A Combinational Logic Implementation of S-box of AES

P.V.S.Shastry, Anuja Agnihotri, Divya Kachhwaha, Jayasmita Singh
Department of Electronics and Telecommunication,
MKSSS's Cummins College of Engineering for Women,
Pune, INDIA

Dr.M.S.Sutaone
Department of Electronics and Telecommunication,
PIET's College of Engineering,
Pune, INDIA

***Abstract*—This paper presents a combinational logic based Rijndael S-box implementation for the SubByte transformation on ASIC. Combinational implementation of S-box results in low cost, small area occupancy and high throughput as compared to the typical ROM based lookup table implementation with fixed and unbreakable access time. S-box has been implemented using 0.18µm CMOS standard cell library at 1.62V and runs at clock frequency of 71.43MHz. We could achieve throughput of 571.5Mbps with core utilization of 85%, core area occupied is 39.88.4µm^2 using only 178 cells. Total power dissipation of the S-box implementation is 0.611mW which is quite lower than other literature available.**

***Keywords* – AES, ASIC, S-Box, Composite Field Arithmetic.**

I. INTRODUCTION

Advanced Encryption Standard (AES) successor of DES was awarded to the Rijndael algorithm[1] by National Institute of Standards and Technology in 2001. It was first ever kind of encryption standard finalized after considering the Hardware implementation of the finalists. While multiple rounds of evaluations were conducted, they were primarily conducted on FPGAs and ASIC designs on the three optimization fronts viz. throughput, area and power.

AES was realized using various architectural transforms like fully rolled, fully unrolled, pipelined as well as partial pipelined techniques. Typically ASIC implementation of AES [2], [3], [4], [5] and [6], while implementation on FPGA by [8], [9], [10], [11] and [12]. There are implementations which have concentrated on sub processes like S-box [7], [13] and [14]. Implementation of S-box have been done using ROM based look-up-tables and logic combinational methods. Computation of S-box exploiting Galois Fields[13] resulted better results than direct implementation. Transforming GF(2^8) to GF(((2^2)^2)^2) then computing multiplicative inverse helped in reducing the complexity[15]. In this paper we have tried to convert the GF(2^8) representation into GF(2^4) to compute the multiplicative inverse, which has given a reasonable amount of reduction in complexity.

The sections of this paper are organized as follows. Section II explains the basic understanding of transformations in Galois Fields while computing S-box, Section III explains the proposed architecture using combinational logic, Section IV provides insight of implementation and result analysis.

II. PRELIMINARIES

The construction of S-Box involves 8-bit multiplicative inverse followed by affine transform. The byte representations in GF(2^8) are viewed in terms of polynomial forms. All the operations in Galois Field GF(2^8) are accomplished by taking modulo operations using corresponding fixed irreducible polynomial *m(x)*.

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

Inversion in GF(2^8) has many levels of hierarchy. Multiplicative inverse will be computed by decomposing complex GF(2^8) into lower order fields of GF(2^4) and GF(2^2). Multiplicative inverse can be viewed as in equation (1).

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

A Transforming GF(2^8) to GF(((2^2)^2)^2)

In order to achieve transformation of higher order Galois Fields in to lower order, irreducible polynomials[16] as given in equation (2) are used.

$GF(2) \rightarrow GF(2^2)$: $I_0(X)=x^2+x+1;$

$GF(2^2) \rightarrow GF(2^4)$: $I_1(X)=x^2+x+\varphi;$

$GF(2^4) \rightarrow GF(2^8)$: $I_2(X)=x^2+x+\lambda;$

Where $\varphi = \{10\}_2$, $\lambda = \{1100\}_2$. (2)

978-1-61284-857-0/11/$26.00

B. *Multiplicative inverse in Glois Fields*

In order to find the multiplicative inverse to a polynomial which is based on GF(2^8), the polynomial has to be firstly represented in composite field. An isomorphic mapping function $f(x) = \delta^* x$ and its inverse need to be applied to map the representation of an element GF(2^8) to its composite field and vice versa.

The δ required for isomorphic mapping of GF(2^8) into composite field and vice versa is done using the equations (3).

$$\delta = \begin{pmatrix} 1&0&1&0&0&0&0&0 \\ 1&1&0&1&1&1&1&0 \\ 1&0&1&0&1&1&0&0 \\ 1&0&1&0&1&1&1&0 \\ 1&1&0&0&0&1&1&0 \\ 1&0&0&1&1&1&1&0 \\ 0&1&0&1&0&0&1&0 \\ 0&1&0&0&0&0&1&1 \end{pmatrix} \quad \delta^{-1} = \begin{pmatrix} 1&1&1&0&0&0&1&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&1&0&0&0&1&0 \\ 0&1&1&1&0&1&1&0 \\ 0&0&1&1&1&1&1&0 \\ 1&0&0&1&1&1&1&0 \\ 0&0&1&1&0&0&0&0 \\ 0&1&1&1&0&1&0&1 \end{pmatrix} \quad (3)$$

The input polynomial is first multiplied by the δ and after computing the multiplicative inverse it is mapped back to GF(2^8) using δ^{-1} matrix.

C. *Affine Transformation*

After the multiplicative inverse has been computed, affine transformation is carried out on the product polynomial. The affine transformation is a simple column multiplication and XOR with a constant column matrix.

$$\alpha = \begin{pmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{pmatrix} X \begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \quad \text{-- (3)}$$

III. PROPOSED ARCHITECTURE

As explained above the multiplicative inverse of the polynomial was found using the architecture in Figure 1. The architecture consists of addition, multiplication and squaring which can be implemented separately as given in the following sections. Primarily all the operations are conducted in GF(2^4) resulting into lowering the complexity.

The upper nibble of $f(x)$ is squared and then multiplied with λ. Lower and upper nibble of $f(x)$ are XORed and multiplied with lower nibble in GF(2^4). As shown in the figure 3.1 multiplicative inverse in GF(2^4) is computed and used to get upper and lower nibble of final multiplicative inverse in GF(2^8).

Addition in GF(2^4)

Addition of 2 elements which are obtained after splitting the number in GF(2^8) can be translated into bitwise XOR operation.

A. *Squaring in GF(2^4)*

Squaring in GF(2^4) can be performed by converting the GF(2^4) elements into GF(2^2) with the help of irreducible polynomial as given in the Section II which yields to simple Boolean expression as given in equation (4).

$$\begin{aligned} d_3 &= b_3 \\ d_2 &= b_3 \oplus b_2 \\ d_1 &= b_2 \oplus b_1 \\ d_0 &= b_3 \oplus b_1 \oplus b_0 \end{aligned} \quad (4)$$

Where d is the element of GF(2^4).

B. *Multiplication with constant λ*

The polynomial 'd' obtained after squaring 'b' is to be multiplied with the constant λ where $\lambda = \{1\ 1\ 0\ 0\}$. The product is reduced using the polynomial $x^2 = x + \varphi$. This operation can be further simplified and 'g' could be achieved using simple Boolean expressions as given in equation (5).

$$\begin{aligned} g_3 &= d_2 \oplus d_0 \\ g_2 &= d_3 \oplus d_2 \oplus d_1 \oplus d_0 \\ g_1 &= d_3 \\ g_0 &= d_2 \end{aligned} \quad (5)$$

C. *Multiplication in GF(2^4)*

The multiplication in GF(2^4) for $f = e.c$; $j = e.i$; and $k = b.i$ where $b, c, d, e, f, g, h, i, j$ and k are elements in GF(2^4). In order to simplify this step further, a sub block 'm' is used as shown in Figure 2 which involves actual multiplication in GF(2^2) using equations (6)

$$\begin{aligned} i_1 &= h_1 w_1 \oplus h_0 w_1 \oplus h_1 w_0 \\ i_0 &= h_1 w_1 \oplus h_0 w_0 \end{aligned} \quad (6)$$

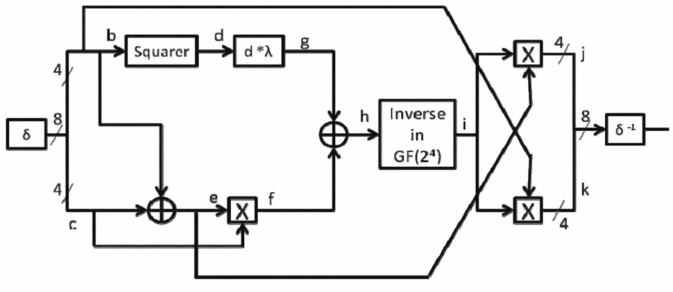

Figure 1. Multiplicative Inverse in GF(2^4)

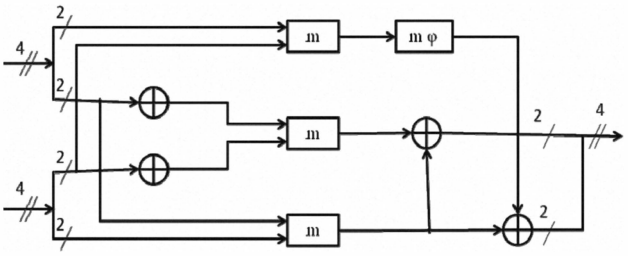

Figure 2. Multiplication in GF(2^2)

Multiplication with φ is done using Boolean expression of equation (7).

$$x_1 = y_1 \oplus y_0$$
$$x_0 = y_1 \quad (7)$$

Where x,y,i,h and w are elements in GF(2^2).

D. *Multiplicative inverse in GF(2^4)*

Using the expressions from [17], the inverse of the individual bits can be computed from equations (8).

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

$$q_2^{-1} = q_3.q_2.q_1 \oplus q_3.q_2.q_0 \oplus q_3q_0 \oplus q_2 \oplus q_2.q_1$$

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

$$q_0^{-1} = q_3.q_2.q_1 \oplus q_3.q_2.q_0 \oplus q_3.q_1 \oplus q_3.q_1.q_0 \oplus q_3.q_0 \oplus q_2 \oplus q_2.q_1 \oplus q_2.q_1.q_0 \oplus q_1 \oplus q_0 \quad (8)$$

IV. ASIC IMPLEMENTATION

This section explains various results obtained by synthesis of S-box architecture. The design is coded and simulated in VHDL in ActiveHDL to verify the functionality of the architecture. All compiled VHDL files are saved in RTL as source files. Scripts file defining a clock period of 14000ps i.e. a frequency of 71.43MHz. This frequency selection was result of multiple iterations of synthesis and analysis for reasonable slack and optimized core utilization. Also external delay constraint was set at 1000ps for all input and output ports. We have used TSMC 0.18µm technology file for synthesis.

RTL Compiler was used for synthesis and Encounter for floor planning, place and route. The obtained synthesis results are mentioned in Table I.

TABLE I
SYNTHESIS RESULTS

	Parameters	*Values*
1.	Number of cells	178
2.	Area(µm^2)	3988.35
3.	Total Power (mW)	0.611
4.	Slack(ps)	+1601
5.	Memory usage	78328K

The slack resulted after the synthesis is sufficiently large to accommodate further delay contingencies. Net-list file is imported in Encounter for place and route of the design. Floor planning resulted in core utilization of up to 85% which a reasonable to allow easy routing of clock and other signals without much addition of additional load on clock. Power grid layout is of type Stripe Breaking with stripes inside the block ring, which were implemented using M6/M5 metal layers for rings and M6 for stripe. Core works at 1.62V Vdd. Placement and intra-cell routing was done using M2 to M4. We have selected medium effort for optimization and obtained timing, power and area reports.

Timing analysis is performed on wire load model. Pre CTS setup time analysis for all paths results in WNS of 3.975ns and density of 74%. A two level clock tree generated for minimum clock skew. The post CTS hold time analysis gave us WNS of 0.903ns and density of 74.38%. Total power dissipation for the design achieved is 0.611mW and leakage power contribution was only 0.184nw.

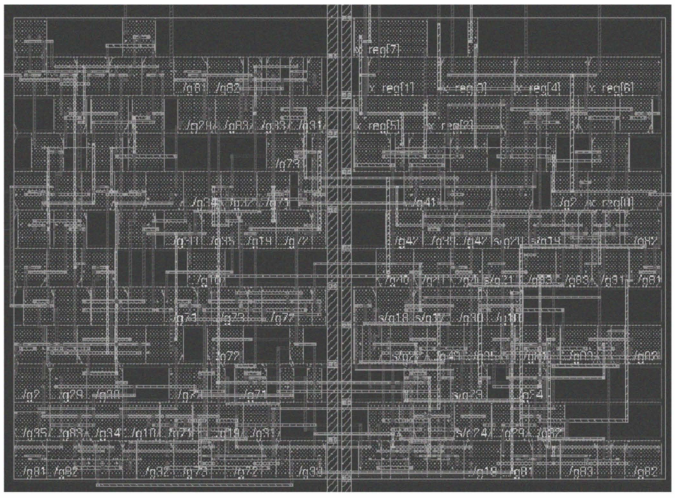

Figure 3. Final Layout of S-box

TABLE II
COMPARISON OF ASIC IMPLEMENTATION OF S-BOX USING 0.18µm CMOS TECHNOLOGY

S-box Architecture	*Our Design*	*[13]*	*[14]*
Number of Cells	178	104	--
Power (mW)	0.611	1.64	0.85
Area (μm^2)	3988.35	4740.00	--
Latency (ns)	0.123	--	1.00
Core Voltage (V)	1.62	1.80	--

V. CONCLUSION

The results of our implementation is compared with previous implementations and tabulated in Table II. The final layout achieved after routing is shown in Figure 3. Even though there are many ASIC implementations of AES as mentioned in the Chapter I of this paper, we have considered only those for comparison which have used 0.18µm technology standard cell library and implemented S-box.

From Table II we conclude that our S-box implementation is compact, consumes low power and has low complexity. It has low latency of 0.123ns while consuming 0.611mW power and occupying 3988.35μm^2 at 85% of core utilization, which is quite better than implementations being compared in Table II. The design was tested at 71.43MHz clock frequency; hence a throughput of 571.5Mbps has been achieved.

ACKNOWLEDGEMENT

We acknowledge to the support extended by the Department of Electronics and Telecommunication of MKSSS's Cummins College of Engineering for Women by providing the Cadence tools. We also extend our gratitude to the technical support team of Cadence.

REFERENCES

[1] J. Daeme and V.Rijmen. AES proposal: Rijndael. *NIST AES Proposal, June 1998.*

[2] Edwin NC Mui, "*Practical Implementation of Rijndael S-Box Using Combinational Logic"*, Custom R & D Engineer Texco Enterprise Pvt.Ltd.

[3] Y.X. Su, J. Mathew, J. Singh *a*nd D. K. Pradhan, *"Pseudo Parallel Architecture for AES with Error Correction"* 21st IEEE International System on Chip Conference (IEEE SoCC 2008), September 2008, pp.187 – 190.

[4] Yibo Fan, Takeshi Ikenaga, Yukiyasu Tsunoo, and Satoshi Goto, *"A Low-cost Reconfigurable Architecture for AES Algorithm*", World Academy of Science, Engineering and Technology 41, 2008.

[5] Namin Yu and Howard M. Heys. *"A compact ASIC implementation of the advanced encryption standard with concurrent error detection.*". Proceeding CSS '07 Proceedings of the Fifth IASTED International Conference on Circuits, Signals and Systems.

[6] Qingfu Cao *, Shuguo Li." *A High-Throughput Cost-Effective ASIC Implementation of the AES Algorithm*" IEEE 8th International Conference on ASIC, 2009. ASICON '09, pp 805-808.

[7] Cheng Wang and Howard M. Heys. "*Using a Pipelined S-Box in Compact AES Hardware Implementations*", Proceedings of IEEE International NEWCAS Conference, Montreal, Canada, June 2010.

[8] Hua Li. "A New CAM Based S/S−1-Box Look-up Table in AES", IEEE International Symposium on Circuits and systems 2005, pp-4634-4636 Vol.5.

[9] Oscar Perez, Yves Berviller, Camel Tanougast, Serge Weber, "*Comparison of various strategies of implementationof the algorithm of encryption AES on FPGA",* IEEE ISIE 2006, July 9-12, 2006, Montreal, Quebec, Canada.

[10] Yu-Jung Huang, Yang-Shih Lin, Kuang-Yu Hung, Kuo-Chen Lin, "*Efficient Implementation of AES IP*", IEEE Asia Pacific Conference on Circuits and Systems 2006, pp 1418-1421.

[11] Ming-Haw Jing, Zih-Heng Chen, Jian-Hong Chen, Yan- Haw Chen "*Reconfigurable system for high-speed and diversified AES using FPGA*", Journal on Microprocessors and Microsystems, Vol 2, Issue 2, March 2007.

[12] Chi-Wu Huang, Chi-Jeng Chang, Mao-Yuan Lin, Hung-Yun Tai, "*Compact FPGA Implementation of 32-bits AES Algorithm Using Block RAM*", IEEE Region 10 Conference TENCON 2007, pp 1-4.

[13] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh, "*A Low-Cost S-box for the Advanced Encryption Standard Using Normal Basis*", IEEE International Conference on Electro/Information Technology, 2009, pp 52-55.

[14] Guido Bertoni and Marco Macchetti and Luca Negri, "*Power-efficient ASIC Synthesis of Cryptographic Sboxes*", GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA. Page 277-281.

[15] Nele Mentens and Lejla Batina and Bart Preneel, and Ingrid Verbauwhede , "*A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-Box*", A.J. Menezes (Ed.): CT- RSA 2005, LNCS 3376, pp. 323–333.

[16] V. Rijmen, "*Efficient Implementation of the Rijndael S-box*," Katholieke Universiteit Leuven, Dept. ESAT, Belgium, 2000. Available at: http://www.esat.kuleuven.ac.be/rijmen/rijndael/sbox.pdf.

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