FPGA Implementation of Enhanced Throughput Design of AES Architecture using Nikhilam Sutra

BINDU SWETHA PASULURI, V. J. K. KISHOR SONTI
School of Electronics & Electrical Engineering
Sathyabama Institute of Science & Technology, Chennai- 600119,
INDIA

Abstract: The exponential growth of the internet and contemporary communications users have established safety as a fundamental design feature for encrypted transmission. The Enhanced Cryptography Standard is perhaps the most widely used cryptography information security algorithm standard that has been authorized by NIST. This paper proposes a high-throughput design for the AES Algorithm with huge key sizes. AES would be a block cipher that ensures data security by using key lengths of 128,192 and 256-bits. The design concept focuses on a 256-bit key size classification algorithm since a big key size is required to ensure excellent security. Additionally, simultaneous key expansion & encryption/decryption processes would be pipelined to maximize speed. Parallelization of a key expansion module's sub-processes would be used to reduce the critical chain latency. The S-box comprising sub-byte & inverse sub-byte operations has been developed with compound field arithmetic operations to reduce time and area further. The work Increased throughput by 50%, area reduced by 34.32 %, and latency by 20% compared to the old approach with modified nikhilam sutra. Additionally, integrated AES encryption/decryption is planned and implemented on the FPGA Zed board utilizing Verilog HDL in Xilinx Vivado.

Key-words: - Advanced encryption standard, throughput, FPGA, high security, nikhilam sutra

Received: September 17, 2021. Revised: August 18, 2022. Accepted: September 21, 2022. Published: October 31, 2022.

1 Introduction

The need to document every meaningful occurrence in one's everyday life became one of the key motives. Messages from uninvited persons should be treated with caution. Encryption has become one of the security components used to safeguard data accessible to the public. Cryptography is a technique of communication security in which data representations are transformed from one configuration to the other to cover up and safeguard them [1], [5], [6], [7], [8], [9], [10].

Cryptography's value continues to expand in lockstep well with the volume of sensitive data exchanged across open networks. The more data delivered in a format understandable by a personal computer, the more exposed we become too automated espionage. Cryptography is critical in both resistant and certifiable applications.

On January 2, 1997, the National Institute of Standards and Technology (NIST) accepted suggestions for innovative Advanced Encryption Standard (AES) computations. The goal was to supersede the older Data Cryptography Standard, published in November 1976 in response to the disclosure of DES's vulnerability, [10]. Rijndael was selected and called the AES Measurement on November 26, 2001, following two rounds of evaluation. The Rijndael would be a block cypher that operates on a set string of bits. Joan Daemen & Vincent Rijmen, two of the company's Belgian founders, inspired the name. AES would be used in several sectors, such as database servers, ATMs, mobile networking, and optical video recorders. AES is a cryptographic algorithm used in both hardware and software components. On either hand, implementation is better suited to high-speed real-world applications, [11].

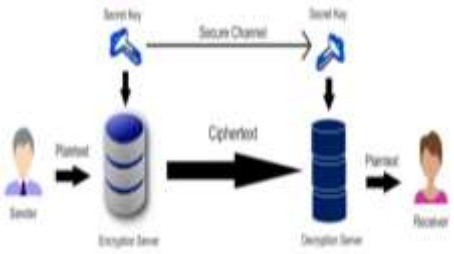


Fig. 1: The fundamental concept of the AES algorithm.

AES has 3 different round procedures. A specific input block, typically 128 bits in size, is required, as is a comparable yield of the same size. The conversion process necessitates adding a second kind of information, which is the key. The cypher determines the key length, and AES supports three alternative key sizes: 128, 192, and 256 bits. Each of the three primary sizes has been determined to be acceptable for Federal Government various applications from classified to top secret. TABLE I summarizes the round count for three separate AES implementations. Nevertheless, the final round key for each iteration is 128 bits. The first-round key is initialized by entering 128 bits of text into the first-round key, [12],[13], [19]. The key generation function multiplies the value of the input key to create a round key per round. The AES method is based on a single 4x4 byte cluster known as a state. The status changes four times during encryption: Add Round Key, Substitution Byte (sub-byte), Shift Row, and MixColumn (a combination of these four

Table 1. In AES, the width of the round key and the number of iterations

Cipher Key size	Number of Rounds (Nr)	Round Key size
256bits	14	128bits
192bits	12	128bits
128bits	10	128bits

Operations are referred to as a round); and for decryption, the state undergoes four changes: Add Round Key, converse sub-byte, transposed Shift rows, and Inverse Mix column. The AddRoundKey technique combines a bitwise XOR operation upon that current state and the Key Expansion Function's Round Key. Encryption or decryption may be performed in our integrated architecture by changing the multiplexer option to 0 or 1.

There are two ways to do sub-byte substitutions: I via a ROM table and (ii) through CFA. Substitution is the costliest and most time-consuming mode of operation. As a result, hardware optimization in VLSI implementation would be crucial for reducing the AES architecture's area and power consumption. The ROM-based method consumes a huge memory space and introduces substantial latency due to ROM access time. As a consequence, composite field arithmetic facilitates S-box completion. Cryptography's practical usage for encryption is contingent upon properly managing cryptographic keys. Greater is considered to be superior. Because they can, and because it is available, top-secret military applications may demand a key length of 256 bits. Consequently, this research emphasizes the need to provide strong protection on a large key scale.

The remainder of this paper is organized as follows: The AES method and composite field arithmetic needed to implement the S-box was discussed in Section II. Section III details the S-modified box's structure. Sections IV&V contain the findings of the FPGA achievement, as well as comparisons to other new S-box techniques. Section VI concluded.

2 Integrated Encryption/Decryption Architecture

Figure 1 illustrates the Efficient Encryption/Decryption Recommended AES round model interactively. The design of each module is discussed in detail in the following sections, as are the approaches for maximizing throughput. Every byte therein state sequence is replaced by another byte acquired using a multiplicative inverse using GF (28) and a sub-byte translation using an affine transformation.

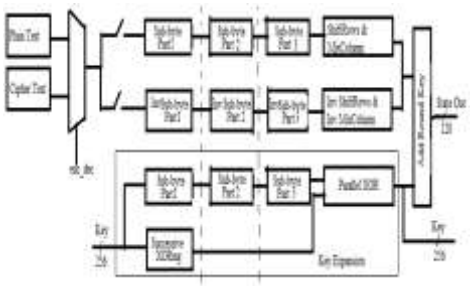


Fig. 2: Hybrid Encryption/Decryption Round Topology with Sub-pipelines

The inverse affine transformation is usually performed in the same way that its opponent, the Inverse Sub-Byte transformation, has been performed.

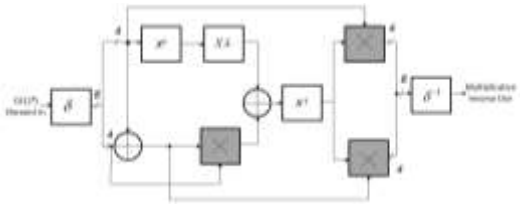


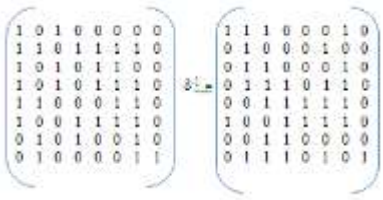
Fig. 3: Block diagram for determining the S-Multiplicative box's inverse.

Isomorphic and inverse isomorphic mapping

The reverse multiplicative approach is obtained by decomposing the much more complicated GF (28) into the lower order fields GF (21), GF (22), and GF ((22)2). This would be done by the use of the irreducible polynomials listed below:

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

The multiplicative inverse formula could be clearly associated with a variable that really is GF dependent within structural fields (28). The part must always be isomorphic ally related to its own composite field descriptions. Likewise, every result must always be determined by its hybrid field following the multiplicative inverse. -1 may be represented mathematically as an eight-by-eight matrix. Let q become the vector from GF (28) and explain the isomorphic and inverse relations. As δ*q and δ-1*q.



Multiplication of every input byte to an isomorphic from the most important to the least significant bit and the multiplicative inverse outputs to a reverse isomorphic provides the expression which may be logically realized as:

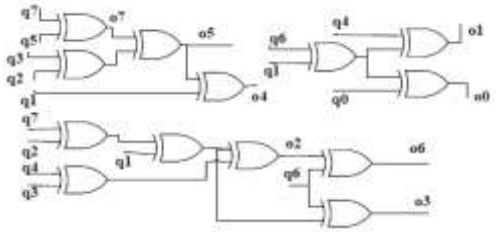


Fig. 4: Logical Implementation of δ*q.

Squaring:

The very first nibble from the output of the isomorphic algorithm is square into itself. The following operations have been involved:

Assume $k = q^2$, in which GF (2^4) k and q are elements which can be expressed in Boolean as $\{k_3, k_2, k_1, k_0\}$ and $\{q_3, q_2, q_1, q_0\}$.

$$k = k_H x+k_L = (q_H x+q_L)^2 \quad (2.2)$$

The above expression is in the form of $(a+b)^2$ and thus can be expanded as $a^2+2ab+b^2$.

$$\therefore k = q_H^2x^2+q_Hxq_L+q_Hxq_L+q_L^2 = q_H^2x^2+q_L^2$$

x^2 can be reduced by an irreducible polynomial $x^2+x+\varphi$, that yields

$$k = q_H^2(x+\varphi) + q_L^2 \Rightarrow q_H^2x+ (q_L^2+ q_H^2\varphi)$$
$$k_H= (q_3x+q_2)^2 \Rightarrow k_H=q_3x^2+q_2$$

which is in the form of bx+c. The last term has now been reduced to GF (22). Further decomposing kH and kL to GF (2) yields:

The irreducible polynomial may be used to decrease this. x^2+x+1

Similarly $k_H=k_3x+k_2= q_3x+(q_2+q_3) \in GF(2)$ $k_L=$

From the above equations, we obtain

$K^3=q^3$

$K^2=q^2 \oplus q^3$

$K^1= q^2 \oplus q^1$

$K^0= q^3 \oplus q^1 \oplus q^0$

From the above equations, the logical implementation for squaring is

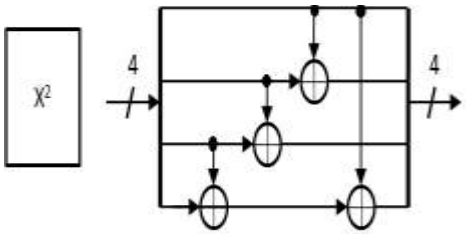


Fig. 5: In GF, there is a logical structure for squaring (24).

Multiplication of a nibble with constant λ:

Assume k=q **λ,** k and **λ**= {1100} are components in GF (2^4).

$k= (q_3, q_2, q_1, q_0)$ (1100). The first 2 bits are considered as q_H and lower 2-bits are considered as q_L.

$\therefore k=q_H\lambda_Hx^2+q_L\lambda_Hx$ (λ_L is cancelled as its value is 00)

Above expression can be reduced by substituting $x^2=x+\varphi$

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

By separating the higher and lower terms, we get

$k_H= q_3x^2+(q_3+ q_2) x+q_2+q_1x^2+(q_1+q_0) x+ q_0$

Substituting $x^2=x+1$, gives

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

similarly

$k_3x+k_2= (q_2+q_0) x+(q_3+ q_2+ q_1+ q_0)$

and for the lower terms

$k_1x+k_0= (q_3) x+ q_2$

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

This can be implemented as

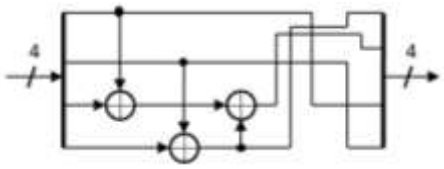


Fig. 6: Multiplication using Constants Logic Representation.

GF(2^4) Multiplication

Assume k=qw, q and w are components in GF(2^4).

$k=(q_3q_2q_1q_0)(w_3w_2w_1w_0)=(q_Hx+q_L)(w_Hx+w_L)$.

By expanding the equation and substituting $x^2=x+\varphi$, we get

$k=(q_Hw_H)(x+\varphi)+(q_Hw_L+q_Lw_H)x+q_Lw_L$

This can be logically implemented as in figure 7, and the multiplication in GF(2^2) can be implemented as in figure7.

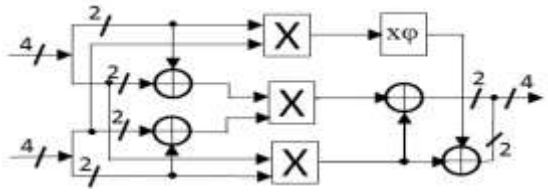


Fig. 7: Logical implementation of multiplication in GF (24).

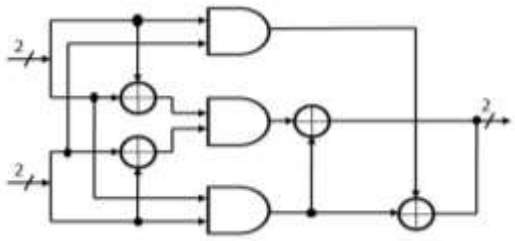


Fig. 8: Completion of multiplication in GF in a logical manner (2).

Multiplication with constant φ

Assume $k=q\varphi$, q and $\varphi=\{10\}$ are components in GF(2^2).

$k=(q_1x+q_0)(10)=q_1x^2+q_0x$

By replacing $x^2=x+1$, we can get $k_1=q_1\oplus q_0$, $k_0=q_0$that can be logically implemented as

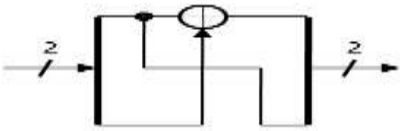


Fig. 9: Multiplication Logical Implementation as φ.

The multiplicative inverse in GF(2^4)

Inverse for the independent bits [1], [2], [3], [4], [5] can be performed as follows:

$q_3^{-1}=q_3\oplus q_3q_2q_1\oplus q_3q_0\oplus q_2$

$q_2^{-1}=q_3q_2q_1\oplus q_3q_2q_0\oplus q_3q_0\oplus q_2\oplus q_1q_2$

$q_1^{-1}=q_3\oplus q_3q_2q_1\oplus q_3q_0\oplus q_2\oplus q_2q_0\oplus q_1$

$q_0^{-1}=q_3q_2q_1\oplus q_3q_2q_0\oplus q_3q_1\oplus q_3q_1q_0\oplus q_3q_0\oplus q_3q_0q_0\oplus q_3q_0\oplus q_2\oplus q_2q_1\oplus q_2q_1q_0\oplus q_1\oplus q_0$

Equations can be logically implementable as

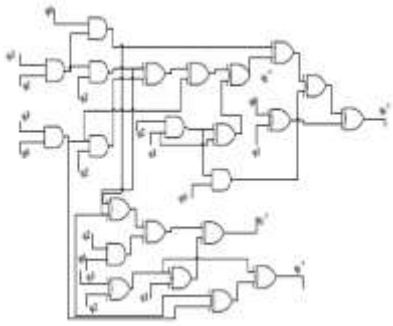


Fig. 10: Logical implementation for the multiplicative inverse.

Each output byte of the multiplicative inverse is multiplied by an affine transformation, and in inverse sub-byte transformation, every input byte is multiplied by an inverse affine transformation.

3 Shift Row and Inverse Transformation of Shift row

The 128-bit display of the s-box is laid out in a 4x4 matrix. The rows are moved to the left by x number of bytes when x is the row number. This is not a really good decision.

The first row has indeed been relocated 0 places to the left. The second row has indeed been shifted to the left by one position. The third row has already been shifted to the left by two places. The fourth row was being pushed to the left by three spaces.

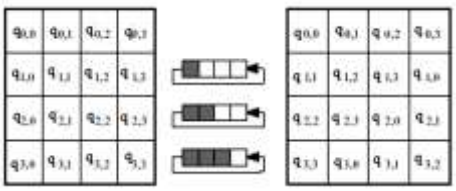


Fig. 11: Shift Row Transformation.

Inverse shift rows take the output byte of the inverse sub-byte as reference. Each row of the state is circularly moved to the right, based on the row index, in this process. Following are the steps:

The 0 positions had been shifted to the right in the first row. The second row has also been relocated one space to the right. The third row has indeed been moved to the right by two spaces. The fourth row is being pushed to the right by three spaces.

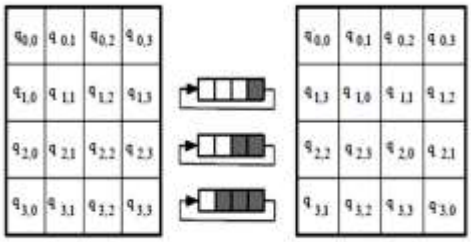


Fig. 12: Inverse Shift Row Transformation.

4 Transformation of Mixed Columns and Inverse Mixed Columns

Within Mix Columns and Inverse Mix Columns transformations, every column byte becomes transformed to the correct value, a function among all four bytes across the column. Every transformation involves multiplying every byte of the output shifting rows or reverse shift rows by a predetermined matrix, which would then be sorted as a state, [19].

5 Vedic Multiplier

The Vedic multiplier's work focuses on Vedic multiplication equations (Sutras). Several Vedic sutras have traditionally is often used to multiply 2 decimal integers. This paper extends the same concepts to the binary system to develop a very good solution for digital electronics [12], [13]. This paper provides an overview and application of the notion of high-speed multiplier designs. The multiplier concept has been extensively researched in the Nikhilam and U T sutras.

Nikhilam Sutra

Furthermore, Nikhilam Sutra signifies 'all nine and the final ten.' Even though it is valid across all multiplication situations when the numbers are the same, it's much more efficient. Because it regulates the addition of a huge number to execute a multiplication operation out of its neighboring base, the beginning number is more straightforward, and the multiplication difficulty is lower than in Figure 2. Using the multiplication of two decimal integers (96 * 93), we will first explain this Sutra, in which the selected base is 100, which would be more robust and more extensive than the two. Figure 13 depicts the Nikhilam sutra.

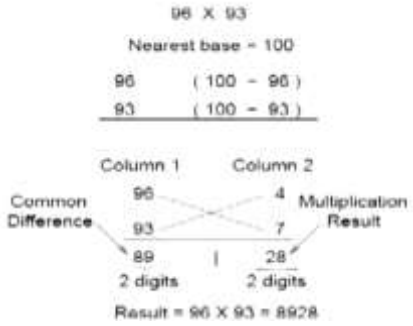


Fig. 13: Illustration of Nikhilam Sutra.

Proposed Modified Nikhilam Multiplier

Multiplication is crucial among the several operations employed in creating the AES algorithm since it requires more resources to function correctly. The multiplier's efficiency is dependent mainly on the adders used in the design to calculate the final number. A new multiplication unit based on the KSA and Nikhilam Vedic sutra multipliers is suggested to address earlier shortcomings. This multiplication offers the benefits of using less power, less latency, and taking up less space.

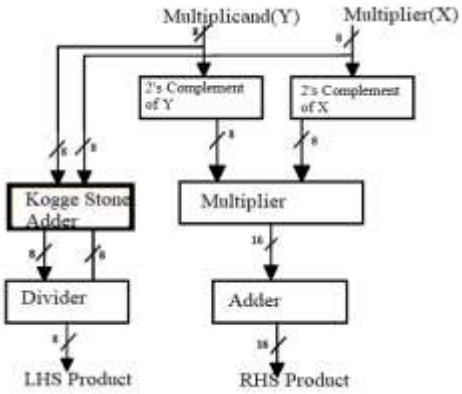


Fig. 14: Modified Nikhilam Vedic multiplier.

The Nikhilam Vedic multiplier is applied to two binary integers. The CSA has been substituted with a KSA high-speed adder in this manner. This method allows for successful implementation by multiplying one bit by another, calculating the partial product total, and providing output [14], [15]. Figure 5 depicts the block design for the proposed Vedic multiplier, which employs the KSA adder instead of the RCA or CLA. KSA has more incredible benefits than CSA since it permits high-speed processing and lowers propagation delays. As a result, utilizing KSA [15], the result of the Nikhilam multiplier may be achieved in less time. [16], [17], [18].

The proposed Nikhilam multiplier is implemented by replacing the existing conventional multiplier.

6 Results

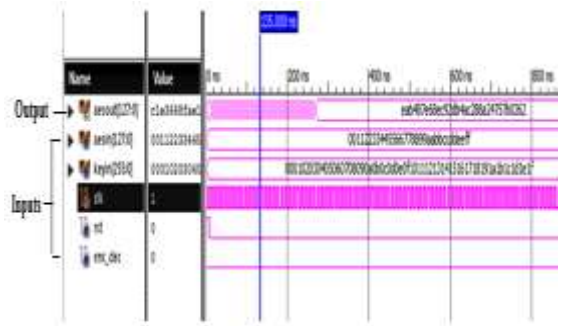


Fig. 15: Simulation Result for Encryption.

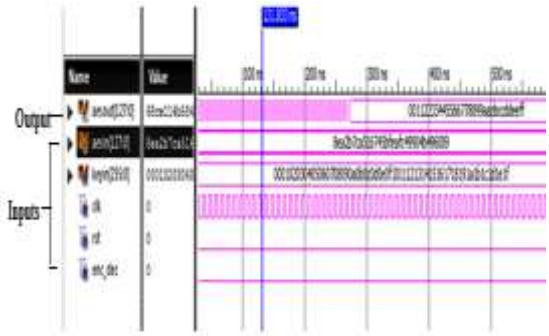


Fig. 16: Simulation Result for Decryption.

For a '00112233445566778899aabbccddeeff' (128-bit in hexadecimal) and 256-bit key' 000102030405060708090a0b0c0d0e0f10111213 141516 1718191a1b1c1d1e1f', enc- dec as 1,rst as 0 and clock inputs the encrypted data of 128-bit 'ea2b7ca516745bfeafc49904b496089' is generated.

By selecting the multiplexer selector as 1, the encrypted data 'ea2b7ca516749bf eafc 499 04b496089' with256-bitkey 000102030405060708090a0b0c0d0e0f10111213 141516 1718191a1b1c1d1e1f'data is decrypted back to '001122334455667788 99 abb ccddeeff'.

From table 2, it is evident that the proposed method showed better improvement in terms of area, latency and throughput. Some relevant studies can be found in [19], [20], [21], [22].

Table 2. Performance Comparison

Parameters	Existing with the conventional multiplier	Proposed one	% Improvement
Throughput (Gbps)	23	35	50
Slices	3289	2120	34
Latency	0.5	0.41	20

7 Conclusion

Internal pipelining for the composite sector S-box is utilized to construct an improved encryption-decryption architecture for the Advanced Encryption Protocol algorithm. This pipelining enabled the processing of state array columns simultaneously and S-box communication between the main round unit and the expansion key unit. Furthermore, this design is employed with on-the-fly production of all-round keys, which minimizes the need for an ample space to retain all of the keys while also cancelling the extra delay caused by pre-calculation and storage for all-round keys. The new design outperformed earlier s-box systems in terms of throughput. The usage of 256-key size offers the highest degree of security, which is employed in top-secret military applications. Compared to the old technique, we improved throughput by 50%, slices by 34%, and latency by 20%. The design is created using the Xilinx Vivado tool and implemented on the Xilinx FPGA Zynq board.

References:

[1] I. J. Bahram Rashidi, Implementation of An Optimized and Pipelined Combinational Logic RijndaelS-Box on FPGA, Computer *Network and information security*, Vol.1, 2013, pp. 41-48, DOI: 10.5815/ijcnis.2013.01.05.

[2] Salma Hesham, Mohamed A. Abd ElGhany, and Klaus Hofmann, High Throughput Architecture for the Advanced Encryption Standard Algorithm, Electronics Department, German University in Cairo, Egypt Electronics Department, German University in Cairo, Egypt, *Integrated Electronic Systems Lab, TU Darmstadt, Germany, IEEE Integrated Electronic Systems Lab, TU Darmstadt, Germany,* 2014.

[3] TanzilurRahman, Shengyi Pan, and Qi Zhang, Design of a High Throughput 128-bit AES (Rijndael Block Cipher), *International multiconference of engineers and computer scientists*, 2010.

[4] Shen-Fu Hsiao, Ming-Chih Chen, and Chia-Shin Tu Memory-Free Low-Cost Designs of Advanced Encryption Standard Using Common Sub Expression Elimination for Subfunctions in Transformations, *IEEE transactions on circuits and systems—i: regular papers,* Vol.53, No. 3, 2006.

[5] Xinmiao Zhang, Student Member, *IEEE,* and Keshab K. Parhi, Fellow, IEEE High-Speed VLSI Architectures for the AES Algorithm,

IEEE transactions on very large-scale integration (VLSI) systems, Vol.12, No.9, 2004.

[6] Naga M. Kosaraju University of South Florida, MuraliVaranasi, Saraju P. Mohanty, University of North Texas, Denton TX 76203, *A High-Performance VLSI Architecture for Advanced Encryption Standard (AES) Algorithm*.

[7] Hardware Implementation of High-Performance AES Using MinimalResources, *International Journal of Engineering Research,* Vol.3, No.2, pp. 68-72.

[8] Xinmiao Zhang, Student Member, and Keshab K. Parhi, Fellow, High-Speed VLSI Architectures for the AES Algorithm, *IEEE transactions on very large-scale integration (VLSI) systems,* Vol.12, No.9, 2004.

[9] Purnima Gehlot, S. R. Biradar, and B. P. Singh MITS University, Implementation of modified Two fish Algorithm using 128 and 192-bit keys on VHDL, *International Journal of Computer Applications,* Vol.70, No.13, 2013.

[10] S. Karthik, and A. Muruganandam, Research Scholar, Periyar University, Salem, Tamilnadu, India, Data Encryption and Decryption by Using Triple-DES and performance Analysis of Crypto System, *International Journal of Scientific Engineering and Research (IJSER) ISSN (Online),* 2014, pp. 2347-3878.

[11] Snehal Wankhade, and Rashmi Mahajan Dynamic Partial Reconfiguration Implementation of AES Algorithm, *International Journal of computer applications*, Vol.97, No.3, 2014.

[12] H. D. Tiwari, G. Gankhuyag, C. M. Kim, and Y. B. Cho, Multiplier design based on ancient Indian Vedic mathematics, *Proc. Int SoC Design Conf,* pp. 65-68. 2008.

[13] S. Patil, Design of speed and power-efficient multipliers using Vedic Mathematics with VLSI implementation, *IEEE*, 2014.

[14] M. Akila, C. Gowribala, and S. M. Shaby, Implementation of high-speed Vedic multiplier using modified adder, *IEEE Conference on Communication and signal processing (ICCSP),* 2016, pp. 2244 -2248.

[15] N. Singh, and M. Singh, Design and Implementation of 16 X 16 High-speed Vedic multiplier using Brent Kung adder, *International Journal of Science and Research (IJSR),* Vol.5, No.12, 2016, pp. 239-242.

[16] B. S. Pasuluri, V. J. K. Kishor Sonti, Performance Analysis of 8-Bit Vedic Multipliers Using HDL Programming, ICDSMLA 2019, *Lecture Notes in Electrical Engineering, Springer, Singapore,* Vol.601, 2020.

[17] R. Nikhil Mistri, S. B. Somani, and Dr. V. V. Shete, Design & Comparison of Multiplier using Vedic Mathematics, *Proceedings of IEEE.*

[18] Bindu Swetha Pasuluri, and V. J. K. Kishor Sonti, "Design of Vedic multiplier-based FIR filter for signal processing applications, *J Phys: Conf. Ser,* 2021.

[19] B. S. Pasuluri, and V. J. K. Kishor, Application of UT multiplier in AES algorithm and analysis of its performance, *Information Technology in Industry ITI*, Vol.9, No.3, 2021, pp. 647-652.

[20] B. Pasuluri, V. J. K. Kishor Sonti, S. M. M. Trinath, and N. Bala Dastagiri, Design of CMOS 6T and 8T SRAM for memory Applications, *Proceedings of Second International Conference on Smart Energy and communication. Algorithms for Intelligent Systems Springer Singapore,* 2021, doi.org/10.1007/978-981-15-6707-0_44.

[21] Bindu swetha Pasuluri, and V. J. K. Kishor Sonti, Design and Analysis of Instrumentation amplifier using 45nm technology, *in Informatica Journal*, Vol.32, No. 11, 2021.

[22] Suganthi Venkatachalam, and Seok –Bumko, Design of Power and Area Efficient approximate multipliers, *IEEE Transactions on VLSI Systems,* Vol.25, No. 5, 2017.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0
https://creativecommons.org/licenses/by/4.0/deed.en_US