cryptography

MDPI

Article

Flexible and Area-Efficient Codesign Implementation of AES on FPGA

Oussama Azzouzi [1,2], Mohamed Anane [2], Mohamed Chahine Ghanem [3,4,*], Yassine Himeur [5] and Dominik Wojtczak [3]

[1] Centre Universitaire El Cherif Bouchoucha Aflou, Aflou 03001, Algeria; o_azzouzi@esi.dz
[2] Laboratory of System Design Methods, National Higher School of Computer Science, BP 68M, Algiers 16309, Algeria; m_anane@esi.dz
[3] Cybersecurity Institute, School of Computer Science and Informatics, University of Liverpool, Liverpool L693BX, UK; d.wojtczak@liverpool.ac.uk
[4] Cyber Security Research Centre, London Metropolitan University, London N78DB, UK
[5] College of Engineering and Information Technology, University of Dubai, Dubai 14143, United Arab Emirates
* Correspondence: mohamed.chahine.ghanem@liverpool.ac.uk

Abstract

As embedded and IoT systems demand secure and compact encryption, developing cryptographic solutions that are both lightweight and efficient remains a major challenge. Many existing AES implementations either lack flexibility or consume excessive hardware resources. This paper presents an area-efficient and flexible AES-128 implementation based on a hardware/software (HW/SW) co-design, specifically optimized for platforms with limited hardware resources, resulting in reduced power consumption. In this approach, key expansion is performed in software on a lightweight MicroBlaze processor, while encryption and decryption are accelerated by dedicated hardware IP cores optimized at the Look-up Table (LuT) level. The design is implemented on a Xilinx XC5VLX50T Virtex-5 FPGA, synthesized using Xilinx ISE 14.7, and tested at a 100 MHz system clock. It achieves a throughput of 13.3 Gbps and an area efficiency of 5.44 Gbps per slice, requiring only 2303 logic slices and 7 BRAMs on a Xilinx FPGA. It is particularly well-suited for resource-constrained applications such as IoT nodes, secure mobile devices, and smart cards. Since key expansion is executed only once per session, the runtime is dominated by AES core operations, enabling efficient processing of large data volumes. Although the present implementation targets AES-128, the HW/SW partitioning allows straightforward extension to AES-192 and AES-256 by modifying only the software Key expansion module, ensuring practical scalability with no hardware changes. Moreover, the architecture offers a balanced trade-off between performance, flexibility and resource utilization without relying on complex pipelining. Experimental results demonstrate the effectiveness and flexibility of the proposed lightweight design.

Keywords: AES; encryption; FPGA; codesign; LuT; key expansion; MicroBlaze

Received: 18 October 2025
Revised: 21 November 2025
Accepted: 26 November 2025
Published: 1 December 2025

Citation: Azzouzi, O.; Anane, M.; Ghanem, M.C.; Himeur, Y.; Wojtczak, D. Flexible and Area-Efficient Codesign Implementation of AES on FPGA. *Cryptography* **2025**, *9*, 78. https://doi.org/10.3390/cryptography9040078

1. Introduction

The rapid growth of connected devices, digital services, and cloud-based platforms has significantly increased the demand for secure and energy-efficient cryptographic solutions [1]. This need is especially pressing in embedded systems and the Internet of Things (IoT), where devices must operate under strict limitations in terms of hardware resources,

processing power, and energy consumption. Ensuring robust data protection while meeting these constraints has become a key challenge for researchers and system designers [2,3].

Security is a critical concern in resource-constrained embedded and IoT environments, where cryptographic operations can be vulnerable to side-channel and memory-based attacks. Effective countermeasures include secure key storage, masking, constant-time operations, and memory protection mechanisms to reduce leakage and mitigate potential vulnerabilities. Recent research has explored these aspects in depth: efficient implementations of supersingular isogeny-based Diffie–Hellman key exchange on ARM processors highlight techniques for protecting sensitive computations from side-channel exposure [4]; gamified hardware security education demonstrates practical approaches for teaching secure design and evaluation of cryptographic circuits [5]; and high-performance FPGA implementations of modular arithmetic for schemes like ML-DSA and ML-KEM show methods to maximize throughput while maintaining security through careful handling of intermediate data [6]. Collectively, these studies illustrate the importance of integrating security-aware design practices alongside performance optimization in modern cryptographic systems.

Among the many encryption methods, the Advanced Encryption Standard (AES) [7] standardized by the National Institute of Standards and Technology (NIST) remains the most widely used symmetric key algorithm. AES is valued for its strong resistance to cryptanalysis, its high computational efficiency, and its suitability for a wide range of applications, from personal devices and financial transactions to defense and secure communications.

AES implementations [8] generally fall into two categories: software-based and hardware-based. Software implementations [9] are relatively simple, portable, and easier to update, but they usually provide limited performance and are more vulnerable to physical and side-channel attacks. Hardware implementations [10], on the other hand, deliver higher throughput and lower latency, making them preferable in scenarios that require real-time security, such as mobile communications and defense systems. To improve efficiency, hardware designers often use advanced strategies like pipelining, parallelization, or loop unrolling, though these techniques can significantly increase hardware cost and power consumption [11,12]. A promising solution is the hybrid HW/SW co-design approach [13], where cryptographic tasks are split between hardware accelerators and lightweight software modules. This method offers a good balance of flexibility, energy efficiency, and performance, which is especially important in embedded systems with strict limits on chip area and battery life.

Field-Programmable Gate Arrays (FPGAs) [14] are particularly attractive for AES implementation because they are reconfigurable, support custom hardware acceleration, and offer a good compromise between performance and design flexibility. Previous FPGA-based AES designs have typically focused on either resource efficiency or maximum throughput. For example, resource-conscious designs [15,16] often use iterative architectures that reuse the same hardware across multiple rounds of AES, significantly reducing area usage but at the expense of speed. In contrast, performance-oriented designs rely on deep pipelining and fully unrolled loops to achieve multi-gigabit throughput, but this comes with high area and energy costs, limiting their scalability in low-power systems.

Although many high-performance AES architectures exist, they often use complex pipelining and large hardware resources, making them unsuitable for compact embedded systems. To overcome this, we present a hybrid HW/SW AES-128 design tailored for resource-constrained platforms. The design uses a MicroBlaze soft-core processor for key expansion, reducing hardware cost and adding flexibility, while encryption and decryption are handled by optimized hardware IP cores. By avoiding pipelining and hardware

duplication, the architecture lowers complexity and power use. Our implementation achieves 13.3 Gbps throughput with only 2303 logic slices and 7 BRAMs on an XC5VLX50T FPGA. With its small footprint, scalability to different key sizes, and energy efficiency, it is well-suited for restricted environment applications such as smart cards, secure IoT devices, and portable encryption modules. Since key expansion is carried out only once per session, its cost becomes negligible for large data volumes, making the runtime dominated by AES IP core operations and ensuring high efficiency for continuous encryption and decryption in embedded and IoT systems. The main contributions are as follows:

- A hybrid HW/SW AES-128 architecture that balances performance and flexibility in resource-limited systems.
- An optimized hardware AES core at the LuT level, with no need for deep pipelining, keeping high speed and low area usage.
- A flexible software-based key expansion module on MicroBlaze, enabling scalable key sizes with minimal hardware overhead.
- A comprehensive performance evaluation and comparison with state-of-the-art AES designs, demonstrating an effective trade-off between speed, flexibility and area efficiency.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 gives an overview of the Advanced Encryption Standard. Section 4 details the FPGA-based implementation. Section 5 presents experimental results and comparisons with existing approaches. Finally, Section 6 concludes the paper.

2. Related Works

The implementation of AES on FPGAs has been extensively investigated, with research efforts targeting two major directions: resource efficiency for constrained devices and performance maximization for high-speed applications.

Early compact AES cores were introduced by Satoh et al. [17] and Moradi et al. [18], emphasizing low resource utilization. Their designs consumed a minimal number of slices and block RAMs, making them suitable for low-end and embedded systems, but at the expense of reduced throughput.

To address the need for high-speed cryptography, performance-driven approaches emerged, relying heavily on loop unrolling and deep pipelining. Henzen and Fichtner [12] proposed a deeply pipelined architecture achieving multi-gigabit throughput, while Wang et al. [11] presented a fully unrolled AES design optimized for speed. However, both solutions incurred significant area and power overheads, limiting their applicability in resource-sensitive platforms.

Recent advancements attempt to balance both efficiency and throughput by introducing architectural optimizations. Zhang, Li, and Hu [19] achieved 60.29 Gbps on a Virtex-6 FPGA by merging round functions into compact Look-up Tables (LuTs), leveraging dual-port ROMs, and employing a three-stage pipeline. Zodpe and Sapkal [20] focused on enhancing the S-box and key expansion using a PN Sequence Generator, demonstrating improvements in both throughput and security on Spartan-6 FPGAs. Shahbazi and Ko [21] further advanced the field by designing a high-throughput AES-128 architecture with dual-level pipelining, round reordering, and affine transformations, achieving 79.7 Gbps on a Virtex-5 FPGA while maintaining resource efficiency.

These studies highlight the continuous trade-off between area efficiency and high-speed performance, with recent works exploring hybrid approaches to achieve balanced architectures that can meet the diverse requirements of embedded and high-performance systems. A comparative overview of AES FPGA implementations is provided in Table 1.

Table 1. Comparative Summary of AES FPGA Implementations.

Work	Device	Approach	Key Features	Area (Slice/BRAM)	Remarks
[17]	ASIC	Compact design	Composite-field compact S-box.	5.4 Kgates	Suitable for constrained systems
[18]	ASIC	Very compact AES-128	Resource-optimized core	2400 gates	Targeted at embedded devices
[11]	Virtex-6	Loop unrolling	Fully unrolled AES core	15,612	Large area with competitive speed
[22]	Virtex-6	Optimized FPGA design	LUT-based round transformation; improved S-box and MixColumns implementation	2252, 244	Balanced efficiency and performance
[20]	Spartan-6	Optimized design	PN Sequence Generator, modified S-box	5566	Focused on efficiency and cryptographic strength
[21]	Virtex-5	Hybrid pipeline	Dual-level pipelining, round reordering, affine transformations	5974	High throughput with efficiency

3. Advanced Encryption Standard

The AES is a symmetric key encryption algorithm developed by Joan Daemen and Vincent Rijmen, collectively known as Rijndael. It was introduced as part of a competition initiated by the NIST in 1997 and was officially adopted in November 2001 as the encryption standard for U.S. government agencies.

AES employs a block cipher mechanism, where both encryption and decryption operate on fixed-size blocks of 128 bits. It supports secret keys of three different lengths: 128, 192, or 256 bits. The number of transformation rounds (N_r) applied to the plaintext depends on the key length—10 rounds for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys.

Despite the variation in key lengths, the underlying structure of AES-192 and AES-256 closely resembles that of AES-128, differing primarily in the number of processing rounds. In all cases, the plaintext is segmented into 128-bit blocks, which are then represented as 4×4 byte matrices referred to as the *State*. Similarly, the encryption key is arranged into a 4×4 matrix known as the *Key*. Each element of these matrices corresponds to a byte in the Galois Field $GF(2^8)$ and is typically expressed in hexadecimal format [7].

The encryption process begins with an initial transformation known as *AddRoundKey*, where the plaintext is xored with the initial key. This is followed by a series of transformation rounds—each consisting of four core operations: *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. These operations are repeated for each round, with a specific subkey generated via a key expansion procedure. In the final round of AES, the *MixColumns* step is omitted to ensure invertibility during decryption. An overview of the AES encryption and decryption process is illustrated in Figure 1.

In AES encryption, each round uses four main operations. These steps ensure two important security properties: confusion, which makes it hard to see the link between the key and the ciphertext (done mainly with the S-box), and diffusion, which spreads each input bit across many output bits (using ShiftRows and MixColumns). Together, they make it very difficult for attackers to detect patterns or guess the key.

1. **SubBytes**: This is a nonlinear substitution step in the Galois Field $GF(2^8)$, where each byte of the State matrix is replaced using a fixed and reversible substitution box (S-box). It introduces nonlinearity and is responsible for approximately 75% of the algorithm's security strength.
2. **ShiftRows**: A transposition step that operates on the rows of the State matrix. The first row remains unchanged, while the second, third, and fourth rows are cyclically shifted

to the left by 1, 2, and 3 bytes, respectively. This enhances diffusion by rearranging the byte positions.

3. **MixColumns**: A column-wise mixing transformation that performs matrix multiplication in $GF(2^8)$. Each column is treated as a four-term polynomial and multiplied by a fixed polynomial modulo $x^4 + 1$. Although it contributes about 15% to AES's security, it is the most computationally demanding operation due to its arithmetic complexity.
4. **AddRoundKey**: A bitwise xor operation that combines the current State matrix with a round-specific subkey. This operation, performed in $GF(2^8)$, directly incorporates the secret key into the encryption process.

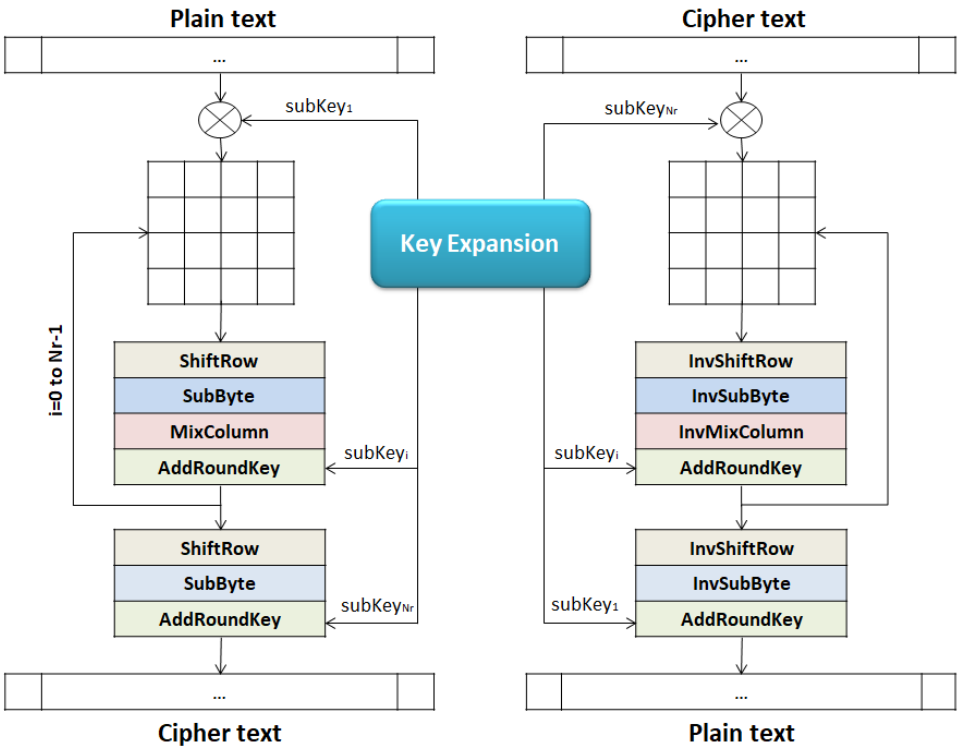

Figure 1. AES Encryption/Decryption Algorithm Block Diagram.

To support these rounds, AES uses a *Key Expansion* process that generates a series of round keys from the original secret key. In AES-128, a total of 11 subkeys are produced, with the first being the original key and the remaining ten used across the subsequent rounds. These subkeys can either be precomputed and stored or dynamically generated during runtime.

Decryption in AES essentially reverses the encryption steps, with slight modifications to each transformation:

- **Inverse SubBytes**: Uses the inverse S-box to revert the nonlinear substitution.
- **Inverse ShiftRows**: Performs circular shifts to the right to restore the original byte positions.
- **Inverse MixColumns**: Applies an inverse matrix transformation in $GF(2^8)$ to undo the column mixing.

The next section presents the proposed hardware/software co-design methodology for implementing AES-128 on FPGA, emphasizing efficiency and scalability in resource-constrained environments.

4. FPGA Implementation

4.1. Proposed Method

AES implementations on FPGA platforms generally follow one of three architectural paradigms. The first is a purely software-based design, where all encryption and decryption operations are implemented in high-level languages such as C and executed on an embedded processor. The second approach adopts a purely hardware-based solution, with all cryptographic functions mapped to dedicated logic components. The third, and increasingly preferred, method is a hybrid HW/SW co-design, which partitions the cryptographic workload between hardware and software modules to balance performance, flexibility, and resource efficiency.

The proposed design adopts the HW/SW co-design strategy and targets a flexible embedded cryptographic system. At the core of this system is the MicroBlaze processor—a 32-bit soft-core RISC CPU provided by Xilinx as an Intellectual Property (IP) block within the Embedded Development Kit (EDK) library. The MicroBlaze processor offers extensive configurability in terms of pipeline structure, memory architecture, and peripheral integration. This makes it particularly well-suited for embedded applications that require a fine balance between performance, power consumption, and hardware utilization.

In the proposed architecture, the AES Key Expansion process is implemented in software and executed on the MicroBlaze processor, while the core encryption and decryption functions are realized in hardware using VHDL. This separation of functionality allows for an efficient use of system resources while maintaining high processing throughput. The overall architecture of the system is illustrated in Figure 2.

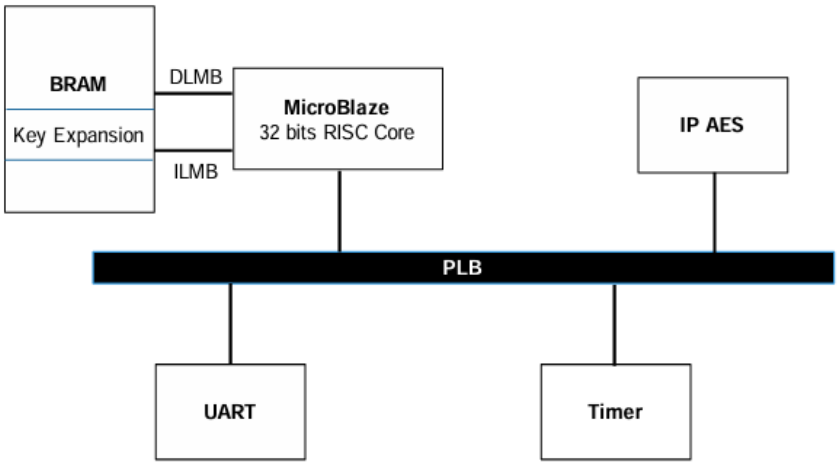

Figure 2. The hardware architecture of the proposed system.

The rationale behind this HW/SW partitioning is based on the distinct computational characteristics of AES operations:

- **Key Expansion**: This operation is performed only once at the beginning of the encryption or decryption process. It involves generating a set of subkeys from the main encryption key, which are then stored in memory. Implementing this process in software facilitates easy adaptation to different key lengths (128, 192, or 256 bits) without requiring additional hardware resources or redesign.
- **AES Rounds (Encryption/Decryption)**: These operations involve repetitive transformations on 128-bit data blocks and must be applied to every block of input data. For applications dealing with large volumes of data, performance becomes critical. Offloading these operations to hardware significantly accelerates processing, as hardware modules execute these computations much faster than software running on the MicroBlaze processor.

Overall, the proposed system leverages the flexibility of software for key scheduling while exploiting the speed and parallelism of hardware for data path processing. This strategic partitioning results in a highly efficient architecture optimized for performance, scalability, and low resource consumption, making it ideal for embedded environments that demand both cryptographic strength and implementation efficiency.

4.2. Key Expansion Implementation

The key expansion process is implemented in software using the MicroBlaze processor. Written in C, this module generates the round subkeys required for the AES encryption and decryption stages. These subkeys are stored in a 128×11 bit RAM structure within the AES IP core, where each 128-bit subkey corresponds to one encryption round. The original key is used during the initial round, while the remaining subkeys are applied in subsequent rounds. The key expansion procedure follows the standard AES specification and is detailed in Algorithm 1.

Algorithm 1: Key Expansion Algorithm

Data: N_r // Number of AES rounds.
N_k // Number of 32-bit words in the key.
N_b // Number of 32-bit words in the state (=4).
Key [32] // The original encryption key.
Rcon [255] // Round constants.
Variables: W[4]
Result: RoundKey[240] // Array of generated subkeys.
Begin
for $i \leftarrow 0$ **to** N_k **do**
RoundKey$[i*4] =$ Key$[i*4]$;
RoundKey$[i*4+1] =$ Key$[i*4+1]$;
RoundKey$[i*4+2] =$ Key$[i*4+2]$;
RoundKey$[i*4+3] =$ Key$[i*4+3]$;
end
while $(i < (N_b * (N_r + 1)))$ **do**
for $j \leftarrow 0$ **to** 4 **do**
$W[j] =$ RoundKey $[(i-1)*4+j]$;
end
if $i\%N_k == 0$ **then**
RotWord(W); // Circular byte-wise rotation.
Subword(W); // Byte-wise substitution using S-Box.
AddRoundKey(W, Rcon); // XOR with round constant.
end
RoundKey $[i*4+0] =$ RoundKey$[(i-N_k)*4+0] \oplus$ W[0];
RoundKey $[i*4+1] =$ RoundKey$[(i-N_k)*4+1] \oplus$ W[1];
RoundKey $[i*4+2] =$ RoundKey$[(i-N_k)*4+2] \oplus$ W[2];
RoundKey $[i*4+3] =$ RoundKey$[(i-N_k)*4+3] \oplus$ W[3];
$i++$;
end
End

4.3. Encryption/Decryption Implementation

The encryption and decryption modules are implemented in hardware as a custom IP core. The design employs an iterative looping architecture that allows the system to process four 32-bit AES data blocks in parallel, while executing the round operations serially. This approach achieves an effective balance between parallelism and resource efficiency. An overview of the encryption architecture is depicted in Figure 3.

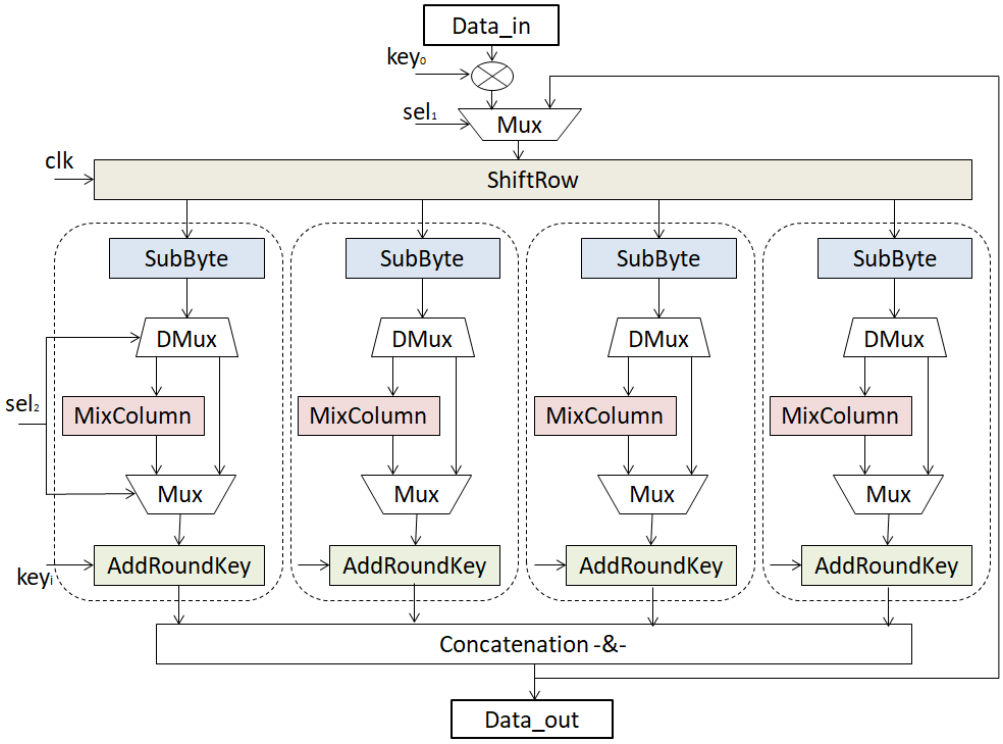

Figure 3. AES Encryption Architecture.

During encryption, the 128-bit State matrix goes step by step through the AES stages. The *ShiftRows* operation keeps temporary results in a separate unit, while the *SubBytes* step uses four ROM-based S-Boxes filled with precomputed values. The most demanding step, *MixColumns*, is optimized on the FPGA with low-level logic and LuTs. Our method in [23] provides a smart hardware design for both MixColumn and InvMixColumn by using LuT-based logic and function decomposition. This reduces delays on the critical path, improves speed and energy efficiency, and ensures that all operations run smoothly with minimal hardware cost.

In the architecture shown in Figure 3, a multiplexer/demultiplexer is incorporated within the AES block to enable efficient resource sharing. It is important to note that the *MixColumns* operation is intentionally excluded from the final encryption round, following the AES specification.

A more detailed view of the AES hardware structure is provided in Figure 4, illustrating the integration of the encryption and decryption blocks, a centralized control circuit, and a dedicated memory module (MEM_Key).

The control circuit coordinates all cryptographic operations, ensuring synchronization across the AES pipeline. It also generates 4-bit address signals to retrieve the appropriate subkeys from MEM_Key during each encryption or decryption round.

In decryption mode, the same operations are applied in reverse order. The primary difference lies in the application of subkeys, which are used in reverse sequence—from the final round to the initial round. To optimize memory usage and avoid redundancy, the MEM_Key module is shared between the encryption and decryption paths.

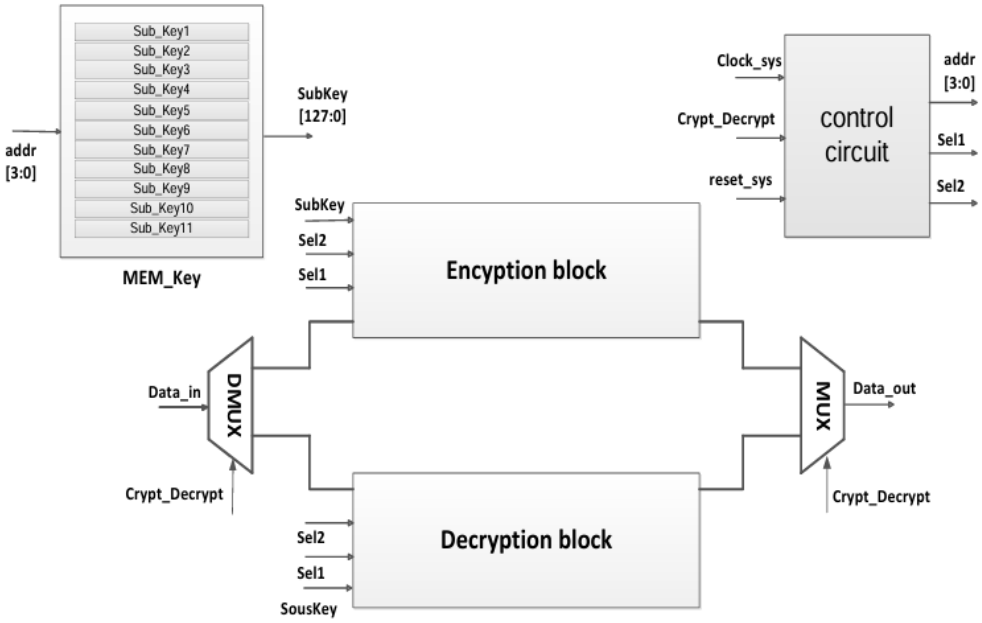

Figure 4. AES IP Hardware Implementation.

4.4. AES IP Integration with MicroBlaze

The integration of the AES IP core with the MicroBlaze processor is illustrated in Figure 5. To enable effective communication between the custom AES hardware and the processor, an interface layer is introduced to connect the AES core to the Processor Local Bus (PLB). This interface is built using a *User_Logic* module, which leverages the Intellectual Property Interface (IPIF) standard provided by Xilinx [15]. The IPIF serves as a bridge between the AES IP and the system bus, facilitating seamless data and instruction exchange.

The IPIF module decodes the PLB communication protocol and provides internal components such as the following:

- `Din_reg` —A data input register for receiving 32-bit data from the MicroBlaze.
- `Dout_reg`—A data output register for sending processed results back to the processor.
- An instruction register—Used to interpret control commands issued by the processor.

The *User_Logic* block is structured into three primary units:

1. **MEM_Key**: A memory unit responsible for storing the subkeys generated during the Key Expansion phase.
2. **Control Circuit**: Manages synchronization and control signals for AES operations.
3. **AES Core**: Executes the core encryption and decryption processes.

In addition to overseeing synchronization, the control circuit is responsible for the following key operations:

- Concatenating four incoming 32-bit words into a single 128-bit data block.
- Generating address signals for read and write operations in the MEM_Key unit.
- Splitting 128-bit encrypted or decrypted outputs into four 32-bit segments for transmission over the PLB bus.

Data transfers between the processor and the AES IP are handled via control signals generated by the IPIF interface, which include:

- `Bus2IP_Clk`: Clock signal used to synchronize the PLB bus and the AES IP.
- `Din`: Data signal carrying input values from the MicroBlaze to the AES core.
- `Dout`: Data signal used to return processed output from the AES core to the processor.
- `Bus2IP_WrCE`: Write enable signal indicating a data write request from the processor.
- `Bus2IP_RdCE`: Read enable signal used to retrieve the result of AES processing.

The AES IP is controlled through a predefined set of processor-issued instructions. Each instruction corresponds to a specific operation within the cryptosystem and is encoded in hexadecimal format, as listed in Table 2.

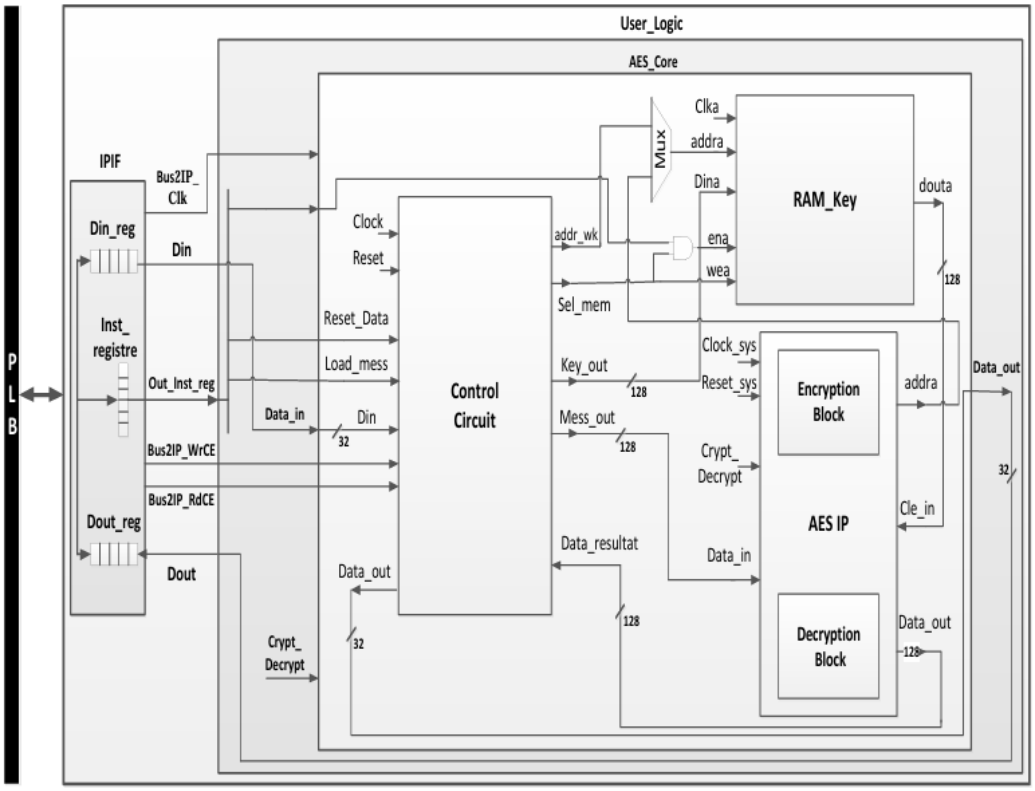

Figure 5. AES Hardware Architecture Integrated with MicroBlaze.

Table 2. Instruction Set for AES IP Control.

Instruction	Hex Code	Description
`run_write_key`	0x80000000	Transfers the round subkeys from software to MEM_Key.
`Reset_ip_aes`	0x80000001	Resets and initializes the AES IP for a new encryption/decryption session.
`Run_write_mess`	0x00000000	Loads the plaintext or ciphertext message into the AES input buffer.
`Run_aes`	0x04000000	Initiates the AES encryption or decryption process.

5. Implementation and Result

The HW/SW AES implementation was developed with VHDL for the custom AES IP core and C for the Key Expansion module using the Xilinx SDK compiler on a MicroBlaze soft-core processor within the EDK environment. Functional correctness was validated through ModelSim and ISim simulations, and the design was synthesized and deployed on a Xilinx XC5VLX50T Virtex-5 Genesys FPGA using ISE Design Suite (version 14.7).

Post-synthesis and post-routing analyses were performed to verify maximum operating frequencies and timing, using a 100 MHz clock for fair comparison with previous works. The Virtex-5 Genesys FPGA provides high logic density, speed, and energy efficiency, featuring CLBs with LuTs and flip-flops, DSP48E slices, Block RAMs, and Clock Management Tiles. Its versatile I/O interfaces (DDR2, USB, Ethernet, VGA) make it ideal for research and prototype development, fully supported by the Xilinx ISE toolchain for

synthesis, simulation, and timing analysis. A summary of the implementation results is provided in Table 3.

It is important to note that the reported resource utilization in Table 3 includes both the MicroBlaze soft-core processor and the custom AES IP core. Specifically, the MicroBlaze consumes 1063 slices and 7 BRAMs, while the remaining resources are attributed to the AES hardware module.

Table 3. Hardware Performance Summary.

Device		Virtex XC5VLX50T
Slice Utilization		2303/7200 (31%)
BRAM Usage		7/60 (11%)
Minimum Clock Period (ns)		9.61
Maximum Frequency (MHz)		103.97
Latency (cycles)	Key Expansion	28,577
	AES IP	172
	Total	30,180
Execution Time (µs)	Key Expansion	285.77
	AES IP	1.72
	Total	301.8

The total execution time is defined as follows:

$$T_{\text{global}} = T_{\text{Key\_Expansion}} + T_{\text{AES\_IP}}.$$

The key expansion step is carried out only once at the start of a session, and the generated round keys are reused throughout the encryption or decryption process. This means that when large amounts of 128-bit data blocks are processed, the cost of key expansion becomes negligible compared to the overall computation. As a result, the runtime is mainly determined by the AES core operations rather than the key setup phase. This makes the proposed AES IP particularly efficient for continuous encryption and decryption tasks, such as streaming data, secure communications, or storage encryption, where large data volumes are processed in embedded and IoT systems.

To evaluate the proposed design, several key hardware performance metrics are considered: maximum operating frequency (F_{max}), area utilization, throughput, and efficiency. These indicators collectively provide insight into the design's computational effectiveness and hardware resource optimization.

Throughput, which measures the data rate (in Gbps), is computed using the following formula:

$$\text{Throughput} = \frac{\text{Bitwidth} \times F_{\text{max}}}{\text{Latency}}.$$

In this work, the bitwidth is set to 128 bits. Consistent with prior studies [11,20], latency is omitted from the effective throughput calculation to ensure fair benchmarking. This choice can be justified by the streaming-oriented nature of the intended IoT and mobile use cases. In such scenarios, where encrypted data is transmitted as continuous flows (e.g., sensor telemetry or multimedia streams), overall system performance is governed mainly by sustained throughput rather than block-level latency.

Efficiency is calculated as the ratio of throughput to hardware resource usage, expressed as follows:

$$\text{Efficiency} = \frac{\text{Throughput (Mbps)}}{\text{Area (Slice Equivalents)}},$$

where 1 BRAM is approximated to 20 logic slices for normalization.

Table 4 provides a detailed benchmark of the proposed HW/SW AES architecture against representative state-of-the-art implementations across different FPGA families. The comparison underlines the inherent trade-offs between throughput, hardware cost, and efficiency that arise from distinct architectural choices such as pipelining, unrolling, and hybrid co-design.

Table 4. Performance Comparison with State-of-the-Art AES Implementations.

Ref.	Device	Design	Bitwidth (bits)	Slice, BRAM	F_{max} (MHz)	Throughput (Gbps)	Efficiency (Mbps/Slice)
This work	XC5VLX50T	HW/SW (Non-Pipelined)	128	2303, 7	103.97	13.3	5.44
[24]	XC6VLX240T	HW/SW (Non-Pipelined)	128	12,611	100.66	-	-
[11]	XC6VLX240T	HW (Non-Pipelined)	128	15,612	14.69	1.88	0.11
		HW (Pipelined)	128	9071, 400	319.29	40.86	6.89
[22]	XC6VLX240T	HW (Pipelined)	128	2252, 244	470.99	60.28	8.45
[20]	XC4VLX60	HW (Non-Pipelined)	128	20,818	214.48	27.45	1.31
	XC5VSX240T	HW (Non-Pipelined)	128	3420	199.18	25.49	7.45
	XC6SLX150	HW (Non-Pipelined)	128	5566	237.45	30.39	5.46
[21]	XC5VLX85	HW (Pipelined)	128	5974	622.4	79.66	13.33

The proposed design achieves a throughput of 13.3 Gbps with only 2303 slices and 7 BRAMs, yielding an efficiency of 5.44 Mbps/Slice. Unlike fully pipelined designs, our approach avoids significant area and memory overheads while still maintaining reasonable throughput, making it particularly attractive for resource-constrained embedded and IoT systems.

MicroBlaze-based implementations, such as Renuka et al. [24], demonstrate the feasibility of HW/SW co-design for AES. However, their design consumes 12,611 slices on a Virtex-6, considerably more than our proposed implementation's 2303 slices on a Virtex-5, making it less suitable for resource-constrained environments.

Looking at the work of Wang et al. [11], the contrast between pipelined and non-pipelined strategies is clear. Their non-pipelined version consumes more than 15k slices while delivering only 1.88 Gbps, resulting in very poor efficiency (0.11 Mbps/Slice). Conversely, their pipelined design boosts throughput to 40.86 Gbps, but still requires 9071 slices and 400 BRAMs, a prohibitive cost for compact platforms. In this context, our design demonstrates that careful co-design and selective hardware/software partitioning can achieve a much more favorable trade-off, with efficiency nearly 50× higher than their non-pipelined core, and without the memory burden of their pipelined solution.

Similarly, the fully unrolled and double-pipelined design in Zhang et al. [22] represents a high-throughput extreme, reaching 60.28 Gbps at nearly 471 MHz. However, this comes at the cost of 244 BRAMs, which heavily constrains its applicability in embedded systems where memory blocks are scarce and must often be shared with other IP cores. By comparison, our solution reduces BRAM usage by a factor of 35×, while still achieving double-digit throughput, underscoring its compactness and suitability for low-power platforms.

The implementations of Zodpe et al. [20] also emphasize high-speed operation, reporting throughputs of 25–30 Gbps across Virtex-4, Virtex-5, and Spartan-6 devices. However,

these results require 3 k–20 k slices, which severely impacts scalability when integrating AES into larger SoCs. Although their Virtex-5 variant reaches an efficiency of 7.45 Mbps/Slice, slightly above ours, it relies on a significantly larger device class. Our design, on the other hand, preserves a balanced performance-to-area ratio that can be more easily adapted to mid-range or cost-sensitive FPGAs.

Finally, Shahbazi et al. [21] report the highest throughput among the compared works (79.66 Gbps) at a frequency exceeding 622 MHz. While highly impressive, this design leverages deep pipelining and logic duplication, resulting in 5974 slices. Such aggressive optimization is less compatible with embedded applications, where minimizing complexity and power consumption is often prioritized over peak performance. By contrast, the proposed HW/SW non-pipelined implementation emphasizes simplicity, reduced area, and memory efficiency, which are critical factors in embedded applications.

In summary, the comparative analysis in Table 5 shows that architectural design is the key factor influencing trade-offs in AES FPGA implementations. Fully unrolled and deeply pipelined designs (e.g., Zhang et al. [22]; Shahbazi and Ko [21]) achieve the highest throughput, but require excessive slices and BRAM, making them unsuitable for cost- and energy-constrained platforms. Masked or side-channel-hardened implementations (e.g., Wang and Ha [11]; Moradi et al. [18]) focus on security and compactness, but typically reduce throughput or depend on ASIC-specific features. Efficiency-oriented solutions, such as those by Zodpe and Sapkal [20], apply S-box and key-expansion optimizations, yet their results remain moderate and device-dependent.

Table 5. Comparative summary of representative AES FPGA implementations.

Reference	Year	Device	Architecture/Key Features	Slices/BRAM	F_{max} (MHz)	Throughput (Gbps)	Efficiency (Mbps/Slice)
This Work	2025	XC5VLX50T	HW/SW co-design (MicroBlaze key-expansion; LuT-optimized AES core); non-pipelined, low-area design. *(Target: resource-constrained IoT/embedded systems).*	2303/7	103.97	13.3	5.44
Wang and Ha (2013) [11]	2013	XC6-series (Virtex-6)	Masked AES with area optimizations; non-pipelined and pipelined implementations for high throughput.	15612	14.69	1.88	0.11
Zhang, Li and Hu [22]	2018	Virtex-6 (XC6VLX240T)	Optimized AES FPGA design; improved round-based architecture with resource reduction strategies.	2252, 244	470.99	60.29	8.45
Zodpe and Sapkal (2020) [20]	2020	Spartan-6/assorted	S-box and key-expansion optimizations (PN-generator S-box); balanced efficiency.	5566	237.45	30.39	5.46
Shahbazi and Ko (2020) [21]	2020	Virtex-5/Virtex-6	High-throughput dual-level pipelining, round reordering, affine transformations.	5974	622.4	79.66	13.33
Moradi et al. (2011) [18]	2011	ASIC/FPGA	Very compact AES hardware and threshold implementation for S-box (side-channel resistance).	2400 GE (ASIC)	n/a	n/a	n/a

By contrast, the proposed HW/SW co-design follows a minimalist approach, combining a non-pipelined AES datapath with a LuT-optimized core and MicroBlaze-based key expansion. This avoids the high resource usage of unrolled pipelines while maintaining reasonable throughput. With only 2303 slices and negligible BRAM, the design achieves an efficiency of 5.44 Mbps/slice, surpassing most related works. Overall, it provides a balanced solution at the intersection of throughput, area efficiency, and scalability, making it well-suited for embedded, IoT, and mobile platforms where compactness and energy efficiency are more critical than maximum speed.

Although the comparative evaluation demonstrates clear performance and area advantages, the analysis remains focused primarily on throughput and resource utilization. A more comprehensive assessment would incorporate additional system-level metrics such as power consumption, energy per bit and latency profiling—factors that are particularly relevant to embedded and IoT deployments. Moreover, examining scalability to AES-192/256, conducting security-oriented evaluations (e.g., side-channel exposure on the MicroBlaze), and comparing cost-efficiency across multiple FPGA families would provide a deeper and more complete characterization. These extensions are left for future work to further reinforce the robustness of the experimental study.

6. Ethical Considerations

This paper presents a compact, area-efficient HW/SW AES-128 implementation tailored for resource-constrained embedded platforms. The proposed hybrid architecture assigns key expansion to a MicroBlaze soft-core processor and encryption/decryption to optimized hardware IP cores, eliminating deep pipelining and replication. This design achieves 13.3 Gbps throughput with high area efficiency (5.44 Gbps/slice) while minimizing power and resource usage. Its scalability, adaptability to different key sizes, and efficient-area make it well-suited for secure IoT devices, smart cards, and portable encryption modules.

7. Conclusions

This paper presented a high-efficiency HW/SW co-design for AES-128 encryption, specifically tailored for embedded platforms with strict resource and power constraints. The proposed architecture strategically delegates the key expansion process to a lightweight MicroBlaze soft-core processor, while executing the core encryption and decryption tasks in dedicated hardware modules optimized at the LuT level. This partitioning enables a streamlined design that avoids the complexity and overhead of pipelining or hardware duplication.

Implemented on a Xilinx XC5VLX50T FPGA, the system achieves a throughput of 13.3 Gbps and an area efficiency of 5.44 Gbps per slice, using only 2303 slices and 7 BRAMs. These results demonstrate that the proposed design offers a strong trade-off between performance and resource utilization, making it a viable solution for a wide range of embedded security applications, including IoT devices, smart cards, and edge computing nodes.

In comparison with existing state-of-the-art AES implementations, the proposed approach is distinguished by its compact footprint, architectural simplicity, and adaptability to different key sizes. Furthermore, its non-pipelined structure contributes to reduced power consumption and easier integration into lightweight embedded systems. While this work does not include explicit power measurements, the compact HW/SW co-design and minimized resource usage inherently suggest potential energy efficiency. A detailed power analysis using tools such as Xilinx XPower Analyzer or hardware measurements is planned for future work to quantify these benefits.

Future work will focus on enhancing both the security and scalability of the HW/SW AES co-design. Since key expansion is performed in software, there is a potential risk of side-channel and memory-based attacks, particularly in IoT environments where the processor may share resources with other tasks. Investigating lightweight countermeasures, secure key management techniques, and memory-protection strategies will be essential to further strengthen the system's resilience against such vulnerabilities. Additionally, extending support for AES modes such as CBC, GCM, and XTS, enabling dynamic partial reconfiguration for real-time adaptation of cryptographic parameters, and exploring the

extension of the HW/SW co-design framework to other lightweight symmetric algorithms and post-quantum cryptographic primitives represent promising directions to enhance the versatility, scalability, and robustness of the proposed methodology.

Author Contributions: Conceptualization, O.A., M.A., M.C.G., Y.H. and D.W.; Methodology, O.A., M.A., M.C.G., Y.H. and D.W.; Software, O.A., M.A., Y.H. and D.W.; Validation, O.A., M.A., Y.H. and D.W.; Formal Analysis, O.A., M.A., M.C.G., Y.H. and D.W.; Investigation, O.A., M.A., M.C.G., Y.H. and D.W.; Resources, O.A., M.A., M.C.G., Y.H. and D.W.; Data Curation, O.A., M.A., M.C.G., Y.H. and D.W.; Writing—Original Draft, O.A., M.A., M.C.G., Y.H. and D.W.; Writing—Review and Editing, O.A., M.A., M.C.G., Y.H. and D.W.; Visualization, O.A., M.A., M.C.G., Y.H. and D.W.; Supervision, O.A., M.A., M.C.G., Y.H. and D.W.; Project Administration, O.A., M.A., M.C.G., Y.H. and D.W.; Funding Acquisition, O.A. and D.W. All authors have read and agreed to the published version of the manuscript.

Funding: The Open Access (OA) fee for this paper was funded by the University of Liverpool.

Institutional Review Board Statement: This research was deemed not to require the University's Ethical Committee approval, as it does not fall under any of the cases requiring ethical approval.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Farshadinia, H.; Barati, A.; Barati, H. A secure and energy-efficient architecture in Internet of Things–cloud computing network by enhancing and combining three cryptographic techniques via defining new features, areas, and entities. *J. Supercomput.* **2025**, *81*, 944. [CrossRef]
2. Selvi, M.; Santhosh Kumar, S.; Thangaramya, K.; Abdul Gaffar, H. Energy efficient trust aware secure routing algorithm with attribute-based encryption for wireless sensor networks. *Sci. Rep.* **2025**, *15*, 19724. [CrossRef] [PubMed]
3. Hudda, S.; Haribabu, K. DySwitch: Dynamic Switching to Enable Secure and Energy Efficient Data Communication in Resource Constrained IoT Environment. In Proceedings of the International Conference on Advanced Information Networking and Applications, Barcelona, Spain, 9–11 April 2025; Springer: Berlin/Heidelberg, Germany, 2025; pp. 58–69.
4. Jalali, A.; Azarderakhsh, R.; Mozaffari-Kermani, M. Efficient implementation of supersingular isogeny-based Diffie-Hellman key exchange on ARM. In Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing, Washington, DC, USA, 26–30 June 2017; pp. 1–8.
5. Karam, R.; Katkoori, S.; Mozaffari-Kermani, M. Engaged student learning with gamified labs: A new approach for hardware security education. In Proceedings of the 2023 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Auckland, New Zealand, 28 November–1 December 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–4.
6. Taghavi, B.; Azarderakhsh, R.; Mozaffari Kermani, M. ParallelNTT: Maximizing performance of forward and inverse NTT on FPGA for ML-DSA and ML-KEM. In Proceedings of the Proceedings of the Great Lakes Symposium on VLSI 2025, New Orleans, LA, USA, 30 June–2 July 2025; pp. 372–378.
7. Rijmen, V.; Daemen, J. Advanced encryption standard. *Proc. Fed. Inf. Process. Stand. Publ. Natl. Inst. Stand. Technol.* **2001**, *19*, 22.
8. Hasija, T.; Kaur, A.; Ramkumar, K.; Sharma, S.; Mittal, S.; Singh, B. A survey on performance analysis of different architectures of AES algorithm on FPGA. In *Modern Electronics Devices and Communication Systems: Select Proceedings of MEDCOM 2021*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 39–54.
9. Oussama, A.; Mohamed, A.; Nassim, H. Software implementation of pairing based cryptography on FPGA. In Proceedings of the International Conference on Computer Science and Its Applications, Melbourne, VIC, Australia, 2–5 July 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 102–112.
10. Deshpande, A.M.; Deshpande, M.S.; Kayatanavar, D.N. FPGA implementation of AES encryption and decryption. In Proceedings of the 2009 International Conference on Control, Automation, Communication and Energy Conservation, Erode, India, 4–6 June 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 1–6.
11. Wang, Y.; Ha, Y. FPGA-based 40.9-Gbits/s masked AES with area optimization for storage area network. *IEEE Trans. Circuits Syst. II Express Briefs* **2013**, *60*, 36–40. [CrossRef]
12. Henzen, L.; Fichtner, W. FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications. In Proceedings of the 2010 Proceedings of ESSCIRC, Sevilla, Spain, 14–16 September 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 202–205.

13. Azzouzi, O.; Anane, M.; Koudil, M.; Issad, M.; Himeur, Y. Novel area-efficient and flexible architectures for optimal Ate pairing on FPGA. *J. Supercomput.* **2024**, *80*, 2633–2659. [CrossRef]
14. Farooq, U.; Marrakchi, Z.; Mehrez, H. FPGA Architectures: An Overview. In *Tree-Based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization;* Springer: Berlin/Heidelberg, Germany, 2012; pp. 7–48. [CrossRef]
15. Ling, A.C.; Singh, D.P.; Brown, S.D. FPGA PLB architecture evaluation and area optimization techniques using boolean satisfiability. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2007**, *26*, 1196–1210. [CrossRef]
16. Kumar, T.M.; Reddy, K.S.; Rinaldi, S.; Parameshachari, B.D.; Arunachalam, K. A low area high speed FPGA implementation of AES architecture for cryptography application. *Electronics* **2021**, *10*, 2023. [CrossRef]
17. Satoh, A.; Morioka, S.; Takano, H.; Munetoh, S. A Compact Rijndael Hardware Architecture with S-Box Optimization. In Proceedings of the 2001 International Symposium on Circuits and Systems (ISCAS), Sydney, Australia, 6–9 May 2001; pp. 1–4. [CrossRef]
18. Moradi, A.; Paar, C.; Poschmann, A.; Schellenberg, J.; Shalmani, M.T.M.; Wang, H.; Yamamoto, H. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In Proceedings of the 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), San Diego, CA, USA, 5–6 June 2011; pp. 36–41. [CrossRef]
19. Zhang, Y.; Li, X.; Hu, Y. Fully Unrolled Three-Stage Pipelined AES Implementation on Virtex-6 FPGA. In Proceedings of the International Conference on Advanced Computing, Ho Chi Minh City, Vietnam, 23–25 November 2015.
20. Zodpe, H.; Sapkal, A. An efficient AES implementation using FPGA with enhanced security features. *J. King Saud-Univ.-Eng. Sci.* **2020**, *32*, 115–122. [CrossRef]
21. Shahbazi, K.; Ko, S.B. High throughput and area-efficient FPGA implementation of AES for high-traffic applications. *IET Comput. Digit. Tech.* **2020**, *14*, 344–352. [CrossRef]
22. Zhang, X.; Li, M.; Hu, J. Optimization and implementation of AES algorithm based on FPGA. In Proceedings of the 2018 IEEE 4th International Conference on Computer and Communications (ICCC), Chengdu, China, 7–10 December; IEEE: Piscataway, NJ, USA, 2018; pp. 2704–2709.
23. Azzouzi, O.; Anane, M.; Ghanem, M.C.; Himeur, Y.; Kheddar, H. Efficient Fine-Grained FPGA Optimization for AES MixColumn and InvMixColumn. *Electronics* **2025**, 1–15. [CrossRef]
24. Renuka, G.; Shree, V.U.; Reddy, P.C.S. Comparison of AES and des algorithms implemented on virtex-6 FPGA and Microblaze soft core processor. *Int. J. Electr. Comput. Eng.* **2018**, *8*, 3544. [CrossRef]