I. J. Computer Network and Information Security, 2013, 1, 41-48
Published Online January 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2013.01.05

Implementation of An Optimized and Pipelined Combinational Logic Rijndael S-Box on FPGA

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 — In this paper, presents an optimized combinational logic based Rijndael S-Box implementation for the SubByte transformation(S-box) in the Advanced Encryption Standard (AES) algorithm on FPGA. S-box dominated the hardware complexity of the AES cryptographic module thus we implement its mathematic equations based on optimized and combinational logic circuits until dynamic power consumption reduced. The complete data path of the S-box algorithm is simulated as a net list of AND, OR, NOT and XOR logic gates, also for increase in speed and maximum operation frequency used 4-stage pipeline in proposed method. The proposed implemented combinational logic based S-box have been successfully synthesized and implemented using Xilinx ISE V7.1 and Virtex IV FPGA to target device Xc4vf100. Power is analized using Xilinx XPower analyzer and achieved power consumption is 29 mW in clock frequency of 100 MHz. The results from the Place and Route report indicate that maximum clock frequency is 209.617 MHz.

Index Terms — Rijndael S-box, Combinational logic, Pipelining, FPGA, VHD*L*

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 [1]. Byte substitution and Inverse Byte Substitution are the most complex steps in the encryption and decryption processes. In these steps each byte of the state array will be replaced with its equivalent byte in the S-box or the Inverse S-box. As AES algorithm use elements within the GF (2^8), each element in the state array represents a byte with a value that varies between 00H-FFH. The S-box has a fixed size of 256 bytes represented as (16*16) bytes matrix [2]. In this paper propose an optimized and pipelined architecture for S-box block in AES based on combinational logic. We used minimum number of logic gate in proposed design. In recent years, a number of researches have been proposed for Implementation of S-box by using the FPGA by [3-17]. In continue we present some researches, in [3], a software method of producing the multiplicative inverse values, which is the generator of S-box values and the possibilities of implementing the methods in hardware applications will be discussed. The method is using the log and antilog values. The method is modified to create a memory-less value generator in AES hardware-based implementation. In [4], they propose an improved masked AND gate, in which the relationship between inputs masked values and masks, is nonlinear. Usually, when converting S-box operations from GF (2^8) to GF ($(((2^2)^2)^2$), all the necessary computations become XOR and AND operations. Therefore, to fully mask AES S-box is to substitute the unmasked XOR and AND operations with the proposed masked AND gate and protected XOR gate. In [5], a general method for sharing common sub-expressions derived from the algebraic finite fields is proposed. Furthermore, they present a randomly configurable architecture for protecting S-box transformation. [6], presents a compact implementation of the S-box of Pomaranch stream cipher using composite field arithmetic in GF ($(2^3)^3$). It describes a systematic exploration of different choices for the irreducible polynomials that generate the extension fields. It also examines all possible transformation matrices that map one field representation to another. In [7], they propose countermeasure techniques for AES with S-box hiding using four different implementations of S-boxes using composite fields. The proposed work by [8], employs a combinational logic design of S-box implemented in FPGA. The architecture employs a Boolean simplification of the truth table of the logic function with the aim of reducing the delay. The S-box is designed using basic gates such as AND gate, NOT gate, OR gate and multiplexer. In [9], presents FPGA implementation and overhead evaluation for an algorithmic Differential Power Attack (DPA) countermeasure for AES. In [10], presents a new efficient method for implementation of the AES byte substitution function. It is aimed at the AES implementation in non-volatile FPGAs featuring volatile embedded RAM blocks. The method uses a pair of linear

Copyright © 2013 MECS

feedback shift registers to generate substitution tables into embedded RAMs. The proposed solution requires less space and is faster than the one implementing whole S-boxes in the logic area, and it is especially suited to a power-aware AES implementation. In[11], investigate a new compact digital hardware implementation of AES Structure with integrated S-box and Inverse S-box transformation which minimizes the computation cost of the relevant arithmetic in the finite field GF (2^8), including the cost of the mapping. This approach has advantages over a straightforward implementation using read-only memories for table lookups. The resulting S-box design with subfield operations in GF (((((2^2)^2)^2)^2 offers a reduction in the reconfigurable logic by 81% low gate count as compared to Look Up Table(LUT) and 23% better performance in area and faster by 3% in comparison with one using GF ((2^4)^2). A high speed architecture for composite field arithmetic based S-boxs transformation used in AES is present by[12]. In [13], two instructions for S-box access are designed by constructing a novel flexible on-chip parallel substitution box unit that consists of multiple LUT and a post-processing module. The box unit is integrated into the 32-bit configurable Leon2 processor. Configuration of Leon2 is presented. Implementing this extended processor core on FPGA shows that the parallel substitution box unit uses very small amount of hardware resources. The proposed architecture is derived by extending the pre-computation technique suggested recently by Liu and Parhi [14] to a recently proposed architecture of AES S-box due to Rashmi, Mohan and Anami [15]. To reduce implementation overhead the masked compact S-box, proposed by Canright [16], was chosen to implement a DPA countermeasure on an SRAM FPGA. This paper is organized as follows. In section II description of the sub-byte transformation, proposed method and proposed architecture is presented. Section III discusses comparison of the hardware implementation and chip utilization taken from Xilinx ISE that verifies the performance of the proposed work. Section IV is the conclusion.

II. The SubByte Transformation

Paper presents a combinational logic based Rijndael S-box implementation for the Sub Byte transformation in the AES algorithm for FPGA. We for implementation S-box use from [17-18]. Using combinational logic for implement S-box has small area occupancy and high throughput, and as compared to the typical ROM based LUT 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 [17].

SubByte:

1- Multiplicative Inversion in GF (2^8)

2- Affine Transformation

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. The individual bits in a byte representing a GF (2^8) element can be viewed as coefficients to each power term in the GF (2^8) polynomial. For instance, $\{10001011\}_2$ is representing the polynomial $q7 + q3 + q + 1$ in GF (2^8). From [18], 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 [18].

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

From [17], 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 [17].

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

The above equation indicates that there are multiply, addition, squaring and multiplication inversion in GF (2^4) operations in Galois Field. Each of these operators can be transformed into individual blocks when constructing the circuit for computing the multiplicative inverse. From this simplified equation, the multiplicative inverse circuit GF (2^8) can be produced as shown in Fig.1.

Copyright © 2013 MECS

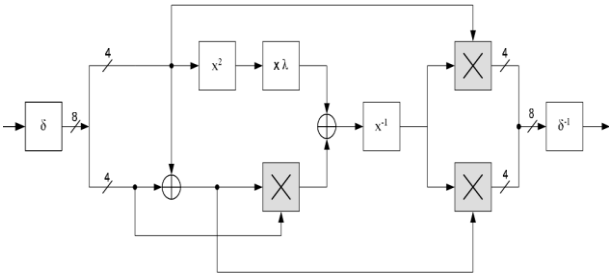

Fig.1: Multiplicative inversion module for the S-box.

The legends for the blocks within the multiplicative inversion module from above are illustrated in Table I.

Table I: 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 λ	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 [14].

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

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 a case of matrix multiplication as shown in below , where q_7 is the most significant bit and q_0 is the least significant bit [17]. Proposed implementation of the affine transformation is shown in Fig.3.

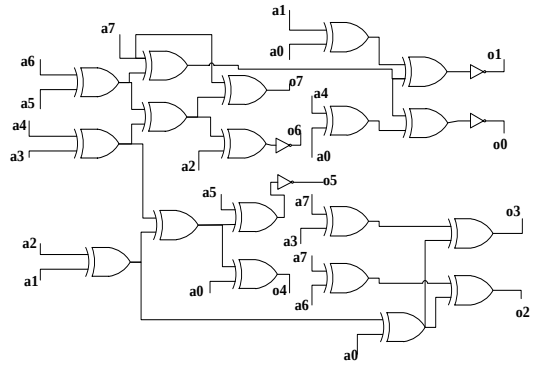

Fig.2: Proposed implementation of the affine transformation.

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

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

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. Proposed implementation of δ*q is shown in Fig.3.

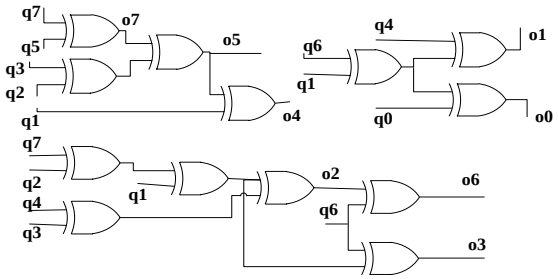

Fig.3: Proposed implementation of δ*q.

Also proposed implementation of δ^{-1}*q is shown in Fig.4.

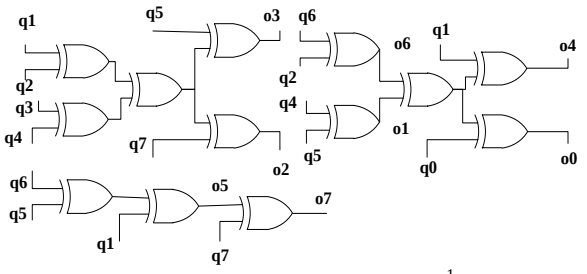

Fig.4: Proposed implementation of δ^{-1}*q.

From [18] and [19], 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 $q_L = \{11\}_2$. q_H and q_L can be further decomposed to $\{1\}_2x+\{0\}_2$ and $\{1\}_2x+\{1\}_2$ respectively. Using this idea, the logical equations for the addition, squaring, multiplication and inversion can be derived.

3) Squaring in GF(2^4)

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.

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

$$k_L = {q_H}^2x^2 + q_Hq_Lx + q_Hq_Lx + {q_L}^2 = {q_H}^2x^2 + {q_L}^2$$

The x^2 term can be modulo reduced using the irreducible polynomial from (1), $x^2+x+\varphi$. By setting $x^2=x+\varphi$ and replacing it into x^2. Doing so yields the new expressions below.

$$k = {q_H}^2(x+\varphi) + {q_L}^2$$

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

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).

$$k_H = {q_H}^2 = (q_3q_2)^2 = (q_3x+q_2)^2$$

$$k_H = {q_3}^2x^2 + q_3q_2x + q_3q_2x + {q_2}^2 = q_3x^2 + q_2$$

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

$$k_H = q_3(x+1) + q_2$$

$$k_3x + k_2 = q_3x + (q_2+q_3) \in GF(2) \qquad (2)$$

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

$$k_L = (q_3x+q_2)^2(\{1\}_2x+0) + (q_1x+q_0)^2$$

$$k_L = {q_H}^2\varphi + {b_L}^2 = (q_3q_2)^2\{10\}_2 + (q_1q_0)^2$$

$$k_L = ({q_3}^2x^2 + q_2q_3x + q_2q_3x + {q_2}^2)(x) + ({q_1}^2x^2 + q_0q_1x + q_0q_1x + {q_0}^2)^2\{10\}_2 + (q_1q_0)^2$$

$$k_L = q_Hx^3 + q_2x + q_1x^2 + q_0$$

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.

$$k_L = q_3(1) + q_2x + q_1(x+1) + q_0$$

$$k_1x + k_0 = (q_2+q_1)x + (q_3+q_1+q_0) \in GF(2) \qquad (3)$$

From equations (2) and (3), the formula for computing the squaring operation in GF (2^4) is acquired as shown below.

$$k_3 = q_3$$
$$k_2 = q_3 \oplus q_2$$
$$k_1 = q_2 \oplus q_1$$
$$k_0 = q_3 \oplus q_1 \oplus q_0$$

Proposed implementation of above equations is shown in Fig.5.

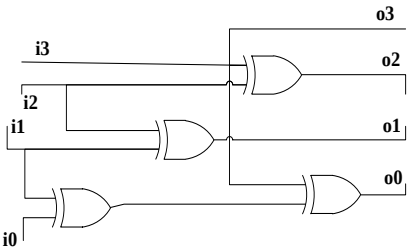

Fig.5: Proposed implementation of Squarer in GF(2^4).

4) Multiplication with constant, λ

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

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

$$k = (q_Hx+q_L)(\lambda_Hx+\lambda_L)$$

Modulo reduction can be performed by substituting $x^2=x+\varphi$ using the irreducible polynomial in (4) to yield the expression below.

$$k = q_H \lambda_H (x+\varphi) + q_L \lambda_H x$$

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

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

$$k_H = q_H \lambda_H + q_L \lambda_H$$
$$k = (q_3 q_2)(11_2) + (q_1 q_0)(11_2)$$
$$k_H = (q_3 x + q_2)(x+1) + (q_1 x + q_0)(x+1)$$
$$k_H = q_3 x^2 + (q_3 + q_2)x + q_2 + q_1 x^2 + (q_1 + q_0)x + q_0$$

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

$$k_H = q_3(x+1) + (q_3 + q_2)x + q_2 + q_1(x+1) + (q_1 + q_0)x + q_0$$

$$k_H = (q_3 + q_3 + q_2 + q_1 + q_1 + q_0)x + (q_3 + q_2 + q_1 + q_0)$$

$$k_3 x + k_2 = (q_2 + q_0)x + (q_3 + q_2 + q_1 + q_0) \in GF(2) \qquad (4)$$

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

$$k_L = q_H \lambda_H \varphi$$
$$k_L = (q_3 q_2)(11_2)(10_2)$$
$$k_L = (q_3 x + q_2)(x+1)(x)$$
$$k_L = q_3 x^3 + q_2 x^2 + q_3 x^2 + q_2 x$$

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

$$k_L = q_3(1) + q_2(x+1) + q_3(x+1) + q_2 x$$

$$k_L = (q_3 + q_2 + q_2)x + (q_3 + q_3 + q_2) \qquad (5)$$

$$k_1 x + k_0 = (q_3)x + (q_2) \in GF(2)$$

From equations (4) and (5) combined, the formula for computing multiplication with constant λ is shown below.

$$k_3 = q_2 \oplus q_0$$
$$k_2 = q_3 \oplus q_2 \oplus q_1 \oplus q_0$$
$$k_1 = q_3$$
$$k_0 = q_2 \qquad (6)$$

Proposed implementation of multiplication with constant λ is shown in Fig.6.

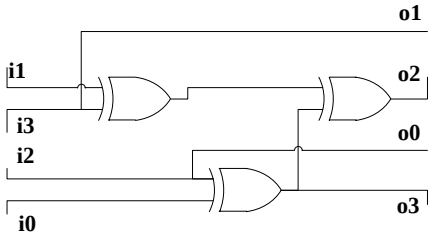

Fig.6: Proposed implementation of multiplication with constant λ.

5) GF(24) Multiplication

Let k = qw, where k= {k3 k2 k1 k0}2, q = {q3 q2 q1 q0}2 and w = $\{w_3 w_2 w_1 w_0\}_2$ are elements of GF (2^4).

$$k = \left(\underbrace{k_3 k_2}_{k_H}\underbrace{k_1 k_0}_{k_L}\right) = k_H x + k_L = \left(\underbrace{q_3 q_2}_{q_H}\underbrace{q_1 q_0}_{q_L}\right)\left(\underbrace{w_3 w_2}_{w_H}\underbrace{w_1 w_0}_{w_l}\right) = (q_H x + q_L)(w_H x + w_L)$$

$$k = (q_H w_H)(x+\varphi) + (q_H w_L + q_L w_H)x + q_L w_L$$

Substituting the x^2 term with $x^2 = x + \varphi$ yields the following.

$$k = (q_H w_H)x^2 + (q_H w_L + q_L w_H)x + q_L w_L$$
$$k = k_H x + k_L = (q_H w_H + q_H w_L + q_L w_H)x + q_H w_H \varphi + q_L w_L \in GF(2^2) \qquad (7)$$

Equation (7) 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 (7) were to be broken down to GF (2). Thus, the formula for multiplication in GF (2^2) and constant φ will be derived instead. Fig.7 below shows the hardware implementation for multiplication in GF (2^4).

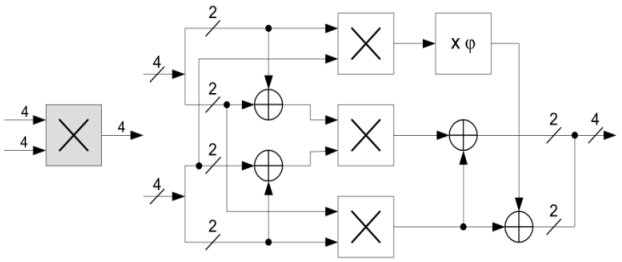

Fig.7: Hardware implementation of multiplication in GF (2^4).

6) GF(22) Multiplication

Let k=qw, where k = $\{k_1 k_0\}2$, q= $\{q_1 q_0\}_2$ and w = $\{w_1 w_0\}_2$ are elements of GF (2^2).

$$k = (k_1 k_0) = k_1 x + k_0 = (q_1 q_0)(w_1 w_0) = (q_1 x + q_0)(w_1 x + w_0)$$

$$k = q_1 w_1 x^2 + q_0 w_1 x + q_1 w_0 x + q_0 w_0$$

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

$$k = q_1 w_1 (x+1) + q_0 w_1 x + q_1 w_0 x + q_0 w_0$$
$$k_1 x + k_0 = (q_1 w_1 + q_0 w_1 + q_1 w_0)x + (q_1 w_1 + q_0 w_0) \in GF(2) \quad (8)$$

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

$$k_1 = q_1 w_1 \oplus q_0 w_1 \oplus q_1 w_0$$
$$k_0 = q_1 w_1 \oplus q_0 w_0 \quad (9)$$

Fig.8 illustrates its hardware implementation.

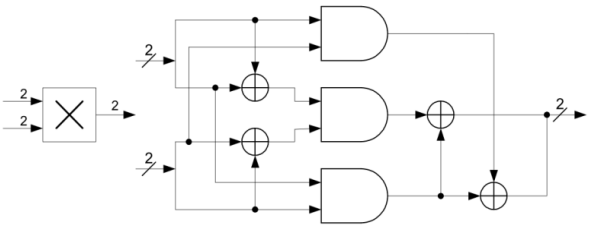

Fig.8: Hardware implementation of multiplication in GF (2^2).

The above hardware implementation is different of the (9) for the computation of k_1. It can be proven that the implementation above for computing k_1 would result to the expression in (9), as shown below.

$$k_1 = (q_1 \oplus q_0)(w_1 \oplus w_0) \oplus (q_0 w_0)$$
$$k_1 = (q_1 w_1) \oplus (q_0 w_1) \oplus (q_1 w_0) \oplus (q_0 w_0) \oplus (q_0 w_0)$$
$$k_1 = (q_1 w_1) \oplus (q_0 w_1) \oplus (q_1 w_0)$$

7) Multiplication with constant φ

Let k=qφ, where k = $\{k_1 k_0\}_2$, q = $\{q_1 q_0\}_2$ and φ = $\{10\}_2$ are elements of GF(2^2).

$$k = k_1 x + k_0 = (q_1 q_0)(10_2) = (q_1 x + q_0)(x)$$

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

$$k_1 = q_1 x^2 + q_0 x$$
$$k = q_1 (x+1) + q_0 x$$
$$k = (q_1 + q_0)x + (q_1) \in GF(2) \quad (10)$$

From (10), the formula for computing multiplication with φ can be derived and is shown below.

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

The hardware implementation of multiplication with φ is shown below in Fig.9.

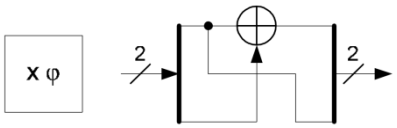

Fig.9: Hardware implementation of multiplication with constant φ.

8) Multiplicative Inversion in GF(24)

In [19] has derived a formula to compute the multiplicative inverse of q (where *q* is an element of GF (2^4)) such that $q^{-1}=\{q_3^{-1},q_2^{-1},q_1^{-1},q_0^{-1}\}$ The inverses of the individual bits can be computed from the equation below [19].

$$q_3^{-1} = q_3 \oplus q_3 q_2 q_1 \oplus q_3 q_0 \oplus q_2$$
$$q_2^{-1} = q_3 q_2 q_1 \oplus q_3 q_2 q_0 \oplus q_3 q_0 \oplus q_2 \oplus q_1 q_2$$
$$q_1^{-1} = q_3 \oplus q_3 q_2 q_1 \oplus q_3 q_1 q_0 \oplus q_2 \oplus q_2 q_0 \oplus q_1$$
$$q_0^{-1} = q_3 q_2 q_1 \oplus q_3 q_2 q_0 \oplus q_3 q_1 \oplus q_3 q_1 q_0 \oplus q_3 q_0 \oplus q_2 \oplus q_2 q_1 \oplus q_2 q_1 q_0 \oplus q_1 \oplus q_0$$

Proposed implementation of these equations is shown in Fig.10.

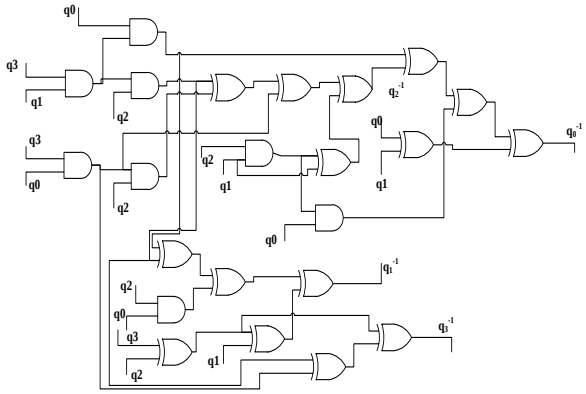

Fig.10: Proposed implementation of Multiplicative Inversion in GF (2^4)

As explained proposed implementation for S-box is based on pipelining until performance and speed is increased. Fig.11 shows proposed pipelined S-box.

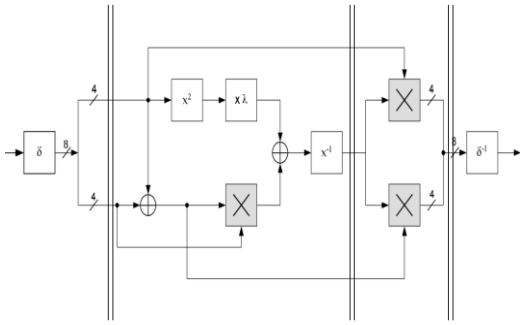

Fig.11: Proposed pipelined S-box.

Copyright © 2013 MECS

IV. COMPARISON

We design a FPGA implementation of the S-box algorithm based on combinational logic. In this paper proposed method has been written by VHDL hardware description language. In order to get actual numbers for the hardware usage this work was synthesized and implemented using Xilinx 7.1 software, Virtex-4 FPGA to target device Xc4vfx100 also power is analyzed using Xilinx XPower analyzer. Table II shows utilization hardware and performance in different works and proposed method for S-box also Table III shows power consumption in proposed method for S-box.

Table III: Power consumption in proposed method for S-box.

Clock(MHz)	25	50	75	100
Power(mW)	20	23	26	29

Table II: Comparisons of hardware utilization and performance

Implementation	Device	FFs	4 input LUTs	Slices	F_Max (MHz)	Gates	Delay (ns)
[4]	Virtex-5	---	---	82	---	635	---
[6], (TA11)	Virtex-4 xc4vfx120ff668-10	120	1401	730	67.924	---	---
[8]	Virtex2 XC2V1000	---	---	153	---	1650	10.80
[9], Unsecured	Virtex-4 LX25 FF676	---	---	36	88	---	---
[9], Secure with masking	Virtex-4 LX25 FF676	---	---	100	60	---	---
[11]	XC2V1000	50	---	380	---	126	14.65
[12]	XC2V6000-6	---	---	133	560	---	3.56
[13] (No. of BRAM= 20)	Virtex2 XC2V3000-FG676-6	---	---	5148	60	---	---
Proposed work	Xc4vfx100	24	88	45	209.61	---	2.581

in different works and proposed method for S-box.

V. CONCLION

The aim of paper is design and implementation of the optimized combinational logic based Rijndael S-Box on FPGA. Proposed method is based on combinational logic, thus it is low power and number of logic gates is very low. The approach used for increase performance is pipelining technique we use 4-stage pipelining in S-Box design. The proposed architecture only is based on *XOR, AND, NOT*, and *OR* logic gates. This method has more speed and low power than other work.

REFERENCES

[1] Aseem Jagadev, "Advanced Encryption Standard (AES) Implementation", Bachelor of Technology THESIS, Department of Electronics and Communication Engineering 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] Naziri, S.; Idris, N., "The memory-less method of generating multiplicative inverse values for S-box in AES algorithm**"**, Electronic Design, ICED. IEEE International Conference, 2008, pp. 1-5.

[4] Zeng, Juanli et al, "Improvement on masked S-box hardware implementation", Innovations in Information Technology (IIT), IEEE International Conference, 2012, pp 113-116.

[5] Jun-Hong Chen et al; Ming-Der Shieh, "Exploration of Low-Cost Configurable S-box Designs for AES Applications", Embedded Software and Systems, ICESS '08. IEEE International Conference, 2008, pp. 422-428

[6] Ebrahimi Atani et al, "Low cost implementation of Pomaranch S-Box", Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronics Systems Technology, Wireless VITAE, IEEE International Conference, 2009, pp. 875-879.

[7] Jae Seong Lee et al, "S-box hiding methods for AES hardware chips against differential power analysis based on composite field", Network Infrastructure and Digital Content, 2nd IEEE International Conference on, 2010, pp. 778-782.

[8] Ahmad, N et al, "Design of AES S-box using combinational logic optimization ", Industrial Electronics & Applications (ISIEA), IEEE Symposium , 2010, pp. 696- 699.

[9] Kamoun, N et al.," SRAM-FPGA implementation of masked S-Box based DPA countermeasure for AES", Design and Test Workshop. 3rd IEEE International Conference, 2008, pp. 74-77.

[10] Gaspar, L et al "Efficient AES S-boxes implementation for non-volatile FPGAs " , Field Programmable Logic and Applications, IEEE International Conference, 2009, pp. 649- 653.

[11] Nalini, C. et al "Optimized S-box design AES core ", Information and Communication Technology in Electrical Sciences (ICTES), ICTES. IET-UK IEEE International Conference, 2007, pp. 843- 849.

[12] Rachh, R.R. et al., "High speed S-box architecture for Advanced Encryption Standard ", Internet Multimedia Systems Architecture and Application (IMSAA), 5th IEEE International Conference, 2011, pp. 1- 6.

[13] Duan Cheng-Hua et al, "Fast S-Box Substitution Instructions and Their Hardware Implementation for Accelerating Symmetric Cryptographic Processing ", E-Business and Information System

Security, EBISS '09 IEEE International Conference , 2009 , pp. 1- 4.
[14] R. Liu, K.K.Parhi, "Fast composite field architectures for Advanced Encryption tandard", Proceedings GLSVLSI'08, Orlando, Florida, USA, pp 65-70, May 4–6, 2008.
[15] Rashmi Ramesh Rachh, et al "Efficient Implementations of AES S box and Inverse S-box", Proc. IEEE TENCON, Singapore,pp 1-6, 2009.
[16] D. Canright, "A Very Compact S-Box for AES", Workshop on Cryptographic Hardware and Embedded Systems (CHES2005), Lecture Notes in Computer Science 3659, pp.441-455, Springer-Verlag 2005.
[17] Akashi Satoh, et al "A Compact Rijndael Hardware Architecture with S-Box Optimization.", Springer-Verlag Berlin Heidelberg, 2001.
[18] Vincent Rijmen, "Efficient Implementation of the Rijndael S-Box.", Katholieke Universiteit Leuven, Dept. ESAT. Belgium.
[19] Tim Good and Mohammed Benaissa, "Very Small FPGA Application-Specific Instruction Processor for AES." IEEE Transactions on Circuits and Systems – I: Regular Papers, Vol. 53, No. 7, July 2006.

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.

Copyright © 2013 MECS