BYTE SYSTOLIC FULLY PARALLEL AES ARCHITECTURE

Submitted to the Department of Computer Engineering in fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

by

Hina Raja

2010-NUST-MS PhD- comE-05

MS-65 (CE)

Thesis Supervisor

Dr Shoab Ahmed khan

College of Electrical & Mechanical Engineering

National University of Sciences & Technology

2013

In the name of Allah, the most

Beneficent and the most Merciful

DECLARATION

I hereby declare that I have developed this thesis entirely on the basis of my personal efforts under the sincere guidance of my supervisor Dr. SHOAB AHMED KHAN. All the sources used in this thesis have been cited and the contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

_____ _____

HINA RAJA

ACKNOWLEDGEMENTS

Innumerable words of praise and thanks to Allah, the Almighty, and the Creator of the universe for carving the path for me and always helping me out in the best possible way. Without His Will and Mercy, I would not have been able to accomplish this milestone. I am grateful to my parents especially to my mother, for their immense love, moral support, encouragement and prayers throughout my academic career. I am deeply beholden to my supervisor, Dr. SHOAB AHMED KHAN, for his continuous guidance, inspiration, and patience. His ability of management and foresightedness taught me a lot of things which will be more helpful for me in my practical life. I am highly thankful to Head of Department Dr, SHOAB AHMED KHAN, for his continuous and valuable suggestions and guide, especially for the provision of all kinds of facilities during my thesis work.

Sir Sajid Gul Khawaja (Department of Computer Engineering CEME) helped me throughout the course of implementation and accomplishment of this thesis and I really grateful to him for taking out time out of his busy schedules.

I gratefully acknowledge the help and guidance provided by Examination Committee members. Their valuable suggestions and comments were a great source to improve the research work presented in this thesis. Thank you all.

DEDICATION

To my family members especially parents and teachers.

ABSTRACT

A novel byte systolic fully parallel architecture is proposed for mapping 128-bit AES encryption algorithm. The plain text of 128-bit block is encrypted using the 128-bit key so number of rounds are 10. All the 10 rounds are implemented in parallel by cascading the stages, so the resulting architecture does not reuse the logic resources instead all the computations are in parallel. The proposed architecture works on in-place indexing; a single byte of plain text is input and after the initial latency of 10x16 cycles a byte of cipher text is output in every clock cycle. The novelty of the proposed architecture is more pronounced around in-place indexing. By employing the in-place indexing byte systolic fully parallel architecture best utilizes the memory and works in a lock step manner. The same data memory of each stage is used for next coming frame thus reducing the hardware resources. The technique intelligently removes all the inter and across round dependences by tracing out a single byte so design works in byte systolic fashion. This scheme speeds up the implementation 10x16 times thus increasing the data rate and throughput. The proposed design for AES encryption offers the data rates of 200Mbs while utilizing 2063 slices and 1.6GHz throughput on Xilinx Virtex V xc5v2x5ot. Comparison results clearly show that proposed architecture offers the best tradeoff between area, data rate and throughput.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AES	Advance Encryption standards
CBC	Cipher Block Chaining
CFB	Cipher Feedback
DES	Data Encryption standards
DSA	Digital Signature Algorithm
DSS	Digital Signature standard
ECB	Electronic Codebook
KEA	Key Exchange Algorithm
NBS	National Bureau of Standards
NIST	National Institute for Standards and Technology
OFB	Output Feedback
PGP	Pretty Good Privacy
PKC	Public-Key Cryptography
RSA	Rivest Shamir and Adleman
SKC	Secret-Key Cryptography
TDES	Triple Data Encryption standards

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Overview

The thesis is the mapping of the 128-bit AES algorithm to a byte systolic fully parallel architecture. The AES algorithm is well known encryption scheme that convert the plain text into non readable text using the key. The receiver will decrypt the message into original plain text by using the same key. The design of AES is based upon the principle of Substitution Permutation; it has fast hardware and software implementation. There are number of AES architectures proposed in the literature, selection of a particular architecture depends upon throughput, area, power and data rate of the input. The novel architecture Byte Systolic Fully Parallel Architecture works on byte in-place indexing, the design encrypts the 128-bit of plaintext using key of size of 128-bit. In this technique a byte (8-bit) of plain text is input to the architecture and result in a byte of cipher text as an output in every clock cycles after an initial latency of 16x10 cycles. All the 10 rounds are implemented by cascading all the stages thus resulting architecture does not reuse the logic resources instead executes all the rounds in parallel, thus maximizing the data path size that leads to significantly increase the throughput and data rate.

The hardware descriptive language "VERILOG" is used as a programming language and Xilinx platform is used for the implementation and simulations.

1.2 Background and Motivation

By the invention of internet need for the encryption algorithm emerges, before this encryption was rarely used by the general public mostly used for military purposes. In this modern world it's very uncommon for a person not to have a computer on his desk, work or at home, it's become an essential device now days. The main purpose of a computer is to store, receive and send data between two parties. To communicate with other computer, computers are connected in some fashion (LAN, MAN, WAN etc) [2].While communicating everyone wants to maintain

privacy, so data and communication security have been an important topic. Encryption is a process for providing the privacy to your information [1].

The importance of cryptography applied to security in electronic data transactions has acquired an essential relevance during the last years [3]. Each day millions of users generate and interchange large volumes of information in various fields, such as financial and legal files, medical reports, and bank services via Internet, telephone conversations, and e-commerce transactions. Network security has three major security goals: confidentiality, availability and message integration between senders and receivers. Many algorithms are available in each of these three goals of security. One of the frequently used security algorithm in block cipher is the AES algorithm [7].

AES has variable key length 128-bit, 192-bit and 256-bit but fixed length block of 128-bits. Rijndeal has the key and block multiples of 32 and minimum length is 128-bit, maximum block size is 256-bits and no theoretical maximum value for key length. The design of AES is based upon the principle of Substitution Permutation; it has fast hardware and software implementation.

1.3 Objective

The proposed design Byte Systolic Fully Parallel Architecture implements the 128-bit AES based encryption. Following are the objectives to create architecture that result into high throughput, high data rate and efficient memory utilization.

- To make a design Byte systolic that takes a byte as input; work in lock step fashion and giving out a byte as output. All the operation of AES algorithm; shift row, add round key, byte substitution and mix column will be done on 8-bit data.
- To executes the all rounds of the algorithm in parallel for every upcoming frame.
- To remove the inter and across round dependencies to fully avail the benefit of parallel architecture.
- Implementation on the Xilinx Project Navigator 12.1i suite using Verilog as a programming language.

- Comparison of the result with pervious publications.

1.4 Contributions

A number of AES architectures are proposed in the literature but few of them are mapped to an 8-bit data path. The existing 8-bit AES architecture offers the compact design and best suited for the low area and power applications but the throughput of the 8-bit designs is very low. So the previous 8-bit designs only suited for embedded systems, mobile computing and smart cards. The proposed Byte Systolic Fully Parallel AES architecture offers the highest throughput with a small increase in area compared to other 8-bit designs. So the resulting architecture works for the systems which require the high throughput with low area. The proposed design is also of special interest of the application in which input changes at every clock cycle at faster rate. In addition to this the research conducted has several application areas, image processing, voice communication and multimedia transfer over internet.

1.5 Dissertation Outline

The chapterwise dissertation is given below:

- **Chapter 2: Literature Review**

  Related background topics are discussed in this chapter. The evolution of the encryption algorithms and also discuss the different algorithm for encryption. All the operation of AES i.e. Shift Row, Add Round Key, Byte Substitution and Mix column are explained in detail. This chapter also presents the some of the work by the other researchers in the field of cryptography and especially in AES encryption algorithm.
- **Chapter 3: Design of the Proposed Architecture**

  The complete design of the Byte Systolic Fully Parallel Architecture is discuss in the chapter 3.the comparison of standard AES algorithm with the proposed algorithm is also included in this chapter. The chapter further explains the top level design of architecture; data flow of the algorithm and detail design of Byte Systolic Fully Parallel Architecture.

- **Chapter 4: Implementation of Byte Systolic Architecture**

  The chapter 4 describes how the all stage are implemented using a tool Xilinx. The implementation is divided into two parts the DATA PATH and the CONTROLLER. The chapter 4 is comprises of complete design and implementation of the controller and data path of Byte Systolic Fully Parallel Architecture.

- **Chapter 5: Results**

  This chapter shows the result of simulation on Xilinx ISE 12.1-1, it includes the wave form of input plain text, all the signals of the controller, all rounds and final output cipher text. Comparison of different AES designs with proposed architecture.

- **Chapter 6: Conclusion**

  Last chapter presents the conclusion of the dissertation; it also includes the future work.

CHAPTER 2: ENCRYPTION AND ALGORITHMS

2.1 Introduction

In this modern world it's very uncommon for a person not to have a computer on his desk, work or at home, it's become an essential device these days. The main purpose of a computer is to store, receive and send data between two parties. To communicate with other computer, computers are connected in some fashion (LAN, MAN, WAN etc) [2].While communicating everyone wants to maintain privacy, so data and communication security have been an important topic. Encryption is a process for providing the privacy to your information [1].

To explain why we need to have encryption, let's have an example of computer network or a phone network. Although they both run on the computer but with phone network you have a voice on the other end, this voice will provide the authentication. Both the parties caller and callee know to whom they are talking there would be no misunderstanding of whom you are talking. If you want to make a call to your friend, you know that on the other end it's your friend nobody else, whatever your friend says you will hear after a fraction of delay. It's assured that until and unless you are not using, cellular or cordless phone, making an international call, your conservation is secured. If you are using cellular or cordless phone it would be easy for a third party to listen your conservation using radio equipments [2]. Government routinely monitored international calls, in making an international call it may possible some law enforcement agency has interest in you or in your friend and agency have order from the court to tap your conservation.

On the other hand sending a message to your friend using the computer network, it may possible your message being changed by someone during the journey before it actually delivered to your friend. Let's see what happens when you sent a message to your friend, first the software (of sending and receiving message) will find your friend on the same computer, if it succeed in finding your friend then it will deliver the message. Otherwise next step would be to check that message can be sent directly to your friend or not, if yes message will be delivered. If software can't find the direct path then it will deliver to computer that can talk to friend's computer. Now

your message will be stored on disk of each computer, it seems too appeared as you are directly connected to your friend but actually it's not true. Your message passes through several computers thus making the network. Now you are on people mercy, your message can be changed before it actually delivers to your friend. Encryption provides the ways to secure your information, so that the intended recipient can get the information [1]. Encryption algorithm is being used to provide privacy and confidentiality for email communication named as Pretty Good Privacy (PGP) [9]. It involves a serial combination of public-key cryptography, symmetric-key cryptography, and hashing and data compression. The combination of symmetric-key and public-key provides the confidentiality to your message. A symmetric encryption algorithm requires the symmetric key to encrypt the message and each key is used only once in a session, it is also called session key. The session key is being encrypted by receiver public key for ensuring that only intended recipient get the session key. The encrypted session key and message is sent to receiver. PGP also supports integrity and authentication. Encryption and decryption algorithm are shown in Figure 2.1.

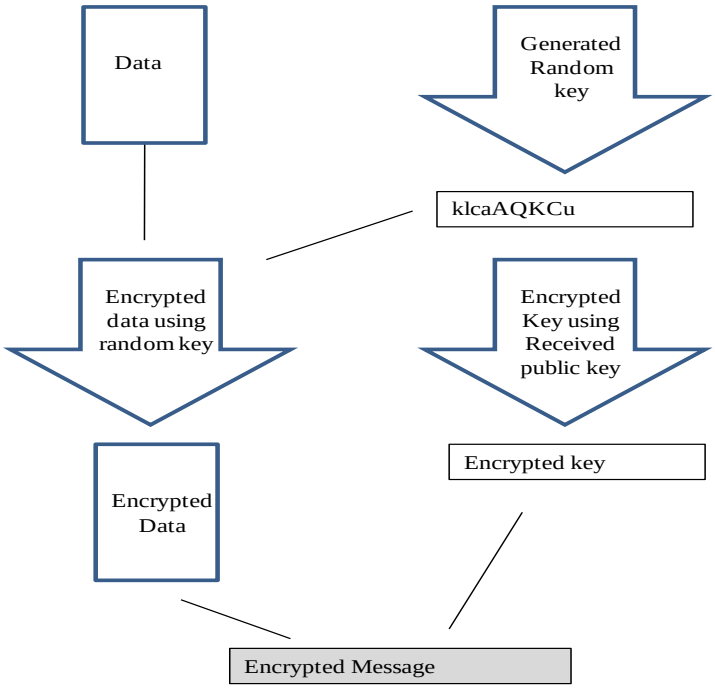


Figure 2.1: PGP Algorithm

Most of our daily activities are being done through internet e.g. bank transaction, credit card transaction, and secret information exchange. So encryption is a necessary step for the successes

of commerce over the internet. Without authentication of whom you are communicating people are unwilling to transact any kind of business because transacts are not secure [3]. Encryption can provide the confidence to the people so that they can transact over the internet. The Figure 2.2 and 2.3 shows the two different scenarios with and without encryption.

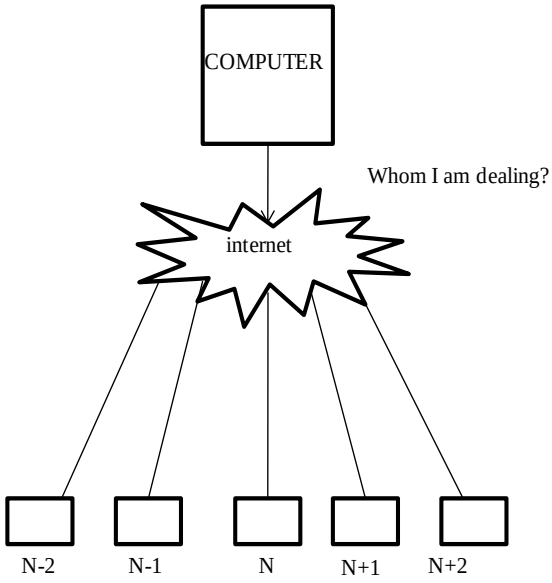


Figure 2.2: No encryption algorithm no authentication to whom one is talking.

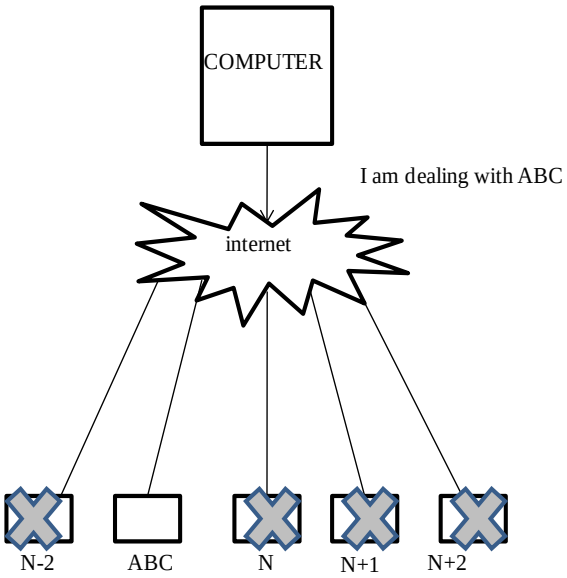


Figure 2.3: Encryption provides integrity and authentication.

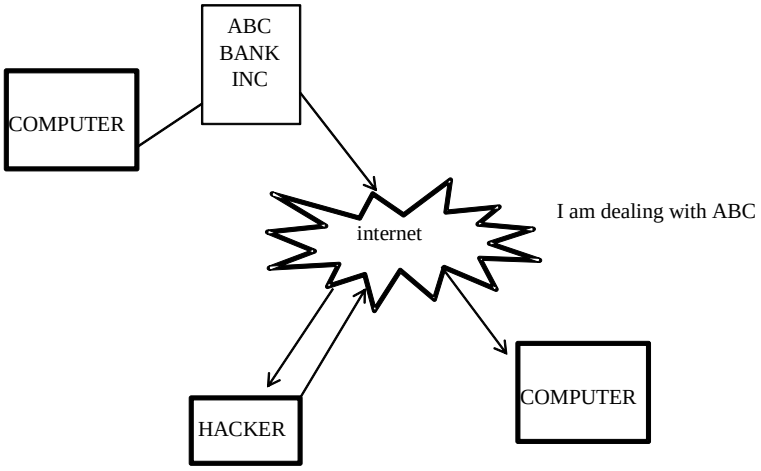


Figure 2.4: How transaction over the internet being attacked

The Figure 2.4 shows the transaction over the internet is being attacked. So need of encryption has found in many application of life starting from electronic financial transactions, data storage, smart card, wireless communication and cellular phone [8]. Different encryption algorithm have been used since last two decades, RSA, DSA, DES, 3DES, AES etc. the latest algorithm outperforming the previous one.

2.2 Cryptography

Cryptography is defined as "the science and study of techniques for providing secure communication in the presence of third party" [11]. Cryptography is science and study of secure communication and encryption is a component of that science, it use the mathematics to encrypt and decrypt [29]. More generally cryptography is concern about analyzing and construction the protocol that overcomes the influence of third party thus providing the authentication, confidentiality, data integrity and non-repudiation.

Authentication: It provides the identity to user.

Data integrity: Ensuring the receiver that original message has not been changed by any intruder in the communication path.

Non-repudiation: This is a process of proving to receiver that message is really sent by the sender.

Confidentiality: Ensuring that only the intended receiver can read the message.

Prior to modern age cryptography is synonymous with encryption; the conversion of readable message into unreadable state, in this modern age cryptography is based on computer science and mathematical theory. Cryptography algorithms are based on computationally hard assumption thus making impossible for intruder to break it. It's theoretical possible to break the algorithm but it's impossible to break it by any practical means, so algorithm are termed as computationally secure. These algorithms are difficult to implement than theoretical breakable schemes.

2.3 Types of Cryptography

Cryptography not just only provides confidentiality but also integrity and authentication, to achieve this goal there are three schemes of cryptography which are as follow:

- Symmetric Cryptography
- Asymmetric Cryptography
- Hashing Function

2.3.1 Asymmetric Cryptography

Asymmetric cryptography also called public-key cryptography (PKC). Public-key cryptography uses the asymmetric algorithm, where key use to encrypt the message will not be used to decrypt the message [9]. There is a pair of cryptography key; one is public key other one is private key. The public key is known by everyone but private key is secret only intended user will have that. The message is encrypted using public key of recipient and it's only decrypted using corresponding private key (only recipient will have that). Both the keys are related mathematically, but it's not feasibly to derive private key from public key [13]. Figure 2.5 shows the asymmetric cryptography.

Many cryptosystem and cryptographic algorithm based on public-key cryptographic approach; there is no need of initial exchange of one or more secret key unlike symmetric-key cryptography.

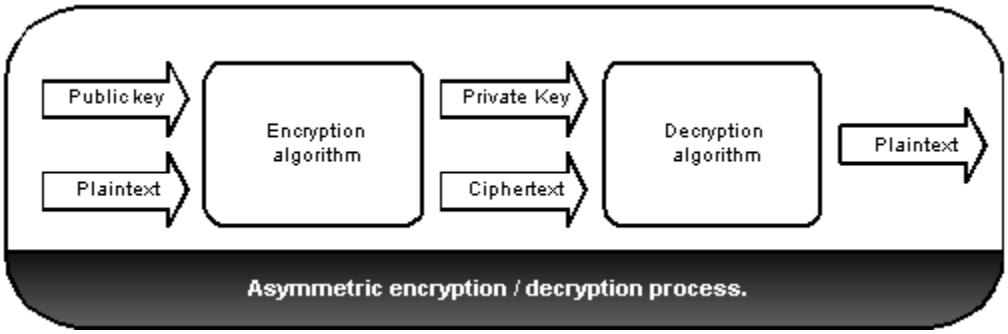


Figure2.5: Asymmetric Cryptography [14]

It is also used to create digital signature. Public-key cryptographic algorithm widely used to create digital signature and key exchange.

- RSA
- Diffie-Hellman
- DSA
- ElGamal
- Cramer-Shoup
- KEA

2.3.1.1 RSA

It is the first and mostly used algorithm for signing as well as encryption.RSA is named after three mathematicians Ronald Rivest, Adi Shamir and Leonard Adleman [13].RSA is widely used in 100 of software products especially electronic commerce protocols.RSA uses a variable size key and encryption block. The key is derived from "n" that is very large number, according to special rules these two prime numbers are selected. The product of these two prime numbers gives rise to "n" very large number, if each of prime numbers is of 100 digits then "n" will be of 200 digits. The public key includes the very large number "n" and derivates of one of the prime number, so attacker is unable to determine the one of factor of the "n" is in public- key. With very high computing power it's difficult to find out the prime factors of "n", according to test

held in 2005 to find the prime factor of 200 digit number took one and half year aver the fifty years of computing time. Thus by making the size of key large RSA becomes more secure [16].

2.3.1.2 Diffie-Hellman

After RSA algorithm Whitfield Diffie and Martin Hellman published their own algorithm in 1976. It was later emerged that this algorithm had been invented by Malcolm J. Williamson with GCHQ (the British intelligence agency).D-H is used for only secret key exchange not for digital signature and authentication.

In 2002 Hellman and Ralph Merkle suggested the algorithm called Diffie- Hellman- Merkle key exchange. Diffie-Hellman is a non authenticated key agreement protocol but it provides basis for the variety of authenticated protocols. It provides the perfect forward security in Transport Layer Security modes.

2.3.1.3 Digital Signature Algorithm

Digital signature algorithm is a standard used for digital signature only; it's not an encryption algorithm [19].DSA specified in NIST as Digital Signature standard (DDS).

2.3.1.4 ElGamal

It was designed by Taher Elgamal in 1985, it is an asymmetric cryptography based on Diffie-Hellman. ElGamal is used in the free Privacy Guard Software a recent version of Pretty Good Protocol PGP. ElGamal is used to establish the common keys similar to Diffie-Hellman.

2.3.1.5 Cramer shoup

It is developed by Victor shoup and Ronald Cramer in 1998 and it is an extension of Elgamal algorithm. Elgamal is extremely malleable but cramer shoup adds the additional features to ensure the non-malleability [12]. It is an asymmetric cryptography; it was the first scheme that is proven to be more secure against the adaptive cipher text attacks by employing the standard cryptographic assumptions. The Cramer shoup is based on the computational intractability of the Diffie hellman assumption. The non-malleability is achieved by using the collision resistant hash function, additional computation and as a result the cipher text formed is twice larger than Elgamal.

2.3.1.6 Key Exchange Algorithm

Key exchange algorithm is an asymmetric cryptography; it is a variation of Diffie Hellman and proposed as the exchange of key for Capstone [12].

2.3.2 Symmetric Cryptography

In symmetric cryptography a single key is used for encryption and decryption. The key used for encryption at sender the same key used for decryption at the receiver [1]. So sender and receiver must have the key and it is the most difficult problem of symmetric cryptography to distribute the key.

Stream cipher and block cipher are general categories of symmetric cryptography. The Stream cipher works on a single bit, byte or on a word, it implements the feedback algorithms so key is changing constantly [28]. The block ciphers works on one block of data at a time and use the same key on each of the block. In general, the block cipher result in the same cipher text if same key is used for the encryption of plaintext block whereas in stream cipher same data result into different ciphers text [26].

The most common modes of the block cipher are as follows:

Electronic Code Book (ECB): It is the simplest and common mode of the block cipher; it encrypts the plaintext block using the secret key into cipher text and vulnerable to brute-force attacks. The same plaintext block encrypts into same cipher text [39].

Cipher Block Chaining (CBC): It adds the feedback feature to the encryption, before encryption CBC XOR the plaintext with the previous block of cipher text ,the same plaintext block never encrypts into same cipher text.

Cipher Feed Back (CFB): It encrypts the data into smaller units than the block size which is useful for the encryption of interactive terminal input. If one byte CFB is used so every upcoming character is placed into the register when block size is achieved then block is transmitted and at receiver block is decrypted and extra bits are discarded, this mode is similar to self synchronization stream cipher.

Output Feed Back (OPF): It is similar to synchronous stream cipher; OPF uses the internal feedback mechanism that is independent of plaintext and cipher text streams so it prevents the generation of same cipher text from the same plaintext [12].

Symmetric cryptographic algorithms used today are as follows:

- Data Encryption Standards (DES)
- Advance Encryption Standards (AES)
- Blowfish
- Twofish
- Camellia
- SEED
- ARIA
- CLEFIA

2.3.2.1 Data Encryption Standard

It the most common algorithm used today, it was designed in 1970 by IBM and adopted by NBS for unclassified government and commercial applications.DES is based upon block cipher, it operates on 64-bit and use 56-bit key having complex set of rules and transformations that result in slow software and fast hardware implementation [11]. Now a day's DES is insecure for many applications because of 56-bit key size which is very small. In 1999 DES is break by Electronic Frontier Foundation and distributed.net in 22 hours and 15 minutes although it is unfeasible to mount. FIPS 46-3 developed the triple DES and it can be used until 2030 as approved by FIPS as encryption algorithm. TDES employs the 3 rounds of the DES and having the key length of 168-bits; brute force attacks are currently not possible.

2.3.2.2 Advance Encryption Standard

It was announced in November 26, 2001 and based on Rijndeal. AES is a block cipher and can use the variable lengths of key and block, latest version allow any combination of block length 128 and key length (128,192,256). The AES describe in detail in section 2.4.

2.3.2.3 Blowfish

It is invented by Bruce Schneier, works on 64-bit cipher block and key length can vary from 32-bit to 448-bits.Blowfish is especially designed for 32-bit architecture with large data caches and faster than DES on Pentium or power PC [30].

2.3.2.4 Twofish

It is designed by a team supervised by Bruce Schneier which is highly flexible, highly secures, well suited for 8-bit smart microprocessors and dedicated hardware. Twofish works on 128-bit block with the key of length 128-bit, 192-bit and 256-bit.

2.3.2.5 Camellia

It was developed in 2000 by joint collaboration of Mitsubishi Electric Corporation (MEC) and Nippon Telegraph and Telephone (NTT). Some of the features of Camellia is similar to AES that is 128-bit block with key of variable length 128-bit, 192-bit, 256-bit; suitable for both hardware and software implementations. Camellia is also well suited for 32-bit as well as 8-bit microprocessors.

2.3.2.6 SEED

It is invented by Korea Information Security Agency (KISA) and in South Korea it is adopted as a national standard. SEED is block cipher it works on 128-bit block with 128-bit key.

2.3.3 Hashing Function

Hashing function can also called one way encryption and message digests. A hash value is fixed length which is computed from plaintext, hash value has no key so plaintext can never recovered. Hash functions are usually used for the generation of digital fingerprint of the content of the file and ensure the integrity of the file, that no third party had altered the contents. Many operating systems use the hashing algorithm for the encryption of passwords.

It is misunderstood that two file will not have the same hash value, it is wrong two file can have the same hash value. Hash function which generates the 128-bit hash value so possible hash values 2^{128} are but infinite number of possible files $\infty >> 2^{128}$can have the same hash value. To

find the two files having the same hash value is very difficult for this reason extensively used for the information security and computer forensics applications.

2.4 Advance Encryption Algorithm

It was announced by NIST in November 26, 2001 and based on Rijndeal after standardization of five years in which 15 competing algorithm were presented and before this Rijndeal was considered to be the best algorithm. In cryptography AES is approved as a standard by Federal Information Processing Standard Publications FIPS and adopted by US government [22].

AES has variable key length 128-bit, 192-bit and 256-bit but fixed length block of 128-bits. Rijndeal has the key and block multiples of 32 and minimum length is 128-bit, maximum block size is 256-bits and no theoretical maximum value for key length. The design of AES is based upon the principle of Substitution Permutation; it has fast hardware and software implementation and unlike DES it does not apply Feistel network.

AES operates on a state matrix of 16 bytes (4 x 4) and most of the operations are in finite filed. A number of transformations on a state matrix convert the plaintext into cipher text and a set reverse transformation on cipher text result into original plaintext [14]. The Figure 2.6 shows the flow of the AES encryption algorithm. The key length of AES will determine the number of rounds; for different key length numbers of rounds are as follows:

- 128-bit 10 rounds
- 192-bit 12 rounds
- 256-bit 14 rounds.

2.4.1 AddRound Key

As all the operations are on state matrix, addround key is bitwise XOR of state matrix with round key.

2.4.2 Shift Row

It is the cyclic rotation of rows to the left by a certain offset. For AES shift row is simple, first row is unchanged, second row is shifted by 1 to the left, third row is shifted by 2 and third row is

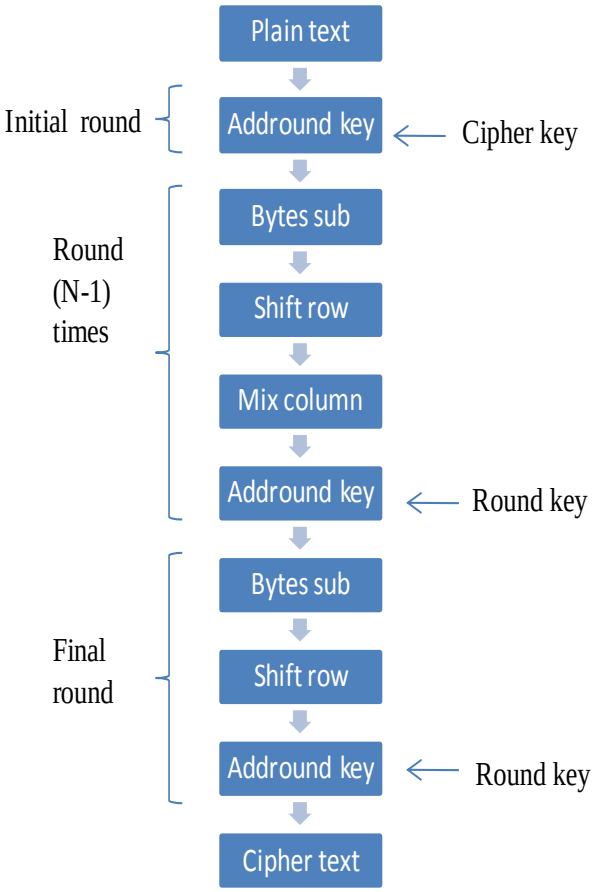


Figure 2.6: Flow diagram of AES Encryption

shifted by 3.unlike AES, Rijndeal have variable block lengths so row shit operation is same as of AES for 128-bit and 192-bit block. For 256-bit block first row is unchanged and second, third and fourth row offset are 1, 3 and 4 respectively. Figure 2.7 shows the row shift operation.

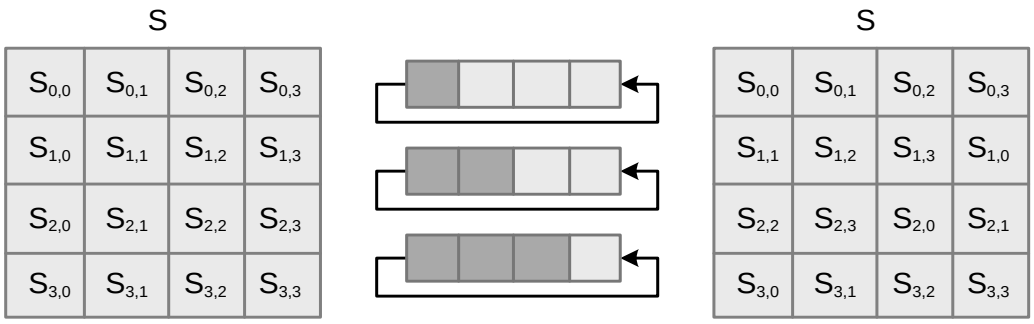


Figure 2.7: Shift Row operation for 128-bit block

2.4.3 Byte Substitution

Thus is the non linear transformation, each byte of state matrix is replaced by a byte from lookup table 16 x16 called S-BOX. This S-BOX provides the nonlinearity to the algorithm and it is derived from multiplicative inverse over the finite field GF (2^8) and combines with invertible affine transformation [24]. The property of S-BOX avoids the attacks which are simply based upon algebraic computations and also handle the fixed and opposite fixed points. The S_BOX is used for the encryption and inverse S_BOX is used for decryption, both of them are shown in Figure 2.8 and 2.9 respectively.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure2.7: Lookup table 16x16 of S-BOX

2.4.2 Mix Column

It is the most complicated operation of AES algorithm, all the operation is in finite field GF (2^8). Before explaining the mix column operation in detail first have look on finite field operations.

2.4.2.1 Finite Field

Finite field arithmetic's is different from the normal integer arithmetic's; in finite field there are limited numbers of elements and result after the operation of finite field also lies in that field. The elements are of form p^x; where x is positive integer called the *dimension* of the field and p is a prime number called the *characteristics* of the field [14]. Finite field itself is not infinite but

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure2.8: Lookup table 16x16 of inverse S-BOX

there are infinite numbers of finite fields, two finite fields with same number of elements are called *isomorphic*.

Finite field is used in many applications; including linear block codes, BCH codes, Reed Solomon error correction, classic coding theory and cryptography.

Each element in GF (2^8) represents a polynomial of degree 7 having coefficient in GF (2^8), there are 8 terms in the polynomial and coefficient can have any value 0 or 1[15].

Addition in Finite Field:

Addition and subtraction of two elements in finite field is simple bit wise exclusive OR of two elements.

1^{st} Polynomial: $x^6+ x^4+x +1$

2^{nd} Polynomial: $x^7+x^6+ x^3+x$

$(x^6+ x^4+x +1) + (x^7+x^6+ x^3+x) = x^7+x^4+ x^3+x$

01010011 + 11001010 = 1001001

{53} + {CA} = {99}

Multiplication in Finite Field:

In finite field multiplication is XOR and AND operation. Following is the algorithm for finite field multiplication.

Run the following loop 8 times. It is ok to stop when a or b are zero an iteration.

1. If right most bit of b is set, exclusive OR the product p by the value of a. this is polynomial addition
2. Shift b one bit to the right, discarding the right most bit and making the left most bit zero. This divides the polynomial by x. discarding the x^0 term.
3. Keep the track of whether the left most bit of "a" is set to one and call this value carry.
4. Shift "a" 1 bit to the left, discarding the leftmost bit of "a" and making rightmost bit zero. This multiplies the polynomial by x, the coefficient of x^7.
5. If carry had a value of one, exclusive OR "a" with the hexadecimal number 0x1b, 0x1b corresponds to irreducible polynomial with the high term eliminated. Conceptually the high term of the irreducible polynomial and carry add modulo 2 to 0.

2.4.2.2 Mixcolumn in AES

In AES mix column operation is multiplication of state matrix 4 x4 with fixed predefined matrix, multiplication is done in such a way that the mixcolumn takes the input a column ($a_0\ a_1\ a_2\ a_3$) of state matrix and multiple it with the matrix P and results into a column vector B ($b_0\ b_1\ b_2\ b_3$) which is replaced with input column in a state matrix. Each column of a state matrix is multiplied with a fixed polynomial described below and represented by matrix.

$$\mathbf{F(x) = 3x^3 + x^2 + x + 2} \qquad (2.1)$$

The inverse polynomial is as follows:

$$\mathbf{F^{-1}(x) = 11x^3 + 13x^2 + 9x + 14} \qquad (2.2)$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{02} & \mathbf{03} & \mathbf{01} & \mathbf{01} \\ \mathbf{01} & \mathbf{02} & \mathbf{03} & \mathbf{01} \\ \mathbf{01} & \mathbf{01} & \mathbf{02} & \mathbf{03} \\ \mathbf{03} & \mathbf{01} & \mathbf{01} & \mathbf{02} \end{bmatrix}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

B P Ist column of state matrix

The partial products are as follow:

$$\mathbf{b_0} = 2a_0 \oplus 3a_1 \oplus 1a_2 \oplus 1a_3 \quad (2.2)$$

$$\mathbf{b_1} = 1a_0 \oplus 2a_1 \oplus 3a_2 \oplus 1a_3 \quad (2.3)$$

$$\mathbf{b_2} = 1a_0 \oplus 1a_1 \oplus 2a_2 \oplus 3a_3 \quad (2.4)$$

$$\mathbf{b_3} = 3a_0 \oplus 1a_1 \oplus 1a_2 \oplus 2a_3 \quad (2.5)$$

In this way one by one column of state matrix is multiplied with matrix P and the result of multiplication will replace the input column in the state matrix as shown in Figure 2.9.

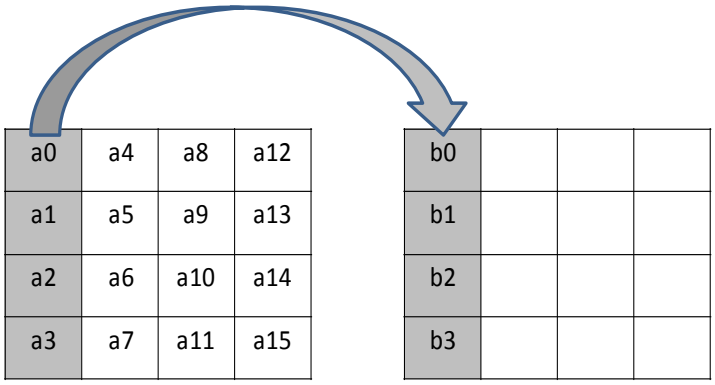


Figure2.9: Replacement of a column of state matrix with the result of Mixcolumn

2.5 Related Work

This section presents the some of the work by the other researchers in the field of cryptography and especially related to AES encryption. There are number of architectures proposed in the literature, selection of a particular architecture depends upon throughput, area, power and data of the input. The intensively pipelined architecture has been used for achieving high throughput,

on-the-fly key generation for avoiding the storage key memory, optimized Mixcolumn and efficient S-BOX implementations for the applications with area constrain. Recent research work for AES implementation has focused on compact AES architecture to target the applications in consumer electronics with low cost and low power.

In paper [41], minimizes the data path size that leads to significant reduction in hardware resources. The in 8-bit time shared architecture recourses are shared, different operations are performed using the same resources at different time intervals. The key expansion scheduling algorithm use the same logic resources thus reported architecture offer the data rate in the range of 41 Mbs while utilizing the 236 slices. This design is viable for wireless communication where high throughput is not required.

In reference to paper [31], AES is modified to reduce the computational overhead. In the modified AES mixcolumn is replaced by the permutation step which is taken from the Data Encryption Algorithm. Modified AES is a fast lightweight algorithm which ensures the security to multimedia applications.

The standard AES encryption algorithm is slow and not suitable for the application which requires the high speed. As the need of high performance computation increased, the Graphic - Processing Unit (GPU) is hot topic of research. The paper [26] presents the AES algorithm is based upon the high computation performance of GPU. The AES encryption based on GPU is higher than on CPU, because of its high data transfer and better parallelism.

The image encryption algorithm alone is not efficient to provide image integrity; AES algorithm is used with them to provide the integrity to images. The paper [35] presents the low power and high speed encryption technique for images. To increase the throughput and speed, pipelining of 4 stages with the optimization of mixcolumn is implemented. The design offer the data rate in range 100MHz and maximum frequency is 475MHz on Altera Company of FPGA and power is achieved as 301Mw using Xilinx Xpower Analyzer The limitation of this architecture is it’s not fully parallel so throughput is not drastically increased.

An area optimized AES encryptor on a reconfigurable device is presented in [21]. In this design, the S-box implementation is achieved by mapping the S-box table using composite field

arithmetic. The S-box is implemented using combinational logic for area optimization. The design is capable of processing data with all supported key lengths. It lacks efficient utilization of FPGA hardware resources as the reported gate count to implement this design is more than 21K on Virtex xcv1000 device from Xilinx.

A reduced data path AES design is reported in [22]. The 8-bit AES architecture uses embedded memory devices available in the FPGA to store round keys and S-box table entries. The use of memory devices replaces the hardware required to implement key schedule and S-box as the round keys and S-box entries are stored inside FPGA memories. A circuitry is used to fetch the stored keys from the memory and distribute them during encryption and decryption processes. The use of embedded memories makes the design memory intensive. The design is implemented on Flex 10k20 FPGA from Altera and utilizes 957 logic elements.

A 64-bit data path version of the AES architecture with internal pipelining is presented in [23] by Mónica Liberatori. The architecture uses 8 S-box implementations. It uses on-the-fly key generation and does not require any memory to store round keys. The 64-bit internal data path is converted into 128-bit data path in the mix column stage to obtain results in a single cycle. The number of logic resources reported for this architecture is 822 slices for the encryption module implementation when mapped on Spartan 3 xc3x200 device from Xilinx.

Two architectures of AES are presented in [7] implemented as sequential and pipelined designs. A memory less solutions is obtained in the sequential design of the AES algorithm. This makes the design highly portable as it can be virtually mapped on any FPGA device with no dependency of the FPGA embedded resources. Since the design is sequential in nature, the number of logic resources required to implement this solution is 2744 configurable logic block (CLB) slices. The design offers a sustained throughput of 258.5 Mbps. The pipelined version of the same design utilizes 2136 CLB slices offering throughput in the range of 2.868 Gbps. The latter uses 100 Block Select RAMs (BRAMs) available as embedded FPGA memories which make it a memory intensive design. The sequential and the pipelined versions are implemented on Virtex E family of the Xilinx FPGA devices (xcv812).

AES designs reported by Chodwiec [45] and Rouvroy [24] are amongst area efficient FPGA designs to the best of author's knowledge. The design presented in [45] uses 222 slices of the

Spartan 2 xc2s30-6 FPGA device from Xilinx whereas the design in [24] utilizes 163 slices on Spartan 3 xc3s50-4 FPGA device.

The design presented in [19] is yet another compact and interesting AES architecture. The design is implemented as an Application Specific Instruction Processor (ASIP) with the capability of performing encryption and decryption processes through the same module. The architecture is based on 8-bit data path and uses only 128-bit cipher key length. The design obtains area optimization by implementing S-box using existing composite field arithmetic. It uses considerably low amount of FPGA resources when mapped on one of the smallest FPGA device from Xilinx (Spartan 2 xc2s15). The number of reported logic resources utilized by this design is 262 slices.

The design presented in [18] is author's initial research work which transforms the 32-bit AES algorithm to 8-bit through architectural transformation. The work presents an efficient hardware implementation for the encryption process. A high degree of hardware re-usability results in a complex controller in this design. However, the amount of logic resources utilized to implement this architecture is still low and amounts to 337 slices when mapped on a Virtex 2 xc2v1000-6 FPGA device from Xilinx. The design uses two embedded memory blocks of the FPGA.

The paper [38] pipeline architecture of AES algorithm using key generation which is based on search based memory. The pipelining is introduce to increase the throughput and search based S-BOX technique is used to reduce the hardware constrains. The comparison results presented shows the slices utilization 402, 2700 and 3898 of iterative loop 128-bit AES, pipeline and pipeline with search based engine architecture respectively.

CHAPTER 3: Design of a Proposed Architecture

3.1 Overview

The proposed design Byte Systolic Fully Parallel Architecture implements the 128-bit AES based encryption using in-place indexing, parallel computation and by pass techniques to create architecture that result into high throughput, high data rate and efficient memory utilization.

The novel architecture Byte Systolic Fully Parallel Architecture works on byte in-place indexing, the design encrypts the 128-bit of plaintext using key of size of 128-bit. In this technique a byte (8-bit) of plain text is input to the architecture and result in a byte of cipher text as an output in every clock cycles after an initial latency of 16x10 cycles. All the 10 rounds are implemented by cascading all the stages thus resulting architecture does not reuse the logic resources instead executes all the rounds in parallel, thus maximizing the data path size that leads to significantly increase the throughput and data rate.

The standard AES works on 32-bit word so hardware implementation requires 32-bit data path, register, buses and memory. Though some of 8 bit architectures are proposed in the literature but they require accumulation of four 8-bit words for computation of mix column operation [6]. As proposed architecture is a byte systolic it works on a single byte thus reducing the width of data path and other hardware resources thus result in minimizing the area and dissipate less power. The proposed architecture is also fully parallel that is all the rounds are implemented in parallel. The parallel architectures proposed in literature have limitations that inter round dependences produces some extra latency, the novelty of proposed architecture is more pronounced around in-place indexing which removes the inter round and within round dependences . The 16 cycles are required to write 16 bytes of a data after writing the data round can starts its execution. As single memory 8x16 bits is used for each round that is the upcoming next frame uses the same memory. Now for the next frame to be written also requires 16 cycles, but our proposed architecture works on in-place indexing so the technique writes the input of next frame at the location that is already used in the current cycle. As one round is in process the input data for the next frame getting into its location, on completion of one round data of next frame is also ready so more latency of 16 cycles. This schemes speed up the design 10x16 times thus increasing the data rate and

throughput. The removal of all the dependencies makes the proposed architecture to execute in lock step and implements all the rounds in parallel without creating some extra latency. The proposed architecture offers the best tradeoff between area, power, data rate and throughput for AES hardware implementation. Figure 3.1 shows the comparison of standard AES algorithm and proposed architecture.

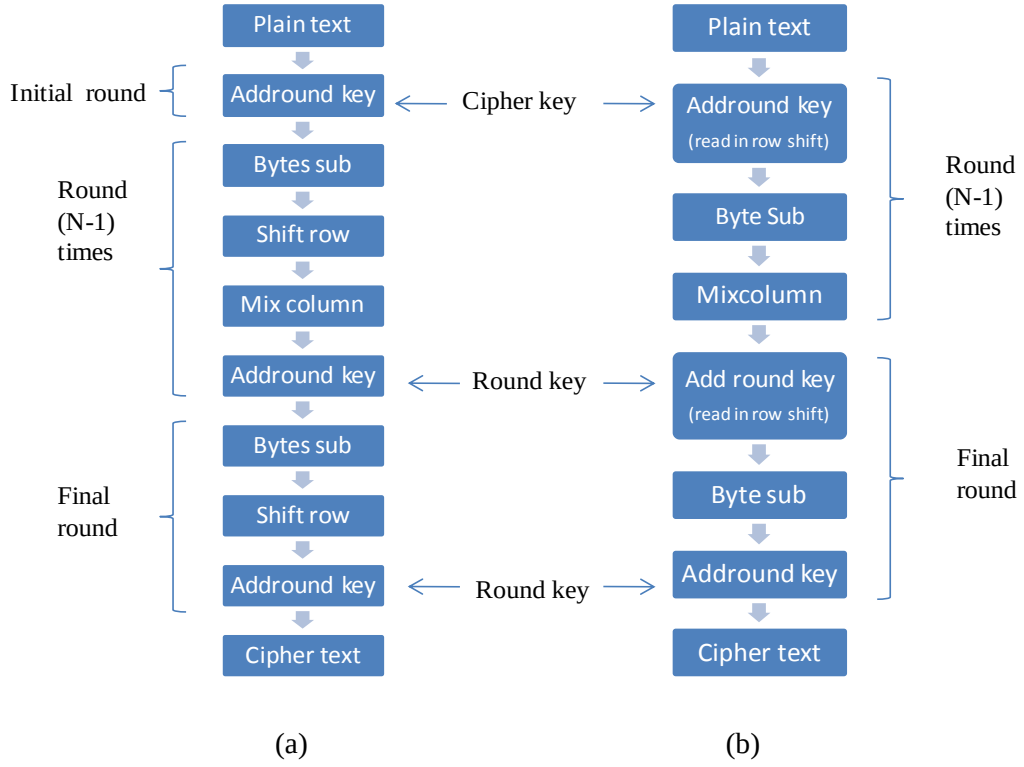


Figure 3.1: Comparison of standard AES algorithm and proposed architecture. (a) Standard AES. (b) Proposed AES Algorithm.

3.2 Proposed Architecture

3.2.1 Byte Systolic Fully Parallel Architecture

The proposed architecture mapped the AES algorithm to Byte Systolic Fully Parallel Architecture that encrypts the 128-bit block of plaintext using the key of size 128-bit so number of rounds for each data block is 10. The proposed architecture works on byte in-place indexing; a single byte of a plaintext is input and after an initial latency of 16x10 cycles a byte of cipher text is output in every clock cycle. Top level design of architecture is shown in Figure 3.2.

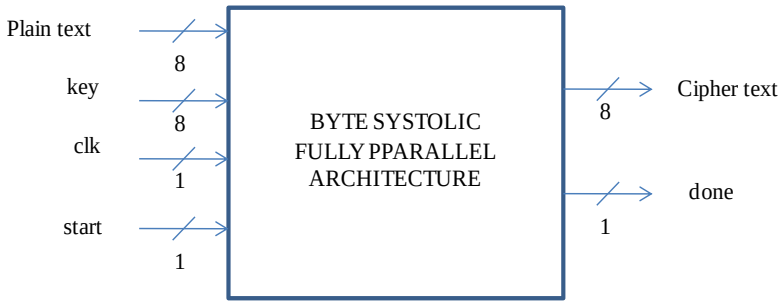


Figure 3.2: Top Level Design

3.2.1.1 Pipelining of Architecture

The pipelining is way to process the data in a continuous manner without waiting for the completion of current process, the concept of pipelining is seen in many processors [44]. The pipelined architecture is shown in the Figure 3.3. Each stage is separated by register so the output of the one stage is saved and can pass to next stage when it is required without any latency. In iterative loop architecture next round can't start its execution until and unless current round is in process because same hardware is used again for the next round. Thus, the pipelined architecture drastically increased the speed and throughput.

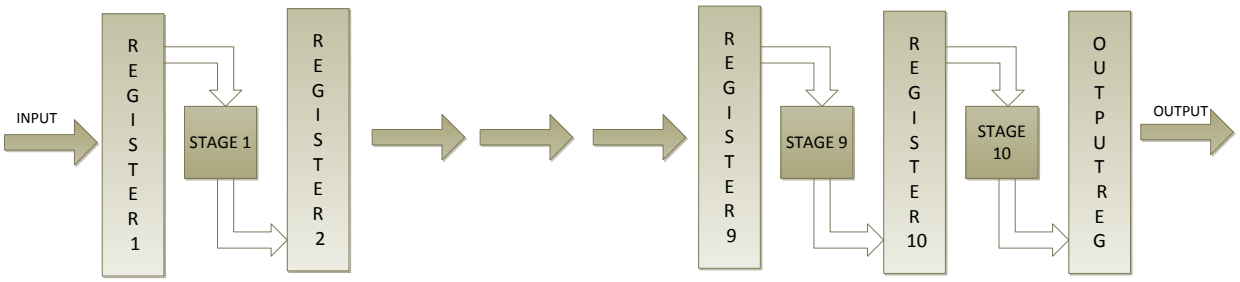


Figure 3.3: Pipelined Architecture.

The design of a byte systolic fully parallel architecture is shown in Figure 3.4. All the rounds are implemented in parallel and this is achieved by cascading all the stages. Every stage has its own data and key memory only last stage has two key memories, the text which has to be encrypted is stored in data memory. In key memory corresponding key is stored in each stage which is expanded offline by key scheduling algorithm. For key size of 128-bit there are 10 rounds keys;

for stage 1 cipher key; for stage 2 round key (R1); for stage 3 round key (R2) and so on but in stage 10 round keys R9 and R10 are saved in key memory. The data is input in byte serial fashion to stage 1, execution of the round starts when the 16 bytes have been written into the data memory. As the round 1 for 1st frame is in process at the same time input for the second frame is written into the data memory of the stage 1 by applying in-place index addressing which is explained in next section. In-place index addressing scheme writes the input at the location from where the data is read and currently used in the cycle. The addressing for writing and read from memory is done by using the address generation unit that is with each data memory and key memory. As the 1st frame completes the execution of the round1 data for the 2nd frame is stored in data memory of stage one, now 1st frame starts the execution of the round 2 and 2nd frame goes for the round1. On completion of round 1 of 2nd frame, input data for the 3rd frame has been written into the data memory of stage 1.the 3rd frame starts the execution of round 1,2nd frame goes for the execution of round 2 and 1st frame is ready for the execution of round 3. Similarly as one frame completes the execution of one round, it move forward for the next round and upcoming frame takes the position of pervious frame. In this way frames can executes in parallel, on arrival of 10th pipeline is fully occupied thus achieving the maximum throughput.

3.2.1.2 In-place Indexing

The novelty of the proposed architecture is more pronounced around in-place indexing. By employing the in-place indexing data rate of the architecture increases with best utilization the memory. The same data memory of each stage is used for next coming frame thus reducing the hardware resources which results into reduction of area. The technique intelligently removes all the dependences by tracing a single byte and implements the fully parallel architecture that a single byte of plain text is input to the architecture and a single byte is output as cipher in every clock cycle after an initial latency of 16x10 cycles. The in-place indexing technique drastically increases the through put and data rate of the architecture.

Let's explain the mechanism of in-place indexing in detail; basically it is a scheme which writes the upcoming input at the location from where the value is used in the current cycle. The address generation unit will generate the address for reading and writing value from data and key memory. The address is incremented by employing the Eq (3.1) and (3.2).

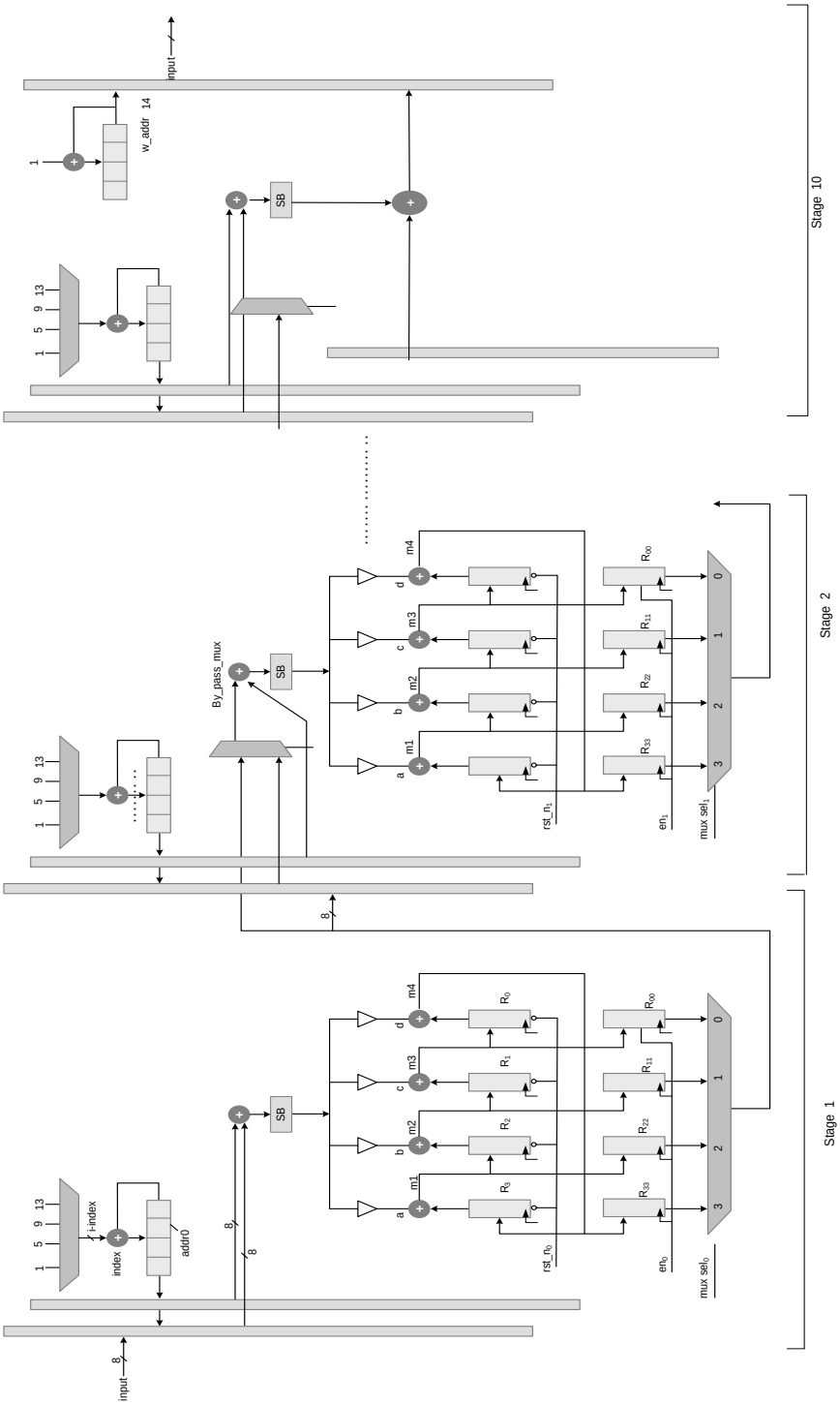


Figure 3.4: Design of Byte Systolic Fully Parallel Architecture

$$addr = (addr + index) \% 16 \quad (3.1)$$

$$index = (index +4) \% 16 \quad (3.2)$$

The value of "addr" a change 16 times within a round, index changes its value after each round and remains constant within a round.

32	88	31	e0
43	5a	31	37
f6	30	a8	07
a8	8d	a2	34

(a)

01	05	09	13
02	06	10	14
03	07	11	15
04	08	12	16

(b)

0a	0e	ac	ba
0b	0f	ad	bb
0c	aa	ae	bc
0d	ab	af	bd

(c)

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

(d)

Figure 3.5: Input data (a) First frame (b) Second frame (c) Third frame (d) Corresponding index of the frames

The plaintext is input to first round in byte serial fashion, the 16 cycles are required to write 16 bytes. The address is initialized and incremented by 1, so for the first write in each data memory is sequential. The Figure 3.5 shows the input data of first, second and third frame with their indices. When 16 bytes of the first frame have been written into the data memory of stage 1 shown in Figure 3.6, the 1st frame starts executing the Round1 of the algorithm. At the same time when first Round of 1st frame is in process the input data for the second frame has been written into the data memory of the stage 1 by employing the in-place indexing. The techniques in-place indexing writes the input data of the second frame at the locations from where the value is read for the execution of Round 1 of the first frame. As in proposed design the data is read in row is

shift order directly from the memory so address are generated by address generation unit which listed in Table 1.

Index = 5

addr = 0

Addr	Adrr=(index+addr)%16
0	5+0 =5
5	5+5 =10
10	5+10 =15
15	5+15 =4
4	5+4 =9
9	5+9 =14
14	5+14 =3
3	5+3 =8
8	5+8 =13
13	5+13 =2
2	5+2 =7
7	5+7 =12
12	5+12 =1
1	5+1 =6
6	5+6 =11

Table 1: Address generation for reading 1st frame in Row Shift order

Memory locations	Indices of frame 1	Data
0	0	32
1	1	43
2	2	F6
3	3	A8
4	4	88
5	5	5A
6	6	30
7	7	8D
8	8	31
9	9	31
10	10	A8
11	11	A2
12	12	E0
13	13	37
14	14	07
15	15	34

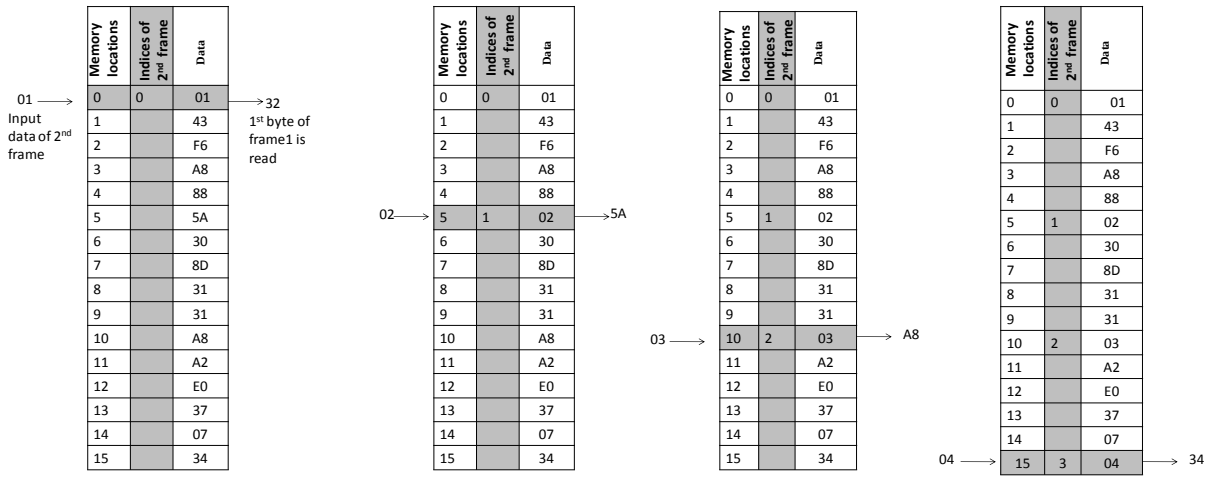
Figure 3.6: Data Memory of Stage 1 after writing first frame of data

As the data is read from the index “0” the first byte of second frame is written at location “0”. The second byte of 1st frame from the read from the index “5”; so the second byte of the 2nd frame is placed at same index. Now as third value of 1st frame is read from location “10” the third byte of the 2nd frame latched into that position.

Similarly as values are read in row shift order from the indices the input data of second frame getting into those locations. The Figure 3.7(a) shows data memory of stage 1 employing in-place indexing for first and second frame and the Figure 3.7(b) shows data memory of stage 1 at the end of round 1 of first frame. On completion of round 1 of first frame the input data for the second frame have been written into the data memory of stage 1; result of round1 of first frame have been written into data memory of stage2. Now first frame goes for the execution of round 2 and second frame can starts the execution of round1. As the first write in each data memory is sequential so result of the round 1of the first frame is written into data memory of stage in sequential order and for reading the data in row shift order index has the value 5. As the Figure 3.7 (b) shows input data of second frame is not saved sequential so now for reading data “addr” is not incremented by 5 instead of it “index” value is 9. By putting the value of index in equation [5] address for the reading the data of 2nd frame can be generated easily. The address generation unit will generate the address as shown in the Table 2 for reading data in row shift order.

Addr	Adrr=(index+addr)%16
0	9+0 =9
9	9+9 =2
2	9+10 =11
15	9+15 =4
4	9+4 =13
13	9+13 =6
6	9+6 =15
15	9+15 =8
8	9+8 =1
1	9+1 =10
10	9+10 =3
3	9+3 =12
12	9+12 =5
5	9+5 =14
14	9+14 =7

Table 2: Address generation for reading 2nd frame in Row Shift order



(a)

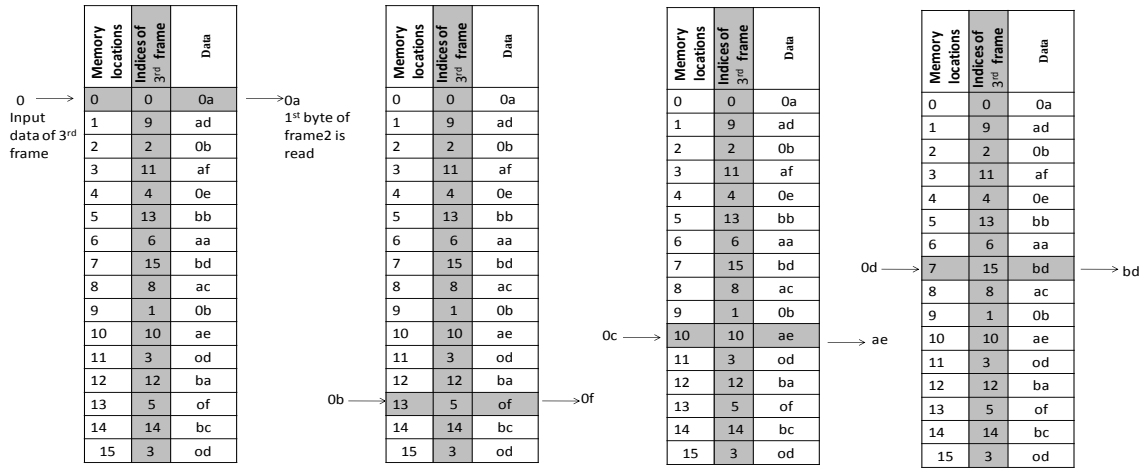
Memory locations	Indices of 2nd frame	Data
0	0	01
1	13	14
2	10	11
3	7	8
4	4	5
5	1	2
6	14	15
7	11	12
8	8	9
9	5	10
10	2	11
11	15	12
12	12	13
13	9	10
14	6	15
15	3	16

(b)

Figure 3.7: In-place Indexing for 1st and 2nd frames (a) Reading and Writing into data memory of stage 1 by employing In-place Indexing (b) Data memory of stage 1 after completion of Round1 of first frame.

As values are read from the indices 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12, 5, 14, 7 at the same time input data for the third frame have been written into the data memory of stage 1. Now first frame can go for round 3, second frame can go for the execution round 2 and third frame can start the execution of the round1. The Figure 3.8 shows that data for the 3rd frame is also not in a sequential order so for reading in row shift order by implementing the Eq (3.1) and (3.2).Similarly for the fourth frame of input data same equations are implemented to read in row shift order. The value of index changes after the completion of each round and remains constant

with the each round, its value repeats after every four frame. The Figure 3.9 (a) shows the write address for the first four frames in the data memory of stage 1 and (b) shows the indices for reading the data in row shift order. This address pattern repeats after every fourth frame of data.



Memory locations	Indices of 3rd frame	Data
0	0	0a
1	9	ad
2	2	0b
3	11	af
4	4	0e
5	13	bb
6	6	aa
7	15	bd
8	8	ac
9	1	0b
10	10	ae
11	3	od
12	12	ba
13	5	of
14	14	bc
15	3	od

(a)

Memory locations	Indices of 3rd frame	Data
0	0	0a
1	9	ad
2	2	0b
3	11	af
4	4	0e
5	13	bb
6	6	aa
7	15	bd
8	8	ac
9	1	0b
10	10	ae
11	3	od
12	12	ba
13	5	of
14	14	bc
15	3	od

(b)

Figure 3.8: In-place Indexing for 2nd and 3rd frames (a) Reading and Writing into data memory of stage 1 by employing In-place Indexing (b) Data memory of stage 1 after completion of Round1 of second frame.

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

0	4	8	12
13	1	5	9
10	14	2	6
7	11	15	13

0	4	8	12
9	13	1	5
2	6	10	14
11	15	3	7

0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11

(a)

0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11

0	4	8	12
9	13	1	5
2	6	10	14
11	15	3	7

0	4	8	12
13	1	5	9
10	14	2	6
7	11	15	13

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

(b)

Figure 3.9: Byte in place indexing for byte systolic AES architecture. (a) Indices for writing data for first four frames. (b) Indices for reading in row shift order for first four frames

3.3 Mapping of AES on Byte Systolic Fully Parallel Architecture

The 128-bit AES encryption algorithm is mapped to proposed architecture 8-bit Systolic Architecture. All the operations of the algorithm are modified according to the 8-bit design. The flow of the proposed algorithm is shown in the Figure 3.10, all the rounds are similar expect the last round in which mixcolumn is replaced by add round key operation. Let's have the brief introduction of the basic operation of the proposed algorithm, which are as follow:

- Shift Row
- Add Round Key
- Byte Substitution
- Mix Column

3.3.1 Shift Row

Normally Shift Row operation is performed in such a way that row 1,2,3,4 is circularly shifted to

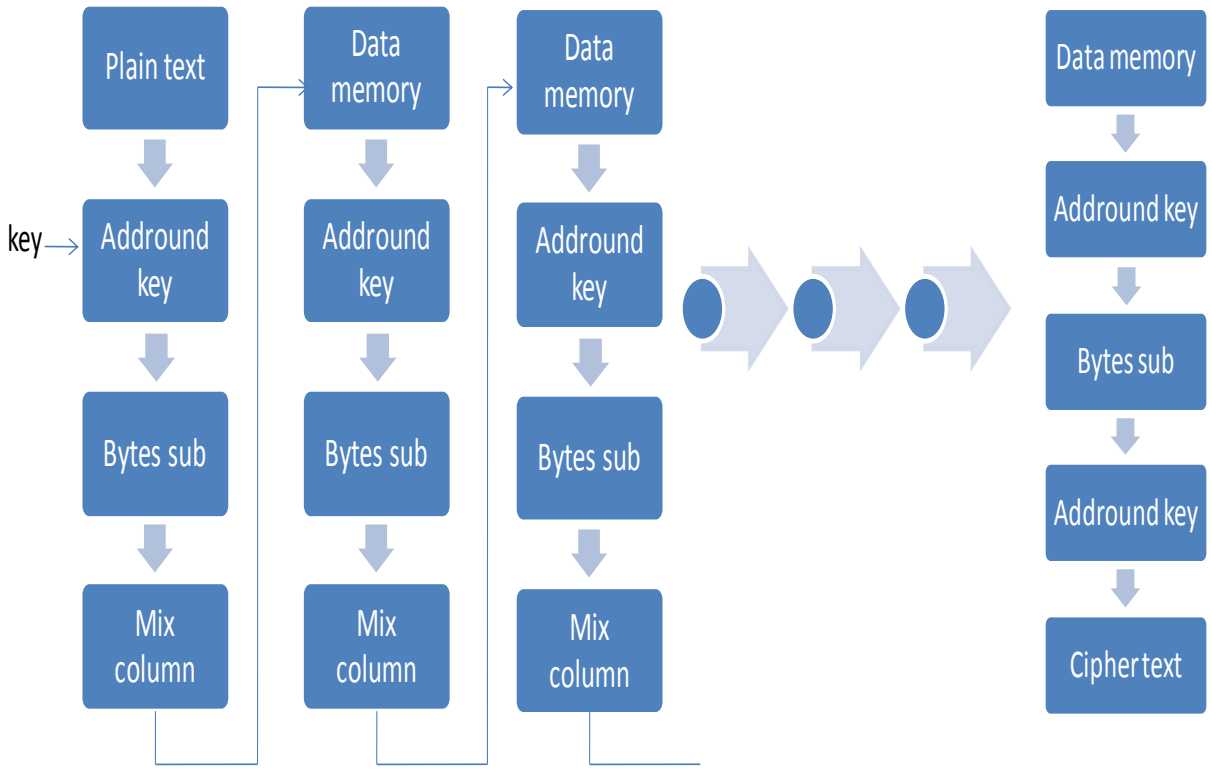


Figure 3.10: Flow of algorithm for Encryption

left by an offset of 0,1,2,3 respectively. Shift row operation in done on the state matrix (16 bytes) as shown in the Figure 3.11.

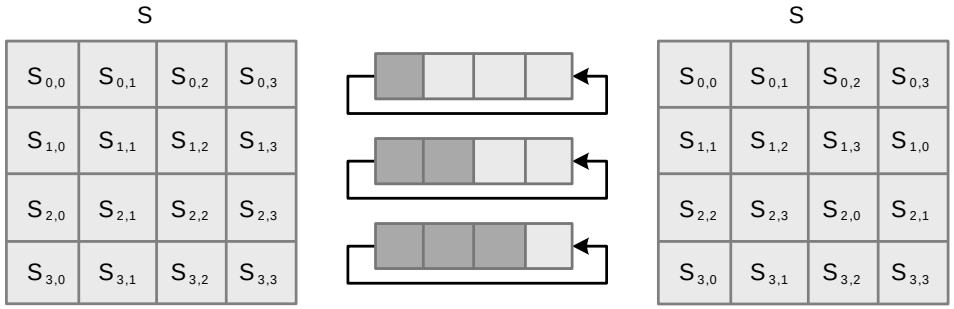


Figure 3.11: Shift Row operation

As proposed architecture is byte systolic so all the computation is on a single byte of a data to achieve this goal the technique proposed is; the data is read from the memory in shift row order as shown in Figure 3.12(b), so logic of shift row operation is no more required.

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

(a)

0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11

(b)

Figure 3.12: State Index for Shift Row operation (a) Original index (b) After Shift Row operation.

Each iteration of the algorithm reads the 8-bit data directly from the memory and the address to access the memory is generated by address generation unit by implementing the following equation.

$$\textbf{AddrSR = (AddrSR + out) \% 16} \qquad (3.3)$$

The value of 'out' varies for each frame for first, second, third and fourth frame of data values of 'out' is 5,9,13 and 1 respectively; it repeats its value after four frames.

3.3.2 Add Round Key

In this step data is exclusive OR with the round key; normally in Add Round Key state matrix is XOR with the key as shown in Figure 3.13. The proposed technique maps the AES to byte systolic architecture so all the operation will be on 8-bit data. A single byte of data is read from the memory is XOR with the single byte of round key, as data is read in row shift order so round key is also read in row shift order. To synchronize the value of data and a key same address generation unit will generate the address for both data and key memory as shown in Figure 3.14.

32	88	31	e0
43	5a	31	37
f6	30	a8	07
a8	8d	a2	34

⊕

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

=

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

Figure 3.13: Add Round Key

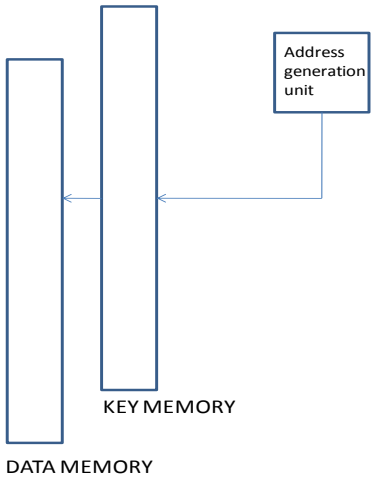


Figure 3.14: Address generation unit deriving the data and key memory

3.3.3 Byte Substitution

The result from the Add Round key is fed into byte substitution module, the proposed byte substitution is same as of normal BS of AES algorithm except computation is done on 8-bit data. Figure 3.15 shows the byte substitution of a proposed architecture.

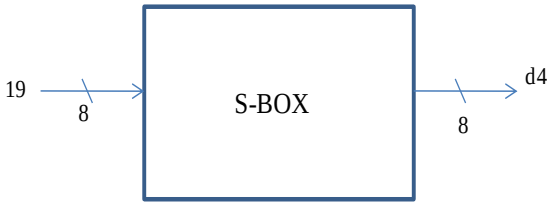


Figure 3.15: Proposed Byte Substitution

3.3.4 Mix Column

In all the rounds, 8-bit result computed from Byte Substitution is than passed to Mix Column module, except the last round which does not have the mix column module. Mix column is a linear transformation based upon GF (2^8). The Figure 3.16 shows the multiplication of a column in mix column stage. One column of a multiplicand matrix is multiply with multiplier matrix and result we get is a column vector "M". The column of multiplicand which goes for multiplication is replaced with column vector "M".

$$\begin{bmatrix} C_{00} & C_{10} & C_{20} & C_{30} \\ C_{01} & C_{11} & C_{21} & C_{31} \\ C_{02} & C_{22} & C_{22} & C_{32} \\ C_{03} & C_{33} & C_{23} & C_{33} \end{bmatrix} \quad \textbf{(.*)} \quad \begin{bmatrix} \mathbf{02} & \mathbf{03} & \mathbf{01} & \mathbf{01} \\ \mathbf{01} & \mathbf{02} & \mathbf{03} & \mathbf{01} \\ \mathbf{01} & \mathbf{01} & \mathbf{02} & \mathbf{03} \\ \mathbf{03} & \mathbf{01} & \mathbf{01} & \mathbf{02} \end{bmatrix}$$

Input data (multiplicand) **Predefined matrix (multiplier)**

(a)

$$\begin{bmatrix} C_{00} \\ C_{01} \\ C_{02} \\ C_{03} \end{bmatrix} \quad \textbf{(.*)} \quad \begin{bmatrix} \mathbf{02} & \mathbf{03} & \mathbf{01} & \mathbf{01} \\ \mathbf{01} & \mathbf{02} & \mathbf{03} & \mathbf{01} \\ \mathbf{01} & \mathbf{01} & \mathbf{02} & \mathbf{03} \\ \mathbf{03} & \mathbf{01} & \mathbf{01} & \mathbf{02} \end{bmatrix} \quad = \quad \mathbf{M} \quad = \quad \begin{bmatrix} \boldsymbol{P0} \\ \boldsymbol{P1} \\ \boldsymbol{P2} \\ \boldsymbol{P3} \end{bmatrix}$$

(b)

$$\mathbf{P_0 = 2C_{00}+3C_{01}+C_{02}+C_{03}}$$

$$\mathbf{P_1 = C_{00}+2C_{01}+3C_{02}+C_{03}}$$

$$\mathbf{P_2 = C_{00}+C_{01}+2C_{02}+3C_{03}}$$

$$\mathbf{P_3 = 3C_{00}+C_{01}+ C_{02}+2C_{03}}$$

(c)

Figure 3.16: Multiplications in Mix column (a) Multiplier and Multiplicand (b) Multiplication of a column(c) Partial Product Equations

The proposed architecture is byte systolic so it works on a single byte to accomplish this goal mix column is modified in such a way that instead of implementing the partial product equation horizontally equations are implemented vertically and we get new equation which is as follow:

$$X_0 = \begin{bmatrix} 2C_{00} \\ C_{00} \\ C_{00} \\ 3C_{00} \end{bmatrix}, \quad X_1 = \begin{bmatrix} 3C_{01} \\ 2C_{01} \\ C_{01} \\ C_{01} \end{bmatrix}, \quad X_2 = \begin{bmatrix} C_{02} \\ 3C_{02} \\ 2C_{02} \\ C_{02} \end{bmatrix}, \quad X_3 = \begin{bmatrix} C_{03} \\ C_{03} \\ 3C_{03} \\ 2C_{03} \end{bmatrix}$$

$$\mathbf{M = X_0 + X_1 + X_2 + X_3} \qquad (3.4)$$

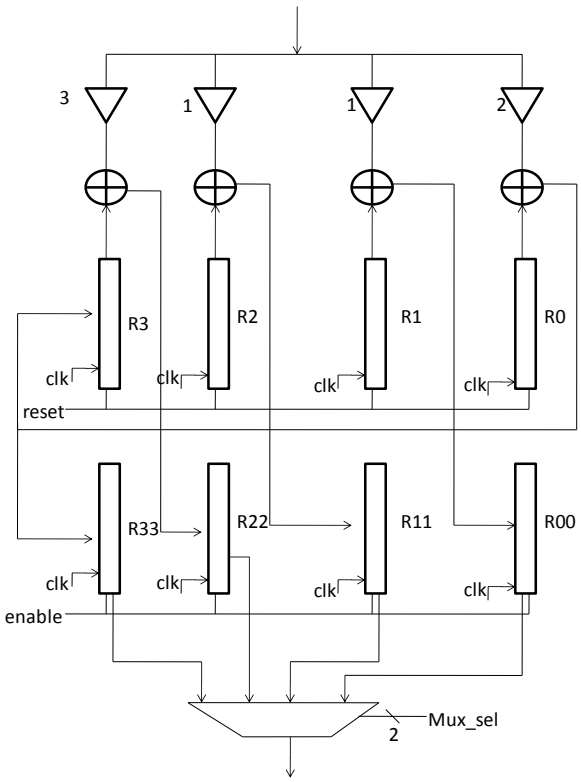


Figure 3.17: Proposed design of Mix column for Byte Systolic Architecture

The Figure 3.17 shows the proposed mix column design for Byte systolic architecture. The design multiplies each upcoming byte with 4 constants, the constant values in matrix are in such a way that each row has the same value but shifted by one to the right. Instead of shifting the multiplier the partial product are shifted to the right by one and saved into the registers (R_0, R_1, R_2, R_3). Each coming byte is multiplied with the constant than resulted value is XOR with the partial product saved into registers. Multiplication of one column requires four cycles and each cycle is discussed.

1st Cycle of Mix Column:

In the first cycle of mix column as shown in Figure 13.18, input C_{00} is multiplied with the 4 constant values and the output is saved into the register R_0, R_1, R_2 and R_3, having the following values:

$$R_0 = C_{00}$$

$$R_1 = C_{00}$$

$R_2 = 3\ C_{00}$

$R_3 = 2\ C_{00}$

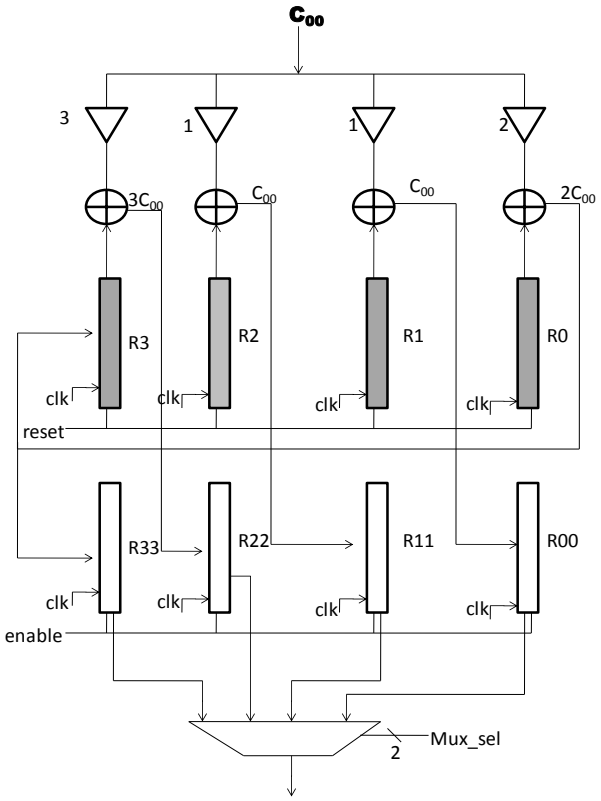


Figure 3.18: 1st Cycle of Mix Column

2nd Cycle of Mix Column:

In second cycle shown in Figure 13.19, C_{01} is input to the mix column module and multiples with 4 constant values than XOR with the shifted values saved into the register R_0, R_1, R_2 and R_3, now output is latched into same registers. Register are having the following values:

$R_0 = R_1 + C_{01} = C_{00} + C_{01}$

$R_1 = R_2 + C_{01} = 3C_{00} + C_{01}$

$R_2 = R_3 + 3\ C_{01} = 2C_{00} + 3C_{01}$

$R_3 = R_0 + 2\ C_{01} = C_{00} + 2C_{01}$

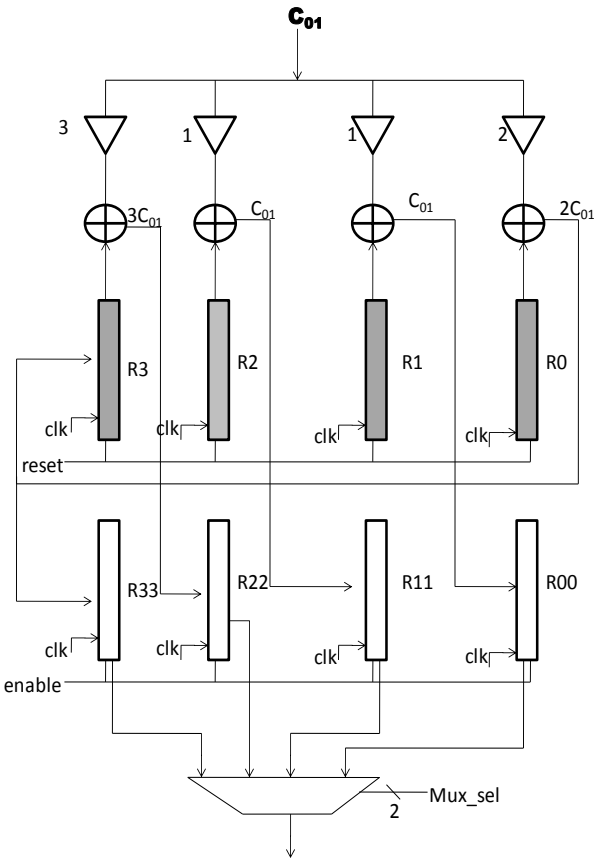


Figure 3.19: 2nd Cycle of Mix Column

3rd Cycle of Mix Column:

C_{02} is input to the mix column module in third cycle, after the multiplication of C_{02} with 4 constant values the result we get is XOR with the values saved into registers now final output is latched into registers R_0, R_1, R_2 and R_3, so registers have the following values. The Figure 13.20 shows the third cycle of mixcolumn.

$$R_0 = R_1 + C_{02} = 3C_{00} + C_{01} + C_{02}$$

$$R_1 = R_2 + C_{02} = 2C_{00} + 3C_{01} + C_{02}$$

$$R_2 = R_3 + 3C_{02} = C_{00} + 2C_{01} + 3C_{02}$$

$$R_3 = R_0 + 2C_{02} = C_{00} + 2C_{01} + 2C_{02}$$

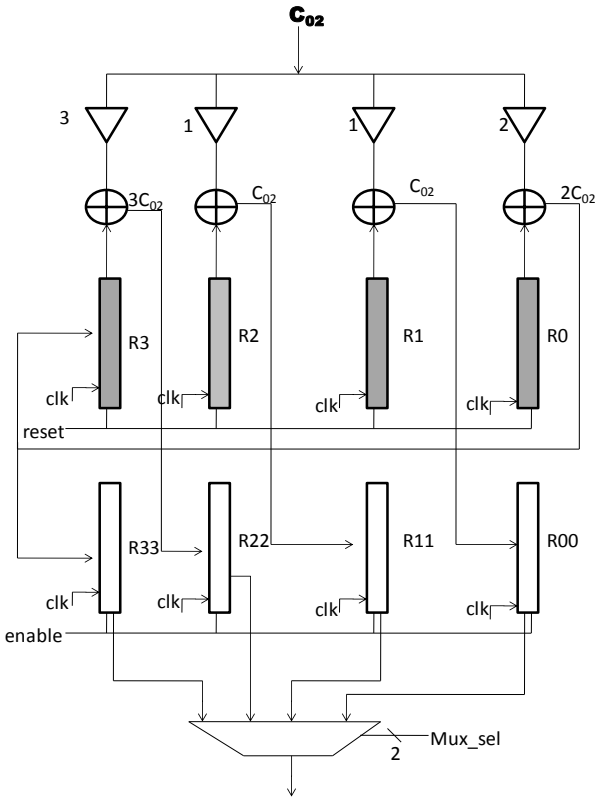


Figure 3.20: 3rd Cycle of Mix Column

4th Cycle of Mix Column:

In the last cycle of mix column C_{03} is input to mix column, after multiplication with constant the result is XOR with the values saved into the register R_0, R_1, R_2 and R_3 now the final result is latched into registers R_{00}, R_{11}, R_{22} and R_{33} and R_0, R_1, R_2, R_3register are reset in every fourth cycle. The Figure 3.21 shows the last cycle of mixcolumn.

$$R_{00} = R_1 + C_{02} = 2C_{00} + C_{01} + C_{02} + C_{03}$$

$$R_{11} = R_2 + C_{02} = C_{00} + 2C_{01} + 3C_{02} + C_{03}$$

$$R_{22} = R_3 + 3C_{02} = C_{00} + C_{01} + 2C_{02} + 3C_{03}$$

$$R_{33} = R_0 + 2\,C_{02} = 3C_{00} + C_{01} + C_{02} + 2C_{03}$$

$$R_0 = 0$$

$$R_1 = 0$$

$$R_2 = 0$$

$$R_3 = 0$$

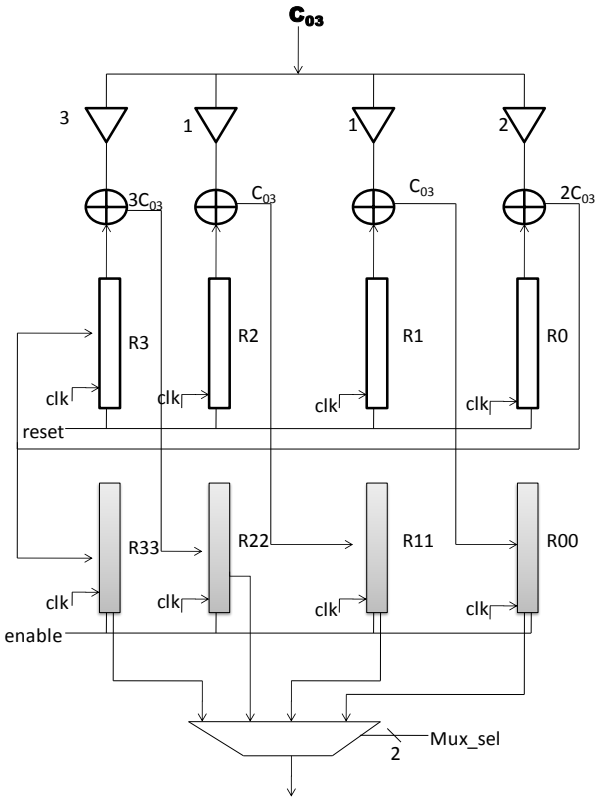


Figure 3.21: 4th Cycle of Mix Column

CHAPTER 4: Implementation of a Byte Systolic Fully Parallel Architecture

The proposed architecture has been implemented in Verilog for encryption. There are two main parts one is the CONTROLLER and other is a DATAPATA; controller will generate the signals to drive the data path. The first part of the chapter is dedicated to the design and implementation of the controller and second part is dedicated to data path implementation. The controller takes the input and generates the signals for the data path when to start and stop the execution of the algorithm, it controls the data reading and writing into the memory and the address generation to access the memory is also controlled by the controller. The register reset is also controlled by the controller. In short Controller is the mind of the implementation which thinks and decides what to do and when to do. The top level design for the implementation of byte systolic fully parallel architecture is shown in figure 4.1.

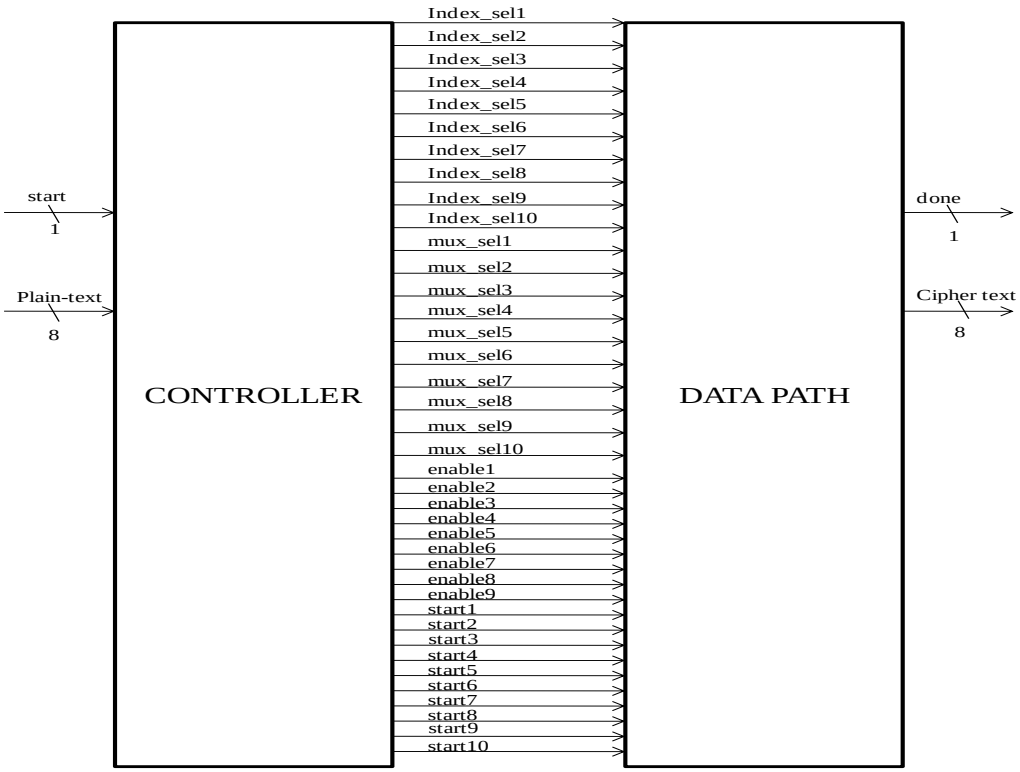


Figure 4.1: Top Level Design for the implementation of byte systolic AES architecture.

Let’s explain the data path and complete design of the controller.

4.1 Data Path Byte Systolic Fully Parallel Architecture

The figure 4.2 shows the RTL diagram of the byte systolic fully parallel architecture. The data path is consists of three main modules which are instantiated to implement the 10 rounds of the AES algorithm.

- Module I
- Module II
- Module III

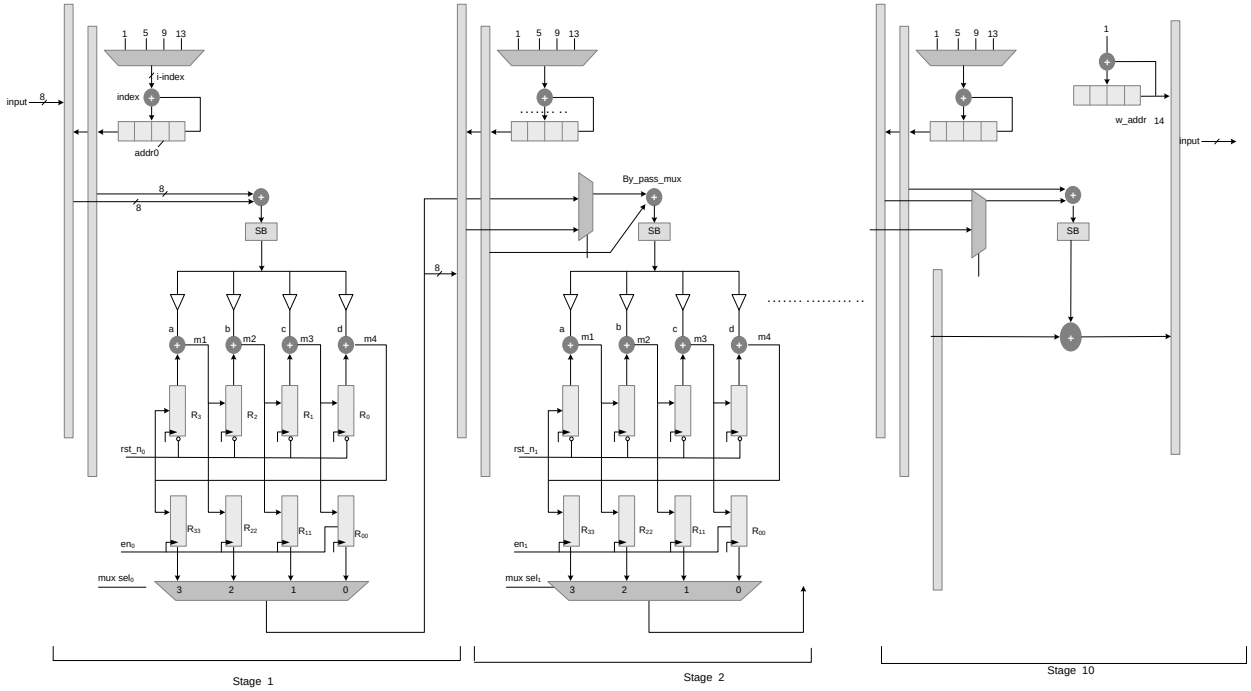


Figure 4.2: RTL diagram of byte systolic fully parallel AES architecture.

4.1.1 Module I

The module I implements the first round of algorithm for each upcoming frame. The module has the data and key memory of 16x8 size, 8-bit wide and 16–bit deep. The signals plain text, clk, reset, start, enable, key, index_sel, muxsel and out_text are the output signal of the module. All the input signals are generated by the controller according to frame number.

Firstly the data is written into the data memory and controller will generate the signals index_sel1 that controls the address of memory where to write the plain text. The address generation unit will generates the address for reading and writing into the memory by implementing the equation [6].The value of index changes after every round; values which are repeated after every four frames are 1, 5, 9 and13. So it is implemented by using the 4:1 MUX and different values of index are as input to the MUX. The 2 bit "index_sel" line is used to select the desired input, the index_sel is controlled by the Controller. Each stage has its own select line "index_sel" i.e. round1 has the index_sel1, round 2 has the index_sel2, round 3 has the index_sel3 and so on.

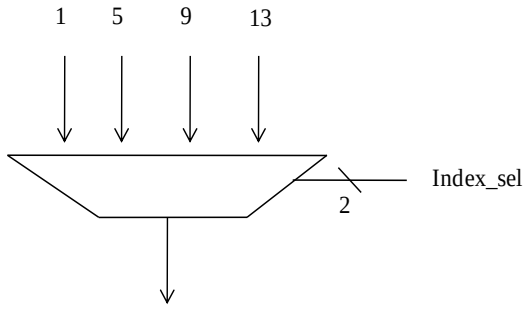


Figure 4.3: MUX for the selection of desired value of index

The implementation of this MUX in verilog is given below:

```
always @ (index_sel)
begin
if(index_sel==2'b00)

out=4'd 1;

else if(index_sel==2'b01)

out=4'd 5;

else if(index_sel==2'b10)

out=4'd 9;

 else if(index_sel==2'b11)

out=4'd 13;
```

```
end
```

The address of the memory is incremented by employing the equation [6], for writing and reading the data. The modulo 16 is implemented by having the register "addr" of 4-bit, so the result will remains in range 0 to 15. As the "addr" is the register so its value is update at every positive edge of the clock cycle. The Verilog code for writing the data into the memory is given below:

```
always @ (posedge clk or negedge reset)

begin
        if(!reset)

        addr<=0;

        else

        if (start)

        addr<=index;

end

//writing  TEXT into data reg and key into reg Rkey

always @ (posedge clk or negedge reset)

begin
if(!reset)

        begin
         for (i = 0; i < 16; i = i + 1)
         begin

         data[i] <= 0;

        end
end

        else
        begin
        data[addr]<=input_text;
```

```
        end
end
```

Step 1: Shift Row

The first step of the proposed algorithm is to read the data in row shift directly from the memory the index value is selected by the controller according to the frame number.

Step 2: Add Round Key

In this step data is XOR with the key both the data and key is read in row shift order and same "addr" will generate the address for two memories.

```
assign A_ARK= data[addr]^Rkey[addr];
```

Step 3: Byte Substitution

The result from the step 2 is fed into the 16x16 lookup table S_BOX for the byte substitution. The S-BOX is saved into the memory of 256x8; 256-bits deep and 8-bit wide. The 8-bit input data to S-BOX for byte substitution results into 8-bit output data. The verilog code of the S-BOX is given below:

```
initial

begin

S_BOX[0] = 8'h 63;

S_BOX[1] = 8'h 7c;

S_BOX[2] = 8'h 77;

S_BOX[3] = 8'h 7b;

S_BOX[4] = 8'h f2;

S_BOX[5] = 8'h 6b;
```

-
-
-

```
S_BOX[251] = 8'h 0f;

S_BOX[252] = 8'h b0;

S_BOX[253] = 8'h 54;

S_BOX[254] = 8'h bb;

S_BOX[255] = 8'h 16;

end
```

Step 4: Mix Column

After byte substitution result is fed into the mix column, where the input byte is multiplied by four constants 3, 1, 1 and 2 which are saved as a, b, c and d respectively. The multiplied result is XOR with the values in registers R_0, R_1, R_2, and R_3 and the final result is latched into the register in accordance with the number of cycle of mix column. The Verilog code of the mix column is given below:

```
assign a= d ^ A_SB;                                    //by 3 (multipl by 2 and xor with its self

assign b= A_SB;                                        //by 1

assign c= A_SB;                                        //by 1

assign d= A_SB[7]  ?  (A_SB<<1  ^  8'h1b ) : ( A_SB<<1 );      //multiplication by 2(right shift 1)

assign m1=a^R3;

assign m2=b^R2;

assign m3=c^R1;

assign m4=d^R0;

always @ (posedge clk or negedge reset)

begin

if((!reset))// & (!enable))
```

```
            begin
            R0<=0;
            R1<=0;
            R2<=0;
            R3<=0;
            count<=0;
            end
else if(count<4)
            begin
            R0<=m3;
            R1<=m2;
            R2<=m1;
            R3<=m4;
            count<=count+1;
             end
else
            begin
            R0<=0;
            R1<=0;
            R3<=0;
            R00<=m3;
            R11<=m2;
            R22<=m1;
            R33<=m4;
```

```
        count<=1;

 end
end
```

Step 5: Dispatching Of Result

The last step of each stage is to dispatch the result of mix column to the next stage. The values saved in registers R_{00}, R_{11}, R_{22} and R_{33} have to be latched into the data memory of the next stage. This achieved by placing a 4:1 MUX with the select line "mux_sel" and the value of the "mux_sel" is selected by the controller. Similar to "index_sel" each stage has its own "mux_sel" line.

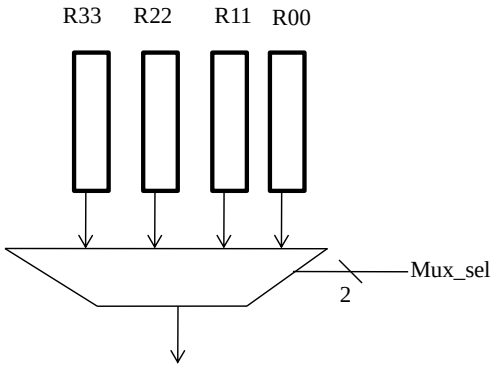


Figure 4.4: MUX for the latching the result of mix column to next stage

Verilog code is given below:

```
//mux for writing the byte in next block

always @ (muxsel, R00, R11, R22, R33)

begin

if(muxsel==2'b00)

        out1<=R00;

else if(muxsel==2'b01)

        out1<=R11;

else if(muxsel==2'b10)
```

```
        out1<=R22;
else if(muxsel==2'b11)
        out1<=R33;
end
```

4.1.2 Module II

The module II is same as of module I the only change is of by-pass MUX, which is used to bypass the last value of each stage and directly sent to add round key operation of next stage. This bypass MUX is used to avoid pipeline stall.

4.1.3 Module III

The module III has some changes; having the two key memories; mix column operation is replaced with a XOR operation. The result from the byte substitution is XOR with round key 10 and giving the single byte of cipher text that is latched into the output register.

4.2 Controller for the Byte Systolic Fully Parallel Architecture

All the signals of data path are controlled by the controller. The controller will generate the signal for each frame in accordance of its round number. As all the rounds are executed in parallel this is achieved by cascading the all stages. This cascading of module is implemented in Verilog by instantiate the module I, module II and module III; output of the one module is deriving the input of the next module. The module I, II and III are instantiated in Verilog as given below and to implement round 3 to round9 mod2 will be instantiated.

mod1 round1(.input_text(plain_text), .out_text(out1), .clk(clk), .reset(reset), .index_sel(index_sel1), .muxsel(muxsel1), .key(cipher_key), .start(start) , .enable(enable1));

mod2 round2(.inputdata(out1), .out_text(out2), .clk(clk), .reset(reset2), .index_sel(index_sel2), .data_sel(dsel),.muxsel(muxsel2),key({R1[0],R1[1],R1[2],R1[3],R1[4],R1[5],R1[6],R1[7],R1[8],R1[9],R1[10],R1[11],R1[12],R1[13],R1[14],R1[15]}), .start(start2) , .enable(enable2));

mod3 final(.input_text(out9), .cipher_text(cipher_text), .clk(clk), .reset(reset10), .index_sel(index_sel10), .data_sel(dsel), .start(start10), .start_f(start_f));

4.2.1 States of Controller:

The controller is design in such a way that it divides 10 round of algorithm into 10 states. Each state generates the signals for the every upcoming frame according to its round number. Following are the states with their functionality:

4.2.1.1 State S0

The rod 0 signal is set when start signal is high and as rod is set then state S0 start executing. In this state only data of the first frame has to written into the data memory of the stage 1. So the following signals are set to write the data into the memory.

Rod0 =1;

Index_sel = “00”;

So Out=1;

Start=1;

When data writing is completed then rod 1 signal is set so state S1 can start.

4.2.1.2 State S1

In this state first frame starts the execution of its round 1, at the same time input data for the second frame have been written into the data memory of stage 1.the result of the round 1 of the frame 1 has to be written into the data memory of the stage 2. Following are the signals which are set to achieve this goal.

Index_sel1=”01”;

Index_sel2=”00”;

Mux_sel1=”00” “01” “10” “11”

On completion of round 1 of first frame rod 2 signal is set to activate the next state S2.

4.2.1.3 State S2

In this state first frame go for the execution of round 2 and second frame starts the execution of its round1.the input data for the third frame has been written into the data memory of the stage 1.

Start3=1;

Index_sel1=”10”;

Index_sel2=”01”;

Index_sel3=”00”;

Mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

When state S2 is done then rod 3 is set high for the activation of S3.

4.2.1.4 State S3

As the rod 3 is set so state S3 starts its working. The first frame goes for the execution of round 3, second frame starts executes the round 2 and third frame goes for the round1.

Start4=1;

Index_sel1=”11”;

Index_sel2=”10”;

Index_sel3=”01”;

Index_sel4=”00”;

Mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

The signal rod 4 is set on completion of state S3.

4.2.1.5 State S4

In this state first frame start the execution of round 4, second frame goes for round 3, third frame for round 2, fourth frame for found 1 and fifth frame has to be written into the data memory of the stage 1

Start5=1;

Index_sel1=”00”;

Index_sel2=”11”;

Index_sel3=”10”;

Index_sel4=”01”;

Index_sel5=”00”;

Mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

mux_sel4=”00” “01” “10” “11”

The signal rod 5 is set on completion of state S4.

4.2.1.6 State S5

As rod 5 is set high state S5 starts its working. 1st frame starts the execution of round 5, 2nd frame goes for round 4, 3rd frame for round 3, 4th frame for round 2, 5th frame for round 1 and 6th has to be written into the data memory of stage 1.

Start6=1;

Index_sel1=”01”;

Index_sel2=”00”;

Index_sel3=”01”;

Index_sel4=”10”;

Index_sel5=”01”;

Index_sel6=”00”;

Mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

mux_sel4=”00” “01” “10” “11”

mux_sel5=”00” “01” “10” “11”

The signal rod 6 is set on completion of state S5.

4.2.1.7 State S6

As rod 6 is set high state S6 starts its working. 1st frame starts the execution of round 6, 2nd frame goes for round 5, 3rd frame for round 4, 4th frame for round 3, 5th frame for round 2, 6th frame for round 1 and 7th has to be written into the data memory of stage 1.

Start7=1;

Index_sel1=”10”;

Index_sel2=”01”;

Index_sel3=”10”;

Index_sel4=”11”;

Index_sel5=”10”;

Index_sel6=”01”;

Index_sel7=”00”;

mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

mux_sel4=”00” “01” “10” “11”

mux_sel5=”00” “01” “10” “11”

mux_sel6=”00” “01” “10” “11”

The signal rod 7 is set on completion of state S6.

4.2.1.8 State S7

As rod 7 is set high state S7 starts its working. 1st frame starts the execution of round 7, 2nd frame goes for round 6, 3rd frame for round 5, 4^{th} frame for round 4, 5^{th} frame for round 3, 6^{th} frame for round 2, 7^{th} for round 1and 8^{th} frame has to be written into the data memory of stage 1.
Start8=1;

Index_sel1=”11”;

Index_sel2=”10”;

Index_sel3=”11”;

Index_sel4=”00”;

Index_sel5=”11”;

Index_sel6=”10”;

Index_sel7=”01”;

Index_sel8=”00”;

mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

mux_sel4=”00” “01” “10” “11”

mux_sel5=”00” “01” “10” “11”

mux_sel6=”00” “01” “10” “11”

mux_sel7=”00” “01” “10” “11”

mux_sel8=”00” “01” “10” “11”

The signal rod 8 is set on completion of state S7.

4.2.1.9 State S8:

As rod 8 is set high state S8 starts its working. 1st frame starts the execution of round 8, 2nd frame goes for round 7, 3rd frame for round 6, 4th frame for round 5, 5th frame for round 4, 6th frame for round 3, 7th for round 2, 8th for round1and 9th frame has to be written into the data memory of stage 1.

Start9=1;

Index_sel1=”00”;

Index_sel2=”11”;

Index_sel3=”00”;

Index_sel4=”01”;

Index_sel5=”11”;

Index_sel6=”11”;

Index_sel7=”10”;

Index_sel8=”01”;

Index_sel9=”00”;

mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

mux_sel4=”00” “01” “10” “11”

mux_sel5=”00” “01” “10” “11”

mux_sel6=”00” “01” “10” “11”

mux_sel7=”00” “01” “10” “11”

mux_sel8=”00” “01” “10” “11”

mux_sel9=”00” “01” “10” “11”

The signal rod 9 is set on completion of state S8.

4.2.1.10 State S9

As rod 9 is set high state S9 starts its working. 1st frame starts the execution of round 9, 2nd frame goes for round 8, 3rd frame for round 7, 4th frame for round 6, 5th frame for round 5, 6th frame for round 4, 7th for round 3, 8th for round 2, 9th for round 1and 10th frame has to be written into the data memory of stage 1.

Start9=1;

Index_sel1=”01”;

Index_sel2=”01”;

Index_sel3=”01”;

Index_sel4=”10”;

Index_sel5=”00”;

Index_sel6=”00”;

Index_sel7=”11”;

Index_sel8=”10”;

Index_sel9=”01”;

Index_sel9=”00”;

mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

mux_sel4=”00” “01” “10” “11”

mux_sel5=”00” “01” “10” “11”

mux_sel6=”00” “01” “10” “11”

mux_sel7=”00” “01” “10” “11”

mux_sel8=”00” “01” “10” “11”

mux_sel9=”00” “01” “10” “11”

mux_sel9=”00” “01” “10” “11”

The signal rod 10 is set on completion of state S9.

4.2.1.11 State S10

As rod 10 is set high state S10 starts its working. 1st frame starts the execution of round 10, 2nd frame goes for round 9, 3rd frame for round 8, 4th frame for round 7, 5th frame for round 6, 6th frame for round 5, 7th for round 4, 8th for round 3, 9th for round 2and 10th frame starts executing the round 1.

Start10=1;

Index_sel1=”10”;

Index_sel2=”10”;

Index_sel3=”10”;

Index_sel4=”11”;

Index_sel5=”01”;

Index_sel6=”01”;

Index_sel7=”00”;

Index_sel8=”11”;

Index_sel9=”10”;

Index_sel9=”01”;

Index_sel10=”00”;

The signals for dispatching the result of one round into the data memory of the next stage.

mux_sel1=”00” “01” “10” “11”

Mux_sel2=”00” “01” “10” “11”

Mux_sel3=”00” “01” “10” “11”

mux_sel4=”00” “01” “10” “11”

mux_sel5=”00” “01” “10” “11”

mux_sel6=”00” “01” “10” “11”

mux_sel7=”00” “01” “10” “11”

mux_sel8=”00” “01” “10” “11”

mux_sel9=”00” “01” “10” “11”

mux_sel10=”00” “01” “10” “11”

CHAPTER 5: Results

The Xilinx Project Navigator ISE 12.1I Suite is used for simulation, synthesizing and implementation (translate, map, place and route). All the modules are simulated and the result can be depicted in the form of wave forms. Different value of all the signals at different instant can be viewed easily; the section 5.1 shows the simulated results of the AES encryption. The chapter also includes the synthesis report and the comparison with other AES architectures.

5.1 Simulation

The built in simulator ISlim of Xilinx 12.1 is used for the simulations .Following are simulated wave forms of plaintext, round key, signals of address generation unit and the final cipher text.

5.1.1 Plain text

Following is wave form of input plain text. At every clock a byte of pain text is input into the system.

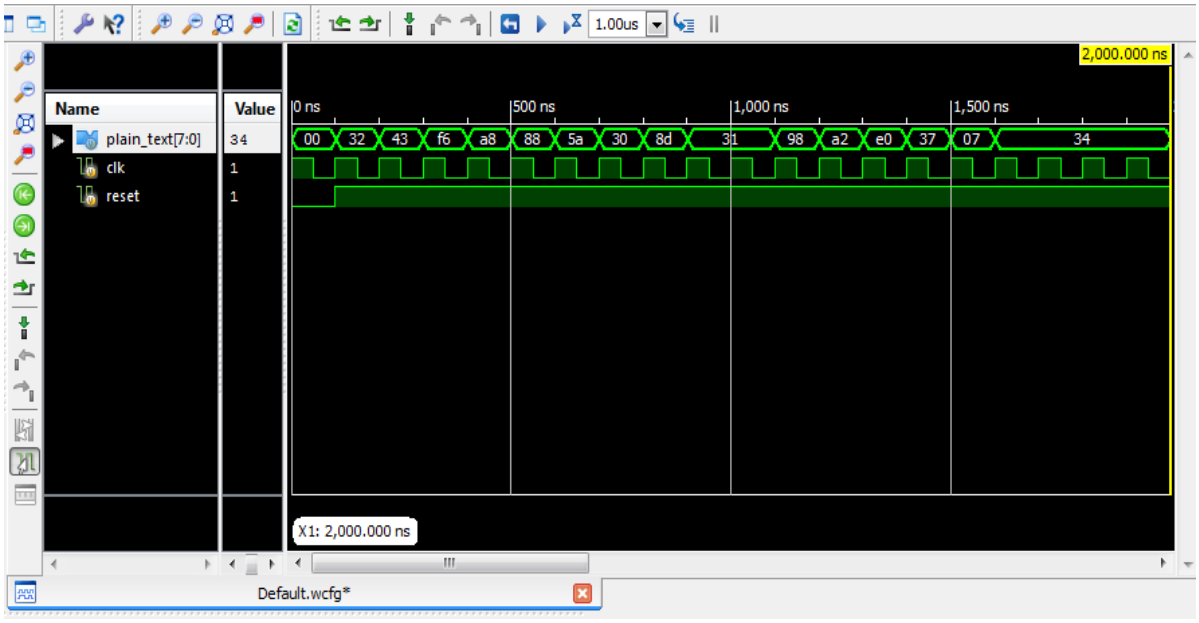


Figure 5.1: Plain wave form of input plain text

5.1.2 Round Key

Each stage has its corresponding round key, key are saved in the memory is shown below:

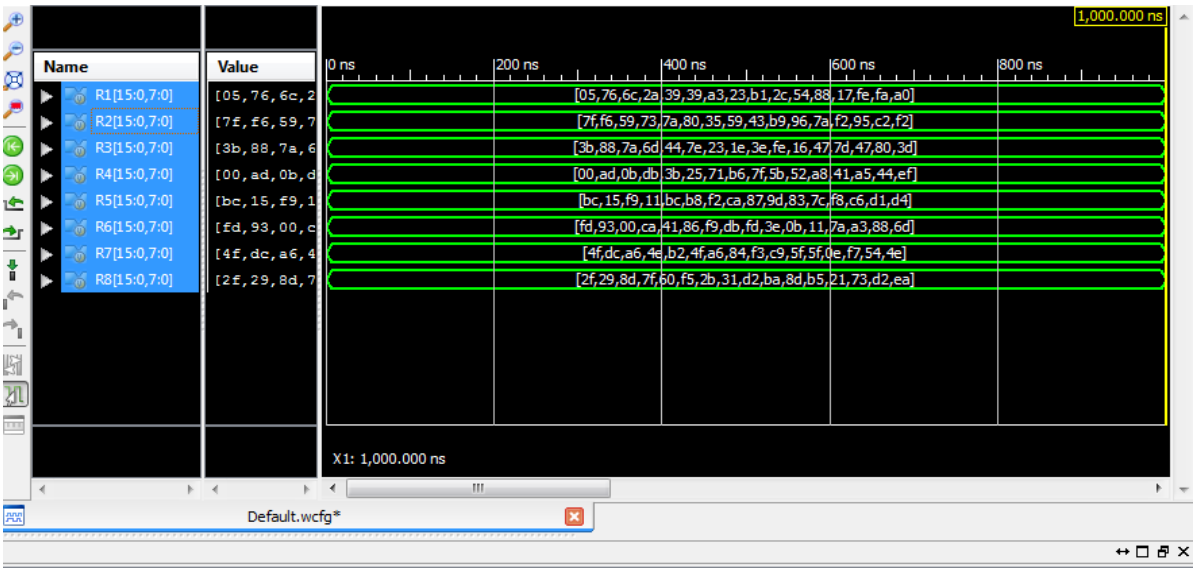


Figure 5.2: Stage Round Key stored in memory

5.1.3 Address Generation Unit

All the 10 rounds are implemented in parallel for every upcoming frame. Index-sel is different for every incoming frame in accordance with its round number. Following are 10 wave forms of all the rounds with their index-sel values.

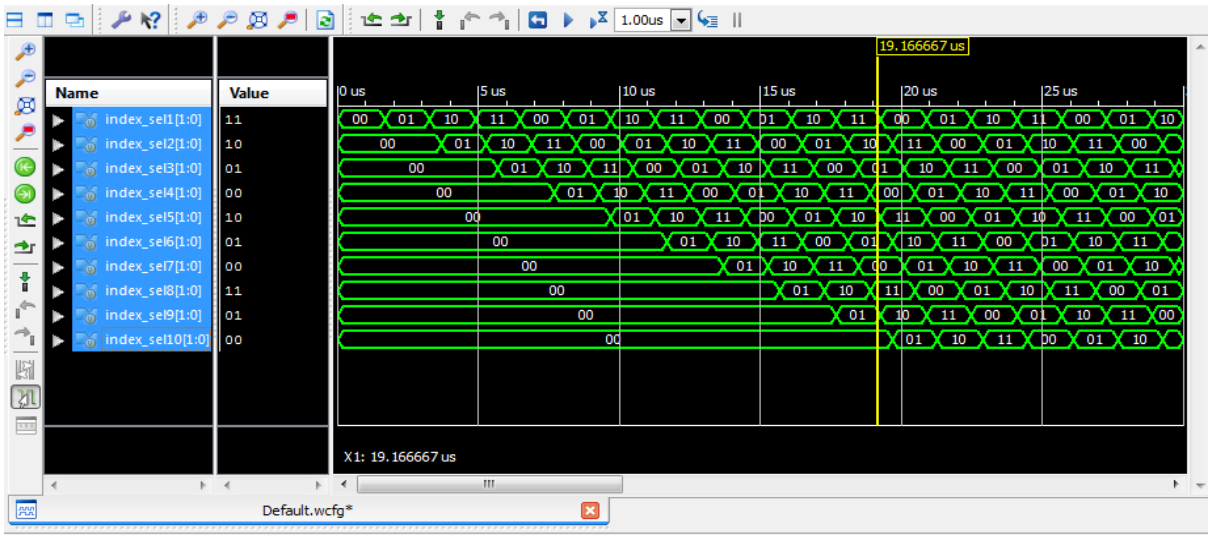


Figure 5.3: 10 wave forms of all the rounds with their index-sel values

5.1.4 Cipher text

Following is wave form of cipher text, at every clock cycle a byte of cipher text is coming out.

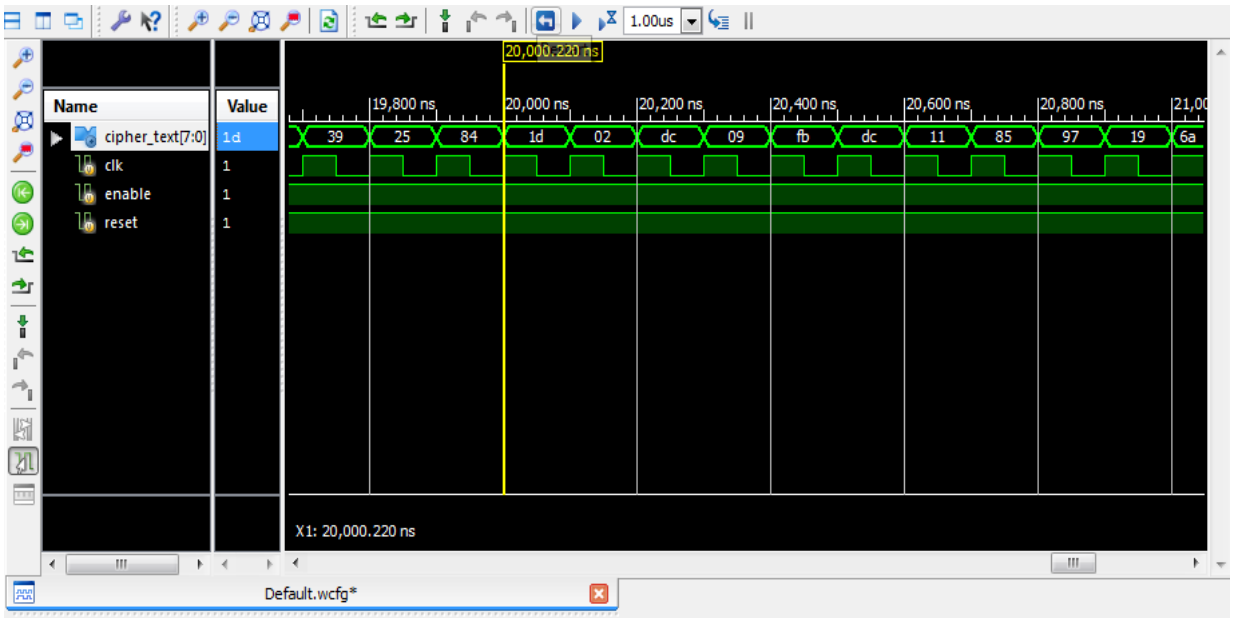


Figure 5.4: Waveform of cipher text

5.3 Synthesis Report

The device utilization and timing summary on a target device XC4VSX55 is given below:

```
Timing Summary:
---------------
Speed Grade: -12

   Minimum period: 4.909ns (Maximum Frequency: 203.705MHz)
   Minimum input arrival time before clock: 2.765ns
   Maximum output required time after clock: 9.882ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 4.909ns (frequency: 203.705MHz)
  Total number of paths / destination ports: 215421 / 1367
-------------------------------------------------------------------------
```

```
Device utilization summary:
---------------------------

Selected Device : 4vsx55ff1148-12

 Number of Slices:                      934  out of  24576     3%
 Number of Slice Flip Flops:            921  out of  49152     1%
 Number of 4 input LUTs:               1677  out of  49152     3%
 Number of IOs:                         147
 Number of bonded IOBs:                  11  out of    640     1%
 Number of GCLKs:                         2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


================================================================================
```

5.4 Comparison of Hardware Utilization

In this section a comparison of hardware utilization of the proposed design with other architectures. It is clear from the table 3 that proposed architecture utilizes the least hardware resources.

Author	*Devices*	*Slices*	*BRAM*	*Throughput (MHz)*	*Operating speed (MHz)*
Farhan et al [18]	XC2V1000-6	337	2	53	110
Frahan et al [16]	XC2V1000-6	236	1	41.6	117
Good et al [19]	XCV-100-4	119	3	0.71	90
Good et al [15]	XC2S15	174	2	2.3	70
Our	**XC4vsx55**	**934**	**0**	**1600**	**200**

Table 3: Comparison of Results

5.5 Performance Analysis

The performance of proposed architecture and other AES designs are determined on the basis of throughput. The Figure 5.1 clearly depict the throughput of the Byte Systolic Fully Parallel Architecture is higher than the iterative AES algorithm. The comparison result shows that proposed design offers the best tradeoff between throughput and area.

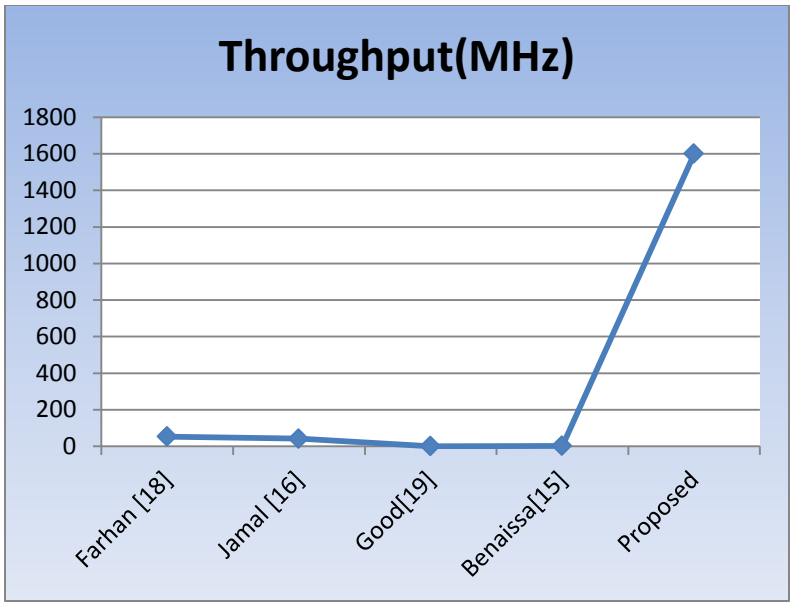


Figure 5.5: Performance Analysis on the basis of Throughput.

CHAPTER 6: Conclusion and Future Work

This chapter concludes the whole dissertation by summarizing the all research work done and presenting the results. In addition to this, it presents possible direction for the future work, which and expand the ideas presented by the dissertation.

6.1 Summary of Research done

There are number of architectures proposed in the literature, selection of a particular architecture depends upon throughput, area, power and data of the input. The intensively pipelined architecture has been used for achieving high throughput, on-the-fly key generation for avoiding the storage key memory, optimized Mixcolumn and efficient S-BOX implementations for the applications with area constrain. Recent research work for AES implementation has focused on compact AES architecture to target the applications in consumer electronics with low cost and low power. A number of AES architectures are proposed in the literature but few of them are mapped to an 8-bit data path. The focus of the research was to design an 8-bit architecture of AES encryption algorithm that works in a systolic manner with high throughput.

The architecture Byte Systolic Fully Parallel Architecture works on byte in-place indexing, the design encrypts the 128-bit of plaintext using key of size of 128-bit. In this technique a byte (8-bit) of plain text is input to the architecture and result in a byte of cipher text as an output in every clock cycles after an initial latency of 16x10 cycles. The 128-bit AES encryption algorithm is mapped to proposed architecture 8-bit Systolic Architecture. All the operations of the algorithm are modified according to the 8-bit design

Pipelining was introduced to achieve high throughput, all the round are implemented in parallel by cascading the stages. Thus resulting architecture does not reuse the logic resources instead executes all the rounds in parallel, thus maximizing the data path size that leads to significantly increase the throughput and data rate. . Every stage has its own data and key memory only last stage has two key memories, the text which has to be encrypted is stored in data memory. In key memory corresponding key is stored in each stage which is expanded offline by key scheduling algorithm. For key size of 128-bit there are 10 rounds keys; for stage 1 cipher key; for stage 2

round key (R1); for stage 3 round key (R2) and so on but in stage 10 round keys R9 and R10 are saved in key memory. The data is input in byte serial fashion to stage 1, execution of the round starts when the 16 bytes have been written into the data memory. As the round 1 for 1st frame is in process at the same time input for the second frame is written into the data memory of the stage 1 by applying in-place index addressing.

The novelty of the proposed architecture is more pronounced around in-place indexing. By employing the in-place indexing data rate of the architecture increases with best utilization the memory. It is a scheme which writes the upcoming input at the location from where the value is used in the current cycle. The address generation unit will generate the address for reading and writing value from data and key memory. The same data memory of each stage is used for next coming frame thus reducing the hardware resources which results into reduction of area. The technique intelligently removes all the dependences by tracing a single byte and implements the fully parallel architecture that a single byte of plain text is input to the architecture and a single byte is output as cipher in every clock cycle after an initial latency of 16x10 cycles. The in-place indexing technique drastically increases the throughput and data rate of the architecture.

6.2 Summary of Results

The Xilinx Project Navigator ISE 12.1I Suite is used for simulation, synthesizing and implementation (translate, map, place and route). The device has been used as a target device. The proposed design utilizes the 934 slices 1% of total recourses, number of bounded I/O is 11 out of 640 thus 1% utilization. Maximum frequency on which design can work is 203.705 Hz and clock period is 4.909ns. The comparison result shows that our design achieves highest throughput and data rate than previous 8-bit architecture.

6.3 Achievement of Research Goals

The proposed architecture fully accomplishes the desired objectives that is mapping of AES algorithm to byte systolic fully parallel architecture for high throughput and data rate. The proposed design is capable of giving out the single byte of cipher in every clock after initial latency of 10x16 cycles. The design works in lock step manner by employing the in-place indexing techniques. The technique intelligently removes all the dependences by tracing a single

byte. The same data memory of each stage has been used for next coming frame thus reducing the hardware resources which results into reduction of area. The proposed AES design for the encryption offers the data rate in the range 200MHz while utilizing the 934 slices and throughput is 1.6 GHz. The result of comparison shows that proposed design offer the best tradeoff between throughput and area.

6.4 Contributions of the Research

A number of AES architectures are proposed in the literature but few of them are mapped to an 8-bit data path. The existing 8-bit AES architecture offers the compact design and best suited for the low area and power applications but the throughput of the 8-bit designs is very low. So the previous 8-bit designs only suited for embedded systems, mobile computing and smart cards. The proposed Byte Systolic Fully Parallel AES architecture offers the highest throughput with a small increase in area compared to other 8-bit designs. So the resulting architecture works for the systems which require the high throughput with low area. The proposed design is also of special interest of the application in which input changes at every clock cycle at faster rate. In addition to this the research conducted has several application areas, image processing, voice communication and multimedia transfer over internet.

6.5 Future Work

Future work, mapping of AES to 4-bit data path for more area optimization.

The proposed design can be updated by optimizing the linear part of S-BOX, for the efficient memory utilization. So the modified architecture will be of interest for the application requires high speed and low power.

By using the fault tolerance technique with proposed design, the resulting design ensures integrity and reliability of the image with high data rate and throughput. This technique targets the satellite applications; it can also be applicable for unnamed aerial vehicles and nuclear reactors.

In order to enhance the algorithm for multi-object evolution techniques the proposed design can

be modified by using the Fuzzy set theory, as fuzzy logic is a powerful tool for modeling the uncertain system.

REFERENCES

[1] Encryption (Jan 2012) [online] http://w.w.w.education.illinois.edu/wp/privacy/encrypt.html/

[2] Cryptography (Jan 2012) [online] http:// w.w.w.sculptors.com/~salsbury/why_use_crypto.txt/

[3] Encryption Standards (Feb. 2012)
[Online] http://w.w.w.instantssl.com/ssl_certificate/product/encrypt

[4] National Institute of Standards and Technology, "Federal Information Processing Standard Publication 197, the Advanced Encryption Standard (AES)," Nov. 2001.

[5] M. Vucha, A. Rajawat, "Design and FPGA Implementation of Systolic Array Architecture for Matrix Multiplication,"Iternational Journal of computer Application, vol. 26, pp. 0975–8887, July 2011.

[6] A. C. Zigiotto, R. d'Amore, "A Low-cost FPGA Implementation of the Advanced Encryption Standard Algorithm," 15th symposium on Integrated Circuits and Systems Design, SBCC1, 2002

[7] N.A. Saqib, F.R. Henriquez and A.D. Perez. "AES Algorithm Implementation- an efficient approach for sequential and pipeline Architecture," Fourth Mexican International Conference on computer Science, EN, 2009.

[8] A. Dandalis, V.K. Prasanna and J.D.P. Rolim, "A Comparative Study of Performance of AES Final Candidates Using FPGAs," Proc. Third Advanced Encryption Standard (AES) Candidate Conference, April 2000.

[9] P.Mroczkowski, "Implementation of the Block Cipher Rijndael Using Altera FPGA," pubcmnts.htm, 2010.

[10] V.Fischer and M. Drutarovsky, "Two Methods of Rijndael Implementation in Reconfigurable Hardware," Proc. CHES 2001, May 2001.

[11] K.Gaj and P.Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," Proc. Third Advanced Encryption Standard (AES) Candidate Conference, April 2000.

[13] AES Encryption Algorithm. (Feb. 2012)

[Online]http://www.sans.org/reading_room/whitepapers/vpns/aes-making-encryption

[14] searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard.

[15] T. Good, M. Benaissa, "Very small FPGA application-specific instruction processor for AES", IEEE Trans. Circuits and Systems-I: Regular papers, vol.53, no.7, pp.1477-1486, July 2006.

[16] S. M. Farhan, S. Ahmed, H. Jamal, "An 8-bit systolic AES architecture for moderate data rate applications", Microprocessors and Microsystems Embedded Hardware Design 33, pp.221-231, 2009.

[17] Finite_field_arithmetic w.w.w.en.wikipedia.org/wiki/Finite_field_arithmetic.

[18] S. M. Farhan, S. A. Khan, H. Jamal, "Mapping of high bit algorithm to low bit for optimized hardware implementation", in Proc 16th International Conference on Microelectronics, Tunis, pp. 148 – 151, December 2004.

[19] T. Good, M. Benaissa, "Very small FPGA application-specific instruction processor for AES", IEEE Trans. Circuits and Systems-I: Regular papers, vol.53, no.7, pp.1477-1486, July 2006.

[20] R. Shahid, M.U. Sharif, M. Rogawski, and K. Gaj, "Use of embedded FPGA resources in implementations of 14 Round 2 SHA-3 candidates," The 2011 International Conference on Field-Programmable Technology, FPT 2011, December 2011.

[21] M. A. Sonai, R. D. Mukhopadhayay, S. Ghosh, D. R. Chowdhury, I. Sengupta, "An area optimized reconfigurable encryptor for AES-Rijndael", in Proc. Design, Automation and Test in Europe, pp. 1116-1121, 2007.

[22] M. Liberatori, J. C. Bonadero, "AES-128 cipher. Minimum area, low cost FPGA implementation," in Transtion The Latin American Applied Research Journal, Vol. 3, pp. 71-77, 2007.

[23] M. Liberatori, F. Otero, J. C. Bonadero, J. Castifieira, "AES-128 cipher. High speed, low cost FPGA implementation," in Proc. Third Southern Conference on Programmable Logic SPL, pp. 195-198, 2007.

[24] G. Rouvroy, F. X. Standaert, J. J. Quisquater, and J. D. Legat, "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," in Proc. International Conference on Information Technology: Coding and Computing , Vol. 2, pp. 583–587, Apr. 2004.

[25] N. E. Fishawy, "Quality of encryption measurement of bitmap images with RC6, MRC6, and rijndael block cipher algorithms," International Journal of Network Security, pp. 241-251, Nov 2007.

[26] X. Zhang, K.K. Parhi and Fellow, "High-Speed VLSI Architectures for the AES Algorithm," IEEE Transactions on vlsi systems, vol. 12, no. 9, pp.957-966, September 2004.

[27] M.H. Sami. and A.R. Reddy, "Performance Analysis of AES and MARS Encryption Algorithms," IJCSI International Journal of Computer Science Issues, Vol. 8, No 1, July 2011.

[28] F. Shao, Z. Chang and Y. Zhang,"AES Encryption Algorithm Based on the High Performance Computing of GPU," Second International Conference on Communication Software and Networks, Nov 2010.

[29] H.O. Alanazi, B.B. Zaidan, A.A. Zaidan, H.A. Jalab, M. Shabbir and Y. Al-Nabhani, "New Comparative Study Between DES, 3DES and AES within Nine Factors," Journal of computing, vol. 2, pp. 2151-9617, March 2010.

[30] S. Soni, H. Agrawal and M. Sharma, "Analysis and Comparison between AES and DES Cryptographic Algorithm," International Journal of Engineering and Innovative Technology (IJEIT) Vol. 2, December 2012.

[31] J. Thakur, N. Kumar, "DES, AES and Blowfish: symmetric Key Cryptography Algorithms Simulation Based performance Analysis," International Journal of Emerging Technology and Advanced Engineering, Vol.2, 2011.

[32] S. Hameed, F. Riaz , R. Moghal, G. Akhtar, Anil Ahmed, Abdul Ghafoor Dar, "Modified Advanced Encryption Standard For Text And Images," Computer Science Journal Vol.1, December 2011.

[33] J. Nechvatal, E. Barker,L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback. "Report on the Development of the Advanced Encryption Standard (AES)," Journal of Research of the National Institute of Standards and Technology, Vol. 106, No. 3, June 2001.

[33] Diaa.S. A. Elminaam, Hatem. M. A. Kader and Mohiy .M. Hadhoud, "Evaluating The Performance of Symmetric Encryption Algorithms," International Journal of Network Security, Vol.10, No.3, pp.213 ~219, May 2010.

[34] J.G.H. Karimian, B. Rashidi, and A.farmani, "A High Speed and Low Power Image Encryption with 128-Bit AES Algorithm," International Journal of Computer and Electrical Engineering, Vol. 4, No. 3, June 2012

[35] S.S. Dhenakaran and N. Kavinilavu, "A New Method for Encryption using Fuzzy Logic," International Journal of Engineering Trends and Technology, Vol.3, Issue3, 2012.

[36] P. Karthigaikumar and S. Rasheed, "Simulation of Image Encryption using AES Algorithm", IJCA Special Issue on Computational Science -New Dimensions & Perspectives, NCCSE, 2011

[37] S.J. Manangi, P. Chaurasia, M. Pratap and S. Global, "Simplified AES for Low Memory Embedded Processors," Journal of Computer Science and Technology, Vol.10, November 2010.

[38] S. T, A. R, G. V. Kumar and Vaidehi , “Pipelining Architecture of AES Encryption and Key Generation with Search Based Memory,” International journal of VLSI design & Communication Systems (VLSICS), Vol.1, No.4, December 2010.

[40] A.T. Hashim, Y.A. Mohammed and E.H. Karam, “FPGA Simulation of Type-3 Feistel Network of the 128 bits block Size Improved Blowfish Cryptographic Encryption,” Eng. & Tech. Journal, vol.28, no.9, 2010.

[41] S.E. Adib and N. Raissouni, “AES Encryption Algorithm Hardware Implementation Architecture: Resource and Execution Time Optimization,” International Journal of Information & Network Security (IJINS) Vol.1, No.2, pp. 110~118, June 2012.

[42] Pravell .H.L , H.S. Jayasumna and M.Z. Kuriaz, “Satellite Image Encryption using AES,” International Journal of Computer Science and Electrical Engineering (IJCSEE) ISSN No. 2315-4209, Vol.1, 2012.

[43] A. Labbe and A. Perez, "AES Implementation on FPGA: Time and Flexibility Tradeoff," in Proceedings of FPL, pp. 836~844, 2002.

[44] G.P. Saggese, A. Mazzeo, N. Mazzocca and A.G.M. Strollo, "An FPGA-Based Performance Analysis of the Unrolling, Tiling, and Pipelining of the AES Algorithm, " In Lecture Notes in Computer Science, Vol.2778, pp.292~302,January 2003.

[45] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," In Cryptographic Hardware and Embedded Systems-CHES, pp. 319~333, 2003.

[46] F. X. Standaert, G. Rouvroy, J.J. Quisquart and J.D. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," In Cryptographic Hardware and Embedded Systems CHES, pp.334~350, 2003.