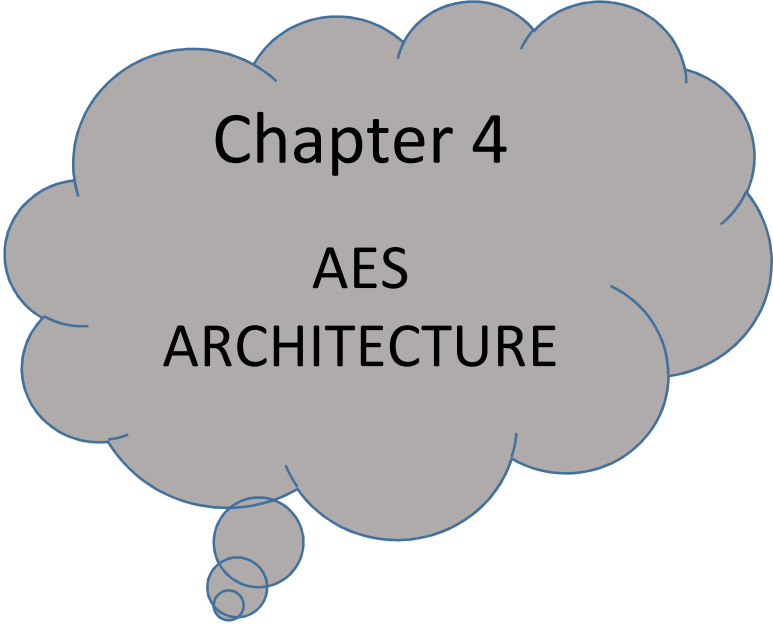
The AES key extension calculation takes input as a 4-word key and crops a direct cluster of 44 words (32 bits). Every one round uses 4 of these words. Each one expression holds 32 bytes which implies each one sub key is 128 bits in length.

The key is duplicated into the initial four expressions of the stretched key. The rest of the stretched key is packed in 4 words at once. Each one included word  $w[i]$  hinge on the promptly going before word,  $w[i - 1]$ , and the expression 4 places back  $w[i - 4]$ . In 3 out of 4 cases, upfront XOR is utilized. For a statement whose location in the  $w$  exhibit is a numerous of four, a more random capacity is utilized.

1. **RotWord** just rotates the word data by a one-byte round left movement. This implies that a data word [a0, a1, a2, a3] is changed into [a1, a2, a3, a0].
2. **SubWord** substitutes every bytes of the word using byte substitution method , utilizing the S-box portrayed prior.
3. The consequence of above processes is bitwise XORed with round constant, known as Rcon[j].

The round constant (RCON) is a word (32 bit) which has the 3 right hand-side bytes are zero every time. Therefore, the result of an XOR of the word with Rcon is just only to achieve an XOR on the left hand-side byte of word. The round constant is dissimilar for each iteration and is well-defined as  $Rcon[j] = (RC[j], 0, 0, 0)$ , with  $RC[1] = 1$ ,  $RC[j] = 2 \cdot RC[j - 1]$  and the multiplication is done over the GF ( $2^8$ ).

The key expansion remained intended to be impervious to recognized cryptanalytic assaults. The consideration of a round-needy round steady dispenses with the symmetry, or comparability, between the courses in which adjust keys are produced in diverse iterations [1].



Chapter 4  
AES  
ARCHITECTURE

## 4. AES ARCHITECTURE

We have designed our AES architecture in VHDL Language using Xilinx 14.2 for Spartan 3E XC3s500e FPGA. Our architecture takes 128 bits of data as input along with the 128 bits of key along with three control signal clk, go\_i, and reset signal each of single bit. The block diagram of the AES block is provided below in fig 4.1(a).

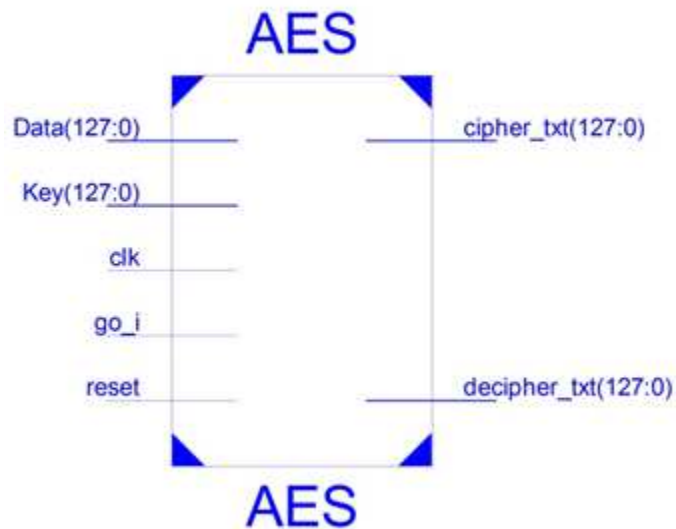


Fig 4(a): AES Block Diagram

Inside of AES comprises of Cipher block and a Decipher block. Cipher block is connected to decipher block as given below in fig 4(b). Cipher block takes all the input provided to AES block and give us a Cipher Text of 128 bits as output. Cipher block controls the processing of decipher block, it keeps the decipher block in wait state unless it is ready with the cipher text, detail explanation would be provided later. Decipher block takes the cipher text as input and provide us the decipher text which would be exactly similar to the input data.



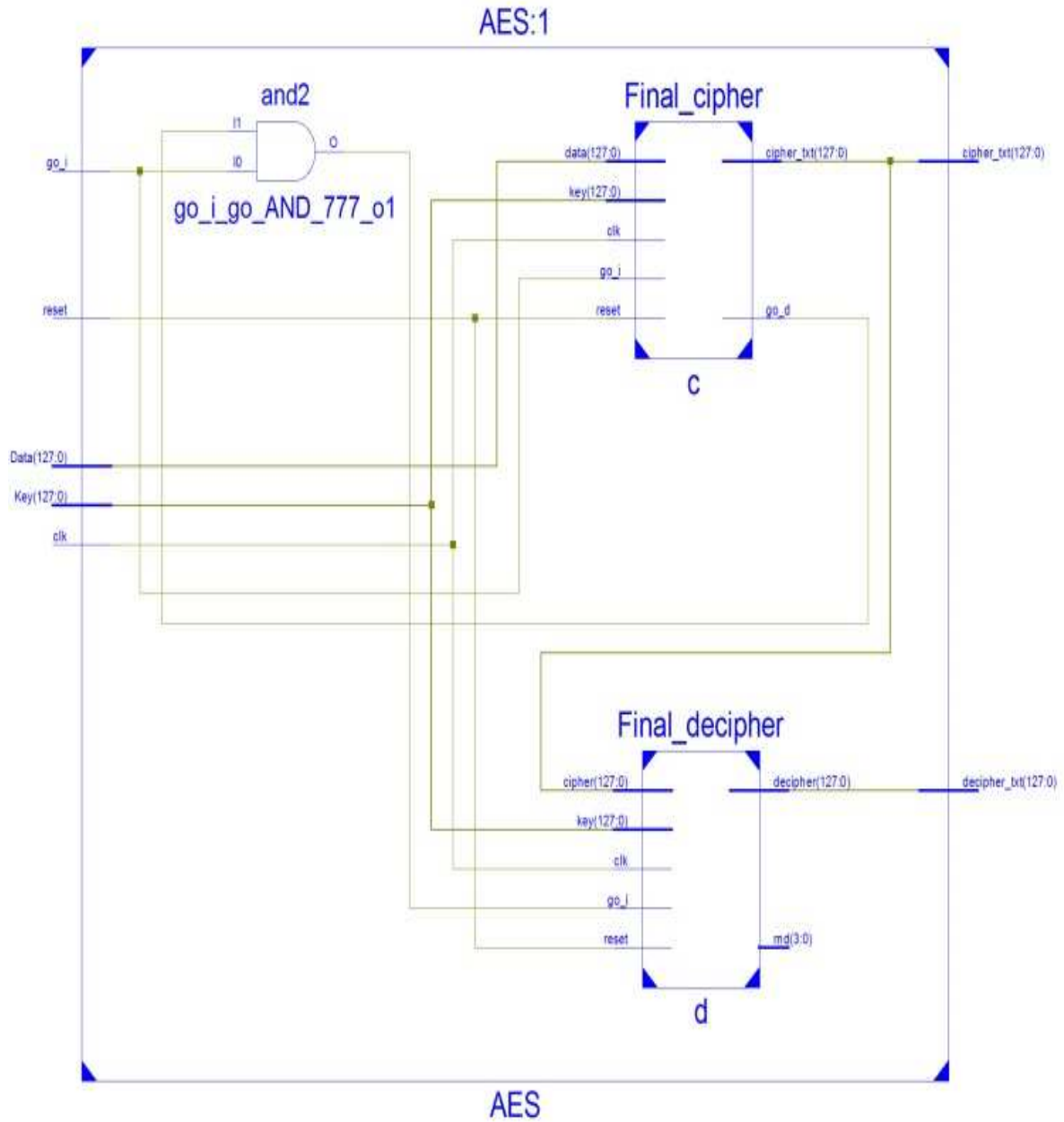


Fig 4(b): Internal Schematic of AES block

## 4.1. CIPHER

Cipher block is basically used for the encryption of data, which takes in 128 bits of input data and 128 bits of key. This block process the data only at the appropriate signal given by the three control signal namely clk (it is a clock signal), reset (used to reset various data), and go\_i (it controls the control unit). It provide us with the cipher text of 128 bits and a control signal go\_d of one bit used to control the decipher block. We will discuss the complete schematic, data path, control unit of the cipher block below in fig 4.1.

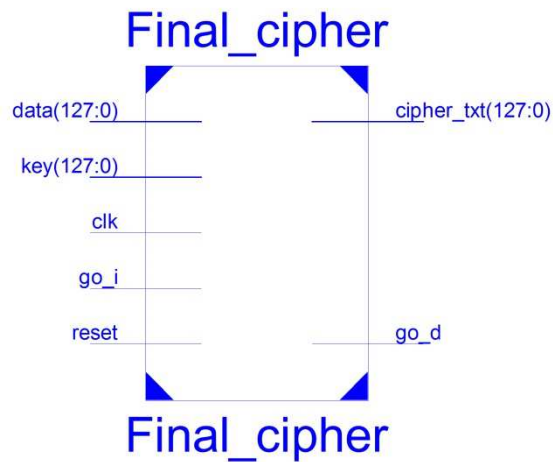


Fig 4.1(a): block schematic of data path of cipher block.

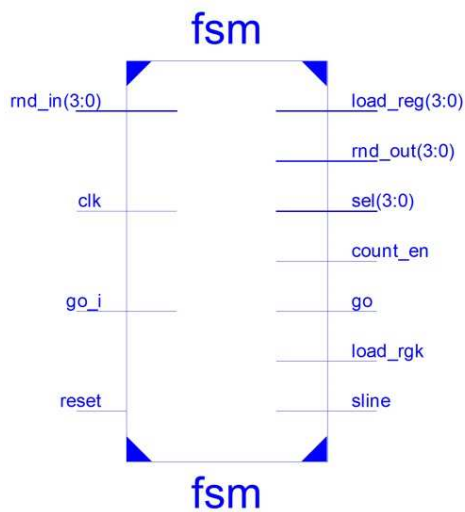


Fig 4.1(b): Control Unit of Cipher block

### 4.1.1. SCHEMATIC

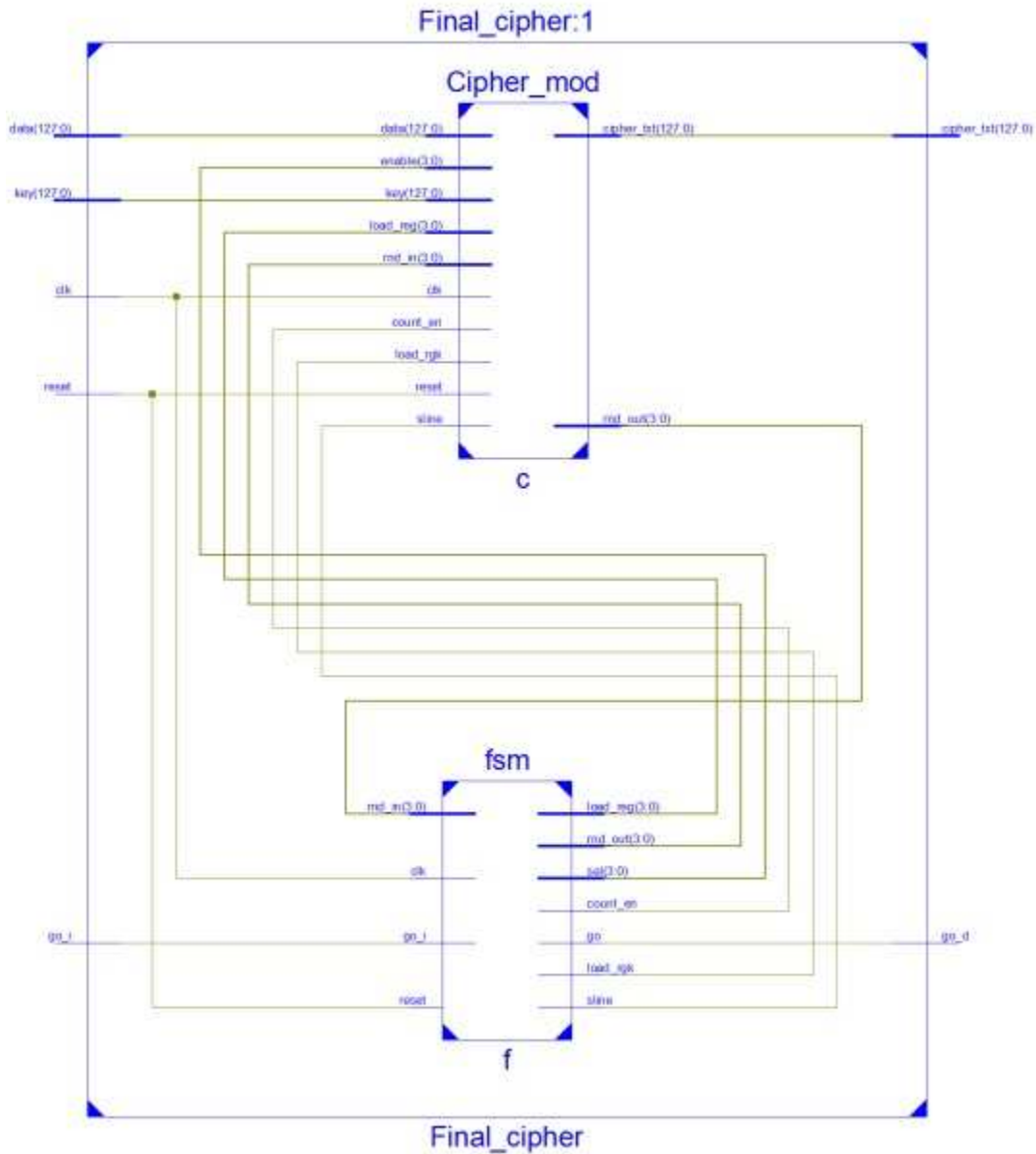


Fig.4.1.1: internal schematic of Cipher block.

The above figure (Fig.4.1.1) shows the connection between the control unit and the datapath of the cipher block. The control unit controls the datapath (cipher\_mod) with various control signal namely count\_en (used to control the counter in datapath), load\_reg (used to load data at the registers), load\_rgk (used to load key register), enable (used to control various transformation blocks), sline (used as select signal to select between input data and last round result), rnd\_in and rnd\_out are the round signal containing round count.

### 4.1.2. DATA PATH

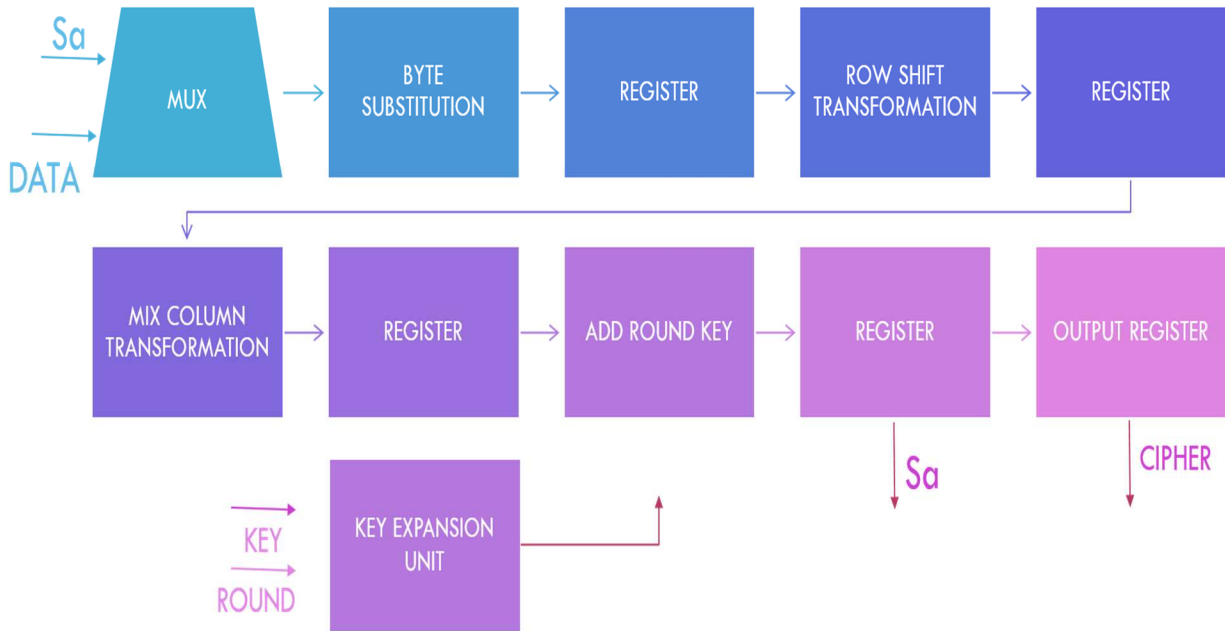


Fig.4.1.2: Datapath for AES CIPHER

The above figure (Fig .4.1.2) is the datapath of the cipher block which shows the connection of the various components of the datapath and the flow of data. It consist of 4 transformation processes named as byte substitution, row shift transformation, mix column transformation and add around key which uses a key expansion unit which produces a new key each round. The above 4 processes takes in a 128 bit data and transform it according an algorithm. The output signal Sa is feedback to multiplexer which selects between it and input data depending upon the output of the control unit. The registers above load on the appropriate condition provided by the control unit. Each transformation block also execute there algorithms if the control unit allows them to. In short the data path works under the guidance of the control unit.

### 4.1.3. STATE DIAGRAM

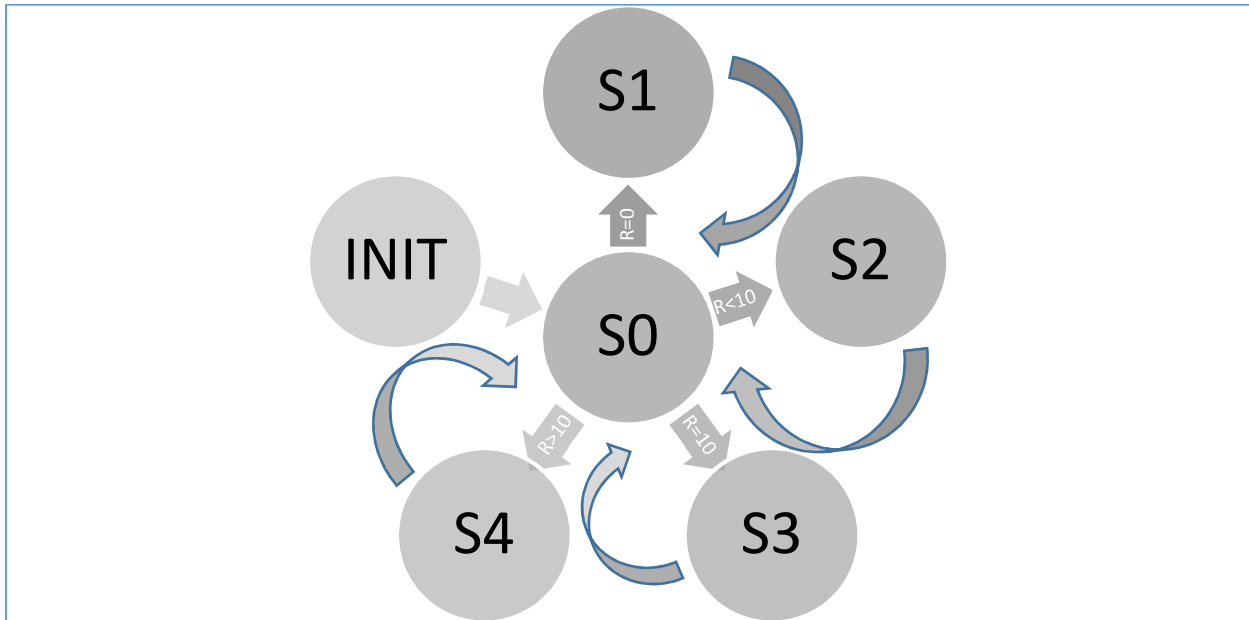


Fig.4.1.3: State diagram of Cipher

The above figure (Fig – 4.1.3) is the state diagram of cipher which shows how the state transformation takes place which is started from init state. It proceeds to S0 when it receives a signal named as go\_i. This state has four states to follow, it goes to state S1 when round is 0. It goes to S2 when round is less than 10, to S3 for round = 10 and to S4 for round > 10. Each state from S1 – S4 returns back to S0. S4 sets the round count back to 0.

### 4.1.4. Tabular Description of State

The table 4.1.4 provided below clearly shows what control unit asks the data path to perform at various state which depends on round count. State init is the initial state or a reset state which resets all data to 0 and maintain the round count to 0. State S0 is the selection state where we select the data which is needed to process further i.e. between input data and the result of the previous round, in round 0 it selects the input data whereas for rest of the round it reselects the output of the previous round. State S1 is used to process only add round key. State S2 is used to process all the four transformation processes. State S3 is used to process all the processes except mix column transformation and also to load the output register. State S4 is just like the init state

used to reset round to 0. All the register are loaded excluding output register in state S1 and S2 and including output register in state S3. Counter is enabled in state S1, S2 and S3.

|                           | INIT | S0    | S1    | S2    | S3    | S4  |
|---------------------------|------|-------|-------|-------|-------|-----|
| BYTE SUBSTITUTION         | NO   | NO    | NO    | YES   | YES   | NO  |
| ROW SHIFT TRANSFORMATION  | NO   | NO    | NO    | YES   | YES   | NO  |
| MIX COLUMN TRANSFORMATION | NO   | NO    | NO    | YES   | NO    | NO  |
| ADD ROUND KEY             | NO   | NO    | YES   | YES   | YES   | NO  |
| RESULT REGISTER LOADING   | NO   | NO    | NO    | NO    | YES   | NO  |
| REGISTERS LOADING         | NO   | NO    | YES   | YES   | YES   | NO  |
| SLINE                     | 0    | 0/1   | 0     | 1     | 1     | 1   |
| ROUND                     | 0    | Input | Input | Input | Input | 0   |
| COUNTER                   | OFF  | OFF   | ON    | ON    | ON    | OFF |

Table 4.1.4 State description of control unit of cipher block

## 4.2. DECIPHER

Decipher block is basically used for the decryption of cipher text, which takes in 128 bits of cipher text from cipher block and 128 bits of key. This block process the data only at the appropriate signal given by the three control signal namely clk (it is a clock signal), reset (used to reset various data), and go\_i (it controls the control unit). It provide us with the decipher text of 128 bits. We will discuss the complete schematic, data path, control unit of the cipher block below in fig 4.2.

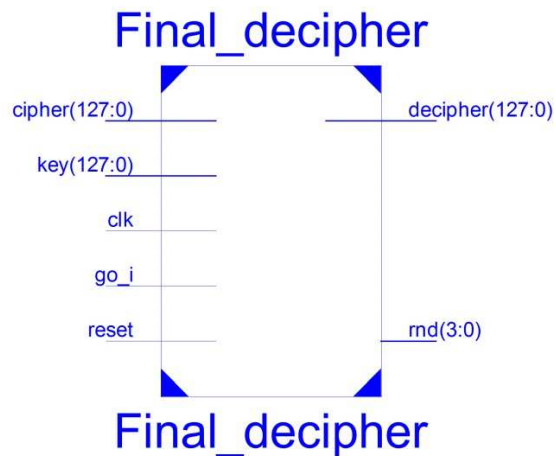


Fig 4.2(a): block schematic of decipher.

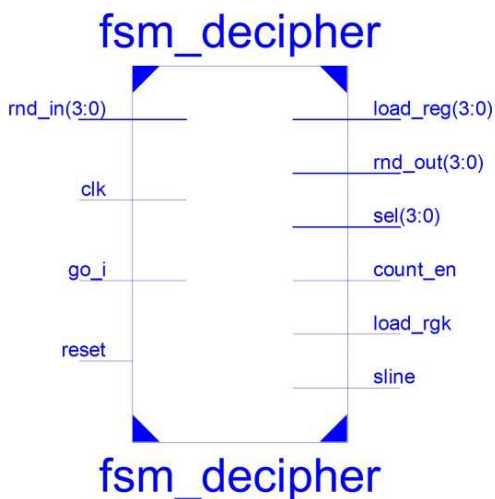


Fig 4.2(b): Control Unit of Decipher block

### 4.2.1. SCHEMATIC

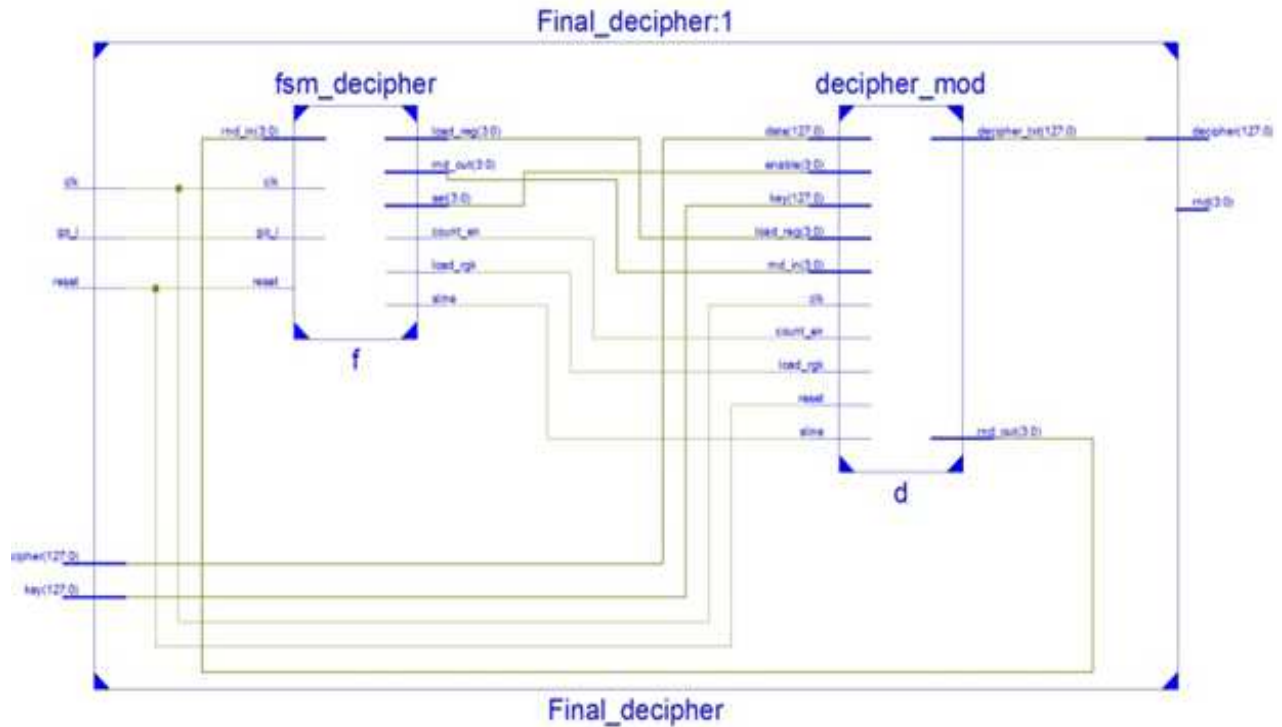


Fig.4.2.1: internal schematic of decipher block.

The above figure (Fig.4.2.1) shows the connection between the control unit and the datapath of the decipher block. The control unit controls the datapath (decipher\_mod) with various control signal namely count\_en (used to control the counter in datapath), load\_reg (used to load data at the registers), load\_rgk (used to load key register), enable (used to control various transformation blocks), sline (used as select signal to select between input data and last round result), rnd\_in and rnd\_out are the round signal containing round count.



## 4.2.2. DATA PATH

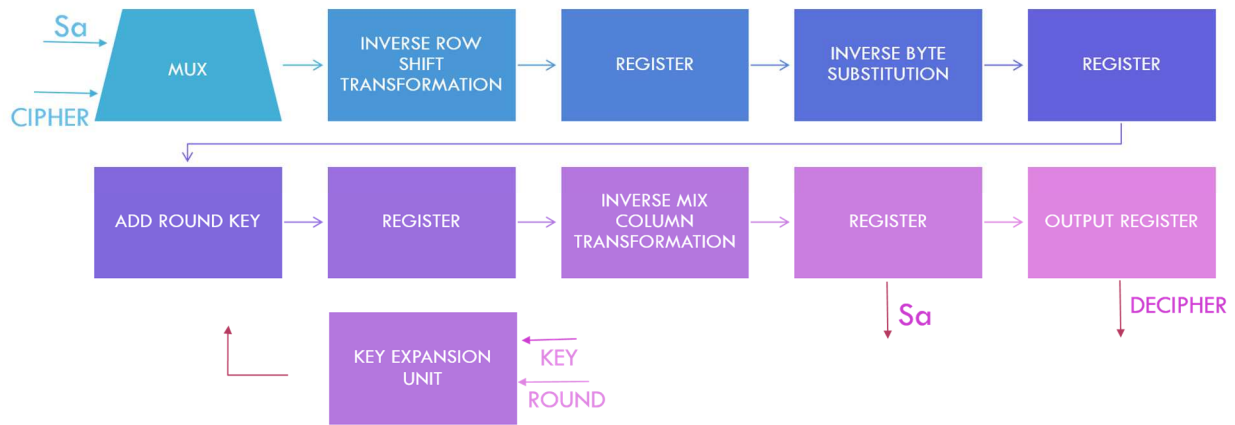


Fig.4.2.2: Datapath for AES DECIPHER

The above figure (Fig – 3.2) is the datapath of the decipher block which shows the connection of the various components of the datapath and the flow of data. It consist of 4 transformation processes named as inverse byte substitution, inverse row shift transformation, inverse mix column transformation and add around key which uses a key expansion unit which produces a new key each round. The above 4 processes takes in a 128 bit data and transform it according an algorithm. The output signal Sa is feedback to multiplexer which selects between it and input data depending upon the output of the control unit. The registers above load on the appropriate condition provided by the control unit. Each transformation block also execute there algorithms if the control unit allows them to. In short the data path works under the guidance of the control unit.

### 4.2.3. STATE DIAGRAM

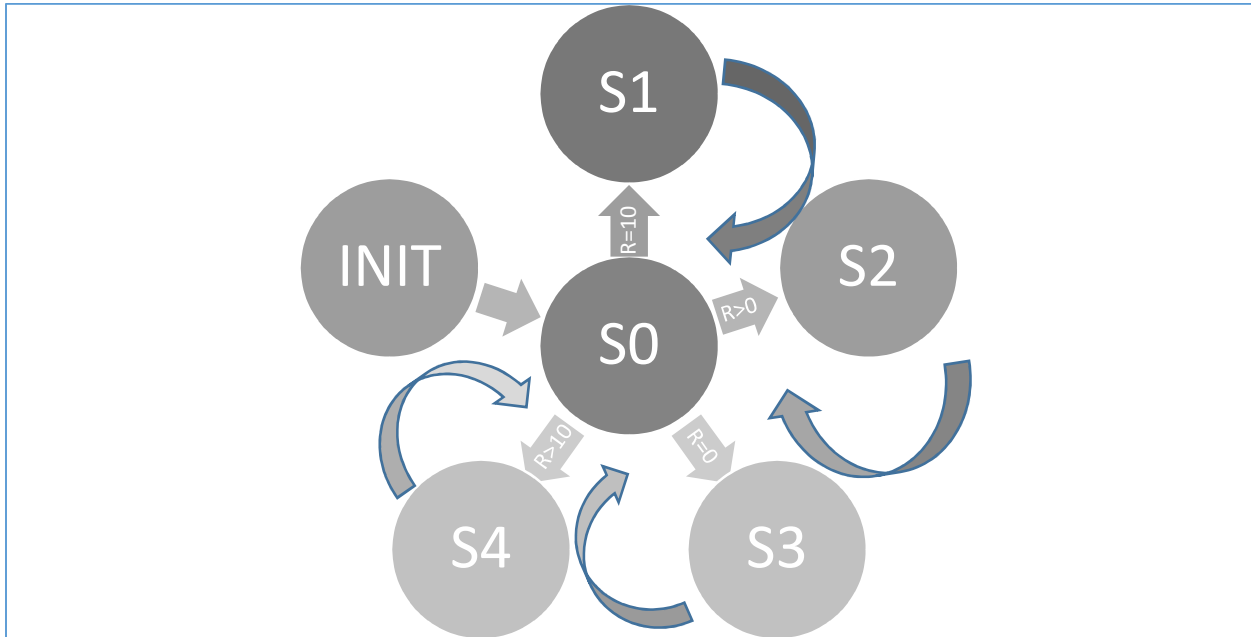


Fig.4.2.3: State diagram of Cipher

The above figure (Fig – 4.2.3) is the state diagram of decipher which shows how the state transformation takes place which is started from init state. It proceeds to S0 when it receives a signal named as go\_i. This state has four states to follow, it goes to state S1 when round is 10. It goes to S2 when round is greater than 0, to S3 for round = 00 and to S4 for round > 10. Each state from S1 – S4 returns back to S0. S4 sets the round count back to 10.

### 4.2.4. TABULAR DESCRIPTION OF STATE

The table provided below clearly shows what control unit asks the data path to perform at various state which depends on round count. State init is the initial state or a reset state which resets all data to 0 and maintain the round count to 10. State S0 is the selection state where we select the data which is needed to process further i.e. between input data and the result of the previous round, in round 0 it selects the input data whereas for rest of the round it reselects the output of the previous round. State S1 is used to process only add round key. State S2 is used to

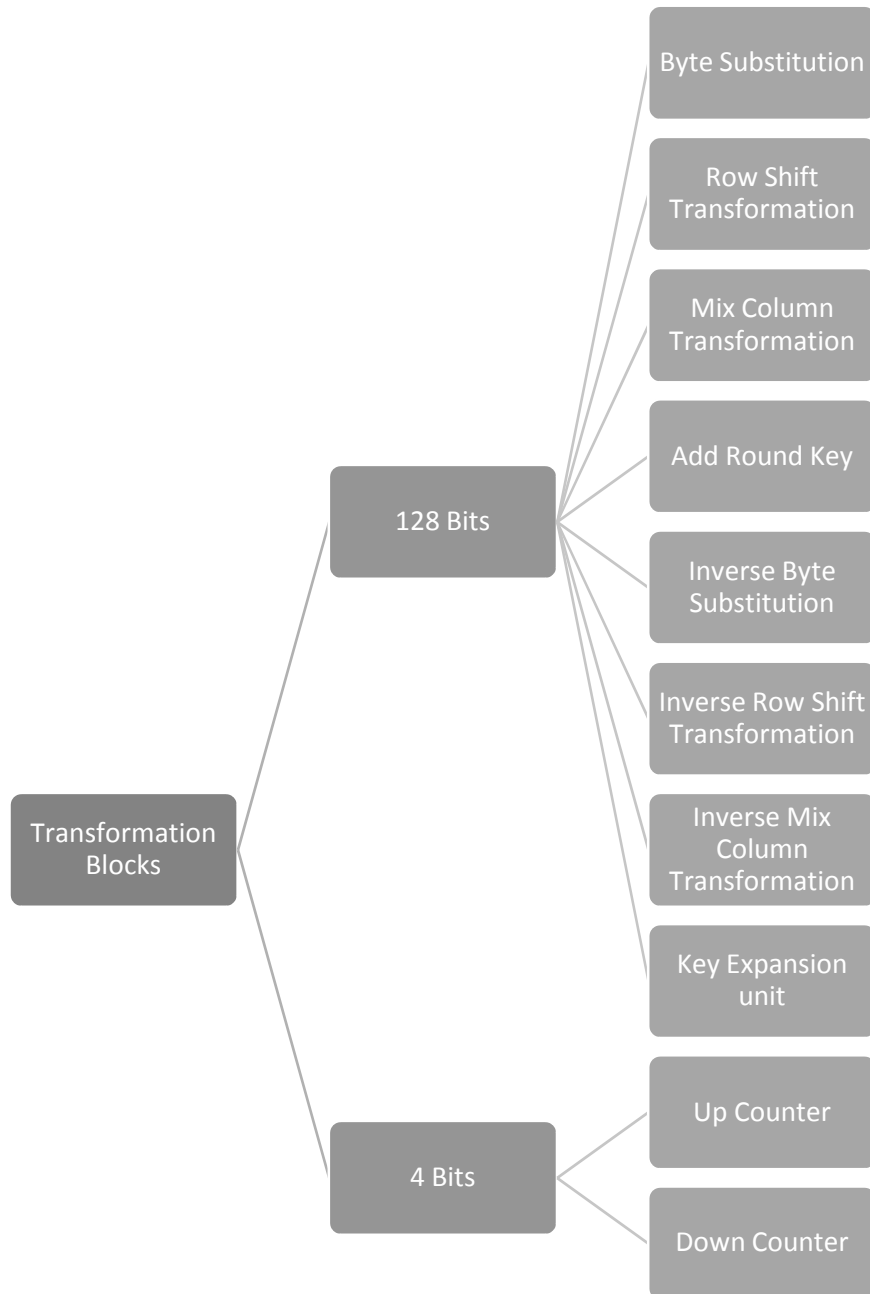
process all the four transformation processes. State S3 is used to process all the processes except inverse mix column transformation and also to load the output register. State S4 is just like the init state used to reset round to 10. All the register are loaded excluding output register in state S1 and S2 and including output register in state S3. Counter is enabled in state S1, S2 and S3.

|                                  | INIT | S0    | S1    | S2    | S3    | S4  |
|----------------------------------|------|-------|-------|-------|-------|-----|
| INVERSE BYTE SUBSTITUTION        | NO   | NO    | NO    | YES   | YES   | NO  |
| INVERSE ROW SHIFT TRANSFORMATION | NO   | NO    | NO    | YES   | YES   | NO  |
| INVERSE MIX COLUMN               | NO   | NO    | NO    | YES   | NO    | NO  |
| ADD ROUND KEY                    | NO   | NO    | YES   | YES   | YES   | NO  |
| RESULT REGISTER LOADING          | NO   | NO    | NO    | NO    | YES   | NO  |
| REGISTERS LOADING                | NO   | NO    | YES   | YES   | YES   | NO  |
| SLINE                            | 0    | 0/1   | 0     | 1     | 1     | 1   |
| ROUND                            | 10   | Input | Input | Input | Input | 10  |
| COUNTER                          | OFF  | OFF   | ON    | ON    | ON    | OFF |

Table 4.2.4: State description of control unit of decipher block

### 4.3. TRANSFORMATION BLOCKS

Transformation blocks takes the data and transform it according to an algorithm to another data. Each block transforms when the enable signal is high else it return the input as output when the signal is low. In this architecture we have designed two types of block depending upon the bit size, one for 128 bits of input and output other 4 bit of input and output.



### 4.3.1. Byte Substitution

We have already explained the main procedure of the byte substitution in section 3.4.1, here we are introducing the schematic of the process which is shown below in the fig 4.3.1. Its internal schematic uses a function called as STATE\_SUB, this function is defined along with other function stated in later stage is defined in package created by us as AES\_package. All the block process with a high enable, else returns the input as output.

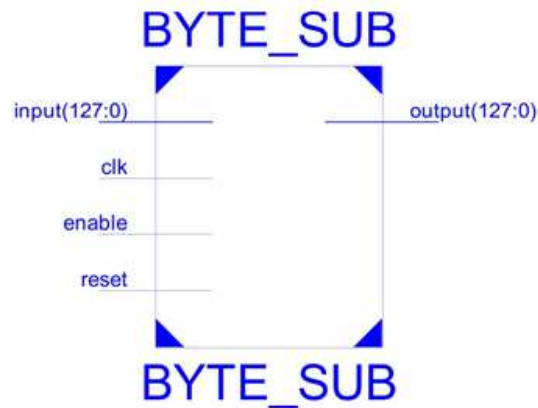


Fig 4.3.1: Block Schematic of Byte Substitution process

### 4.3.2. Row Shift Transformation

We have already explained the main procedure of the row shift transformation in section 3.4.2, here we are introducing the schematic of the process which is shown below in the fig 4.3.2. Its internal schematic uses a function called as row\_shift.

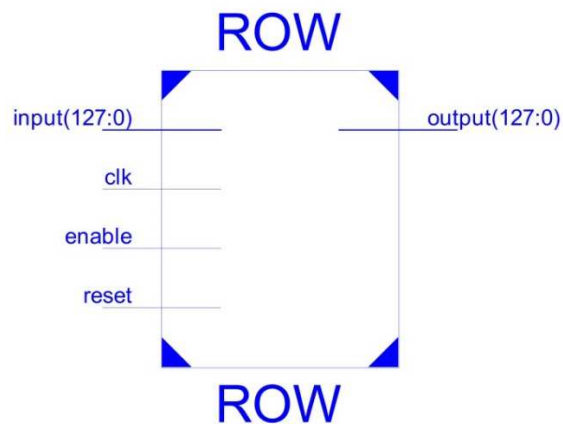


Fig 4.3.2: Block Schematic of Row Shift Transformation

### 4.3.3. Mix Column Transformation

We have already explained the main procedure of the mix column transformation in section 3.4.3, here we are introducing the schematic of the process which is shown below in the fig 4.3.3. Its internal schematic uses a function called as mix\_column.

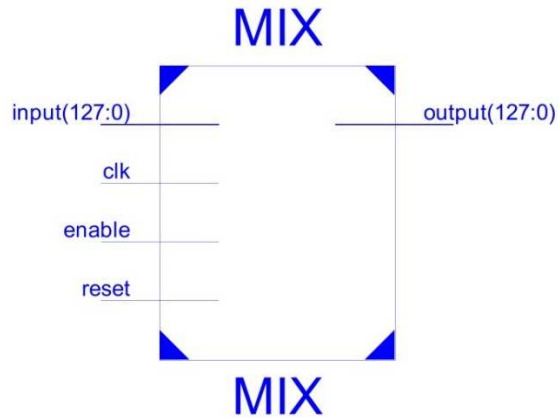


Fig 4.3.3: Block Schematic of Mix Column Transformation

### 4.3.4. Add Round Key

We have already explained the main procedure of add round key in section 3.4.4, here we are introducing the schematic of the process which is shown below in the fig 4.3.4. Its internal schematic uses a function called as add\_round\_key.

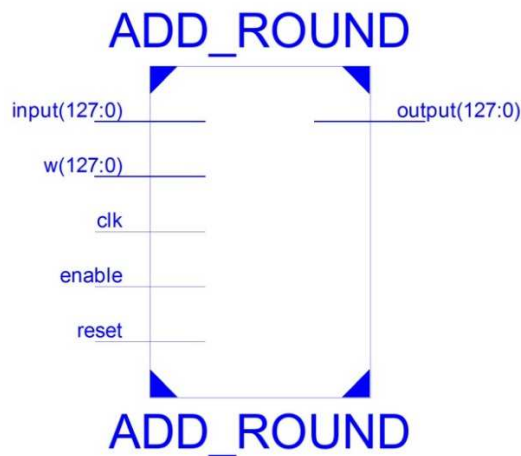


Fig 4.3.4: Block Schematic of Add Round Key

### 4.3.5. Inverse Byte Substitution

We have already explained the main procedure of the inverse byte substitution in section 3.4.1, here we are introducing the schematic of the process which is shown below in the fig 4.3.5. Its internal schematic uses a function called as STATE\_INV\_SUB.

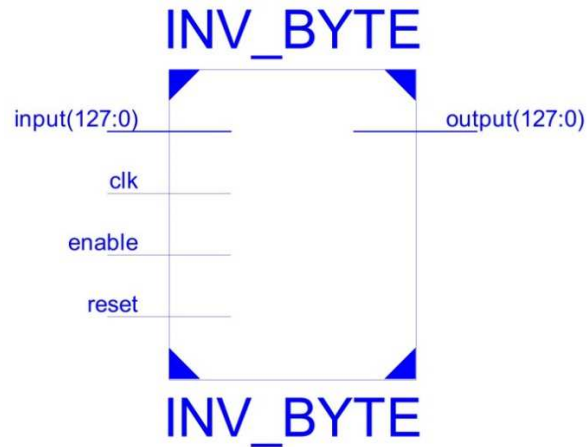


Fig 4.3.5: Block Schematic of Inverse Byte Substitution process

### 4.3.6. Inverse Row Shift Transformation

We have already explained the main procedure of the inverse row shift transformation in section 3.4.2, here we are introducing the schematic of the process which is shown below in the fig 4.3.6. Its internal schematic uses a function called as inv\_row\_shift.

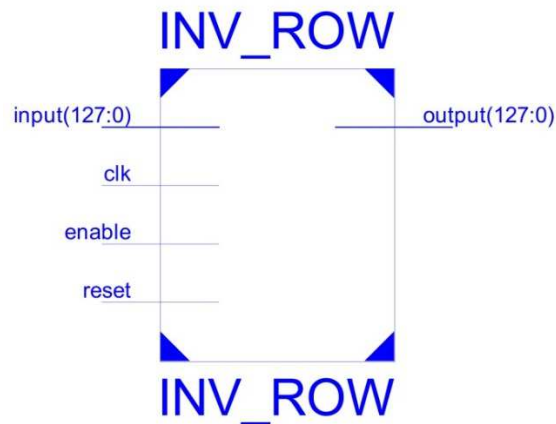


Fig 4.3.6: Block Schematic of Inverse Row Shift Transformation

### 4.3.7. Inverse Mix Column Transformation

We have already explained the main procedure of the inverse mix column transformation in section 3.4.3, here we are introducing the schematic of the process which is shown below in the fig 4.3.7. Its internal schematic uses a function called as `inv_mix_column`.

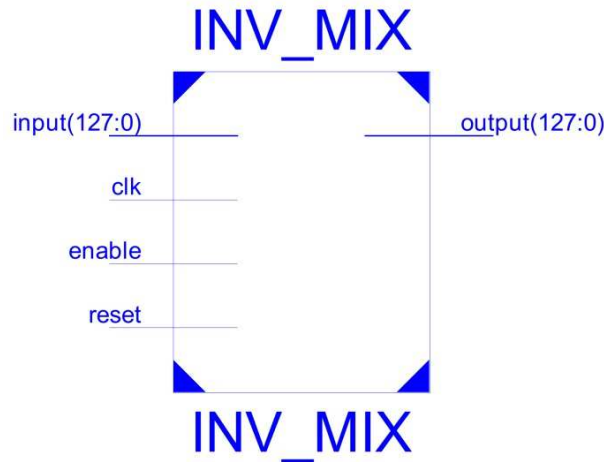


Fig 4.3.7: Block Schematic of Inverse Mix Column Transformation

### 4.3.8. Key Expansion unit

We have already explained the main procedure of the key expansion unit in section 3.4.5, here we are introducing the schematic of the process which is shown below in the fig 4.3.8. Its internal schematic uses a function called as `key_exp_unit`.

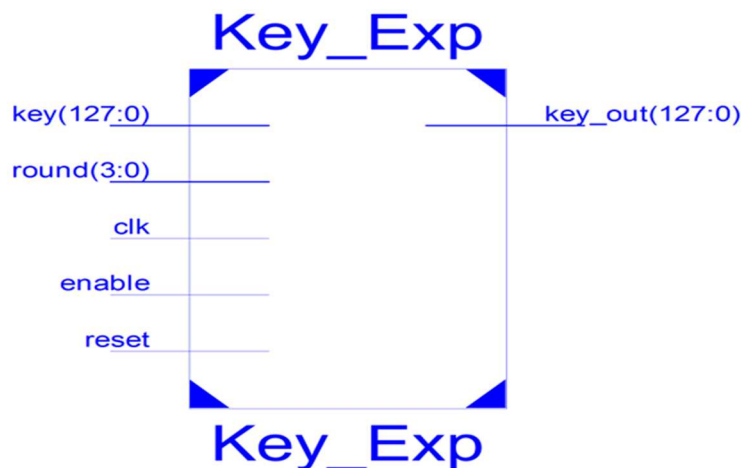


Fig 4.3.8: Block Schematic of Key Expansion Unit



### 4.3.9. Up Counter

This block is used to increase the round number in cipher block when we need to move to the next round. The schematic of the block is shown below in the fig 4.3.9.

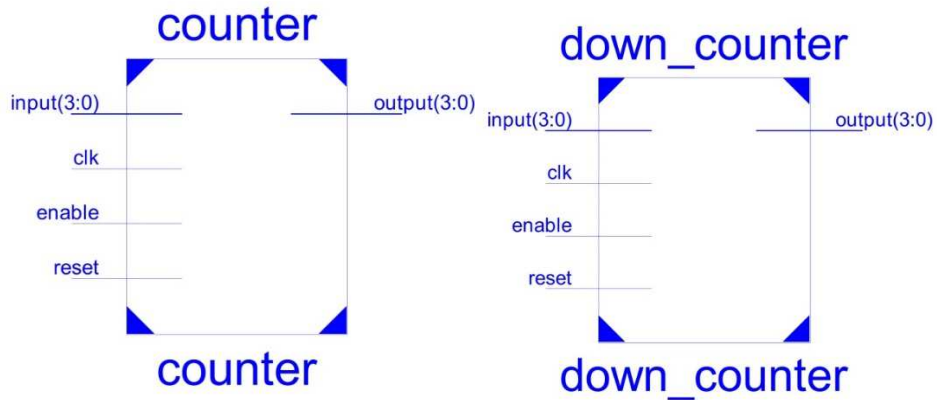


Fig 4.3.9: Block Schematic of the Up Counter, Fig 4.3.10: Block Schematic of the Down Counter

### 4.3.10. Down Counter

This block is used to decrease the round number in decipher block when we need to move to the next round. The schematic of the block is shown above in the fig 4.3.10.

### 4.3.11. Register

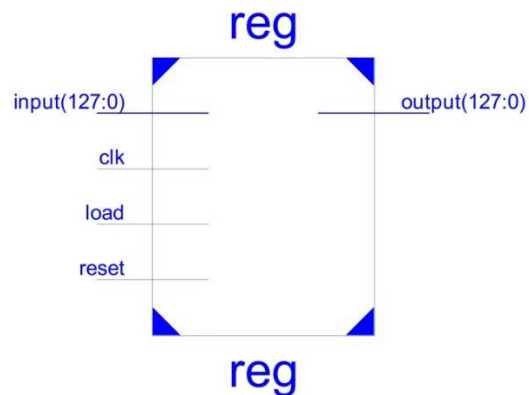
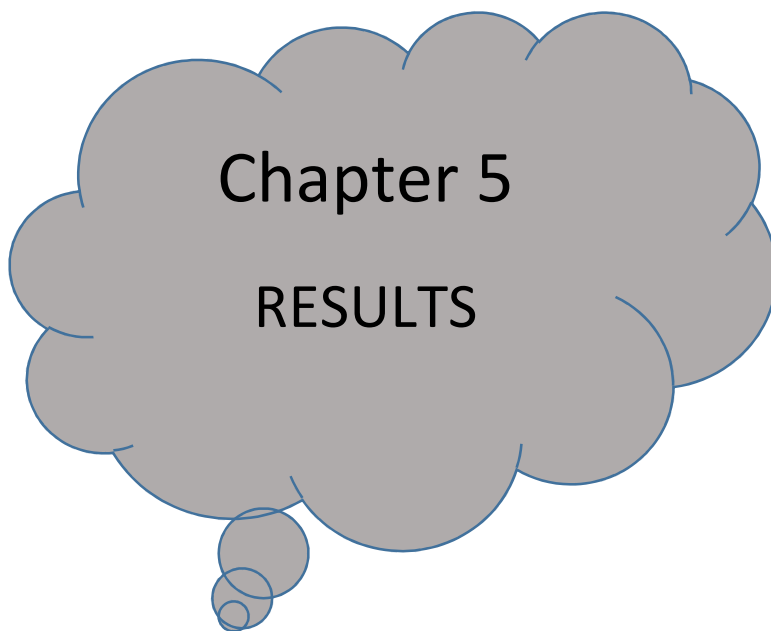


Fig 4.3.11: Block Schematic of a register

This block is used to load the 128 bits of data and save it for future processing, the block schematic of the register is shown above in fig 4.3.11.



## 5. RESULTS

### 5.1. MATLAB GUI IMPLEMENTATION

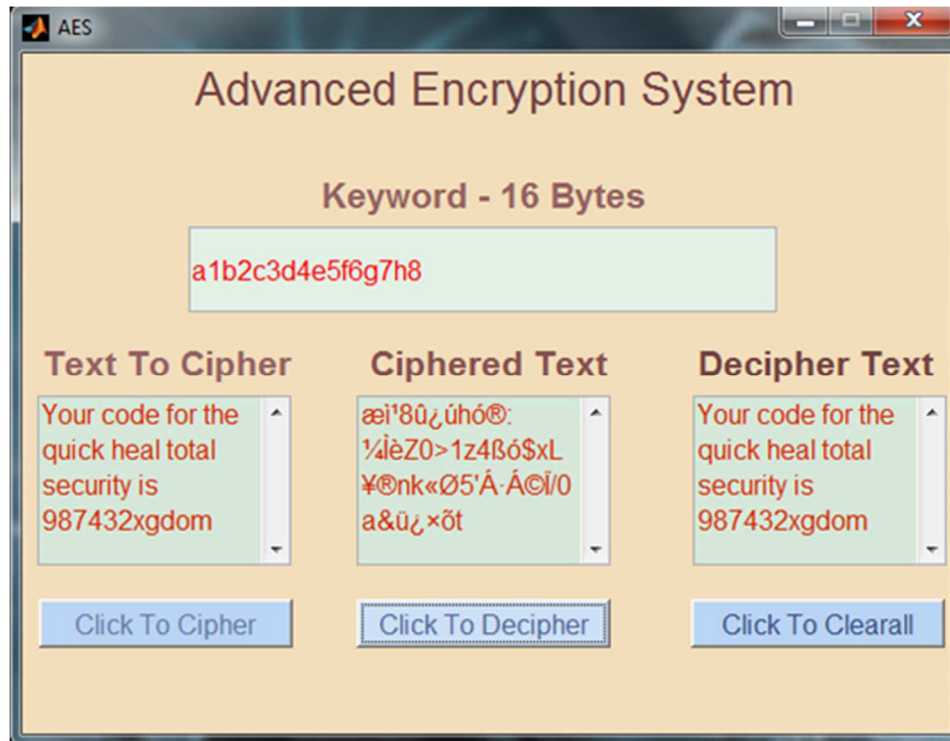


Fig 5.1 Gui Result

A Graphical User Interface was designed as shown above for the purpose of encryption and decryption using Advanced Encryption Standard algorithm. Here we have provided a 16 bytes (128 bits) key word and plain text of unknown length. We can clearly see the cipher text and the decipher text as generated.

## 5.2. VHDL SIMULATION RESULTS

### 5.2.1. Byte Substitution



Fig. 5.2.1: Byte Substitution waveform.

The above figure 5.2.1 signifies the waveforms produced by the substitution byte transformation. The input clock is of 10ns time period, Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.1.

### 5.2.2. Row Shift Transformation

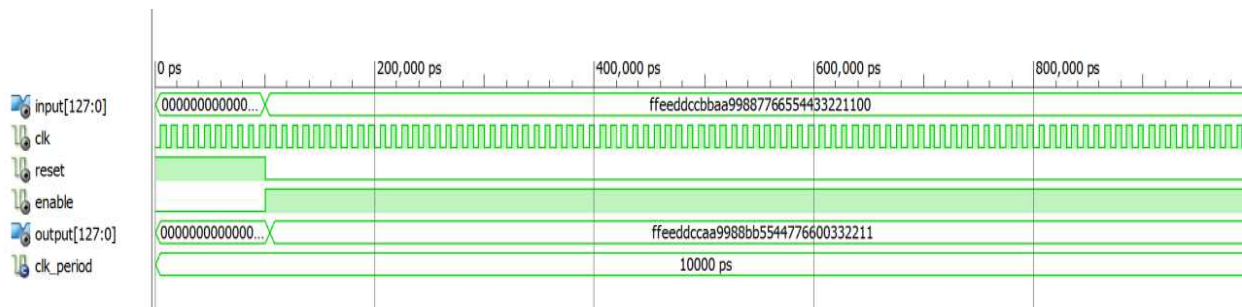


Fig.5.2.2: row shift transformation waveform.

The above figure 12 signifies the waveforms produced by the row shift transformation. The input clock is of 10ns time period Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.2.

### 5.2.3. Mix Column Transformation

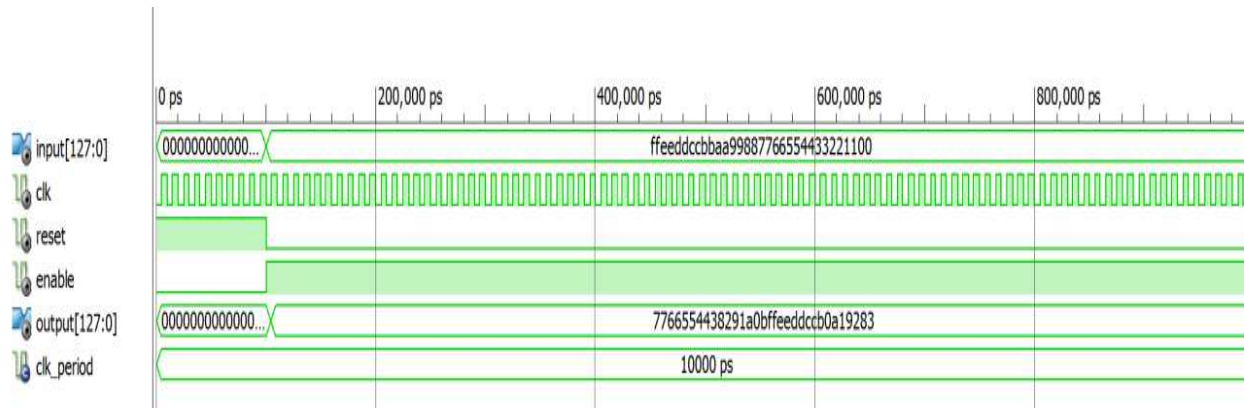


Fig.5.2.3: mix column transformation waveform.

The above Fig: 5.2.3 signifies the waveforms produced by the Mix Columns transformation and its block architecture. The input clock is of 10ns time period, Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.3.

### 5.2.4. Add Round Key

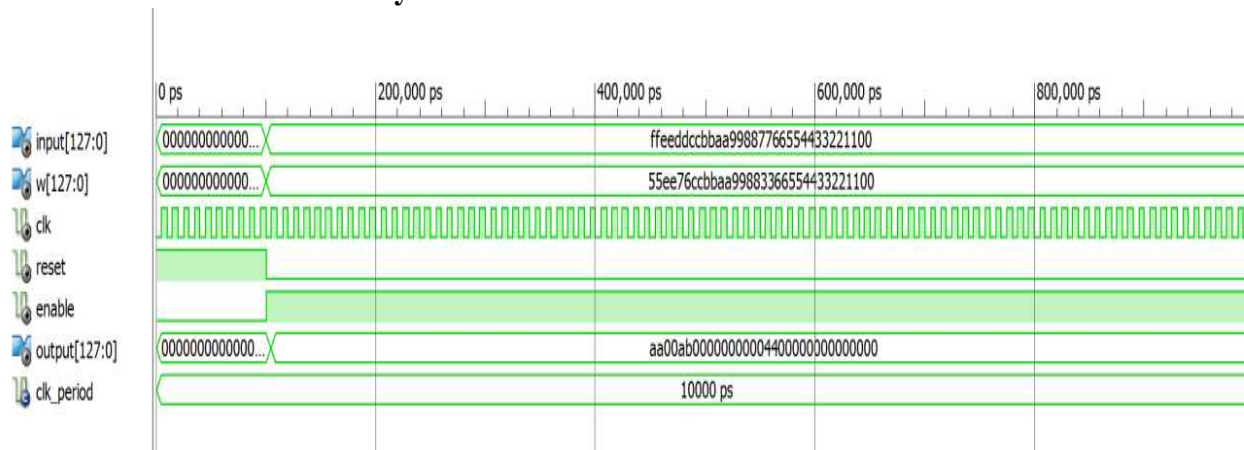


Fig.5.2.4: add round key transformation waveform

The above figure 14 signifies the waveforms produced by add round key transformation. The input clock is of 10ns time period, Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.4.

### 5.2.5. Inverse Byte Substitution

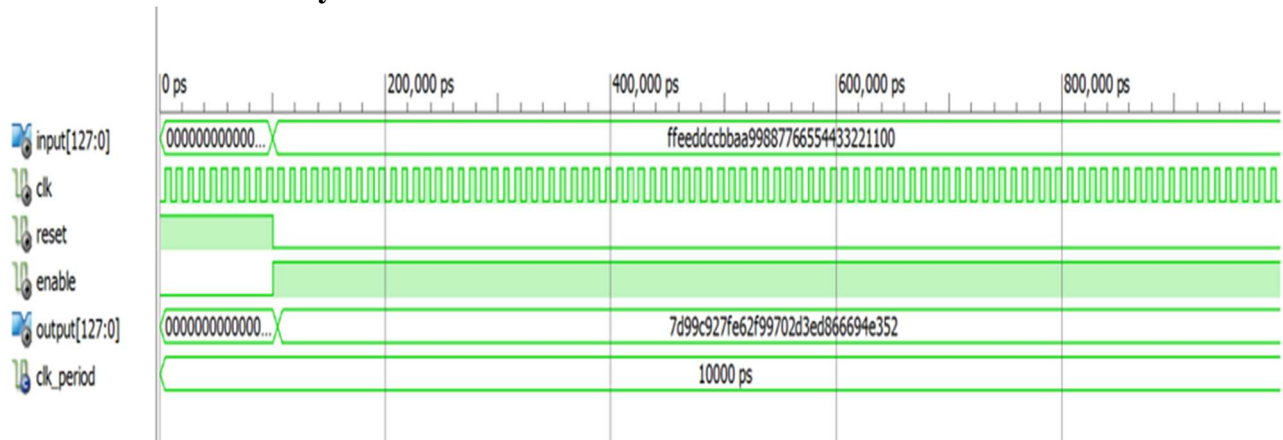


Fig.5.2.5: inv. byte substitution waveform.

The above figure 5.2.5 signifies the waveforms produced by the Inverse byte substitution transformation. The input clock is of 10ns time period, Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.1. It is to be noted that this block reversed the effect of the byte substitution transformation.

### 5.2.6. Inverse Row Shift Transformation



Fig.5.2.6: inv. row shift transformation waveform

The following figure 5.2.6 signifies the waveforms produced by the Inverse row shift transformation. The input clock is of 10ns time period, Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.2. It is to be noted that this block reversed the effect of the row shift transformation.

### 5.2.7. Inverse Mix Column Transformation

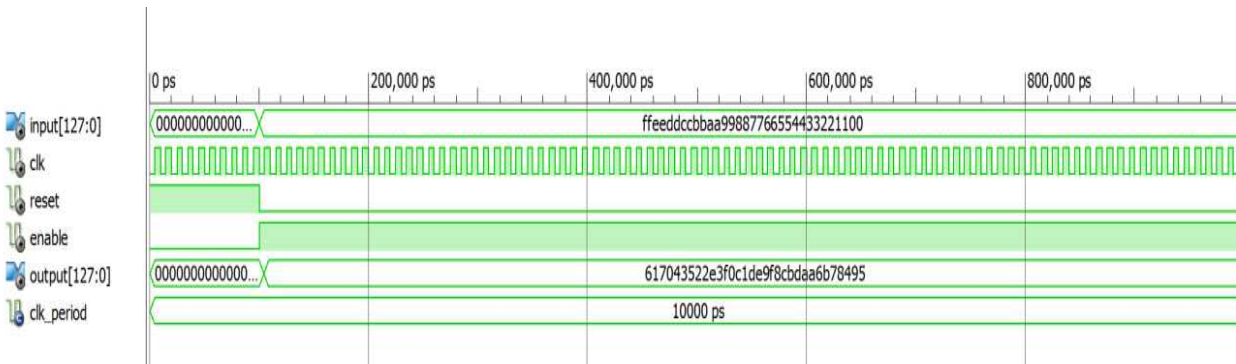


Fig.5.2.7: inv. row shift transformation waveform

The following figure 5.2.7 signifies the waveforms produced by the Inverse mix column transformation. The input clock is of 10ns time period, Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.3. It is to be noted that this block reversed the effect of the mix column transformation.

### 5.2.8. Key Expansion unit

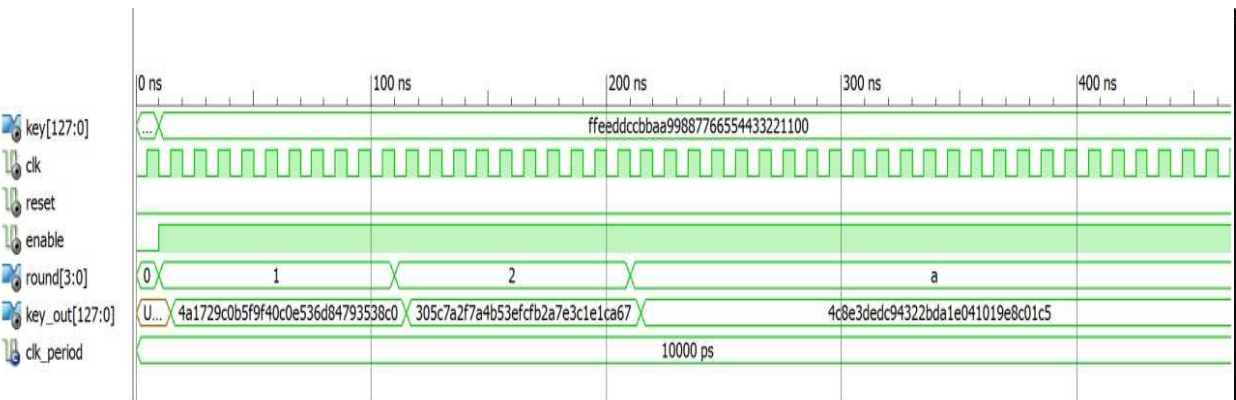


Fig.5.2.8: key expansion waveform

The above figure 15 signifies the waveforms produced by the key expansion unit and its block architecture. The input clock is of 10ns time period, Reset is high, and 128 bits state as a std\_logic\_vector. The output obtained is exactly as described in the segment 3.4.5.





### 5.2.11. AES

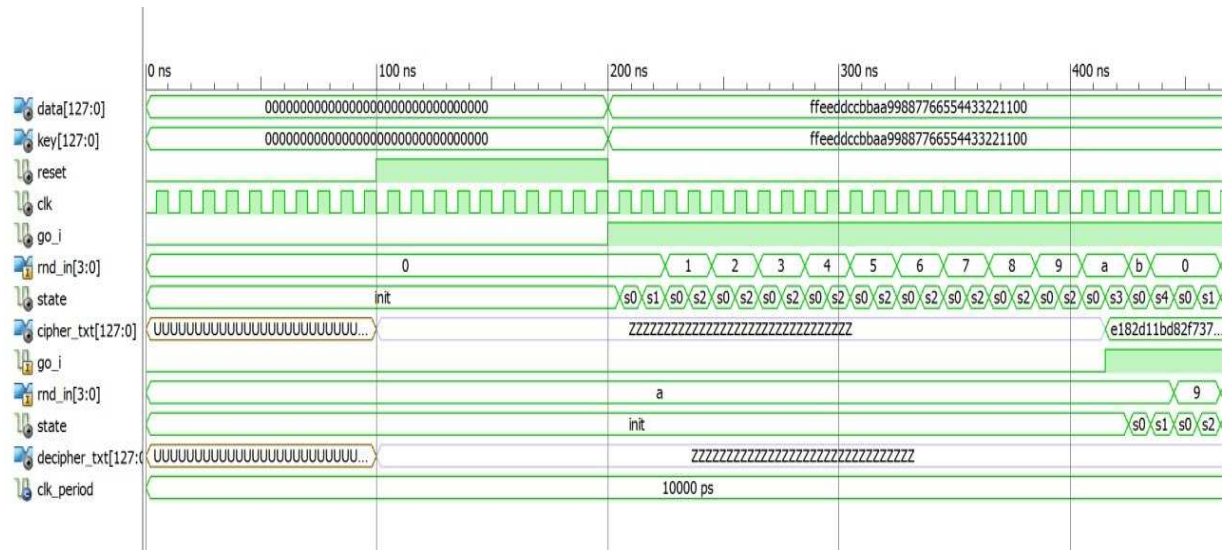


Fig. 5.2.11(A): test bench waveform of Decipher block up to 400ns.

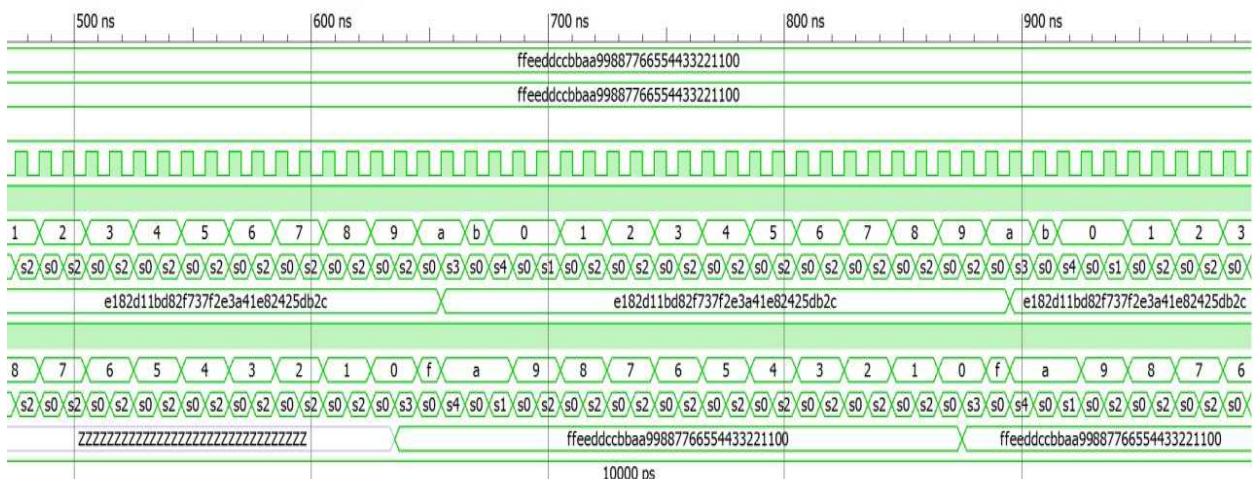


Fig. 5.2.11(B): test bench waveform of Decipher block from 400ns to 900ns period.

Above waveform shows the complete result of AES where data is the input data, key is 128 bit input key, cipher is the ciphered text and decipher is the deciphered text.

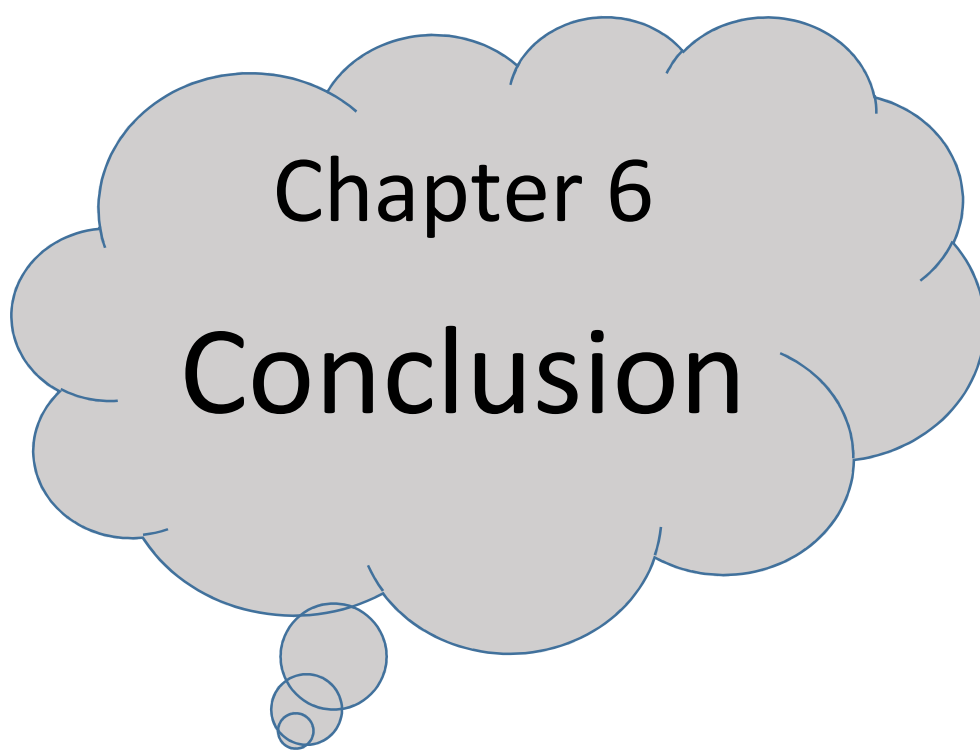
## 5.2.12. DESIGN SUMMARY

| Key_Exp Project Status (05/07/2014 - 16:51:03) |   |                       |             |
|--|---|-----------------------|-------------|
| Project File:                                  | AES_Final.xise                            | Parser Errors:        | No Errors   |
| Module Name:                                   | AES                                       | Implementation State: | Synthesized |
| Target Device:                                 | xc7a100t-3csg324                          | • Errors:             |             |
| Product Version:                               | ISE 14.2                                  | • Warnings:           |             |
| Design Goal:                                   | Balanced                                  | • Routing Results:    |             |
| Design Strategy:                               | <a href="#">Xilinx Default (unlocked)</a> | • Timing Constraints: |             |
| Environment:                                   | <a href="#">System Settings</a>           | • Final Timing Score: |             |

| Device Utilization Summary (estimated values) |       |           |             |  |
|---|-------|-----------|-------------|--|
| Logic Utilization                             | Used  | Available | Utilization |  |
| Number of Slice Registers                     | 2987  | 126800    | 2%          |  |
| Number of Slice LUTs                          | 13812 | 63400     | 21%         |  |
| Number of fully used LUT-FF pairs             | 1579  | 15220     | 10%         |  |
| Number of bonded IOBs                         | 515   | 210       | 245%        |  |
| Number of BUFG/BUFGCTRLs                      | 7     | 32        | 21%         |  |

Fig 5.2.12: Design Summary

The Final design summary of the project is as shown in the above figure 5.2.12. This design summary is done keeping in the view that we are using Spartan 3E XC3s500e FPGA. Though IOBs count is quite high it can be managed by decreasing the input and output parameters like taking data and key as an input one at a time and visualizing the cipher text and decipher text one at a time, this can decrease the IOBs to quite low. Rest of the logic blocks utilization's are quite low, thus we can implement our project in the above stated FPGA board.



Chapter 6

Conclusion

## 6. CONCLUSION

- We have developed an optimized and process able VHDL code for the implementation of both encryption and decryption process.
- The FPGA resource used was drastically decreased from past result which can be seen in the fig 5.2.12.
- Hence, Advanced Encryption Standard architecture designed by us can be executed with rational efficiency on a Spartan 3E XC3s500e FPGA.

## 7. REFERENCE

1. ADVANCED ENCRYPTION STANDARD, Federal Information Processing Standards Publication 197, November 26, 2001.
2. Google Images: [www.images.google.co.in](http://www.images.google.co.in).
3. Wikipedia: [www.wikipedia.org](http://www.wikipedia.org).
4. B.A. Forouzan and D. Mukhopadhyay, Cryptography and Network Security, 2nd Ed.,Tata McGraw Hill, New Delhi, 2012.
5. VHDL Primer (3rd edit ion) by J. Bhasker