FPGA IMPLEMENTATIONS OF ADVANCED ENCRYPTION STANDARD: A SURVEY

Shylashree.N[1], Nagarjun Bhat[2] and V. Shridhar[3]
[1]Research Scholar (R.N.S.I.T),in E.C.E, at PESCE, Mandya, Karnataka, India
[2]Student, B.E (4th semester), Dept of ECE, RNSIT
[3]Professor, in E.C.E, at P.E.S.C.E, Mandya, Karnataka, India

ABSTRACT

Advanced Encryption Standard (AES) is the most secure symmetric encryption technique that has gained worldwide acceptance. The AES based on the Rijndael Algorithm is an efficient cryptographic technique that includes generation of ciphers for encryption and inverse ciphers for decryption. Higher security and speed of encryption/decryption is ensured by operations like SubBytes (S-box)/Inv. SubBytes (Inv.S-box), MixColumns/Inv. Mix Columns and Key Scheduling. Extensive research has been conducted into development of S-box /Inv. S-Box and MixColumns/Inv. MixColumns on dedicated ASIC and FPGA to speed up the AES algorithm and to reduce circuit area. This is an attempt, to survey in detail, the work conducted in the aforesaid fields. The prime focus is on the FPGA implementations of optimized novel hardware architectures and algorithms.

KEYWORDS: *Advanced Encryption Standard (AES), FPGA, Block Cipher, NIST (National Institute of Standards and Technology), Mixcolumn (MC), Inverse mixcolumn (IMC), Subytes (S-Box), Inverse Subbytes (S^{-1}- Box).*

I. INTRODUCTION

Communication and transfer of data in the present days invariably necessitatethe use of encryption. Besides its uses in Military and Government's secret communication, Encryption is also used for protecting many kinds of civilian systems such as Internet e-commerce, Mobile networks, automatic teller machine transactions, copy protection (especially protection against reverse Engineering and Software piracy), and many more. Data encryption is achieved by following a systematic algorithm called encryption algorithm. An encryption algorithm provides Confidentiality, Authentication, Integrity and Non-repudiation. Confidentiality is the requirement that information is kept secret from people who are not authorized to access it. Authentication is the certainty that the message indeed originates from the purported sender. Integrity is the requirement that information is unaltered and complete, or, that information "is modified only by those users who have the right to do so." Non-repudiation means that the sender or receiver of a message cannot deny having sent or received the message.

This paper is organized as follows: Section 2 reviews the related works on FPGA.Section 3 gives a general introduction to cryptography. Section 4 explains the Advanced encryption standard. Section 5 deals with Approaches for efficient hardware implementation of AES: A Survey. Section 6 discusses the performance summary and the conclusions arrived at, in Section 7.

II. RELATED WORK ON FPGA

With the advent of new technologies in the field of FPGAs, they are increasingly preferred over ASICs. Advantages of FPGAs include the ability to re-program in the field to fix bugs. FPGA design

flows support the use of third party EDA tools to perform design flow tasks such as static timing analysis, formal verification, and RTL and gate level simulation.
Applications of FPGAs include digital signal processing, software – defined radio, aerospace and defence systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.
Considering the innumerable advantages of using a FPGA platform over others as can be noted, we have chosen to review AES implementations over FPGA exclusively.
A Rijndael cipher for encryption using a basic 64-bit iterative architecture was developed and presented in this paper. The proposed architecture implemented on FPGA achieves high speed,low area and cost effectiveness. Key scheduling unit has been added as a part of this work and the number of cycles required to encrypt text has been reduced and hardware optimization achieved. The results of this paper have been expressed in table 2 [15]. In this work a 32-bit datapath FPGA implementation of AES has been proposed. A significant improvement of 3.4 was achieved over the existing designs in terms of throughput. The results of this paper have been discussed in table 2 [18].A hardware implementation of AES algorithm suitable for wireless military communication has been suggested in this work. An optimized code was proposed for the Rijndael algorithm with 128-bit keys. A significant improvement in throughput and reduction of slices was achieved. The results of this paper have been discussed in table 2 [16]. In this paper a novel high-speed non-pipelined architecture for implementing both encryption and decryption operations of the Rijndael algorithm on the same FPGA implementation has been proposed. The results of this paper have been presented in table 2 [10].A high speed, non-pipelined FPGA implementation of AES-CCMP has been proposed. Where in the CCMP consists of two modes, that is Counter mode for Data Privacy and CBC mode for authenticity. The results of this paper have been presented in table 2[20].
A high throughput design combining both encryption and decryption on a single FPGA architecture has been proposed in this paper. Low area and low cost was achieved [23]. This work proposes a new flexible AES architecture that performs both encryption and decryption. A key generation module that generates both encryption and decryption keys is provided. Flexibility is achieved so that key generation depends on the data and hence hardware need not be changed every time[25]. FPGA implementation of AES algorithm has been presented in this paper. The encryption and decryption transformations have been performed using iterative design thus cutting hardware implementation costs. The result of this paper has been discussed in the Table 2 of this paper. The results of this paper have been described in table 2 [22].This paper proposes to combine both encryption and decryption on a single FPGA implementation with focus on low area and high throughput. The Rijndael's algorithm for 128-bit key is used on the fully pipelined AES encryptor /decryptor core proposed in this work. The results of this paper have been described in table 2 [17].
In this paper hardware implementation of optimized area for block cipher AES has been proposed. Time sharing of resources and iteration architecture has been used to reduce the area[24]. Designs achieving area, latency and bandwidth optimizations have been reviewed in this paper. A FPGA implementation of AES algorithm has been presented in this work incorporating these optimization techniques for better throughput and lower latency[27]. Two new designs of FPGA implementation of AES algorithm, one achieving a very high throughput and the other with a very small area have been presented in this paper. The high throughput design supports continued throughput during key changes for both encryption and decryption processes. The results of this paper can be observed in table 2 [29]. A new combinational logic to improve the efficiency of inner round pipelining has been developed in this paper. Composite field arithmetic reduced the area. A fully sub-pipelined encrypter/decrypter with three sub stage pipelining per round has been used to achieve higher throughput. The results of this paper can be found in table 2 [28].
Efficiency and high throughput issues for FPGA implementation of AES-GCM have been addressed in this paper. Both the AES engine and the modular multiplication over G.F (2m) have been discussed. The Karatsuba's algorithm has been used in the multiplication [26]. In this paper an extension of a public –key cryptosystem has been proposed to support a private-key cryptosystem. A new arithmetic unit has been developed in which the polynomial modular multiplication of ECC is extended to compute the polynomial arithmetic operations over binary extended field of AES. Higher hardware efficiency was achieved [41].

Pipelining techniques have been used in the architectural optimization for the FPGA implementation of AES presented in this paper. Significant improvement in terms of speed has been achieved by processing multiple rounds simultaneously though cost in terms of area increased. Exclusion of Shift Row stage and on-the-fly key generation have been incorporated to enhance throughput. The results of this paper have been presented in table 2 [21]. A throughput as high as 44.5Gbps has been achieved in the FPGA implementation of AES algorithm proposed in this paper. A fully pipelined architecture which uses a 128-bit cipher key has been proposed in it[33].This paper uses a pipelined architecture only for the outer-round in the FPGA implementation of the AES algorithm. Very high throughput and efficiency are the merits of the proposed work. The results of this paper can be seen in table 2 [31].The fully pipelined architecture with high throughput for data security applications has been proposed in this work. The hardware is implemented on FPGA. The results of this paper have been presented in table 2 [36]. A high-speed parallel pipelined architecture has been developed to build a hardware-efficient design for the implementation of AES algorithm in this paper. It uses an efficient inter-round and intra-round pipeline design. The results of this paper have been shown in table 2 [14]. A fully pipelined architecture for implementation of AES algorithm over FPGA has been presented in this paper. A trade-off between cost and area has been achieved with a considerable reduction in area and improvement in throughput. The results of this paper have been discussed in table 2[12].A new methodology for secret key block ciphers based on optimum number of pipeline stages has been proposed. The results of this work can be seen in table 2 of this paper [5].

This paper describes a FPGA implementation of Rijndael algorithm built using Reduced Residue of Prime Numbers or RRPN. The S-Box LUT entries form a set of Reduced Residue Prime Numbers which in turn forms a mathematical field. This makes the system resistant to algebraic attacks. The results of this paper have been shown in table 2 [19]. FPGA implementation of AES algorithm presented in this paper use the content-addressable memory (CAM) based scheme to realize the high speed Sub-bytes block. The mix-columns block is implemented by the application of hardware sharing and the real time key generations is performed using a cost efficient Add Round Key architecture [30]. A high performance S-box implementation of AES algorithm over FPGA using reduced residue of prime numbers (RRPN) has been proposed in this work. The proposed design adds to the complexity and efficiency of the AES algorithm thus enhancing resistance to algebraic attacks[33].In this paper three novel composite field arithmetic(C.F.A) S-Boxes of field $GF(((2^2)^2)^2)$ for the implementation of AES algorithm have been derived and the best of them chosen by algorithmic and architectural optimizations. The design with minimal area cost and highest throughput was chosen[38]. A 32-bit bus width architecture for implementation of AES algorithm on FPGA has been proposed in this work. Higher speed was achieved using pipelining whilst Sub-bytes method was implemented using both composite field and fixed ROM techniques [37]. An efficient high speed hardware implementation of AES algorithm has been presented in this paper. Composite field arithmetic in normal bases has been used and a novel key expansion architecture is also presented. Key variation during encryption is also supported by the proposed architecture. The results of this paper can be found in table 2 [39].

A highly optimized, high performance efficient hardware realisation of the AES algorithm on FPGA has been proposed in this paper[32].Speed enhancement has been achieved in this work by insertion of compact and flexible architecture for the Mix-column transformation operation. It also proposes to use a fixed coefficient multiplier for MixColumn transform. High throughput is achieved by incorporating a change in the inner process order in round transformation [35]. This paper proposes a method for integrating AES encrypter and decrypter, thus developing low-complexity architecture. Hardware resources used in the Sub-Bytes module and mix columns module have been saved thus making it an efficient design for both encryption and decryption, suitable for PDAs, mobile phones and smart cards[34].A significant increase in throughput of about 230% has been achieved in the architecture proposed in by using a block and key size of 512-bits.The architecture for AES algorithm was implemented on FPGA. High resistance to cryptanalysis attacks was achieved with only a minimal increase in area cost[40].A higher throughput/area was achieved in the architecture proposed .The paper proposes a FPGA implementation for data storage encryption [42].

III. CRYPTOGRAPHY

Cryptography is the study of encryption principles/methods. It plays an important role in the security of data transmission. Cryptography is governed by encryption algorithms. These are broadly classified as Symmetric or Asymmetric algorithms based on the kind of Keys used. The symmetric algorithms are also called single key or secret key or private key encryption, similarly asymmetric algorithm are also called two key or public key encryption respectively. Since Symmetric algorithms use only one key, the key needs to be private or secret to maintain confidentiality. So both the sender and the receiver share a common secret key. Asymmetric algorithms involve the use of two keys. One is the public key, which may be known to anybody, and can be used to encrypt messages, and verify signatures while the other is the private key known only to the recipient, used to decrypt messages, and sign (create) signatures. Asymmetric algorithms are computationally more intensive as compared to the Symmetric algorithms. There are still disputes over which kind of algorithm is more secure but Symmetric Ciphers are always preferred over Asymmetric Ciphers for speed and simplicity. Advanced Encryption Standard, based on the Rijndael algorithm is one such Symmetric algorithm for Encryption.
The National Institute of Standards and Technology (NIST) have 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. NIST has solicited candidate algorithms for inclusion in AES, Fifteen, (15), algorithms were selected from the first set of submittals. After a study and selection process five, (5), were chosen as finalists. The five algorithms selected were MARS, RC6, RIJNDAEL, SERPENT and TWOFISH. The conclusion was that the five Competitors showed similar characteristics. On October 2nd 2000, NIST announced that the Rijndael Algorithm was the winner of the contest.
The Rijndael Algorithm was chosen since it had the best overall scores in security, performance, efficiency, implementation ability and flexibility, [NIS00b]. The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Fijmen of Katholieke University at Leuven. The AES (Rijndael) algorithm is suitable for both hardware and software applications.

IV. ADVANCED ENCRYPTION STANDARD (AES)

The AES is a cryptographic algorithm that is used to encrypt (encipher), and decrypt, (decipher), information. Key Expansion generates a Key Schedule that is used in Cipher and Inverse Cipher procedures. Cipher and Inverse Cipher are composed of specific number of rounds (Table 1). For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key length [1].

Table 1 Comparison of block size, key length and number of rounds of AES keys.

Type	**Block Size Nbwords**	**Key Length Nkwords**	**Number of Rounds Nr**
AES -128 bits key	4	4	10
AES -192 bits key	4	6	12
AES -256 bits key	4	8	14

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte oriented transformations: SubBytes, ShiftRows, MixColumns and Add Round Key. Fig. 1 shows the basic schematic of the structure [4] of AES.

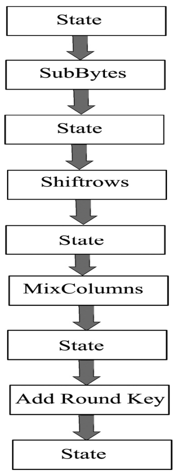


Fig 1: Basic structure of AES

4.1 AES operation

Fig2 indicates [3] the transformation in AES algorithm based on the structure in Fig 1.The brief introduction is listed as below:

1) SubBytes: The SubBytes operation is a non- linear byte substitution that operates on each byte of the State using a substitution table.

2) ShiftRows: In the ShiftRows operation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes.

3) MixColumns: Mixing operation which operates on the columns of the State using a linear transformation.

4) Add Round Key: A Round Key is added to the State by a simple bitwise XOR operation.

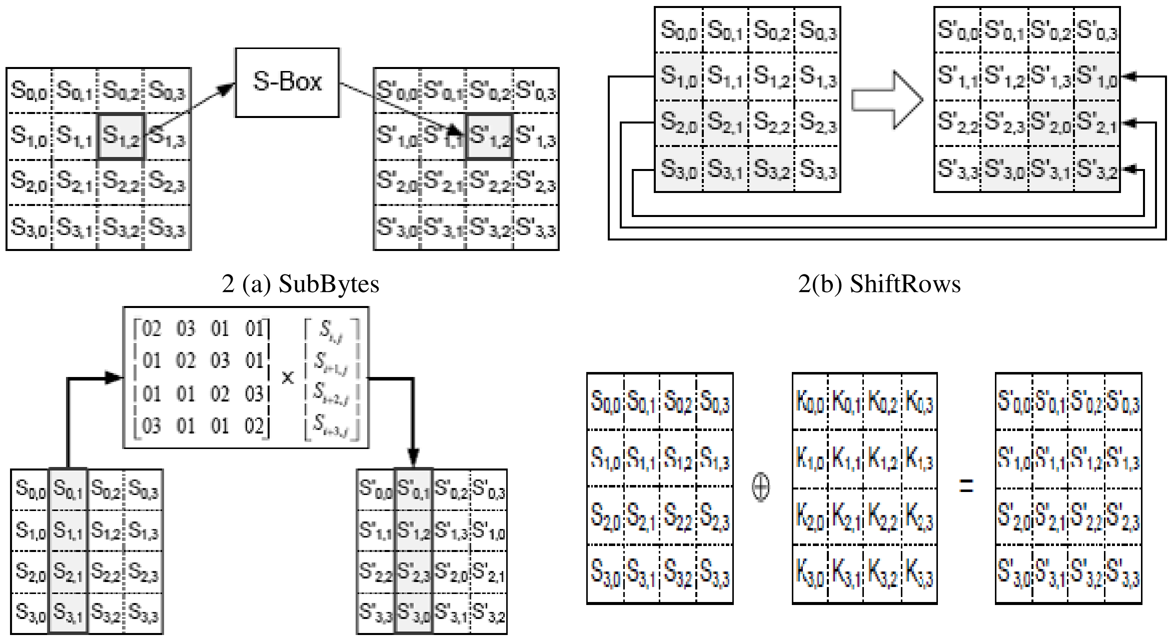


Fig. 2 Transformations in AES Algorithm

4.2 Inputs and outputs

The input and output for the AES algorithm consists of sequences of 128 bits. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. The basic unit for processing in the AES algorithm is a byte (a sequence of eight bits), so the input bit sequence is first transformed into byte sequence. In the next step a two-dimensional array of bytes (called the State) is built. The State array consists of four rows of bytes, each containing Nb bytes, where Nb is the block size divided by 32 (number of words). All internal operations (Cipher and Inverse Cipher) of the AES algorithms are then performed on the State array, after which its final value is copied to the output (State array is transformed back to the bit sequence).

4.3 Cipher

Using round function, which is composed of four different byte-oriented transformations, the Cipher converts input data (the input data is first copied to the State array) to an unintelligible form called ciphertext. After an initial Round Key addition, the State array is transformed by implementing a round function with the final round differing slightly from the first Nr−1 (Table 1) rounds. The round function is parameterized using a key schedule that consists of a one dimensional array of four-byte words (Round Key) derived using the Key Expansion routine. All Nr rounds (see Table 1) are identical with the exception of the final round, which does not include the MixColumns transformation.

4.4 Key Schedule

Key scheduling is a critical process in AES that generates (Nr+1) round keys based on a external single key. The Key expansion process of AES algorithm uses a Cipher Key K to generate a key schedule. This generates Nb(Nr+1) words, of which the algorithm requires initial Nb words and each of the Nr rounds require Nb words of Key Data. This results in a key schedule of a linear array of 4-byte words denoted by [wi], where $0 \leq i < \boldsymbol{Nb}(\boldsymbol{Nr} + 1)$.The SubWord( ) function accepts a 4-byte input word and on each of the four bytes S-box transformation is applied. The RotWord( ) performs cyclic permutations on the input word[a0,a1,a2,a3] and returns [a1,a2,a3,a0].The round constant word array or Rcon[i] holds
[x^{i-1},{00},{00},{00}], where in x^{i-1} being powers of x (x is denoted as {02}) in the field GF(2^8).The key expansion routine for 128-bit using 10 rounds has been shown in figure 3.
Key scheduling can produce keys either on-the –fly, or store them in an internal key memory during the key setup phase and then read them from this memory whenever required by the encryption/decryption unit. The critical path of Key Expansion is shorter than that of any round, speed of the system can't be enhanced by reducing the critical path of Key Expansion. Generating round keys on the fly eliminates the requirement for key storage, but brings overhead for decryption since decryption begins after the last roundkey is generated.

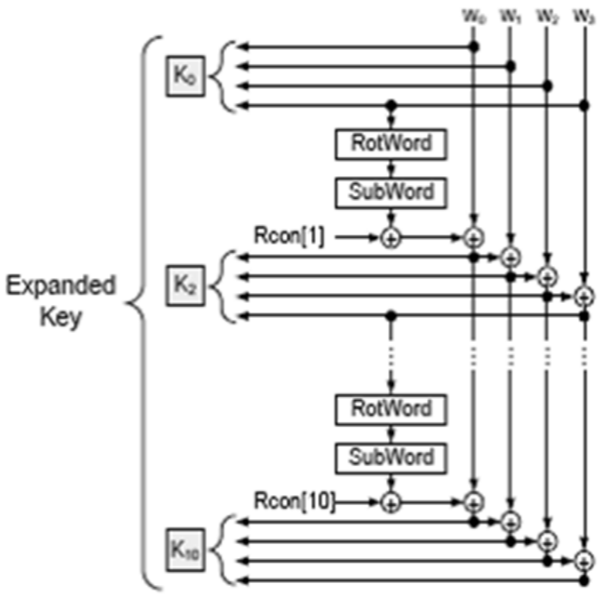


Figure 3: AES 128 bit key expansion operations

4.5 Inverse Cipher

At the start of the Inverse Cipher, the input (ciphertext) is copied to the State array. After Round Key addition (the last Round Key is added), the State array is transformed by implementing a round function, that is composed of three different inverse transformations and Add Round Key transformation (Round Keys are applied in the reverse order when decrypting), with the final round differing slightly from the first Nr−1 rounds. So this procedure converts ciphertext back to its original form called plaintext. All Nr rounds are identical with the exception of the final round, which does not include the Inverse Mix Columns transformation
The Block diagram of AES algorithm [2] is sketched in Figure 4.

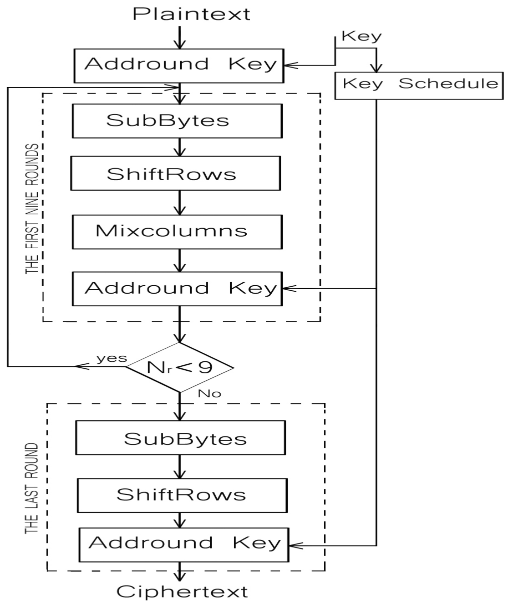


Fig.(a) Encryption

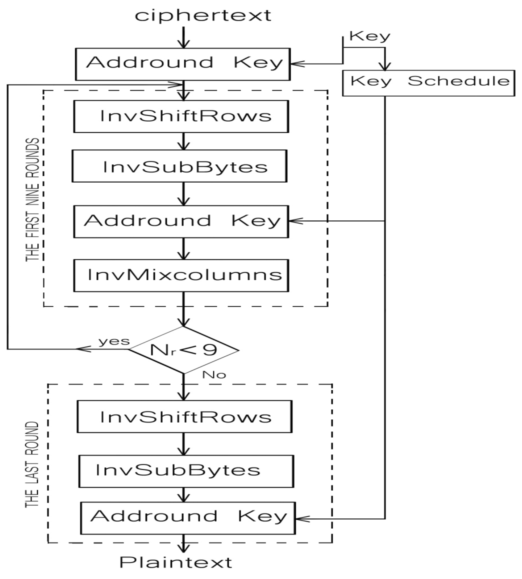


Fig.(b) Decryption

Fig. 4 Block diagram of AES algorithm

4.6 The algorithm

Rijndael is designed to use only simple byte (8-bits) operations. The encryption of a data block is composed of an initial XOR step with a temporary key. Next step is several rounds of transformations (depending on the key or block size) and an additional round performed and at the end with one round omitted. In case of 128 bits block size, there are 9 rounds, each involving the following four transformations (also known as round transformations):

ByteSub
ShiftRow
Mixcolumn
AddRoundKey

For the final step, the Mixcolumn transformation is not performed. For data decryption, however, different transformations are used for the first three round transformations, respectively InvByteSub, InvShiftRow and InvMixColumn. In addition, the round transformations are applied in an inverse sequence inside one round. This makes the encryption and the decryption procedures different, enforcing separate performance investigation for both directions.

V. Approaches for efficient hardware implementation of AES: A Survey

The optimization techniques for implementation of AES can be broadly classified into two categories: Architectural optimization and Algorithmic optimization.

Architectural optimization deals with the strength of pipelining, sub-pipelining and loop unrolling. The speed can be increased at the cost of increased area. Whilst Algorithmic optimization deals with algorithmic strength inside each round unit.
Before proceeding to the architectural optimization techniques let us go through a few Hardware architectures of symmetric block ciphers.

5.1 Hardware architectures of symmetric block ciphers

Symmetric-key block ciphers are used in several operating modes. From the point of view of hardware implementations, these modes can be divided into two major categories:
1. Non-feedback modes, such as Electronic Code Book mode (ECB) and counter mode (CTR).
2. Feedback modes, such as Cipher Block Chaining mode (CBC), Cipher Feedback Mode (CFB), and Output Feedback Mode (OFB).
In the non-feedback modes, each subsequent block of data is encrypted independently from processing other blocks. In particular, all blocks can be encrypted in parallel. In the feedback modes, encryption of the next block of data cannot be completed until encryption of the previous block is completed. As a result, all blocks must be encrypted sequentially, with no possibility of parallel processing. As a result of which the advantage of hardware implementations of secret key ciphers, based on parallel processing of multiple blocks of data could not be utilized. The feedback modes offer great deal of security but barely achieve any speed .Whilst ECB(non-feedback mode) mode offers less security but achieves a great speedup by parallel processing of multiple blocks of data. Hence The NIST recommendations on the AES modes of operation a counter mode(non-feedback mode) provided some respite.

5.2 Architectural Optimization

Three types of architectures can be used to increase the speed of encryptor/decryptor by duplicating hardware for implementing each round. These architectures are based on- Pipelining, Sub-pipelining, loop unrolling.

5.2.1 Pipelining

Pipelining processes multiple blocks of data simultaneously thus resulting in a speed up of encryption/decryption operations. The basic iterative architecture is shown in figure 5(a). It is constructed by inserting rows of registers among combinational logic. The rounds between consecutive registers form the pipelined structure of one stage. At every clock cycle the processed data moves to the next stage and its position is occupied by the subsequent data block. The number of round units k is chosen as a divisor of the total number of rounds Nr. If k equals Nr then it is called fully pipelined architecture. If the available circuit area is not large enough to fit all cipher rounds, architecture with partial outer-round pipelining is applied as shown in figure 5(b). However if the available area is large enough to fit all cipher rounds, architecture with full outer-round pipelining can be applied as shown in figure 5(c),[46][47].

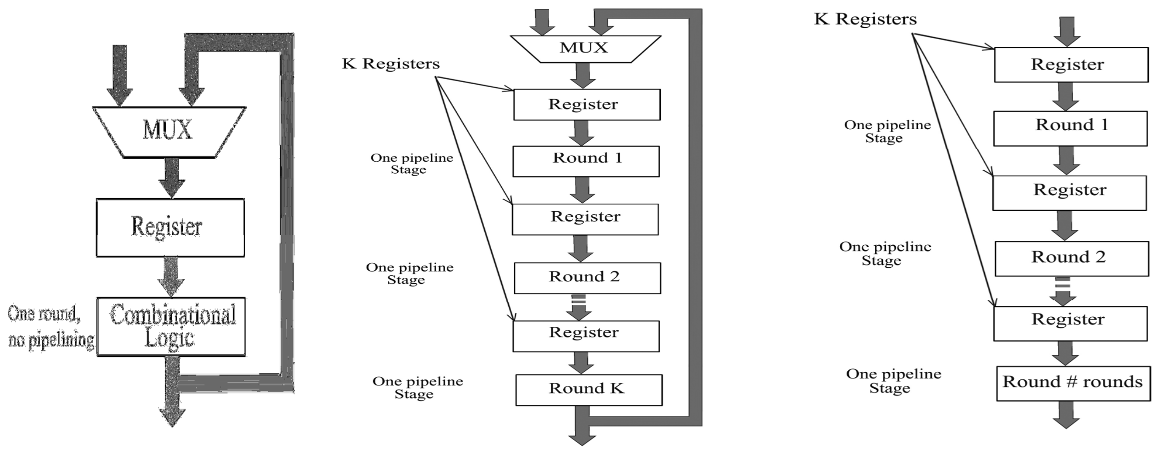


Fig.(a) Fig.(b) Fig.(C)

Fig. 5. a) basic iterative, b) with partial outer-round pipelining, c) with full outer-round pipelining

5.2.2 Sub-Pipelining

In case of sub-pipelining, the optimum number of pipeline registers is inserted inside of a cipher round, as shown in figure.6(a). The entire round, including internal pipeline registers is then repeated K times as shown in figure 6(b). The number of unrolled rounds K depends on the maximum available area or the maximum required throughput. That is when the circuit area is limited, mixed inner- and outer-round pipelining offers significantly higher throughput. However if the circuit area limit is large enough, all rounds of the cipher can be unrolled, as shown in figure 6(c). Inserting registers inside of a cipher round significantly increases cipher throughput at the cost of only marginal increase in the circuit area. Inserting additional registers may still increase the circuit throughput, but the throughput to area ratio will deteriorate [46][47].

5.2.3 Loop unrolling

An architecture with partial loop unrolling is shown in figure7(a). In the partial loop unrolling the combinational part of the circuit implements K rounds of the cipher, instead of a single round. Once again K must be a divisor of the total number of rounds, Nr. The number of clock cycles necessary to encrypt a single block of data decreases by a factor of K. At the same time the minimum clock period increases by a factor slightly smaller than K, leading to an overall relatively small increase in the encryption throughput, and decrease in the encryption latency. The combinational part of the circuit accounts for most of the circuit area; consequently the total area of the enc./dec. unit increases proportionally to unrolled rounds K.

Architecture with full loop unrolling is shown in figure7(b). The input multiplexer and the feedback loop become unnecessary, causing a small increase in the cipher speed and decrease in the circuit area compared to the partial loop unrolling. In summary, loop unrolling enables increasing the circuit speed in both feedback and non-feedback operating modes. The increase however is minimal and benefits limited since it comes with a large increase in area. As a result, choice of this architecture is best suited only for feedback cipher modes, where no other architecture offers speed greater than the basic iterative architecture, and only for implementations where large increase in the circuit area can be tolerated[46][47].

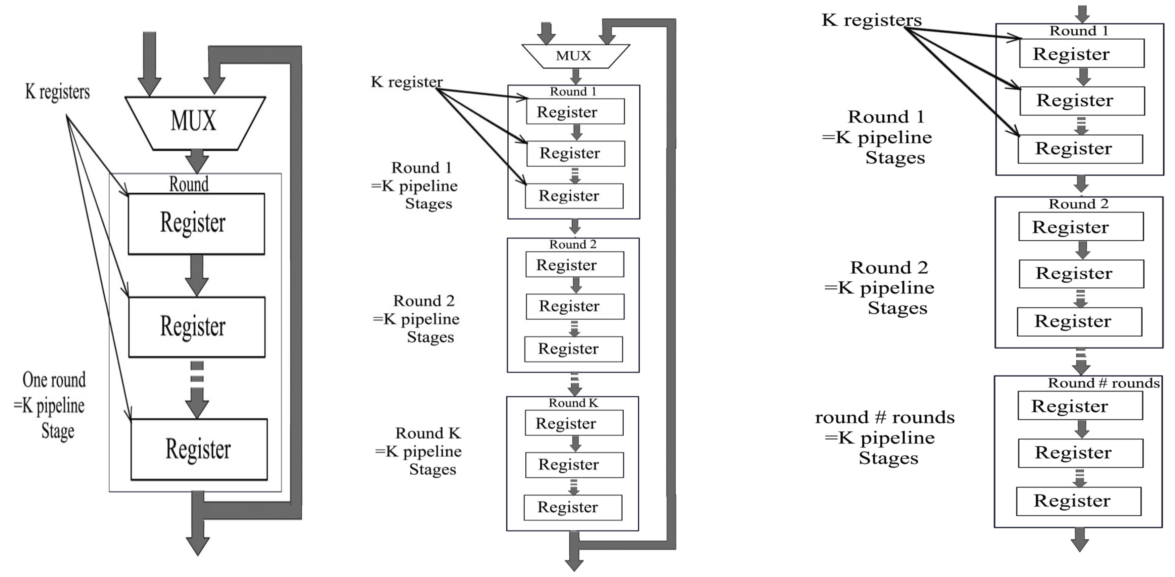


Fig.(a) Fig.(b) Fig.(C)

Fig. 6.a) with inner-round pipelining, b)with partial inner-and outer-round pipelining, c)with full inner-and outer- round pipelining

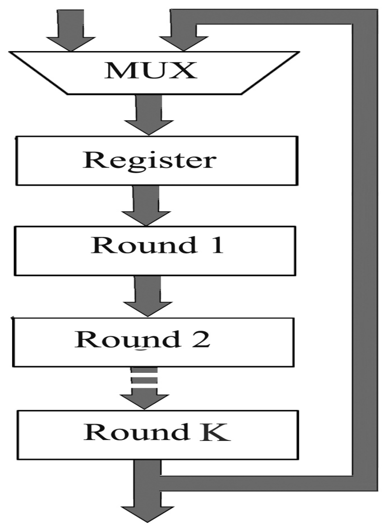


Fig.(a)

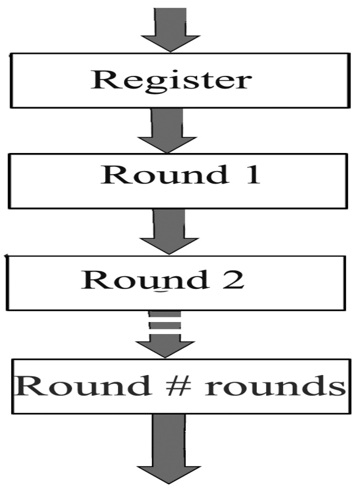


Fig.(b)

Fig. 7. a) With partial loop unrolling, b) With full loop unrolling

5.3 Algorithmic Optimization

A particular round consists of Shift Rows/Inv.Shift Rows, Add Round Key Transformation, Sub-Bytes/Inv.Sub-Bytes and Mix Rows/Inv.Mix Rows. Algorithmic optimization handles optimization techniques used over these. But Shift Rows/Inv.Shift Rows transformation involves no logic gates whilst the Add Round Key Transformation requires just one XOR operation. Therefore no optimization can be performed over these. However optimizations can be performed over Sub-Bytes/Inv.Sub-Bytes and Mix/Inv.Mix Columns transformations which have been discussed in the following section[46][48][49].

5.3.1 Implementation of SubBytes/Inverse SubBytes(S-Box/S^{-1}- Box)

The Sub-Bytes/Inv.Subbytes transformation is usually implemented using lookup tables or LUTs. This however consumes a huge area especially when multiple rounds are implemented.

5.3.1.1 S – BOX

Substitution table (s-box) is used in order to perform sub bytes transformation on each byte of the state. Construction of s-box will involve computation of multiplicative inverse of each element in GF(2^8), followed by an affine transformation and this can be viewed as

$$AT(a) = \begin{pmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \qquad (1)$$

Where AT is the affine transformation while the vector 'a' is the multiplicative inverse of the input byte from the state array, [46][48].

5.3.1.2 S^{-1} – Box

Inverse subbyte transformation referred as Inv subbytes, is performed by applying inverse s-box to individual bytes of the state. The inverse s-box(s^{-1} box) is obtained by considering inverse of the affine transformation followed by computation of multiplicative inverse in GF(28)and this can be viewed as

$$AT^{-1}(a)=\begin{pmatrix}0&1&0&1&0&0&1&0\\0&0&1&0&1&0&0&1\\1&0&0&1&0&1&0&0\\0&1&0&0&1&0&1&0\\0&0&1&0&0&1&0&1\\1&0&0&1&0&0&1&0\\0&1&0&0&1&0&0&1\\1&0&1&0&0&1&0&0\end{pmatrix}\times\begin{pmatrix}a_7\\a_6\\a_5\\a_4\\a_3\\a_2\\a_1\\a_0\end{pmatrix}\oplus\begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\1\end{pmatrix} \quad (2)$$

Where $AT^{-1}$is the affine transformation while the vector ‘a’ is the multiplicative inverse of the input byte from the state array, [46][48].

5.3.1.3 Joint S - Box

Now, we see that from above, both transformations (s-box and s^{-1} box) use the same multiplicative inverse in GF(2^8). Therefore, both transformations can share the same multiplicative inverse module and they can be implemented using a combined architecture as shown in figure 8.

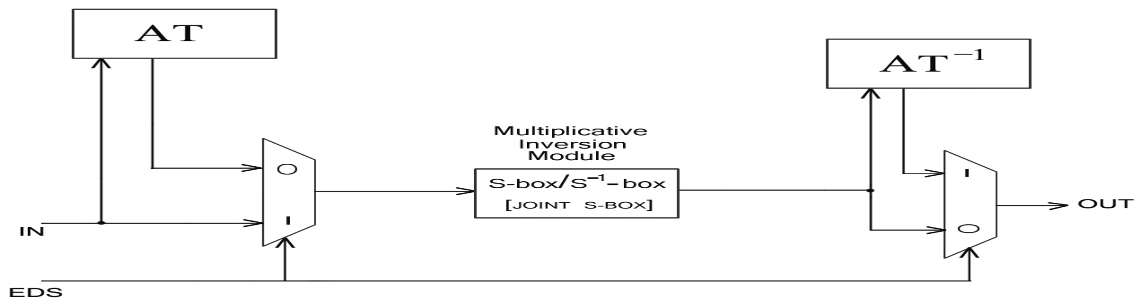


Figure 8. Combined architecture of subbyte/Inv subbyte transformation.

Subbyte or Inverse subbyte transformation can be suitably selected by selecting the signal, EDS(encrypt decrypt select), to a value of ‘0’(subbyte) or ‘1’ (Inverse subbyte)[48].

5.3.1.4 S-Box Construction Methodology using composite field arithmetic

In architectures where area constraints become important a better choice would be to map the arithmetic operations on GF(2^8) to isomorphic field GF($(2^4)^2$). This implementation requires smaller area for look-up tables but has longer delay[11][43].

In a GF(2^8) element each bit can be considered to be the coefficient to corresponding power term in GF(2^8) polynomial. bx+c represents any arbitrary polynomial ,given an irreducible polynomial x^2+Ax+B, b referring to most significant nibble while c stands for the least significant nibble. The multiplicative inverse can be obtained using the equation as shown:

$$(bx+c)^{-1}=b(b^2\lambda+c(b+c))^{-1}x+(c+b)(b^2\lambda+c(b+c))^{-1}..................(3)$$

The above equation (3) shows the multiplication, addition, squaring and multiplicative inverse operations in GF(2^4).Individual blocks can be used for each of these operators. From the above simplified equation(3), multiplicative inverse circuit for GF(2^8) can be produced as shown in figure 9,[48].

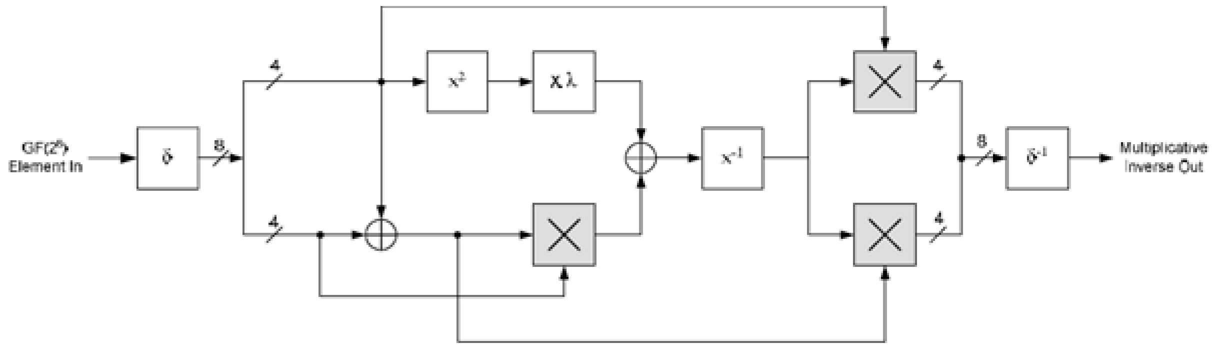


Figure 9. Multiplicative inversion module for the S-Box.

The indexes for the blocks within the multiplicative inversion module from above are illustrated in the Figure 10.

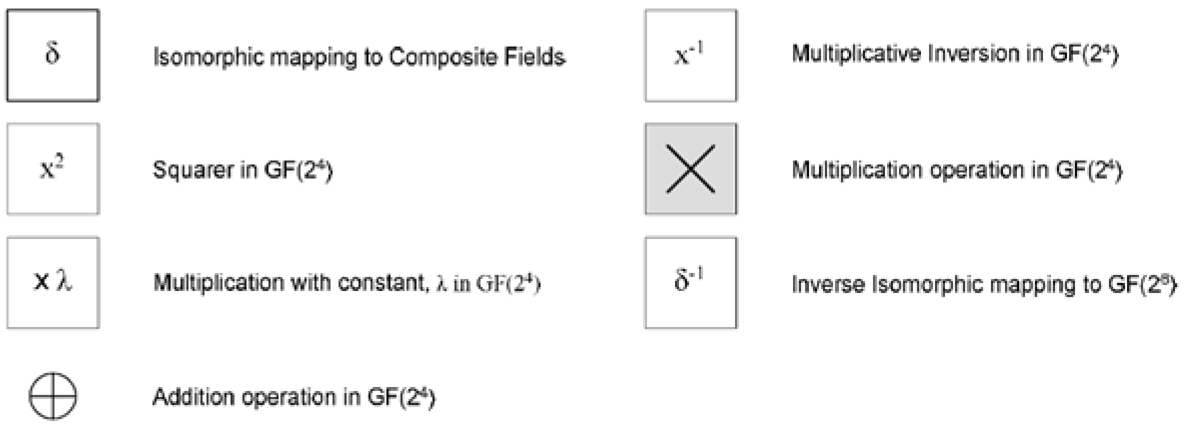

δ
Isomorphic mapping to Composite Fields
x²
Squarer in GF(2⁴)
x λ
Multiplication with constant, λ in GF(2⁴)
Addition operation in GF(2⁴)
x⁻¹
Multiplicative Inversion in GF(2⁴)
Multiplication operation in GF(2⁴)
δ⁻¹
Inverse Isomorphic mapping to GF(2⁸)

Figure 10. Index for the building blocks within the multiplicative inversion module.

5.3.1.5 Isomorphic and Inverse Isomorphic Mapping

Computation of Multiplicative inverse can be achieved by decomposition of complex GF(28) to lower order fields of GF(2^1),GF(2^2),GF($(2)^2$).To perform this the following irreducible polynomials are used(4).

$$\begin{aligned} GF(2^2) &\rightarrow GF(2) &&: x^2+x+1 \\ GF((2^2)^2) &\rightarrow GF(2^2) &&: x^2+x+\varphi \\ GF(((2^2)^2)^2) &\rightarrow GF((2^2)^2) &&: x^2+x+\lambda \end{aligned} \qquad (4)$$

Where $\varphi=\{10\}_2$ and $\lambda=\{1100\}_2$

Multiplicative inverse cannot be found directly for an element based on GF(2^8).Mapping to composite field through an isomorphic function δ becomes essential. Once the multiplicative inversion has been performed conversion from composite field back to its GF(2^8) equivalent through δ^{-1} becomes essential. This is performed using a 8X8 matrix.

5.3.2 Implementation of Mix Columns/ Inverse Mix Columns

5.3.2.1 Mix column (MC)

The Mix Columns (MC) transformations operate column-by-column on the 4*4 state arrays.
The matrix expression of the MC on the Cth column can be expressed as

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{bmatrix} \bullet \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad (5)$$

In the Mix Columns transformation, we implement constant multiplication of {02} and{03} in GF(28). Suppose X is a byte in the State, {02}X can be implemented by shifting and bit-wise XOR operations, and {03}X can be computed by ({02}X) ⊕X, where ⊕denotes Exclusive-OR operation.
If X is expressed in binary form as {x7, x6, x5, x4, x3, x2,x1, x0}, {02}X can be calculated by
{02}X = {x7, x6, x5, x4, x3, x2, x1, x0, 0} ⊕{0, 0, 0, x7, x7, 0, x7, x7}-------(6)
From (5)[shown in the figure.11],the critical path has 4 XOR gates and hence (4 x 8 + 4)x 4 = 144 two-input XOR gates are needed to implement the Mix Columns transformation for one column of the State[46].

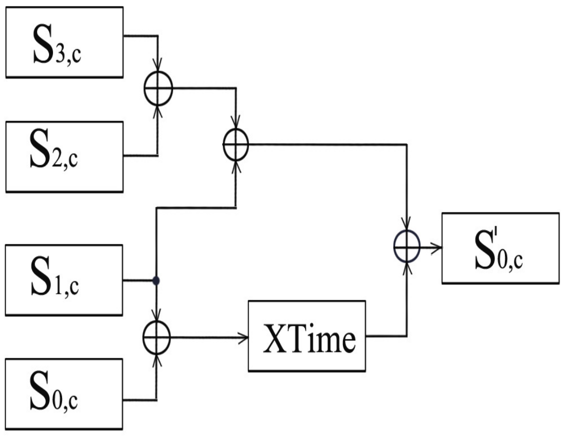


Figure12. Block diagram for substructure sharing implementation of Mix Columns transformation[46]

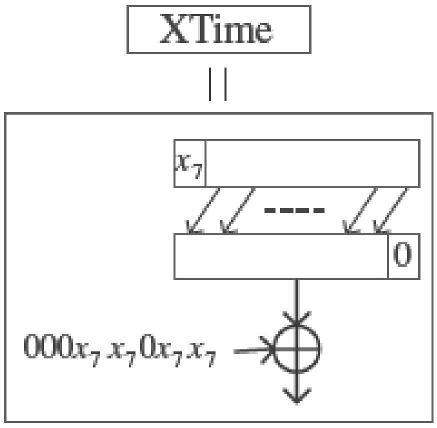


Figure 13. Block diagram of XTime.[46]

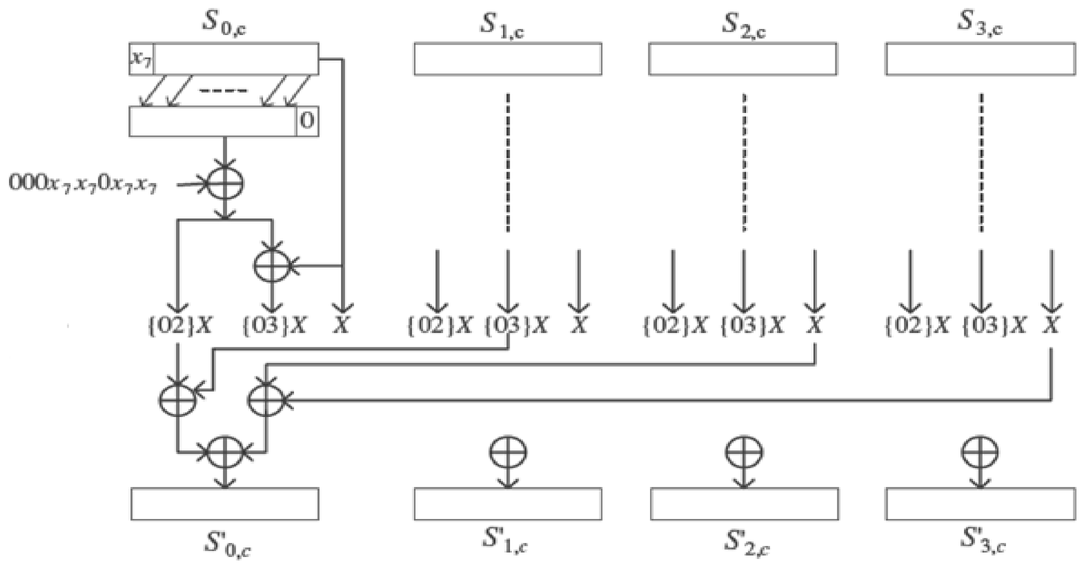


Figure 11. Block diagram for straightforward implementation of the MixColumns transformation[46]

Alternate method used to implement MixColumn transformations have been proposed in [44],[45].The idea of substructure sharing has been studied in these.

$S'_{0,C} = [\{02\}\ \{03\}\ \{01\}\ \{01\}] \times [S_{0,C}\ S_{1,C}\ S_{2,C}\ S_{3,C}]^T$ can be rewritten as:

$$S'_{0,C} = \{02\}(S_{0,C} + S_{1,C}) + S_{1,C} + (S_{2,C} + S_{3,C}), \tag{7}$$

Which can then be realized as shown in fig 12.

The computation of the other bytes in the State can be implemented by similar structure and the same number of XOR gates. Compared to Fig.10, the total number of XOR gates for computing one column of the State remains 144, but the critical path has been reduced to 3 XOR gates.

5.3.2.2 Inverse mixcolumn (IMC)

Inv Mix Columns is the inverse transformation of Mix Columns. Similar to Mix Columns transformation, The Inverse Mix Columns (IMC) transformations operate column-by-column on the 4*4 state arrays

The matrix expression of IMC on the Cth column can be expressed as

$$\begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} = \begin{bmatrix} \{0e\} & \{0b\} & \{0d\} & \{09\} \\ \{09\} & \{0e\} & \{0b\} & \{0d\} \\ \{0d\} & \{09\} & \{0e\} & \{0b\} \\ \{0b\} & \{0d\} & \{09\} & \{0e\} \end{bmatrix} \bullet \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{8}$$

The IMC transformation used in decryption is more complicated. Constant multiplications used in the Inv Mix Columns transformation can be expressed as

$\{0b\}X = \{08\}X \oplus \{02\}X \oplus X$, $\{0d\}X = \{08\}X \oplus \{04\}X \oplus X$, $\{09\}X = \{08\}X \oplus X$,
$\{0e\}X = \{08\}X \oplus \{04\}X \oplus \{02\}X$.

$S'_{0,C}$ $(0 \leq c < 4)$ can be calculated in the IMC transformation is illustrated in Fig. 14. An XTime block which is shown in Fig. 13 is used in fig 14.to simplify the diagram. XTime block implements the constant multiplication by {02} in $GF(2^8)$, each XTime block consists of 4 XOR gates and the critical path includes only one XOR gate. In the IMC transformation, the calculation for the other bytes can be carried out similarly according to (8). As shown in Fig. 14, the critical path is 6 XOR gates, and a total of (10 x 8 +3 x 4) x 4 = 368 XOR gates is needed to implement the MixColumns transformation for one column of the State[46].

The same substructure sharing idea can be used in the InvMixColumns transformation. For example, the bytes in row one of the State are calculated by

$$S'_{0,C} = [\{0e\}\ \{0b\}\ \{0d\}\ \{09\}] \times [S_{0,C}\ S_{1,C}\ S_{2,C}\ S_{3,C}]^T, \tag{9}$$

which can be rewritten as

$$S'_{0,C} = \{04\}(\{02\}(S_{0,C} + S_{1,C}) + \{02\}(S_{2,C} + S_{3,C}) + (S_{0,C} + S_{2,C})) + \{02\}(S_{0,C} + S_{1,C}) + S_{1,C} + (S_{2,C} + S_{3,C}). \tag{10}$$

The equation (10) can be implemented as shown in Fig.15.Compared to the straightforward implementation in Fig. 14, the number of XOR gates is reduced to (8 x 8 + 4 x 4) x 4 = 320 for computing one column of the State, but the critical path is increased to 7 XOR gates.

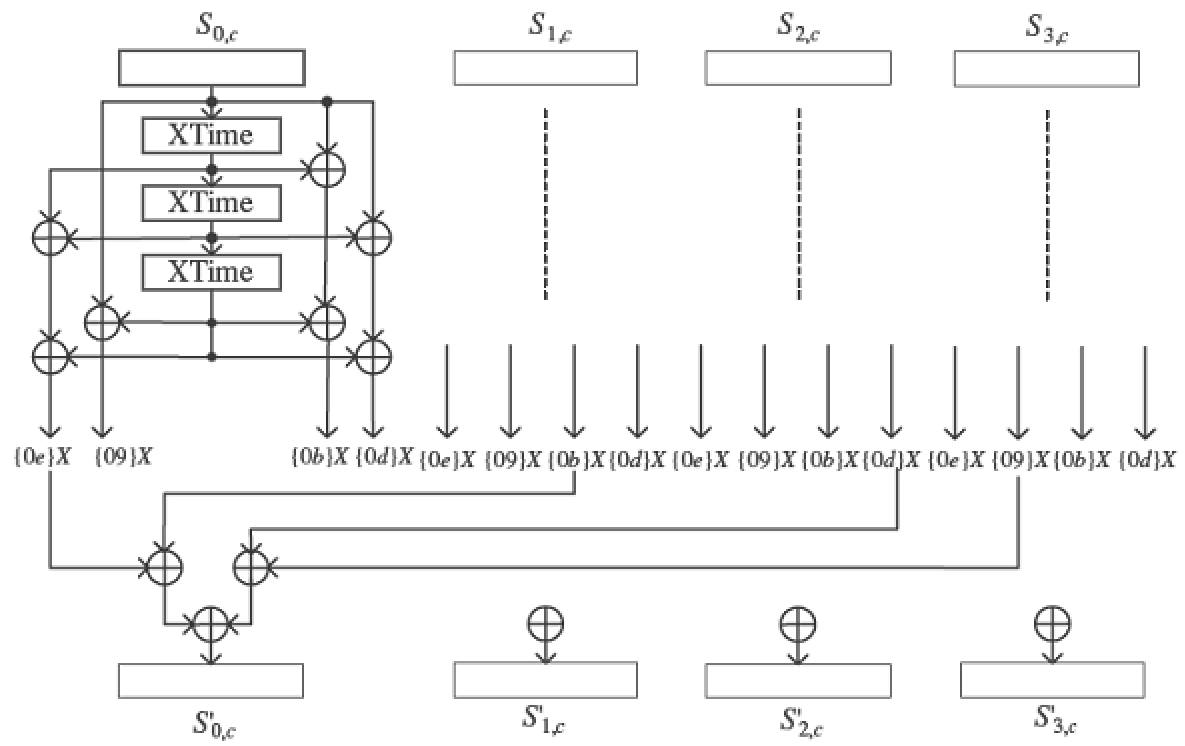
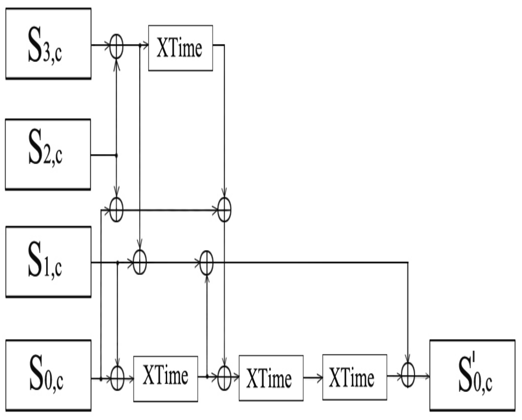


Figure 14. Block diagram for straight forward implementation of the Inv Mix Columns transformation[46]



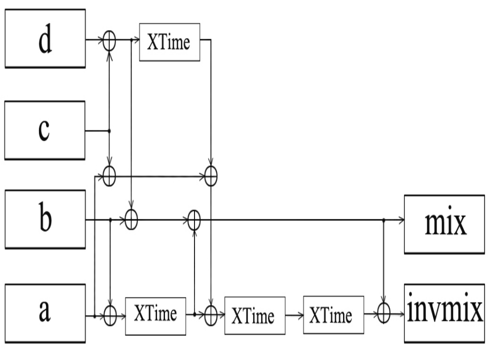


Figure 15. Block diagram for substructure sharing implementation of the Inv Mix Columns transformation.[46] transformations.[46]

Figure 16. Joint implementation of the MixColumns and the Inv Mix Columns (bytes in the first row of the State)

Although (10) leads to an Inv- MixColumns implementation with neither the shortest delay nor the smallest area, combined with (7), it leads to the hardware implementation with least area of joint MixColumns/ InvMixColumns transformation. Figure 16 illustrates the diagram according to (7) and (10). The four inputs, ‘a’, ‘b’, ‘c’, ‘d’, and two outputs, ‘mix’ and ‘invmix’, all represent single bytes. ‘a’, ‘b’, ‘c’, ‘d’ are the four bytes in a column of the State with ascending row numbers. ‘mix’ and ‘invmix’ are the outcomes of applying MixColumns and InvMixColumns transformation to the inputs, respectively[46].

5.3.3 T- Box / Inverse T-Box Approach

5.3.3.1 T-Box

The subbytes, shiftrows and mixcolumns transformations are combined and can be implemented as a whole. This is referred as T-box approach. Starting from the SubBytes transformation, the updated

State after the MixColumns transformation can be expressed as shown in (11), for $0 \le c < Nb$ [46][49].

$$\begin{array}{c}\left[\begin{array}{cccc} 02*S_{0,C} & +\,03*S_{1,C} & +\,01*S_{2,C} & +\,01*S_{3,C} \\ 01*S_{0,C} & +\,02*S_{1,C} & +\,03*S_{2,C} & +\,01*S_{3,C} \\ 01*S_{0,C} & +\,01*S_{1,C} & +\,02*S_{2,C} & +\,03*S_{3,C} \\ 03*S_{0,C} & +\,01*S_{1,C} & +\,01*S_{2,C} & +\,02*S_{3,C} \end{array}\right] \\ \begin{array}{cccc} \uparrow & \uparrow & \uparrow & \uparrow \\ T_0 & T_1 & T_2 & T_3 \end{array}\end{array} \tag{11}$$

T-Box Tables for Encryption:

$$T_0[S^{'}_{0,c}]=\begin{bmatrix} 02* & S_{0,C} \\ 01* & S_{0,C} \\ 01* & S_{0,C} \\ 03* & S_{0,C} \end{bmatrix} \quad T_1[S^{'}_{1,c}]=\begin{bmatrix} 03* & S_{1,C} \\ 02* & S_{1,C} \\ 01* & S_{1,C} \\ 01* & S_{1,C} \end{bmatrix} \quad T_2[S^{'}_{2,c}]=\begin{bmatrix} 01* & S_{2,C} \\ 03* & S_{2,C} \\ 02* & S_{2,C} \\ 01* & S_{2,C} \end{bmatrix} \quad T_3[S^{'}_{3,c}]=\begin{bmatrix} 01* & S_{3,C} \\ 01* & S_{3,C} \\ 03* & S_{3,C} \\ 02* & S_{3,C} \end{bmatrix} \tag{12}$$

The resultant matrix shown above(in 12) has each column with fixed constants multiplied by state inputs. These state inputs are updated with the next row elements for the next iteration. That is, first row elements of the state inputs are always multiplied with the first column constants of the multiplication matrix. Similarly, the second row elements of the state inputs are always multiplied with the second column constants of the multiplication matrix and so on. The first row of the state input uses table T_0, the second row uses the table T_1, the third row uses the table T_2 and fourth row uses the table T_3.
Finally, the combination of subbytes, shiftrows and mixcolumns transformations can be implemented by XORing the output's of T-boxes. In the last round of encryption, there is no mixcolumn transformation and so S-box is used instead of T-box[49].

5.3.3.2 Inverse T-Box

The Invsubbytes, Invshiftrows and Invmixcolumns transformations can also be combined and implemented as a whole. This is referred as Inverse T-box (T^{-1} box) approach. Starting from the InvSubBytes transformation, the updated State after the InvMixColumns transformation can be expressed as shown in (13), for $0 \le c < Nb$

$$\begin{array}{c}\left[\begin{array}{cccc} 0E*S_{0,C} & +\,0B*S_{1,C} & +0D*S_{2,C} & +\,09*S_{3,C} \\ 09*S_{0,C} & +\,0E*S_{1,C} & +\,0B*S_{2,C} & +\,0D*S_{3,C} \\ 0D*S_{0,C} & +\,09*S_{1,C} & +\,0E*S_{2,C} & +\,0B*S_{3,C} \\ 0B*S_{0,C} & +\,0D*S_{1,C} & +\,09*S_{2,C} & +\,0E*S_{3,C} \end{array}\right] \\ \begin{array}{cccc} \uparrow & \uparrow & \uparrow & \uparrow \\ T_0^{-1} & T_1^{-1} & T_2^{-1} & T_3^{-1} \end{array}\end{array} \tag{13}$$

T-Box Tables for Decryption:

$$T_0^{-1}[S'_{0,c}]=\begin{bmatrix}0E^* & S_{0,C}\\09^* & S_{0,C}\\0D^* & S_{0,C}\\0B^* & S_{0,C}\end{bmatrix}\quad T_1^{-1}[S'_{1,c}]=\begin{bmatrix}0B^* & S_{1,C}\\0E^* & S_{1,C}\\09^* & S_{1,C}\\0D^* & S_{1,C}\end{bmatrix}\quad T_2^{-1}[S'_{2,c}]=\begin{bmatrix}0D^* & S_{2,C}\\0B^* & S_{2,C}\\0E^* & S_{2,C}\\09^* & S_{2,C}\end{bmatrix}\quad T_3^{-1}[S'_{3,c}]=\begin{bmatrix}09^* & S_{3,C}\\0D^* & S_{3,C}\\0B^* & S_{3,C}\\0E^* & S_{3,C}\end{bmatrix} \tag{14}$$

Similar to the encryption case, the resultant matrix shown above (in 14) has each column with fixed constants multiplied by state inputs. These state inputs are updated with the next row elements for the next iteration. The first row of the state input uses table T_0^{-1}, the second row uses the table T_1^{-1}, the third row uses the table T_2^{-1} and fourth row uses the table T_3^{-1}.
Finally, the combination of Inverse subbytes, Inverse shiftrows and Inverse mixcolumns transformations can be implemented by XORing the output's of T^{-1}-boxes. In the last round of decryption, there is no IMC transformation and so S^{-1}-box is used instead of T^{-1}-box[49].
Today's research concentrates on improving these primary optimizations for better performance and area.

VI. PERFORMANCE SUMMARY

In an effort to summarize the work conducted in the relevant field so far we have chosen only the most suitable results of FPGA implementations. These results are summarized in the table 3. LUTs are Look Up Tables, CLBs are Configurable Logic Blocks, enc./dec. is encryption/decryption, RRPN is Reduced Residue Prime Numbers, CBC is (cipher Block Chaining) mode, CFB is (cipher feedback) mode, CTR (counter) mode.

Table2: Performance summary of state of the art

Name	**Year of Publication**	**Process**	**Number of Gates**	**Frequency/ Throughput**	**Comments**
P. Chodowiec et.al 2[5]	2001	Enc	12.6k CLB	95 MHz/ 12.2 Gbit/s	Pipelined
M. McLoone et.al 1 [6]	2001	Enc.	2.7kCLB+ 82RAM	54.4 MHz/ 6.95 Gbit/s	Pipelined
M. McLoone et.al 2 [6]	2001	Dec	4.3k CLB + 82 RAM	49.9 MHz/ 6.38 Gbit/s	Pipelined
N. Sklavos et.al 1[7]	2002	Enc/Dec	2358 CLB	22 MHz/ 259 Mbit/s	-------------
N. Sklavos et.al 2[7]	2002	Enc/Dec	17.3k CLB	28.5 MHz/ 3.65 Gbit/s	Pipelined
J. H. Shim et.al [8]	--------	Enc/Dec	2580 CLB	38.8 MHz/ 452 Mbit/s	-------------
R. Karri et.al [9]	2002	Enc	3973 CLB	47 MHz/ 137 Mbit/s	-------------
Nazar A. Saqib et.al 1 [12]	2003	Enc	2744 CLB	20.192MHz/ 258.5Mbits/s	sequential
Nazar A. Saqib et.al 2 [12]	2003	Enc	2136 CLB	22.41 MHz/ 2868 Mbits/s	pipelined
Refik Sever et.al [10]	2004	Enc/Dec	4189 CLB + 4 RAM	65 MHz/ 1.19 Gbit/s	Block RAM
A. Hodjat et.al[13]	2004	Enc	5177 Slices	168.3 MHz/ 21.54 Gb/s	Fully Pipelined
DeenKotturi et.al [14]	2005	Enc	5408 Slices	232.6 MHz/ 29.77 Gb/s	Fully Pipelined

Tim Good et.al[29]	2005	Enc/Dec	16,693 Slices	184.8 MHz/ 23,654 Mbps	Fully Parallel Loop Unrolled
Nalini C et.al [17]	2006	Enc/Dec	4626 CLB Slices	241.313MHz/ 30.88Gbps	Fully Pipelined
Monica Liberatori et.al[15]	2007	Enc	1643 Slices	91.049MHz/ 224.12 Mbps	Single round loop unrolling
Chi-Wu Huang et.al[18]	2007	Enc	148 slices	287 MHz/ 647Mbps	Block RAM
C. Sivakumar et.al[20]	2007	Enc	6766 CLB Slices	194 MHz/ 2257Mbps	CTR mode & CBC Mode
C. Sivakumar et.al[20]	2007	Dec	7301 CLB Slices	148 MHz/ 1722Mbps	Counter mode & CBC Mode
SwinderKaur et.al[21]	2007	Enc/Dec	6279 Slices	119.954MHz/ 1.18 Gbps	pipelining
Chih-Peng Fan et.al[30]	2008	Enc/Dec	139357 Slices	222.2MHz/ 28.4 Gbits/s	Fully pipelined
BanraplangJyrwa et.al[16]	2009	Enc/Dec	6211 Slices	142.5MHz/ 1458 Mbps	iterative approach
Muhammad H. Rais et.al[19]	2009	Enc	1745 slices.	242.153MHz/ 3.09 Gbps	RRPN
Yulin Zhang et.al[31]	2010	Enc	2389 Slices	271.15MHz/ 34.7 Gbps	Pipelined
Atul M. Borkar et.al[22]	2011	Enc/Dec	1853 Slices	140.390MHz/ 352 Mbits/sec	CFB mode
Gurmail Singh et.al [36]	2011	Enc	6352 Slices	347.6MHz/ 44.5Gbits/s	Fully pipeleined
Sumanth Kumar Reddy S et.al[28]	2011	Enc	8896 Slices	202.26 MHz/ 25.89Gbps	Fully sub pipelining
HadiSamiee et.al [39]	2011	Enc	7865 slices	341.53MHz/ 43.71 Gbps	Fully sub pipelining

VII. CONCLUSION

AES is a powerful cryptographic technique that is gaining popularity of late with NIST recommending in favour of it. Thus analysis of improvements in this field becomes all the more significant. With cryptanalysis attacks like the algebraic attacks growing both frequent and powerful it becomes increasingly important to develop a private key cryptosystem that is resistant to such attacks. Another important consideration is the area-throughput tradeoffs and resource and energy efficiency of hardware implementations. Though a high speed architecture is ever desirable the limitations imposed by hardware restrictions and area of circuits present the problem of selecting only those designs with optimum throughput/area ratios. Parallelism adds to the speed significantly but also increases the area costs considerably. Architectural optimization is not an effective solution in feedback mode. Loop unrolling is the only architecture that can achieve a slight speedup with significantly increased area. In non-feedback mode, sub pipelining can achieve maximum speed up and the best speed/area ratio. Therefore achieving a balance of the two poses the biggest challenge. Our objective is to choose a suitable hardware based on the factors mentioned.
The choice of an optimum hardware architecture for AES depends on the following major factors:

1. Optimization criteria, such as minimum area, minimum power consumption, maximum throughput, maximum throughput to area ratio, etc.
2. Support for feedback modes of operation, such as CBC and CFB, or non-feedback modes of operation, such as ECB and CTR modes.

3. Support for AES encryption only (e.g., in the block cipher modes of operation that require encryption only, such as CTR and CFB modes) or encryption and decryption (e.g., in the modes that require both operations, such as ECB and CBC).
4. Semiconductor technology of choice, such as ASIC or FPGA.
5. Resistance to side channel attacks, such as Differential Power Analysis, Timing Analysis, etc.

We have discussed Architectural and algorithmic optimization techniques for efficient hardware implementations of the AES algorithm. Additionally resource sharing issues of Encryptor and Decryptor when implemented on a common hardware have also been discussed. No design has been developed so far to implement all of the discussed optimization methods together as discussed in this paper. In the applications with limited area, generating round keys on the fly is a better choice.

Adding a note on future work, one could work on selection of a larger key size which would make the algorithm more secure, and a larger input block to increase the throughput. The extra increase in area can however be tolerated. Such an algorithm with high level of security and high throughput can have ideal applications in which such as in multimedia communications. Furthermore study of optimization approaches for the implementations supporting multiple key lengths and modes of operation have tremendous scope for future work.

REFERENCES

[1] DAEMEN, J.—RIJMEN, V.(1999): AES Proposal: Rijndael, The Rijndael Block Cipher, AES Proposal, pp. 1–45.(http://csrc.nist.gov/CryptoToolkit/aes/)

[2] Marko Mali, Franc Novak, Anton Biasizzo,(2005) "Hardware Implementation Of AES Algorithm", Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, Volume: 56, Issue: 9, Pages: 265-269

[3] Yibo Fan, Takeshi Ikenaga, YukiyasuTsunoo, and Satoshi Goto,(2008) "A Lowcost Reconfigurable Architecture for AES Algorithm" proceedings of world academy of science, engineering and technology volume 31 july 2008 ISSN 2070-3740

[4] William Stallings, "Cryptography and Network Security", Third Edition, www.williamstallings.com/Crypto3e.html

[5] P. Chodowiec, P. Khuon and K. Gaj,(2001) "Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining," Proc. ACM/SIGDA Int. Symposium on Field Programmable Gate Arrays, FPGA'01, Monterey, CA..

[6] M. McLoone and J. McCanny,(2001) "Single-chip FPGA Implementation of the Advanced Encryption Standard Algorithm," in Proc. 11th Int. Conf. Field-Programmable Logicand Applications (FPL 2001), LNSC 2147, pp. 152-161.

[7] N. Sklavos and O. Koufopavlou,(2002) "Architectures and VLSI Implementations of the AES-Proposal Rijndael," IEEE Trans.on Computers, vol. 51, Issue 12, pp. 1454-1459.

[8] J. H. Shim, D. W. Kim, Y. K. Kang, T.W. Kwon and J. R. Choi,(2002) "A Rijndael Crypto processor Using Shared On-the-fly Key Scheduler,", pp147-150, 2002.

[9] R. Karri, K. Wu, P. Mishra, and Y. Kim,(2002) "Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers," IEEE Trans. on Computer-AidedDesign of Integrated Circuits and Systems, vol. 21, No. 12, pp.1509 - 1517.

[10] Refik Sever, A. NeslinI smailoglu, Yusuf C. Tekmen, Murat Askar, BurakOkcan,(2004)"A High speed fpga Implementation of the Rijndael Algorithm" Proceedings of the EUROMICRO Systems on Digital System Design (DSD'04),IEEE, pp.358-362.

[11] A.Rudra, P. K. Dubey, C. S. Jutla, V.Kumar, J. R. Rao, and P. Rohatgi, (2001)"Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic", Proceedings CHES 2001, pp. 171–184, Paris, France,Volume 2162/2001,pp. 171–184.

[12]Nazar A. Saqib, Francisco Rodríguez-Henríquez and Arturo Díaz-Pérez,(2003)"AES Algorithm Implementation—An efficient approach for Sequential and Pipeline Architectures" Proceedings of the Fourth Mexican International Conference on Computer Science (ENC'03),IEEE, pp126 – 130.

[13] A. Hodjat, I. Verbaushede,(2004)" A 21.54 Gbits/s fully pipelined AES processor on FPGA", in: IEEE Symp. On Field-Programmable Custom Computing Machines, pp 308 – 309.

[14] DeenKotturi, Seong-Moo Yoo, and John Blizzard,(2005)" AES Crypto Chip Utilizing High-Speed Parallel Pipelined Architecture", IEEE, Vol. 5, pp 4653-4656

[15]Monica Liberatori, Fernando Otero, J. C. Bonadero, Jorge Castifieira,(2007)"AES-128 cipher. high speed, low cost fpgaimplementation", IEEE

[16] BanraplangJyrwa, Roy Paily,(2009)" An Area-Throughput Efficient FPGA implementation of Block Cipher AES algorithm",IEEE.

[17] Nalini C, Nagaraj, Dr. Anandmohan P.V*, &Poornaiah D.V, V.D.kulkarni,(2006)"An FPGA Based Performance Analysis of Pipelining and Unrolling of AES Algorithm", IEEE, pp 477 - 482.
[18] Chi-Wu Huang, Chi-Jeng Chang, Mao-Yuan Lin, Hung-Yun Tai, (2007)"Compact FPGA Implementation of 32-bits AES Algorithm Using Block RAM", IEEE, pp126.
[19] Muhammad H. Rais, and Syed M. Qasim,(2009)" FPGA Implementation of Rijndael Algorithm using Reduced Residue of Prime Numbers", IEEE, pp 1 - 4 .
[20]C. Sivakumar and A .Velmurugan,(2007)"High Speed VLSI Design CCMP AES Cipher for WLAN (IEEE 802.11i)", IEEE, pp398 - 403 .
[21] SwinderKaur Prof. RenuVig,(2007)"Efficient Implementation of AES Algorithm in FPGA Device" International Conference on Computational Intelligence and Multimedia Applications,, IEEE ,Volume2,pp 179 - 187.
[22] Mr. Atul M. Borkar, Dr. R. V. Kshirsagar, Mrs. M. V. Vyawahare,(2011)" FPGA Implementation of AES Algorithm", IEEE, Volume : 3,pp 401-405.
[23]Ga¨el Rouvroy, Franc¸ois-Xavier Standaert, Jean-Jacques Quisquater and Jean-Didier Legat,(2004)"Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael VeryWell Suited for Small Embedded Applications",ITCC'04,IEEE, Vol.2, Pp 583 - 587.
[24]Ahmed Rady,Ehab EL Sehely,A.M. EL Hennawy,(2007) "Design and Implementation of area optimized AES algorithm on reconfigurable FPGA", IEEE, pp 35 – 38.
[25]H.Li,(2006)" Efficient and flexible architecture for AES",IEEE, Vol153 , Issue:6,pp 533 - 538.
[26]Gang Zhou, HaraldMichalik, LaszloHinsenkamp,(2007) "Efficient and High-Throughput Implementations of AES-GCM on FPGAs", ICFPT 2007,IEEE,pp 185-192 .
[27]Nicholas Weaver and John Wawrzynek,(2002) " High Performance, Compact AES Implementations in Xilinx FPGAs",<{nweaver,johnw}@cs.berkeley.edu>.
[28]Sumanth Kumar Reddy S , R.Sakthivel, P Praneeth,(2011) "VLSI Implementation of AES Crypto Processor for High Throughput", IJAEST, Vol No. 6, Issue No. 1,pp 022 - 026 .
[29]Tim Good and Mohammed Benaissa , "AES on FPGA from the fastest to the smallest" Springer Berlin / Heidelberg, pp427-440.
[30]Chih-Peng Fan, Jun-Kui Hwang ,(2008) "FPGA implementations of high throughput Sequential and Fully pipelined aes algorithm", International Journal of Electrical Engineering, Vol 15,No 6,PP 447-455.
[31]Yulin Zhang, Xinggang Wang, (2010)"Pipelined Implementation of AES Encryption Based on FPGA", IEEE,pp170 - 173.
[32]Muhammad H. Rais and Syed M. Qasim,(2009) "Efficient Hardware Realization of Advanced Encryption Standard Algorithm using Virtex-5 FPGA", International Journal of Computer Science and Network Security, VOL.9 No.9 ,pp201-205.
[33]Muhammad H. Rais and Syed M. Qasim (2010) "Efficient FPGA Realization of S-Box using Reduced Residue of Prime Numbers", International Journal of Computer Science and Network Security, VOL.10 No.1,pp 754-757.
[34]Ashwini M. Deshpande, Mangesh S. Deshpande and Devendra N. Kayatanavar, (2009)"FPGA Implementation of AES Encryption and Decryption", International Conference on Control, Automation, Communication and Energy Conservation -2009.
[35]M. Rajaram, J. Vijaya,(2011) "High Speed Pipelined AES with Mixcolumn Transform", European Journal of Scientific Research
[36] Gurmail Singh, Rajesh Mehra,(2011) "FPGA Based High Speed and Area efficient Aes Encryption for Data Security", International Journal of Research and Innovation in Computer Engineering(IJCTA), Vol 1(2), pp 53-56.
[37]Tanzilur Rahman, Shengyi Pan, Qi Zhang ,(2010) "Design of a High Throughput 128-bit AES (Rijndael Block Cipher)",International Multi Conference of Engineers and Computer Scientists,IEEE,vol2.
[38]M. M. Wong, M. L. D. Wong, A. K. Nandi, and I. Hijazin,(2011)"Construction of Optimum Composite Field Architecture for Compact High-Throughput AES S-Boxes", IEEE, VoI PP, Issue 99,pp 1-5.
[39]HadiSamiee, RezaEbrahimiAtani, HamidrezaAmindavar, (2011)"A Novel Area-Throughput Optimized Architecture for the AES Algorithm", International Conference on Electronic Devices, Systems and Applications (ICEDSA),IEEE, pp 29-32.
[40]Abidalrahman Moh'd,YaserJararweh, Lo'ai Tawalbeh,(2011)"AES-512: 512-Bit Advanced Encryption Standard Algorithm Design and Evaluation", IEEE, pp 292-297.
[41]Yi Wang and Renfa Li (2011)"A Unified Architecture for Supporting Operations of AES and ECC", Fourth International Symposium on Parallel Architectures, Algorithms and Programming, IEEE, pp 185-189.
[42]ShakilAhmed, KhairulmizamSamsudin, AbdulRahmanRamli, FakhrulZamanRokhani,(2011) "Effective Implementation of AES-XTS on FPGA", IEEE, pp 184-186.
[43] V. Rijmen, "Efficient Implementation of the Rijndael S-box", Available at http://www.esat.kuleuven.ac.be/~rijmen/rijndael.

[44] C. C. Lu and S. Y. Tseng,(1991) "Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter", IEEE Transactions on Information Theory, vol. 37, no. 5, pp. 1241–1260.
[45]V. Fischer,(2000) "Realization of the Round 2 Candidates Using Altera FPGA", The Third AES Conference (AES3), New York. Available at http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html.
[46] Xinmiao Zhang and Keshab K. Parhi,(2002),"Implementation Approaches for the Advanced Encryption Standard Algorithm", IEEE, pp 24-46.
[47]Kris Gaj and Pawel Chodowiec," FPGA and ASIC Implementations of AES".
[48] Edwin NC Mui," Practical Implementation of Rijndael S-Box Using Combinational Logic".
[49] Bhupathi Kakarlapudi and Nitin Alabur," FPGA Implementations of S-box vs. T-box iterative architectures of AES".

Authors

Shylashree .N received B.E (2006) in Electronics and Communication Engineering from VTU, Karnataka, India. She received M.Tech (2008) in VLSI Design and Embedded Systems from VTU, Karnataka, India. She is currently a research scholar (part-time) in PESCE, Mandya, Karnataka, India and is also an assistant professor in ECE Dept, RNSIT, Karnataka, India.

Nagarjun Bhat is currently a student pursuing his B.E (4th semester),in ECE, RNSIT, Karnataka, India.

V. Sridhar is currently The Principal of P.E.S College of Engineering, Mandya, Karnataka, India. He was the Registrar (Evaluation), VTU, Karnataka , India. His research interests include Cryptography, VLSI and Embedded Systems, Bio-Medical Engineering. He has published more than 40 research papers in various international journals and conferences.