# Design and Implementation of Pipelined AES Processor on FPGA

Mrs. Roopa Rao[1], Mr.Amaresha S.K[2], Dr. Siva Yellampalli[3]
[1]*Student, UTL Technologies Ltd*
[2]*Assistant Professor, UTL Technologies Ltd*
[3]*Principal, UTL Technologies Ltd*

***Abstract*- The paper presents high speed architecture for the hardware implementation of the Advanced Encryption Standard (AES) Algorithm. The proposed design employs a Galois Field, GF(28), SubBytes (S-Box) transformation based on Rijndael algorithm in the Field Programmable Gate Arrays (FPGAs). The implementation of S-Box carried out by two stages pipelining for the small area occupancy and high throughput. In addition key expansion architecture suitable for the pipelined AES also presented. The design is implemented and synthesized using Xilinx ISE v13.4 and Xilinx Spartan-3E XC3S2500E-4 as a target device. The timing results from the Place and Route report indicate that area occupied by this architecture is 67% of the slices.**

***Index Terms*- AES, Composite Field, FPGA, Galois Field, Key expansion, pipelined S-Box, Rijndael.**

## I. INTRODUCTION

In the introduction, a brief description of AES and previous hardware implantation of the S-Box is provided.

### *A. The Advanced Encryption Standard:*

In today's digital world, encryption is emerging as a disintegrable part of all communication networks and information processing systems for protecting both stored and in transit data. AES algorithm is based on Rijndael algorithm and was accepted as a Federal Information Processing Standard (FIPS) 197 in November 2001 [1]. AES can be implemented using software or hardware. Hardware implementation can be either based on Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC) [2]. AES implemented on FPGA offer more flexibility and security.The efficiency of AES hardware implementation in terms of size, speed security and power consumption depends largely on the AES architecture [3]. For high speed throughput, pipelined [4] register is used.

The two main components in AES design is the cipher and the key expander. The cipher performs encryption or decryption on blocks of input data, while key expander is responsible for the input key for use by the cipher in each round. The AES algorithm has a fixed block size of 128 bits and the length of the cipher key K, is 128, 192 or 256 bits.

Figure 1 shows the block diagram of the AES algorithm.

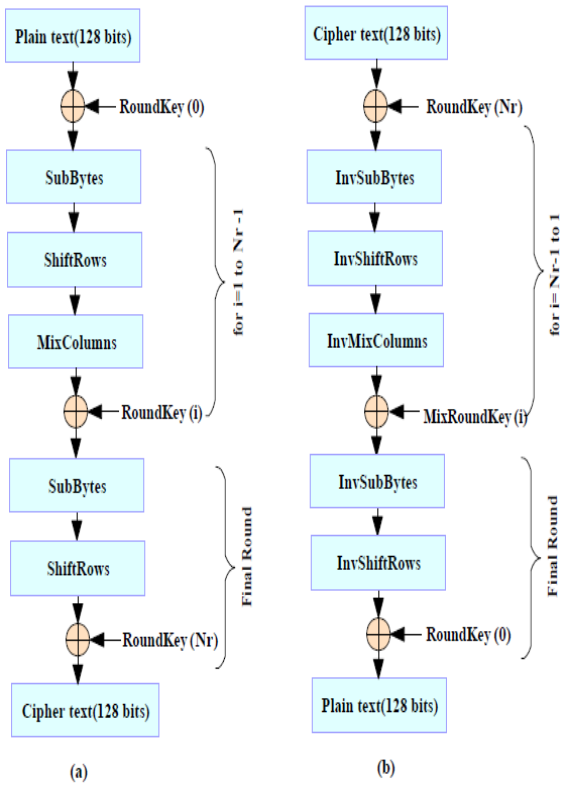


Figure 1: AES Algorithm (a) Encryption (b) Equivalent decryption

The four different functions involved in AES are RoundKey, SubBytes, ShiftRows and MixColumns. The RoundKey transformation involves a bitwise XOR operation between the state array and the resulting RoundKey i.e. output from the Key Expansion function. SubBytes transformation is a highly non-linear byte substitution where each byte in the state array is replaced with another form of look up table called an S-Box. ShiftRows transformation is done by cyclically shifting the rows in the array with different offsets. MixColumns transformation is a column mixing operation, where the bytes in the new column are a function of the 4 bytes of a column in the state array [5]. From Figure 1we observed each round except the final round does not have MixColumns.

*B. Previous implementation of the S-Box:*

There are several options for implementing an AES, S-Box. The most common and straight forward method of SubBytes implementation is to have the pre computed values stored in 512x8 bit ROM based lookup table. However it suffers from an unavoidable delay since ROMs have a fixed access time for its read and write operation. In terms of hardware, such implementation is expensive. The combinational logic is the more suitable method of implementing the S-Box. The design uses composite field approach for calculating multiplicative inverse in the Galois Field GF($2^8$) followed by affine transformation in the binary extension field [6]. This S-Box has the advantages of having small area occupancy and capable of being pipelined by inserting register stages for increased performance in clock frequency.

The structure of this paper is as follows. The section II, describe the pipelined SubBytes transformation design and implementation. Section III briefs the implementation of the key expansion. Section IV presents performance of the FPGA implementation of AES algorithm. Section V outlines the Conclusion of the design implementation

## II. DESIGN IMPLEMENTATION OF THE SUBBYTES TRANSFORMATION

*S-Box construction methodology:*

In this section, the architecture of the two stage pipelined S-Box on Galois Field GF($2^8$) is presented. The S-Box is designed initially performing the multiplicative inverse and then applying affine transformation. Affine transformation is polynomial multiplication followed by the XOR with a constant [7], [14]

*Multiplicative Inversion in GF($2^8$):*

The multiplicative inversion in SubBytes is a hardware demanding operation [8] where 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.

Let$\{100111\}^2$ representing the polynomial, q7+ q3+q+1 in GF($2^8$). Reference [9] stated that any arbitrary polynomial can be represented as bx+c, for the irreducible polynomial $x^2+Ax+B$. So the multiplicative inverse can be computed using (2.1)

$$(bx+c)^{-1} = b(b^2B+bcA+c^{2)-1}x + (c = bA)(b^2B+bcA+c^2)^{-1} \quad 2.1$$

The irreducible polynomial can be simplified as shown in (2.2) and [10]

$$(bx+c)^{-1} = b(b^2 \lambda+c(b+c))^{-1}x+(c+b)(b^2 \lambda+c(b+c))^{-1} \quad 2.2$$

According to (2.2), the multiplicative inversion in GF($2^8$) can be carried out in GF(($2^4$)$^2$) by the architecture as shown in the Figure 2

Here the multipliers in GF($2^4$) can be further decomposed into multipliers in GF($2^2$) and then to GF(2). So the multiplication is simple AND operation. Figure 3 illustrates the implementation of the individual blocks of multiplicative inversion module.

The multiplicative inverse computation will be done by decomposing the more complex GF($2^8$) to a lower order fields of GF($2^1$), GF($2^2$) and GF(($2^2$)$^2$).

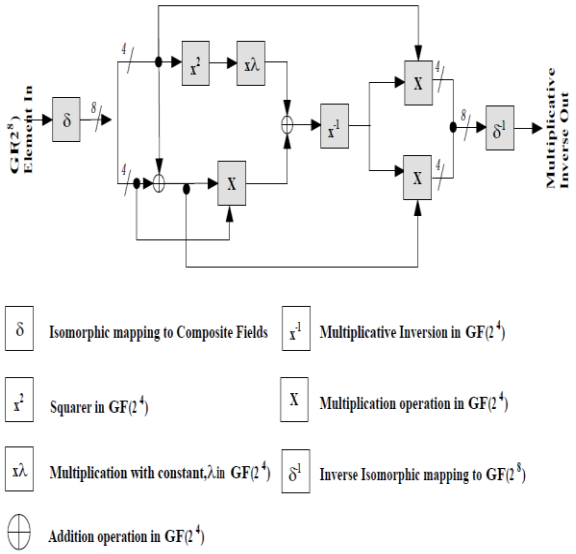


Figure 2: Multiplicative inversion module of S-Box

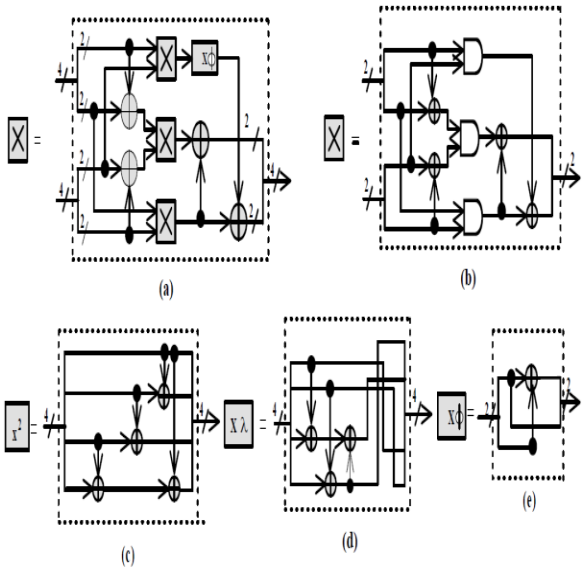


Figure 3: Individual blocks (a) Multiplier in GF($2^4$); (b) Multiplier in GF($2^2$); (c) Squarer in GF($2^4$); (d) constant multiplier(xλ); (e) Constant Multiplier(xφ)

From [3] this can be accomplished from the following irreducible polynomials

$$GF(2^2) \rightarrow GF(2) \qquad : x^2 + x + 1$$
$$GF((2^2)2) \rightarrow GF(x^2) \qquad : x^2 + x + \varphi$$
$$GF(((2^2)^2)^2) \rightarrow GF((2^2))^2 \qquad : x^2 + x + \lambda$$

Where $\varphi = \{10\}_2$ and $\lambda = \{1100\}_2$ 2. 3

The computation of the multiplicative inverse in composite fields cannot be directly applied to an element which is based on GF($2^8$). It has to be mapped to its composite field representation by an isomorphic function δ. After performing the multiplicative inversion the results also to be mapped back from its composite field representation to its equivalent in GF($2^8$) by an inverse isomorphic function $\delta^{-1}$. Both δ & $\delta^{-1}$ can be represented as an 8 x8 matrix. If q be the element in GF($2^8$) , isomorphic and its inverse can be written as δ x q & $\delta^{-1}$x q as shown in (2.4) and (2.5)

$$\delta \text{ x } q = \begin{pmatrix} q_7 \oplus q_5 \\ q_7 \oplus q_6 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_7 \oplus q_5 \oplus q_3 \oplus q_2 \\ q_7 \oplus q_5 \oplus q_3 \oplus q_2 \oplus q_1 \\ q_7 \oplus q_6 \oplus q_2 \oplus q_1 \\ q_6 \oplus q_4 \oplus q_1 \\ q_6 \oplus q_1 \oplus q_0 \end{pmatrix} \quad 2.\ 4$$

$$\delta^{-1} \text{ x } q = \begin{pmatrix} q_7 \oplus q6 \oplus q5 \oplus q1 \\ q6 \oplus q2 \\ q_6 \oplus q5 \oplus q1 \\ q6 \oplus q5 \oplus q4 \oplus q2 \oplus q1 \\ q5 \oplus q4 \oplus q3 \oplus q2 \oplus q1 \\ q7 \oplus q4 \oplus q3 \oplus q2 \oplus q1 \\ q5 \oplus q4 \\ q6 \oplus q5 \oplus q4 \oplus q2 \oplus q0 \end{pmatrix} \quad 2.\ 5$$

The brief description of the individual blocks in the multiplicative inverse module as per Figure 3 :

i. Multiplier in GF($2^4$): Hardware for the GF($2^4$) multiplication are addition and multiplication in GF($2^2$) and multiplication with constant φ. The multiplication in GF($2^2$) requires decomposition to GF(2) .

ii. Multiplier in GF($2^2$): Let k = sw, where k = $\{k_1,k_0\}$, s = $\{s_1,s_0\}$ and w = $\{w_1,w_0\}$ are elements of GF($2^2$) . So the formula to compute the multiplication in GF(2) is as shown in (2.6)

$k_1 = s_1w_1 \oplus s_0w_1 \oplus s_1w_0$
$k_0 = s_0w_0 \oplus s1w1$ 2. 6

iii. Square in GF($2^4$): Let k = $r^2$, where k and r the elements in GF ($2^4$), represented by the binary number :{k3,k2,k1,k0} and {r3,r2,r1,r0} . The formula to compute the squaring operation in GF($2^4$) is as shown in (2.7)

$$\begin{matrix} k3 \\ k2 \\ k1 \\ k0 \end{matrix} = \begin{pmatrix} r_3 \\ r_3 \oplus r_2 \\ r_2 \oplus r_1 \\ r_2 \oplus r_1 \oplus r_0 \end{pmatrix} \quad 2.\ 7$$

iv. Constant multiplier (xλ): Let k = xλ, where k={$k_3,k_2,k_1,k_0$} and s={$s_3,s_2,s_1,s_0$} and λ={1100}are elements of GF($2^4$)

The formula for computing multiplication with constant λ is as per (2.8)

$k_3 = s_2 \oplus s_0$
$k_2 = s_3 \oplus s_2 \oplus s_1 \oplus s_0$
$k_1 = s_3$
$k_0 = s_2$ 2. 8

v. Constant multiplier (xφ): Let k= sφ, where k={$k_1,k_0$}, s={$s_1,s_0$} and φ={1 0}are elements of GF($2^2$) The formula for computing the

multiplication with constant φ is as shown in (2.9)

$k_1 = s_1 \oplus s_0$

$k_0 = s_1$ 2. 9

vi. Multiplicative inverse in GF($2^4$): The inversion in GF($2^4$) has to be implemented by different approach.

1. Inversion can be implemented by repeat squaring and multiplying. This is shown in Figure 4(a).

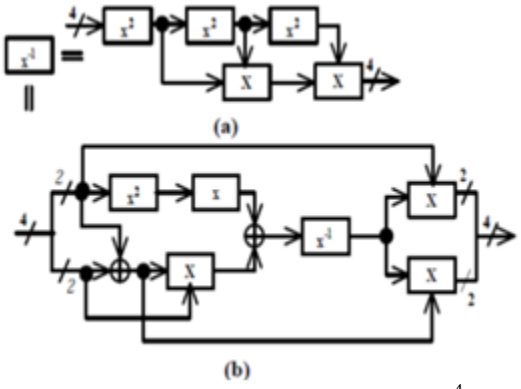


Figure 4: Multiplication inversion in GF($2^4$); (a) Square-multiply approach; (b) Multiple decomposition approach

2. The inversion in GF($2^4$) can be further decomposed by applying formula similar to (2.2) iteratively.Figure 4(b) shows the decomposition of inversion in GF($2^4$) to GF(($2^2$)$^2$). The combination of the squarer in GF($2^2$) and the constant multiplier (xφ) can be implemented by only exchanging the position of the two input bits so that no logic gate is involved. It can be derived from the inversion of $x'=\{x_1', x_0'\} \in GF(2^2)$ is $\{x_1', x_1'+x_0'\}$ .
3. Taking the four bits of $x \in GF(2^4)$ as $\{x_3, x_2, x_1, x_0\}$, it can be derived, each bit in $x^{-1}=\{x_3^{-1}, x_2^{-1}, x_1^{-1}, x_0^{-1}\}$ can be computed by the (2.10)

$x_3^{-1} = x_3+x_3x_2x_1+x_3x_0+x_2$

$x_2^{-1}= x_3x_2x_1+x_3x_2x_0+x_3x_0+x_2+x_2x_1$

$x_1^{-1}=x_3+x_3x_2x_1+x_3x_1x_0+x_2+x_2x_0+x_1$

$x_0^{-1}=x_3x_2x_1+x_3x_2x_0+x_3x_1+x_3x_1x_0+$

$x_3x_0+x_2+x_2x_1+x_2x_1x_0+x_1+x_0$ 2. 10

Composite field decomposition can reduce the hardware complexity significantly when the order of the field involved is large. However for small fields, such as GF($2^4$), further decomposition may not be the optimum approach.

The SubBytes and InvSubBytes transformations are similar apart from affine transformation and its inverse, so only the implementation of the SubBytes operation is designed in this section. The InvSubBytes transformations can be implemented as shown in Figure 5.

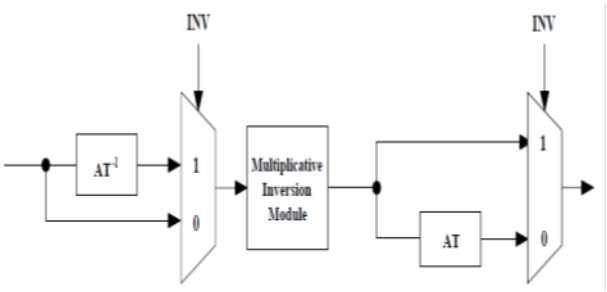


Figure 5: Block diagram of the InvSubBytes Transformation

*Composite Field Arithmetic:*

Composite field arithmetic can be employed to reduce the hardware complexity [13].

Taking isomorphic mapping into consideration not all the transformations in the AES algorithm are suitable to be implemented in the composite field. The constant multiplications in the MixColumns and InvMixColumns transformation are more expensive [10]. The hardware overhead of the mapping of constant can be eliminated by computing the mapping beforehand. The same argument also holds for the constant multiplication used in the InvMixColumns transformation. Therefore it is more efficient to implement the MixColumns/InvMixColumns in the original field GF($2^8$). The ShiftRows and InvShiftRows is a trivial transformation, only cyclical shifting is involved so its implementation does not depend on representation of Galois Field elements. The affine and inverse affine transformation can be combined with the isomorphic and InvIsomorphic transformation. Based on the above observations, it is more efficient to carry out only the multiplicative inversion in the SubBytes and InvSubBytes in the composite field, while keeping the rest of the transformations in the original field GF($2^8$)

*Pipelining architecture:*

To improve the performance and to reduce the area of AES algorithm, three architectural approaches i.e. pipelining, sub pipelining and loop-unrolling can be used in non-feedback modes by duplicating the hardware for implementing it in each round [12]. Here the pipelined architecture is realized and implemented in S-Box by inserting rows of registers

as shown in the Figure 6 to achieve efficient resource utilization

For a k-round pipelined architecture , $\tau = t_{set\ up} + t_{prop}$, where $t_{set\ up} + t_{prop}$ stands for the setup time and propagation delay of a register. Usually $\tau$ is small, so there is almost r time speedup over pipelining at the cost of k(r-1), additional rows of registers and slightly increased area caused by larger control logics.

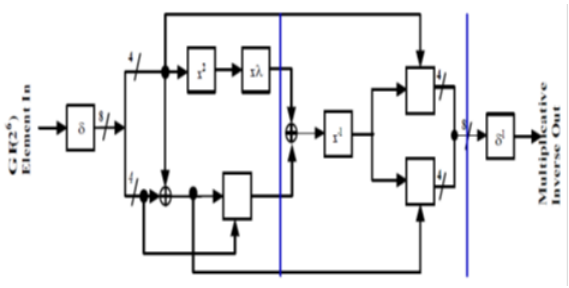


Figure 6: S-Box with two stage pipeline

## III. IMPLEMENTATION OF KEY EXPANSION

The generation of RoundKeys can be done in advance and stored in memory or generated during run time. The fig. 7 shows the architecture of Key Expansion suitable for r pipelined AES algorithm with 128 bit key. .

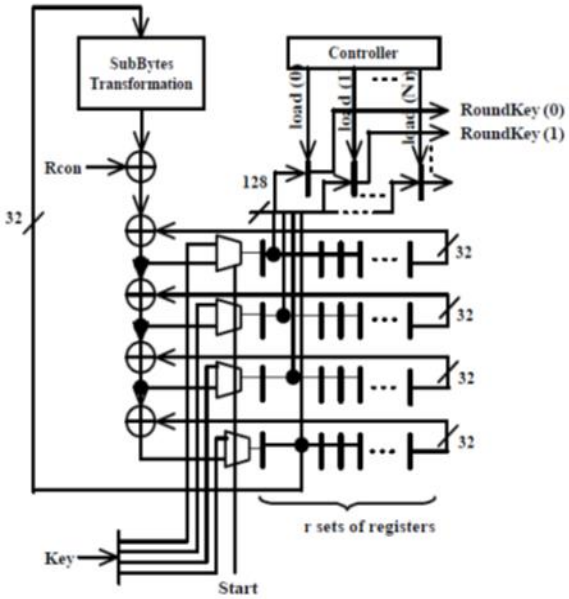


Figure 7: Key expansion architecture for r-pipelined AES algorithm with 128 bit key

In the first case, RoundKeys can be read out from memory using appropriate addresses, and there is no extra delay for decryption but the approach is not suitable for the applications where the key changes constantly. Here the delay of memory access is unbreakable, which may offset the speedup achieved by pipelining in the SubBytes transformation. The key expansion architecture is capable of generating all the Nr+1 RoundKeys after r x Nr clock cycles. Only the top row clock input, which use "load(i)" ($0 \le i \le Nr$) as the "clock" input. At clock cycle r x i, the output of the registers in the first column is the corresponding "RoundKey(i).". The controller is designed so that the signal "load(i)" is initially "0" goes to "1" in clock cycle r x i, and stays at "1" afterwards. The rising edge active registers in the top row load "RoundKey(i)" at clock cycle r x i. After r x Nr clock cycles all the Nr +1 RoundKeys are available at the output of the top row registers and are held there for the entire encryption and decryption process.

The data encryption and key expansion can start simultaneously in the above architecture. RoundKeys are used in reverse order, the decryption process can only start after the last RoundKeys generated. InvMixColumns transformation needs to be performed on the RoundKeys to get the MixRoundKeys.

## IV. IMPLEMENTATION RESULTS

The proposed architecture of AES processor of 128 bit key is coded by Verilog HDL [11] and implemented and synthesized using Xilinx ISE v13.4 and Xilinx Spartan-3E XC3S2500E-4. The two stage pipeline register S-Box simulation result shown in Figure 8. The values are verified and matching against the pre calculated values of S-Box look up table.

Figure 8: Two stage S-Box

Table 1 shows the realization results of the AES algorithm of two stage pipelined S- Box after placement and routing. From the device utilization summary, the number of occupied slices is 3138 out of the available 4656 slices. The maximum delay from the static timing report is 20 ns as shown in Table 2

Table 1: Device Utilization

| Device Utilization Summary | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of external IOBs | 1 | 232 | 1% |
| Number of LOCed external input IBUFs | 1 | 1 | 100% |
| Number of BSCANs | 1 | 1 | 100% |
| Number of BUFGMUXs | 3 | 24 | 12% |
| Number of DCMs | 1 | 4 | 25% |
| Number of RAMB16s | 7 | 20 | 35% |
| Number of slices | 3138 | 4656 | 67% |
| Number of SLICEMs | 146 | 2328 | 6% |

Table 2: Timing report

| Constraint | Check | Worst case slack | Best case available | Timing errors | Timing Score |
|---|---|---|---|---|---|
| PERIOD analysis for net "U_CLK/CLKIN_IBUFG_OUT" PERIOD=20 ns HIGH 40% | SETUP<br><br>HOLD | 0.062 ns<br><br>0.621 ns | 19.938 ns | 0<br><br>0 | 0<br><br>0 |

AES algorithm functionality verified in the FPGA kit with the test setup as shown in Figure 9


Figure 9: AES processor in Spartan-3E

The S-Box two stage pipelined encrypted implementation results for the clock cycle consumption shown in Figure 10 and Figure 11. The data blocks are accepted every clock cycle. The number of cycles is Nr+1 i.e 33 cycles in the initial round and subsequent cycles consume 11 cycles.

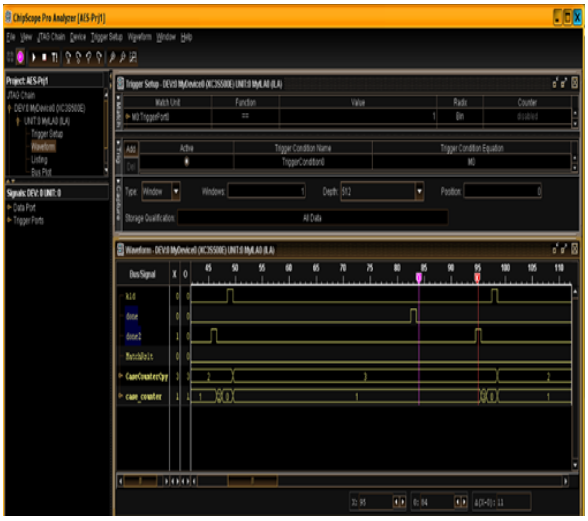

Figure 10: AES processor for first round

Figure 11: AES processor with subsequent round

## V. CONCLUSION

The paper proposed efficient two stage pipelined S-Box in AES algorithm. A combinational logic based S-Box for the SubBytes transformation implemented to avoid unbreakable delay of LUTs in the ROM based design. Composite Field arithmetic is used to reduce the hardware complexity and key expansion implementation presented for resource utilization and performance improvement.

REFERENCES


[1] K J. Daemen and V. Rijmen, The Design of Rijndael: Springer-Verlag New York, Inc., 2002.

[2] S. Tillich, M. Feldhofer, T. Popp, J. Gro, "Area, delay, and power characteristics of standard-cell implementations of the AES S-Box," J. Signal Process. Syst., vol. 50, pp. 251-261, 2008 mm

[3] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," in Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology: Springer-Verlag, 2001. 222

[4] N. Sklavos and O. Koufopavlou, "Architectures and VLSI implementations of the AES-Proposal Rijndael," Computers, IEEE Transactions on, vol. 51, pp. 1454-1459, 2002.

[5] The Advanced Encryption Standard http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[6] E. NC Mui, "Practical implementation of Rijndael S combinational Logic," unpublished.888

[7] "Practical Implementation of Rijndael S-Box using combinational logic" by Edwin NC Mui, Custom R &D Engineer, Texco Enterprise Ptd.Ltd.

[8] "High Speed VLSI Architecture for the AES Algorithm", Xinmiao Zhang, Student member, IEEE, and Keshab K. Parthi, Fellow, IEEE, IEEE transactions on Very Large Scale Integration (VLSI)Systems, Vol 12, No. 9 September 2004

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

[10] J. Wolkerstorfer, E Oswald, and M Lamberger, "An ASIC implementation of the AES SBoxes," In Bart Preneel, editor, Topics in Cryptology RSA 2002,The Cryptographers Track at the RSA Conf. 2002. --- REMOVED from the paper

[11] G. KUMAR and P. MAHESH, "Implementation of AES algorithm using Verilog," International Journal of VLSI and Embedded Systems Vol 04, Article 05090; June 2013.

[12] D. Kenney, "Energy efficiency analysis and implementation of AES on an FPGA," M.S. thesis, Dept. Elect. Eng., Univ. of Waterloo, Canada, 2008

[13] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao and P Rohatgi, "Efficient rijndael encryption implementation with composite field arithmetic," in CHES '01: Proceedings of the Third International Work shop on Cryptographic Hardware and Embedded Systems, 2001.

[14] N. Ahmad, R. Hasan and W.M Jubadi, "Design of AES S combinational logic optimization," in IEEE Symposium on Industrial Electronics & Applications (ISIEA), 2010.