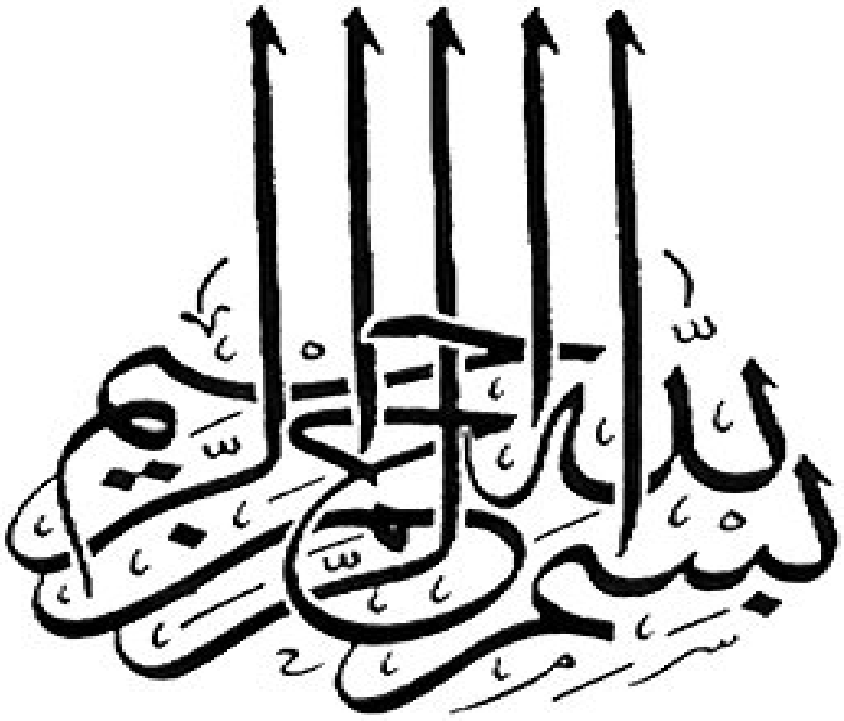


In the name of Allah
The most Beneficent, most Merciful

AREA AND RESOURCE EFFICIENT ARCHITECTURE FOR AES ON FPGA

Submitted by :

Saleha Zaka

Supervised by:

Dr. Arshad Aziz

Thesis Submitted in Partial Fulfillment of the Requirement for the Degree of Master of Science in Electrical Engineering with Specialization in Communication

At The

Department of Electronic and Power Engineering
Pakistan Navy Engineering College, Karachi
National University of Sciences & Technology,
H-12, Islamabad, Pakistan.

March 2011

Dedicated to my parents and siblings.

Acknowledgements

All thanks and praise is for Allah alone. I am humbly grateful to Allah Almighty Who bestowed blessings on me and gave me the energy and faith to complete my work successfully. And without Whose help I couldn't have done any part of my work.

I am grateful to my supervisor Dr. Arshad Aziz for his consistent guidance throughout the course of my work. He encouraged me to keep doing better than my best. I am also thankful to my GEC members; Dr. Pervez Akhtar, Dr. Vali Uddin, Dr. Athar Mahboob and external examiner Dr. Imran Baig.

I am grateful to my colleagues who proved to be a source of inspiration for me during the tough moments in the research, documentation and presentation phase. My special thanks go to my parents, sister, brothers and friends who relentlessly and devotedly stood with me through all the ups and downs, and who had a smile ready for me in every bleak moment.

Abstract

Cryptography, the encryption and decryption of data, is the need of the day. It fulfills the security requirements of computer and communication systems. Our banking transactions, email accounts, home appliances, hospital data, online transactions, all and more, are clients of cryptography. They are as safe as the encryption technique itself. In the year 2001, National Institute of Standards and Technology (NIST) announced the Advanced Encryption Standard (AES) to be the new US government standard for encryption. Since then, there has been rapid adoption of this standard by both government and private organizations.

An encryption cipher may be implemented on a hardware device or on software. While software-based encryption provides flexibility, hardware-based encryption provides better system performance. Reconfigurable devices, Field Programmable Gate Arrays (FPGAs) to be specific, are unique in the sense that they combine the advantages of hardware and software implementations.

The aim of this thesis is to present a novel area-efficient technique to implement AES on FPGA. Keeping in view the consistent demand of the programmable logic industry for low cost, this is an ideal method to reduce resource utilization and hence cut down on costs. SubBytes transformation, which is part of AES, is an intensive and resource-hungry transformation. The proposed method in this thesis optimizes the implementation of SubBytes transformation, using embedded BRAM in an FPGA.

Table of Contents

Abstract iv

Table of Contents v

List of Figures viii

List of Tables ix

1 Introduction 1

1.1 Background 1

1.2 Aim of Thesis 2

1.3 Thesis Outline 2

2 Symmetric Cryptography 3

2.1 Introduction 3

2.2 Encryption Using Advanced Encryption Standard 4

2.2.1 SubBytes 6

2.2.2 ShiftRows 6

2.2.3 MixColumns 7

2.2.4 AddRoundKey 7

2.3 Key Expansion 8

2.4 Decryption 8

2.5 AES Key Sizes 10

2.6 AES in Practice 10

2.7 Summary 10

3 FPGA 11

3.1 Introduction 11

3.2 Advantages of FPGA 11

3.3 XILINX FPGA 11

3.4 Spartan-3 FPGA Family 12

3.4.1 Architectural and Functional Overview 12

3.4.1.1 Configurable Logic Block (CLB) 12
3.4.1.2 Input/Output Block (IOB) 13
3.4.1.3 Block RAM (BRAM) 13
3.4.1.4 Multiplier Blocks 14
3.4.1.5 Digital Clock Manager (DCM) 14
3.5 XILINX ISE 14
3.6 Summary 14
4 Our Work 15
4.1 Overview 15
4.1.1 Conventional Technique of Implementing SubBytes Transformation 16
4.1.2 BRAM Technique of Implementing SubBytes Transformation 16
4.2 Choice of Architecture 17
4.2.1 Pipeline Architecture 17
4.2.2 Sub Pipeline Architecture 17
4.2.3 Loop Unrolled Architecture 18
4.3 Our AES Implementation Based on LUT Approach 18
4.3.1 SubBytes 20
4.3.2 ShiftRows 20
4.3.3 MixColumns 20
4.3.4 AddRoundKey 21
4.3.5 Implementation Results 21
4.4 Previous Designs on BRAM Approach 21
4.5 Drawbacks of Conventional BRAM Approaches 22
4.6 Our Resource Efficient SubBytes Transformation 22
4.6.1 Architecture of Our Resource Efficient SubBytes Transformation 22
4.6.2 Implementation of the Proposed System 26
4.6.3 Implementation Results 28
4.7 Our AES Iterative Architecture Based on Resource Efficient SubBytes Transformation 28
4.7.1 Implementation Results 28
4.8 Summary 29
5 Results 31
5.1 Results of Implementation of S-Box Based AES 31
5.2 Results of Implementation of Resource Efficient SubBytes Transforamation 32

5.3 Results of Our AES Iterative Architecture Based on Resource Efficient SubBytes Transformation 33

6 Conclusion and Future Work 35

6.1 Conclusion 35

6.2 Future Work 35

6.3 Publication 36

Bibliography 37

List of Figures

Figure 2.1	State	4
Figure 2.2	AES Encryption	5
Figure 2.3	SubBytes	6
Figure 2.4	ShiftRows	6
Figure 2.5	AddRoundKey	8
Figure 2.6	AES Decryption	9
Figure 3.1	XILINX SPARTAN-3 Family Architecture	13
Figure 4.1	S-box in Hexadecimal format	15
Figure 4.2	Architecture of AES Implementation	18
Figure 4.3	2 S-Box per BRAM	20
Figure 4.4	Resource Efficient S-Box Architecture	24
Figure 4.5	Timing Diagram	27
Figure 4.6	Optimization of ShiftRows Transformation	29

List of Tables

Table 2.1 Number of Rounds as a Function of Key Size 4

Table 5.1 Comparison of Results of Implementation of S-Box based AES 32

Table 5.2 Comparison of Results of Resource-Efficient SubBytes Transformation 33

Table 5.3 Comparison of Results of Our AES Iterative Architecture Based on Resource Efficient SubBytes Transformation with Results of Previous Works 34

1 Introduction

1.1 Background

Every day we come across examples of how our world is increasingly becoming automated. The more we rely on gadgets in our daily lives the more we communicate electronically. Today, professionals in the fields of medicine, commerce, engineering, human resource, social sciences and liberal arts; all store their work on computers and they all use computers to communicate with their peers. Cryptography is an essential need of this overwhelming and ongoing use of electronic devices for storage and communication of data. Email, cellular communications, secure web access and digital cash are just a few examples of daily life activities that require cryptography.

Cryptography includes encryption and decryption. The purpose of cryptography is to achieve confidentiality, integrity, authenticity, availability and non repudiation. Encryption and decryption are both done by following a certain predefined algorithm and using a 'key'. Encryption is the process of converting useful data, referred to as 'plaintext', into meaningless data, called 'ciphertext'. Decryption is the process of converting the ciphertext back into the plaintext. Cryptography is of two types; Symmetric cryptography and Asymmetric cryptography. Symmetric cryptography, also known as secret key cryptography, uses the same key for both encryption and decryption. Asymmetric cryptography, also known as public key cryptography, uses a separate key for encryption and decryption.

In 1997, the National Institute of Standards and Technology initiated a process to choose an Advanced Encryption Standard (AES) [1]. The standard used at that time was Data Encryption Standard (DES), which is now vulnerable to attack by key exhaustion [2]. NIST received significant feedback and interest from public interested in cryptography. In 1998, NIST announced the acceptance of fifteen candidate algorithms. After reviewing the security and efficiency of these algorithms, five were chosen for the final round of selection [3]. After further public analysis, NIST announced Rijndael cipher algorithm as the AES, on October 2,

2000 [4]. On November 26, 2001, NIST announced that AES was approved as Federal Information Processing Standards Publication 197, 'FIPS PUB 197' [5].

The choice of platform to use for implementation of this standard, in this thesis, is the Field Programmable Gate Array, FPGA. Other options are Application Specific Integrated Circuit, ASIC and Software only. Software is a low cost option, with a lot of flexibility in changing or upgrading one's application. But software does not deliver good performance in terms of speed. ASICs on the other hand are better in performance than software, but they are very high in cost and lack flexibility completely. FPGAs, our choice for implementation for AES, provide maximum flexibility and speed with low costs.

1.2 Aim of Thesis

The aim of this research is to implement the AES on Xilinx FPGA by utilizing the FPGA's dedicated embedded memories to store the S-box in a fully area-optimized way.

1.3 Thesis Outline

The rest of the chapters of this thesis are organized as follows:

Chapter 2 gives an introduction of symmetric cryptography and AES. AES encryption is explained in detail. Chapter 3 covers details of the FPGA and its advantages. The architecture and function of the Spartan 3 FPGA family of Xilinx is also discussed. Chapter 4 gives the complete detail of achieving the aim of this thesis. It highlights the architectural and software optimization in this thesis and also gives description of our proposed technique for efficient utilization of BRAM in SubBytes transformation. Chapter 5 presents a comparison of the results of our work with previously published works. Chapter 6 gives the conclusion and future work ideas.

2 Symmetric Cryptography

2.1 Introduction

Symmetric cryptography or symmetric key cryptography is a popular cryptographic technique. It uses a single private or secret key to both encrypt and decrypt the data. Since, the key at the sender and receiver end is same, that's why the name 'symmetric'. Symmetric cryptography has a number of advantages over asymmetric cryptography. It is relatively inexpensive to produce a strong key for these ciphers. The keys tend to be much smaller for the same level of protection [6]. Symmetric cryptography algorithms are relatively inexpensive to process [7]. Implementation of symmetric cryptography is also quite fast. Since the key is secret, the same publicly known algorithm can be used by all parties always, and no effort has to be made to develop new or secret algorithms.

As long as the key remains secret with the sender and receiver, symmetric cryptography provides authentication as well as confidentiality. However, it is evident that incase the key is compromised, any third party can not only decrypt confidential encrypted data, but also encrypt any new message and send it as if it came from one of the parties originally authorized to use the key. So it is crucial that the key is exchanged securely between the authorized parties.

One disadvantage of symmetric cryptography is that it has to be ensured that both sender and receiver have the same key. For this there has to be a way that they exchange a key. One way is to encrypt that key with another key, but then they must know this second key for decryption to obtain the first key. This leads to a vicious cycle of depending on keys.

Symmetric cryptography is classified into two types, based on how the data is encrypted; 'Block cipher' and 'stream cipher'. Block ciphers take the whole plaintext and divide it into n-bits blocks, and then encrypt each block individually. Whereas the stream ciphers encrypt the plaintext on the fly, in where plaintext is encrypted bit (or byte or word) by bit (or byte or word). So, stream ciphers do not need the whole plaintext before encrypting.

Popular symmetric ciphers are AES, Twofish, RC2, IDEA, Serpent, CAST5, DES and 3DES.

2.2 Encryption Using Advanced Encryption Standard

The Advanced Encryption Standard (AES) [5] is a symmetric block cipher. It takes the plaintext and breaks it up into blocks of 128 bits. The key sizes allowed by this standard are 128, 192 and 256 bits. Each block of 128 bits is arranged in a 4 x 4 array of bytes, as shown in figure 2.1. This is called the 'State'. A single State has 16 bytes. The cipher repeats a round of four transformation Nr number of times, where Nr is a function of the key size.

S

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Figure 2.1 State

Table 2.1 Number of Rounds as a Function of Key Size

Key Length	128 bits	192 bits	256 bits
Number of Rounds (Nr)	10	12	14

AES, the most widely used encryption cipher [8], subjects the plaintext to some simple transformations to get the ciphertext. These transformations are explained in detail in the following sections. The transformations implemented in AES are AddRoundKey, SubBytes, ShiftRows and MixColumns. The first transformation applied on the State is AddRoundKey. After this a round is implemented in the order of first SubBytes, then ShiftRows, then MixColumns and then AddRoundKey. This Round is repeated 'Nr' number of times as shown in table 1. However, the Nr^{th} round omits the MixColumns transformation. This encryption flow is shown in figure 2.2.

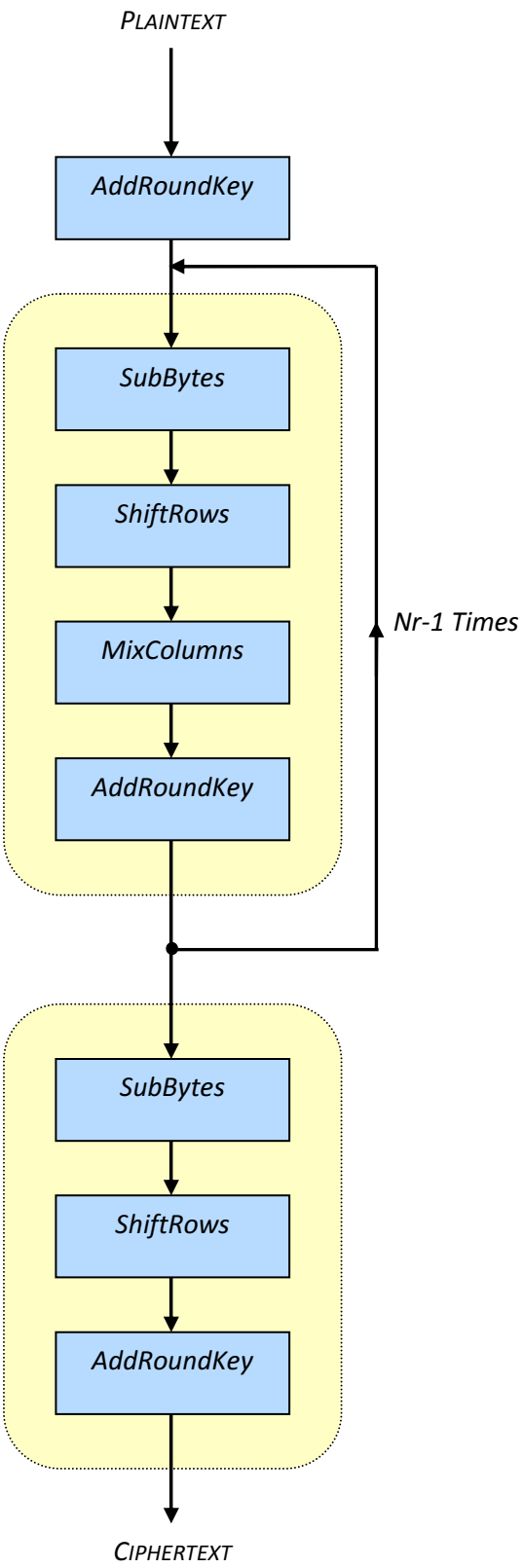


Figure 2.2 AES Encryption

2.2.1 SubBytes

The SubBytes transformation is a non linear transformation that operates on the state byte by byte. Each byte of the state is substituted by a byte from a pre defined look up table. The values of this pre defined table are calculated by performing two transformations, a multiplicative inverse in the finite field GF (2^8), and a standard affine transformation (over GF (2)). If the value of the byte to be substituted by another byte, is '01' for example, then the substitution value would be the byte at the intersection of the row with index '0' and the column with index '1' in the S-box. Figure 2.3 depicts the SubBytes transformation.

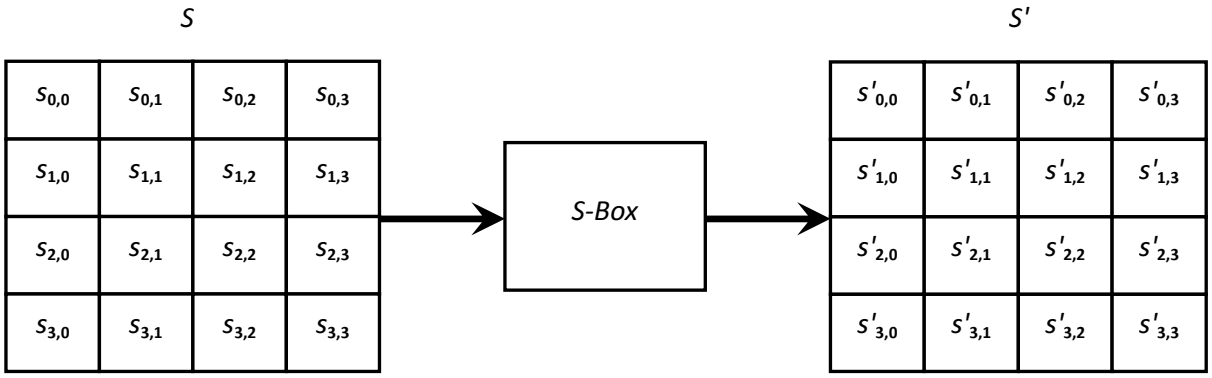


Figure 2.3 SubBytes

2.2.2 ShiftRows

This is a transposition stage where each row of the state is shifted cyclically a certain number of times. The first row is not shifted at all. The second row is rotated to the left by one byte, third row by two bytes and fourth row by three bytes. This transformation ensures that the four bytes of one column are spread out to four different columns. In figure 2.4, this shift is depicted by the shifted subscripts in the state S' on the right.

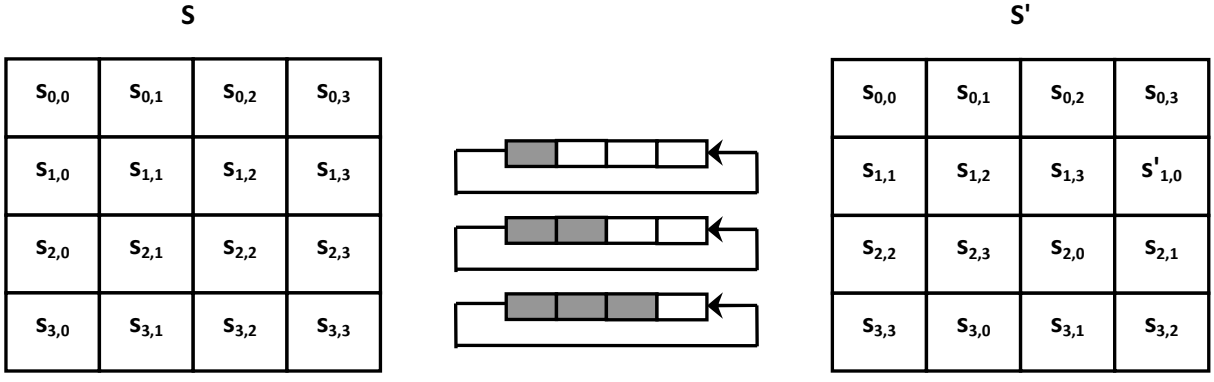


Figure 2.4 ShiftRows

2.2.3 MixColumns

The MixColumns transformation operates on every column of the state such that each byte of a column is mapped into a new value that is a function of all four bytes in that column. Each column is considered as a four-term polynomial over GF (2^8) and multiplied modulo x^4+1 with the fixed polynomial c(x):

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

The above can be written in terms of simple matrix multiplication.

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

As a result of this multiplication the four bytes in a column are replaced by the following equations, where '$\bullet$' is simple multiplication and '$\oplus$' is bit-wise XOR:

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

2.2.4 AddRoundKey

This transformation adds a Round Key to the State by simple bitwise XOR operation. The round key for each round is derived from the cipher key or from the previous round key. Each round key is 128 bits long. The AddRoundKey transformation is depicted in figure 2.5.

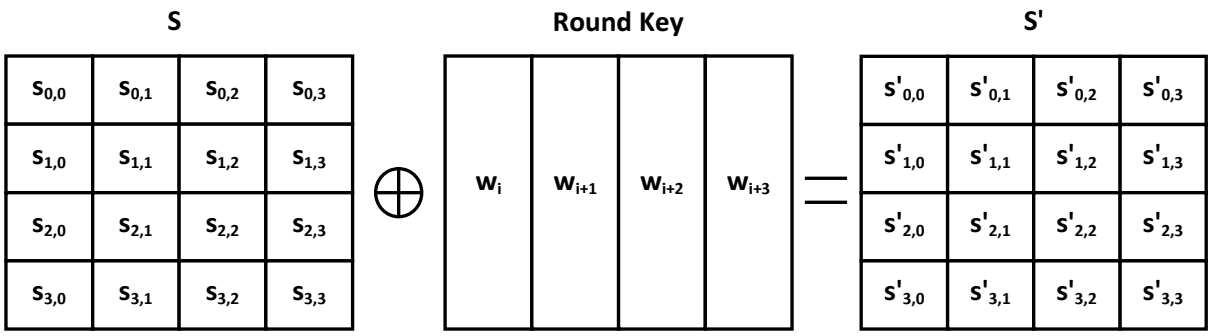


Figure 2.5 AddRoundKey

2.3 Key Expansion

For AES-128, ten round keys are needed in addition to the cipher key. Each round key is derived from the last four bytes of the previous key. The process used for generation of round keys is called key expansion routine. This routine includes an XOR step, a substitution transformation and a cyclic permutation. The same key schedule (set of round keys) is used for encryption and decryption.

2.4 Decryption

The decryption algorithm for AES basically involves inverse of the transformations done in encryption. Decryption for AES 128 is done by performing an AddRoundKey transformation, and then performing nine rounds of AddRoundKey, InverseMixColumns, InverseShiftRows and InverseSubBytes. The tenth round then includes InverseShiftRows, InverseSubBytes and AddRoundKey only.

AddRoundKey uses keys in the reverse order, as that of the key schedule for encryption. For InverseShiftRows, the direction of the shift is changed, while the number of shifts for each row remains the same as in encryption. An Inverse S-box is used to perform the substitutions in InverseSubBytes. InverseMixColumns is also done in the same way as the MixColumns transformation in encryption; however the coefficients of the polynomial are different here and subsequently, so are the values in the matrix. This decryption flow is shown in figure 2.6.

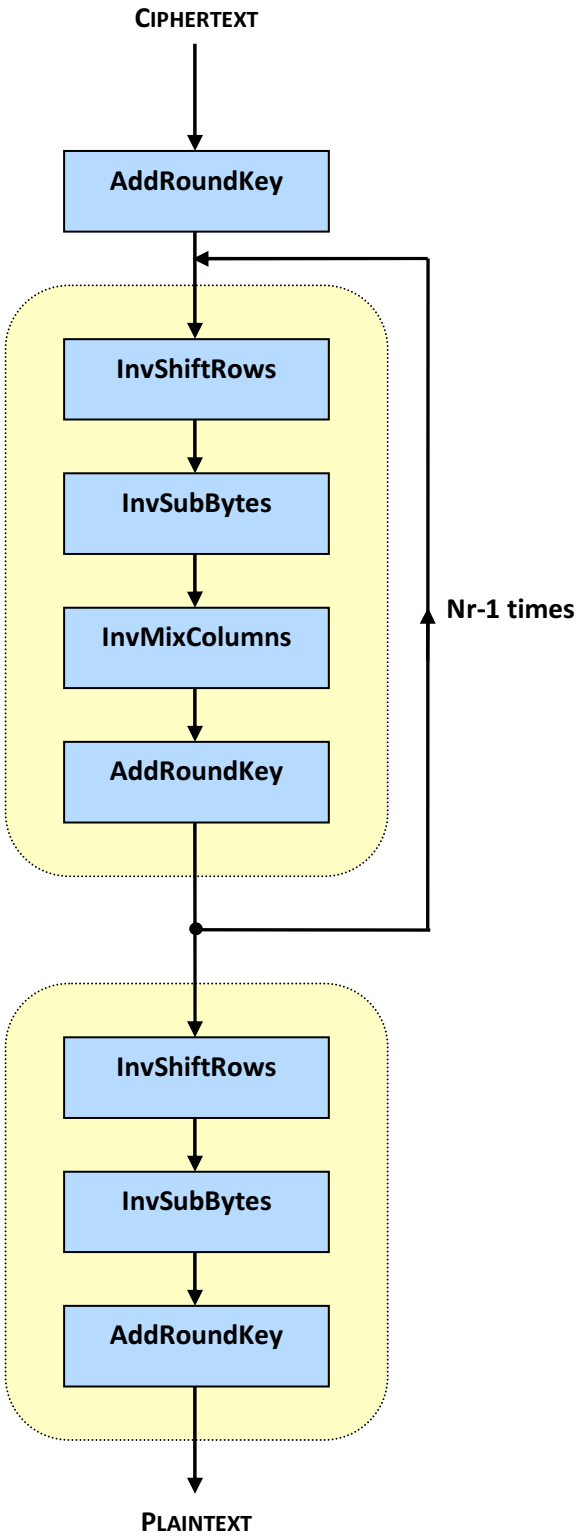


Figure 2.6 AES Decryption

2.5 AES Key Sizes

There are three key sizes allowed in the AES; 128-bit, 192-bit and 256-bit. The strength of the encryption increases with the increase in key size [9]. However, this also increases the computational and memory requirement for the encryption process [10].

2.6 AES in Practice

AES is widely used for encryption at private, commercial and government levels [11]. Given its excellent security capability, the National Security Agency (NSA) of US has set AES as the standard for national security information [12]. It is used by BlackBerry services to encrypt and decrypt data [13]. AES is also used for encrypted compression by WinZip [14]. It has also been incorporated into Wireless 802.11n standard [15].

2.7 Summary

In this chapter symmetric cryptography and its types were describes with detailed explanation given for Advanced Encryption Standard (AES). The transformations that form the AES encryption process were then explained in sequence. The decryption process of AES, the inverse of AES encryption, was briefly explained. At the end, the use of AES in the industry was illustrated.

In the next chapter, FPGA, its advantages and its families are explained. The Spartan-3 FPGA used in this research, its features and the software tool used to program it are also explained in the sub-sections of the chapter.

3 FPGA

3.1 Introduction

A Field Programmable Gate Array (FPGA) is a programmable integrated circuit that includes a two dimensional array of logic blocks. The design of an FPGA is standardized by its manufacturers, but its function is defined by the programmer solely. So, an FPGA is a reconfigurable device which instantly takes on a new function whenever we configure it with a new code.

3.2 Advantages of FPGA

An FPGA combines the best parts of hardware and software. It is cheaper and easily upgradable than Application Specific Integrated Circuits (ASICs). A new idea or concept can be tested and verified very quickly on an FPGA, instead of going through a long fabrication process which is required for ASICs.

The cost of upgrading an FPGA system is negligible when compared to the cost of making changes to ASICs. Compared to a software implementation, an FPGA implementation offers more processing speed and demands lesser power. Owing to these advantages, most of the current cryptographic modules rely on FPGA implementations [16].

3.3 XILINX FPGA

The platform we choose to work on is the Xilinx FPGA. The reason of this choice is that for AES, the majority of research results, which we came across, are on Xilinx Platform. Also Xilinx FPGA is supported by software resources available in the college. Moreover, Xilinx is the choice of researchers in other institutes and industry in Pakistan. This allowed us to share expertise. Xilinx holds the larger part of the international programmable market place [17] [18]. Xilinx has many silicon devices that one can choose from. Our choice for our area-efficient design is the Spartan-3 FPGA.

3.4 Spartan-3 FPGA Family

Various FPGA families differ in their sizes, in the way flip-flops and LUTs are packaged together, in their embedded features and in the performance they offer. The Spartan-3 family of FPGAs is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications; including broadband access, home networking, display/projection and digital television equipment [19].

The architecture and function of this family of FPGA are briefly discussed in the following sections.

3.4.1 Architectural and Functional Overview

The Spartan-3 family architecture consists of five fundamental programmable elements:

- Configurable Logic Blocks (CLBs)
- Input/Output Blocks (IOBs)
- Block RAM (BRAM)
- Multiplier blocks
- Digital Clock Manager (DCM)

These logic blocks can be connected to each other by a programmable interconnect architecture. FPGAs also have other specialized blocks, such as Block Random Access Memories (BRAMs) and Digital Signal Processors (DSPs). These specialized blocks perform many flexible yet specific tasks, and provide a lot of ease to the programmer.

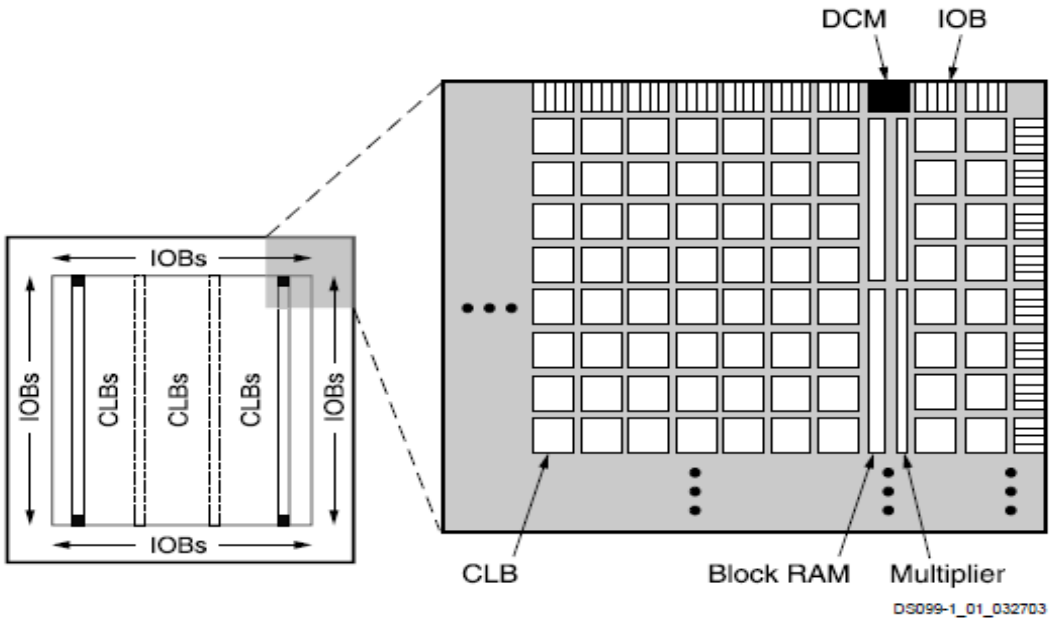
3.4.1.1 Configurable Logic Block (CLB)

Each CLB is composed of four interconnected slices. Each slice has two logic function generators, two storage elements, wide-function multiplexers, carry logic and arithmetic gates. The function of these logic blocks is not specified by the manufacturer;

instead it is programmable by the user. CLBs can be programmed to perform some logical function or to store data.

3.4.1.2 Input/Output Block (IOB)

The CLBs are surrounded by configurable IOBs, as shown in Fig 3.1. The input/output blocks provide a bidirectional connection between these blocks and the FPGA's internal logic. IOBs are used to drive any signal onto or off the FPGA. So, these are connection between external devices in the outside world and the FPGA. Each IOB supports bidirectional data flow plus 3-state operation.



Notes:
1. The two additional block RAM columns of the XC3S4000 and XC3S5000 devices are shown with dashed lines. The XC3S50 has only the block RAM column on the far left.

Figure 3.1 XILINX SPARTAN-3 Family Architecture [19]

3.4.1.3 Block RAM (BRAM)

The Block RAM is a salient feature of all Spartan-3 FPGAs. The Block RAM is organized as reconfigurable, synchronous 18kbit blocks. The Block RAM effectively provides storage for large amounts of data; and also offers configurable aspect ratio i.e. width vs. depth. Multiple Block RAMs can be cascaded in order to increase the total addressable locations. The Block RAM has a true dual port structure, with each port having its own dedicated set of data, control and clock lines for synchronous read and write operations.

The number of Block RAMs in one FPGA varies depending on the size of the device. The smallest Spartan 3 FPGA, XC3S50 has 4 Block RAMS, whereas the largest one, XC3S5000 has 104 Block RAMs.

3.4.1.4 Multiplier Blocks

All Spartan-3 FPGAs have embedded multipliers that accept two 18-bit words as inputs to produce a 36-bit product. The input buses to the multiplier accept data in two's-complement form (either 18-bit signed or 17-bit unsigned). Cascading multipliers permits multiplicands more than three in number as well as wider than 18-bits.

3.4.1.5 Digital Clock Manager (DCM)

The DCM is a valuable embedded feature, as it provides control over clock frequency, phase shift and skew. The three main functions supported by DCM are clock-skew elimination, frequency synthesis and phase shifting.

3.5 XILINX ISE

ISE is the central design suite, provided by Xilinx, for studying, simulating and automating designs for FPGAs. ISE Design Suite provides many features, including design entry and synthesis. It supports Verilog/VHDL, design verification and debug tools and creation of bit files for configuring the chips. Xilinx ISE provides an ideal software-based platform to examine and test a design for varying different inputs. It also helps to configure the target device using a programmer. In this thesis Xilinx ISE 10.1 is used [20].

3.6 Summary

In this chapter Field Programmable Gate Array (FPGA) was introduced and its various advantages were described. Xilinx Spartan-3FPGA was then described and its various components and features explained, with emphasis on those that were used in this research.

In the next chapter, the core of this report is presented.

4 Our Work

4.1 Overview

FPGAs are a popular choice for cryptographic implementations because they combine the ease and flexibility of software with the speed and computational power of hardware. Initially, in the field of cryptography, the implementation of AES was verified on FPGA. Then the efforts were directed at improving the speed of these cores implemented on FPGA. Gradually, researchers started focusing on minimum FPGA resource utilization.

Among the transformations of AES, AddRoundKey involves an XOR operation only and ShiftRows is only a cyclic shift of rows. So, no optimization needs to be done for these two transformations. The SubBytes is the only non linear transformation that operates independently on each byte of the state. It is also the most intensive in terms of resource utilization [21]. The S-box is presented in hexadecimal form in figure 4.1 [5].

	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

Figure 4.1 S-box in Hexadecimal format

The aim of our work is to design an area-efficient encryption AES core, with emphasis on the SubBytes transformation. In addition to developing an area-efficient technique for implementing the SubByte transformation, we also chose the iterative implementation for the AES core on FPGA, for better resource utilization.

There are two ways of implementing the SubBytes round, as described in the following sections.

4.1.1 Conventional Technique of Implementing SubBytes Transformation

The conventional method used in implementing SubBytes Transformation is to calculate the S-box on the fly by performing two transformations. These transformations include; taking a multiplicative inverse in the finite field GF (2^8), and applying a standard affine transformation (over GF (2)). The computation of this multiplicative inverse, though hardware-demanding, can be divided into multiple sub stages to improve the frequency of the encryption core.

4.1.2 BRAM Technique of Implementing SubBytes Transformation

The second method to implement SubBytes transformation is to directly store pre-calculated S-box values in a lookup table (LUT) and then access the required values with their respective addresses in the LUT. However, in this approach the delay of memory access is unavoidable. The possible memory storage choices can be; configuring the FPGA slices as distributed RAM, or using embedded BRAMs [22] of the device.

When S-box is implemented using the FPGAs as LUT only, it utilizes more than 75% of the resources [23]. The state consists of 16 bytes, each of which has to be substituted by a byte from the S-box, in each SubBytes transformation. So the S-box has to be accessed 16 times for every SubBytes transformation. Therefore, 16 instances of the S-box have to be hardwired for every SubBytes transformation. The size of one S-box is 256 (16 x 16) bytes. So it takes only 2kbits for storage. But the S-box replicated 16 times, for just one round, requires considerable amount of memory. In addition to this, the key expansion routine also requires accessing the S-box four times for every round key.

Our SubBytes round has been implemented using the embedded BRAMs in dual port configuration and Read Only mode. This is a simple LUT approach. By using BRAMs we

have avoided all the calculations needed for determining the S-box values. Efficient access of LUT demands utilization of large embedded memories.

4.2 Choice of Architecture

The choice of architecture to implement AES has an impact on the resource utilization and the throughput. The resource utilization refers to; area of the cryptographic unit, utilized to implement the cipher, which in our case is AES. The throughput of the cryptographic unit to encrypt/decrypt the plaintext is measured by the number of bits encrypted or decrypted per unit of time (second).

The architectures that offer varying area utilization and throughput are pipeline, sub pipeline and loop unrolled architectures. All these architectures offer different area and speed tradeoffs. We made our choice of architecture in accordance with our requirement of resource efficiency. Iterative architecture, a form of loop unrolled architecture, offers the least resource utilization at the expense of speed. Since, our aim is to propose a compact architecture for small and cost effective implementations of AES; we opted to implement the iterative architecture. All these three architectures are discussed below.

4.2.1 Pipeline Architecture

Pipelining inserts rows of registers between each round unit. After an initial delay, a fixed number of blocks are processed simultaneously. This increases the speed significantly at the cost of increase in resource utilization.

4.2.2 Sub Pipeline Architecture

Sub pipelining inserts registers inside the round units also. Like this each round is divided into smaller segments, with registers between rounds and inside rounds. Sub pipelining provides little increase in speed with considerable increase in resource utilization. In this architecture, although more blocks of data are processed simultaneously, but the average number of clock cycles to process one block also increases.

4.2.3 Loop Unrolled Architecture

In this architecture, multiple rounds are performed in each clock cycle, but only one block of data is processed at a time. A loop unrolled architecture that performs only one round in one clock cycle is an ***Iterative architecture***. Iterative architecture is a subset of loop unrolled architecture.

4.3 Our AES Implementation Based on LUT Approach

This efficient and compact, iterative architecture of AES Encryption core was implemented using the S-Box Approach. My colleagues, Dur e Shahwar and Qurat-ul-Ain; and I developed an iterative implementation of the AES encryption. The resources that are used for the first round are then reused for all subsequent rounds, thus saving on device resources significantly.

In this work we have developed an iterative architecture as shown in figure 9. Figure 2 may be referred to, for easy understanding of this proposed iterative architecture. Our cipher key was of length 128. So, in addition to the first AddRoundKey transformation, 10 rounds had to be implemented.

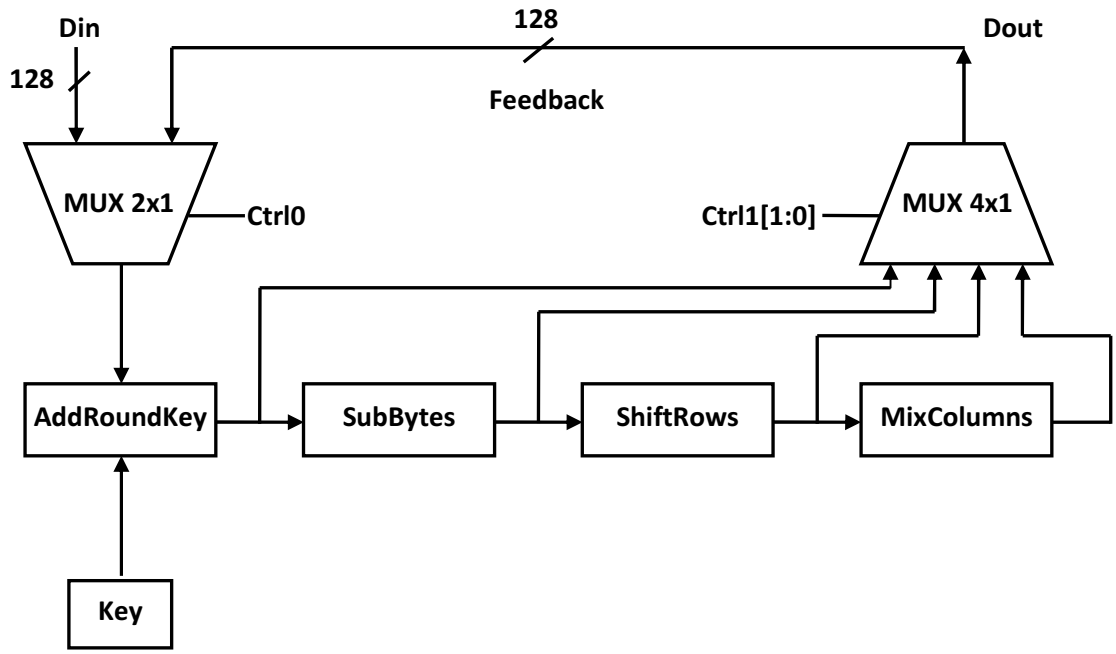


Figure 4.2 Architecture of AES Implementation

In our implementation of the AES, the required intermediate states are fed back through a MUX to the encryption core. The 2x1 MUX selects the input data for the initial round and then onwards it selects the feed-back data for all remaining rounds.

As seen in figure 2.2, flow diagram for AES encryption, there are three blocks in total for the AddRoundKey transformation. AddRoundKey is the first transformation to be performed in AES encryption. In the following 10 rounds it is then the last transformation. In our AES architecture, we have saved resources by using only one AddRoundKey. After implementing the first AddRoundKey transformation, we used the resources of this AddRoundKey block for all the subsequent transformations of AddRoundKey. This was achieved by sending the AddRoundKey output to the feedback at appropriate stages during the encryption, by using the 4 x 1 multiplexer as shown in figure 4.2.

Similarly, in figure two, there are two blocks each, for the SubBytes and ShiftRows transformation. Again, we made use of multiplexers to keep resource utilization to a minimum, by selecting the required transformation output and feeding it back into the round. The output for SubBytes is never selected as a feedback in a round. However, it is fed into the multiplexer to conform to the required 4 inputs.

The logical flow of our AES implementation is given in order below:

1. First DIN enters the AddRoundKey. It then goes through all the other transformation blocks and at the end the output of MC is fed back

2. The fed back state undergoes AddRoundKey transformation. After going through the subsequent transformations, the output of MixColumns is again fedback. This process is repeated eight times.

3. In the next iteration, to skip the mixcolumns transformation, the ShiftRows output is fed back.

4. After that the AddRoundKey output is fed back in the last round.

In this way, the complete AES encryption process was implemented without duplication of resources for any transformation. Each stage was implemented and tested as an individual module. The instantiations of these modules were used in the main code of our design. The resources used for the first round are then reused for all subsequent rounds, thus saving on device resources significantly.

4.3.1 SubBytes

In our design two S-boxes are stored per BRAM as shown in figure 4.3. Thus, 8 BRAMs are needed for each round.

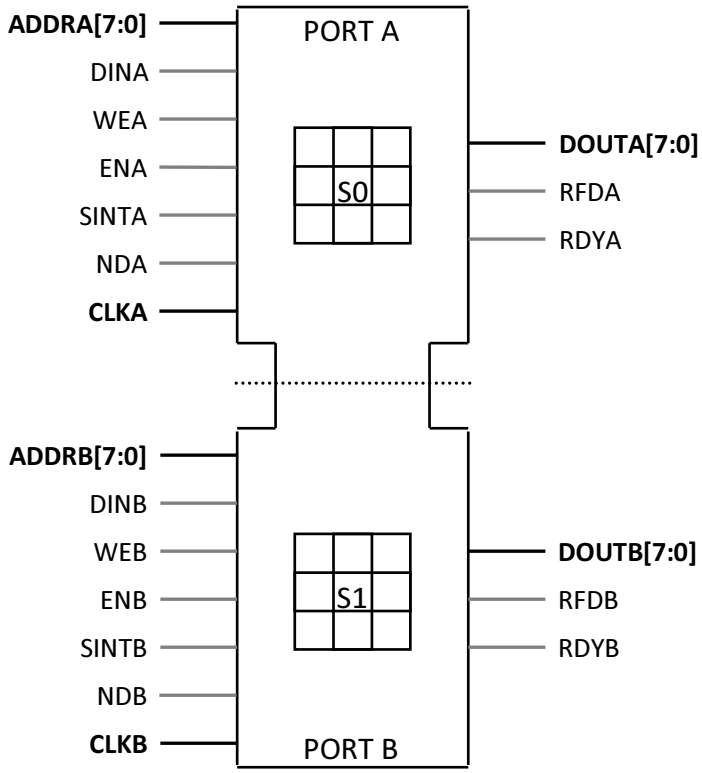


Figure 4.3 2 S-Box per BRAM

4.3.2 ShiftRows

ShiftRows transformation is a simple operation of shifting each row by a fixed number of steps. We achieved this by simply assigning the bytes from input (of ShiftRows round) state to output state.

4.3.3 MixColumns

The MixColumns transformation is done by multiplication modulo x^4+1, in the Galois field (2^8). The input state of this round is multiplied by a constant matrix to obtain the output state. For encryption, multiplication of the bytes is done with the constants 2 or 3 only. Multiplication with 2 is done by left shift of each byte. Multiplication with 3 is done as a sum of the byte and its product with 2.

4.3.4 AddRoundKey

In the AddRoundKey transformation, the already calculated round keys are stored in registers. For every round, the key is accessed and bitwise XOR operation is done with the state.

4.3.5 Implementation Results

We implemented our compact LUT-based AES design on Spartan-3(XC3S4000), using 8 out of the total 96 BRAMs available (8.33% BRAM usage), while occupying 390 out of 27,648 number of slices. The number of four input LUTs were 627 out of 55,296 and the number of slice flip flops utilized were 405 out of 55,296; with a maximum output frequency of 206.28 MHz. Henceforth, providing an excellent foundation for our ensuing work.

4.4 Previous Designs on BRAM Approach

BRAMs are the dedicated embedded memory blocks of an FPGA. In the BRAM of a Xilinx FPGA, only one memory location can be accessed per port per clock cycle. BRAMs are ideal for implementing SubBytes transformation, which involves accessing the S-Box. Since BRAM allows just one read operation per clock cycle, the first BRAM implementation accommodated only one S-Box per BRAM in a Single Port configuration. After that Dual Port configuration is used that allows synchronous read operations on the BRAM. This makes it possible that the stored S-Box is accessed twice in one clock cycle.

During our research, various implementations of S-Boxes were studied with varying degree of memory utilization, area occupied and throughput. The work, relevant to this thesis is mentioned here and is also presented in Chapter 5 - Results. In the works of I. Algredo-Badillo et al. [24], J. Zambreno et al. [25] and Swankoski et al. [26] 8 BRAMs were used in dual port configuration. In each clock cycle two substitution bytes were read from the S-Box in the BRAM. Therefore, in order to obtain 16 substitution bytes for the complete state, 8 BRAMs were used.

In E. Lopez-Trejo et al.'s [27] work 16 BRAMs were used, each storing an S-Box. For a single SubByte transformation each BRAM contributed a substitution byte, corresponding to the 16 bytes of the state. Realizing the importance of area-efficiency and its

subsequent impact on the cost and size of the core, in the most recent work of Arshad Aziz et al. [28], 4 BRAMs were utilized to achieve SubBytes transformation.

4.5 Drawbacks of Conventional BRAM Approaches

Block Memories in Spartan-3 and Virtex-II Pro Series FPGAs are 18 Kbits; 16 Kbits (2000 bytes) for data and 2 Kbits for parity. The size of an S-Box is 256 bytes. The aforementioned conventional techniques for the implementation of SubBytes transformation are not fully area-efficient, implying that each BRAM is not utilized fully or near to its maximum available space.

As a result, all these techniques use more than the minimum number of BRAMs needed to implement the SubBytes transformation. This results in higher memory requirement for the AES implementation and hence more area is required and that too at a higher cost. In this thesis, we have proposed a novel clocking technique that reduces this usage to only 2 BRAMs, hence reducing the area and cost of the AES implementation.

4.6 Our Resource Efficient SubBytes Transformation

We have proposed a technique that allows eight S-Boxes to be accommodated in one BRAM. Its architecture and design implementation are explained in this section.

4.6.1 Architecture of Our Resource Efficient SubBytes Transformation

To have an area-efficient design we have used the following embedded features of Xilinx FPGA:

i. Digital Clock Manager (DCM)

ii. Block RAM (BRAM)

The DCM [29] has been used to access the BRAMs at four times clock speed than the rest of the system. Verilog and VHDL code templates for generating a DCM are available in the Xilinx ISE. Another method for generating DCM is to use the clocking wizard. The clocking wizard has a front end that allows the user convenient and quick generation of DCM. Main system clock, CLK is sent as an input to the DCM. Main system clock is the one at which all other AES transformations are performed.

We have used the DCM to generate a clock *CLK0*, *CLK2X* and *CLKFX*. *CLK0* has the same frequency and phase as *CLKIN*. *CLK0* is also conditioned to 50% duty cycle because we have enabled the duty cycle correction feature of the DCM in this work. *CLK2X* is a clock signal that has a frequency twice that of *CLK0*. *CLKFX* is configured to have a frequency four times that of *CLK0*. This has been done by using 'multiply value (M)' as 4 and 'divide value (D)' as 1. Like this we get an output clock, *CLK4X*, which has four times the frequency of the input clock. *CLK0*, *CLK2X* and *CLK4X* (*CLKFX*) are all in phase with each other.

Like the Clocking Wizard, the Block Memory Generator is a useful feature of Xilinx ISE. We used it to generate a block memory that had all parameters according to the requirements of our design. We have configured the BRAM as a Dual Port ROM. This means that both ports can be read simultaneously. Thus, allowing us to access two memory locations per clock cycle. Our design, shown in figure 4.4, uses two BRAMS to implement the SubBytes transformation.

The S-Box is stored in a coefficient (COE) file, which is loaded in the BRAM to initialize its memory locations. The S-Box stored in the COE file is in the form of a lookup table of width 8 and depth 256. The addresses of this table correspond to the rows and columns of the S-box and the values stored against these addresses correspond to the values of the S-box.

In the Dual Port ROMs, each port functions as an individual ROM, Port A and Port B, for the first BRAM and Port C and Port D, for the second BRAM. We have to give two inputs to each port. An address that has to be read and a clock signal. Each port will be synchronous to its own clock. The output of DCM, *CLK4X*, is used as an input to *CLKA*, *CLKB, CLKC* and *CLKD* of the dual port BRAMs. Thus, both the ports of both BRAMs are synchronous with *CLK4X*. This means that we can access the S-box four times in one system clock and consequently get four outputs from each port of the BRAM, in one system clock.

Our proposed resource-efficient module produces four outputs from each port of the two BRAMs, in one CLK0. This means that we get eight substitution bytes from one BRAM, in each CLK0 cycle. So, using two BRAMs operating synchronously makes a total of sixteen outputs per system clock.

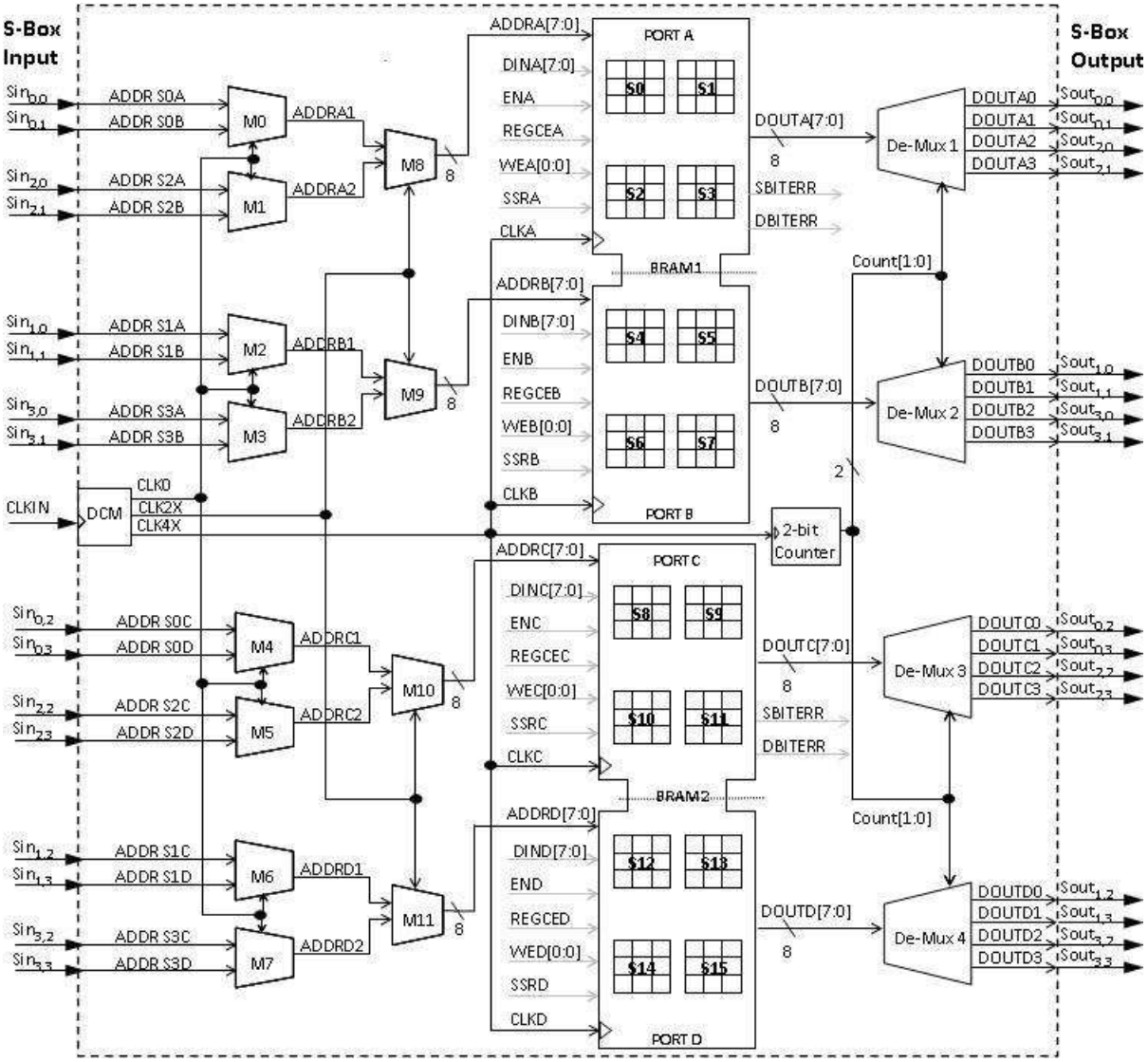


Figure 4.4 Resource Efficient S-Box Architecture

In our design, the inputs to the *ADDRA*, *ADDRB*, *ADDRC* and *ADDRD,* is a byte of the state. This input is changed at every rising edge of *CLK4X*, and is used as an address of the lookup table we loaded in the BRAM initially. From this we get the value in the lookup table at that address, *DOUTA*, *DOUTB*, *DOUTC* and *DOUTD* after clock to out time of the BRAM. This value is then registered at its correct location in the state. How are *ADDRA*, *ADDRB, ADDRC* and *ADDRD* changed at every *CLK4X* in our design? This is explained in detail in the following content.

As shown in figure 4.4, the addressing circuit of the BRAM1 comprises of a DCM and six 2x1 multiplexers. The first stage of multiplexers includes the multiplexers *M0*, *M1*, *M2* and *M3*. The select signal for these multiplexers is the *CLK0*. The second stage includes multiplexers *M8* and *M9*. The select signal for *M8* and *M9* is the *CLK2X*.

The S-Box module shown in figure 4.4, receives a state at every rising edge of *CLK0*. Eight bytes of the state are sent to one BRAM via its addressing circuit, and eight bytes are sent to the second BRAM via its addressing circuit. The complete flow, from input to output of our S-box module; of the eight bytes sent to BRAM1 is explained here in full detail. Two bytes are sent to *ADDR S0A* and *ADDR S0B* of *M0*, two bytes to *ADDR S2A* and *ADDR S2B* of *M1*, two bytes to *ADDR S1A* and *ADDR S1B* of *M2* and two bytes to *ADDR S3A* and *ADDR S3B* of *M3*. When *CLK0* is high, *M0* selects *ADDR S0A*, *M1* selects *ADDR S2A*, *M2* selects *ADDR S1A* and *M3* selects *ADDR S3A*. When *CLK0* is low, *M0* selects *ADDR S0B*, *M1* selects *ADDR S2B*, *M2* selects *ADDR S1B* and *M3* selects *ADDR S3B*. The output of these first stage multiplexers is sent to *M8* and *M9*.

The input that *M8* receives from *M0* and *M1*, and the input that *M9* receives from *M2* and *M3*, both are changed at every change in *CLK0*. When *CLK2X*, the select signal of *M8* and *M9*, is high, *M8* selects *ADDR A1* and *M9* selects *ADDR B1*. When *CLK2X* is low, *M8* selects *ADDR A2* and *M9* selects *ADDR B2*. The outputs of *M8* and *M9* are sent to *ADDRA* and *ADDRB* of the *BRAM*. The changing of inputs of *M8* and *M9* at every change in *CLK0* (i.e. every rising edge of *CLK2X*) and the select signal of *M8* and *M9* being *CLK2X*, both ensure that *ADDRA* and *ADDRB* change at every change in *CLK2X* (i.e. every rising edge of *CLK4X*). All data paths are eight-bit wide as shown in figure 4.4.

When *ADDRA* and *ADDRB* of the BRAM receive an input, it is used as an address to look up the corresponding substitution byte value stored in the BRAM. As already mentioned above, both ports of our BROM are synchronous with *CLK4X*. So, for every change in *ADDRA* and *ADDRB*, we get outputs *DOUTA* and *DOUTB*, after clock-to-out time [30] of the BRAM. These outputs are registered in there appropriate locations in the state.

The outputs of the BRAMs in our S-Box architecture change four times in every *CLK0*. So, to register *ADDRA* at four separate locations we have used a 2-bit counter and de-multiplexer *De-Mux 1*. The counter is incremented four times, synchronously with CLK4X, in every CLK0 cycle. *DOUTA* is sent as an input to *De-Mux 1*. The output value *Count [1:0]* is used as a select signal for assigning *DOUTA* to one of the four outputs of *De-Mux1*. These values *DOUTA0*, *DOUTA1*, *DOUTA2* and *DOUTA3* remain valid for one complete *CLK0* cycle and are registered at their appropriate locations in the state. Similarly, outputs appearing at *DOUTB, DOUTC* and *DOUTD* are also assigned to four separate locations in each *CLK0* cycle.

The architecture and function of the addressing circuit at the input and de-multiplexer at the output of BRAM1 have been explained. Figure 4.4 shows that BRAM2 has a similar architecture, and hence, function also. BRAM2 also receives eight bytes of the state and performs synchronously with BRAM1, to provide eight substitution bytes.

4.6.2 Implementation of the Proposed System

Figure 4.5 shows the timing diagram for implementation of the resource efficient S-box architecture in figure 11. Figure 12 shows the timing diagrams of each port for one CLK0 cycle. *CLK0* is the main system clock and *CLK4X* is the input to *CLKA*, *CLKB, CLKC and CLKD* of Port A, Port B, Port C and Port D respectively, of BRAM1 and BRAM2.

For Port A, at point 0, the synchronized rising edge of *CLK*, *CLK2X* and *CLK4X*; *ADDR S0A* and *ADDR S0B* of multiplexer M0; each receive a byte, $\sin_{0,0}$ and $\sin_{0,1}$ respectively, of the state. *ADDR S0A* ($sin_{0,0}$) is selected when *CLK0* is high, and *ADDR S0B* ($sin_{0,1}$) is selected when *CLK0* is low. *ADDR S2A* and *ADDR S2B* of multiplexer *M1*; each receive a byte, $sin_{2,0}$ and $sin_{2,1}$ respectively, of the state. *ADDR S2A* ($sin_{2,0}$) is selected when *CLK0* is high, and *ADDR S2B* ($sin_{2,1}$) is selected when *CLK0* is low. The outputs of *M0* and *M1* are connected to the address lines of *M8*. *M8* selects *ADDR A1* ($sin_{0,0}$) when *CLK2X* is high and *ADDR A2* ($sin_{2,0}$) when *CLK2X* is low.

The output of *M8* is sent to *ADDRA*, the input of Port A of the BRAM1. So, *ADDRA* changes at every change in *CLK2X* (i.e. every rising edge of *CLK4X*). This *ADDRA* is used as a look up address for the S-Box stored in the BRAM. This address is registered into the memory at next rising edge of *CLK4X*. The substitution value at this address is available at *DOUTA*, after clock to out time of the BRAM. The function of multiplexer set *M2* and *M3* (of Port B), *M4* and *M5* (of Port C), and *M6* and *M7* (of Port D) is similar to the set *M0* and *M1*. The function of *M9, M10* and *M11* is similar to *M8*. At point 1, the inputs to *M0*, *M1, M2, M3, M4, M5, M6, M7* are changed because of a change in the 'state' at every rising edge of *CLK0*.

After we get the *DOUT, DOUT* of each port is sent to four separate locations to be registered so that it is available when needed. These four separate lines are also shown in the timing diagram.

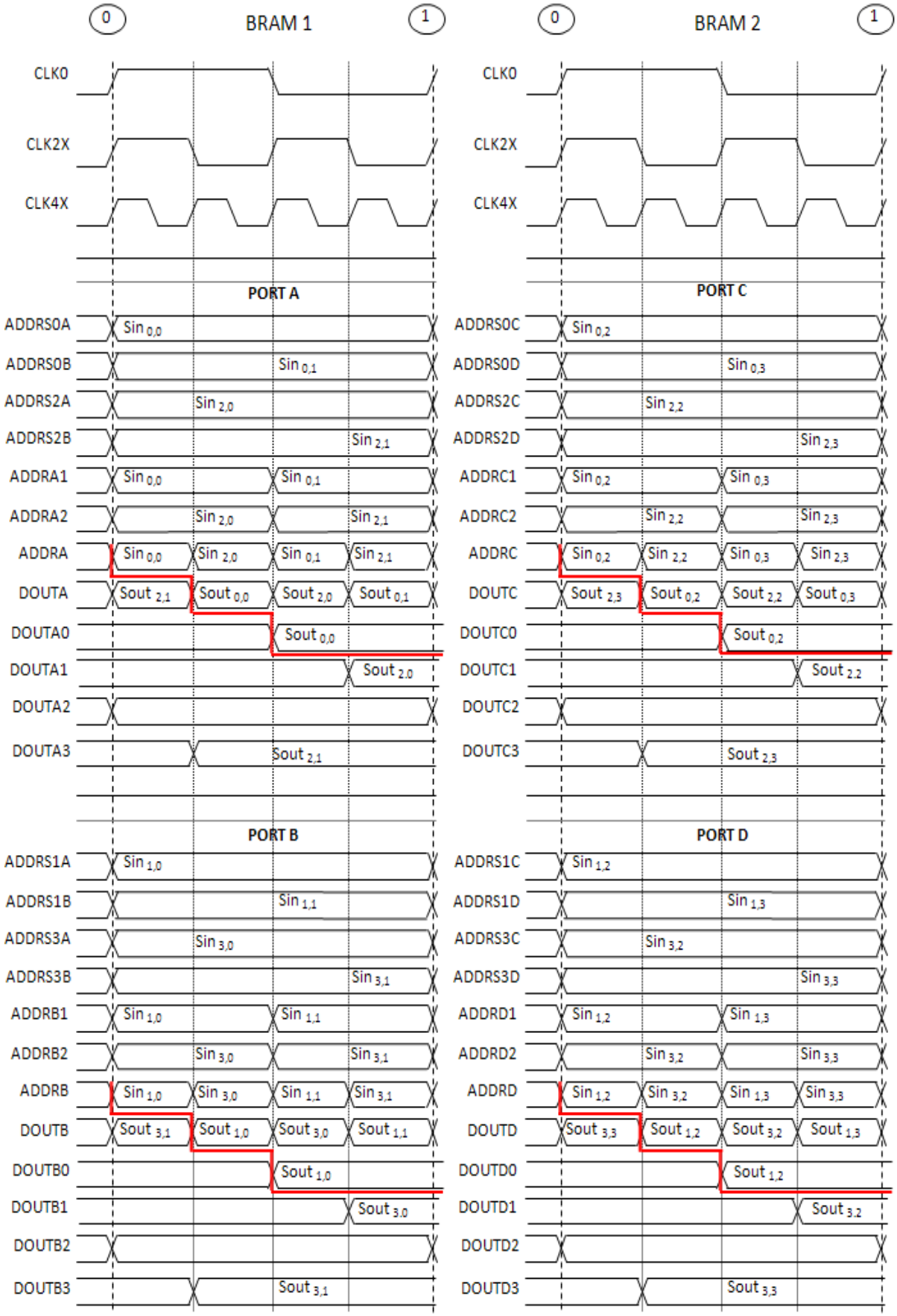


Figure 4.5 Timing Diagram

The timing diagram in figure 4.5 shows all clock, input and output signals for one *CLK0* cycles. In this figure, sixteen substitution bytes are read from the S-Box in BRAM in one *CLK0* cycle, eight bytes from BRAM1 and eight from BRAM2.

4.6.3 Implementation Results

Our resource efficient SubBytes transformation design was implemented on Spartan-3(XC3S4000), using just 2 out of the total 96 BRAMs available (2.08% BRAM usage), while occupying only 194 out of 27,648 number of slices. The number of four input LUTs were only 102 out of 55,296 and the number of slice flip flops utilized were 354 out of 55,296. One out of the four available DCM was used giving a maximum output frequency of 155.198 MHz. These results clearly indicate dramatic improvement in resource utilization and they will be further discussed and compared in Chapter 5.

4.7 Our AES Iterative Architecture Based on Resource Efficient SubBytes Transformation

The iterative architecture rules over other architectures when it comes to area and cost effectiveness. After successfully developing a synchronous clocking technique to utilize two BRAMs only, to implement the resource efficient SubBytes transformation; we incorporated this S-Box module in our iterative AES implementation presented in section 4.5.

This implementation of the AES was improved further by merging the SubBytes transformation and ShiftRows transformation. This was done by assigning the substitution values of the SubBytes transformation to the positions in the output state that corresponds to the output of the ShiftRows transformation, as shown in figure 4.6. Like this, the ShiftRows transformation becomes a part of the SubBytes transformation but without any additional delay or resource utilization in the original SubBytes transformation

4.7.1 Implementation Results

Our AES iterative architecture was implemented on Spartan-3(XC3S4000), using only 2 out of the total 96 BRAMs available (8.33% BRAM usage), while occupying 483 out of 27,648 number of slices. The number of four input LUTs were 794 out of 55,296 and the number of slice flip flops utilized were 491 out of 55,296. One out of the four available DCM

was used giving a maximum output frequency of 155.198 MHz. These results again prove the excellent resource utilization achieved using our approach.

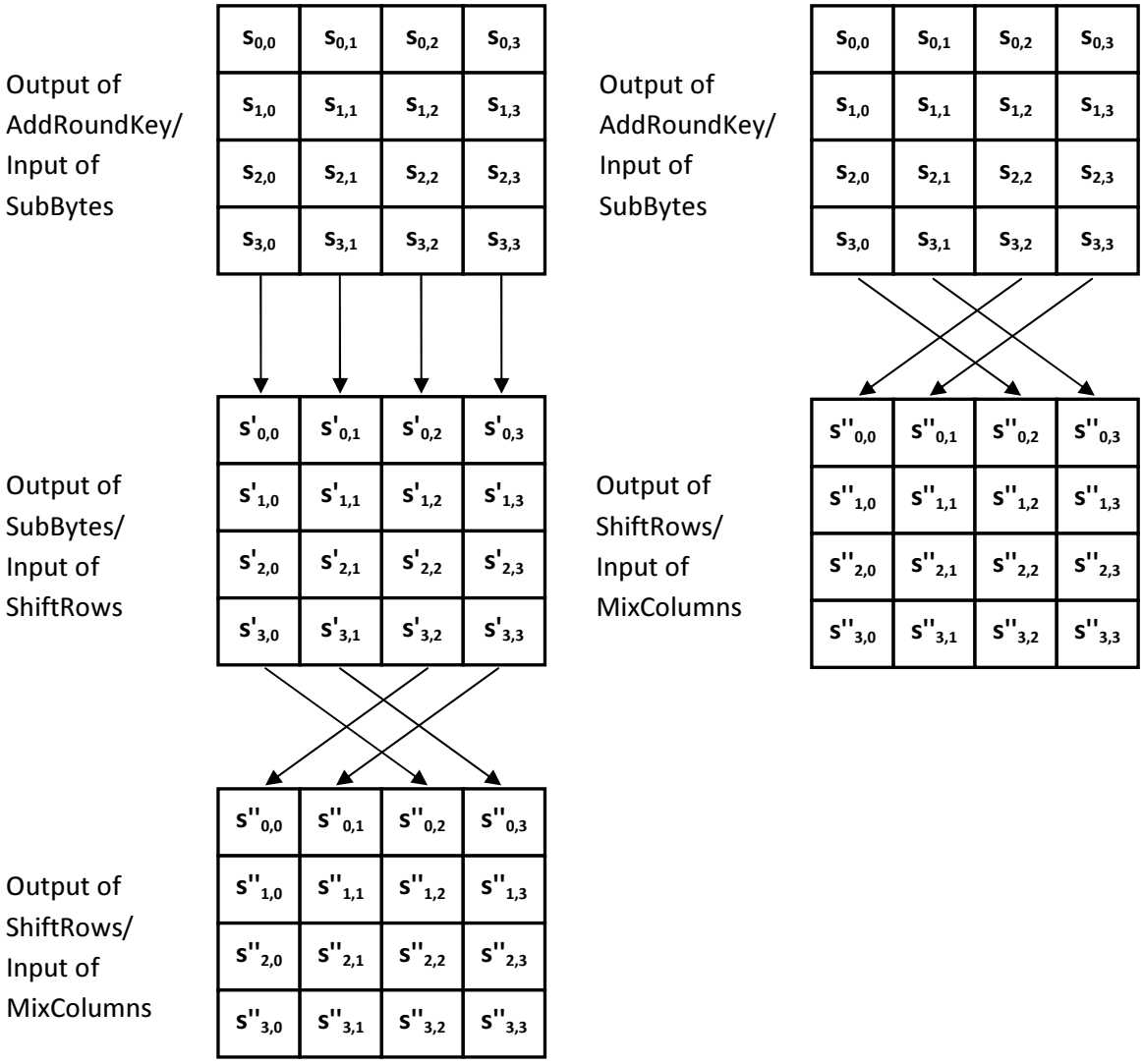


Figure 4.6 Optimization of ShiftRows Transformation

4.8 Summary

This chapter is the core of this thesis. It first described the options of architectures for an AES encryption core, the options for implementing the SubBytes transformation and the previous work done that is related to the proposed work in this thesis. Then our work was covered in detail in the chapter.

The first proposed work was an iterative implementation of the AES encryption cipher. The next proposed work was a unique clocking technique to implement a resource efficient SubBytes transformation. Finally this work was incorporated in our iterative AES

encryption core to give the excellent results with respect to resource utilization. In the following chapter, these results are compared with results of other work done in this field.

5 Results

In the following section results of our implementation are compared with various other implementations.

5.1 Results of Implementation of S-Box Based AES

We have compared our results with previous conventional FPGA based BRAM implementation of AES [24-28] that uses the S-Box approach with data path of 128 bits. The FPGA used by Algredo-Badillo et al. [24] and Zambreno et al. [25] are Virtex-II series, similarly the FPGA used by Swankoski et al. [26] is Virtex-II Pro series and by Lopez-Trejo et al. [27] and Arshad Aziz et al. [28] is Spartan-3 series. In all these devices, the size of BRAM is 18kbit.

To make fair comparison we have also utilized the same series devices, i.e. Spartan-3, having BRAM size of 18Kbit for our proposed S-box design. Our encryption core uses only 8 BRAMs and 405 slices and by comparing it with results of Lopez-Trejo et al. because of use of same device as given in Table 5.1, it is considered to be an area efficient.

The effective frequency of previous implementations ranges from 75MHz to 165MHz while our effective frequency is 206.28MHz as clearly shown in Table 5.1.

Table 5.1 Comparison of Results of Implementation of S-Box based AES

Implementation	Device	Data Path	Area (Slices)	BRAMs	Frequency (MHz)	Throughput (Gbps)
I. Algredo-Badillo [24]	Virtex-II (XC2V1000)	128	586	10	96.42	1.450
J. Zambreno [25]	Virtex-II (XC2V4000)	128	387	10	110.16	1.41
E. J. Swankoski [26]	Virtex-II Pro (XC2VP50)	128	1319	16	145.052	1.857
E. Lopez-Trejo [27]	Spartan-3 (XC3S4000)	128	713	53	100.08	1.051
Arshad Aziz [28]	Spartan-3 (XC3S50)	128	-	4	165	-
Our Design	***Spartan-3 (XC3S4000)***	***128***	***405***	***8***	***206.28***	***2.640***

5.2 Results of Implementation of Resource Efficient SubBytes Transformation

The resource efficient technique proposed in this thesis, for the implementation of SubBytes transformation, employs the BRAM for the storage and access of S-Box. Table 5.2 shows the comparison of the results of this technique and the results of other AES implementations on FPGA.

We implemented our work on the Spartan-3 XC3S4000 device, and also on Virtex-II XC2VP2. We can see that the utilization of BRAMs has been reduced by 50% as compared to the most recent previous work of Arshad Aziz [28]. Our work utilizes only two embedded BRAMs for the implementation of the SubBytes transformation.

Table 5.2 Comparison of Results of Resource-Efficient SubBytes Transformation

Implementation	Device	Data Path	BRAMs	Frequency (MHz)
J. Zambreno [25]	Virtex-II (XC2V4000)	128	8	110.16
I. Algredo-Badillo [24]	Virtex-II (XC2V1000)	128	8	96.42
E. J. Swankoski [26]	Virtex-II Pro (XC2VP50)	128	8	145.052
E. Lopez-Trejo [27]	Spartan-3 (XC3S4000)	128	16	100.08
Arshad Aziz [28]	Spartan-3 (XC3S50)	128	4	165
Arshad Aziz [28]	Virtex-II Pro (XC2VP2)	128	4	210
Our S-Box Module only	***Spartan-3 (XC3S4000)***	***128***	***2***	***155.198***
Our S-Box Module only	***Virtex-II Pro (XC2VP2)***	***128***	***2***	***206.186***

5.3 Results of Our AES Iterative Architecture Based on Resource Efficient SubBytes Transformation

The design of our AES iterative architecture based on resource efficient SubBytes transformation was implemented on the Spartan 3 device XC3S4000. The maximum frequency for this design is 155.198 MHz. For better comparison with the previous results of BRAM based approach of AES, our design was then also implemented on the Virtex II device XC2VP2.

In table 5.3, the column for BRAMs shows the number of BRAMs used in the SubBytes transformation only of these implementations. The column for Area (slices) shows the utilization of resources other than the BRAMs, and even for the same design, this quantity varies with the target device also. It is evident from the table that in our design there is a drastic reduction in number of BRAMs while offering the maximum frequency 155.198 MHz for Spartan 3. This frequency has been achieved by our efficient AES implementation even when we are using a quadrupled clock in our work.

Table 5.3 Comparison of Results of Our AES Iterative Architecture Based on Resource Efficient SubBytes Transformation with Results of Previous Works

Implementation	Device	Data Path	BRAMs	Area (Slices)	Frequency (MHz)
J. Zambreno [25]	Virtex-II (XC2V4000)	128	8	387	110.16
I. Algredo-Badillo [24]	Virtex-II (XC2V1000)	128	8	586	96.42
E. J. Swankoski [26]	Virtex-II Pro (XC2VP50)	128	8	1319	145.052
E. Lopez-Trejo [27]	Spartan-3 (XC3S4000)	128	16	713	100.08
Arshad Aziz [28]	Spartan-3 (XC3S50)	128	4	-	165
Arshad Aziz [28]	Virtex-II Pro (XC2VP2)	128	4	-	210
Our Design	***Spartan-3 (XC3S4000)***	***128***	***2***	***483***	***155.198***
Our Design	***Virtex-II (XC2VP2)***	***128***	***2***	***443***	***206.186***

6 Conclusion and Future Work

6.1 Conclusion

The aim of this research was to implement the AES on Xilinx FPGA by utilizing the FPGA's dedicated embedded memories, to store the S-box in a fully area-optimized way. This aim has been successfully achieved by optimizing the implementation of SubBytes transformation, using embedded BRAMs in an FPGA.

The number of BRAMs available on an FPGA device is limited and varies according to the device size; and so does the cost. Specially, when we are implementing AES as part of a larger system, we may require additional BRAMs for some other application on the same FPGA. The most recent ROM based approach of implementing the SubBytes transformation utilized four BRAMs for one complete SubBytes transformation in one clock cycle. We aimed to make the ROM approach for area-efficient and to actually halve this utilization of BRAMs.

The challenge for this thesis work was to develop a suitable synchronous clocking technique. This was done effectively as explained in chapter 4 of this thesis. The work presented in this thesis successfully accesses the S-Box in one BRAM, eight times in one clock cycle; and uses only two BRAMs for implementing the complete SubBytes transformation in one clock cycle only.

6.2 Future Work

To give a complete cryptographic unit, the incorporation of Key Scheduler and AES decryption process in our area-efficient AES encryption core, may be taken up as future work. All this work would ideally be done with consistent focus on improving throughput and size of this core. To achieve this, research should be focused on the target device, clocking techniques and area constraints.

For future, we recommend to store the S box and inverse S box in the same BRAM and implement the clocking technique presented in this thesis, to implement a resource-efficient encryption and decryption AES core. Furthermore, appropriate pipelining of the design and use of efficient embedded FPGA resources may result in very high throughput rates.

6.3 Publications

- A Compact AES Encryption Core on Xilinx FPGA, Dur-e-Shahwar Kundi, Saleha Zaka, Qurat-Ul-Ain, Arshad Aziz, *International Conference on Computer, Control and Communication (IC4)*, 14-15 February 2007, available at http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4909251&tag=1.

- Area Efficient S-Box Approach for SubBytes transformation in AES, Saleha Zaka, Arshad Aziz, Submitted to IEE Electronic Letters and is under review.

Bibliography

1. Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard, NIST Public Notice, available at http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt
2. Selecting the Advanced Encryption Standard, Burr W.E, Paper in *Security & Privacy*, IEEE Vol. 1, Issue 2, March-April 2003, available at http://ieeexplore.ieee.org/iel5/8013/26759/01193210.pdf%3Farnumber%3D1193210&authDecision=-203.
3. The Advanced Encryption Standard, NIST Status Report, available at http://csrc.nist.gov/publications/nistbul/itl99-08.txt.
4. Report on the Development of the Advanced Encryption Standard (AES), October 2000, available at http://csrc.nist.gov/archive/aes/round2/r2report.pdf.
5. Announcing The Advanced Encryption Standard (AES), Federal Information Processing Standards (FIPS) Publication 197, available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
6. The Enduring Value of Symmetric Encryption, White Paper, August 2000, available at http://www3.safenet-inc.com/Blog/pdf/WP-SymmetricEncryption.pdf.
7. Symmetric vs. Asymmetric Encryption, White Paper, available at http://www.ketufile.com/Symmetric_vs_Asymmetric_Encryption.pdf.
8. Journal of Research of the National Institute of Standards and Technology, Volume 107, Number 3, NIST Reports Measurable Success Of Advanced Encryption Standard, May–June 2002, available at http://Nvl.Nist.Gov/Pub/Nistpubs/Jres/107/3/J73nbr.pdf.
9. A Discussion of the Importance of Key Length in Symmetric and Asymmetric Cryptography, Lorraine C. Williams, available at http://www.giac.org/certified_professionals/practicals/gsec/0848.php.
10. Recommendation for Key Management – Part 1: General (Revised), NIST Special Publication, March 2007, available at http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf.

11. Advanced Encryption Standard Algorithm Validation List, NIST, 2011, available at http://csrc.nist.gov/groups/STM/cavp/documents/aes/aesval.html.
12. NSA Suite B Cryptography, National Security Agency, Central Security Service, available at http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml.
13. Recommendation on the use of Triple DES or AES for BlackBerry transport layer encryption, available at http://www.blackberry.com/btsc/search.do?cmd=displayKC&docType=kc&externalId=KB05429&sliceId=SAL_Public&dialogID=3704185&stateId=0%200%203706001.
14. AES Encryption Information: Encryption Specification AE-1 and AE-2, WinZip, January 2009, available at http://www.winzip.com/aes_info.htm.
15. IEEE Standard for Information technology-Telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 5: Enhancements for Higher Throughput, available at http://standards.ieee.org/getieee802/download/802.11n-2009.pdf.
16. Exploiting Cryptographic Architectures over Hardware Vs. Software Implementations: Advantages And Trade-Offs, N. Sklavos, K. Touliou, And C. Efstathiou, Proceedings of the *5th WSEAS International Conference On Applications Of Electrical Engineering*, Prague, Czech Republic, March 12-14, 2006, pp.147-151, available at http://www.wseas.us/e-library/conferences/2006prague/papers/513-177.pdf.
17. Survey of new trends in Industry for Programmable hardware: FPGAs, MPPAs, MPSoCs, Structured ASICs, eFPGAs and new wave of innovation in FPGAs, available at http://www.lirmm.fr/~ahmed/files/FPL10/Zahid_FPL10_SurveyPaper.pdf.
18. Xilinx Named EE Times ACE Awards finalist for Design Team of the Year, Xilinx Press Release, March 2010, available at http://press.xilinx.com/phoenix.zhtml?c=212763&p=irol-newsArticle&ID=1402188&highlight=.
19. Spartan-3 FPGA Family Data Sheet, DS099, December 2009, available at http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf.
20. XILINX ISE In-Depth Tutorial, Ver. 10.1, available at http://www.xilinx.com/direct/ise10_tutorials/ise10tut.pdf

21. Analysis of AES Hardware Implementations, Song J. Park, available at http://cs.ucsb.edu/~koc/cs290g/project/2003/park.pdf.
22. Block Memory Generator, V2.8 Ds512 September 19, 2008.
23. An Efficient FPGA Based Sequential Implementation Of Advanced Encryption Standard, Proc. *IEEE ITI 3rd Int. Conf. Information And Communication Technology (ICICT 2005)*, Aziz, A. And Ikram, N., Cairo, Egypt, December 2005, pp. 875–882.
24. Design And Implementation of an FPGA-Based 1.452 Gbps Non-Pipelined AES Architecture, I. Algredo-Badillo, C. Feregrino-Uribe and R. Cumlido-Parra, available at http://www.springerlink.com/content/c1v4508g566t0032/.
25. Exploring Area/Delay Tradeoffs In An AES FPGA Implementation, J. Zambreno, D. Nguyen and A. Choudhary, *Proc. Int. Conf. Field-Programmable Logic and Its Applications (FPL)*, Lecture Notes in Computer Science, Vol. 3203 (Springer-Verlag 2004), available at http://www.springerlink.com/content/86anebk8xcyaktx4/.
26. A Parallel Architecture For Secure FPGA Symmetric Encryption, E. J. Swankoski, V. Narayanan, M. Kandemir and M. J. Irwin, *18th Int. Parallel And Distributed Processing Symp. (IPDPS'04)* — Workshop, Santa Fe, New Mexico, April 2004, p. 123, available at http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1303101.
27. An Efficient FPGA Implementation of CCM Using AES, E. Lopez-Trejo, F. Rodriguez-Henriquez and A. Diaz-Perez, 2005, available at http://delta.cs.cinvestav.mx/~francisco/ICISC_2005_ETL.pdf.
28. Memory Efficient Implementation Of AES S-Boxes On FPGA, Arshad Aziz And Nassar Ikram, *Journal Of Circuits, Systems, And Computers*, Vol. 16, No. 4, 2007, pp. 603–611.
29. Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs, January 2006, available at http://www.xilinx.com/support/documentation/application_notes/xapp462.pdf.
30. Using Block RAM in Spartan-3 Generation FPGAs, March 2005, available at http://www.xilinx.com/support/documentation/application_notes/xapp463.pdf.