International Journal of Modern Developments in Engineering and Science
Volume 2, Issue 12, December 2023
https://www.ijmdes.com | ISSN (Online): 2583-3138

# Area-Efficient AES Design for IoT Devices

G. Shyam Kishore[1], Koppula Krishna Murthy[2], Polasa Vamshika[3], Sai Kiran Shinde[4*]

[1]*Associate Professor, Department of Electronics and Communication Engineering, CMR College of Engineering & Technology, Hyderabad, India*
[2,3,4]*UG Student, Department of Electronics and Communication Engineering, CMR College of Engineering & Technology, Hyderabad, India*

***Abstract*: This project prioritizes both low power consumption and space efficiency while improving AES implementation for Internet of Things devices with limited resources. Clock gating methods and the integration of the Sub-Bytes function into the State Register greatly minimize power consumption and area overhead by lowering the number of clock cycles that are allocated to inactive circuit parts. This solution provides notable reductions in area overhead over traditional AES implementations through careful hardware design decisions and algorithmic optimization. This work makes a substantial contribution to the development of effective and safe cryptographic systems designed to meet the unique requirements of IoT settings, where security is of utmost importance yet resources are scarce.**

***Keywords*: AES cryptographic algorithm, security, area efficiency, cryptography, clock gating.**

## 1. Introduction

During the Internet of Things (IoT) era, where billions of devices are interconnected to facilitate automation, data collection, and seamless communication, ensuring security becomes paramount. Because of its widespread use and robustness, the Advanced Encryption Standard (AES) cryptographic method is considered a cornerstone for protecting sensitive data, among other options. However, implementing [10] AES on resource-constrained IoT devices poses significant challenges, particularly in terms of area efficiency, where the objective is to attain maximum performance with the least amount of gear.

This introduction outlines the significance of area-efficient nano AES implementations tailored specifically for IoT devices. Nano AES refers to ultra-compact implementations of AES suitable for devices with strict area constraints, such as microcontrollers and sensor nodes. Here, we go over the significance of these solutions in terms of IoT security and investigate the main causes for the demand for area efficiency.

### *1) IoT Security Imperatives*

In the IoT, a myriad of devices, ranging from tiny sensors to interconnected appliances, constantly exchange sensitive data. [5] Protecting this data's confidentiality and integrity is essential to avoiding illegal access, alteration, or eavesdropping. [2] AES, with its strong cryptographic properties, offers a viable solution for protecting communications via the Internet of Things. communications. However, traditional AES implementations may not be feasible due to the limited computational and memory resources available on IoT devices.

### *2) Resource Constraints*

IoT devices typically operate under stringent resource constraints, including limited processing power, memory, and energy supply. Traditional implementations of AES, optimized for performance on desktop or server environments, often require excessive hardware resources that surpass the capabilities of IoT devices. [3] Therefore, there is a need for nano AES implementations that create a harmony between safety and resource utilization, specifically targeting the constraints inherent in IoT environments.

### *3) Area-Efficiency Challenges*

Achieving area efficiency in nano AES implementations involves optimizing the cryptographic algorithm's hardware implementation to minimize the application of silicon area while maintaining adequate security levels. [4] This entails exploring techniques such as algorithmic optimizations, hardware acceleration, and trade-offs between performance and resource consumption. The difficulty is in designing compact yet secure implementations that meet the stringent area constraints imposed by Internet of Things gadgets without compromising on security.

### *4) Key Objectives*

The principal aim of area-efficient nano AES implementations for Internet of Things gadgets is to provide robust cryptographic protection while conserving hardware resources. [12] This entails designing lightweight cryptographic cores that consume minimal silicon area, memory, and power while delivering adequate encryption and decryption performance. Additionally, these implementations should be scalable and adaptable to accommodate many Internet of Things device architectures and application scenarios.

## 2. Literature Survey

*Secure communication requires the Advanced Encryption Standard (AES) algorithm, but conventional implementations may be too resource-intensive for Internet of Things (IoT) devices with constrained processing and memory. This section examines several methods for developing effective and lightweight AES implementations that work in these resource-constrained settings.*

*Combined Operations:* Merging SubBytes and ShiftRows stages reduces the number of required operations, streamlining

*Corresponding author: saikiranshinde01@gmail.com the overall process. [8] Alternative MixColumns: Employing alternative methods like polynomial multiplication for MixColumns can decrease hardware complexity compared to traditional implementations. Efficient Key Expansion: Utilizing efficient key expansion algorithms like the key schedule helps minimize the area footprint of the design. Shared Resources: Sharing resources like adders and multiplexers across different stages of AES computation reduces the overall hardware footprint, minimizing the required silicon area. Pipelining and Time-Multiplexing: Implementing pipeline stages or time-multiplexing resources allows for parallel processing of different AES rounds, improving throughput without significantly increasing area usage. Optimized Data Paths and Representations: Designing efficient data paths, employing compact data representations, and implementing efficient data flow control all contribute to minimizing the overhead associated with data manipulation during encryption and decryption. [4] Memory Optimization Techniques: Utilizing efficient memory architectures for storing intermediate cipher states, round keys, and S-box tables reduces the memory footprint, further contributing to a resource-constrained design. Bit-Slice Implementations: Bit-slice implementations offer high parallelism and efficient utilization of hardware resources, especially in Field-Programmable Gate Arrays (FPGAs). [6] Hardware Accelerators: Designing custom hardware accelerators or co-processors dedicated solely to AES computation can be highly area-efficient by focusing only on the necessary functionalities. Power Reduction Techniques: Techniques like clock gating, power gating, and voltage scaling are employed to reduce overall power consumption without significantly impacting the area footprint. [2] Area-Power Efficiency Synergy: Optimizing for area efficiency can often lead to improvements in power efficiency as well, due to reduced switching activity in the circuit. Approximate Computing: Approximate computing explores trading off a small degree of accuracy for significant area savings. However, the security implications of such an approach need careful evaluation and may not be suitable for all applications. Quantum-Inspired Techniques: Drawing inspiration from quantum computing algorithms, techniques like quantum-inspired linear algebra or reversible logic might offer new avenues for developing area-efficient hardware implementations of AES. [1] High-Level Synthesis (HLS) tools offer an automated approach by generating hardware implementations directly from high-level algorithmic descriptions. These tools can optimize resource utilization by exploring various architectural options and scheduling strategies, leading to efficient designs.

By carefully combining these techniques, designers can create lightweight and secure AES implementations suitable for deployment in resource-constrained IoT devices. This ensures secure communication and data protection within the IoT ecosystem while minimizing resource overhead.

*Trade-offs between area efficiency, performance, and security are crucial considerations in the design of AES implementations for IoT devices*

*Area Efficiency vs. Performance:*

*Trade-off:* Increasing area efficiency often involves simplifying hardware designs or using resource-sharing techniques, which can lead to reduced performance.

*Discussion:* Designs optimized for area efficiency may sacrifice performance in terms of throughput or latency. For example, resource-sharing techniques or pipeline optimizations can introduce additional latency, impacting overall performance. [8] Balancing area efficiency with performance requires careful consideration of the IoT application and the desired trade-offs between area footprint and computational speed.

*Area Efficiency vs. Security:*

*Trade-off:* Optimizing for area efficiency may involve using simplified cryptographic algorithms or reducing the size of cryptographic keys, which can compromise security.

*Discussion:* Lightweight cryptography algorithms tailored for resource-constrained devices often sacrifice some level of security for improved area efficiency. For example, reducing the key size or using simplified substitution-permutation network (SPN) structures can make AES implementations more vulnerable to certain cryptographic attacks. [9] Designers must carefully assess the security implications of area optimization techniques and ensure that the chosen implementation meets the security requirements of the IoT application.

*Performance vs. Security:*

*Trade-off:* Improving performance, such as increasing throughput or reducing latency, may require sacrificing certain security features or using less secure cryptographic algorithms.

*Discussion:* Some performance optimization techniques, such as parallel processing or algorithmic simplifications, may inadvertently weaken the security of AES implementations. For instance, parallel processing of multiple data streams can introduce timing side-channel vulnerabilities, while algorithmic simplifications may make the encryption scheme more susceptible to cryptanalysis. Designers must strike a balance between performance and security requirements, considering factors such as acceptable risk levels, threat models, and potential attack vectors.

*Optimizing for Balance:*

*Trade-off:* Achieving an optimal balance between area efficiency, performance, and security often requires making trade-offs and compromises in each aspect.

*Discussion:* Designers must carefully evaluate the requirements and constraints of the target IoT application to determine the appropriate trade-offs. For example, applications with strict area constraints may prioritize area efficiency over performance, while applications requiring high levels of security may prioritize security over performance optimizations. [7] Ultimately, the optimal design depends on the specific use case, resource constraints, and security requirements of the IoT deployment.

*Dynamic Adaptation:*

Trade-off: Implementations that dynamically adapt based on workload or security requirements may incur additional overhead or complexity.

*Discussion:* Dynamic adaptation mechanisms, such as reconfigurable hardware architectures or runtime security monitoring, can help balance area efficiency, performance, and security dynamically. However, there's a chance that these processes will increase computing complexity, power consumption, or area overhead. [12] Designers must carefully assess the trade-offs associated with dynamic adaptation and determine whether the benefits outweigh the additional costs in the context of the IoT application.

In summary, achieving an optimal balance between area efficiency, performance, and security is a complex challenge in the design of AES implementations for IoT devices. [3]-[5] Designers must carefully evaluate trade-offs and make informed decisions based on the particular limitations and specifications of the intended use.

## 3. Existing System

### *A. Tiny AES*

[2] Tiny AES aims to provide a minimalistic yet efficient AES implementation suited for devices with low power, memory, and energy resources. AES operates on blocks of data, whose key size varies among 128, 192, or 256 bits, and supports various modes of operation.

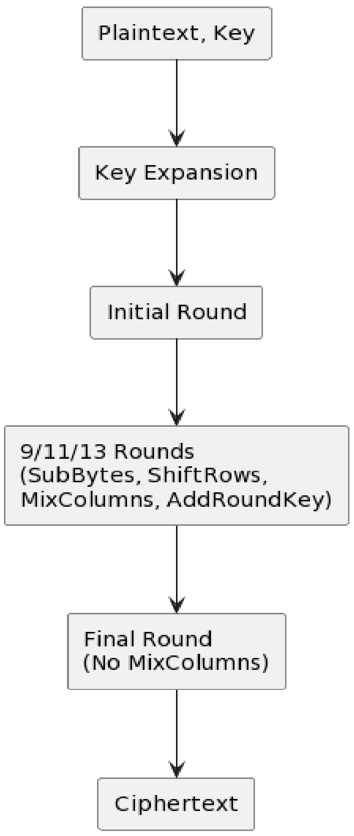
Plaintext, Key
Key Expansion
Initial Round
9/11/13 Rounds (SubBytes, ShiftRows, MixColumns, AddRoundKey)
Final Round (No MixColumns)
Ciphertext

Fig. 1. Traditional AES flow

This flowchart represents the AES encryption process with the ECB mode of operation, where each plaintext block is encrypted independently with the key. [10] For other modes of operation, such as CBC or CTR, additional steps such as initialization vector (IV) generation and chaining would be incorporated into the flowchart.

## 4. Proposed System

Explanation of the Proposed 8-Bit Datapath nano -AES Architecture.

This section discusses the architecture of a novel lightweight AES implementation designed for resource-constrained devices (Fig. 2).

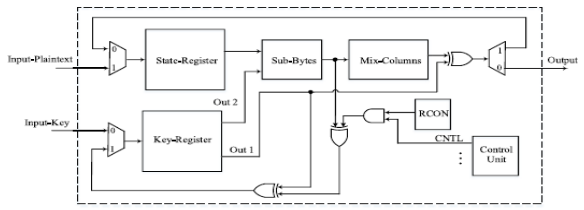

Fig. 2. Structure of AES

*Key Components:*

*Key-Register and State-Register:* These register banks store the encryption key, plain text, and intermediate results during the encryption process.

*RCON Block:* This block generates round constants used during the key expansion phase.

*Control Unit:* This unit orchestrates the entire encryption process by controlling the data flow and triggering various operations.

*Efficiency Focus:*

The design prioritizes minimizing unnecessary operations. The Mix-Columns and Sub-Bytes functions are combined to streamline the process.

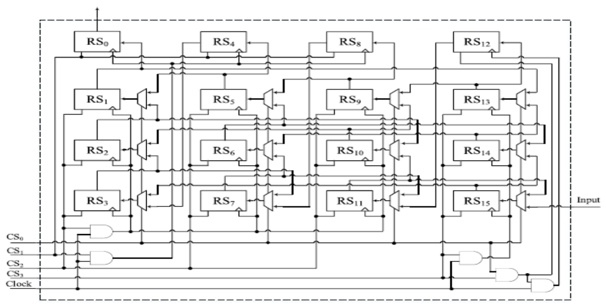

Fig. 3. State-Register

*Structure:* The State-Register consists of 16 individual 8-bit registers, each containing eight flip-flops.

*Shift-Rows Integration:* This design integrates the Shift-Rows functionality within the State Register itself, eliminating the need for a separate block (saving area).

*Data Flow:* Each register receives data from the previous one during encryption and vice versa during decryption. The state register utilizes multiplexers for selecting data inputs.

*Control Signals:* Control signals manage register activation and data flow during different operations.

*Benefits of Integrated Shift-Rows:*

*Area Efficiency:* Embedding Shift-Rows within the State-Register reduces hardware footprint compared to dedicated Shift-Rows blocks used in previous designs (e.g., by Jarvinen et al. and Zhao et al.).

*Reduced Complexity:* This approach eliminates the need for complex control units required for separate Shift-Rows blocks, leading to lower power consumption.

*Mix-Columns Operation:*

Mix-Columns operates on a single column of data at a time.

The State Register facilitates data transfer to and from the Mix-Columns block for processing.

*State-Register Control Signals:*

Four control signals (CS0, CS1, CS2, and CS3) manage register access during different operations.

CS1 and CS2 select the active row of registers. CS0 and CS3 combined activate the second row, while CS1 alone activates the first row.

*State-Register Management:*

*Managing the State-Register involves:*

Data loading and unloading selecting active registers based on the operation (Shift-Rows, Mix-Columns), controlling data flow during encryption and decryption, and synchronizing operations with the control unit, ensuring proper timing for each step. Overall, the proposed architecture prioritizes area efficiency and reduces complexity by integrating functionalities like Shift-Rows within the State-Register. This lightweight design caters to the resource limitations of devices commonly used in the Internet of Things (IoT).

This section details the operation of the State-Register, a crucial component in the proposed nano-AES architecture.

*1) Loading the Plain Text*

To load the plain text into the State-Register, all 16 internal registers need to be activated simultaneously. This is achieved by setting all control signals (CS0, CS1, CS2, CS3) to "1."

The design feeds 8 bits of data in each clock cycle. This data is stored in the last register (RS15) of the State-Register (refer to Fig. 3).

Thanks to the shift-register memory structure and internal connections, new incoming data triggers a shift in the existing data. The data stored in RS15 moves to RS14, RS14 to RS13, and so on, ultimately reaching RS0.

*2) Executing Shift-Rows*

The Shift-Rows operation is cleverly integrated within the State-Register through internal connections and wiring.

To activate Shift-Rows, only the registers in the second, third, and fourth columns need to be active. This is achieved by setting CS0 and CS1 to "0" while setting CS2 and CS3 to "1."

*3) Handling First Add-Round-Key and Last Round*

These rounds involve feeding the design with new data while simultaneously storing it in the State-Register, as Mix-Columns is not performed in these stages.

Similar to loading the plain text, all internal registers need to be activated (all control signals set to "1") for this operation.

*4) Feeding Data and Executing Mix-Columns*

During Mix-Columns, data from the first column of the State Register (RS0 to RS3) is fed one byte at a time for four clock cycles.

Simultaneously, the data in the registers is shifted to facilitate storing the Mix-Columns results. All control signals remain set to "1" during this process.

After four clock cycles, the data is shifted in a way that prepares the fourth column to store the outcome of the Mix-Columns operation.

*5) Storing Mix-Columns Results*

As explained earlier, storing the Mix-Columns results requires four clock cycles. This data is written only in the last column of the state register.

To achieve this, only the data in the last column is shifted to make space for the incoming Mix-Columns result. The data in other columns remains stationary.

This operation involves deactivating the internal registers in the first three columns (setting CS3, CS2, and CS1 to "1" and CS0 to "0") to temporarily cut off connections between the fourth and third columns.

Table I (mentioned in the original text) is likely to detail the specific data movement within the State Register for the Add-Round-Key and first round operations. The value of these registers would then be repeated for subsequent rounds.

## *A. Sub-Bytes Optimization*

The Sub-Bytes operation is essential for the security of the AES algorithm, but it also comes at a cost. It consumes valuable resources like power, chip space (area), and processing time (latency). This section becomes especially critical when designing for devices with limited resources. Here, we'll explore efficient ways to implement Sub-Bytes in such constrained environments.

There are traditional approaches like Lookup Tables (LUTs) and Boolean simplification maps. While these are easy to implement, they require a lot of space on the chip, making them a poor choice for resource-constrained devices. Another option is Decode-Switch-Encode (DSE), which offers a better balance between power consumption and area compared to LUTs. However, DSE still requires more space than ideal.

The most efficient approach for resource-constrained designs is called composite field arithmetic. This method breaks down the complex calculations involved in Sub-Bytes into simpler operations within smaller subfields. Imagine it as dividing a big problem into smaller, more manageable ones.

Here's how it works: We know that the inverse of a specific element in a mathematical field called Galois Field (GF(2^8)) can be calculated more efficiently if we break down GF(2^8) into smaller, simpler subfields like GF(2^1), GF(2^2), and GF(2^(2^2)). By doing this, composite field arithmetic significantly simplifies the calculation of the inverse, which is a key step in Sub-Bytes.

Choosing the right formulas (irreducible polynomials) is crucial for this method. Research suggests that decomposing GF(2^8) into specific subfields leads to the most efficient results. We can borrow these formulas from existing efficient Sub-byte designs.

Once we have the subfields defined, we can map elements from the larger field (GF(2^8)) to their corresponding elements in the smaller subfields. This allows us to perform the calculations more efficiently.

A special transformation matrix can be generated to handle this mapping between the larger field and the composite field. This transformation is achieved using special functions and their inverses.

In conclusion, composite field arithmetic offers a significant advantage for resource-constrained AES designs. By breaking down complex calculations and utilizing smaller subfields, it allows for a more space-efficient implementation of the Sub-Bytes operation, making it ideal for devices with limited resources.isomorphic function is,

$$\delta(x) = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix}.$$

inverse isomorphic function is,

$$\delta^{-1}(x) = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix}$$

After calculating the multiplicative inverse and applying the inverse isomorphic function (δ^-1), an additional step called Affine Transformation (AT) is applied to achieve the final result for the Sub-Bytes operation.

f(x): This represents the result obtained after performing the multiplicative inverse and then applying the inverse isomorphic function (δ^-1).

φ: This is a constant value denoted as {63} in base-8 notation.

g(x): This represents the final output of the Sub-Bytes block after applying the Affine Transformation.

To put it another way, the equation shows how the initial result (f(x)) is further processed using a constant value (φ) and the Affine Transformation (AT) to arrive at the final Sub-Bytes output (g(x)).

$$g(x) = AT(f(x)) + \phi = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

The Sub-Bytes of f (x) is equal to g(x) and it is obtained by,

$$\begin{aligned} g(x) &= \mathrm{Sub}(f(x)) \\ &= (\mathrm{IMUL}(\delta \times f(x)) \times \delta^{-1}) \times \mathrm{AT} \oplus \phi \\ &= (\mathrm{IMUL}(\delta \times f(x))) \times (\delta^{-1} \times \mathrm{AT} \oplus \phi) \\ &= (\mathrm{IMUL}(\delta \times f(x))) \times \gamma . \end{aligned}$$

To simplify the equation further, a new term called gamma (γ) is introduced. Gamma represents a mix of the inverse isomorphic function and the Affine Transformation. To put it another way, γ is calculated by multiplying the inverse isomorphic function (δ^-1) with the Affine Transformation (AT), and then multiplying the result by another constant value (φ).

By defining gamma (γ), the equation becomes more concise, highlighting the combined effect of the inverse isomorphic function and the Affine Transformation on the Sub-Bytes output in this composite field arithmetic approach.

$$\begin{aligned} \gamma_7 &= x_7 \oplus x_3 \oplus x_2 \\ \gamma_6 &= \overline{x_7 \oplus x_6 \oplus x_5 \oplus x_4} \\ \gamma_5 &= \overline{x_7 \oplus x_2} \\ \gamma_4 &= x_7 \oplus x_4 \oplus x_1 \oplus x_0 \\ \gamma_3 &= x_2 \oplus x_1 \oplus x_0 \\ \gamma_2 &= x_6 \oplus x_5 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_0 \\ \gamma_1 &= \overline{x_7 \oplus x_0} \\ \gamma_0 &= \overline{x_7 \oplus x_6 \oplus x_2 \oplus x_1 \oplus x_0}. \end{aligned}$$

The layout of the gamma (γ) function is optimized to minimize the resources it requires. Figure 4 details this optimized architecture. It leverages a mix of basic logic gates: 12 XOR gates, 3 XNOR gates, and just 1 NOT gate. This efficient design achieves a significant reduction in chip space (area) compared to previous implementations. Specifically, it uses 6.1% less area than one design and a remarkable 19% less area than another (references for these designs are not provided). These reductions are estimated according to the required number of logic gates and assuming a specific chip manufacturing process. Overall, the optimized gamma design offers a space-efficient solution for the Sub-Bytes operation in the nano-AES architecture.

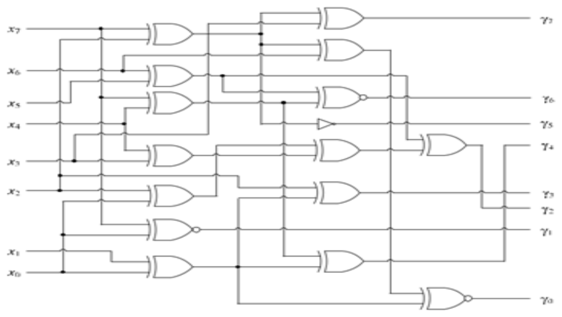

Fig. 4. Combination of inverse isomorphic and AT for sub-byte optimisation

TABLE I
CONTENT OF STATE REGISTER DURING DIFFERENT OPERATIONS FOR THE FIRST ROUND

| t | $RS_{15}$ | $RS_{14}$ | $RS_{13}$ | $RS_{12}$ | $RS_{11}$ | $RS_{10}$ | $RS_9$ | $RS_8$ | $RS_7$ | $RS_6$ | $RS_5$ | $RS_4$ | $RS_3$ | $RS_2$ | $RS_1$ | $RS_0$ | Operation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | $a_{15}$ | $a_{14}$ | $a_{13}$ | $a_{12}$ | $a_{11}$ | $a_{10}$ | $a_9$ | $a_8$ | $a_7$ | $a_6$ | $a_5$ | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ | Initial value |
| 0 | $a'_0$ | $a_{15}$ | $a_{14}$ | $a_{13}$ | $a_{12}$ | $a_{11}$ | $a_{10}$ | $a_9$ | $a_8$ | $a_7$ | $a_6$ | $a_5$ | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $RS_{15} \leftarrow$ ARK$(a_0)$ |
| 1 | $a'_1$ | $a'_0$ | $a_{15}$ | $a_{14}$ | $a_{13}$ | $a_{12}$ | $a_{11}$ | $a_{10}$ | $a_9$ | $a_8$ | $a_7$ | $a_6$ | $a_5$ | $a_4$ | $a_3$ | $a_2$ | $RS_{15} \leftarrow$ ARK$(a_1)$ |
| ⋮ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 | $a'_{15}$ | $a'_{14}$ | $a'_{13}$ | $a'_{12}$ | $a'_{11}$ | $a'_{10}$ | $a'_9$ | $a'_8$ | $a'_7$ | $a'_6$ | $a'_5$ | $a'_4$ | $a'_3$ | $a'_2$ | $a'_1$ | $a'_0$ | $RS_{15} \leftarrow$ ARK$(a_{15})$ |
| 16 | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $a'_{15}$ | $a'_{10}$ | $a'_5$ | $a'_0$ | Shift-Rows |
| 17 | $a'_{11}$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $a'_{15}$ | $a'_{10}$ | $a'_5$ | $RM_0 \leftarrow$ Sub-Bytes$(a'_0)$ |
| 18 | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $a'_{15}$ | $a'_{10}$ | $RM_0 \leftarrow$ Sub-Bytes$(a'_5)$ |
| 19 | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $a'_{15}$ | $RM_0 \leftarrow$ Sub-Bytes$(a'_{10})$ |
| 20 | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $RM_0 \leftarrow$ Sub-Bytes$(a'_{15})$ |
| 21 | $b_0$ | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $b_0 \leftarrow$ ARK$(RM_3)$ |
| 22 | $b_1$ | $b_0$ | $a'_{11}$ | $a'_{11}$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $b_1 \leftarrow$ ARK$(RM_3)$ |
| 23 | $b_2$ | $b_1$ | $b_0$ | $a'_{11}$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $b_2 \leftarrow$ ARK$(RM_3)$ |
| 24 | $b_3$ | $b_2$ | $b_1$ | $b_0$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $a'_4$ | $b_3 \leftarrow$ ARK$(RM_3)$ |
| 25 | $b_3$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ | $a'_{11}$ | $a'_6$ | $a'_1$ | $a'_{12}$ | $a'_7$ | $a'_2$ | $a'_{13}$ | $a'_8$ | $a'_3$ | $a'_{14}$ | $a'_9$ | $RM_0 \leftarrow$ Sub-Bytes$(a'_4)$ |
| ⋮ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 48 | $b_{15}$ | $b_{14}$ | $b_{13}$ | $b_{12}$ | $b_{11}$ | $b_{10}$ | $b_9$ | $b_8$ | $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ | The last value |

Fig. 4 shows the architecture of the proposed Sub-Bytes that includes the isomorphic function, and "γ " is the mix of inverse isomorphic and AT.

## 5. Results

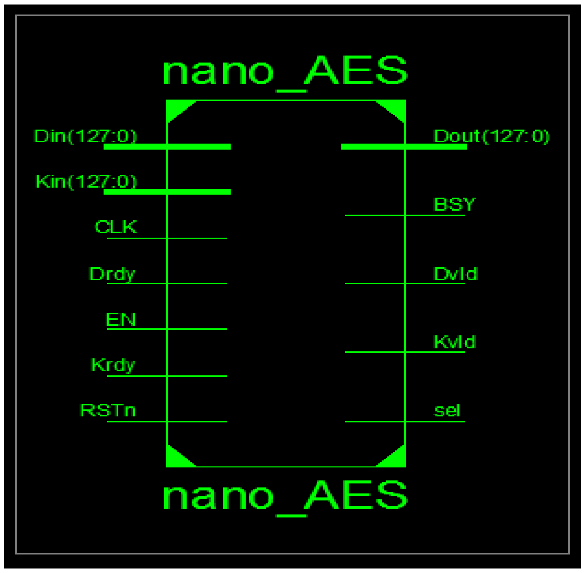

Fig. 5. Entity diagram for nano-AES

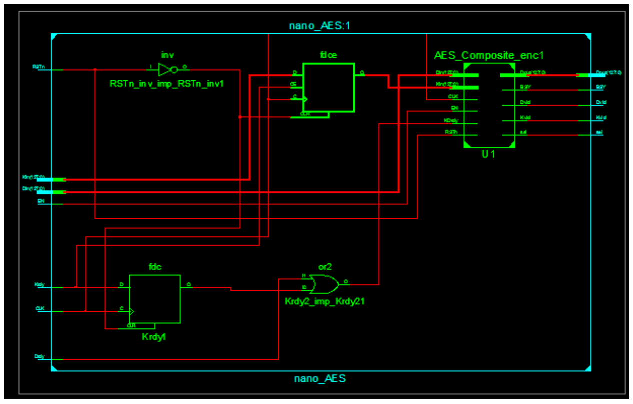

Fig. 6. RTL schematic for nano-AES

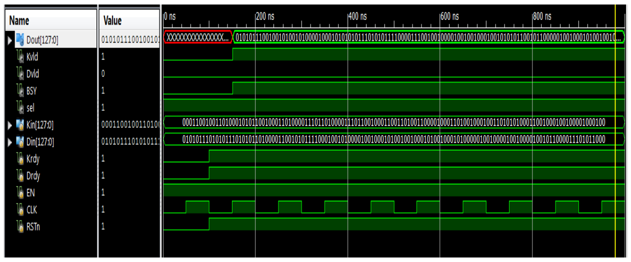

Fig. 7. Simulation results

## 6. Conclusion

In conclusion, our lightweight AES architecture addresses the crucial need for cryptography on devices with resource constraints. The nano AES implementation is intended to be compatible with existing AES standards, ensuring interoperability with other systems and devices that utilize AES encryption. This compatibility facilitates seamless integration into IoT ecosystems and networks. Along with area efficiency, the implementation may also prioritize energy efficiency, minimizing power consumption during cryptographic operations. This aspect is crucial for resource-constrained devices powered by batteries or energy harvesting mechanisms, extending their operational lifespan.

## References

[1] Amini et al., A focus on secure and power-efficient implementations of nano-AES for resource-constrained IoT devices (2021).
[2] Chodowiec et al., An FPGA-based design for AES encryption tailored for IoT applications (2022).
[3] Ghosh et al., Optimizing the AES cryptosystem for energy and area efficiency with relation to IoT devices (2021).
[4] Zhe et al., A compact and efficient hardware design for implementing AES on resource-limited IoT devices (2020).
[5] Kaur & Singh, Exploring low-power AES encryption using FPGAs about IoT applications (2020).
[6] Son et al., Research on achieving compactness and speed in AES for resource-constrained IoT devices (2019).
[7] Thapliyal & Saxena, Designing an ultra-low power AES crypto-processor specifically about IoT applications (2018).
[8] Yan et al., A lightweight AES encryption scheme proposed about IoT applications (2021).
[9] Singh & Sharma, Designing an energy and area-efficient AES S-box suitable for IoT gadgets (2021).
[10] Khrais & Al-Ali, An area-efficient implementation of an AES encryption core for IoT gadgets (2020).
[11] Kim & Lee, Energy-efficient design considerations for AES cryptosystems in IoT devices (2020).
[12] Patel & Patel, Designing lightweight AES encryption for limited resources IoT devices (2019).
[13] Hu et al., A lightweight AES encryption algorithm design and implementation for IoT security systems (2018).