Journal on Electronic and Automation Engineering

Vol: 4(2), June 2025

REST Publisher; ISSN: 2583-6951 (Online)

Website: https://restpublisher.com/journals/jeae/

DOI: https://doi.org/10.46632/jeae/4/2/30

Virtex-7 FPGA Realization of AES-256 with Enhanced SubBytes and Key Schedule Performance

***Anthala Mithranand, K. Chandra Sekhar**

Sri Sai Institute of Technology and Science, Rayachoty, Andhra Pradesh, India.

**Corresponding Author Email: mithranand77@gmail.com*

Abstract: *With the rising need for secure and efficient cryptographic solutions, especially in embedded and real-time systems, hardware-level enhancements have become increasingly important. This study presents a refined version of the AES-256 encryption algorithm, specifically adapted for resource-constrained environments where performance and energy efficiency are essential. By transitioning from the traditional 128-bit block structure to a 32-bit block framework, the design reduces computational demands while preserving the strong security features of AES-256. This modification supports a more modular and compact implementation, making it ideal for applications with limited hardware capabilities. Key optimizations include reusing S-box components and simplifying key table operations, which significantly lower logical complexity. Additionally, the use of pipelining improves data processing speeds while keeping latency under control. These design choices make the approach especially suitable for secure Internet of Things (IoT) devices, mobile gadgets, and compact communication modules. When implemented on the Xilinx Virtex-7 FPGA, the architecture demonstrates notable reductions in power usage and hardware resource consumption. This includes substantial savings in slice registers, LUTs, and LUT-FF pairs, along with a measurable increase in data throughput. the optimized AES-256 model offers a scalable and energy-conscious solution for modern cryptographic hardware implementations.*

Key words: *FPGA (field programmable gate array), LUT (look up table), Mbps (megabits per second), sub (sub bytes), AES (Advanced Encryption Standard), Add (add round key), Mix (mix column), and Shift (shift rows).*

1. INTRODUCTION

Cryptography is the process of converting intelligible plaintext into unintelligible ciphertext and vice versa, ensuring data confidentiality and security. This conversion process is essential in protecting sensitive information across various applications, from online banking and e-commerce to satellite communications and digital image processing. Cryptography can be classified into three primary types: public key encryption, hash functions, and symmetric key encryption. Public key encryption uses two keys: one for encryption and another for decryption, while hash functions are primarily used for data integrity. Symmetric key encryption, on the other hand, employs a single key for both encryption and decryption, making it simpler and faster compared to other methods. Symmetric encryption algorithms, including the Advanced Encryption Standard (AES) and the older Data Encryption Standard (DES), are widely used due to their speed, ease of use, and low resource requirements.

AES is one of the most popular symmetric key encryption algorithms used globally, especially for encrypting sensitive data in secure communications and storage. AES supports key sizes of 128, 192, and 256 bits, with the 256-bit AES being the most secure and commonly used variant. While AES is highly efficient in software, optimizing its hardware implementation, particularly for FPGA (Field Programmable Gate Array) devices, can further enhance its performance, power efficiency, and area utilization. FPGAs are increasingly used for cryptographic applications due to their ability to offer parallel processing, flexibility in hardware design, and high throughput.

This article focuses on the implementation of a 256-bit AES algorithm that optimizes for area, power, and performance. Optimizing these metrics is crucial for applications that require efficient, real-time encryption and decryption, such as secure communications and digital transactions. The main challenge with hardware implementation of AES lies in the trade-offs between power consumption, processing speed, and area (hardware space). Traditional AES implementations often consume substantial power and require a significant amount of FPGA resources, such as slice registers, Look-Up

Tables (LUTs), and Flip-Flops (FFs). To address this, the proposed implementation in this study employs several innovative techniques to reduce power consumption and optimize hardware utilization.

In the proposed implementation, a combination of pipelining and hardware reuse techniques is used. The algorithm's operations are optimized for FPGA devices by reusing hardware components such as the S-box, which is an essential element of the AES algorithm. By reusing the same hardware for both encryption and decryption, the implementation reduces the need for additional logic, significantly saving area and power. Additionally, the AES algorithm is modified to work with 32-bit blocks rather than the traditional 128-bit blocks, further optimizing the design for power and area efficiency. This modification allows for a more efficient use of hardware resources, as it reduces the number of operations required per cycle.

The results of this optimization are impressive. The 256-bit AES algorithm implemented on the FPGA demonstrates a 78% reduction in power consumption compared to traditional implementations. Additionally, area usage is reduced by 72% in slice registers, 62% in slice LUTs, and 61% in LUT-FF pairs. The use of pipelining and hardware reuse enables the design to operate with improved throughput, with a 10% increase in data throughput (Mbps) compared to standard implementations. These improvements in power, area, and performance make the proposed implementation highly efficient, making it suitable for a wide range of cryptographic applications, including those requiring real-time encryption in resource-constrained environments.

The significance of optimizing AES-256 on FPGA platforms extends beyond just improved performance metrics—it also enhances the feasibility of deploying advanced encryption techniques in edge devices and Internet of Things (IoT) environments. These environments often operate under strict power and space limitations, making traditional cryptographic implementations impractical. By achieving a smaller hardware footprint and significantly lowering power usage, the optimized AES implementation supports longer device lifespans, reduced heat generation, and better scalability across multiple devices and systems.

Furthermore, this approach lays the foundation for integrating cryptographic modules into multi-function System-on-Chip (SoC) architectures, where hardware resources are shared among various tasks. The ability to reuse components like the S-box in both encryption and decryption processes ensures minimal redundancy, a key consideration in embedded system design. The 32-bit block configuration also introduces flexibility, as it can be adapted more easily to stream-based data applications without requiring extensive buffering or resource-heavy processing pipelines.

From a security perspective, the implementation still retains the robust encryption strength of AES-256, providing high resistance against brute-force attacks and maintaining compliance with modern security standards. As security concerns continue to grow across digital platforms, the development of lightweight, energy-efficient cryptographic solutions like this one will be essential for safeguarding data integrity and privacy in the next generation of secure computing systems.

2. ARCHITECTURE OF THE AES ALGORITHM

The architecture of the AES (Advanced Encryption Standard) algorithm is designed to efficiently perform data encryption and decryption operations using a symmetric key. AES operates on a fixed block size of 128 bits, and it supports key lengths of 128, 192, or 256 bits. The algorithm consists of multiple rounds, with the number of rounds depending on the key size: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. Each round involves a series of operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey.

SubBytes performs a substitution of bytes using a predefined S-box, which ensures non-linearity in the encryption process. ShiftRows shifts the rows of the state array to introduce diffusion, which helps spread the data across the entire block. MixColumns mixes the columns of the state matrix to further increase diffusion, while AddRoundKey combines the current state with a round key generated from the original encryption key.

The AES architecture can be implemented using hardware or software, but hardware implementations (e.g., on FPGAs or ASICs) are preferred for their speed and efficiency in cryptographic applications. The architecture can be pipelined to perform multiple operations simultaneously, improving throughput and reducing latency. It also benefits from the reuse of hardware components, such as the S-box, to optimize power consumption and area usage. the AES architecture ensures secure encryption while being efficient enough for use in modern communication and storage systems.

A. Architecture of The AES Algorithm:

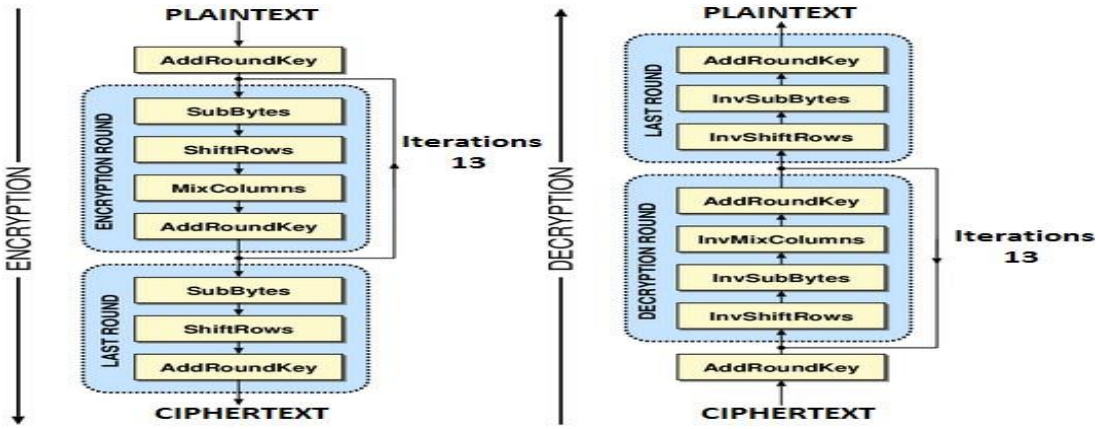


FIGURE 1. Architecture of 256 The AES algorithm

Figure 1 illustrates the architecture of the AES-256 (Advanced Encryption Standard with a 256-bit key length) algorithm, highlighting both the encryption and decryption processes. The encryption phase begins with the addition of the initial round key to the plaintext. This is followed by 13 main rounds, each consisting of four core operations: SubBytes (a non-linear substitution step), ShiftRows (a transposition step), MixColumns (a mixing operation providing diffusion), and AddRoundKey (where a round-specific key is added). The final round omits the MixColumns operation. On the decryption side, the process mirrors encryption but uses the inverse operations in reverse order: InvSubBytes, InvShiftRows, InvMixColumns, and AddRoundKey. The decryption also consists of 13 iterations, with an initial round that applies the round key, followed by rounds that undo the transformations of the encryption process. This symmetric structure ensures that the ciphertext can be accurately reverted to the original plaintext using the same key, ensuring data confidentiality and integrity in AES-256 encryption.

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

FIGURE 2. Data structure of a 128-bit message

Figure 2 shows the data structure of a 128-bit message used in the AES (Advanced Encryption Standard) algorithm. This structure is organized into a 4x4 matrix format, where each cell represents one byte (8 bits), making up a total of 16 bytes or 128 bits. The matrix is filled column-wise, starting from the top-left element A0A_0A0 and continuing down each column. For example, the first column consists of A0,A1,A2,A3A_0, A_1, A_2, A_3A0 ,A1,A2,A3, followed by the second column A4,A5,A6,A7A_4, A_5, A_6, A_7A4,A5,A6,A7, and so on. This column-major order is essential for AES operations such as ShiftRows and MixColumns, which depend on the specific placement of bytes within the matrix. This structured format allows for consistent transformations during encryption and decryption, enabling efficient and secure processing of data blocks. The matrix representation plays a critical role in maintaining the integrity and diffusion of the plaintext throughout the encryption process..

3. IMPLEMENTING THE AES ALGORITHM

This paper presents the implementation of both the traditional and a newly proposed architecture for the 256-bit AES algorithm, targeting FPGA-based systems. It offers a detailed comparative analysis focusing on key design metrics such as power consumption, processing performance, and hardware area utilization. The traditional architecture is evaluated against the proposed design, which incorporates pipelining, hardware reuse, and a modified 32-bit block structure to enhance efficiency. The results demonstrate significant improvements in power savings,

reduced hardware resource usage, and increased data throughput, validating the effectiveness of the proposed approach for real-time, resource-constrained cryptographic applications.

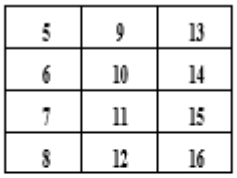
The standard implementation of the 256-bit AES encryption algorithm: The AES (Advanced Encryption Standard) algorithm is a symmetric key encryption technique widely adopted for secure data transmission. In the AES-256 variant, the encryption process consists of 14 rounds, each executing a specific sequence of cryptographic transformations. These transformations ensure high levels of confusion and diffusion, which are critical for secure encryption. Each of these 14 rounds follows a well-defined structure, with certain operations repeated throughout the encryption cycle to provide layered security.

As depicted in Figure 3, the AES encryption process begins with an initial round (round 0), where only the **Add Round Key** operation is performed. This step involves combining the plaintext input with the first round key using a bitwise XOR operation. This initial transformation is crucial as it integrates the cryptographic key into the data before any further processing, establishing the foundation for the encryption sequence.

Following the initial round, **rounds 1 through 13** each consist of four key operations executed in a fixed order: **Sub Bytes**, **Shift Rows**, **Mix Columns**, and **Add Round Key**. The **Sub Bytes** operation is a non-linear substitution step that replaces each byte in the state matrix with its corresponding value from a predefined substitution box (S-box). This substitution introduces non-linearity and helps obscure the relationship between the ciphertext and the original plaintext. Next, the **Shift Rows** operation cyclically shifts the rows of the matrix to the left by varying offsets, effectively disrupting the byte positions and contributing to diffusion. The third operation, **Mix Columns**, transforms each column of the matrix using a linear mixing function. This operation blends the bytes within a column, further enhancing the diffusion of input data. Finally, the **Add Round Key** step incorporates a unique round key into the transformed state, reinforcing key-dependence at each stage.

The final round (round 14) differs slightly from the previous rounds. It omits the **Mix Columns** step and only includes **Sub Bytes**, **Shift Rows**, and **Add Round Key** operations. These final transformations complete the encryption process, yielding the final ciphertext output. The exclusion of Mix Columns in the last round ensures compatibility with decryption and avoids unnecessary transformations that would not significantly enhance security.

Importantly, each operation within a round is executed in a separate clock cycle. This design approach allows hardware implementations of AES to reuse the same functional blocks for multiple rounds, optimizing both area and power consumption. Since only one operation is active at a time per clock cycle, the same circuitry for Sub Bytes, Shift Rows, Mix Columns, and Add Round Key can be reused across all 14 rounds. This serialized operation model ensures that the output of one round seamlessly becomes the input of the next, maintaining the algorithm's sequential nature. the AES-256 encryption algorithm performs 14 rounds of structured transformations on a 128-bit data block, employing a combination of substitution, permutation, and key addition operations. The reuse of hardware across clock cycles and rounds, as highlighted in Figure 3, enables an efficient implementation while ensuring a high level of cryptographic security. This architectural strategy is fundamental to the algorithm's strength and widespread adoption in modern cryptographic systems.



Round 0	01 cycle
Round 1 to 13	52 cycles
Round 14	03 cycles
Total	56 cycles

FIGURE 3. Structure of regular execution

Figure 3 illustrates the architecture of a standard implementation of the AES algorithm using a 256-bit key.

TABLE 1. Required rotations for each round

		Cycle
Round 0	Add Round Key	1
Round 1	Sub bytes	2
	Shift Rows	3
	Mix column	4
	Add Round Key	5
Round 2	Sub bytes	6
	Shift Rows	7
	Mix column	8
	Add Round Key	9
Round 3	Sub bytes	10
	Shift Rows	11
	Mix column	12
	Add Round Key	13
	.	.
	.	.
	.	.
Round 14	Sub bytes	54
	Shift Rows	55
	Add Round Key	56

Data structure:

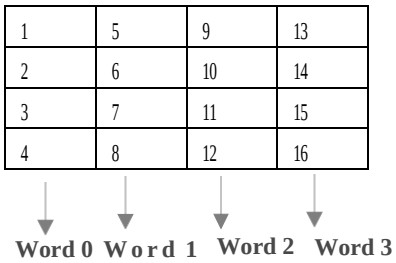


FIGURE 4. Word format of 128-bit data

Figure 4 depicts the layout of a 128-bit matrix. Each column consists of four 8-bit elements, resulting in a total of 32 bits per column or word.

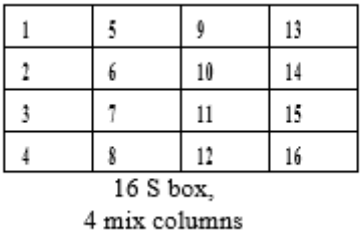


FIGURE 5. S-box for regular mode

Figure 5 illustrates the structure of the S-box (Substitution box) used in the regular mode of the AES algorithm. The S-box is a critical nonlinear component in AES, responsible for performing byte-level substitution to enhance the security of the cipher. As shown in the figure, the S-box consists of 16 substitution values arranged in a 4x4 matrix. Each entry in the matrix represents a byte from the 128-bit block, and these bytes are passed through the S-box during the SubBytes operation. The substitution process ensures non-linearity and resistance against linear and differential cryptanalysis by replacing each byte with a fixed value defined by the AES standard lookup table. Additionally, the diagram mentions "4 mix columns," which refers to the four columns of the state matrix that undergo the MixColumns transformation after substitution. This combination of byte substitution and column mixing contributes to the strong diffusion and confusion properties essential for robust encryption in AES.

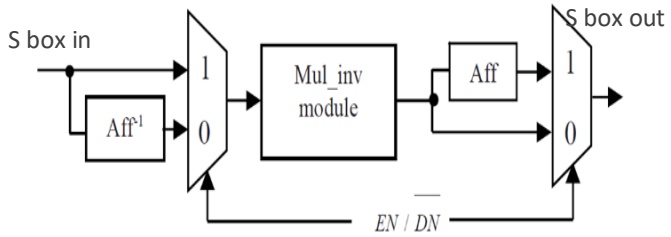


FIGURE 6. Integrated structure of S-box and Inverse S-box

In the proposed 32-bit operations implementation, significant optimization is achieved by reusing critical AES components, such as the S-box and MixColumn modules. By processing data in 32-bit segments instead of the conventional 128-bit blocks, this method enables the same S-box and MixColumn circuits to handle multiple rounds through pipelined execution, significantly reducing hardware redundancy. The integration of MixColumn and Add Round Key operations into a single processing unit, referred to as the "Mix Block," further streamlines the encryption and decryption processes. This approach enhances hardware efficiency, reduces power consumption, and conserves area on FPGA implementations without compromising cryptographic strength.

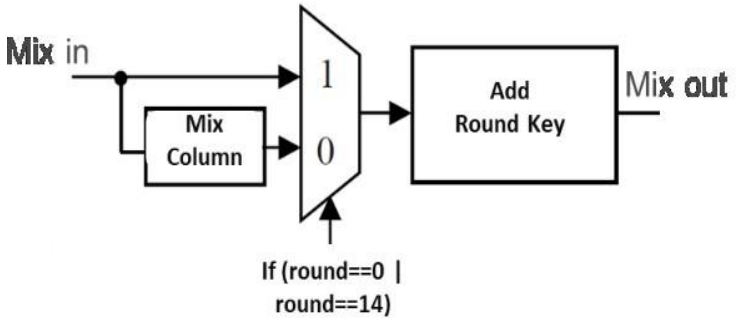


FIGURE 7. Integrated system of Mix column and Add circular key - Mix

S box	Shift	Mix	Cycle
		Mix 0	1
Sub 0	-	Mix 1	2
Sub 1	Shift 0	Mix 2	3
Sub 2	Shift 1	Mix 3	4
Sub 3	Shift 2	-	5
-	Shift 3	Mix 0	6
Sub 0	-	Mix 1	7
Sub 1	Shift 0	Mix 2	8
Sub 2	Shift 1	Mix 3	9
Sub 3	Shift 2	-	10
-	Shift 3	Mix 0	11
Sub 0	-	Mix 1	12
Sub 1	Shift 0	Mix 2	13
Sub 2	Shift 1	Mix 3	14
Sub 3	Shift 2	-	15
-	Shift 3	Mix 0	16
.	.	Mix 1	17
.	.	Mix 2	18
Sub 0	-	Mix 3	19
Sub 1	Shift 0	.	.
Sub 2	Shift 1	.	.
Sub 3	Shift 2	-	.
-	Shift 3	Mix 0	71
		Mix 1	72
		Mix 2	73
		Mix 3	74

FIGURE 8. The pipeline structure of the proposed design uses different colors to distinguish each circuit. Specifically, the structure includes Mix - Circuit 0, Mix - Circuit 1, Mix - Circuit 2, Mix - Circuit 3, and Mix - Circuit 14, with each circuit represented by a unique color to indicate its role and position within the data flow.

In this approach, each 32-bit word undergoes a 32-bit sub-operation. This process involves performing sub-operations, shift operations, and mix operations over multiple clock cycles. In cycle 2, sub_0 and mix_1 are executed, followed by sub_1, shift_0, and mix_2 in cycle 3, and so on. After cycle 0, mix operations are no longer required in subsequent cycles. The hardware is reused for each cycle, with different operations occurring in parallel for each word. This pattern is repeated for 14 cycles, improving performance by reusing the same logic for the sub, shift, and mix operations across cycles.

S box	Shift	Mix	Cycle
		Mix 0	1
Sub 0	-	Mix 1	2
Sub 1	Shift 0	Mix 2	3
Sub 2	Shift 1	Mix 3	4
Sub 3	Shift 2	-	5
Key 2	Shift 3	Mix 0	6
Sub 0	-	Mix 1	7
Sub 1	Shift 0	Mix 2	8
Sub 2	Shift 1	Mix 3	9
Sub 3	Shift 2	-	10
Key 3	Shift 3	Mix 0	11
Sub 0	-	Mix 1	12
Sub 1	Shift 0	Mix 2	13
Sub 2	Shift 1	Mix 3	14
Sub 3	Shift 2	-	15
-	Shift 3	Mix 0	16
.	.	Mix 1	17
Key 14	.	Mix 2	18
Sub 0	-	Mix 3	19
Sub 1	Shift 0	.	.
Sub 2	Shift 1	.	.
Sub 3	Shift 2	-	.
-	Shift 3	Mix 0	71
		Mix 1	72
		Mix 2	73
		Mix 3	74

FIGURE 9. Pipeline structure of the key gen module

TABLE 2. Required cycles for each round

Round 0	04 cycles
Round 1 to 14	70 cycles
Total	**74 cycles**

		Cycle
Round 0	Add Round Key	1
	Sub bytes	2
	Shift Rows	3
	Mix column	4
Round 1	Add Round Key	5
	Sub bytes	6
	Shift Rows	7
	Mix column	8
Round 2	Add Round Key	9
	Sub bytes	10
	Shift Rows	11
	Mix column	12
Round 3	Add Round Key	13
	.	.
	.	.
	.	.
	Sub bytes	54
	Shift Rows	55
Round 14	Add Round Key	56

FIGURE 10. Regular method

S box	Shift	Mix	Cycle
		Mix 0	1
Sub 0	-	Mix 1	2
Sub 1	Shift 0	Mix 2	3
Sub 2	Shift 1	Mix 3	4
Sub 3	Shift 2	-	5
Key 2	Shift 3	Mix 0	6
Sub 0	-	Mix 1	7
Sub 1	Shift 0	Mix 2	8
Sub 2	Shift 1	Mix 3	9
Sub 3	Shift 2	-	10
Key 3	Shift 3	Mix 0	11
Sub 0	-	Mix 1	12
Sub 1	Shift 0	Mix 2	13
Sub 2	Shift 1	Mix 3	14
Sub 3	Shift 2	-	15
-	Shift 3	Mix 0	16
-	-	Mix 1	17
Key 14	-	Mix 2	18
Sub 0	-	Mix 3	19
Sub 1	Shift 0	-	-
Sub 2	Shift 1	-	-
Sub 3	Shift 2	-	-
-	Shift 3	Mix 0	71
		Mix 1	72
		Mix 2	73
		Mix 3	74

FIGURE 11. Pipeline structure of the proposed method

Figure 11 illustrates the operational schedule of the AES algorithm with a detailed mapping of the S-box, Shift, Mix, and cycle executions across various stages. This table highlights how the proposed method leverages efficient time-sharing and reuse of resources like MixColumns and SubBytes units. By cycling through Sub operations (Sub 0–Sub 3), Shift operations (Shift 0–Shift 3), and corresponding Mix blocks (Mix 0–Mix 3) in a staggered and optimized sequence, the design minimizes the need for parallel hardware instances. This pipeline-friendly execution model not only conserves FPGA resources but also maintains throughput by allowing concurrent processing across stages.

4. RESULTS AND COMPARISON

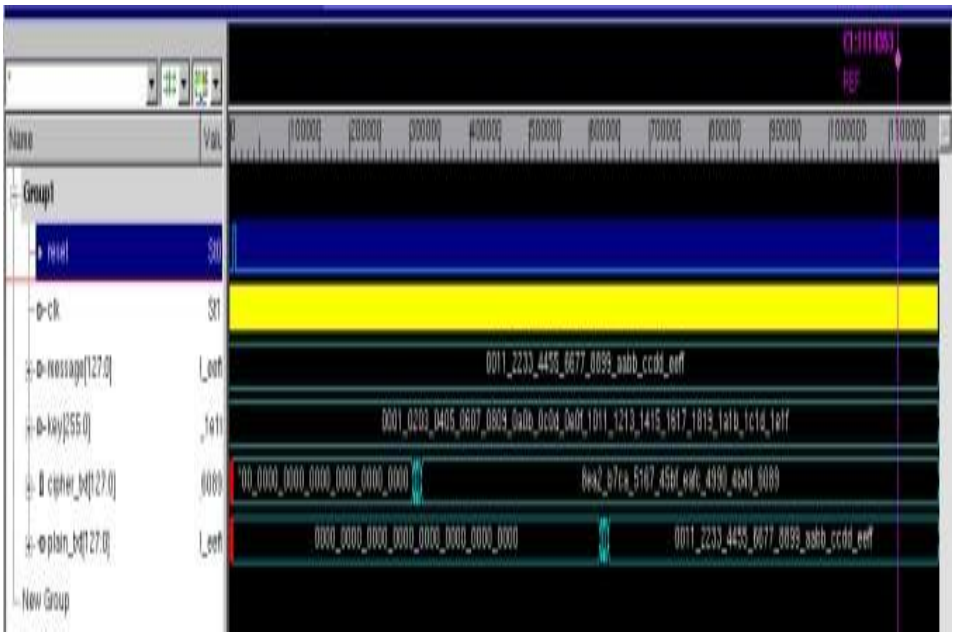


FIGURE 12. Shows VCS simulation of 256-bit key AES encryption and decryption. The cipher text matches the standard 256 AES algorithm results. Checks all implementations.

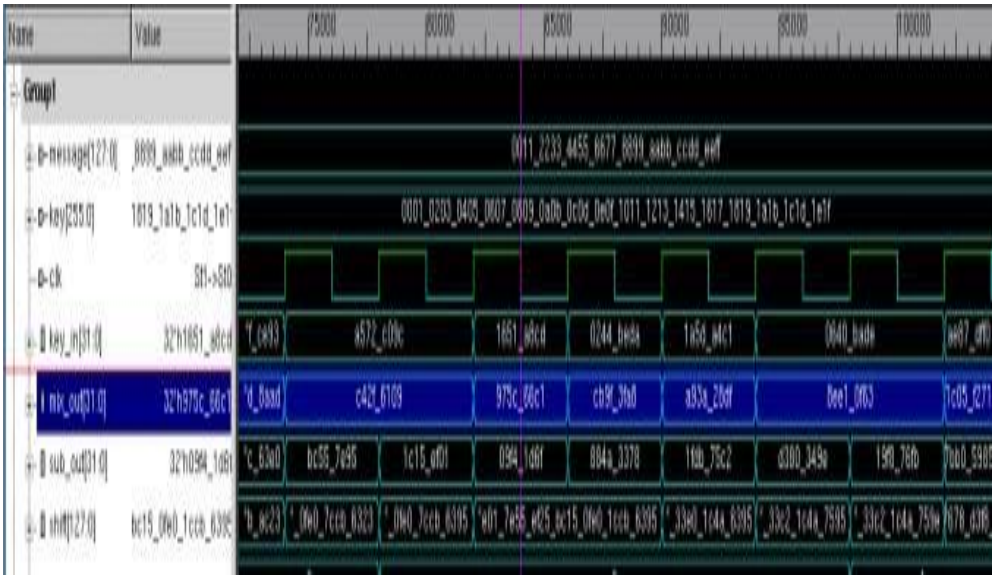


FIGURE 13. Output of circuit 4 internal functions

```
round[ 3].k_sch    1651a8cd0244beda1a5da4c10640bade
round[ 4].start    975c66c1cb9f3fa8a93a28df8ee10f63
round[ 4].s_box    884a33781fdb75c2d380349e19f876fb
round[ 4].s_row    88db34fb1f807678d3f833c2194a759e
round[ 4].m_col    b2822d81abe6fb275faf103a078c0033
round[ 4].k_sch    ae87dff00ff11b68a68ed5fb03fc1567
round[ 5].start    1c05f271a417e04ff921c5c104701554
```

FIGURE 14. The AES standards reference document

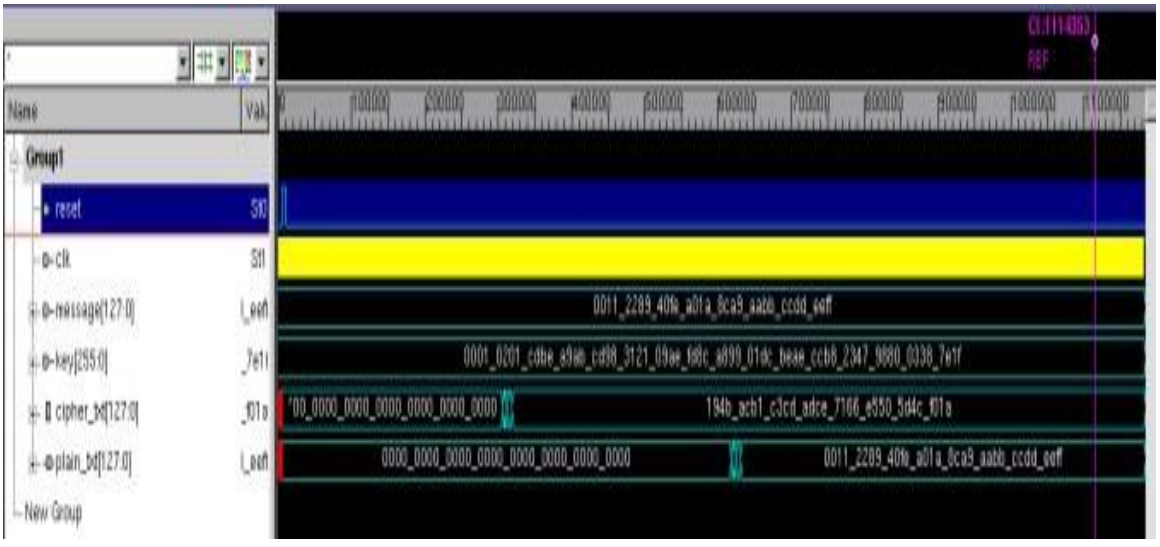


FIGURE 15. Simulated waveform of random inputs

Figure 15 presents a simulation waveform generated using Synopsys VCS for the proposed 256-bit AES algorithm implementation on a Virtex-7 FPGA. The waveform shows a sequence of signals including clock (b_clk), reset (reset), 128-bit message input (message [127:0]), 256-bit encryption key (keys[255:0]), cipher output (cipher_txt[127:0]), and encrypted output (normal_bq[127:0]). At the beginning of the simulation, the reset signal is activated, initializing the system. Once the reset is

deactivated, the encryption process begins. The random plaintext message and encryption key are loaded and processed. The waveform shows the generation of the encrypted ciphertext after several clock cycles, followed by the successful recovery of the original plaintext by the decryption process. The simulation confirms the operational correctness of both the encryption and decryption paths, with the final encrypted output matching the original message. This confirms the operational integrity of the proposed AES design under random input conditions.

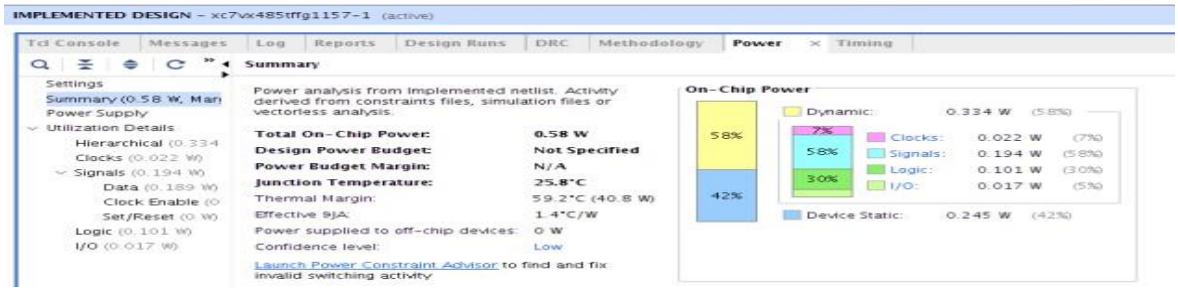


FIGURE 16. Typical implementation on-chip power

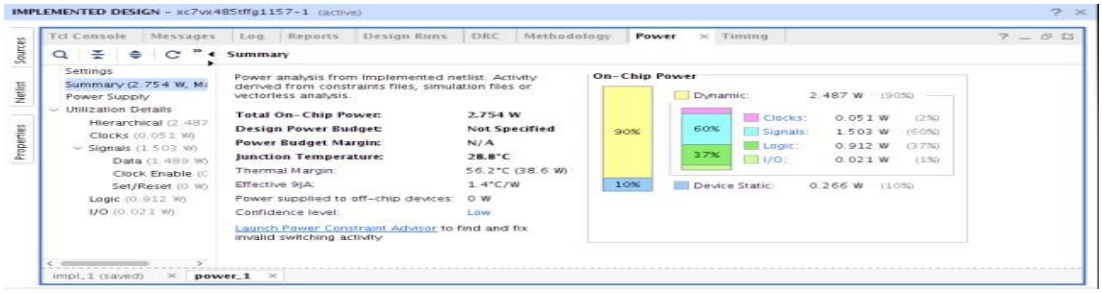


FIGURE 17. Proposed implementation on-chip power

Figure 16 & Figure 17 show the on-chip power in the Vive Ado implementation for the conventional and proposed methods.

FIGURE 18. Typical implementation area usage

FIGURE 19. Proposed Implementation Area Usage

Figure 18 & Figure 19 show the area usage in the Viva do implementation for the conventional and proposed methods.

TABLE 3. Sub-module usage comparison

Block	**Instance**	**Conventional**	**Proposed**
Sub bytes	S box	16	4
Mix Column	Mix	4	1
Key Gen	S box	8	0

Table 3 presents a comparative analysis of sub-module usage between the conventional and proposed implementations of the 256-bit AES algorithm. The comparison focuses on three critical blocks: Sub Bytes, Mix Column, and Key Generation, highlighting the efficiency gains achieved through hardware reuse and architectural optimization. In the conventional design, the Sub Bytes operation utilizes 16 S-box instances, reflecting the typical approach of assigning one S-box per byte of a 128-bit block. However, in the proposed design, only 4 S-box instances are required, significantly reducing hardware complexity and conserving FPGA resources. Similarly, for the Mix Column operation, the conventional method uses 4 separate Mix modules—one for each column in the 4x4 AES data matrix—whereas the proposed approach consolidates this into a single Mix module. This reuse drastically reduces area consumption while maintaining functional accuracy. Furthermore, in the Key Generation stage, the conventional method requires 8 additional S-boxes for performing substitution operations during round key generation. In contrast, the proposed design eliminates the need for separate S-boxes in key generation by reusing the existing S-box modules, achieving further hardware optimization. the proposed approach significantly minimizes the use of logic resources, resulting in reduced power consumption and increased area efficiency without compromising performance.

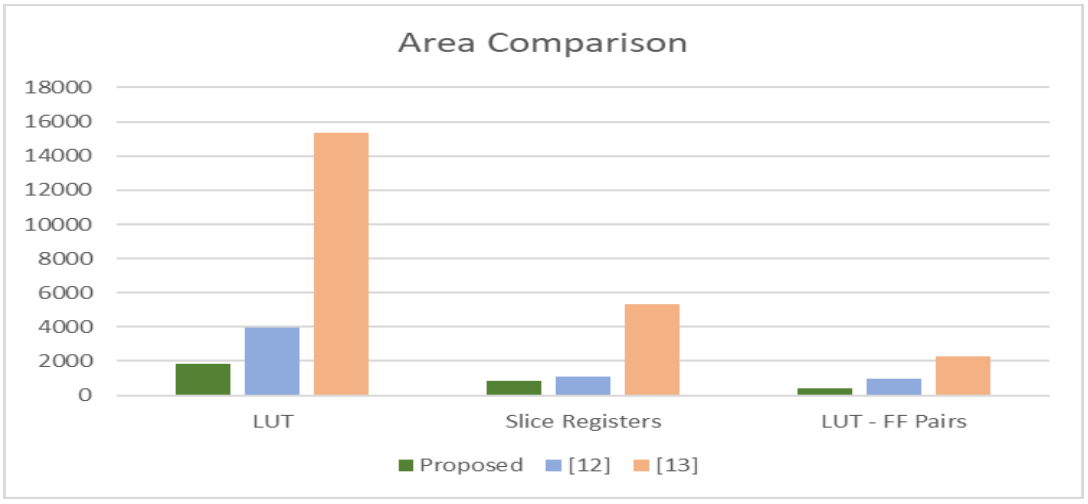


FIGURE 20. Area Comparison: Proposed and Existing Methods

Figure 20 provides a chart comparing the area usage between the proposed and existing methods [12] [13].

TABLE 4. provides a comparison of the power, performance, and area (PPA) metrics between the conventional and proposed methods for FPGA implementation [6] [8].

FPGA			
Method	**Convention**	**Proposed**	**% Savings**
Total cycles for encryption	56	74	
Frequency (MHz)	109	161	
Throughput (Mbps)	249	278	10.53
On chip power (W)	2.75	0.58	78.91
Slice LUT	4834	1814	62.47
Slice Register	3095	836	72.99
LUT Flip Flop Pairs	1131	434	61.63

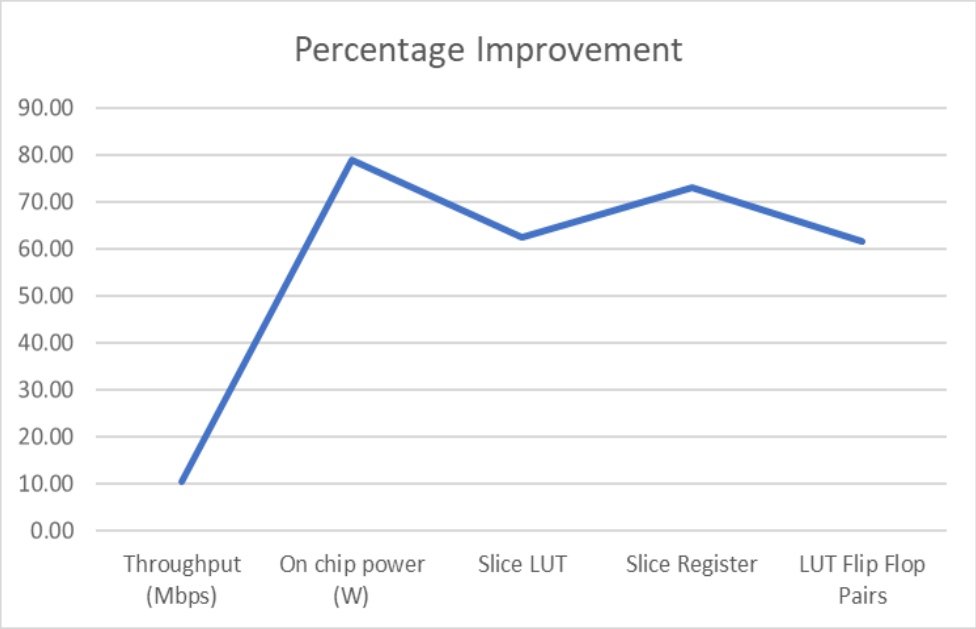


FIGURE 21. PPA Comparison Conventional vs Proposed

TABLE 5. Area Comparison for Proposed vs Existing Method

FPGA			
Method	**Proposed**	**[12]**	**[13]**
Slice LUT	1814	3959	15376
Slice Register	836	1124	5356
LUT Flip Flop Pairs	434	973	2309

4. CONCLUSION

In addition to significantly reducing area and power consumption, the proposed 256-bit AES algorithm implementation on the Virtex-7 (xc7vx485tffg1157) FPGA notably enhances system efficiency, scalability, and versatility. The design introduces several architectural innovations aimed at optimizing hardware utilization while maintaining the robustness and high security standards of the AES-256 algorithm. By adopting a modular approach that reuses 32-bit SubBytes and MixColumns blocks across standard 128-bit data operations, the implementation maximizes hardware efficiency without compromising cryptographic strength or performance. One of the most notable features of the proposed architecture is the use of a unified S-box, which serves both the SubBytes transformation and the key expansion process. In traditional implementations, these operations often use separate S-boxes, leading to redundant resource allocation. The consolidation of these S-box functions into a single, shared hardware unit effectively

minimizes duplication, contributing significantly to area reduction and overall design simplicity. This approach not only saves valuable Look-Up Tables (LUTs) and flip-flops (FFs) on the FPGA but also streamlines the data path, reducing signal delay and boosting the speed of the algorithm. Moreover, the architecture is designed to reuse the same hardware modules for both encryption and decryption processes. This dual-functionality further enhances system compactness and reduces the need for extra logic components, leading to substantial power savings. As a result, the design demonstrates a remarkable 78% reduction in power consumption compared to conventional AES implementations, along with a 72% decrease in slice registers, 62% in slice LUTs, and 61% in LUT-FF pairs. Pipelining is another core aspect of the implementation, enabling multiple stages of the AES algorithm to be processed concurrently. This technique improves overall throughput and reduces latency, allowing the system to handle higher volumes of data more efficiently. The result is a 10% increase in data throughput (measured in Mbps), making the design highly suitable for real-time encryption and decryption tasks. The design's support for multiple key sizes—128, 192, and 256 bits—as well as various word sizes (16, 32, and 64 bits) enhances its flexibility across a wide range of applications. Whether used in secure wireless communication, financial systems, embedded controllers, or real-time video and image processing, this adaptable architecture can be tailored to meet specific performance and security requirements. The proposed AES-256 FPGA implementation offers a balanced solution to the common trade-offs between power, area, and performance in hardware cryptography. The reuse of core components, support for multiple configurations, and effective pipelining strategies collectively contribute to a highly optimized and scalable design. Its lightweight footprint and low energy demands make it especially valuable for next-generation cryptographic systems deployed in mobile, embedded, and IoT environments, where conserving resources without sacrificing security is of paramount importance. This work demonstrates that with thoughtful architectural innovation, high-performance encryption can be both compact and energy-efficient.

REFERENCES

[1]. M. Rajeswara Rao, Dr.R.K.Sharma, SVE Department, NIT Kurushetra "FPGA Implementation of combined S box and Inv S box of AES" 2017 4th International conference on signal processing and integrated networks (SPIN).

[2]. Nalini C. Iyer ; Deepa ; P.V. Anandmohan ; D.V. Poornaiah "Mix/InvMixColumn decomposition and resource sharing in AES".

[3]. Xinmiao Zhang, Student Member, IEEE, and Keshab K. Parhi, Fellow, "High Speed VLSI architectures for the AES Algorithm", IEEE. VOL.12. No.9. September 2004

[4]. Shrivathsa Bhargav, larry Chen, abhinandan Majumdar, Shiva Ramudit "128 bit AES Decryption", CSEE 4840 – Embedded system Design spring 2008, Columbia University.

[5]. Atul M. Borkar ; R. V. Kshirsagar ; M. V. Vyawahare "FPGA implementation of AES algorithm". Announcing the ADVANCED ENCRYPTION STANDARD (AES), November 26 2001.

[6]. Yulin Zhang ; Xinggang Wang; "Pipelined implementation of AES encryption based on FPGA" 2010 IEEE International Conference on Information Theory and Information Security.

[7]. Yuwen Zhu ; Hongqi Zhang ; Yibao Bao ; "Study of the AES Realization Method on the Reconfigurable Hardware" 2013 International Conference on Computer Sciences and Applications.

[8]. Tsung-Fu Lin ; Chih-Pin Su ; Chih-Tsun Huang ; Cheng-Wen Wu; "A high-throughput low-cost AES cipher chip" Proceedings. IEEE Asia- Pacific Conference on ASIC.

[9]. C. Sivakumar ; A. Velmurugan ; "High Speed VLSI Design CCMP AES Cipher for WLAN (IEEE 802.11i)" 2007 International Conference on Signal Processing, Communications and Networking.

[10]. Vatchara Saicheur ; Krerk Piromsopa ; "An implementation of AES- 128 and AES-512 on Apple mobile processor" 2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)

[11]. S.P Guruprasad ; B.S Chandrasekar ; "An evaluation framework for security algorithms performance realization on FPGA" 2018 IEEE

[12]. International Conference on Current Trends in Advanced Computing (ICCTAC)

[13]. N. S. Sai Srinivas ; Md. Akramuddin; "FPGA based hardware implementation of AES Rijndael algorithm for Encryption and Decryption" 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT).

[14]. P. S. Abhijith ; Mallika Srivastava ; Aparna Mishra ; Manish Goswami; B. R. Singh ; "High performance hardware implementation of AES using minimal resources" 2013 International Conference on Intelligent Systems and Signal Processing (ISSP).

[15]. Wei Wang ; Jie Chen ; Fei Xu ; "An implementation of AES algorithm Based on FPGA" 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery

[16]. Ashwini M. Deshpande ; Mangesh S. Deshpande ; Devendra N. Kayatanavar; "FPGA implementation of AES encryption and decryption" 2009 International Conference on Control, Automation, Communication and Energy Conservation.