Shared Architecture for Encryption/Decryption of AES

Richa Kumari Sharma,
MITS University
Lakshmangarh (Raj.)

S.R.Biradar,
MITS University
Lakshmangarh (Raj.)

B.P.Singh
MITS University
Lakshmangarh (Raj.)

ABSTRACT

Security has become an increasingly important feature with the growth of electronic communication. The Symmetric in which the same key value is used in both the encryption and decryption calculations are becoming more popular. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. This standard is based on the Rijndael algorithm. Here this paper presents the shared architectures for both encryption and decryption. Shared architecture reduces the area as well as the path delay. This methodology uses VHDL implementation of all the modules and results are concluded in terms of Delay and Frequency.

General Terms

Security, Algorithm, symmetric key, Path delay.

Keywords

Cryptography , Secret key, AES, Rijndael, FPGA, VHDL.

1. INTRODUCTION

The word cryptography represents 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 [1]. 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 [2]. Cryptography is divided in two categories first is symmetric key cryptography (sender and receiver shares the same key) and the second one is asymmetric key cryptography (sender and receiver shares different keys). Here the major concentration is about symmetric key cryptography due to its use in military application, embedded system design, financial and legal files, medical reports, and bank services via Internet, telephone conversations, and e-commerce transactions etc. [3]. Many symmetric key cryptographic algorithms were proposed, such as the Data Encryption Standard (DES), the Elliptic Curve Cryptography (ECC), the Advanced Encryption Standard (AES) and other algorithms [4]. Here the implementation of AES algorithm is presented by using VHDL to increase the data transfer speed.

1.1 Aim of the Paper

The cryptographic algorithms became the main proceeding for protection of very important data, the security objective called confidentiality being the one taken into account by their hardware implementation and by their integration into the present-day communication systems. An encryption algorithm provides Confidentiality, Authentication, Integrity and Non-repudiation [4]. Confidentiality ensures that the information is accessible to only authorized set of people .Authentication is the act of establishing that the algorithm is genuine. Integrity in general means completeness but in encryption it is adhering to some set of principles. It is based on consistency with some mathematical proof. Non- repudiation in cryptology means that it can be verified that the sender and the recipient were, in fact, the parties who claimed to send or receive the message, respectively [5]. The main Aim of presented AES design is to present mathematical models for the AES algorithm which reduces the hardware Implementations cost. The hardware is designed to achieve minimum delay to provide maximum speed. A modified VHDL implementation is explained for Encryption/Decryption modules to provide less Delay.

2. THE AES ALGORITHM

2.1 Background of Algorithm

The National Institute of Standards and Technology (NIST) has initiated a process to develop a Federal information Processing Standard (FIPS) for the Advanced Encryption Standard (AES),specifying an Advanced Encryption Algorithm to replace the Data Encryption standard (DES) the Expired in 1998. The Rijndael Algorithm was chosen since it had the best overall scores in security, performance, efficiency, implementation, ability and flexibility[2].

2.2 Basic of Algorithm

The Rijndael algorithm is a symmetric block cipher that can process data blocks of 128 bits through the use of cipher keys with lengths of 128, 192, and 256 bits. The AES algorithm as Rijndael, is also a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information [1]. Encryption converts data to an unintelligible form called cipher-text. Decryption of the cipher-text converts the data back into its original form, which is called plaintext [2]. The number of rounds is depends upon the key length described in Table 1.

2.3 Encryption and Decryption in AES

The Basic AES Encryption and Decryption structure with various steps is shown in Fig 1. This block diagram is generic for AES specifications. It consists of a number of different transformations applied consecutively over the data block bits, in a fixed number of iterations, called rounds [2].

The number of rounds depends on the length of the key used for the encryption process as depicted in Table1. The Advanced Encryption Standard can be programmed in software or built with pure hardware [5].

Table 1. Key Block Round Combination

Block Size (N_b words) = 4		
Bit Mode	**Key Length (N_k words)**	**Number of Rounds (N_r)**
128	4	10
192	6	12
256	8	14

The hardware implementation of the Rijndael algorithm can provide either high performance or low cost for specific applications. At backbone communication channels or heavily loaded servers it is not possible to lose processing speed, which drops the efficiency of the overall system while running cryptography algorithms in software [5]. On the other hand, in the performance comparison between software and hardware implementation the priority is to evaluate which system provides higher security. Hardware's inflexibility eliminates possibility for the external changes to the system, and this result in a high quality physical security when compared with software implementations [6]. A low cost and small design can be used in smart card applications, which allows a wide range of equipment to operate securely.

3. TRANSFORMATIONS IN AES

3.1 Sub Byte and Inverse Sub Byte transform

The bytes substitution transformation is a non-linear substitution of bytes that operates independently on each byte of the State using a substitution table (S-box) [2]. The sub byte transform is shown in Fig 2. In AES hardware implementation, S-box design contributes a major role in optimization [7]. There are two approaches for S-box design. Design a multiplicative inversion and affine transformation separately or Construct a logic circuit defining the input and output of the S-box function [8]. Affine transform explains four transformations (matrix operations), which are designed and implemented such that Sub Bytes and Inverse Sub Bytes can use the same module.

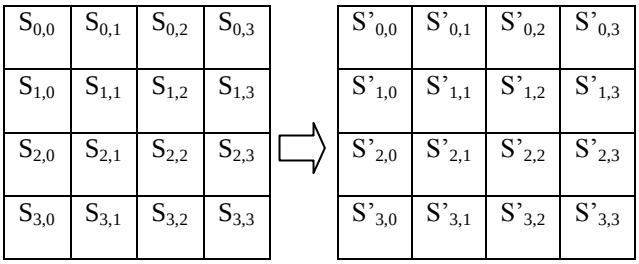

Fig 2: S-Box and Inv S-box to the Each Byte of the State

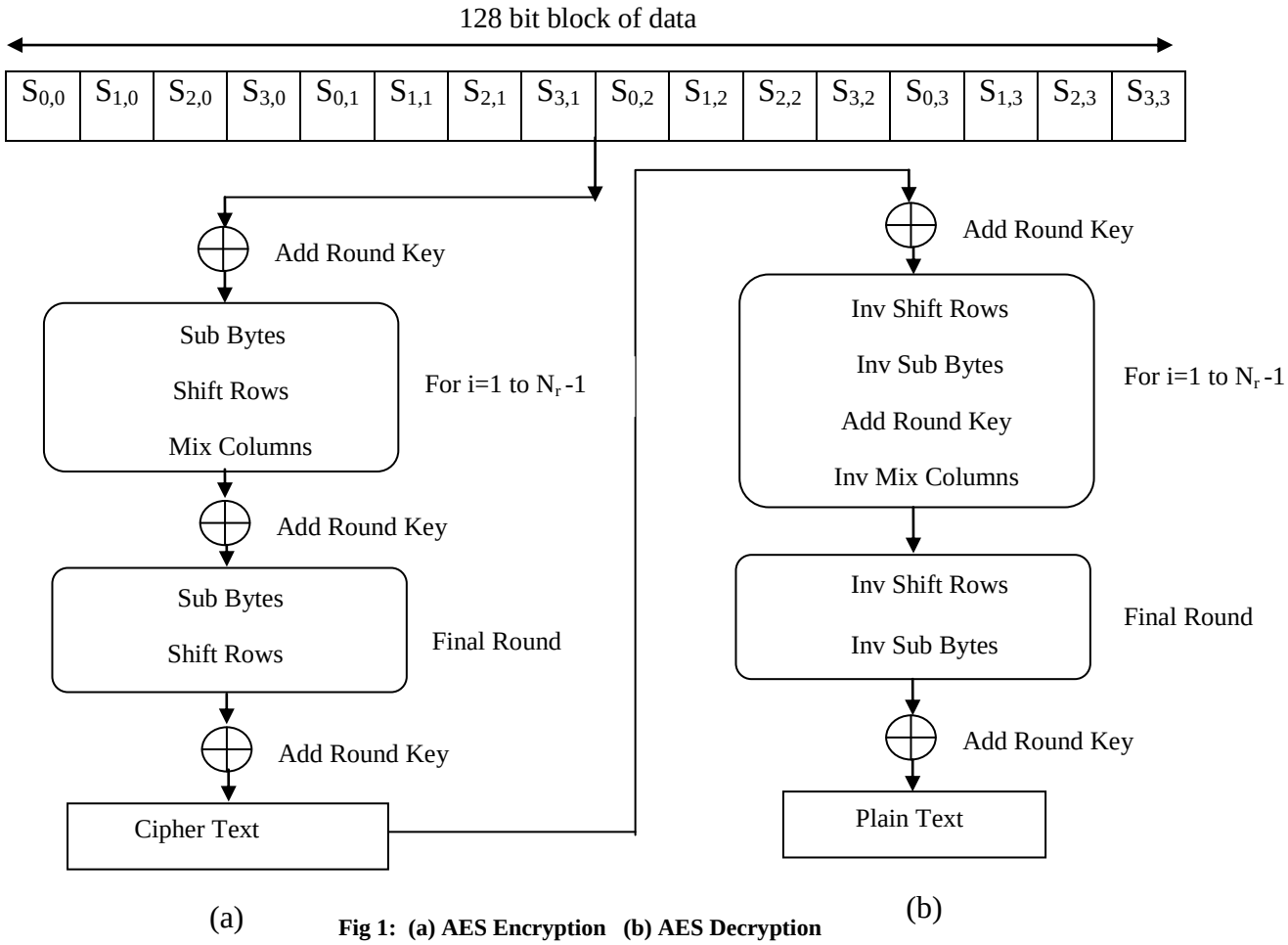

Fig 1: (a) AES Encryption (b) AES Decryption

Each *S*-box and S^{-1} box requires a 2k-bit look-up table, and each round unit needs 32 such look-up tables to implement both encryption and decryption as shown in Fig 3. The Sub Bytes and Inv Sub Bytes transformations can share look-up tables which only implement multiplicative-inverse in GF(2^8)[8]. Another approach is to store the Fvalue of *S* box and S^{-1}box in two separate ROMs, and read the initial values into RAMs at the beginning of encryption/ decryption.

To implement the Shift Row Function on hardware basically two main approaches are there. Designing it by using barrel Shifter [9] or it can be implemented by using Look up Table [8]. In order for this barrel shift to function, at least 2-bits of shift amount lines are needed to specify any number shifts from 0 to 3 as shown in Fig 5.

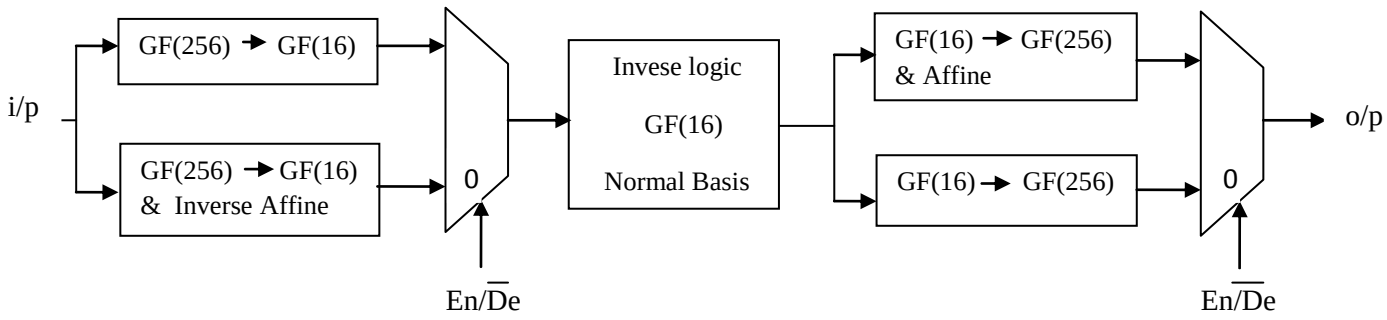

Fig 3: The Sub Bytes and Inverse Sub Bytes transformations.

3.2 Shift Row and Inverse Shift Row transform

In the Shift Rows transformation the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets) as shown in Fig 4. The first row, r =0, is not shifted, while the second, third and fourth rows cyclically shift one byte, two bytes and three bytes to the left, respectively and in Inverse shift it will be shifted towards right [3].

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

⇨

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

With shift amount lines a direction line is also needed to specify the module is shifting bytes to the left (encryption) or to the right (decryption). In the upper line of multiplexors, the least significant bit S0 specifies whether the bytes should be shifted by zero or one positions [8]. In the second line, the most significant bit S1 specifies whether the bytes are shifted by two or zero positions. By performing all 4 possible combinations of S1 and S0, any number of shifts from 0 to 3 can be done. In order to code this module in VHDL, a series of nested if-statements were used within a process sensitive to direction, shift amount, data changes. In the first series of if-statements, the least significant bit S0, was tested and the shift amount and direction was performed accordingly [9]. As the second approach is to implement it by **look up table,** so Look up table often used to replace a runtime computation with a simpler array indexing operation. The tables may be pre calculated and stored in static program storage, calculated as part of a programs initialization phase (memorization), or even stored in hardware in application-specific platforms. The usefulness of a look up table becomes more and more evident with increasing complexity of function.

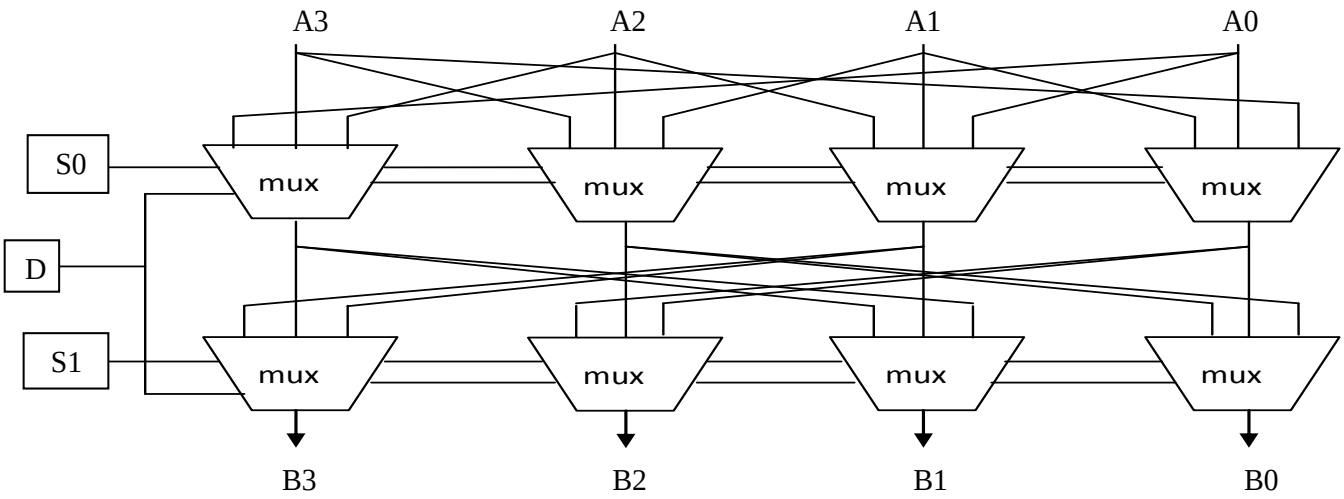

Fig 5: Bi-directional barrel shifter schematic

3.3 Mix Column and Inverse Mix Columns Transformation

Mix Columns and Inverse Mix Columns operations are defined over 4-byte words that represent a column of the State [5]. The AES algorithm has no complex phases and the implementation of all of them is very simple. However, the Mix Columns phase implementation deserves special consideration. This transformation is based on Galois Field multiplication. Each byte of a column is replaced with another value that is a function of all four bytes in the given column. The Mix Columns transformation is performed on the State column-by-column [7]. Each column is considered as a four-term polynomial over GF(2^8) and multiplied by a(x) modulo $x^4 + 1$, Where

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

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

As depicted in Fig 6 four inputs are applied here and the generated output is $S'_{0,c}$ which is with the help of XTIME block. The function of XTIME block is depicted in Fig 7. So if we want to implement both the mix column and Inverse mix column on a single architecture then we can follow the Fig 8.

.

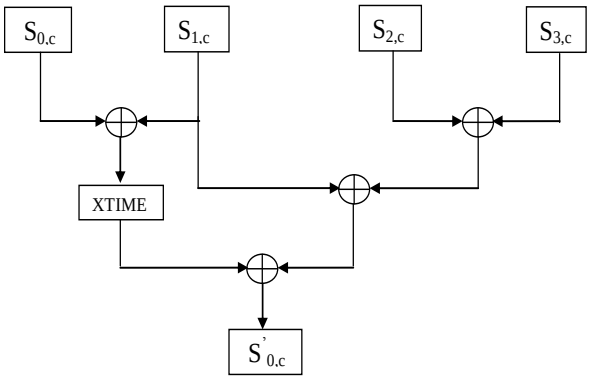

Fig 6: Block diagram for substructure sharing implementation of Mix Columns transformation.

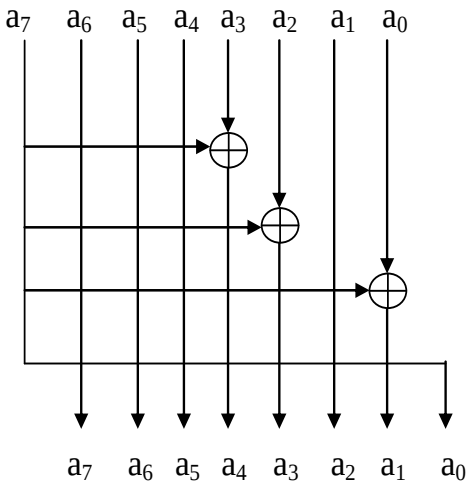

Fig 7: Circuit diagram of XTIME

As 'a', 'b', 'c', 'd' are the four bytes in a column of the State with ascending row numbers, 'mix' and 'inv mix' are the outcomes of applying Mix Columns and Inv Mix Columns transformation to the inputs, respectively. Fig 8 shows the shared architecture in the terms of bytes in the first row of state for Mix Column and Inverse Mix Column Module [9]. The block diagram for applying Mix Columns and Inv Mix Columns to the bytes in other rows can be obtained by exchanging the position of the input bytes according to the operations.

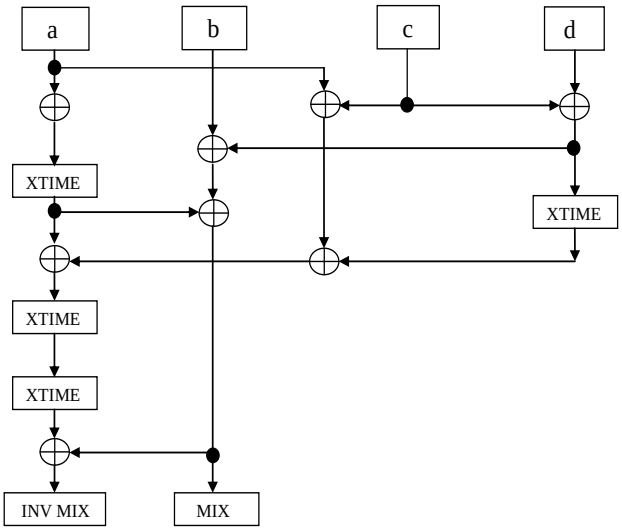

Fig 8: Shared implementation of the Mix Columns and the Inv Mix Columns transformations (States of Bytes in the first row).

Thus we can reduce the delay as well as the number of device count by using the shared architecture.

3.4 Add Round Key and Key Expansion Unit

3.4.1 Add Round Key

Add Round Key step is applied one extra time comparing to the other encryption and decryption steps. This step is common among encryption and decryption. 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 as shown in Fig 9. This round key is called the initial round key [4]. It is implemented in hardware as a simple exclusive-or operation of the 128 bit data and key [7].

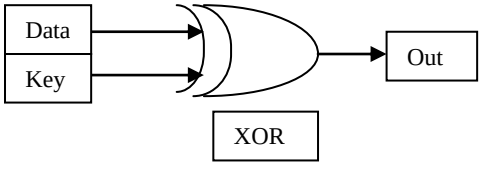

Fig 9: Hardware Implementation of Add Round Key

3.4.2 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 [7]. The whole operation is shown in Fig 10.

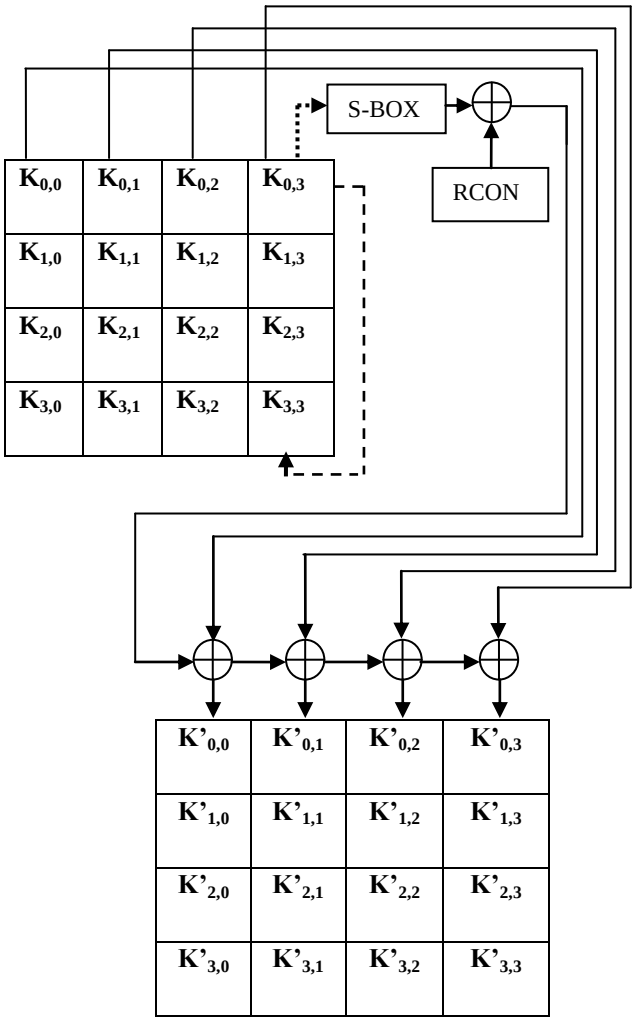

Fig 10: Implementation of Key Expansion

In key Expansion the S-Box is also used so by using shared architecture we can reduce the delay as well as the device count. In shared architecture the S-Box can be shared among sub byte and key expansion unit.

4. SIMULATIONS AND RESULTS

The VHSIC Hardware Description Language is an industry standard language used to describe hardware from the abstract to the concrete level. VHDL resulted from work done in the 70s and early 80s by the U.S. Department of Defense. VHDL usage has risen rapidly since its inception and is used by literally tens of thousands of engineers around the globe to create sophisticated electronic products [11]. VHDL is a powerful language with numerous language constructs that are capable of describing very complex behavior. In 1986, VHDL was proposed as an IEEE standard. It went through a number of revisions and changes until it was adopted as the IEEE standard in December 1987.

VHDL is used as the hardware description language because of the flexibility to exchange among environment [11]. The software used for this work is Xilinx 6.1i and the waveforms are simulated with the help of model sim simulator. This is used for writing, debugging, simulating and checking the performance results using the simulation tools available on Xilinx 6.1i. The delay is calculated with three different Device families. The delays have been generated as result is shown in Table 2 for encryption and in Table 3 for decryption. As from the tables we can see that by using modified shared architecture the delay can be reduced as well as device count is also reduced. As the VHDL simulator is used to verify the functionality of all the modules so the interface and RTL for Encryption and Decryption is shown in Fig 11. The comparative analysis graph is shown in Fig 12.

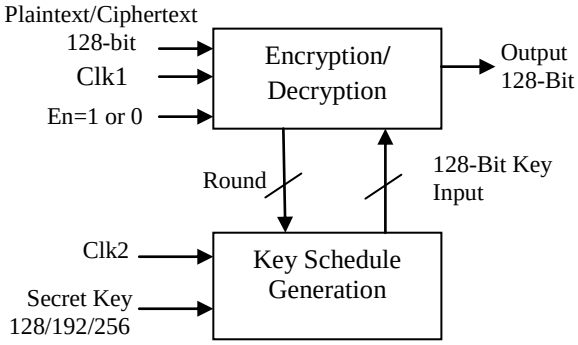

Fig 11: RTL for Encryption and Decryption

Table 2. Delay and frequency of AES algorithm Encryption (128-bit)

Modules	AES algorithm Conventional (128-bit)		AES algorithm Modified (128-bit)	
	Delay (ns)	Frequency (MHz)	Delay (ns)	Frequency (MHz)
Sub Byte	12.71	78.67	10.69	93.54
Shift Row	9.226	108.38	7.158	139.70
Mix Column	10.044	99.56	8.168	122.42
Key Expansion	22.115	45.22	13.475	74.21
Encryption	54.095	18.48	39.491	10.652

Table 3. Delay and frequency of AES algorithm Decryption (128-bit)

Modules	AES algorithm Conventional (128-bit)		AES algorithm Modified (128-bit)	
	Delay (ns)	Frequency (MHz)	Delay (ns)	Frequency (MHz)
Inv Sub Byte	9.36	106.83	8.63	115.87
Inv Shift Row	6.904	144.84	5.703	175.35
Inv Mix Column	11.645	85.87	10.219	97.86
Key Expansion	22.115	45.22	13.475	74.21
Decryption	50.024	19.99	38.027	26.30

So from the graph shown in Fig 12, it can conclude that the delay for the Encryption and decryption is reduced around 15ns and 12ns respectively.

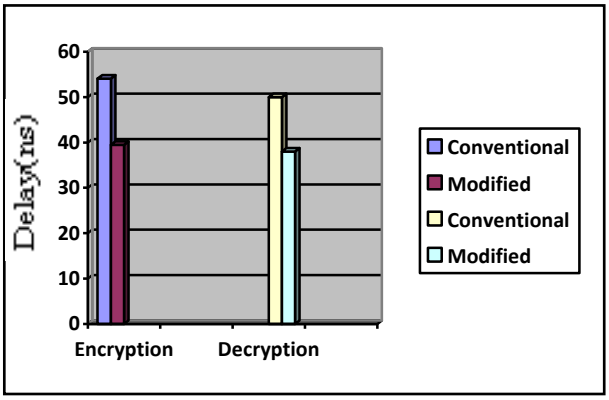

Fig 11: Comparative analysis Graph

5. CONCLUSION

As the cryptography is playing the major role in today's world. So the frequency is the main concern so that the time period can be minimized. Here the basic of AES algorithm is explained in brief and the implementation of its all module in a form of shared architecture for encryption and decryption is presented by using VHDL, so that delay can be reduced. Here the simulations are performed with different device families. The delay is minimum with virtex2 (Xc2v40) family. The software used is Xilinx6.1i and the waveforms are simulated with model sim simulator**.**

Future work will concentrate on the applications where

speed of the transmission is the main objective and the security is main concern like in Bluetooth, 4G, sensor networks, Ad-Hoc networks and cross layer architecture.

6. REFERENCES

[1] Xinmiao Zhang and Keshab K. Parhi, "Implementation Approaches for the Advanced Encryption Standard Algorithm" IEEE 2002.

[2] Hui QIN, Tsutomu SASAO, Yukihiro IGUCHI, "An FPGA Design of AES Encryption Circuit with 128-bit Keys" GLSVLSI'05, ACM 2005

[3] Chih-Peng Fanand and Jun-Kui Hwang, "FPGA Implementations Of High Throughput Sequential And Fully Pipelined AES Algorithm" International journal of Electrical Engineering, vol.15, no.6, pp. 447-455, 2008.

[4] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh, "Efficient and High Performance Parallel Hardware Architecture for the AES-GCM" IEEE Transactions On Computers, vol.61, no. 8, August 2012.

[5] Saambhavi Baskaran and Pachamuthu Rajalakshmi, "Hardware Software Co-Design of AES on FPGA" ICACCI '12,ACM August 2012.

[6] Ashwini M. Deshpande, Mangesh S. Deshpande and Devendra N. Kayatanavar, "FPGA Implementation of AES Encryption and Decryption" International Conference on Control, Automation, Communication and Energy conservation -2009

[7] X. Zhang and K. K. Parhi, "High-speed VLSI architectures for the AES algorithm" , IEEE Transactions on Very Large Scale Integration Systems, vol.12, issue 9, pp.95 967, Sep. 2004.

[8] Jin Gong ,Wenyi Liu, Huixin Zhang, "Multiple Lookup Table-Based AES Encryption Algorithm Implementation" Elseveir-2012 vol.25 pg no.842 – 847.

[9] Saurabh Kotiyal, Himanshu Thapliyal and Nagarajan Ranganathan, "Design of A Reversible Bidirectional Barrel Shifter" IEEE international conference 2011.

[10] J. Vijaya, M. Rajaram, "High Speed Pipelined AES with Mixcolum Transform "European Journal of Scientific Research" 2011.Vol.61 No.2 ,pp. 255-264.

[11] Douglas L. Perry, "VHDL: Programming by Example" McGraw-Hill Fourth Edition 2004.