EFFICIENT HARDWARE IMPLEMENTATIONS FOR THE ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

by

Issam Mahdi Hammad

Submitted in partial fulfilment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
October 2010

© Copyright by Issam Mahdi Hammad, 2010

DALHOUSIE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "EFFICIENT HARDWARE IMPLEMENTATIONS FOR THE ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM" by Issam Mahdi Hammad in partial fulfilment of the requirements for the degree of Master of Applied Science.

Date: October 25, 2010

Supervisor: _____
Dr. Kamal El-Sankary

Co-Supervisor: _____
Dr. Ezz I. El-Masry

Reader: _____
Dr. William J. Phillips

Reader: _____
Dr. Jason Gu

DALHOUSIE UNIVERSITY

DATE: October 25, 2010

AUTHOR: Issam Mahdi Hammad

TITLE: Efficient Hardware Implementations For The Advanced Encryption Standard (AES) Algorithm

DEPARTMENT OR SCHOOL : Department of Electrical and Computer Engineering

DEGREE: MASc CONVOCATION: May YEAR: 2011

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

_____

Signature of Author

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than the brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

To my loving parents Mahdi & Bushra and to all my friends

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
DES	Data Encryption Standard
FIPS	Federal Information Processing Standard
NIST	National Institute of Standards and Technology
S-BOX	Substitution Box
I-BOX	Integrated Box
RFID	Radio Frequency Identification
ATM	Automated Teller Machine
FPGA	Field Programmable Gate Array
KE	Key Expansion
ECC	Elliptic Curve Cryptography
GF	Galois Field
TP	Throughput
RAM	Random Access Memory
BRAM	Block RAM
ROM	Read Only Memory

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Dr. Kamal El-Sankary and Co-supervisor Dr. Ezz I. El-Masry, for their guidance, encouragement and support during my graduate study. Their deep insight and extensive knowledge supported me in my research in digital circuits and especially in the cryptography field. As well, I am would like to thank Dr. Jason Gu and Dr. William J. Phillips for being a part of my supervisory committee.

Many thanks to the group mates in VLSI group for sharing their knowledge and experience. I would also like to express my gratitude to the department staff Selina Cajolais and Nicole Smith.

Most of all, I would like to express my special thanks and appreciation to my parents for their support and encouragement throughout my 2 years masters degree. Also, many thanks to my friends in Halifax for their support.

ABSTRACT

This thesis introduces new efficient hardware implementations for the Advanced Encryption Standard (AES) algorithm. Two main contributions are presented in this thesis, the first one is a high speed 128 bits AES encryptor, and the second one is a new 32 bits AES design. In first contribution a 128 bits loop unrolled sub-pipelined AES encryptor is presented. In this encryptor an efficient merging for the encryption process sub-steps is implemented after relocating them. The second contribution presents a 32 bits AES design. In this design, the S-BOX is implemented with internal pipelining and it is shared between the main round and the key expansion units. Also, the key expansion unit is implemented to work on the fly and in parallel with the main round unit. These designs have achieved higher FPGA (Throughput/Area) efficiency comparing to previous AES designs.

CHAPTER 1 INTRODUCTION

In this chapter the research motivation, research objectives and the thesis organization are presented.

1.1 Motivation

Nowadays cryptography has a main role in embedded systems design. As the number of devices and applications which send and receive data are increasing rapidly, the data transfer rates are becoming higher. In many applications, this data requires a secured connection which is usually achieved by cryptography.

Many cryptographic algorithms were proposed, such as the Data Encryption Standard (DES), the Elliptic Curve Cryptography (ECC), the Advanced Encryption Standard (AES) and other algorithms. Many researchers and hackers are always trying to break these algorithms using brute force and side channel attacks. Some attacks were successful as it was the case for the Data Encryption Standard (DES) in 1993, where the published cryptanalysis attack [22] could break the DES.

The Advanced Encryption Standard (AES) is considered nowadays as one of the strongest published cryptographic algorithms, where it was adopted by the National Institute for Standards and Technology (NIST) after the failing of the Data Encryption Standard (DES). Moreover, it is used in many applications such as in RFID cards, ATM Machines, cell-phones and large servers.

Due to the importance of the AES algorithm and the numerous applications that it has, the main concern of this thesis will be presenting new efficient hardware implementations for this algorithm.

Hardware implementations for the AES algorithm vary according to the application. While some applications require very high throughputs as in e-commerce servers, others require medium throughput range as in designs for cell phones [17]. Some others require very low area implementations to be used in low power application as in RFID cards.

Many hardware designs where suggested for the AES algorithm. Some of these designs targeted high speed applications as in the loop unrolled 128 bits designs [2], [3] and [5], while others targeted medium and low area implementations as in the designs [14], [15] and [17].

As each application requires the AES to have different speed and area, this thesis presents two new hardware implementations for the AES algorithm. The first hardware implementation is a high speed 128 bits AES encryptor with new merging and pipelining techniques, while the second hardware implementation is a medium throughput 32 bits AES design with efficient resources sharing and internal pipelining techniques. Both designs have achieved better efficiencies and performances comparing to previous AES hardware designs.

Field Programmable Gates Arrays (FPGA) is considered as one of the best ways to assess digital system designs. Because of this fact and as most of the previous AES hardware implementations have used FPGA to assess their performances; the presented designs in this thesis have been simulated using FPGA Xilinx devices.

1.2 Research Objective

Based on the previous discussion, the main objectives in the two presented AES designs are:

1. Present new mathematical models for the AES algorithm which reduces the hardware implementations cost.

2. Increasing the systems throughput by parallel processing for the data using pipelining techniques.

3. Reduce the repeated operational blocks in the AES design by merging, relocating and sharing.

1.3 Organization

This thesis is organized as follows:

Chapter 2 will explain the AES algorithm in details. The four encryption/ decryption steps are presented: Shift Rows/Inverse Shift Rows, Byte Substitution/ Inverse Byte Substitution, Mix Column/Inverse Mix Columns and finally Add Round Key.

In Chapter 3, a high speed 128-bits pipelined loop unrolled AES encryptor using new efficient merging technique is presented. A comparison with previous works is also provided.

In Chapter 4, a new 32-bits AES design using S-BOX sharing between the main round unit and the key expansion unit is presented. Comparison using FPGA implementation with previous works is also presented in this chapter.

Finally, the conclusion and the future work are presented in Chapter 5.

CHAPTER 2 ADVANCED ENCRYPTION STANDARD (AES) ALGORITM

This chapter discusses the Advanced Encryption Standard algorithm steps and implementations. Also, a literature review which studies previous proposed hardware designs for the AES algorithm is presented.

2.1 Definition and History of Cryptography

We refer to the word cryptography as the change of data representation from its original form into another different form in order to make it hidden and secured. Cryptography has two processes; the first process is the encryption where the original data is converted into secured form using certain steps. The second process is the decryption, where the encrypted data is restored to the original form by applying the inverse to the steps applied in the encryption process.

Classic Cryptography started thousands of years ago. All over the history classic cryptography was used for secret communications between people. This kind of cryptography is usually applied by substituting the message letters by other letters using certain formula [21], for example substituting each letter in a message with the next letter in the alphabets so that the word “Test” would become “Uftu”.

In modern ages, cryptography development has been a major concern in the fields of mathematics, computer science and engineering. One of the main classes in cryptography today is the symmetric-key cryptography, where a shared key of a certain size will be used for the encryption and decryption processes. Fig. 1 illustrates the concept of symmetric-key cryptography.

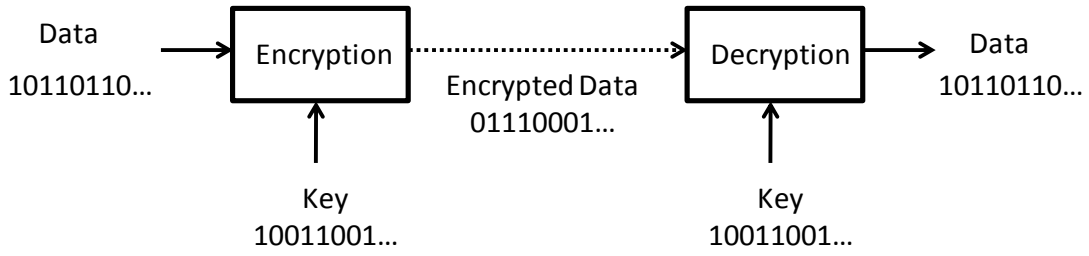


Fig. 1 Symmetric-key cryptography

2.2 Finite Field

In algebra, the field that has a finite number of elements is called finite field or Galois field (GF). Each finite field has a prime integer which represents its characteristic. For example, the finite field GF(p), represents a field with the range of integers {0,1, …., p-1}. The total number of elements in the finite field is called the finite field order. Fields with prime integer orders has characteristic equal to their order.

Some finite fields uses non prime integer orders; in this case the finite field will be represented using the prime number ‘p’ which represent the characteristic along with the power ‘n’. Equation (2.1) shows how to represent the finite field with order ‘k’ and using the prime number ‘p’ and the power ‘n’.

$$k = GF(p^n) \tag{2.1}$$

The finite field in (2.1) has a range of integers that vary between {0,1,…, k-1}.

Finite fields are used in many cryptographic algorithms. The Advanced Encryption Standard uses the finite field GF(2^8), where each data byte represents a value between $(00\text{-}FF)_H$.

Each data byte can be represented as a polynomial over the GF(2^8). Equation (2.2) shows the polynomial representations in GF(2^8)

$$a(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \tag{2.2}$$

Equation (2.2) can be also written as:

$$a(x) = \sum_{i=0}^{7} b_i x^i \tag{2.3}$$

Where $b_i \varepsilon \{0,1\}$.

The next sub-sections will explain the arithmetic in finite fields based on the characteristic p=2.

2.2.1 Addition in Finite Field

Arithmetic in finite field is different than normal algebra arithmetic. In finite field with characteristic of 2, an addition is obtained by applying bit-wise XOR operation between

the operands. Equation (2.5) shows the result of the finite field addition in (2.4).

$$a_3(x) = a_1(x) + a_2(x) = \sum_{i=0}^{7} b_{1i}x^i + \sum_{i=0}^{7} b_{2i}x^i \tag{2.4}$$

$$a_3(x) = \sum_{i=0}^{7} (b_{1i}(XOR)b_{2i})x^i \tag{2.5}$$

Appendix A shows an example for addition in finite field.

2.2.2 Multiplication in Finite Field

In finite field, the multiplication product of two polynomials will be modulo an irreducible polynomial so that the final answer can be within the used finite field. Irreducible polynomial means it cannot be factorized and expressed as a product of two or more polynomials over the same field [26].

Equation (2.6) represents the multiplication operation of the polynomials $a_2(x)$ and $a_1(x)$ using the modulus m(x).

$$a_3(x) = (a_2(x) \times a_1(x)) \, mod \, m(x) \tag{2.6}$$

Appendix A shows a multiplication example using the finite field.

2.3 The Data Encryption Standard (DES)

In the early 1970's, IBM developed the Data Encryption Standard as a symmetric-key cryptography algorithm. This algorithm was adopted by the National Institute of Standard and Technology (NIST) in 1977, where it was published in the Federal Information Processing Standard (FIPS) Publication 46 [20]. The DES consists of 64 bits data block with key size of 56 bits, where 16 encryption rounds will be applied to the data to complete the encryption process.

The DES algorithm starts to fail after several published brute force attacks. The linear cryptanalysis attack [22] could break the DES and made it insecure algorithm. The NIST started to search for another algorithm to replace the DES, where the Rijndael cipher was selected as the new Advanced Encryption Standard (AES).

2.4 The Advanced Encryption Standard (AES)

In 1998 Rijndael cipher developed by the two Belgian cryptographers, John Daemen and Vincent Rijmen was published. This cipher was selected later on by the NIST as the Advanced Encryption Standard to supersede the old Data Encryption Standard. The NIST has published full details of AES under the FIPS publication 197 [1].

The AES according to [1] has a constant block size of 128 bits (16 bytes) with 3 different key sizes of 128 bits, 192 bits and 256 bits, where 10, 12 and 14 encryption rounds will be applied for each key size, respectively. During the encryption and decryption processes, the

16 bytes of data will form a changeable (4*4) array called the state array. During the encryption process, the state array consists initially of the input data, this array will keep changing until reaching the final enciphered data. In the decryption process the state array will start by the enciphered data and will keep changing until retrieving the original data.

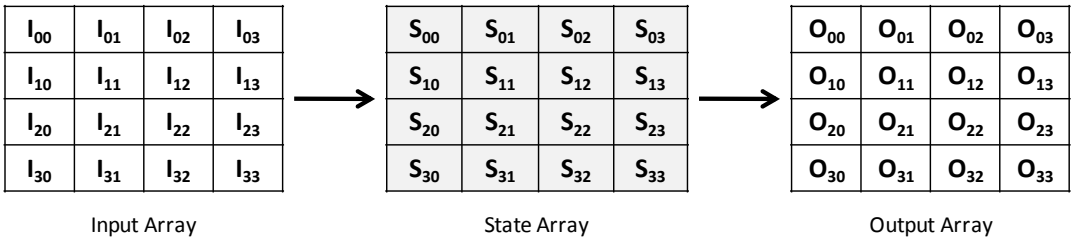


Fig. 2 Input, State and Output arrays

Each encryption round has 4 main steps, Shift Rows, Byte Substitution using the Substitution Box (S-BOX), Mix Columns, and Add Round Key. The decryption process consists of the inverse steps, where each decryption round consists of: Inverse Shift Rows, Byte Substitution using Inverse S-BOX, Add Round Key and Inverse Mix Columns. The round keys will be generated using a unit called the key expansion unit. This unit will be generating 176,208 or 240 bytes of round keys depending on the size of the used key, more details about the key expansion unit will be explained later in this chapter. Fig. 3 Shows the AES encryption and decryption processes.

.

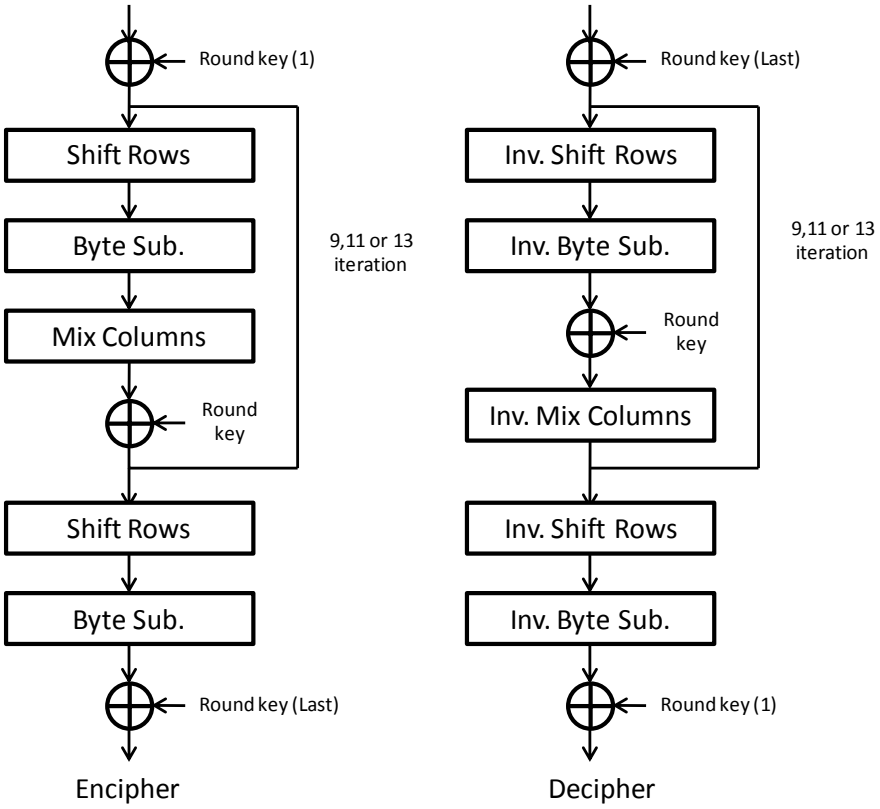


Fig. 3 AES encryption and decryption processes

As can be seen from Fig. 3, the encryption and decryption processes start by adding the round key to the data. This round key is called the initial round key and it consists of the first 16 bytes of round keys in case of encryption and the last 16 bytes in case of decryption. The encryption iteration starts with the Shift Rows step, then the Bytes Substitution is applied, followed by the Mix Columns step, and finally the Round Key is added. In the decryption iteration the Round Key is obtained before the Inverse Mix Columns step. These iterations are repeated 9, 11 and 13 times for the key sizes 128,192 and 256 bits, respectively. The last encryption and decryption iterations exclude the Mix column and Inverse Mix column steps.

This chapter will explain the AES encryption and decryption steps. As most applications and designs use the AES with 128 bits key size, the designs proposed in this thesis are based on this key size. Also, all the used examples and algorithms in this chapter are based on the 128 bits key size. The first sub-section will explain the Shift Rows / Inverse Shift Rows step.

2.4.1 Shift Rows/ Inverse Shift Rows

In Shift Rows step the second, third and fourth row of the state array are shifted one, two and three cyclic shifts to the left, respectively. Most references consider the shift rows step as the first step in the encryption iteration; however it can be done after the Byte Substitution step without affecting the algorithm. Fig. 4 shows how the Shift Rows step is obtained.

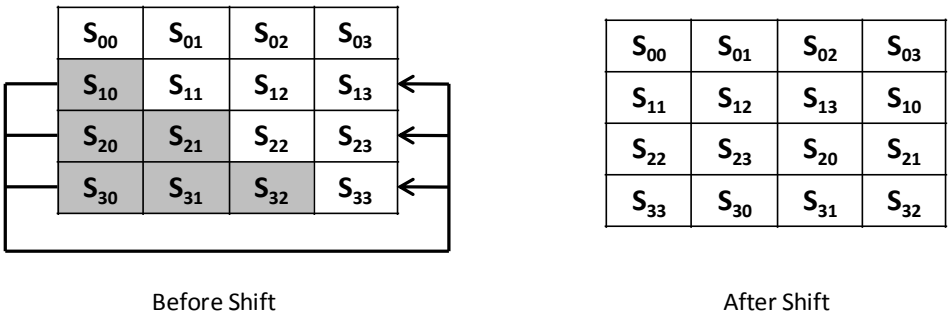


Fig. 4 Shift Rows step

The Inverse Shift Rows step is obtained during the decryption process by shifting, the second, third and fourth rows, one, two and three cyclically shift to the right, respectively. Fig. 5 shows how the Inverse Shift Rows step is obtained.

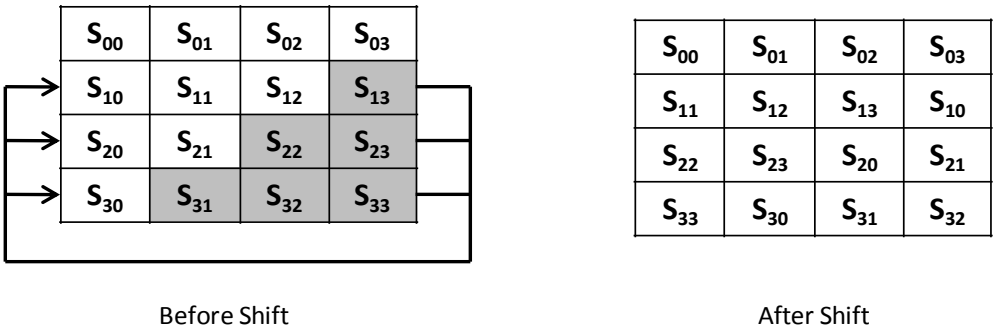


Fig. 5 Inverse Shift Rows step

2.4.2 Byte Substitution and Inverse Byte Substitution Using S-BOX and Inverse S-BOX

Byte substitution and Inverse Byte Substitution are the most complex steps in the encryption and decryption processes. In these steps each byte of the state array will be replaced with its equivalent byte in the S-BOX or the Inverse S-BOX.

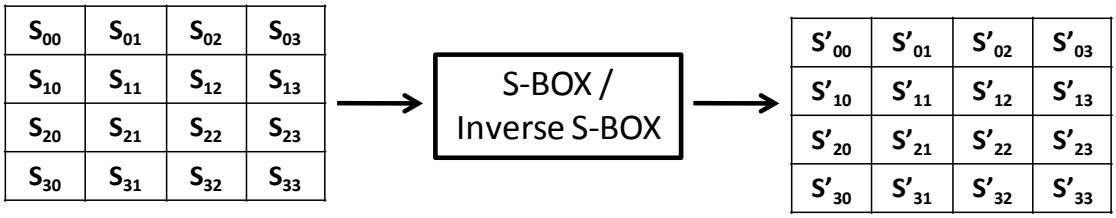


Fig. 6 Byte Substitution

As AES algorithm use elements within the GF(2^8), each element in the state array represents a byte with a value that varies between 00_H-FF_H. The S-BOX has a fixed size of

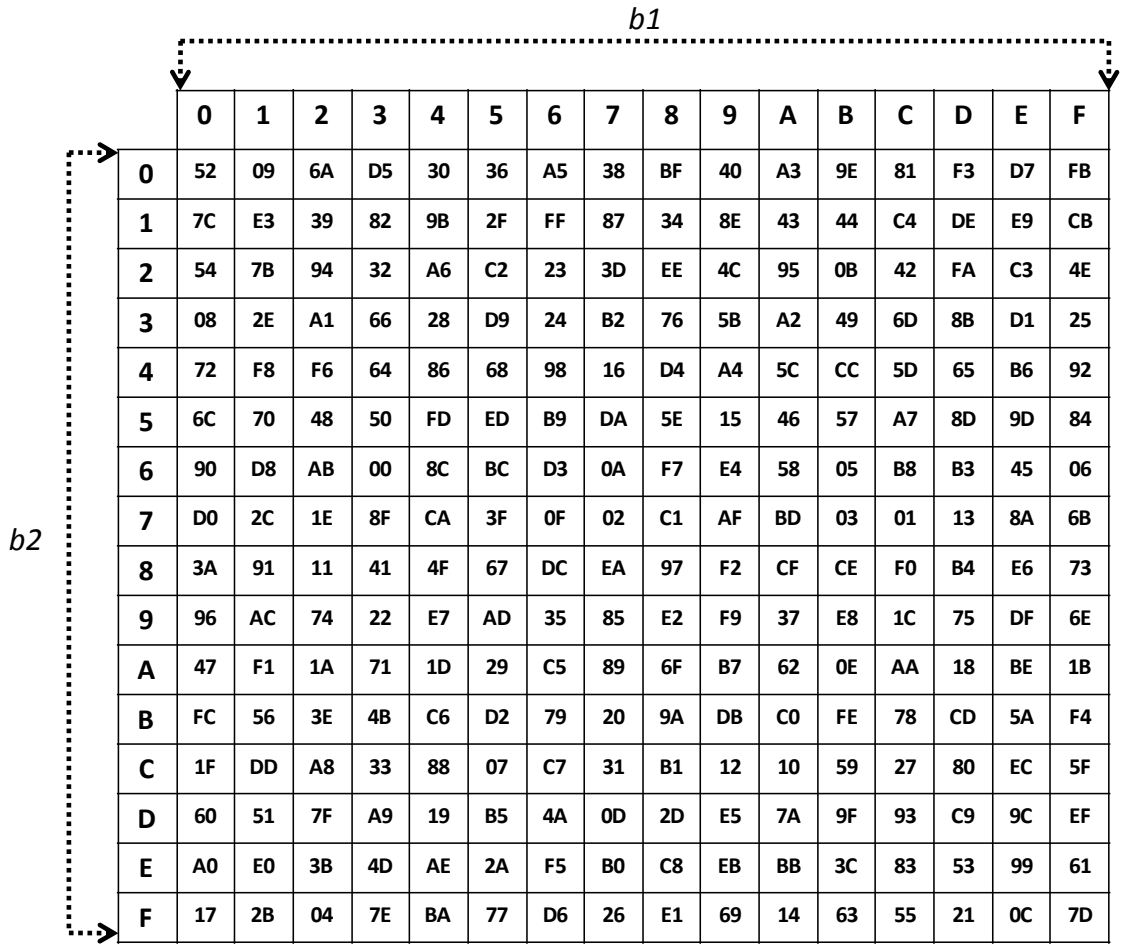
256 bytes represented as (16 * 16) bytes matrix. Fig. 7 shows the AES S-BOX. In this figure the variable ‘b2’ represents the most significant nibble while the variable ‘b1’ represents the least significant nibble.

b1

b2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Fig. 7 The S-BOX

The Inverse S-BOX which is used during the decryption processes will be retrieving the original byte that was substituted using the S-BOX during the encryption process. For example from the S-BOX in Fig. 7 we can see that the S-BOX will substitute the byte ‘00_H’ with the byte ‘63_H’. Also the byte ‘63_H’ in the Inverse S-BOX shown in Fig. 8 will be substituted by ‘00_H’.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Fig. 8 The Inverse S-BOX

The generation of S-BOX is done by two steps, first finding the multiplicative inverse for the numbers 00_H-FF_H in the GF(2^8), then applying the affine transformation on them. On the other hand, the generation of the Inverse S-BOX starts by applying the inverse affine transformation followed by finding the multiplicative inverse. The next sub-sections will explain these sub-steps in more details.

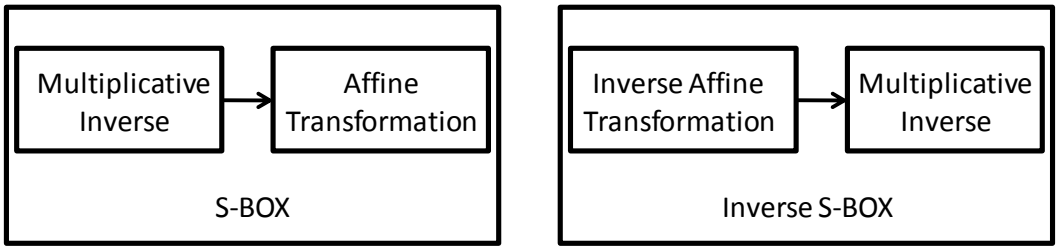


Fig. 9 Generation of S-BOX and Inverse S-BOX

2.4.2.1 Multiplicative Inverse Calculation

The first step of S-BOX generation is finding the multiplicative inverse for the numbers 00_H-FF_H, This requires using the irreducible polynomial p(x) defined in the equation (2.7).

$$p(x) = x^8 + x^4 + x^3 + x + 1 \tag{2.7}$$

Since AES is dealing with numbers within the GF(2^8), it uses the 8^{th} degree irreducible polynomial shown in (2.7) as defined by [1]. This polynomial is used as a reduction polynomial by applying it as a modulus for the multiplication result of two polynomials so that the final result can be within the finite field GF(2^8).

Calculating the multiplicative inverse requires using the Extended Euclidean algorithm [23], which state that for every polynomial a(x) there exists two polynomials b(x) and c(x) such that:

$$a(x).b(x) + p(x).c(x) = 1 \tag{2.8}$$

And since:

$$(a(x).b(x)\,)\ mod\ p(x) = 1 \tag{2.9}$$

We can obtain $a(x)^{-1}$ as:

$$a(x)^{-1} = b(x)\ mod\ (p(x)) \tag{2.10}$$

2.4.2.2 Affine Transformation

The affine transformation is applied after the multiplicative inverse calculation in the Byte Substitution step, while it is applied first in the Inverse Byte Substitution step. The affine transformation and its inverse have two parts, the multiplication part where a constant matrix will be multiplied with the data, then the addition part, where a constant vector is added to multiplication result. The matrix 'A1' and the vector 'C1' are used for the affine transformation as can be seen in (2.11), while the matrix 'A2' and the vector 'C2' are using for the inverse affine transformation as can be seen in (2.12).

$$\mathrm{AT(q)} = \mathrm{A1} * \mathrm{q} + \mathrm{C1} = \begin{bmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{bmatrix} * \begin{bmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{2.11}$$

$$\mathrm{AT(q)^{-1}} = \mathrm{A2} * \mathrm{q} + \mathrm{C2} = \begin{bmatrix} 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \end{bmatrix} * \begin{bmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad (2.12)$$

2.4.3 Mix Columns / Inverse Mix Columns Steps

After performing the Byte Substitution step during the encryption process, Mix Columns step is applied. In the decryption process the Inverse Mix Columns step is applied after adding the Round Key. The Mix Columns step and its inverse are not applied in the last encryption or decryption processes as described in [1]. In these steps each column of the state array will be processed using 4 polynomials. Each polynomial consists of 4 operands representing the old state array column elements and they will be used to obtain the new state array element.

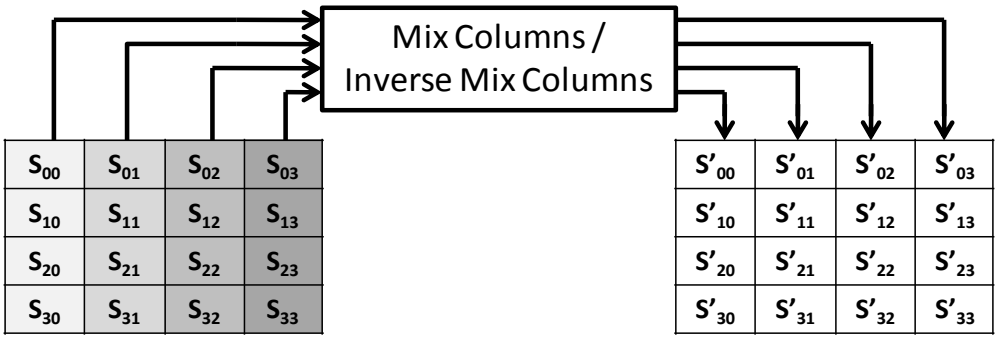


Fig. 10 Mix Columns and Inverse Mix Columns steps

According to [18], the polynomial c(x) given in (2.13) is used to obtain the Mix Column step:

$$c(x) = \{03\}.x^3 + \{01\}.x^2 + \{01\}.x^2 + 02 \tag{2.13}$$

To obtain the Mix Column Step, each 4 bytes state array column is represented as polynomials over GF(2^8) as shown in (2.14). Each polynomial is multiplied by the fix polynomial c(x) modulo the polynomial k(x) described in (2.15).

$$b(x) = S_{3,c}.x^3 + S_{2,c}.x^2 + S_{1,c}.x^2 + S_{0,c} \tag{2.14}$$

$$k(x) = x^4 + 1 \tag{2.15}$$

According to [18], multiplication between the polynomials c(x) and b(x) modulo k(x) will result in the matrix (2.16).

$$\begin{bmatrix} \mathbf{S_{0C}}' \\ \mathbf{S_{1C}}' \\ \mathbf{S_{2C}}' \\ \mathbf{S_{3C}}' \end{bmatrix} = \begin{bmatrix} \mathbf{02} & \mathbf{03} & \mathbf{01} & \mathbf{01} \\ \mathbf{01} & \mathbf{02} & \mathbf{03} & \mathbf{01} \\ \mathbf{01} & \mathbf{01} & \mathbf{02} & \mathbf{03} \\ \mathbf{03} & \mathbf{02} & \mathbf{01} & \mathbf{01} \end{bmatrix} * \begin{bmatrix} \mathbf{S_{0C}} \\ \mathbf{S_{1C}} \\ \mathbf{S_{2C}} \\ \mathbf{S_{3C}} \end{bmatrix} \tag{2.16}$$

Matrix (2.12) can be written in the polynomials (2.17):

$$\begin{aligned} S_{0,c}' &= \{02\}.S_{0,c} + \{03\}.S_{1,c} + S_{2,c} + S_{3,c} \\ S_{1,c}' &= \{02\}.S_{1,c} + \{03\}.S_{2,c} + S_{3,c} + S_{0,c} \\ S_{2,c}' &= \{02\}.S_{2,c} + \{03\}.S_{3,c} + S_{0,c} + S_{1,c} \\ S_{3c}' &= \{02\}.S_{3,c} + \{03\}.S_{0,c} + S_{1,c} + S_{2,c} \end{aligned} \tag{2.17}$$

The Inverse Mix Column step is obtained by multiplying the 4 bytes state array column polynomial b(x) (2.10) by the Inverse Mix Columns polynomial $c(x)^{-1}$ (2.18) Modulo k(x) (2.15).

$$c(x)^{-1} = \{0B\}.x^3 + \{0D\}.x^2 + \{09\}.x + 0E \tag{2.18}$$

The latter multiplication can be represented using the matrix in (2.19):

$$\begin{bmatrix} {S_{0C}}' \\ {S_{1C}}' \\ {S_{2C}}' \\ {S_{3C}}' \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} * \begin{bmatrix} {S_{0C}}' \\ {S_{1C}}' \\ {S_{2C}}' \\ {S_{3C}}' \end{bmatrix} \tag{2.19}$$

Matrix (2.15) can be written using the polynomials (2.20):

$$\begin{aligned} {S_{0,c}}' &= \{0E\}.S_{0,c} + \{0B\}.S_{1,c} + \{0D\}.S_{2,c} + \{09\}.S_{3,c} \\ {S_{1,c}}' &= \{0E\}.S_{1,c} + \{0B\}.S_{2,c} + \{0D\}.S_{3,c} + \{09\}.S_{0,c} \\ {S_{2,c}}' &= \{0E\}.S_{2,c} + \{0B\}.S_{3,c} + \{0D\}.S_{0,c} + \{09\}.S_{1,c} \\ {S_{3c}}' &= \{0E\}.S_{3,c} + \{0B\}.S_{0,c} + \{0D\}.S_{1,c} + \{09\}.S_{2,c} \end{aligned} \tag{2.20}$$

2.4.4 Key Expansion and Add Round Key Step

Add Round Key step is applied one extra time comparing to the other encryption and decryption steps. The first Add Round Key step is applied before starting the encryption and decryption iterations, where in the encryption process the first 128 bits of the input key the whole key in case of using key size of 128 bits are added to the original data block. This

round key is called the initial round key. For the decryption process the initial round key is the last 128 bits of the generated keys as will be explained later.

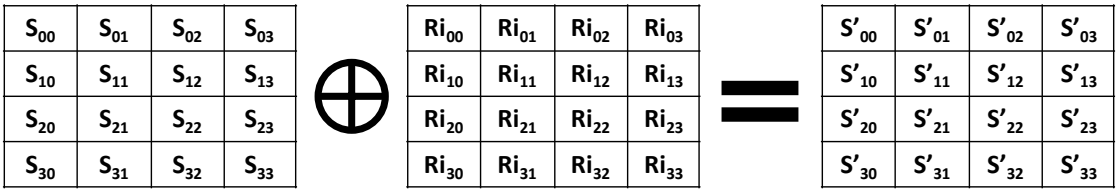


Fig. 11 Add Round Key

In addition to the initial 16 bytes round keys, another 16 bytes of round keys will be required for each encryption or decryptions iterations, this makes the total as 176 bytes, 224 bytes and 240 bytes for the key sizes 128,192 and 256 bits, respectively. These round keys are generated using an operation called the key expansion. In the key expansion all the round keys will be generated from the original input key. The next sub-section explains the round keys generation using the key expansion operation.

2.4.4.1 Key Expansion

The key expansion term is used to describe the operation of generating all Round Keys from the original input key. The initial round key will be the original key in case of encryption and the last group of the generated key expansion keys in case of decryption – the first and last 16 bytes in case of key sizes of 192 and 256 bits. As mentioned previously this initial round key will be added to the input initially before starting the encryption or decryption iterations. Using the 128 bits key size, 10 groups of round keys will be generated with 16 bytes size for each.

The first 4 bytes column in each group will be generated as follows:

1) Taking the S-BOX equivalent to the last column of the previous group (one previous column).
2) Perform one cyclic permutation "rotate elements [R_{0r} R_{1r} R_{2r} R_{3r}] to [R_{1r} R_{2r} R_{3r} R_{0r}].
3) Add the round constant.
4) Add the result to the first column of the previous group (four previous columns).

The remaining second, third and fourth column of each group will be created by adding the direct previous column with the equivalent column in the previous group (four previous columns). This will create a total of 176 bytes of round keys.

In the 192 bits key size, each group consists of six columns. The first column will be created in the same way as in the case of 128 bits key size. The remaining second to sixth column of each group will be also created by adding the direct previous column with the equivalent column in the previous group (six previous columns). In the 192 bits key size, 192 bytes of round keys will be generated in addition to the original 24 bytes key. The round keys will be retaken as groups of four columns each and are applied 13 times during the encryption and decryption processes.

Finally, for the 256 key sizes, each group will consists of 8 columns; the first column in the group will be created exactly the same way as in the 128 and 192 key sizes. The fourth column of each group will be created by applying the byte substitution to the third column values and then adding it to the equivalent column in the previous group (eight previous columns). The remaining columns are created also by adding the direct previous column

with the equivalent column of the previous group. In the 256 bits key size, 208 bytes of generated round keys in addition to the original 32 bytes key are applied 15 times as groups of 16 bytes in the encryption and decryption processes.

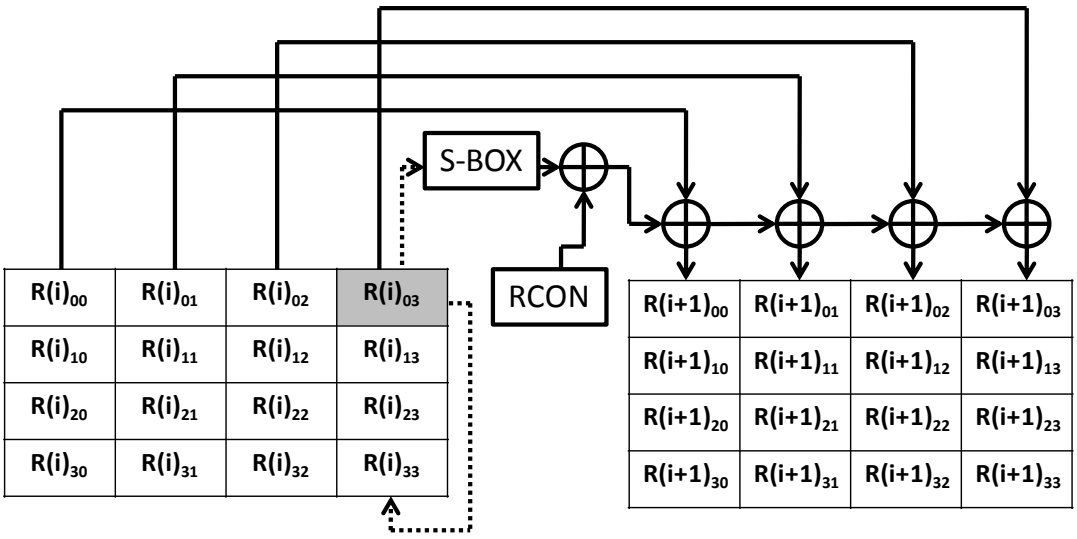


Fig. 12 Key Expansion

The round constant matrix "known as RCON" is a constant matrix used during the key expansion process. Each row of the round constant matrix will be added to the first row of each group during the key generation as explained previously. The first column of this matrix is generated according to (2.21) while the second, third and fourth row are zeros. The standard AES reduction polynomial will be used to keep the elements in the GF(2^8).

$$\begin{aligned} RC[0] &= x^0 = 1 \\ RC[1] &= x \\ RC[j] &= x.RC[j-1] = x^j, \qquad j > 1 \end{aligned} \tag{2.21}$$

Fig. 13 shows the round constant values for the AES with 128 bits key size.

	0	1	2	3
0	1	0	0	0
1	2	0	0	0
2	4	0	0	0
3	8	0	0	0
4	10	0	0	0
5	20	0	0	0
6	40	0	0	0
7	80	0	0	0
8	1B	0	0	0
9	36	0	0	0

Fig. 13 Round Constant

2.5 Literature Review

Since the announcement of the Advanced Encryption Standard algorithm in 2001, various hardware implementations were proposed for it. Most of these implementations have targeted the AES with 128-bits key size. This key size is considered to be sufficient for most of the commercial applications, where using higher key sizes is considered as waste of resources as it requires higher area implementations with longer processing time. Key sizes of 192-bit and 256 bits are used mainly in top secret military applications to ensure the maximum level of security [24].

AES implementations can be divided into three main types depending on data-path width. The first type comes with 8-bits data path as implemented in [17] aiming for low area

architectures. The second type is the 32-bits data path architectures which process each state array row or column together as implemented in [14-16] and [18] targeting a medium throughput applications. The last type of implementations is the 128-bits loop unrolled architectures which targets very high speed applications as presented in [2-3] and [12].

Mainly, designs with 8 bits and 32 bits data paths use looping architectures. Looping architectures use a one stage of AES encryptor/decryptor with a feedback at the end as shown in Fig 14 (a). In this way the data will go through this stage until completing the required number of iterations which is determined according to size of the used key. This AES stage could be only an encryptor or an encryptor with decryptor and it includes the hardware implementation for the four AES steps: Shift Rows Step, Byte Substitution using the Substitution Box (S-BOX), Mix Columns and Add Round Key.

For very high speed applications which is implemented as full 128 bits data path, the throughput can be doubled ideally N times by applying the loop unrolled architecture. In this architecture, replicates of the AES stages are implemented in series, where N number of stages is used. In AES 128 bits key size architecture, N is 10, as 10 AES iterations are required to complete the encryption/decryption processes. Fig 14 (b) shows the loop unrolled architectures with pipelining technique.

In order to get benefit from the loop unrolled architecture, a pipelining stage is implemented at the end of each AES stage which allows entering new data at each clock cycle, therefore, all AES stage will be working in parallel. The design in [25] presents a loop unrolled AES implementation with pipelining techniques.

More advanced pipelining techniques were used in the designs in [2], [5] and [12], where

the idea of using sub-pipelining stages is presented. In sub-pipelining, instead of applying pipelining stage at the end of each AES stage, the latter is divided into certain number of pipelining stages. This method doubles the throughput couple of times compared to what is achievable using normal pipelining. Fig 14 (c) shows the loop unrolled architectures with sub- pipelining techniques.

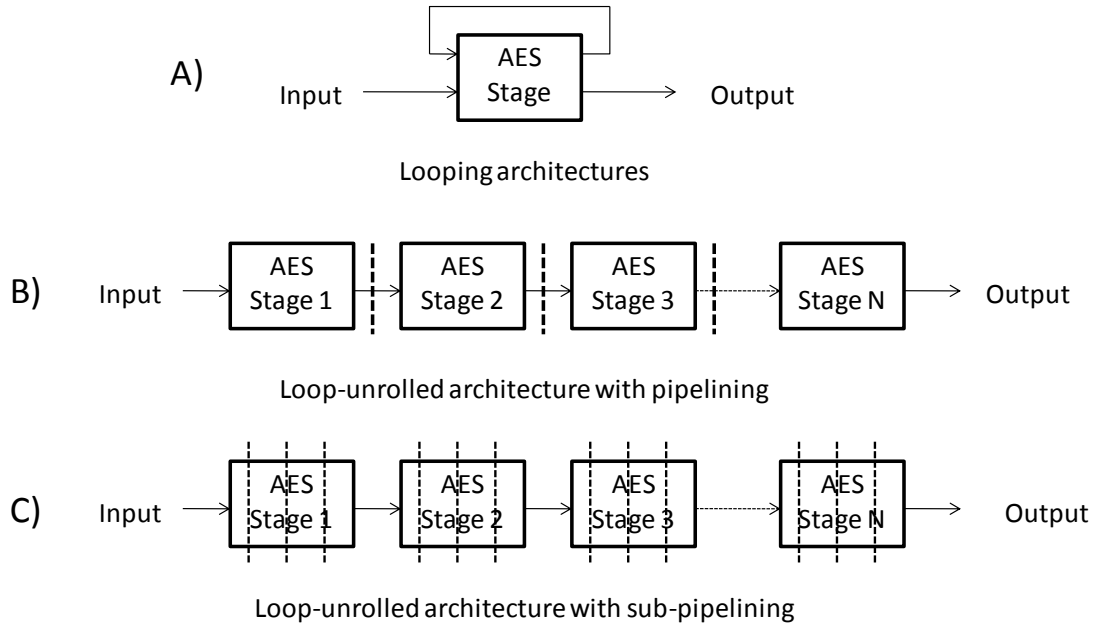


Fig. 14 AES looping and loop-unrolled architectures

These loop unrolled sub-pipelined AES designs which achieves tens of gigabytes of throughput are used in many applications such as highly traffic servers as in e-commerce servers [17].

S-BOX implementation is a main concern in the AES hardware design. Two main methods where proposed for the implementation of the S-BOX. The first method is by pre-storing the S-BOX elements in BRAMs, where BRAM is an FPGA block of RAM which can be used to store data. The design in [3] has used the BRAMs to present high speed loop

unrolled architecture. The second method uses the composite field S-BOX as proposed in [8], and implemented as high speed loop unrolled sub-pipelined AES design in [2].

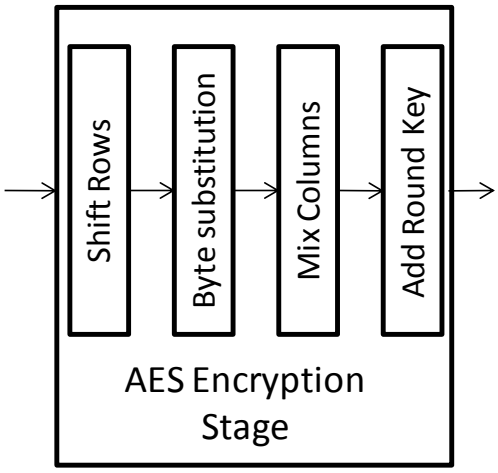


Fig. 15 AES encryption stage

As pipelining cannot be applied to BRAM as it is a one memory block, using it in the implementation of S-BOX will limit the number of sub-pipelining stages in the design. Also implementation using BRAMs usually requites larger area than the composite field arithmetic designs as will be shown later in this thesis.

The next sections will present different proposed designs for the Advanced Encryption Standard algorithm.

2.5.1 Composite Field Arithmetic S-BOX

The implementation of the composite field S-BOX is accomplished using combinational logic circuits rather than using pre-stored S-BOX values. S-BOX substitution starts by finding the multiplicative inverse of the number in GF(2^8), and then applying the affine

transformation. Implementing a circuit to find the multiplicative inverse in the finite field GF(2^8) is very complex and costly, therefore, [4] has suggested using the finite field GF(2^4) to find the multiplicative inverse of elements in the finite field GF(2^8). First detailed implementation for the composite field S-BOX was published in [8].

Each element in a higher order field can be expressed using the polynomial $bx + c$, where b and c are elements in the lower order field. For example any element in GF(2^8) can be expressed using the polynomial $bx + c$, where b and c ε GF(2^4) and they represent the most and the least significant nibbles of that element.

After expressing the GF(2^8) element as a polynomial over GF(2^4), the Multiplicative Inverse can be found using the polynomial shown in (2.22) [4].

$$(\mathrm{bx} + \mathrm{c})^{-1} = \mathrm{b}(\mathrm{b}^2\lambda + \mathrm{c}(\mathrm{b} + \mathrm{c}))^{-1}\mathrm{x} + (\mathrm{c} + \mathrm{b})(\mathrm{b}^2\lambda + \mathrm{c}(\mathrm{b} + \mathrm{c}))^{-1} \quad (2.22)$$

Fig. 16 shows the Composite field S-BOX which was proposed by [8]. This model applies equation (2.22) in finding the multiplicative inverse for GF(2^8) elements.

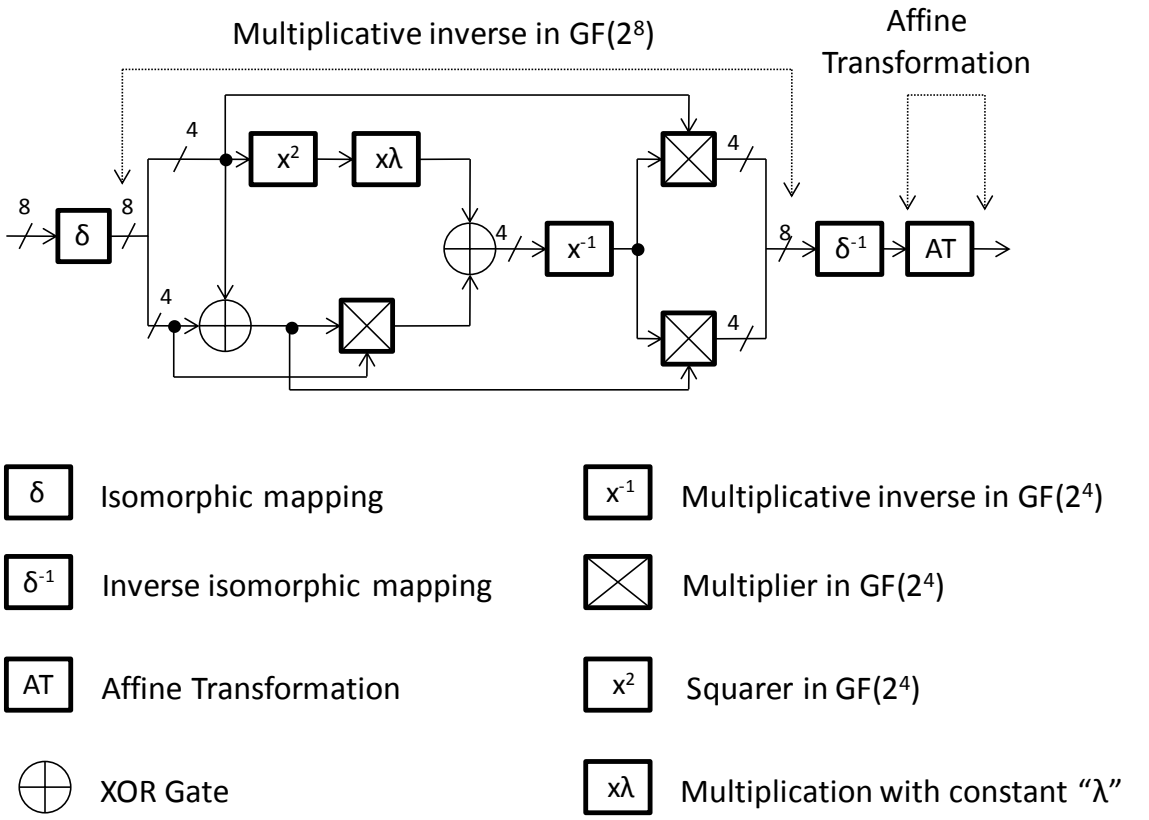


Fig. 16 Composite field S-BOX implementation

As can be seen from the figure, isomorphic mapping must be applied on the $GF(2^8)$ element before applying it as a polynomial over $GF(2^4)$. Also, inverse isomorphic mapping is required after finding the multiplicative inverse for the number.

According to [8] and [26], higher order fields can be built from lower order field using the irreducible polynomials shown in (2.23).

$$\begin{cases} GF(2) \rightarrow GF(2^2) & : x^2 + x + 1 \\ GF(2^2) \rightarrow GF((2^2)^2) & : x^2 + x + \varphi \\ GF((2^2)^2) \rightarrow GF(((2^2)^2)^2) & : x^2 + x + \lambda \end{cases} \quad (2.23)$$

The polynomials $x^2 + x + \lambda$ and $x^2 + x + \varphi$, are used in the implementation of the composite field S-BOX. The constants λ and φ are chosen to ensure the polynomials irreducibility. The used values for these constants according to the proposed in [8] are $\lambda = \{1100\}$ and $\varphi = \{10\}$. These polynomials are mainly used in the derivation of the isomorphic mapping and its inverse in addition to the design of the composite field S-BOX sub-blocks. Detailed explanations on how to use these polynomials in building the composite field S-BOX can be found in [10] and [26].

Chapter 3 will present the internal architecture for each block in the composite field S-BOX, in addition to the used isomorphic and inverse isomorphic matrices.

2.5.2 Loop Unrolled Pipelined Encryptor in [12]

The design in [12] is implemented using a loop unrolled sub-pipelined AES encryptor similarly to the architecture shown in Fig. 14 (c). In this design the two main methods for S-BOX implementation were used. The first one is by using the composite field S-BOX, while the other one is by using pre-stored S-BOX values in BRAMs.

In this design each encryption stage was divided into 4 and 7 pipelining stages. By using the composite field S-BOX design with 7 pipelining stages, a throughput of 21.64 Gbps where achieved using 9446 Slices. This led to an efficiency of 2.3 Mpbs/Slice using Xilinx XC2VP20-7 device. In the BRAM implementation using 4 pipelining stages a throughput of 21.54 Gbits/s was achieved using 84 BRAMs and 5177 Slice. Fig. 17 shows the design for each AES stage in [12].

In the composite field implementation of this design three separate blocks for isomorphic mapping, inverse isomorphic mapping and affine transformation were used. These blocks are merged into one block in the proposed design in chapter 3.

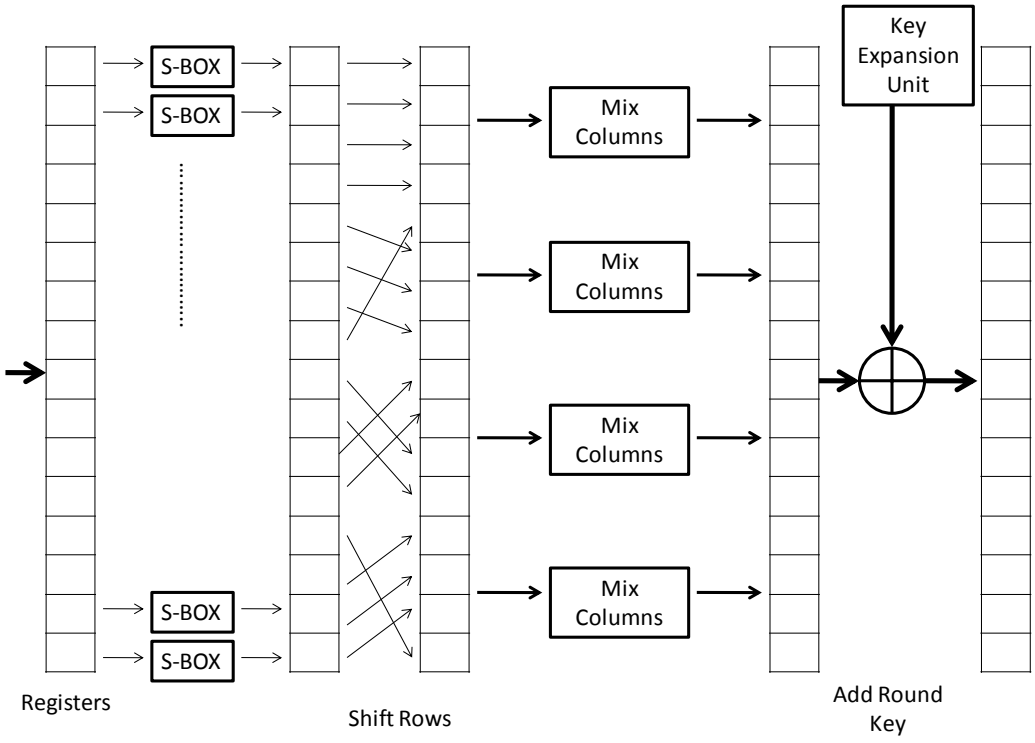


Fig. 17 AES encryption stage in [12]

2.5.3 Very High Speed AES design in [2]

The design presented in [2] is a loop unrolled architecture based on the composite field arithmetic S-BOX proposed in [8]. This design presented a new GF(2^4) inversion block which is used as a part of the GF(2^8) Multiplicative inverse block, in addition to presenting a joint encryptor/ decryptor architecture for loop unrolled designs. In the encryptor design each stage was divided into 3 and 7 sub-pipelining stages. Using 7 sub-pipelining stages, this design was able to achieve an efficiency of 1.956 Mbps/Slice using XCV 1000e-8, where a throughput of 21556 Mpbs was achieved using 11022 Slices.

Fig. 18 shows the proposed encryption stage in [2].

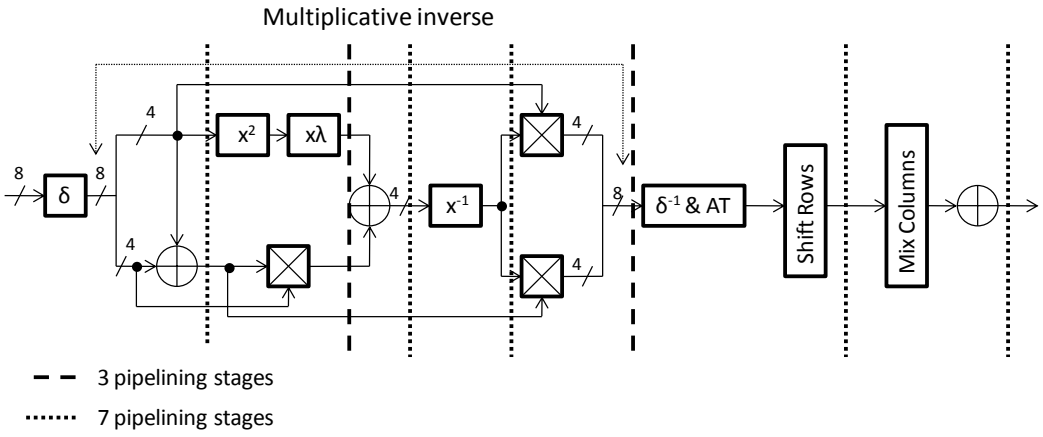


Fig. 18 AES encryption stage in [2]

2.5.4 32 Bits Encryptor/Decryptor Designs

Several 32 bits designs for the AES algorithm were proposed. These designs use the looping architecture which can be seen in Fig. 14 (a). One example is the design in [16], where a 32 bits encrytor/decyptor was proposed. In [16] a block that mixes between the Byte Substitution and Mix Columns steps was implemented. This block pre-stores the multiplication results of the S-BOX bytes and the mix columns coefficients in BRAMs. Applying this is done by pre-storing the multiplication of the S-BOX bytes with the coefficients {2,3} to be used during encryption process and the multiplication results of the Inverse S-BOX bytes with the coefficients {9,B,D,E} to be used during the decryption process. This block required the usage of (512 * 32) bits BRAM. Fig. 19 demonstrates the architecture that was proposed by [16].

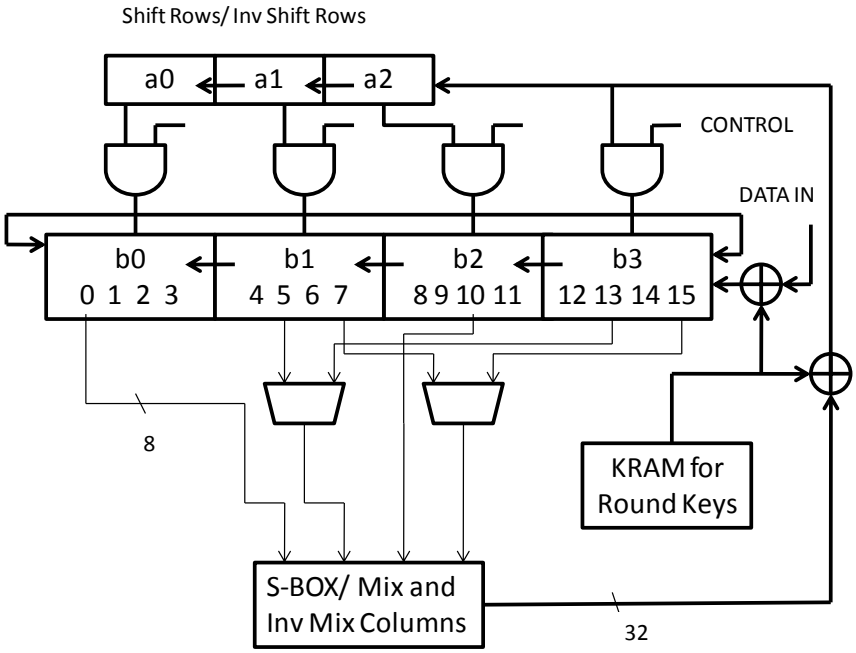


Fig. 19 32-Bits AES design in [16]

As can be seen from the figure, three 32-bits shift registers (a2,a1,a0), and four 32-bits rotate registers (b3,b2,b1,b0), in addition to two 8-bits 2x1 multiplexers are needed to apply the shift columns and inverse shift columns steps. The Byte Substitution and the Mix Columns steps are obtained using the new block that mixes between these steps. All the Round Keys in this design are pre-calculated using a key expansion unit. This unit calculates all round keys prior to the encryption/decryption processes and stores them in the KRAM. This KRAM is a (44 x 32 bits) BRAM.

The method of pre-calculation and storage of all round keys before starting the encryption/decryption processes is used in several 32-bits AES architectures as presented in [14-16]. One example on a key expansion implementation that uses this method is the design presented in [14], where Fig. 20 shows the key expansion unit used in this design.

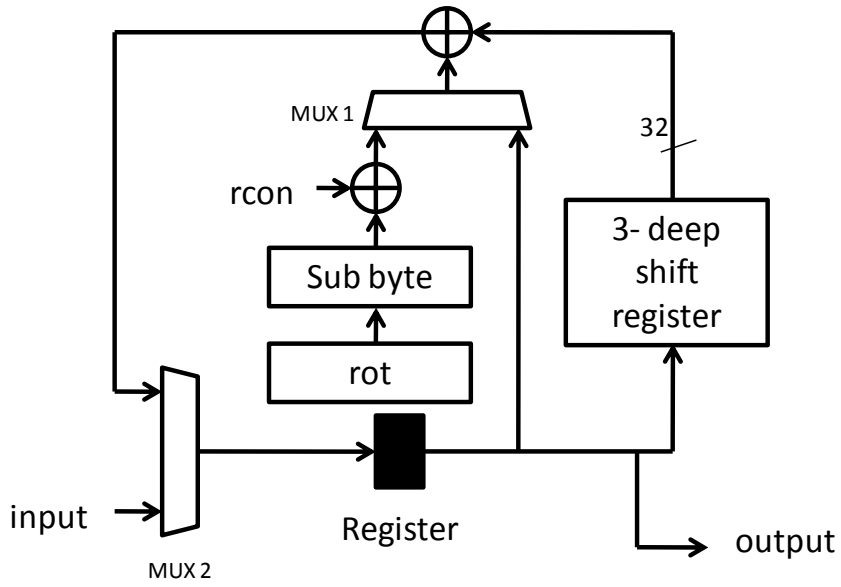


Fig. 20 Key Expansion unit in [14]

As can be seen from Fig. 20, this key expansion unit uses a 3 deep shift registers to store the columns of the previous group, while the first multiplexer will choose the previous columns either directly from the register or after S-BOX substitution and round constant addition. The resulting round keys are stored in BRAM to be used in the encryption/decryption processes later on.

2.6 Conclusion

After reviewing different hardware implementations for the Advanced Encryption Standard algorithm, different techniques and improvements can be applied to improve the efficiency of these designs.

Improvements on high speed loop unrolled designs can be done by applying new methods to decrease the critical path of the AES stage and to allow a lower area implementation. For

Example in the designs [2] and [12] repeated blocks of isomorphic mapping, inverse isomorphic mapping and affine transformation where used at each AES stage. Chapter 3 introduces a new loop unrolled design which merges these 3 blocks into one new block which allows the implementation with shorter critical path and lower area, therefore presenting a system with higher efficiency.

Improvements on 32-bits AES design can be done by applying new methods which reduces the hardware cost and prevents pre-processing delays. The designs in [15] and [16] have used a block that mixes between the Byte Substitution/Inv Byte Substitution and the Mix Column/Inverse Mix Columns Steps. Using this block requires a large memory which reduces the system efficiency. Also, the designs in [14-16] used the method of pre-calculation and storage of all round keys prior to starting the encryption/decryption processes. Using this method requires a pre-processing delay which will be required at the system start and at each key change. Chapter 4 introduces a new design that uses on the fly calculation for the round keys, which means they are calculated in parallel to the encryption/decryption processes. Also, this design uses a shared S-BOX between the key expansion and the round unit. This S-BOX has internal pipelining stages which prevent any extra delays resulting from the S-BOX sharing.

CHAPTER 3 HIGH SPEED AES ENCRYPTOR

In this chapter a new architecture for a high speed AES encryptor using 128-bits key size is presented [13]. This architecture is implemented using loop unrolled technique with sub-pipelining. To apply the sub-pipelining, composite field S-BOX is used which is implemented using combinational logic circuits. In this proposed implementation re-arrangements for the AES encryption sub-steps is applied to allow an efficient merging between them leading to lower area implementation with shorter path length. This new architecture has shown higher efficiency in terms of FPGA (Throughput/Area) comparing to previous loop unrolled pipelined AES encryptors.

3.1 Introduction

The literature review in chapter 2 presented some loop unrolled AES encryptor designs that used the composite field S-BOX in the implementation as in [2] and [12]. This Chapter proposes an encryptor that simplifies the implementation of the composite field S-BOX by relocating the Mix Column step and merge between the inverse isomorphic mapping, the affine transformation multiplication, and the isomorphic mapping of the next encryption stage. This merging combines these three operational blocks into one block. Moreover, the affine transformation vector "C" shown in (2.11) is implemented in the key expansion unit instead of the main round unit. These improvements enabled the implementation to have higher efficiency by reducing the area and shortening the total path length, where a less

number of sub-pipelining stages will be required for achieving certain throughput.

3.2 Implementation of Composite Field Arithmetic S-BOX

Chapter 2 presented the implementation of the composite field S-BOX as it was proposed in [8]. Using the composite field S-BOX requires applying isomorphic mapping and its inverse at the start and the end of the Multiplicative Inverse block. Matrices (3.1) and (3.2) represent the isomorphic mapping and its inverse based on the values $\lambda = \{1100\}$ and $\varphi = \{10\}$. Derivation for the isomorphic mapping matrix can be obtained using the algorithms presented in [7] and [26].

$$\delta * \mathrm{q} = \begin{bmatrix} 1&0&1&0&0&0&0&0 \\ 1&1&0&1&1&1&1&0 \\ 1&0&1&0&1&1&0&0 \\ 1&0&1&0&1&1&1&0 \\ 1&1&0&0&0&1&1&0 \\ 1&0&0&1&1&1&1&0 \\ 0&1&0&1&0&0&1&0 \\ 0&1&0&0&0&0&1&1 \end{bmatrix} * \begin{bmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} \quad (3.1) \qquad \delta^{-1} * \mathrm{q} = \begin{bmatrix} 1&1&1&0&0&0&1&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&1&0&0&0&1&0 \\ 0&1&1&1&0&1&1&0 \\ 0&0&1&1&1&1&1&0 \\ 1&0&0&1&1&1&1&0 \\ 0&0&1&1&0&0&0&0 \\ 0&1&1&1&0&1&0&1 \end{bmatrix} * \begin{bmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} \quad (3.2)$$

Fig. 21 represents the pre-calculated multiplicative inverse results for the numbers in GF(2^8) starting from 00_H till FF_H, b2 and b1 represent the most significant and least significant nibbles , respectively.

b1

b2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

Fig. 21 Multiplicative Inverse results for the numbers in GF(2^8)

The next sub-sections demonstrate the implementation of each sub-block in the composite field S-BOX.

3.2.1 Square Block

According to (2.22) and as can be seen from Fig. 16, square operation is required over GF(2^4) as a part of the multiplicative inverse calculation. This square operation will be applied for the higher nibble of the input byte.

Equation (3.3) shows the implementation of this square block. Derivation of (3.3) can be found in [10].

$$\begin{aligned} q3' &= q3 \\ q2' &= q3 + q2 \\ q1' &= q2 + q1 \\ q0' &= q3 + q1 + q0 \end{aligned} \qquad (3.3)$$

Fig. 22 shows the square results for all the values in GF(2^4), starting from 0_H to F_H.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x^2	0	1	3	2	6	7	5	4	D	C	E	F	B	A	8	9

Fig. 22 Square block input/output

3.2.2 Multiplication with the constant λ

Following the square operation, the higher nibble of the data will be multiplied by the constant λ. This constant is chosen to be equal {1100} to guarantee reduction polynomial irreducibility. Equation (3.4) shows the implementation for the λ multiplication block. Full derivation of this block can also be found in [10].

$$\begin{aligned} q3' &= q2 + q0 \\ q2' &= q3 + q2 + q1 + q0 \\ q1' &= q3 \\ q0' &= q2 \end{aligned} \qquad (3.4)$$

Fig. 23 shows the multiplication with λ results for all the values in GF(2^4), starting from 0_H to F_H.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
xλ	0	C	4	8	D	1	9	5	6	A	2	E	B	7	F	3

Fig. 23 Multiplication with λ block input/output

3.2.3 Multiplicative Inverse in GF(2^4)

Several architectures were suggested for the implementation of GF (2^4) Multiplicative Inverse block. In [2] a comparison between these different implementations is presented. The first design was build using two square blocks with three GF(2^2) multipliers, while the second one used similar block to what is proposed in Fig. 16, but using GF(2^2) in finding the multiplicative inverse for GF(2^4). The third design which is considered the best in terms of gates number and critical path is designed by direct mapping between the input and output bits. Equation (3.5) shows the implementation GF(2^4) Multiplicative Inverse block.

$$
\begin{aligned}
q3^{-1} &= q3 + q3.q2.q1 + q3.q0 + q2 \\
q2^{-1} &= q3.q2.q1 + q3.q2.q0 + q3.q0 + q2 + q2.q1 \\
q1^{-1} &= q3 + q3.q2.q1 + q3.q1.q0 + q2 + q2.q0 + q1 \\
q0^{-1} &= q3.q2.q1 + q3.q2.q0 + q3.q1 + q3.q1.q0 + q3.q0 + q2 + q2.q1 \\
&\quad + q2.q1.q0 + q1 + q0
\end{aligned}
\tag{3.5}
$$

Fig. 24 shows the inversion results for all the values in GF(2^4), starting from 0_H to F_H.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x^{-1}	0	1	3	2	F	C	9	B	A	6	8	7	5	E	D	4

Fig. 24 GF(2^4) Multiplicative inverse block input/output

3.2.4 Multiplication in GF(2^4) and GF(2^2)

As can be seen from Fig. 16 and according to (2.22), three multipliers in GF(2^4) are required as a part of finding the multiplicative inverse in GF(2^8). Fig. 25 shows the GF(2^4) multiplier circuit. As can be seen from the figure the GF(2^4) multipliers consist of 3 GF(2^2) multipliers with 4 XOR Gates and with constant multiplier θ. This constant multiplier which has 2 bits input extracts the lower bit output as the higher bit input, while the higher output bit will be the result of XOR operation between the 2 input bits. Full derivation of this multiplier circuit can be found in [2] and [10].

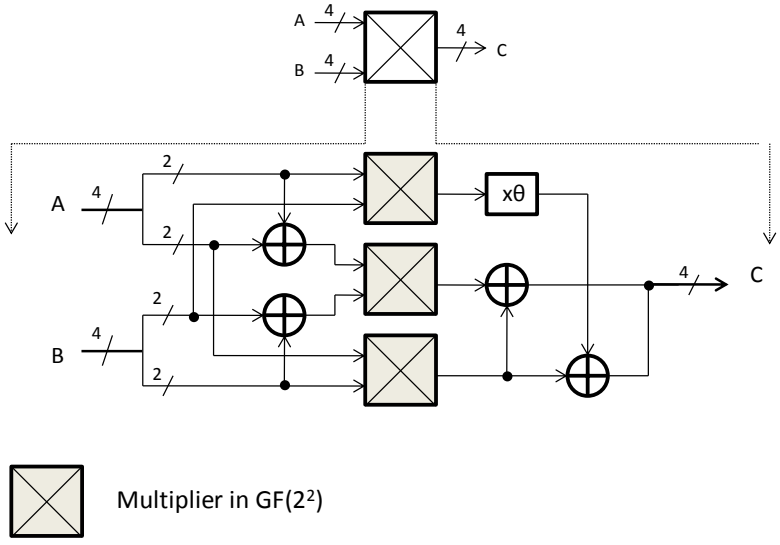


Fig. 25 GF(2^4) Multiplier

The next figure shows the multiplication results in GF(2^4) for the nibble inputs ‘a’ and ‘b’. These inputs vary between 0_H to F_H.

a

b	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	3	1	8	A	B	9	C	E	F	D	4	6	7	5
3	0	3	1	2	C	F	D	E	4	7	5	6	8	B	9	A
4	0	4	8	C	6	2	E	A	B	F	3	7	D	9	5	1
5	0	5	A	F	2	7	8	D	3	6	9	C	1	4	B	E
6	0	6	B	D	E	8	5	3	7	1	C	A	9	F	2	4
7	0	7	9	E	A	D	3	4	F	8	6	1	5	2	C	B
8	0	8	C	4	B	3	7	F	D	5	1	9	6	E	A	2
9	0	9	E	7	F	6	1	8	5	C	B	2	A	3	4	D
A	0	A	F	5	3	9	C	6	1	B	E	4	2	8	D	7
B	0	B	D	6	7	C	A	1	9	2	4	F	E	5	3	8
C	0	C	4	8	D	1	9	5	6	A	2	E	B	7	F	3
D	0	D	6	B	9	4	F	2	E	3	8	5	7	A	1	C
E	0	E	7	9	5	B	2	C	A	4	D	3	F	1	8	6
F	0	F	5	A	1	E	4	B	2	D	7	8	3	C	6	9

Fig. 26 GF(2^4) Multiplication results

Fig. 27 shows the implementation of the GF(2^2) Multiplier.

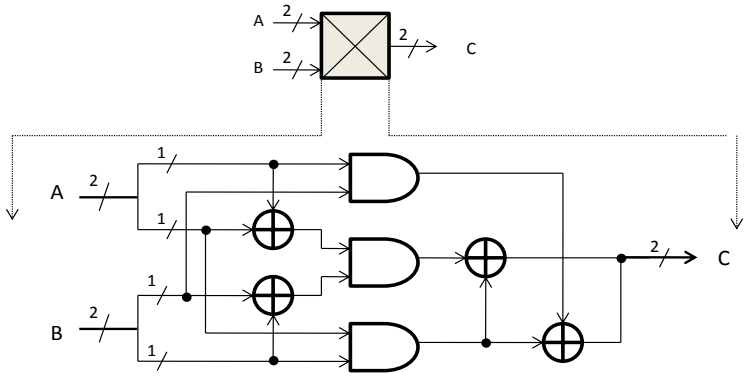


Fig. 27 GF(2^2) Multiplier

Fig. 28 shows all the possible multiplication results in GF(2^2) by varying the inputs a and b between 0_H to 3_H.

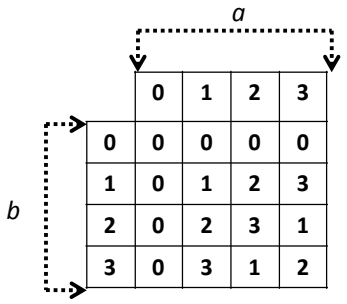


Fig. 28 GF(2^2) Multiplier outputs

3.3 I-BOX

Applying the current composite field design in a loop unrolled system resulted in repeated isomorphic mapping , inverse isomorphic mapping and affine transformation operations in each encryption stage as happened in [2] and [12].

This section presents the new proposed composite field loop unrolled design for the AES encryptor. In this design the mix column step is relocated and performed before the inverse isomorphic mapping and the affine transformation operations which are required in the byte substitution step, while the round keys are mapped with the isomorphic mapping. This relocation for the mix column and the mapping for the round keys allowed an efficient merging between the inverse isomorphic mapping, the affine transformation multiplication and the required isomorphic mapping in the next encryption stage. By this merging these 3 operations where substituted by a new operational block called the "ζ" transformation.

Moreover, the affine transformation addition where placed in the key expansion unit instead of the encryption stages. This allowed the reduction of these addition blocks from 160 blocks in all encryption stages to only 16 blocks implemented in the key expansion unit.

With these merging techniques, a new block which is called the I-BOX "Integrated-BOX" is introduced. This block will be responsible for obtaining one encryption iteration for each state matrix column. To derive the I-BOX equations and to explain the mathematical theory behind this proposed merging and rearrangements, we start by (3.6) which represent the state array element after applying the inverse isomorphic mapping and the affine transformation to the multiplicative inverse result in (2.22). Parameters "r" and "c" represents the row and column locations for the state matrix element, where it is assumed to be at the i'th encryption stage.

$$S_{r,c}'(i) \;=\; AT(\delta^{-1}(bx+c)^{-1}{}_{r,c}) \tag{3.6}$$

After byte substitution is performed using (3.6), the mix column step is applied on the state array elements using the matrix in (3.7):

$$\begin{bmatrix} S_{0C}'' \\ S_{1C}'' \\ S_{2C}'' \\ S_{3C}'' \end{bmatrix} = \begin{bmatrix} 2\ 3\ 1\ 1 \\ 1\ 2\ 3\ 1 \\ 1\ 1\ 2\ 3 \\ 3\ 2\ 1\ 1 \end{bmatrix} * \begin{bmatrix} S_{0C}' \\ S_{1C}' \\ S_{2C}' \\ S_{3C}' \end{bmatrix} \tag{3.7}$$

Where the state array element after the mix columns step can be written as:

$$S_{r,c}{}''(i) = \{02\}.S_{r,c}(i)' + \{03\}_{16}.S_{mod(r+1,4),c}\,(i)' + S_{mod(r+2,4),c}\,(i)' + \quad S_{mod(r+3,4),c}(i)' \tag{3.8}$$

By rewriting (3.10) and adding the Round Key, the state array element will be written as:

$$S_{r,c}{}'''(i) = \{02\}.\left(S_{r,c}' + S_{mod(r+1,4),c}{}'\right) + S_{mod(r+2,4),c}{}' + S_{mod(r+3,4),c}{}' + \quad S_{mod(r+1,4),c}{}' + RK_{r,c} \tag{3.9}$$

By letting $\beta = (\mathrm{bx} + \mathrm{c})^{-1}$ and $S_{r,'c'}(i+1) = \delta\left(S_{r,c}(i)''\right)$ we obtain:

$$S_{r,'c'}(i+1) = \delta\Big(\{02\}.\Big(A\left(\delta^{-1}(\beta)_{r,c}\right) + A\left(\delta^{-1}(\beta)_{mod(r+1,4),c}\right)\Big) + A\left(\delta^{-1}(\beta)_{mod(r+2,4),c}\right) + A\left(\delta^{-1}(\beta)_{mod(r+3,4),c}\right) + A\left(\delta^{-1}(\beta)_{mod(r+1,4),c}\right) + C + RK_{r,c}\Big)\Big) \tag{3.10}$$

$S_{r,'c'}(i+1)$ in (3.10) represents the isomorphic mapped value of state element after shifting and before applying it to the byte substitution in (i+1) stage, where the affine transformation vector "C" in each operand is cancelled by XOR addition with the vector "C" in the next operand. By rearranging the location of the mix columns coefficient {02} matrix, and replacing it with a new matrix {02}', and by taking the affine transformation and the inverse isomorphic mapping as common factors (3.10) can be rewritten as:

$$S_{r,'c'}(i+1) = \delta\Big(A\big(\delta^{-1}\big(\{02\}'.\big((\beta)_{r,c} + (\beta)_{mod(r+1,4),c}\big) + (\beta)_{mod(r+2,4),c} + (\beta)_{mod(r+3,4),c} + (\beta)_{mod(r+1,4),c}\big)\big)\Big) + \delta(RK_{r,c} + C) \tag{3.11}$$

From (3.11) we can define the new transformation ζ as:

$$\zeta = (\delta) * (A) * (\delta^{-1}) \tag{3.12}$$

$$\zeta = \begin{bmatrix} 0&0&0&0&1&0&0&0\\ 0&0&0&1&0&1&0&0\\ 0&1&1&1&0&0&1&0\\ 1&1&1&1&0&0&1&1\\ 1&0&0&0&0&0&0&0\\ 1&1&1&0&0&1&0&0\\ 1&1&1&0&0&0&1&0\\ 1&0&1&1&0&1&1&0 \end{bmatrix} = \begin{bmatrix} 1&0&1&0&0&0&0&0\\ 1&1&0&1&1&1&1&0\\ 1&0&1&0&1&1&0&0\\ 1&0&1&0&1&1&1&0\\ 1&1&0&0&0&1&1&0\\ 1&0&0&1&1&1&1&0\\ 0&1&0&1&0&0&1&0\\ 0&1&0&0&0&0&1&1 \end{bmatrix} * \begin{bmatrix} 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1\\ 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1 \end{bmatrix} * \begin{bmatrix} 1&1&1&0&0&0&1&0\\ 0&1&0&0&0&1&0&0\\ 0&1&1&0&0&0&1&0\\ 0&1&1&1&0&1&1&0\\ 0&0&1&1&1&1&1&0\\ 1&0&0&1&1&1&1&0\\ 0&0&1&1&0&0&0&0\\ 0&1&1&1&0&1&0&1 \end{bmatrix}$$

$$\zeta * \mathrm{q} = \begin{bmatrix} 0&0&0&0&1&0&0&0\\ 0&0&0&1&0&1&0&0\\ 0&1&1&1&0&0&1&0\\ 1&1&1&1&0&0&1&1\\ 1&0&0&0&0&0&0&0\\ 1&1&1&0&0&1&0&0\\ 1&1&1&0&0&0&1&0\\ 1&0&1&1&0&1&1&0 \end{bmatrix} * \begin{bmatrix} q_7\\ q_6\\ q_5\\ q_4\\ q_3\\ q_2\\ q_1\\ q_0 \end{bmatrix} \tag{3.13}$$

Fig. 29 shows ζ transformation for the values 00_H-FF_H, where 'b1' represents the least significant nibble, while 'b2' represents the most significant nibble.

b1

b2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	33	23	45	55	76	66	80	90	B3	A3	C5	D5	F6	E6
1	71	61	42	52	34	24	07	17	F1	E1	C2	D2	B4	A4	87	97
2	37	27	04	14	72	62	41	51	B7	A7	84	94	F2	E2	C1	D1
3	46	56	75	65	03	13	30	20	C6	D6	F5	E5	83	93	B0	A0
4	36	26	05	15	73	63	40	50	B6	A6	85	95	F3	E3	C0	D0
5	47	57	74	64	02	12	31	21	C7	D7	F4	E4	82	92	B1	A1
6	01	11	32	22	44	54	77	67	81	91	B2	A2	C4	D4	F7	E7
7	70	60	43	53	35	25	06	16	F0	E0	C3	D3	B5	A5	86	96
8	1F	0F	2C	3C	5A	4A	69	79	9F	8F	AC	BC	DA	CA	E9	F9
9	6E	7E	5D	4D	2B	3B	18	08	EE	FE	DD	CD	AB	BB	98	88
A	28	38	1B	0B	6D	7D	5E	4E	A8	B8	9B	8B	ED	FD	DE	CE
B	59	49	6A	7A	1C	0C	2F	3F	D9	C9	EA	FA	9C	8C	AF	BF
C	29	39	1A	0A	6C	7C	5F	4F	A9	B9	9A	8A	EC	FC	DF	CF
D	58	48	6B	7B	1D	0D	2E	3E	D8	C8	EB	FB	9D	8D	AE	BE
E	1E	0E	2D	3D	5B	4B	68	78	9E	8E	AD	BD	DB	CB	E8	F8
F	6F	7F	5C	4C	2A	3A	19	09	EF	FF	DC	CC	AA	BA	99	89

ζ-Transform

Fig. 29 ζ-Transformation

Due to changing Mix Columns Step location, the new {02}' multiplication matrix is required because the multiplication is obtained before the inverse isomorphic mapping and the affine transformation. This matrix is derived by implementing a relation between the inputs 00_H-FF_H and the values which will yield to the {02} multiplication after applying the inverse isomorphic mapping and the affine transformation .Equation (3.14) represents the multiplication of input byte 'q' by the new matrix {02}'.

$$\{02\}' * \mathrm{q} = \begin{bmatrix} 1\,0\,1\,1\,1\,1\,1\,0 \\ 1\,1\,0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,1\,0\,0\,0\,0 \\ 1\,1\,1\,0\,1\,0\,0\,1 \\ 0\,1\,1\,0\,0\,0\,1\,1 \\ 0\,0\,1\,0\,1\,1\,0\,1 \\ 0\,0\,1\,0\,1\,0\,0\,1 \\ 1\,1\,1\,1\,0\,1\,0\,1 \end{bmatrix} * \begin{bmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} \tag{3.14}$$

Using (3.12) we can rewrite (3.11) as:

$$S_{r,'c'}(i+1) = \zeta \begin{pmatrix} \{02\}'_{16} \cdot \left((\beta)_{r,c} + (\beta)_{mod(r+1,4),c}\right) + \\ (\beta)_{mod(r+3,4),c} + (\beta)_{mod(r+1,4),c} \end{pmatrix} + \left(\delta(RK_{r,c}) + \delta(C)\right) \tag{3.15}$$

Equation (3.15) represents one encryption stage calculation for state array element in the I-BOX, where each I-BOX will be calculating 4 state array elements. Fig. 30 shows the state array element calculation inside the I-BOX based on (3.15). As can be seen from the figure each state element calculator in I-BOX consists of the multiplicative inverse block GF(2^8), followed by the mix column step which uses the new {02}' multiplier. After the mix columns part the new "ζ" transformation block is used, this block as explained previously resulted from the merging between the inverse isomorphic mapping, the affine transformation multiplication and the isomorphic mapping for the next stage. Finally the Add Round Key XOR gate can be found.

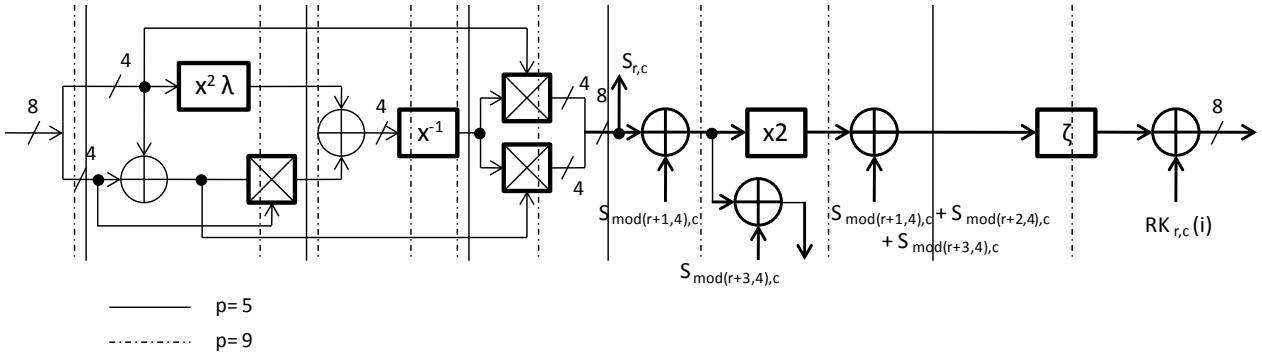


Fig. 30 I-BOX state array element calculation

By placing the mix column step before the inverse isomorphic mapping and the affine transformation and obtaining the Round Keys in the isomorphic mapping the three transformation blocks were placed to be in sequence, therefore, the merging is among them. Fig. 31 depicts the implementation of this merging and rearrangement. Also the affine transformation vector δ(C), has been added to each round key prior to adding it to the state array element, where the vector δ(C) is implemented in the key expansion unit. In this case it will be implemented 16 times "one block for each byte" instead of 160 times in the main round units.

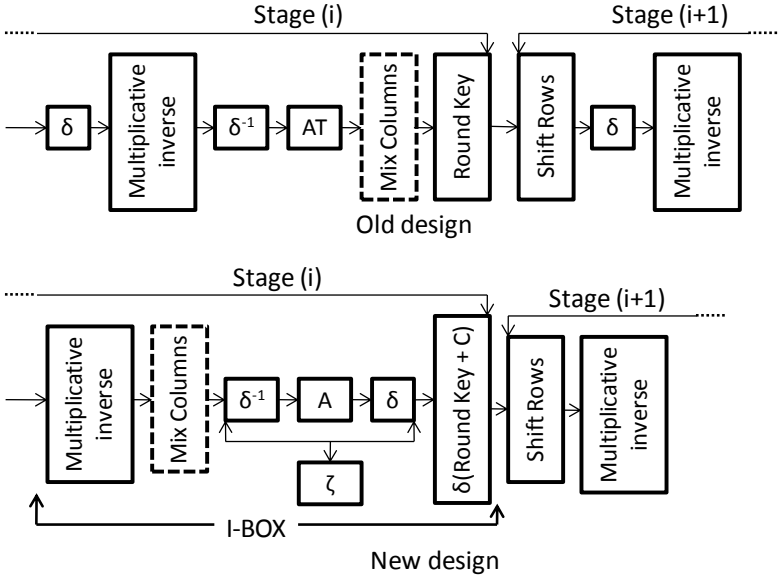


Fig. 31 Rearrangements and merging steps

Table 1 demonstrates the number of gates and the critical path for the isomorphic mapping, inverse isomorphic mapping and the affine transformation compared to the new ζ transformation.

Table 1: Number of gates and critical path for the mappings and the transformations

Operation	δ	δ^{-1}	AT	ζ
Total Number of Gates	12	14	20	15
Critical Path	4	3	4	3

Another simplification is presented in the I-BOX by merging between the λ Multiplier and the Square block. Equation (3.16) presents the merged squaring block with λ Multiplication.

$$\begin{aligned} q3'' &= q2' + q0' = q2 + q1 + q0 \\ q2' &= q3' + q2' + q1' + q0' = q3 + q0 \\ q1'' &= q3' = q3 \\ q0'' &= q2' = q3 + q2 \end{aligned} \tag{3.16}$$

Fig. 32 shows the outputs for the merged Squaring block with λ multiplication input/output for the values 0_H-F_H.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$x^2\lambda$	0	C	8	4	9	5	1	D	7	B	F	3	E	2	6	A

Fig. 32 Merged squaring block with λ multiplication input/output

3.4 Key Expansion Unit

The key expansion unit is responsible of generating the 176 bytes of round keys required during the encryption process. Fig. 33 shows the proposed key expansion unit. This unit will be using 4 S-BOXES in processing the first column of each group of the round keys, while the XOR gates will be used in processing all the columns by adding the direct previous column with the equivalent column from the previous group. The 4 registers following the multiplexers will be used to store the direct previous group of Round Keys to be used in generation for the ongoing group, while the registers (R1-R11) will be storing the all the 176 bytes of round keys.

To work in compatible with the I-BOX state array element calculation described in (3.15). The original input key will be transformed to the isomorphic mapping so that all generated round keys will also be in the isomorphic transformation as required in (3.15). The S-BOX in the key expansion unit is implemented using multiplicative inverse block with the "ζ" transformation block. The affine transformation constant δ(C) are implemented in the key expansion unit, where it is added to the Round Keys before being stored in the registers.

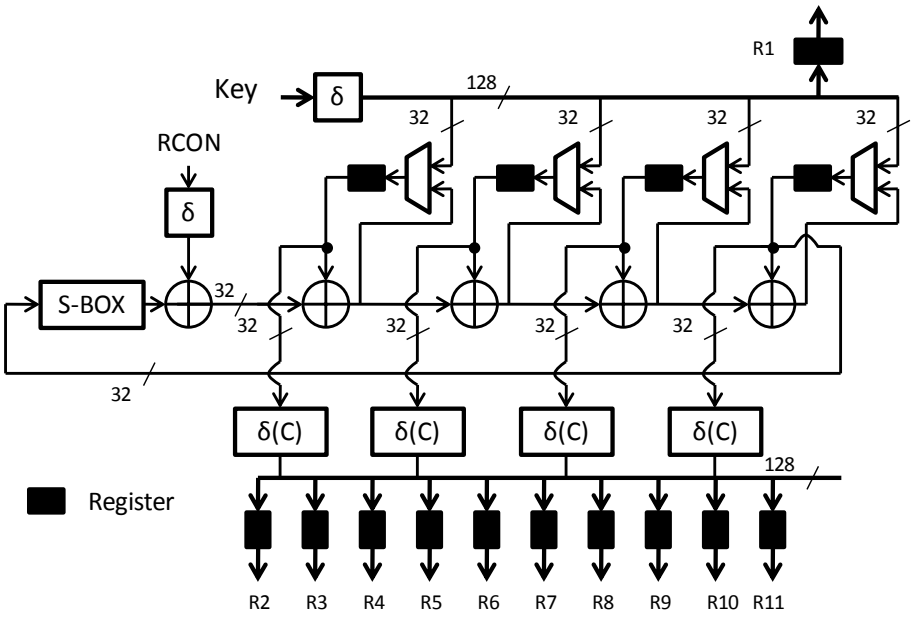


Fig. 33 Key Expansion unit for the 128 bits AES

3.5 AES Encryptor

This AES encryptor consists of 10 encryption stages. The first 9 stages is implemented using 4 I-BOXES in each stage, while the last encryption stage is implemented using normal S-BOXES as the Mix Columns step is not required in the last encryption iteration. Each S-BOX in the last stage is similar to the S-BOX in the key expansion unit, where it will consist of the multiplicative inverse block with "ζ" transformation. As I-BOX uses the "ζ" transformation, isomorphic mapping and inverse isomorphic mapping will be required at the start and the end of the encryptor only. Shift Rows step is obtained directly during the input/output operation between the I-BOX and S-BOX stages. Fig 34 shows the AES encryptor using the I-BOX technique.

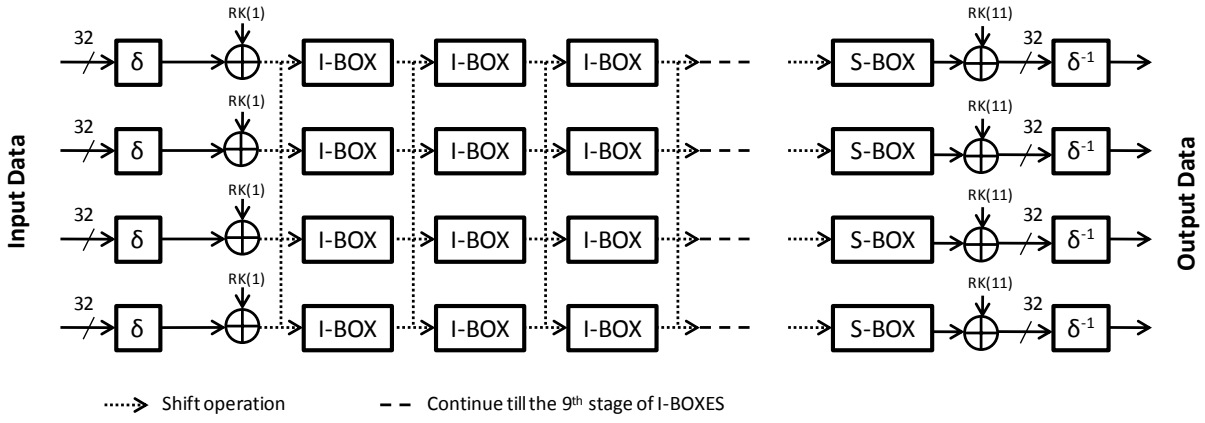


Fig. 34 AES encryptor using I-BOX technique

Appendix A shows a step by step example of AES Encryption using the I-BOX techniques.

3.6 Results and Comparison

The next two sub-sections present the sub-pipelining simulation results and the simulation results in comparison with previous proposed designs.

3.6.1 Sub-pipelining Simulation Results

The I-BOX has a total critical path of 26 gates, where each of its sub-blocks has a different critical path that plays a main role in determining the best number of used pipelining stages. Table 2 shows the number of gates and the critical path for each block for the composite field S-BOX [2].

To determine the number of the used sub-pipelining stages the AES encryptor was

simulated with 1-9 sub-pipelining stages and the best efficiency was achieved using 9 sub-pipelining stages, while the second best efficiency was achieved with 5 sub-pipelining stages. Table 3 shows the simulation results for architectures with 1-9 sub-pipelining stages using Xilinx XC2V6000-6.

Table 2: Gates and the critical path for the GF(2^8) Multiplicative Inverse sub-blocks

Block	Critical Path	Total Number of Gates
GF(2^4) Multiplier	5	30
GF(2^2) Multiplier	3	7
(x^{-1}) Inverse Block	5	25
Merged (x^2) and ($x\lambda$) Block	2	4
ζ-Transformation	3	15

Table 3: Simulation result based on the number of sub-pipelining stages

Sub-Pipelining Stages	Frequency (Mhz)	Throughput (Mpbs)	Slices	Critical Path	Efficiency (Mbps/Area)
1	87.4	11187	6462	26	1.731
2	122.1	15629	6767	13	2.310
3	169.9	21747	7092	9	3.066
4	188.4	24115	7501	7	3.215
5	218.3	27942	7884	6	3.544
6	236.0	30208	8743	5	3.455
7	258.8	33126	9461	4	3.501
8	261.9	33523	9981	4	3.359
9	305.1	39053	10662	3	3.663

Fig. 35 plots the relation between the number of used sub-pipelining stages and the efficiency for the proposed loop unrolled AES Encryptor.

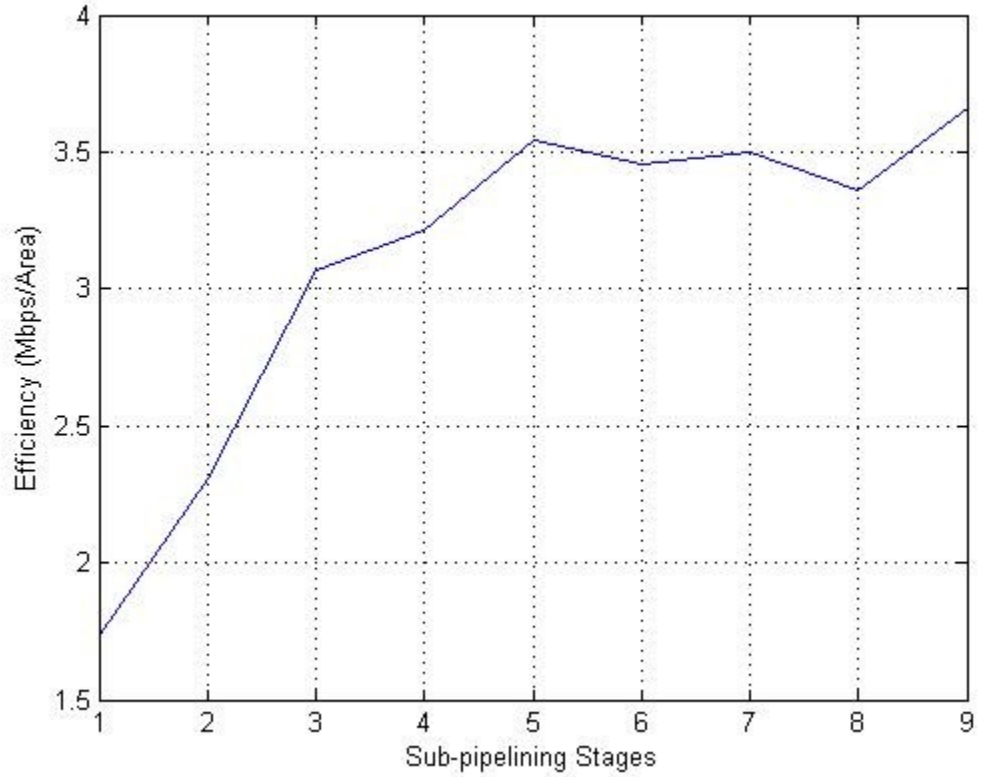


Fig. 35 Pipelining stages and Efficiency relationship for the proposed AES encryptor

As can be seen from Table 3 and Fig 35, applying one pipelining stage achieves the lowest efficiency. This kind of pipelining is shown in Fig 14 (b), where a pipelining stage is applied between the AES stages without the usage of any sub-pipelining stages. In the case of 2-5 pipelining stages the efficiency will increase as more pipelining stages are added. Applying 6 pipelining stages leads to less efficiency comparing to what can be achieved using 5 pipelining stages. In the case of 6 pipelining stages an internal pipelining must be applied in some of the sub-blocks of the composite field arithmetic S-BOX. This requires applying registers in each branch inside these sub-blocks, which result in higher usage of

registers comparing to case of 5 pipelining stages where the pipelining registers are needed only at the main buses in the composite field S-BOX.

Using 8 pipelining stages also achieved less efficiency than using 7 pipelining stages. The main reason for this is that in both cases the critical path of the system is 4 logic gates which prevent a significant increase in the system throughput. Using 9 pipelining stages achieves the best possible efficiency while applying more than 9 pipelining stages will cause a reduction in the efficiency as the critical path cannot be reduced to less than 3 gates until reaching a system with 13 pipelining stages. The key expansion unit has to be divided into the same number of sub-pipelining stages to maintain the synchronization between the main round units and the key expansion unit. Fig 30 shows the implementation of 5 and 9 sub-pipelining stages for the I-BOX where, “p” in this figure represents the number of sub-pipelining stages in each case.

3.6.2 Comparison with Previous Designs

Many designs rely on BRAMs in the implementation of AES encryptors, therefore, [6] suggested using the metric Mbps/Area for a better performance-area relationship instead of the metric Mbps/Slice to take into account the use of BRAMs in the efficiency calculation. The total area is obtained by adding the total slices and by considering each dual port 256*8 bit BRAM as equivalent to 128 slices [6].

Owing to the proposed merging techniques, this design was able to achieve higher efficiencies than the previous loop unrolled designs. By using 5 sub-pipelining stages this design achieves almost the same throughput obtained by 7 sub-pipelining stages in [2]. By

comparing this design with [12], a 37% improvement in efficiency was achieved based on the simulation results of the same FPGA device. The encryptor in [3] used BRAMs to implement the S-BOX. According to [3] these BRAMs are equivalent of 10240 extra slices. Simulated on the same device, the proposed design was able to achieve higher efficiency and throughput. Also, the proposed design was able to achieve higher efficiency than the designs in [11] and [5]. Table 4 summarizes the obtained results and the comparison with previous different implementations.

Table 4: Results and comparison of AES 128-bits encryptors

Design	Device	Freq. (Mhz)	TP (Mbps)	Slices	B-RAMs	Efficiency (Mbps/Area)
Jarvinen et al.[5]	XCV-1000e-8	129.2	16500	11719	0	1.408
Zhang et al.[2]	XCV-1000e-8	168.4	21556	11022	0	1.956
This work (p=5)	XCV1000e-8	168.3	21542	9104	0	2.366
Hodjat et al. [12]	XC2VP20-7	169.1	21645	9446	0	2.291
This work (p=5)	XC2VP20-7	220.7	28250	9028	0	3.129
Zambreno al. [11]	XC2V4000	184.2	23572	16938	0	1.392
This work (p=5)	XC2V4000-6	211.6	27087	8503	0	3.186
Granado. et al [3]	XC2V6000-6	194.7	24920	3576	80	1.804
This work (p=5)	XC2V6000-6	218.3	27942	7884	0	3.544
This work (p=9)	XC2V6000-6	305.1	39053	10662	0	3.663

CHAPTER 4 EFFICIENT 32-BITS AES IMPLEMENATION

This chapter presents a new hardware implementation for the Advanced Encryption Standard (AES) algorithm using 32 bits data path [19]. This architecture is presented using composite field arithmetic approach, with on-the-fly calculation for the round keys. To achieve higher efficiency the main round unit is implemented with internal pipelining and the S-BOX is shared between the main round unit and the key expansion unit. This design is implemented using FPGA technology, where higher efficiency in terms of (Throughput/Area) is achieved compared to previously proposed 32 bits AES designs.

4.1 Introduction

As mentioned in Chapter 2, AES Design can be categorized according to their data path width. Some design targeted very high speeds using 128 bits loop unrolled pipelined architectures as in [2] and [13]. Other architectures have targeted very low area implementations as in [17], where 8-bits data path architecture was implemented. Moreover, several other designs have suggested the implementation with 32-bits data path architectures targeting a medium throughput range as in [14-16].

This medium throughput ranges are needed in many applications such as cell-phones and portable devices, which cannot afford the excessive area needed in the 128 bits loop unrolled systems and do not require their tens of gigabits throughputs.

Most of the previous 32-bits AES designs such as in [14-16], used the technique of pre-calculation and storage of all round keys before starting the encryption/decryption

process. This method requires a large area to store all the round keys; also it is inefficient in case of frequent key changing. In these architectures, each time the key changes a pre-calculation for all round keys must be obtained before starting the encryption/decryption process, which leads to a considerable reduction in the system throughput.

This chapter presents a new efficient 32 bits AES design for the 128 bits key size. In this design the main round unit is implemented with internal pipelining stages to increase the throughput and the efficiency by parallel processing for the data and to allow efficient sharing for the S-BOX between the round and the key expansion units.

Instead of pre-calculation and storage of all round keys as in [14-16], the round keys are fed by the key expansion unit on-the-fly and that permits this unit to work in parallel with the main round unit. This key expansion unit will store only the round keys for the ongoing iteration and the last encryption/decryption iterations, where all the other keys will be calculated in forward and reserve orders during the encryption and decryption processes.

Owing to the internal pipelining for the main round unit, the on-the-fly calculation for the round keys, and the S-BOX sharing between the main round unit and the key expansion unit. This proposed AES architecture achieves higher FPGA efficiency compared to previous reported 32 bits designs in addition to cancelling the delay resulted by pre-calculation of all round keys at each key change.

4.2 Main Round Unit Implementation.

As described in [1], each AES data block consists of 128 bits (16 bytes) of data. During the encryption and decryption processes these 16 bytes will create a changeable (4*4) array called the state array. The proposed main round unit shown in Fig. 36 processes the state array column by column. This round unit is designed mainly with four “4*8 bits”

distributed RAMs, shared S-BOX/ Inverse S-BOX, shared Mix Columns/ Inverse Mix Columns, and finally the Add Round Key XOR gates. In this unit, 3 stages of internal pipelining are inserted inside the S-BOX/ Inverse S-BOX which allows parallel processing of the state array columns. Also with this internal pipelining, the S-BOXES can be shared with the key expansion unit with no extra delay where these S-BOXES will be accessed during the storage of the new state array in the RAMs.

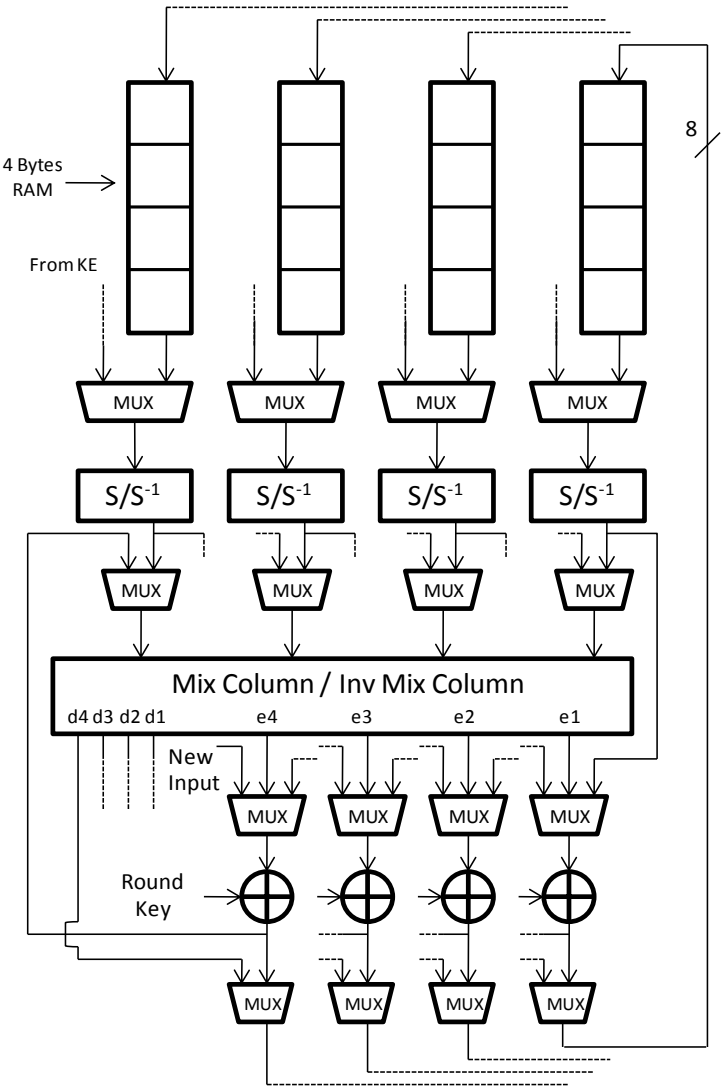


Fig. 36 Round unit for the AES with 32 bits data path

In the next sub-sections the different steps of the proposed AES system are presented.

4.2.1 Shift Rows/Inverse Shift Rows

As mentioned previously, the Shift Rows step is obtained by shifting the second, third, and fourth rows one, two and three cyclic shifts to the left, respectively, as shown in Fig. 37(a). The Inverse Shift Rows is obtained by shifting the second, third and fourth rows three, two and one cyclic shifts to the left, respectively, as shown in Fig. 37(b). In the main round unit, the four "4*8 bits" dual port distributed RAMs are used to store the state array bytes, and hence each distributed RAM will be storing one column of the state array. The Shift Rows and Inverse Shift Rows steps are obtained directly by the read/write operation in these distributed RAMs. While no shift is required in the RAM used to store the first row of the state array, the second, third and fourth RAMs need to perform one, two, and three shifts respectively. The second and fourth column of RAMs will be processing the second and fourth state array rows for the encryption and decryption processes interchangeably.

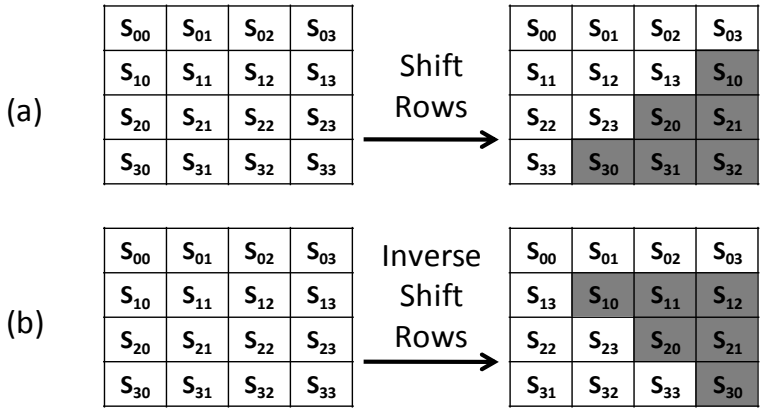


Fig. 37 Shift Rows and Inverse Shift Rows

4.2.2 Byte Substitution / Inverse Byte Substitution

As mentioned before, the main methods for implementing S-BOX are, either by

pre-storing the S-BOX elements, where the S-BOX elements are stored in a BRAM, or by using composite field arithmetic, where the S-BOX elements are calculated using combinational logic circuit.

Using BRAMs in the implementation requires storing the S-BOX and the Inverse S-BOX separately in different BRAMs. This method cannot get benefit from the shared operations between the S-BOX and its inverse and also it is not possible to apply pipelining using the BRAMs. Hence the S-BOX used in this proposed design is implemented using the composite field arithmetic method.

Applying the I-BOX, which was proposed in chapter 3 in looping architectures is not beneficial. The I-BOX applies merging between the inverse isomorphic mapping, the affine transformation and the isomorphic mapping for the next encryption stage in the loop unrolled system. Looping architecture has one looping encryption/decryption stage in comparison with series of stages in loop unrolled system where the merging is beneficial.

However, and as suggested by [8], the inverse isomorphic mapping block is merged with the affine transformation for the S-BOX, while the isomorphic transformation is merged with inverse affine transformation for the Inverse S-BOX. Fig. 39 shows the used shared S-BOX/Inverse S-BOX. As it can be seen from this figure, the Multiplicative inverse block is shared between the S-BOX and the Inverse S-BOX, also; the affine transformation and its inverse are merged with the isomorphic mapping and its inverse before and after the Multiplicative inverse block.

The S-BOX/Inverse S-BOX occupies the major part in the round unit's critical path, therefore, and in order to allow parallel processing for the state array columns, 3 pipelining stages were applied to this block. These pipelining stages allow also the key expansion unit to access the S-BOXES with no extra delay at the time of storing the new state array back into the RAMs.

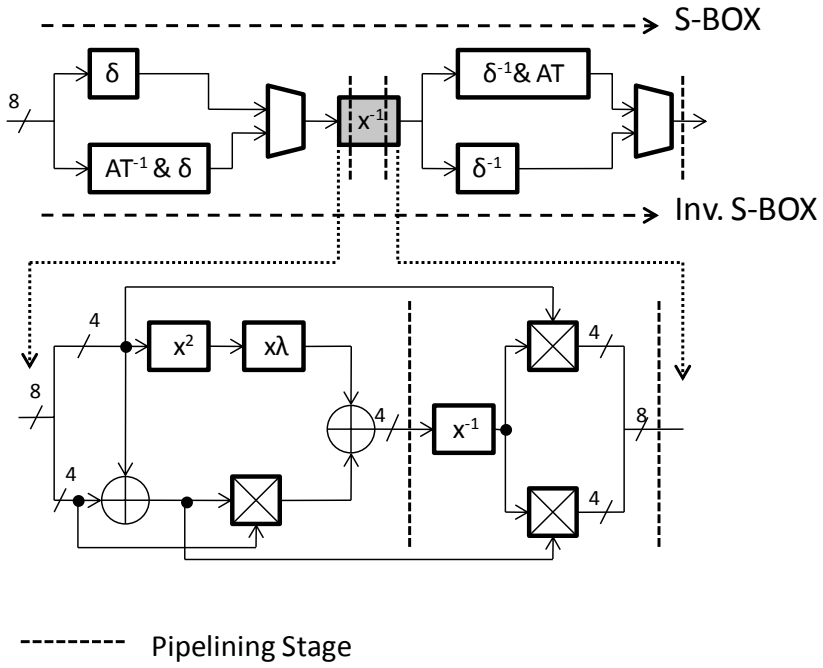


Fig. 38 Merged and pipelined S-BOX/Inverse S-BOX

With this pipelining each encryption/decryption iteration requires 7 clock cycles to be completed. This system has 32 bits data path architecture and it is not taking a new data at each clock cycle as in 128 bits loop unrolled systems, however pipelining this system will increase the throughput, especially, when the S-BOX is shared between the round and the key expansion units. Assume that the latency required to process each state array column is "n" so, ideally 3 pipelining stages will increase the clock frequency by 3 times while the latency is reduced to "n/3". A non-pipelined system with shared S-BOX will have a latency of "5n" for one encryption/decryption iteration as "4n" latency is required to process the 4 columns of the state array while an additional "n" for the access of the S-BOX by the key expansion unit. This system with 3 pipelined stages will have ideally a latency of "7*n/3" to complete one encryption/decryption iteration, which is a substantial reduction compared to the non-pipelined system. Fig. 39 shows the 7 clock cycles iteration for this system, in this figure the first RAM which stores the first state array row and requires no shift is used as an example. The symbols (C),(C') followed by a number, refers to the old and new state array element at the first row and at that specified column, while the (R) symbols refers to

the round key byte which is accessing the S-BOX. The 7 clock cycles starts with the new state array columns stored in the RAM, while the round key byte is under processing waiting to be stored at the second pipelining stage. In the next clock cycles, the state array bytes will go through these 3 pipelining stages to complete the encryption/decryption iteration. In the last clock cycle the element which was at the forth column is under processing and waiting to be stored back in the RAM at the next clock cycle, while the other 3 elements processing is completed. The shift rows step for other RAMs will be done by directly storing the new state array element in the proper location in the RAM.

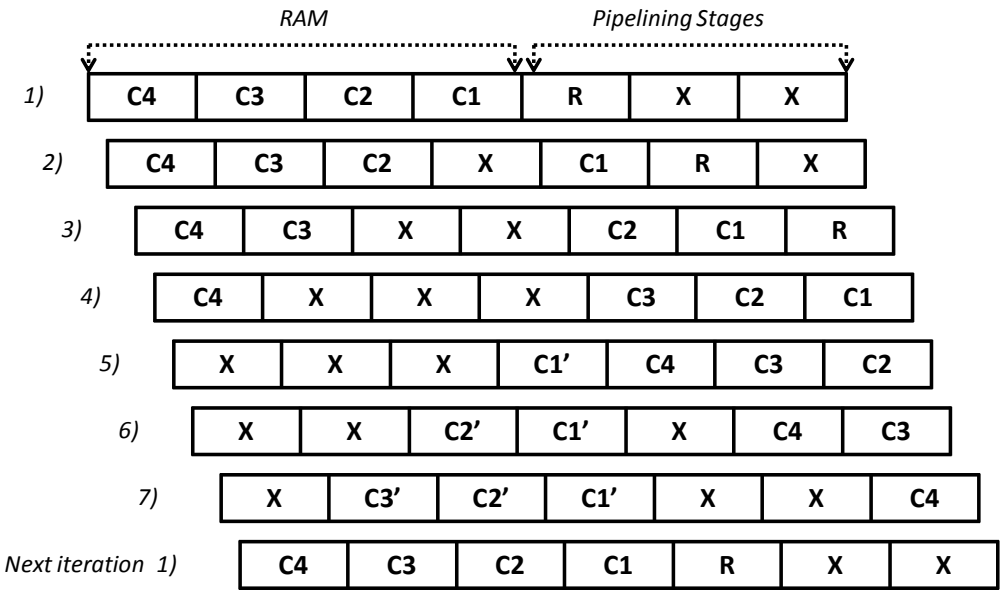


Fig. 39 7 Clock cycles iteration

4.2.3 Mix Columns / Inverse Mix Columns

For compact implementation, resource sharing is applied between the Mix Column and the Inverse Mix Column as will be explained in this section. Equation (4.1) represents the formula of the Mix Column polynomials, while (4.2) represents the Inverse Mix Column polynomials formula. The parameters “r” and “c” in (4.1) and (4.2) represent the state array row and column, respectively.

$$S_{r,c}' = \{02\}.S_{r,c} + \{03\}_{16}.S_{mod(r+1,4),c} + S_{mod(r+2,4),c} + S_{mod(r+3,4),c} \quad (4.1)$$

$$S_{r,c}' = \{0E\}.S_{r,c} + \{0B\}_{16}.S_{mod(r+1,4),c} + \{0D\}.S_{mod(r+2,4),c} + \{09\}.S_{mod(r+3,4),c} \quad (4.2)$$

To achieve resource-sharing, first the Mix column polynomial in (4.1) is written as (4.3) where the constant {02} is taken as a common factor for the state array elements $S_{r,c}$ and $S_{mod(r+1,4),c}$ and another single term for the state array element $S_{mod(r+1,4),c}$ is added to the equation. Inverse Mix Column polynomial in (4.2) is rewritten as in (4.4), where the multiplication with the constants {0E}, {0B}, {0D} and {09} are obtained by the multiplication with the constants {02}, {04} and {08}, and re-arranging the equation according to (4.4).

$$\text{e(c)} = S_{r,c}' = \{02\}.\left(S_{r,c} + S_{mod(r+1,4),c}\right) + S_{mod(r+2,4),c} + S_{mod(r+3,4),c} + S_{mod(r+1,4),c} \quad (4.3)$$

$$\begin{aligned} S_{r,c}'(i) = \{02\}.\left(S_{r,c} + S_{mod(r+1,4),c}\right) + S_{mod(r+2,4),c} + S_{mod(r+3,4),c} + \\ S_{mod(r+1,4),c} + \{04\}.\left(S_{r,c} + S_{mod(r+2,4),c}\right) + \\ \{08\}.\left(S_{r,c} + S_{mod(r+1,4),c}\, S_{mod(r+2,4),c} + S_{mod(r+3,4),c}\right) \end{aligned} \quad (4.4)$$

Finally (4.4) is written in (4.5) using (4.3) to achieve resource sharing:

$$\begin{aligned} d(c) = S_{r,c}' = e(c) + \{04\}.\left(S_{r,c} + S_{mod(r+2,4),c}\right) + \\ \{08\}.\left(S_{r,c} + S_{mod(r+1,4),c} + S_{mod(r+2,4),c} + S_{mod(r+3,4),c}\right) \end{aligned} \quad (4.5)$$

Equation (4.5) reveals that the Inverse Mix Column polynomial uses the Mix column polynomial in part of its calculation to simplify the needed resources.

4.2.4 Add Round Key

In the main round unit each round key XOR gate is preceded by 3-to-1 MUX as shown in Fig 31 The first input is used during the decryption process and the last encryption iteration to select the output of the S-BOX/Inverse S-BOX block. The second input is used to select the Mix Columns step output during the first 9 encryption iterations, while the third input is used for to enter a new data. Before storing the data back to the RAMs 2-to-1 MUXs are used to either choose the output of the Round Key XOR gates in case of encryption or the Inverse Mix Columns Step bytes in case of decryption.

4.3 Key Expansion Unit

The proposed key expansion unit is designed to work using on-the-fly technique, which means generation of all Round Keys during the encryption/decryption processes. This unit will store only the round keys for the present iteration and the last group of round keys. This will make it capable of generating the round keys in forward and reverse orders to be used in the encryption and decryption processes. Generation of Round Keys in forward order will be used during the encryption process starting from the original key, while the reverse order generation will be used during the decryption process starting from the last group of round keys. Fig. 40 shows the proposed design for this unit.

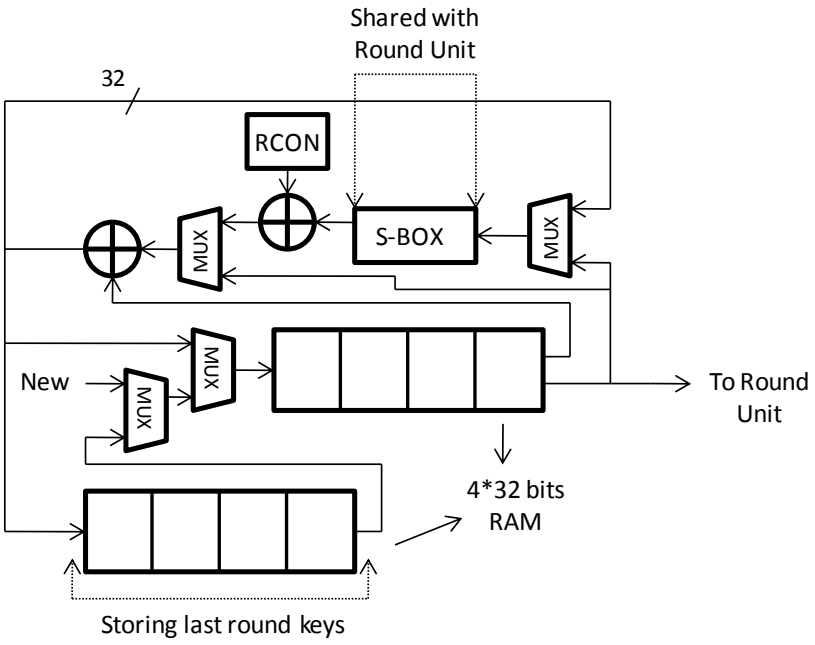


Fig. 40 Key Expansion unit for the 32 bits AES Design

As can be seen from Fig. 40, the key expansion unit has dual port RAM of size (4*32 bits). This RAM will be used to store the Round keys for the ongoing iteration. As mentioned previously, the S-BOXES are shared between the key expansion unit and the round unit. These S-BOXES will be accessed by the key expansion unit before the main round unit, at the time of storing the new state array bytes back in the RAMs. Another (4*32 bits) dual port RAM will be responsible of storing the last 16 bytes of round keys is added to the system. This RAM is beneficial in case of the occurrence of two sequential decryption processes working on the same key or the occurrence of encryption process followed by a decryption one. Decryption process use the reverse order generation of round keys starting from the last group, where by storing them there will be no need to re-calculate all round keys to get the last group back. As shown in Fig. 40, the RAM for the ongoing iteration of round keys have 3 input sources: new key, the last group of the round keys, and the previous round keys which will be used to generate next round keys.

4.4 Results and Comparison

The next two sub-sections present the internal pipelining simulation results and the simulation results in comparison with previous proposed designs.

4.4.1 Internal Pipelining Simulation Results

This design was implemented using 3 internal pipelining stages as it achieves the optimum (Mbps/Area) efficiency comparing to designs with other number of internal pipelining stages. Designs with 1 or 2 pipelining stages allows an efficient S-BOX sharing between the main round unit and the key expansion unit, however, designing with 3 pipelining stages achieves the highest efficiency. Adding more than 3 internal pipelining stages is considered as a waste of resources. In this 32 bits looping design, the system is not receiving a new data at each clock cycle as in 128 bits loop unrolled systems. Since the AES has 128 bits data block and the used bus width is 32 bits, adding more than 3 pipelining stages will result in creating clock cycles with no data under processing, therefore, the systems latency increases without achieving higher throughputs.

Design without pipelining stages requires the same latency of a design with one pipelining stage. Parallel processing between the main round unit and the key expansion unit will not be applicable in case of not applying pipelining stages. Table 5 shows the simulation results for system with 0 to 3 internal pipelining stages using Xilinx XC2VP2. Fig. 41 shows the relation between the number of used internal pipelining stages and the efficiency for the proposed loop 32-bits AES design.

Table 5: Simulation results based on number of internal pipelining stages

Pipelining Stages	Frequency (Mhz)	Throughput (Mpbs)	Slices	Latency (Clock Cycles)	Efficiency (Mbps/Area)
0	59.7	142	372	54	0.382
1	96.8	230	389	54	0.590
2	136.2	272	407	64	0.668
3	172.6	299	426	74	0.702

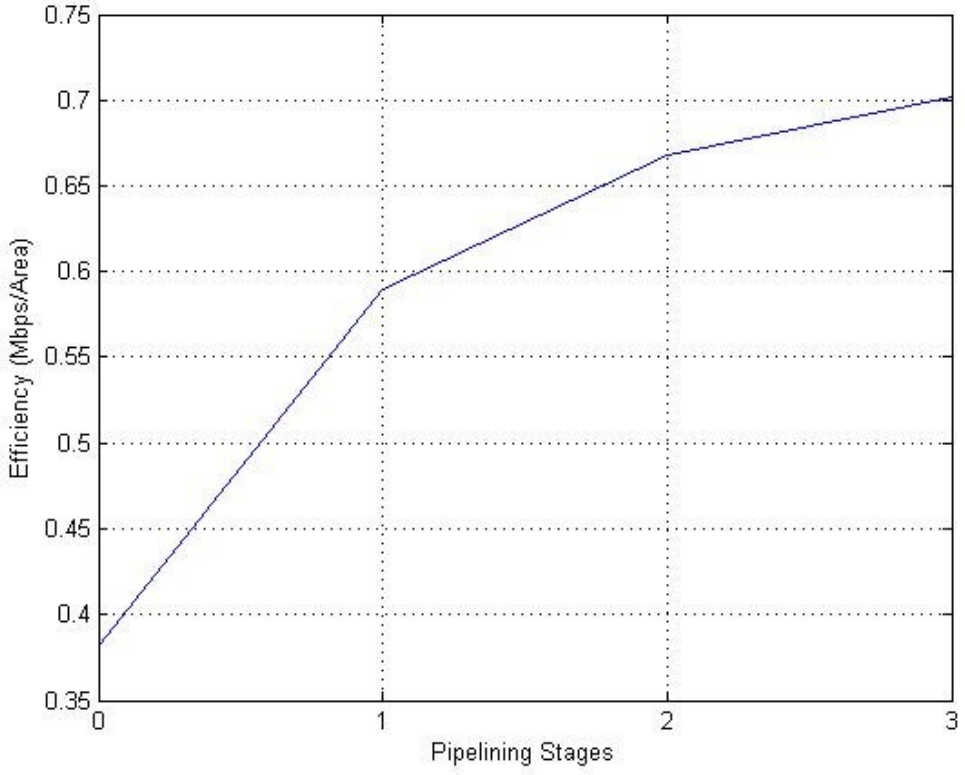


Fig. 41 Pipelining stages and Efficiency relationship for the proposed 32- bits design

4.4.2 Comparison with Previous Designs

The proposed AES design using on-the-fly key expansion unit, the S-BOXES sharing between the key expansion and the main round units and also the internal pipelining, is able to achieve higher FPGA efficiency comparing to the previous 32-bits data path AES design.

Another important factor to be taken into the consideration is the delay required at every key changing. This delay is needed for all the designs that use the pre-calculation and storage for round keys prior to the encryption/ decryption processes. Table 6 shows the simulation results of this design based on different FPGA Xilinx devices, while Table 7 shows a comparison with previous proposed designs.

Table 6: Simulation results of the proposed AES 32-bits design using different devices

Device	XC2VP2	XC2V40-6	XC2S30
Frequency(Mhz)	172.6	150.6	69.1
Throughput(Mpbs)	299	260	120
Slices	426	427	413
Efficiency (Mbps/Area)	0.702	0.609	0.291
BRAMs	0	0	0
Key Expansion Delay Required	No	No	No

Table 7: Comparison with other AES 32-bits designs

Design	Gaj [14]	Rouvroy [15]	Chang [16]	**This Work**
Device	XC2S30	XC2V40-6	XC2VP2	XC2VP2
Frequency (Mhz)	60	71.5	306	172.6
Throughput(Mpbs)	166	358	876	299
Slices	222	146	156	426
BRAMs used	3	3	3	0
Bytes in BRAMs	1200	4676	3248	0
Equiv. BRAM Slices (est.)	600	2338	1624	0
Total Area in Slices (est.)	822	2484	1780	426
Efficiency (Mpbs/Area)	0.202	0.144	0.492	0.702
KE Delay Required	Yes	Yes	Yes	No

As can be seen from Table 6 and Table 7, the proposed architecture shows higher efficiency than the designs in [14], [15] and [16]. Table 6 shows that this design achieved an efficiency of 0.291 (Mbps/Area) using Xilinx Spartan II - XC2S30 , while design [14] as can be seen from Table 5 achieved an efficiency of 0.202 (Mbps/Area) based on the same device. This design also achieved an improvement of 433% in terms of (Mpbs/Area) efficiency comparing to the design in [15], where both designs used Xilinx Vertix-2P XC2V40-6 in the simulation. Moreover, higher efficiency is achieved comparing to design [16] as can be seen from Table 7.

The area required to store all round keys in [14-16] in addition to the area needed for storing the S-BOXES in [14] and the combination of S-BOX/Mix Column in [15] and [16], has a main role in increasing the total area and reducing the efficiency. Moreover, our

proposed design avoids the need for any delay to pre-calculate the Round Keys since it is done on-the-fly. The designs [14-16] require delay of 44 clock cycles at the system start-up and at every key change. We have used distributed selected RAMs instead of BRAMs in our design as we are using very small size RAMs. The area of these RAMs was already counted for in the total number of slices needed for the proposed AES.

CHAPTER 5 CONCULSION AND FUTURE WORK

This chapter presents the conclusion of this thesis and the proposed future work.

5.1 Conclusion

In this thesis, two new hardware architectures for the Advanced Encryption Standard (AES) algorithm were presented. FPGA Xilinx technology was used to synthesis the designs and provide post placement results using Xilinx ISE 10.1.

In the first architecture a new design for high speed loop unrolled sub-pipelined AES encryptor was presented. This design took advantage from the repeated operations in each stage of the encryptor to achieve resources merging and sharing. In this encryptor the mix columns step is relocated and all the round keys are obtained in the isomorphic mapping. By applying these modifications an efficient merging between the inverse isomorphic mapping, the affine transformation multiplication, and the isomorphic mapping for the next encryption stage is achieved. This merging allowed the implementation to have lower area with shorter path length, which allowed higher FPGA (Throughput/Area) efficiency comparing to previous loop unrolled designs.

In the second architecture a new design for 32-bits data path AES encryptor/decryptor was presented. In this design internal pipelining for the composite field S-BOX was applied. This pipelining allowed parallel processing for the state array columns in addition to S-BOX sharing between the main round unit and the key expansion unit. Moreover, this design used on the fly generation for all round keys which prevents using large area to store

all the keys in addition to cancelling the extra delay resulting in pre-calculation and storage for all round keys. This architecture has achieved higher FPGA(Throughput/Area) efficiency compared to previous 32-bit AES designs.

5.2 Future Work

The research works achieved in this thesis are behind our motivation to present the following recommendations for future research investigations in the hardware design for the AES algorithm and other possible cryptography algorithms.

1. The I-BOX technique which was presented could be adopted in design of the AES with 192 and 256 bits key sizes.

2. Very high speed universal AES Processor that works on all key sizes could be implemented by getting benefit from the I-BOX technique using a loop unrolled system.

3. Future AES designs with 8-bits data path could be designed based on the S-BOX sharing and the pipelining techniques presented in Chapter 4.

4. Other cryptography algorithms might benefit from the ideas of merging and relocating techniques, especially in loop unrolled systems.

BIBLIOGRAPHY

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

[2] X. Zhang and K. K Parhi, "High-speed VLSI Architecture for the AES Algorithm", *IEEE Transactions on Very Large Scale Integration (VLSI) System.*, vol.12, no. 9, pp. 957–967, Sep. 2004.

[3] Jose M. Granado-Criado , Miguel A.Vega-Rodrıguez, Juan M. Sanchez-Perez and Juan A. G´omez-Pulido, "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration", Integration, the VLSI Journal 43 (2010) 72-80.

[4] Vincent Rijmen, "Efficient Implementation of the Rijndael S-box". Katholieke Universiteit Leuven, Dept. ESAT. Belgium".

[5] Jarvinen, K., Tommiska, M., and Skytta, "A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor". Proc. ACM/SIGDA 11th ACM Int. Symposium on Field-Programmable Gate Arrays, FPGA 2003, Monterey, CA, USA, February 2003, pp. 207–215.

[6] Kimmo Järvinen, Matti Tommiska and Jorma Skyttä, "Comparative Survey of High Performance Cryptographic Algorithm Implementations on FPGAs", IEE Proceedings - Information Security, vol. 152, no. 1, Oct. 2005, pp. 3-12.

[7] C. Paar, “Efficient VLSI architecture for bit-parallel computations in Galois field,” Ph.D. dissertation, Institute for Experimental Mathematics, University of Essen, Essen, Germany, 1994.

[8] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “A compact Rijndael hardware architecture with S-Box optimization,” in *Proc. ASIACRYPT 2001*, Gold Coast, Australia, Dec. 2000, pp. 239–254.

[9] A. Rudra, P.K. Dubey, C.S. Jutla, V. Kumar, J.R. Rao, and P. Rohatgi. “Efficient Rijndael Encryption Implementation with Composite Field Arithmetic”, *Workshop on Cryptographic Hardware and Embedded Systems (CHES2001)*, pages 175–188, May 2001.

[10] Edwin NC Mui, “Practical Implementation of Rijndael S-Box Using Combinational Logic”, Texco Enterprise Ptd. Ltd, [Online]. Available: http://www.xess.com/projects/ Rijndael_SBox.pdf.

[11] Joseph Zambreno, David Nguyen, and Alok Choudhary, “Exploring Area/Delay Tradeoffs in an AES FPGA Implementation”, Department of Electrical and Computer Engineering, Northwestern University.

[12] Hodjat A. and Verbauwhede. I, “A 21.54 Gbits/s fully pipelined AES processor on FPGA”. Proc.12th Annual IEEE Symposium. Field Programmable Custom Computing Machines, FCCM’04, Napa, CA, USA, April 2004, pp.308–309.

[13] I. Hammad, K. El-Sankary, and E. El-Masry, “High Speed AES Encryptor with Efficient Merging Techniques,” *IEEE Embedded Systems letters*, vol. 2, no. 3, pp. 67-71, Sept. 2010.

[14] K. Gaj and P. Chodowiec. Very Compact FPGA Implementation of the AES Algorithm. In the proceedings of CHES 2003, Lecture Notes in Computer Science, vol 2779, pp. 319-333, Springer-Verlag.

[15] 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", Information Technology Coding and Computing 2004.

[16] Chi-Jeng Chang , Chi-Wu Huang , Kuo-Huang Chang , Yi-Cheng Chen and Chung-Cheng Hsieh, "High Throughput 32-bit AES Implemenation in FPGA" , Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference.

[17] Tim Good and Mohammed Benaissa, "Very Small FPGA Application-Specific Instruction Processor for AES", IEEE Transactions on Circuit and Systems-I, Vol. 53, No. 7, July 2006.

[18] Daemen**,** Joan and Rijmen Vincent, "The Design of Rijndael" – The Advanced Encryption Standard", 2002, Springer.

[19] Hammad, K. El-Sankary, and E. El-Masry, " 32-Bits AES Implementation with Internal Pipelining and S-BOX Sharing", EURASIP Journal on Embedded Systems. Submitted.

[20] Data Encryption Standard (DES), FIPS PUB (46-3), Oct. 25, 1999, Federal Information Processing Standard 46-3.

[21] Dirk Rijmenants, Hand Ciphers, Cipher Machines and Cryptography, [Online]. Available: http://users.telenet.be/d.rijmenants/en/handciphers.htm.

[22] Biham, Eli and Adi Shamir, Differential Cryptanalysis of the Data Encryption Standard, Springer Verlag, 1993.

[23] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. Pages 859–861 of section 31.2: Greatest common divisor.

[24] Seagate – Technology Paper, "128 Bit Versus 256 Bit AES Encryption", Practical business reasons why 128 bit solution provide comprehensive security for every need. www.seagate.com/staticfiles/docs/pdf/whitepaper/tp596_128_bit_versus_256_bit.pdf

[25] Saggese, G.P., Mazzeo, A., Mazzocca, N., and Strollo, A.G.M: 'An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm'. Proc. 13th Int. Conf. Field Programmable Logic and Applications, FPL 2003, Lisbon, Portugal, September 2003, pp. 292–302.

[26] X. Zhang and K. K. Parhi, "On the Optimum Constructions of Composite Field for the AES Algorithm," TCAS-II, vol. 53(10), pp. 1153-1157, Oct. 2006.

APPENDIX A: FINITE FIELD ARITHMETIC EXAMPLES

Example 1: Addition in Finite Field

Adding the polynomials $(x^5+x^3+x^2+x) + (x^6+x^5+x+1)$

$(x^5+x^3+x^2+x) = \{101110\}$

$(x^6+x^5+x+1) = \{1100011\}$

$(x^5+x^3+x^2+x) + (x^6+x^5+x+1) = x^6+2x^5+x^3+x^2+2x+1.$

And since we use XOR operation instead of addition any operands with even coefficients will be eliminated.

Final Answer: $x^6+x^3+x^2+1 = \{1001101\}.$

As can be seen from example 1, the addition operation in finite field is considered as XOR operation, where any operand with even coefficient will be eliminated.

The next example shows the multiplication operation in the GF(2^8) and using the reduction polynomial defined in (2.7).

Example 2: Multiplication in Finite Field

Multiplication of the Polynomials $(x^3+x^2+1).(x^5+x^2+x)$

Modulus $(x^8+ x^4+ x^3+ x+1)$

$(x^3+x^2+1) = \{1101\}$

$(x^5+x^2+x) = \{100110\}$

$(x^8+x^4+x^3+x+1) = \{100011011\}$

$(x^3+x^2+1).(x^5+x^2+x) = x^8+x^5+x^4$

$+ x^7+x^4+x^3$

$+ x^5+x^2+x^1$

$= x^8+x^7+x^3+x^2+x^1 = \{110001110\}$

And by using the modulus $(x^8+x^4+x^3+x+1)$

$(x^8+x^7+x^3+x^2+x^1)$ Mod $(x^8+x^4+x^3+x+1)$

$=(x^8+x^7+ x^3 +x^2+ x^1) + (x^8+x^4+x^3+x+1) = x^7+x^4+x^2+1 = \{10010101\}$

As can be seen from example 2, after the multiplication is obtained. The reduction polynomial will be applied to get a value that exists in the GF(2^8).

The next example shows how modulo operation is obtained on polynomials with high degrees.

Example 3: Using Modulo with Large Numbers.

Finding $(x^{14}+x^{13}+x)$ Mod $(x^8+x^4+x^3+x+1)$

$(x^{14}+x^{13}+x) = \{110000000000010\}$

$(x^8+x^4+x^3+x+1) = \{100011011\}$

First: Shifting the modulus to the same degree of the input

$(x^8+x^4+x^3+x+1) \rightarrow$ Shifting $\rightarrow (x^{14}+x^{10}+x^9+x^7+x^6) = \{100011011000000\}$

Second: Obtain XOR Operation between both numbers.

{110000000000010} XOR {100011011000000} = {10011011000010} = $x^{13}+ x^{10}+x^9+x^7+x^6+x$

Repeat the operation until the output is in 7^{th} degree at max.

$(x^8+x^4+x^3+x+1)$ → Shifting →$(x^{13}+x^9+x^8+x^6+x^5)$ = {10001101100000}

{10011011000010} XOR {10001101100000} = {10110100010} = $x^{10}+x^8+x^7+x^5+x$

Repeat again.

$(x^8+x^4+x^3+x+1)$ → Shifting →$(x^{10}+x^6+x^5+x^3+x^2)$ = {10001101100}

{10110100010} XOR{10001101100} = 111001110 = $x^8+x^7+x^6+x^3+x^2+x$

And Finally,

{111001110} XOR {100011011} = {11010101} = $x^7+x^6+x^4+x^2+1$

And since the value $x^7+x^6+x^4+x^2+1$ exists in GF(2^8), it is the final answer.

Example 3, showed how the reduction polynomial is used repeatedly until the answer exists in the GF(2^8).

APPENDIX B: ENCRYPTION EXAMPLE USING I-BOX

This Appendix shows an encryption example using the I-BOX technique with128 bits key size. According to [1] the test vector shown in (A.1) with the key in (A.2) will result in the enciphered output shown in (A.3).

$$Data = \{32\ 43\ F6\ A8\ 88\ 5A\ 30\ 8D\ 31\ 31\ 98\ A2\ E0\ 37\ 07\ 34\} \quad (A.1)$$

$$Key = \{2B\ 7E\ 15\ 16\ 28\ AE\ D2\ A6\ AB\ F7\ 15\ 88\ 09\ CF\ 4F\ 3C\} \quad (A.2)$$

$$Output = \{39\ 25\ 84\ 1D\ 02\ DC\ 09\ FB\ DC\ 11\ 85\ 97\ 19\ 6A\ 0B\ 32\} \quad (A.3)$$

The new I-BOX technique will result in the same output (A.3) using the data (A.1) and the key (A.2). By using the I-BOX technique all Round keys will be mapped to the isomorphic mapping with the constant δ(C) which equals $A5_H$ added to them as described in (A.4):

$$RK' = \delta(RK) + \delta(C) \quad (A.4)$$

The input key shown in (A.2) will be entered column by column as shown below:

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

Original 128-bits key

By applying the key expansion operation explained in Chapter 2, and according to (A.4), the 176 bytes of round keys, which will be generated as 11 group with 16 bytes each is shown below:

9A	C4	66	75
EA	1B	63	E1
3B	AE	3B	1D
65	6F	88	FE

Initial Round Keys

E9	2D	4B	3E
CF	D4	B7	56
B3	1D	26	3B
C1	AE	26	D8

2nd Group of Round Keys

BB	33	DD	46
4D	3C	2E	DD
62	DA	59	C7
BB	B0	33	4E

3rd Group of Round Keys

5A	CC	B4	57
59	C0	4B	33
CC	B3	4F	2D
11	04	92	79

4th Group of Round Keys

F4	9D	8C	7E
92	F7	19	8F
94	82	68	E0
EF	4E	79	A5

5th Group of Round Keys

28	10	39	E2
55	07	BB	91
31	16	DB	9E
90	7B	A7	A7

6th Group of Round Keys

57	E2	7E	39
2D	8F	91	A5
B7	04	7A	41
33	ED	EF	ED

7th Group of Round Keys

B9	FE	25	B9
D4	FE	CA	CA
C6	67	B0	5C
F2	BA	F0	BB

8th Group of Round Keys

89	D2	52	4E
0B	50	3F	50
46	84	99	60
14	0B	5E	43

9th Group of Round Keys

E1	96	61	8A
3A	CF	55	A0
7D	5C	60	A5
BA	14	EF	09

10^{th} Group of Round Keys

54	67	A3	8C
9F	F5	05	00
91	68	AD	AD
9D	2C	66	CA

11^{th} Group of Round Keys

The AES encryptor using the I-BOXES as shown in Fig. 30 starts by mapping the input data using the isomorphic, then adding the initial Round Key. After adding the initial Round Key the I-BOXES are used for 9 stages. Each I-BOX will process a column of the state matrix, therefore each stage will consist of 4 I-BOXES. Finally, at the last encryption stage the bytes are substituted using the S-BOX and then they are added to the last group of round keys. Shift Rows step is obtained during the input/output operations between the stages.

It is important to notice that the multiplicative inverse calculations shown in the example below are obtained without the isomorphic mapping and its inverse as done in Fig. 21. Also, the S-BOX substitution used in the example exclude the isomorphic mapping from the calculation as the ζ-Transform is applied. This substitution differs from the S-BOX Substitution values shown in Fig. 7 which uses the isomorphic mapping in the calculation.

32	88	31	E0
43	5A	31	37
F6	30	98	07
A8	8D	A2	34

Original Data Input

At Encryption starting:

A9	88	F7	07
15	26	F7	D4
62	F6	CE	22
38	F5	13	8A

After isomorphic mapping

33	4C	91	72
FF	3D	94	35
59	58	F5	3F
5D	9A	9B	74

After adding initial Round Key

First I-BOXES Stage:

33	4C	91	72
3D	94	35	FF
F5	3F	59	58
74	5D	9A	9B

First I-BOXES Input

A0	6B	EC	1C
8F	7B	F8	20
5F	6E	2D	D2
73	DF	3B	CE

After Multiplicative Inverse

B8	6B	CA	11
A5	A1	09	7E
D0	7F	C3	40
CE	14	02	0F

After Mix Columns

D9	A2	9A	61
7D	38	90	86
58	96	0A	36
DF	34	33	E6

After ζ- Transform

30	8F	D1	5F
B2	EC	27	D0
EB	8B	2C	0D
1E	9A	15	3E

After Add Round Key

Second I-BOXES Stage:

30	8F	D1	5F
EC	27	D0	B2
2C	0D	EB	8B
3E	1E	9A	15

Second I-BOXES Input

AA	3D	A7	F5
91	41	BB	34
D3	0E	A3	83
87	82	3B	D9

After Multiplicative Inverse

FA	B9	91	B9
6F	6C	08	52
C1	3C	C1	BE
3B	19	DC	CE

After Mix Columns

DC	C9	7E	C9
E7	C4	80	74
39	83	39	AF
E5	E1	9D	DF

After ζ- Transform

67	FA	A3	8F
AA	F8	AE	A9
5B	59	60	68
5E	51	AE	91

After Add Round Key

Third I-BOXES Stage:

67	FA	A3	8F
F8	AE	A9	AA
60	68	5B	59
91	5E	51	AE

Third I-BOXES Inputs

19	5A	EB	3D
35	1B	E5	30
66	3C	64	2D
EC	62	54	1B

After Multiplicative Inverse

33	43	0E	4A
FC	89	55	E6
4F	00	64	65
26	D5	01	F2

After Mix Columns

65	15	F6	85
AA	8F	12	68
D0	00	44	54
41	0D	10	5C

After ζ- Transform

3F	D9	42	D2
F3	4F	59	5B
1C	B3	0B	79
50	09	82	25

After Add Round Key

Fourth I-BOXES Stage:

3F	D9	42	D2
4F	59	5B	F3
0B	79	1C	B3
25	50	09	82

Fourth I-BOXES Inputs

6E	15	CD	58
4B	2D	64	A8
07	32	72	F9
45	11	06	1E

After Multiplicative Inverse

4C	A6	97	F9
66	13	55	18
F1	21	4D	EB
BC	8F	52	1D

After Mix Columns

F3	5E	08	FF
77	52	12	F1
7F	27	E3	BD
9C	F9	74	A4

After ζ- Transform

07	C3	84	81
E5	A5	0B	7E
EB	A5	8B	5D
73	B7	0D	01

After Add Round Key

Fifth I-BOXES Stage:

07	C3	84	81
A5	0B	7E	E5
8B	5D	EB	A5
01	73	B7	0D

Fifth I-BOXES Input

0B	39	D6	E3
AF	07	31	A9
83	DF	A3	AF
01	74	2A	0E

After Multiplicative Inverse

E6	49	BD	0F
05	2D	E7	0F
BC	05	D5	55
79	F4	E1	BE

After Mix Columns

68	A6	8C	E6
55	E2	78	E6
9C	55	0D	12
E0	2A	0E	AF

After ζ- Transform

40	B6	B5	04
00	E5	C3	77
AD	43	D6	8C
70	51	A9	08

After Add Round Key

Sixth I-BOXES Stage:

40	B6	B5	04
E5	C3	77	00
D6	8C	AD	43
08	70	51	A9

Sixth I-BOXES Input

EE	C4	4D	0F
A9	39	C0	00
84	DB	DC	71
0A	CC	54	E5

After Multiplicative Inverse

6C	D4	D5	10
B3	0C	77	A3
47	82	9E	1F
51	B0	39	37

After Mix Columns

C4	1D	0D	71
7A	C5	16	0B
50	2C	98	97
57	59	D6	20

After ζ- Transform

93	FF	73	48
57	4A	87	AE
E7	28	E2	D6
64	B4	39	CD

After Add Round Key

Seventh I-BOXES Stage:

93	FF	73	48
4A	87	AE	57
E2	D6	E7	28
CD	64	B4	39

Seventh I-BOXES Inputs

38	20	74	6D
16	3E	1B	7A
98	84	13	BC
42	5B	6A	C3

After Multiplicative Inverse

88	9B	20	B6
E8	4E	9B	D7
1B	E8	AB	37
8F	FC	06	3E

After Mix Columns

9F	CD	37	2F
9E	C0	CD	3E
D2	9E	8B	20
F9	AA	76	B0

After ζ- Transform

26	33	12	96
4A	3E	07	F4
14	F9	33	7C
0B	10	86	08

After Add Round Key

Eighth I-BOXES Stage:

26	33	12	96
3E	07	F4	4A
33	7C	14	F9
08	0B	10	86

Eighth I-BOXES Inputs

EF	A0	E9	F1
87	0B	63	16
A0	9D	D4	B3
0A	07	55	6F

After Multiplicative Inverse

7D	DE	6D	CF
C8	07	95	B9
72	82	51	B2
05	6A	A2	FF

After Mix Columns

A5	AE	D4	CF
A9	66	3B	C9
43	2C	57	6A
55	B2	1B	89

After ζ- Transform

2C	7C	86	81
A2	36	04	99
05	A8	CE	0A
41	B9	45	CA

After Add Round Key

Ninth I-BOXES Stage:

2C	7C	86	81
36	04	99	A2
CE	0A	05	A8
CA	41	B9	45

Ninth I-BOXES Inputs

D3	9D	6F	E3
F7	0F	80	FC
9B	08	0C	F3
8D	27	37	25

After Multiplicative Inverse

FB	98	28	0F
10	A0	96	B1
45	AE	E7	5E
9C	2B	8D	29

After Mix Columns

CC	EE	B7	E6
71	28	18	49
63	DE	78	B1
AB	94	CA	A7

After ζ- Transform

2D	78	D6	6C
4B	E7	4D	E9
1E	82	18	14
11	80	25	AE

After Add Round Key

Last S-BOXES Stage:

2D	78	D6	6C
E7	4D	E9	4B
18	14	1E	82
AE	11	80	25

Last S-BOXES inputs

59	A1	84	BF
13	B5	12	4F
61	D4	82	1E
1B	50	99	45

After Multiplicative Inverse

D7	38	5A	BF
52	0C	42	D0
11	1D	2C	87
D2	47	FE	63

After ζ- Transform

Last Encryption Steps:

83	5F	F9	33
CD	F9	47	D0
80	75	81	2A
4F	6B	98	A9

After Adding Last Round Keys

39	02	DC	19
25	DC	11	6A
84	09	85	0B
1D	FB	97	32

After inverse isomorphic mapping

The last block of data after the inverse isomorphic mapping represents the output enciphered data. As can be seen below this data same to enciphered output (A.3) according to [1].

39	02	DC	19
25	DC	11	6A
84	09	85	0B
1D	FB	97	32

Enciphered Output Data