I. J. Computer Network and Information Security, 2013, 2, 10-20
Published Online February 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2013.02.02

FPGA Based A New Low Power and Self-Timed AES 128-bit Encryption Algorithm for Encryption Audio Signal

Bahram Rashidi
Department of Electronic and Computer Engineering, Isfahan University of Technology, IRAN
b.rashidi@ec.iut.ac.ir

Bahman Rashidi
Iran University of Science and Technology, Tehran, IRAN
b_rashidi@comp.iust.ac.ir

***Abstract* —** This paper presents, a low power 128-bit Advanced Encryption Standard (AES) algorithm based on a novel asynchronous self-timed architecture for encryption of audio signals. An asynchronous system is defined as one where the transfers of information between combinatorial blocks without a global clock signal. The self-timed architectures are asynchronous circuits which perform their function based on local synchronization signals called hand shake, independently from the other modules. This new architecture reduced spikes on current consumption and only parts with valid data are working, and also this design does not need any clock pulse. A combinational logic based Rijndael S-Box implementation for the Substitution Byte transformation in AES is proposed, its low area occupancy and high throughput therefore proposed digital design leads to reduction in power consumption. Mix-columns transformation is implemented only based on multiply-by-2 and multiply-by-3 modules with combinational logic. The proposed novel asynchronous self-timed AES algorithm is modeled and verified using FPGA and simulation results from encryption of sound signals is presented, until original characteristics are preserved anymore and have been successfully synthesized and implemented using Xilinx ISE V7.1 and Virtex IV FPGA to target device Xc4vf100. The achieved power consumption is 283 mW in clock frequency of 100 MHz.

***Index Terms* —** Low power, self-timed, AES, FPGA, Combinational Logic**.**

I. Introduction

Cryptography is the science of information and communication security. Cryptography is the science of secret codes, enabling the confidentiality of communication through an insecure channel. It protects against unauthorized parties by preventing unauthorized alteration of use. It uses a cryptographic system to transform a plaintext into a cipher text, using most of the time a key. There exists certain cipher that doesn't need a key at all. The AES is the winner of the contest, held in 1997 by the US Government, after the data encryption standard was found too weak because of its small key size and the technological advancements in processor power. Fifteen candidates were accepted in 1998 and based on public comments the pool was reduced to five finalists in 1999. In October 2000, one of these five algorithms was selected as the forthcoming standard: a slightly modified version of the Rijndael. The Rijndael, whose name is based on the names of its two Belgian inventors, Joan Daemen and Vincent Rijmen, is a block cipher, which means that it works on fixed-length group of bits, which are called blocks. It takes an input block of a certain size, usually 128, and produces a corresponding output block of the same size. The transformation requires a second input, which is the secret key. It is important to know that the secret key can be of any size [1]. In modern ages, cryptography development has been a major concern in the fields of mathematics, computer science and engineering. One of the main classes in cryptography today is the symmetric-key cryptography, where a shared key of a certain size will be used for the encryption and decryption processes [2]. In this paper, we propose a power, speed, and performance trade off analysis for FPGA based implementation of a Cryptography using novel asynchronous self-timed architecture without any clock pulse for encryption of the audio signal. All blocks are design with optimize circuits. We have focused on the effects that occur especially on FPGA based implementation. Hardware cryptographic in FPGAs has the advantage of performance, compared to any software solution. In recent years, a number of techniques have been proposed for implementation of AES algorithm on FPGA, among many FPGA based AES algorithm implementations have been, A 16-bit AES architecture for low power and high bit rate applications has been presented in [3]. The novelty is in breaking the original 32-bit boundary based AES algorithm into a scalable architecture to work with 8-bit and 16-bit data set. Thus offers a choice to the designer to use 8-bit or 16-bit algorithm for area and power efficient FPGA implementation. The novelty of the new development is still around the mix-column design. The complex matrix multiplication component and standard transformations of the 32-bit AES algorithm are transformed now to support 16-bit operations as well, simultaneously qualifying for applications requiring high data rates. The design has been further embellished by a memory based micro-programmed controller, which simplifies the control process of the algorithm and makes the FPGA platform viable for effective hardware utilization. In [4] a key scheduling unit is added. The number of clock cycles required to encrypt a single block has been reduced and the amount of hardware resources has been optimized. Dong Chen et al, presents the outer-round only pipelined architecture for a FPGA implementation of the AES-128 encryption processor. They design uses the block RAM storing the S-box values and exploits two kinds of Block RAM. Combine the operations in a single round. In [5], they propose an area optimized design for the AES, in the Cipher Block Chaining (CBC) mode. A new efficient architecture for high-speed advanced AES, using composite field arithmetic byte substitution implemented in [6], where higher efficiency is achieved by merging and location rearrangement of different operations required in the steps of encryption. An equivalent optimized sub-pipelined architecture is proposed to implement the AES, every round including encryption and decryption needs one clock cycle. The Sub-Bytes/InvSub-Bytes operation using composite field arithmetic in GF (2^4) and block RAMs respectively. In addition, an efficient key expansion which supports the output of 128 bits key per cycle and allows key changes every cycle is also presented by [7]. In [8], Speed is increased by processing multiple rounds simultaneously but at the cost of increased area. Algorithmic optimization techniques have also been used which includes exclusion of shift row stage and on the fly round key generation.

This paper is organized as follows. In section II description of the AES algorithm, Section III discusses proposed self-timed AES architecture and section IV comparison of the hardware implementation and chip utilization taken from Xilinx ISE that verifies the performance of the proposed work. Section V is the conclusion.

II. Description Of The AES Algorithm

AES is an iterated block cipher with a fixed block size of 128 and a variable key length. The different transformations operate on the intermediate results, called state. The state is a rectangular array of bytes and since the block size is 128 bits, which is 16 bytes, the rectangular array is of dimensions 4x4. The cipher key is similarly pictured as a rectangular array with four rows. The number of columns of the cipher key, denoted N_k, is equal to the key length divided by 32[1]. Description of mathematical preliminaries AES is explained in [1].

Algorithm Specification

As explained in [9], the length of the input block, the output block and the State is 128 bits. This is represented by $N_b = 4$, which reflects the number of 32-bit words in the state. The key length is represented by $N_k = 4, 6,$ or 8, which reflects the number of 32-bit words in the cipher key. The number of rounds to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by N_r, where $N_r = 10$ when $N_k = 4$. The AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1- byte substitution using a substitution table (S-box), 2- shifting rows of the state array by different offsets, 3- mixing the data within each column of the state array, and 4- adding a round key to the state.

During each round, the following operations are applied on the state:

1. SubBytes()Transformation

The SubBytes() transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). This S-box, which is invertible, is constructed by composing two transformations:

1- Take the multiplicative inverse in the finite field GF (2^8).

2- Apply the following affine transformation (over GF (2)):

$$b'_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \quad (1)$$

For $0 \le i < 8$, where b_i is the ith bit of the byte, and c_i is the ith bit of a byte c with the value {63} or {01100011}. Here and elsewhere, a prime on a variable indicates that the variable is to be updated with the value on the right.

2. ShiftRows() Transformation

In the ShiftRows() transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, r = 0, is not shifted. Specifically, the ShiftRows() transformation proceeds as follows:

$$S'_{r.c} = S_{r,(c+shift(r,N_b)) \bmod N_b} \quad for \;\; 0 < r < 4 \;\; and \;\; 0 \le c < N_b \quad (2)$$

Where the shift value shift(r, N_b) depends on the row number, r, as follows (recall that $N_b = 4$):

shift(1,4)=1;shift(2,4)=2;shift(3,4)=3.

3. MixColumns() Transformation

The MixColumns() transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF (2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial a(x), given by

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

This can be written as a matrix multiplication. Let

$$S'(x) = a(x) \otimes S(x);$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3c} \end{bmatrix} \qquad for \quad 0 \le c < N_b$$

4. AddRoundKey() Transformation

In the AddRoundKey() transformation, a Round Key is added to the State by a simple bitwise *XOR* operation. Each Round Key consists of N_b words from the key schedule. Those N_b words are each added into the columns of the State, such that

$$[b_{0,c} \quad b_{1,c} \quad b_{2,c} \quad b_{3,c}] = [S_{0,c} \quad S_{1,c} \quad S_{2,c} \quad S_{3,c}] \oplus [W_{round*N_b+c}] \quad for\, 0 \le C \prec N_b \ (4)$$

Where $[w_i]$ are the key schedule words, and round is a value in the range $0 \le$ round $< N_r$.

5. Key Expansion Algorithm

The AES algorithm takes the Cipher Key, ***K***, and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of $\boldsymbol{N_b(N_{r+1})}$ words: the algorithm requires an initial set of $\boldsymbol{N_b}$ words, and each of the $\boldsymbol{N_r}$ rounds requires $\boldsymbol{N_b}$ words of key data [10]. Fig.1 shows the generation of the first eight words of the expanded key. The function "g" consists of the following sub-functions [11]:

1. RotWord() performance a one-byte circular left shift on a word. This means that an input word [b_0, b1, b2, b3] is transformed into [b_1, b2, b3, b0].

2. SubWord() perfoms a byte substitution on each byte of its input word using the S-box.

3. The result of steps 1 and 2 is *xored* with a round constant shown in Table I.

Table I: The value r-con[j] in hexadecimal

J	1	2	3	4	5	6	7	8	9	10
Rcon[j]	01	02	04	08	10	20	40	80	1b	36

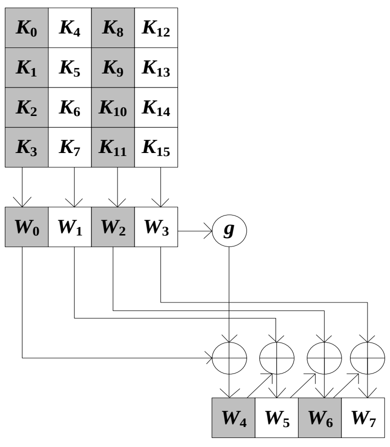

Fig.1: AES Key Expansion Algorithm.

III. Proposed Self-Timed AES Architecture

FPGAs are standard integrated circuits that can be programmed by a user to perform a variety of complex logic functions. FPGAs have the capability of being reconfigurable within a system, which can be a big advantage in applications that need multiple trial versions within development, offering reasonably fast time-to-market. Two major computational areas have found a natural home in FPGAs: cryptography and bioinformatics. So For example, an FPGA can be used in an encryption scheme to perform the encryption using whatever encryption algorithm is programmed into it; and the same chip can be used for multiple rounds of encryption that combine different encryption algorithms [12]. In this paper, in order to reduce power consumption, the self-timed technique has been used. Furthermore, mix-column transformation is optimized and its implementation is based only on multiply-by-2 and multiply-by-3 modules. Self-timed circuits perform their functions by local synchronization signals call hand shake so total system is self controlled and does not need any separate control unit or global clock pulse. This new architecture reduces spikes on consumed current and only those parts with valid data are working and clock-less. Thus lead to reduction in power consumption. Aim is achieve the low power and high performance hardware implementation of AES based on self-timed technique.

A) Self-Timed Circuits

As described in [13], most current designs depend on a global synchronizing signal or clock to make all of the blocks in the circuit communicate correctly, and smaller device sizes with increasing chip areas cause the proportional capacitive loading on the global clock line(s) to increase. Clock skews from the timing distribution network require that either the logic circuits meet certain latency requirements, or that the non overlap times of a multiphase clocked system be increased, negatively impacting the time available for computation. Self-timed circuits with an appropriate handshake protocol can be used to eliminate the requirement for any global clock in a system. In place of a global clock, the chip only needs a reset signal and external handshaking signals to synchronize its operation. Internally, stages communicate at their own speed, which is an advantage since the speed of operation no longer is constrained by the slowest block in the system as in a clocked system. A self-timed system is an asynchronous system made up of self-timed blocks. Instead of synthesizing synchronous circuits, an alternative solution, especially in the case of large circuits, is self-timing. As a generic example in [14], consider the pipelined circuit, to each block, for example number i, are associated a maximum delay t_{max}(i) and an average one t_{av} (i). The latency and throughput of the circuit are equal to n.T_{Clk} And 1/T_{clk}, respectively where $T_{clk} >$ max { t_{max}(0), t_{max}(1), … , t_{max}(n-1)}, that is,

Latency > n.max { t_{max}(0), t_{max}(1), … , t_{max}(n-1)},
Throughout<i/max{t_{max}(0),t_{max}(1),…,t_{max}(n-1)}. (5)

A self-timed version of the same circuit is shown in Fig.2. If the probability distribution of the internal data were uniform, in equalities (5) would be substituted by the following ones:

average latency > $t_{av}(0)+t_{av}(1)+\ldots+t_{av}(n-1)$,
average throughput<1/max{ $t_{av}(0)+t_{av}(1)+\ldots+t_{av}(n-1)$} (6)

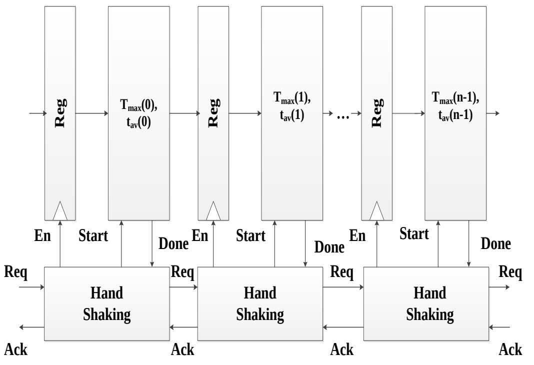

Fig.2: Self-timed pipelined circuit.

Main and important block in self-timed circuits is interconnecting circuit or hand shake block. Thus we for digital design and implementation of the hand shake blocks using combinational logic. Fig.3 shows proposed hand shake block. Hand shake block should be able to detect the valid data of pervious block and if request signal is valid, hand shake block transmits data to next block. Finally to inform pervious block that transmits is successfully acknowledge signal is made valid and pervious block its request signal make invalid thus hand shake block should include four part one for detect valid data of pervious block, two part for receive request signal the previous block three part for asserted enable signal to register between blocks and four part for active acknowledge signal to shown data received and transmits done successfully. As seen in Fig.3 *OR & AND* logic gates used until detect valid output data of any block, and also three multiplexers 2 to 1 is for create en signal, request signal and acknowledge signal to select ‘0’ & ‘1’.

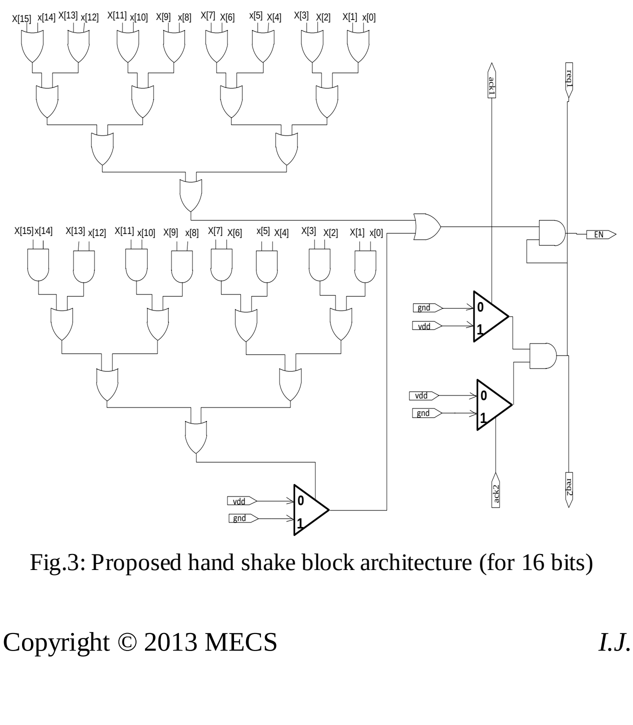
X[15] X[14] X[13] X[12] X[11] X[10] X[9] X[8] X[7] X[6] X[5] X[4] X[3] X[2] X[1] X[0]
ack1
req1
EN
gnd
vdd
0
1
ack2
req2

Fig.3: Proposed hand shake block architecture (for 16 bits)

Transformation of data between blocks is based on hand shake signals (i.e. request and acknowledge). Fig.4 shows proposed self-timed AES algorithm. In Fig.4 proposed architecture is divided to three parts. First part includes four important transformation s-box, shift rows, mix-column, and add-round-key. We insert register between them for pipelining and self-timed. Second part includes hand shake blocks that receive data of pervious block and while these are valid hand shake block transformation data to next block. In this mode system doesn’t need any clock signal and communication between stages is based on a local synchronization called hand shake, and third part in proposed digital design is key-expansion. This part, too, acts base on enable signals of hand shake block. It’s divided into several sub-stages to make it possible to exert registers between them. It is suitable for pipelining. This part, also, has been done for key-expansion. Hand shake signals can synchronize these blocks to work simultaneously together.

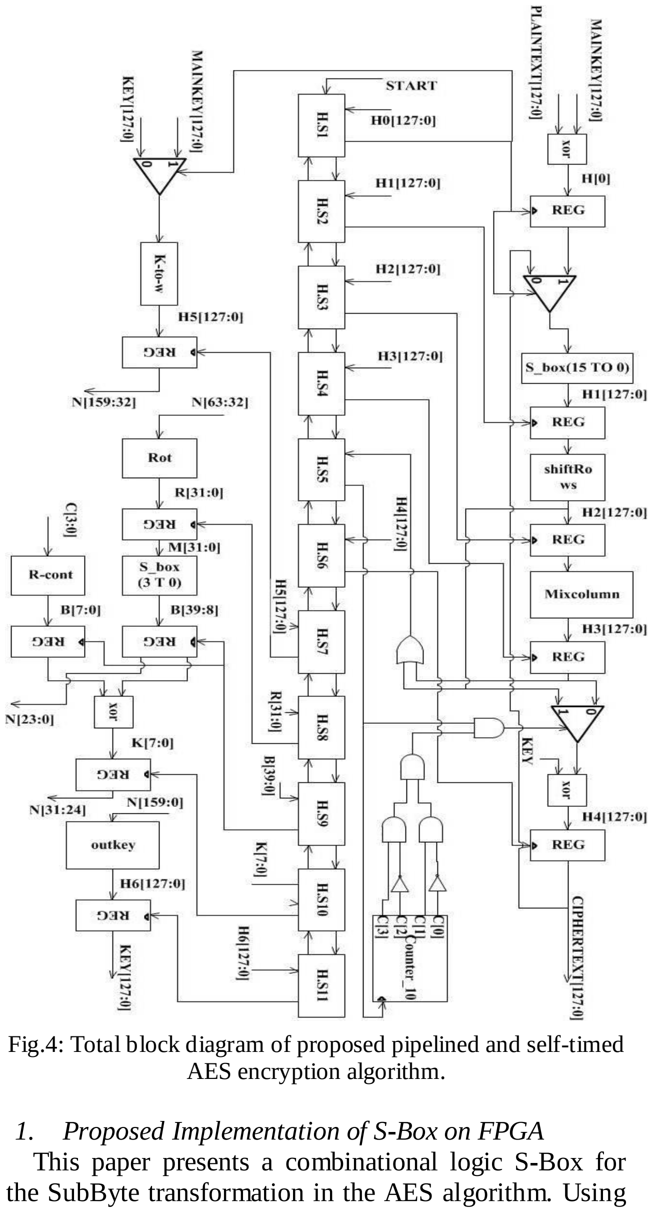

Fig.4: Total block diagram of proposed pipelined and self-timed AES encryption algorithm.

1. *Proposed Implementation of S-Box on FPGA*

This paper presents a combinational logic S-Box for the SubByte transformation in the AES algorithm. Using combinational logic for implement S-Box has small area occupancy and high throughput, and as compared to the typical ROM based look up table implementation which access time is fixed and unbreakable. The SubByte transformation is computed by taking the multiplicative inverse in GF (2^8) followed by an affine transformation [15].

SubByte:

The Affine Transformation can be represented in matrix form and it is shown below:

$$AT(a)=\begin{bmatrix}1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\\1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\end{bmatrix}\times\begin{bmatrix}a7\\a6\\a5\\a4\\a3\\a2\\a1\\a0\end{bmatrix}\oplus\begin{bmatrix}0\\1\\1\\0\\0\\0\\1\\1\end{bmatrix}$$

The AT is the Affine Transformation From here, it is observed that the SubByte transformation involve a multiplicative inversion operation. This section illustrates the steps involved in constructing the multiplicative inverse module for the S-Box using composite field arithmetic. The multiplicative inverse computation will first be covered and the affine transformation will then follow to complete the methodology involved for constructing the S-Box for the SubByte operation. From [16], it is stated that any arbitrary polynomial can be represented as *bx + c*, given an irreducible polynomial of x^2+Ax+B. Thus, element in GF (2^8) may be represented as *bx+c* where b is the most significant nibble while c is the least significant nibble. From here, the multiplicative inverse can be computed using the equation below [16].

$$(\mathrm{bx+c})^{-1}=b(b^2B+bcA+c^2)^{-1}x+(c+bA)(b^2cA+c^2)^{-1}$$

From [15], the irreducible polynomial that was selected was $x^2+x+\lambda$. Since A=1 and B=λ, then the equation could be simplified to the form as shown below;

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

The above equation indicates that there are multiply, addition, squaring and multiplication inversion in GF (2^4) operations in GF. From this simplified equation, the multiplicative inverse circuit GF (2^8) can be produced as shown in Fig.5.

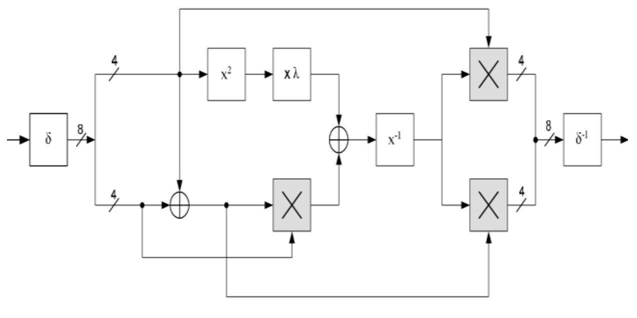

Fig.5: Multiplicative inversion module for the S-Box.

The legends for the blocks within the multiplicative inversion module from above are illustrated in Table II:

Table II: Legends for the building blocks within the multiplicative inversion module.

δ	Isomorphic mapping to composite fields
x^2	Squarer in GF(2^4)
x^{-1}	Multiplication inversion in GF(2^4)
δ^{-1}	Inverse isomorphic mapping to GF(2^8)
$x\lambda$	Multiplication with constant, in GF(2^4)
$\oplus$	Addition operation in GF(2^4)
$\times$	Multiplication operation in GF(2^4)

2. *Isomorphic Mapping and Inverse Isomorphic Mapping*

The multiplicative inverse computation will be done by decomposing the more complex GF(2^8) to lower order fields of GF(2^1), GF(2^2) and GF($(2^2)^2$). In order to accomplish the above, the following irreducible polynomials are used [15].

$$\begin{aligned}&\mathrm{GF}(2^2)\rightarrow\mathrm{GF}(2) && :x^2+x+1\\&GF((2^2)^2)\rightarrow GF(2^2) && :x^2+x+\varphi \quad (7)\\&GF(((2^2)^2)^2)\rightarrow GF((2^2)^2) && :x^2+x+\lambda\end{aligned}$$

Where φ= $\{10\}_2$ and λ= $\{1100\}_2$.

Computation of the multiplicative inverse in composite fields cannot be directly applied to an element which is based on GF (2^8). That element has to be mapped to its composite field representation via an isomorphic function, δ. Likewise, after performing the multiplicative inversion, the result will also have to be mapped back from its composite field representation to its equivalent in GF(2^8) via the inverse isomorphic function, δ^{-1}. Both δ and δ^{-1} can be represented as an 8*8 matrix. Let q be the element in GF (2^8), then the isomorphic mappings and its inverse can be written as δ*q and δ^{-1}*q, which is shown in below [15]. Proposed implementation of the affine transformation is based on XOR, NOT gates.

The matrix multiplication can be translated to logical *XOR* operation. The logical form of the matrices above is shown below.

$$\delta\times q=\begin{bmatrix}q_7\oplus q_5\\q_7\oplus q_6\oplus q_4\oplus q_3\oplus q_2\oplus q_1\\q_7\oplus q_5\oplus q_3\oplus q_2\\q_7\oplus q_5\oplus q_3\oplus q_2\oplus q_1\\q_7\oplus q_6\oplus q_2\oplus q_1\\q_7\oplus q_4\oplus q_3\oplus q_2\oplus q_1\\q_6\oplus q_4\oplus q_1\\q_6\oplus q_1\oplus q_0\end{bmatrix}$$

As seen in above matrix this block is implementation based on *XOR* gates. We for implementation of this block use minimum number of *XOR* gates, until proposed design optimized. Also other blocks in S-box are designed with combinational logic implemented with minimum number of logic gates.

$$\delta^{-1} \times q = \begin{bmatrix} q_7 \oplus q_6 \oplus q_5 \oplus q_1 \\ q_6 \oplus q_2 \\ q_6 \oplus q_5 \oplus q_1 \\ q_6 \oplus q_5 \oplus q_4 \oplus q_2 \oplus q_1 \\ q_5 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_7 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_6 \oplus q_4 \\ q_6 \oplus q_5 \oplus q_4 \oplus q_2 \oplus q_0 \end{bmatrix}$$

Also proposed implementation of δ^{-1}*q is based on *XOR* gate. From [16] and [17], any arbitrary polynomial can be represented by *bx+c* where *b* is upper half term and c is the lower half term. Therefore, from here, a binary number in GF q can be spilt to q_Hx+q_L. For instance, if q= $\{1011\}_2$, it can be represented as $\{10\}_2x+\{11\}_2$, where q_H is $\{10\}_2$ and $q_L = \{11\}_2$. Using this idea, the logical equations for the addition, squaring, multiplication and inversion can be derived.

3. Squaring in GF(24)

Let k =q^2, where k and q is an element in GF(2^4), represented by the binary number of $\{k_3k_2k_1k_0\}_2$ and $\{q_3q_2q_1q_0\}_2$ respectively.

$$\mathrm{k} = \left(\underbrace{k_3k_2}_{k_H}\underbrace{k_1k_0}_{k_L} \right) = k_H x + k_L = \left(\underbrace{q_3q_2}_{q_H}\underbrace{q_1q_0}_{q_L} \right)^2 = (q_H x + q_L)^2$$

$$\mathrm{k_L} = q_H{}^2 x^2 + q_H q_L x + q_H q_L x + q_L{}^2 = q_H{}^2 x^2 + q_L{}^2$$

The x^2 term can be modulo reduced using the irreducible polynomial from (7), $x^2+x+\varphi$. By setting $x^2=x+\varphi$ and replacing it into x^2. Doing so yields the new expressions below.

$$\mathrm{k} = q_H{}^2(x+\varphi) + q_L{}^2$$
$$\mathrm{k} = \underbrace{q_H{}^2}_{k_H} x + \underbrace{\left(q_H{}^2\varphi + q_L{}^2\right)}_{K_L} \in GF(2^2)$$

The expression above is now decomposed to GF(2^2). Decomposing k_H and k_L further to GF(2) would yield the formula to compute squaring operation in GF(2^4).

$$\mathrm{k_H} = q_H{}^2 = (q_3q_2)^2 = (q_3x+q_2)^2$$
$$\mathrm{k_H} = q_3{}^2x^2 + q_3q_2x + q_3q_2x + q_2{}^2 = q_3x^2 + q_2$$

Using the irreducible polynomial from (7) x^2 *+x+1*, and setting it to x^2*=x+1*, x^2 is substituted and the new expression is obtained.

$$\mathrm{k}_3x + k_2 = q_3x + (q_2+q_3) \in GF(2) \tag{8}$$
$$\mathrm{k_H} = q_3(x+1) + q_2$$

The k_L term is also decomposed in the similar manner as shown below.

$$\mathrm{k_L} = q_H{}^2\varphi + b_L{}^2 = (q_3q_2)^2\{10\}_2 + (q_1q_0)^2$$
$$\mathrm{k_L} = (q_3x+q_2)^2(\{1\}_2x+0) + (q_1x+q_0)^2$$
$$\mathrm{k_L} = (q_3{}^2x^2 + q_2q_3x + q_2q_3x + q_2{}^2)(x) +$$
$$(q_1{}^2x^2 + q_0q_1x + q_0q_1x + q_0{}^2)^2\{10\}_2 + (q_1q_0)^2$$
$$\mathrm{k_L} = q_Hx^3 + q_2x + q_1x^2 + q_0$$

As was done earlier, the x^2 term can be substituted since x^2*=x+1*. For the case of x^3, it can be obtained by multiplying x^2 by *x*. That is, x^3*=x(x) +x=*x^2*+x*. Substituting for x^2, x^3*=x+1+x*. The two x terms cancel out each other, leaving only x^3*=1*. Performing this substitution to the above expression yields the following.

$$\mathrm{k_L} = q_3(1) + q_2x + q_1(x+1) + q_0$$
$$\mathrm{k}_1x + k_0 = (q_2+q_1)x + (q_3+q_1+q_0) \in GF(2) \tag{9}$$

From equations (8) and (9), the formula for computing the squaring operation in GF (2^4) is acquired as shown below.

$$\begin{aligned} \mathrm{k}_3 &= q_3 \\ \mathrm{k}_2 &= q_3 \oplus q_2 \\ \mathrm{k}_1 &= q_2 \oplus q_1 \\ \mathrm{k}_0 &= q_3 \oplus q_1 \oplus q_0 \end{aligned} \tag{10}$$

4. Multiplication with constant, λ

Let k = qλ, where k= $\{k_3k_2k_1k_0\}_2$, q= $\{q_3q_2q_1q_0\}_2$ and λ= $\{1100\}_2$ are elements of GF(2^4).

$$\mathrm{k} = \left(\underbrace{k_3k_2}_{k_H}\underbrace{k_1k_0}_{k_L} \right) = k_H x + k_L = \left(\underbrace{q_3q_2}_{q_H}\underbrace{q_1q_0}_{q_L} \right)\left(\underbrace{11}_{\lambda_H}\underbrace{00}_{\lambda_L} \right)$$
$$\mathrm{k} = (q_Hx + q_L)(\lambda_Hx + \lambda_L)$$

Modulo reduction can be performed by substituting $x^2=x+\varphi$ using the irreducible polynomial in (7) to yield the expression below.

$$\mathrm{k} = q_H\lambda_H(x+\varphi) + q_L\lambda_Hx$$
$$\mathrm{k} = \underbrace{(q_H\lambda_H + q_L\lambda_H)}_{k_H}x + \underbrace{(q_L\lambda_H\varphi)}_{k_L} \in GF(2^2)$$

The k_H and k_L terms can be further broken down to GF (2).

$$\mathrm{k_H} = q_H\lambda_H + q_L\lambda_H$$
$$\mathrm{k} = (q_3q_2)(11_2) + (q_1q_0)(11_2)$$
$$\mathrm{k_H} = (q_3x+q_2)(x+1) + (q_1x+q_0)(x+1)$$
$$\mathrm{k_H} = q_3x^2 + (q_3+q_2)x + q_2 + q_1x^2 + (q_1+q_0)x + q_0$$

Substituting x^2=x+1, would then yield the following.

$$k_H = q_3(x+1)+(q_3+q_2)x+q_2+q_1(x+1)+(q_1+q_0)x+q_0$$
$$k_H = (q_3+q_3+q_2+q_1+q_1+q_0)x+(q_3+q_2+q_1+q_0)$$
$$k_3x+k_2 = (q_2+q_0)x+(q_3+q_2+q_1+q_0) \in GF(2) \qquad (11)$$

The same procedure is taken to decompose k_L to GF (2).

$$k_L = q_H\lambda_H\varphi$$
$$k_L = (q_3q_2)(11_2)(10_2)$$
$$k_L = q_3x^3+q_2x^2+q_3x^2+q_2x$$
$$k_L = (q_3x+q_2)(x+1)(x)$$

Again, the x^2 term can be substituted since $x^2=x+1$. Likewise, x^3 is also substituted with $x^3=1$,

$$k_L = q_3(1)+q_2(x+1)+q_3(x+1)+q_2x$$
$$k_L = (q_3+q_2+q_2)x+(q_3+q_3+q_2) \qquad (12)$$
$$k_1x+k_0 = (q_3)x+(q_2) \in GF(2)$$

From equations (11) and (12) combined, the formula for computing multiplication with constant λ is shown below:

$$k_3 = q_2 \oplus q_0$$
$$k_2 = q_3 \oplus q_2 \oplus q_1 \oplus q_0$$
$$k_1 = q_3$$
$$k_0 = q_2 \qquad (13)$$

5. GF(24) Multiplication

Let k = qw, where k= {k3 k2 k1 k0}2, q = {q3 q2 q1 q0}2 and w = $\{w_3w_2w_1w_0\}_2$ are elements of GF(2^4).

$$k = \left(\underbrace{k_3k_2}_{k_H}\underbrace{k_1k_0}_{k_L}\right) = k_Hx+k_L = \left(\underbrace{q_3q_2}_{q_H}\underbrace{q_1q_0}_{q_L}\right)\left(\underbrace{w_3w_2}_{w_H}\underbrace{w_1w_0}_{w_l}\right) =$$
$$(q_Hx+q_L)(w_Hx+w_L)$$
$$k = (q_Hw_H)x^2+(q_Hw_L+q_Lw_H)x+q_Lw_L$$

Substituting the x^2 term with x^2 = x + φ yields the following.

$$k = (q_Hw_H)(x+\varphi)+(q_Hw_L+q_Lw_H)x+q_Lw_L$$
$$k = k_Hx+k_L = (q_Hw_H+q_Hw_L+q_Lw_H)x$$
$$+q_Hw_H\varphi+q_Lw_L \in GF(2^2) \quad (14)$$

Equation (14) is in the form GF (2^2). It can be observed that there exist addition and multiplication operations in GF (2^2). Addition in GF (2^2) is but bitwise *XOR* operation. Multiplication in GF (2^2), on the other hand, requires decomposition to GF (2) to be implemented in hardware. Also, it the expression would be too complex if equation (14) were to be broken down to GF (2). Thus, the formula for multiplication in GF (2^2) and constant φ will be derived instead.

6. GF(22) Multiplication

Let k=qw, where k = $\{k_1\ k_0\}2$, q=$\{q_1q_0\}_2$ and w =$\{w_1w_0\}_2$ are elements of GF(2^2).

$$k = (k_1k_0) = k_1x+k_0 = (q_1q_0)(w_1w_0) = (q_1x+q_0)(w_1x+w_0)$$
$$k = q_1w_1x^2+q_0w_1x+q_1w_0x+q_0w_0$$

The x^2 term can be substituted with $x^2=x+1$ to yield the new expression below:

$$k = q_1w_1(x+1)+q_0w_1x+q_1w_0x+q_0w_0$$

The equation above can now be implemented in hardware as multiplication in GF(2) involves only the use of *AND* gates. That we use from *AND* gate for its implementation.

$$k_1 = q_1w_1 \oplus q_0w_1 \oplus q_1w_0$$
$$k_1x+k_0 = (q_1w_1+q_0w_1+q_1w_0)x$$
$$+(q_1w_1+q_0w_0) \in GF(2) \qquad (15)$$

The formula for computing multiplication in GF(2) is as follows:

$$k_0 = q_1w_1 \oplus q_0w_0 \qquad (16)$$

Proposed hardware implementation of this block is based on *XOR*, *AND* gate.

The hardware implementation above differs from the (16) for the computation of k_1. It can be proven that the implementation above for computing k_1 would result to the expression in (16), as shown below:

$$k_1 = (q_1 \oplus q_0)(w_1 \oplus w_0) \oplus (q_0w_0)$$
$$k_1 = (q_1w_1) \oplus (q_0w_1) \oplus (q_1w_0) \oplus (q_0w_0) \oplus (q_0w_0)$$
$$k_1 = (q_1w_1) \oplus (q_0w_1) \oplus (q_1w_0)$$

7. Multiplication with constant φ

Let k=qφ, where k =$\{k_1k_0\}_2$, q =$\{q_1q_0\}_2$ and φ =$\{10\}_2$ are elements of GF(2^2).

$$k = k_1x+k_0 = (q_1q_0)(10_2) = (q_1x+q_0)(x)$$
$$k_1 = q_1x^2+q_0x$$

Substitute the x^2 term with $x^2=x+1$, yield the expression below:

$$k = q_1(x+1)+q_0x$$
$$k = (q_1+q_0)x+(q_1) \in GF(2) \qquad (17)$$

From (17), the formula for computing multiplication with φ can be derived and is shown below:

$$k_1 = q_1 \oplus q_0$$
$$k_0 = q_1$$

8. Multiplicative Inversion in GF(24)

In [18] has derived a formula to compute the multiplicative inverse of q. The inverses of the individual bits can be computed from the equation below [18].

$$q_3^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_0 \oplus q_2$$
$$q_2^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_0 \oplus q_2 \oplus q_1q_2$$
$$q_1^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_1q_0 \oplus q_2 \oplus q_2q_0 \oplus q_1$$
$$q_0^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_1 \oplus q_3q_1q_0 \oplus q_3q_0 \oplus q_2 \oplus q_2q_1 \oplus q_2q_1q_0 \oplus q_1 \oplus q_0$$

Proposed implementation of these equations are based on *XOR*, *AND* gates.

9. *Proposed Implementation of Shift Rows on FPGA*

In proposed implementation of Shift Rows, 16*8-bit registers have been used, after shift operation each output byte was placed in new position according to AES algorithm. 16*8-bit registers be used as one of the 128-bit registers in self-timed architecture.

10. *Proposed Implementation of Mix-Column on FPGA*

Mix-column transformation is one important and complex block of AES algorithm, because in this transformation, multiplication operation is significant operation, and multiplication by one number (multiplication by X) is very complex also power consumption and utilized hardware for implementation on FPGA is high. Thus we should design and implement it with minimum power consumption and reduce utilized hardware. Thus we implement multiplication by 2 and 3 with combinational logic until achieve minimum hardware and simple architecture. Also we implement Mix-column transformation use of the parallel processing technique. Thus 128-bit input data enter then are divided into sixteen 32-bit groups as described by the Mix-column transformation algorithm, this groups are fed into sixteen stages and any multiplication is independent of other multiplication operation. Proposed implementation of architecture mix-column block based on multiplication-by-3 and multiplication-by-2 and *XOR* gates are shown in Fig.6.

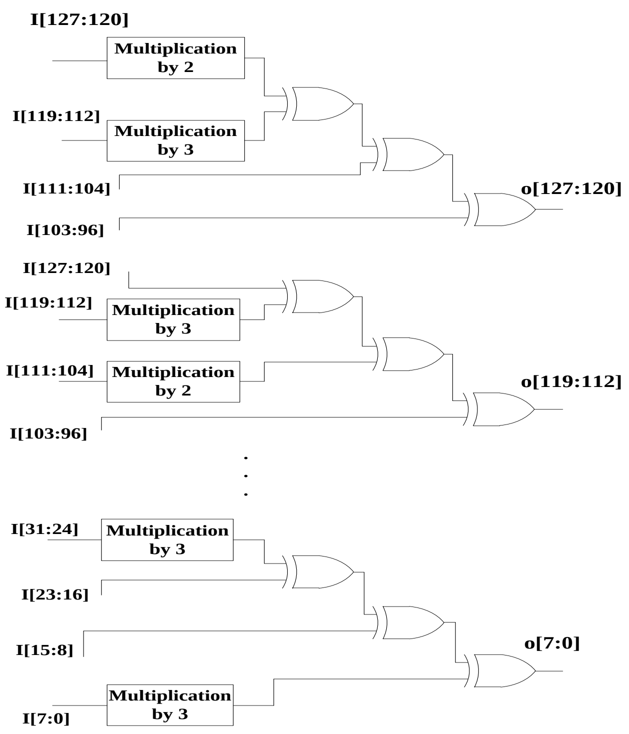

Fig.6: Proposed implementation of mix-column based on multiplication-by-3 and multiplication-by-2.

and in three part of proposed method (Fig.4) that is key-expansion block we proposed two new block in this implementation include:

1. *K-to-w block*

The *k-to-w* is for implementation of below equation according to key-expansion block:

$$W_0 = k_0k_1k_2k_3$$
$$W_1 = k_4k_5k_6k_7$$
$$W_2 = k_8k_9k_{10}k_{11}$$
$$W_3 = k_{12}k_{13}k_{14}k_{15}$$

2. *outkey block*

The *outkey* block is implemented for final stage of the key-expansion that in below proposed verilog code of this block is shown in below:

```
module outkey_1(b,a);
input[159:0]b;
output[127:0]a;
wire[127:0]a;
assign a[127:96]=(b[159:128]^b[31:0]);
assign a[95:64]=(b[127:96]^a[127:96]);
assign a[63:32]=(b[95:64]^a[95:64]);
assign a[31:0]=(b[63:32]^a[63:32]);
endmodule
```

In this verilog code ***b[31:0]*** is output of the function "g" in key-expansion.

3. R-con[j]

The R-con[j] block is for implementation Table I. in below proposed code for this block is shown:

```
Begin
case(sel)
 4'b0001:b5[7:0]=8'b00000001;
 4'b0010:b5[7:0]=8'b00000010;
 4'b0011:b5[7:0]=8'b00000100;
 4'b0100:b5[7:0]=8'b00001000;
 4'b0101:b5[7:0]=8'b00010000;
 4'b0110:b5[7:0]=8'b00100000;
 4'b0111:b5[7:0]=8'b01000000;
 4'b1000:b5[7:0]=8'b10000000;
 4'b1001:b5[7:0]=8'b00011011;
 4'b1010:b5[7:0]=8'b00110110;
endcase
end
```

This block is based on LUT according to Table I.

4. Rot-word block

As already explained in section key-expansion, the Rot-word block is performance a one-byte circular left shift on a word. As for count number of round we use of a counter, this counter has been used to detect tenth round and also for address lines of R-con block because this block is implementation based on LUT. In every tenth round, its output signal multiplexers select line, which is connected to a causes data to bypass the mix column block i.e. until data in this stage after shift rows block enter to add-roundkey block instead mix-column block according to AES algorithm.

5. Encryption Audio Signal on Proposed Implementation

The development of encryption systems for voice is in turn a lot more difficult and mostly not satisfying achievable with analogue techniques. Due to that problem, much pioneering work for many digital capabilities was performed while inventing a system to provide secure voice communications. As outlaid before, encryption often refers to digital technologies, in fact, if we hear about data security and encryption in context with modern technologies, we barely talk about something else but digital encryption. “Digital encryption” can be seen as a much stronger method of protecting speech communications than “analogue scrambling”. The big advantage of digital encryption is that it does not matter what kind of signal is encrypted. That makes digital encryption quite powerful because we can create one standard to handle e.g. text, audio, video and every other kind of data. Certainly, digital encryption takes always the same start point, the analogue to digital conversation, however in voice encryption things are a bit different since there are two different ways to go after this [19].This proposed architecture is to provide a good and efficient method for hiding the data from hackers and sending to a destination in a safe manner. This proposed system will not change the size of the data even after encoding and also suitable for any type of audio file format. The quality of sound depends on the size of the audio which the user selects and length of the message. The quality of the sound in the encrypted audio file can be increased. There are number of ways that this project could be extended. Its performance can be upgraded to higher levels in practical conditions. In this implementation, for application of proposed design we do encryption of the sound. Hex codes of samples audio signal are obtained from MATLAB then these Hex codes are given to the proposed designed AES encrypting, and encrypted data of the sound obtained. Fig.7 shows original audio signal and encrypted audio signal, therefore, it does not provide any indication to employ any statistical attack on the sound under consideration.

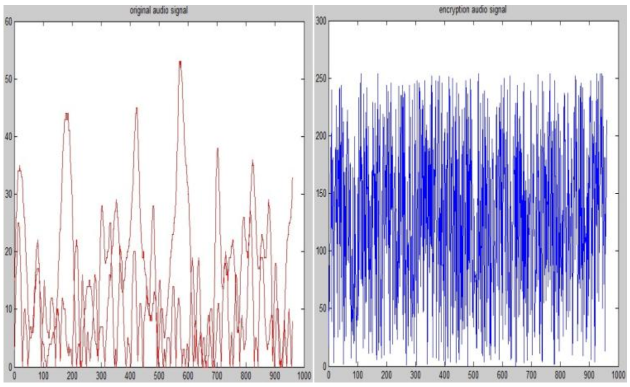

Fig.7: Original audio signal and encrypted audio signal.

IV. COMPARISON

We designed a low power AES algorithm based on novel asynchronous self-timed architecture for encryption of audio signals until original characteristics are preserved anymore. In this paper, proposed method has been writed with verilog hardware description language. The proposed Low Power Self-Timed AES 128-Bit Encryption Algorithm was synthesized and implemented using Xilinx ISE V7.1 and Virtex IV FPGA to target device xc4vfx100 also power is analized using Xilinx XPower analyzer. And for simulation we use of MATLAB7.2. Table III shows the comparison between power consumption, numbers of LUTs, numbers of Slices and FFs and the type of device that has been used in different articles and proposed method, Table IV show the power consumption of proposed method and other works.

Table III: Summary of hardware characteristic obtained of proposed method and other works.

AES algorithm	Device	Slices	FFs	LUTs
[20] open scheme	Virtex PRO	6910	---	---
[20] DOR+K Scheme	Virtex PRO	2439	---	---
[3]	Xc2v1000	228	---	---
[21]	Xc3s200ft256	1643	975	3055
[22]	Xcvp70	2389	2827	4401
[6] p=9	Xc2v6000-6	10662	---	---
[6] p=5	Xc2v6000-6	7884	---	---

[23]	xc4vlx85	8901	---	---
[8] Optimized Area	---	1468	---	---
[8] Optimized Speed	---	18855	---	---
Proposed method	Xc4vf100	2856	949	4743

Table IV: Summary of data obtained from power consumption of our proposed method and other works.

Frequency (MHZ)	Proposed method	[20] open scheme	[20] DOR+K Scheme	[24]
5	43 mw	235.45mw	138.21mw	---
25	93mw	---	---	885 mw
50	156mw	---	---	---
75	220mw	---	---	---
100	283mw	---	---	---

V. CONCLUSION

In this paper, a new low power and self-timed architecture for AES algorithm with encryption of audio signals is proposed. Different approaches are using in proposed method, until reduced power consumption include; using a novel asynchronous self-timed technique based on a proposed new architecture. This new architecture reduced spikes on current consumption and only parts with valid data are working and this design is clock-less, also a combinational logic based S-Box for the SubByte transformation is discussed and its internal operations are explained. As compared to the typical ROM based lookup table, thus power consumption reduction. We applied proposed new self-timed AES algorithm for encryption of audio signals until original characteristics are preserved anymore.

REFERENCES

[1] Jagadev, Vivek Senapati, "Advanced Encryption Standard (AES) Implementation ", Thesis for the degree of Bachelor of Technology in National Institute of Technology, Rourkela May, 2009.

[2] Issam Mahdi Hammad, "Efficient Hardware Implementations For The Advanced Encryption Standard (AES) Algorithm" Master Thesis, Dalhousie University Halifax, Nova Scotia 2010.

[3] Habibullah Jamal et al, "Low Power Area Efficient High Data Rate 16-bit AES Crypto Processor", The 18th International Conference on Microelectronics (ICM) 2006, pp. 186-189.

[4] Yulin Zhang, Xinggang Wang, "Pipelined Implementation of AES Encryption Based on FPGA", 2010 IEEE, pp. 170-173.

[5] Dong Chen et al, "Efficient Architecture and Implementations of AES", 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), V6, pp. 295-298.

[6] M.R.M. Rizk, M.Morsy, "Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA", 2007 IEEE, pp. 207-217.

[7] Ghada Farouk Naiem et al, "An Efficient Implementation of CBC Mode Rijndeal AES on anFPGA",25th National Radio Science Conference (NRSC),March 1820, 2008, Faculty of Engineering, Tanta Univ, Egypt, pp. 1-8.

[8] Swinder Kaur, Prof. Renu Vig, "Efficient Implementation of AES Algorithm in FPGA Device", International Conference on Computational Intelligence and Multimedia Applications 2007, pp. 179-187.

[9] Amir Ahmed Khan, "Implementation Of High Speed AES Algorithm On FPGA", B.E. (EL) Project Report Batch 2003-04, University of Engineering & Technology Karachi.

[10] Federal Information Processing Standards Publication (FIPS PUBS) 197, "Announcing The Advanced Encryption Standard (AES)", November 26, 2001.

[11] William stallings, "Cryptography and Network Security" Pearson Printice Hall, Printed in the United State of America, Fourth Edition 2006 Pearson Education.

[12] "FPGAs: Field-Programmable Gate Arrays for Configurable Computing" written August, 2001 by D. Gaasterland for CMSC 411, Computer Systems Architecture, University of Maryland.

[13] Gordonm.Jacobs, Robertw. Brodersen, "A fully Asynchronous digital signal processor using self-timed circuits", IEEE journal of solid-state circuts, vol, 25, no.6, 1990.

[14] Jean-Pierre Deschamps et al, "Synthesis of Arithmetic Circuits", Published by John Wiley & Sons, Inc. Published simultaneously in Canada, 2006.

[15] Akashi Satoh et al, "A Compact Rijndael Hardware Architecture with S-Box Optimization", Springer-Verlag Berlin Heidelberg, 2001.

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

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

[18] Xinmiao Zhang , Keshab K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm", IEEE Transactions on Very Large Scale Integration(VLSI) Systems, Vol. 12, No. 9, Septemper 2004.

[19] Markus Albert Brandau," Implementation of a real-time voice encryption system", Master Thesis, Universitat Politècnica de Catalunya EUETIT, 2008

[20] Jason Van Dyken, José G. Delgado-Frias, "FPGA schemes for minimizing the power-throughput trade-off in executing the Advanced Encryption Standard algorithm" Journal of Systems Architecture 56 (2010) 116–123

[21] Qitao Zhang, "On a Hardware Implementing Method of the Optimized AES Encryption Algorithm", 2010

Second International Conference on MultiMedia and Information Technology, pp. 82-84.

[22] Monica Liberatori et al, "AES-128 Cipher. High Speed, Low Cost FPGA Implementation", 2007 IEEE, pp. 195-198.

[23] Issam Hammad et al, "High-Speed AES Encryptor with Efficient Merging Techniques", IEEE Embedded Systems Letters, Vol. 2, NO. 3, SEPTEMBER 2010, pp. 67-71.

[24] Roohi banu, Tanya vladimirova," fault-tolerant encryption for space applications", IEEE Transactions on Aerospace and Electronic Systems Vol. 45, NO. 1 JANUARY 2009.

Bahram Rashidi, was born in 1986 in Boroujerd-Lorestan, Iran. He received his B.SC. Degree in Electrical Engineering from the Lorestan University, Iran, in 2009 and he received his M.SC. in the Tabriz university, Iran also he is now Ph.D. student in Isfahan University of technology, respectively. His research interests include digital signal processing, DSP processors, computer vision, modeling with hardware description languages VHDL and VERILOG, He now continues on his interest in digital circuits with research in embedded microprocessor systems and VLSI digital chip design.

Bahman Rashidi, received his B.SC. Degree in Computer Engineering from the Science & Technology Sepahan Isfahan University, Iran, in 2009 and he is now M.SC. in the Iran University of Science and Technology, ,Tehran, IRAN, respectively. He has accepted and published 2 refereed conference papers. His research interests include Computer Architecture, Computer vision, Distributed System, Cloud Computing.