FPGA Implementation of the Advanced Encryption Standard Algorithm Using VERILOG-HDL

Pradeepa S C1, Srushti Benner2, Sushmitha C3 ,Syeda Sameena4, Supreeth B A5

1 Department of Electronics and Communication Engineering, JNNCE, Shimoga, India
2 Department of Electronics and Communication Engineering, JNNCE, Shimoga, India
3 Department of Electronics and Communication Engineering, JNNCE, Shimoga, India
4 Department of Electronics and Communication Engineering, JNNCE, Shimoga, India
5 Department of Electronics and Communication Engineering, JNNCE, Shimoga, India

---

Abstract – This paper presents the hardware design and functional verification of an Advanced Encryption Standard (AES-128) encryption system optimized for real-time security applications. The primary objective is to implement a high-speed cryptographic engine on hardware to overcome the processing latency bottlenecks inherent in software-based encryption. Utilizing a modular design methodology in Verilog HDL, each core algorithmic stage—SubBytes, ShiftRows, MixColumns, AddRoundKey, and Key Expansion—was individually engineered and integrated into a unified 10-round pipeline architecture. Stage-specific verification was executed using standalone testbenches, and functional accuracy was validated through behavioral simulation waveform analysis. The experimental results confirm flawless logical transformation of plaintext inputs into secure ciphertext across all modules without any runtime computational errors. By exploiting hardware-level parallel processing frameworks, this architecture successfully minimizes execution times while improving throughput efficiency and cryptographic data diffusion. This modular accelerator design provides a high-performance hardware baseline suited for physical deployment in FPGA and VLSI environments, including secure telecommunication networks, banking transactions, wireless infrastructures, IoT devices, and real-time embedded systems.

Key Words: AES-128, Verilog HDL, VLSI, IOT, FPGA, Cryptographic hardware, Parallel processing, Waveform simulation.

1. Introduction

In today's digital world, secure data transmission has become very important due to the rapid growth of communication networks, cloud computing, and online services. Sensitive information such as banking details, personal data, and confidential documents must be protected from unauthorized access and cyber attacks. Cryptography plays a major role in ensuring data security by converting readable information into an encrypted format that can only be accessed using a secret key. Among various encryption techniques, the Advanced Encryption Standard (AES) is widely used because of its high security, reliability, and fast processing speed.

AES is a symmetric key encryption algorithm developed by the National Institute of Standards and Technology (NIST). It operates on fixed-size data blocks of 128 bits and supports key sizes of 128, 192, and 256 bits. In AES-128, the encryption process is completed in 10 rounds consisting of operations such as SubBytes, ShiftRows, MixColumns, and AddRoundKey. These transformations provide strong confusion and diffusion, making the encrypted data highly secure. Due to its strong resistance against attacks, AES is extensively used in wireless communication, banking systems, military applications, embedded systems, and secure internet communication.

This project focuses on the hardware implementation of AES-128 encryption using Verilog HDL. The design is developed using modular architecture where each AES stage is implemented separately and verified through simulation. Hardware implementation provides faster execution and better performance compared to software-based encryption methods, making it suitable for FPGA and VLSI applications. The project demonstrates secure and efficient encryption of plaintext into ciphertext using

a 128-bit secret key and highlights the importance of digital hardware design in modern cryptographic systems.

2. Literature Review

Here is the formatted literature comparison table tailored for this project manuscript. Extensive academic literature emphasizes the value of hardware-based cryptographic accelerators over general-purpose computing platforms:

- **Kouser et al. (2022)** validated an FPGA-centered implementation of AES focused on minimizing area and maximizing hardware logic utilization efficiency. Their architecture leveraged parallel structures to boost overall processing speed.
- **Srinivas & Akramuddin (2023)** developed a unified FPGA configuration supporting both encryption and decryption pipelines inside a shared hardware floorplan, establishing notable throughput gains.
- **Shastry et al. (2023)** introduced an Application-Specific Integrated Circuit (ASIC) approach to AES ciphers. Although ASIC implementations reduce active power budgets compared to programmable FPGAs, they suffer from high engineering costs and zero post-fabrication versatility.
- **Hoang & Nguyen (2024)** focused on structural optimization for real-time applications. Their implementation utilized highly optimized pipeline stages to lower processing latency within standard FPGAfabrics.
- **Katkade & Phade (2025)** combined an AES engine directly with serial interfaces (such as UART) to secure active streams of point-to-point data. This design highlights the real-world demand for inline, hardware-level encryption modules.

Table -1: Literature Survey

Research Author(s)	Platform	Core Practical Contributions	Primary Foundational Limitations
Kouser et al. (2022)	FPGA	Optimized multi-channel resource distribution frameworks.	Logic dependencies restricted cross-platform scaling.
Sai Srinivas & Akramuddin (2023)	FPGA	Shared logic blocks execute dual enc/dec operations.	Complex routing raised overall latency metrics.
Shastry et al. (2023)	ASIC	Drastically cut power demands via cellular tuning.	Upfront prototyping costs remain extremely restrictive.
Hoang & Nguyen (2024)	FPGA	Mitigated delay times inside critical path pathways.	Routing complexity spikes under larger key sizes.
Katkade & Phade (2025)	Serial	Successfully fortified low-bandwidth serial line transmissions.	Protocol translation injects additional interface overhead.

3. Proposed AES Architecture and Methodology

The AES-128 processing module handles a state matrix composed of a 4×4 array of bytes. The execution pipeline maps plaintext into a series of modular matrix transformations.

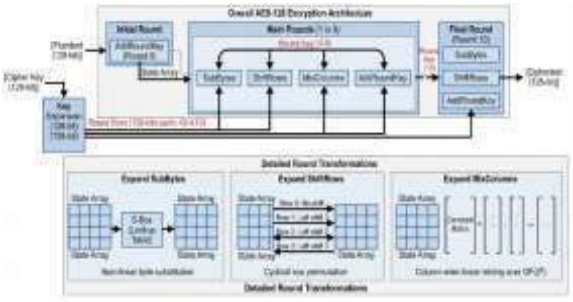


Fig -1: Hierarchal block diagram of AES-128 bit process.

3.1 Initial AddRoundKey

The encryption cycle initiates with a standalone AddRoundKey block. The 128-bit original plaintext is blended with the primary 128-bit secret key through bitwise exclusive-OR (XOR) operations. This step

introduces initial entropy before entering the main iteration loops.

3.2 SubBytes Transformation

The SubBytes stage represents the primary source of non-linear confusion within the cipher. Each individual byte in the state matrix is structurally swapped with an independent byte from a static \(16 \times 16\) lookup configuration called the Substitution Box (S-Box). The replacement maps the upper nibble of the state byte to the S-Box row index (\(x\)) and the lower nibble to the column index (\(y\)). For instance, a state byte containing 0x53 points directly to row 5, column 3, yielding an output value of 0xED.

3.3 ShiftRows Transformation

The ShiftRows block provides structural permutation across the matrix rows to establish linear diffusion. It performs cyclical left shifts on the state array rows as follows:

1. Row 0 is kept static.
2. Row 1 is cyclically shifted left by 1 byte.
3. Row 2 is cyclically shifted left by 2 bytes.
4. Row 3 is cyclically shifted left by 3 bytes.

3.4 MixColumns Transformation

The MixColumns operation processes the state array column-by-column, treating each as a polynomial over Galois Field \(GF(2^8)\). Each column is multiplied against a fixed polynomial matrix to distribute individual byte information across the entire column. The matrix equations are computed via the following linear combinations:

$$b_0 = (02 * a_0) \oplus (03 * a_1) \oplus a_2 \oplus a_3$$
$$b_1 = a_0 \oplus (02 * a_1) \oplus (03 * a_2) \oplus a_3$$
$$b_2 = a_0 \oplus a_1 \oplus (02 * a_2) \oplus (03 * a_3)$$
$$b_3 = (03 * a_0) \oplus a_1 \oplus a_2 \oplus (02 * a_3)$$

3.5 Key Expansion and Round Additions

The Key Expansion module takes the primary 128-bit key and generates 10 independent round keys. During the AddRoundKey step of rounds 1 through 10, the updated state matrix is XORed with the corresponding round key block. This creates a high degree of key-dependent diffusion across the ciphertext output. Notably, the final round (Round 10) omits the MixColumns transformation entirely, in strict accordance with the NIST AES specification.

4. Hardware Implementation & Simulation Results

AES ciphers operate over fixed block widths of 128 bits, dynamically supporting cryptographic key lengths of 128, 192, and 256 bits. AES-128 relies on a mathematical architecture consisting of 10 structured rounds. While software solutions are popular, they are inherently prone to execution latency overheads and side-channel vulnerabilities. Hardware implementation using Hardware Description Languages (HDLs) offloads cryptographic computing to dedicated logical fabric, enabling highly parallel configurations that achieve high throughput and deterministic latency. This study outlines a modular Verilog HDL design targeting FPGA implementation of an AES-128 encryption algorithm.

Table -2: input and output status

Processing Module	Testbench Input State Data (Hexadecimal)	Simulated Vector Output Data (Hexadecimal)	Verification Status
SubBytes	00112233445566778899aabbccddeeff	638293c31bfc33f5c4eeacea4bc12816	Functional Match
ShiftRows	63fca161bee28c3c4c1193f5b8233ea0	63e293a0be113e614c23a13cb8fc8cf5	Functional Match
MixColumns	d4bf5d30d4bf5d30d4bf5d30d4bf5d30	046681e5046681e5046681e5046681e5	Functional Match
AddRoundKey	*State:* 1234567890abcdef1234567890abcdef *Key:* 0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f	1d3b59779fa4c2e01d3b59779fa4c2e0	Functional Match

The behavioral verification waveforms confirm strict adherence to standard test vectors. Stable signal states during simulation verify that the modular hardware blocks maintain consistent data pipelines without logic hazards or executionfaults.

4. RESULTS



Fig -2: subbyte waveform

The SubBytes module was simulated using Verilog HDL in Xilinx Vivado to verify the byte substitution operation of AES-128 encryption. The input stste ***00112233445566778899AABBCCDDEEFF*** was applied to the module, and the output state obtained ***638293C31BFC33F5C4EEACEA4BC12816***. Each byte of the input state was replaced with its corresponding value from the AES S-Box lookup table. The simulation waveform confirms that the module produces the expected substituted state for the given input. This verifies the functional operation of the SubBytes module.

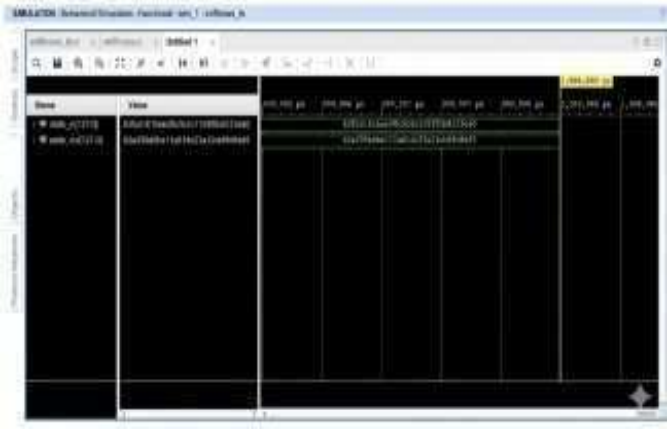


Fig -3: AddRoundKey waveform

The **waveform** confirms the correct **Verilog implementation** of the **AES AddRoundKey module** by demonstrating that the **128-bit input plaintext** (1234567890abcdef1234567890abcdef) and **round key** (0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f) successfully perform a **bitwise XOR operation** to produce the expected **output state** (1d3b59779fa4c2e01d3b59779fa4c2e0).

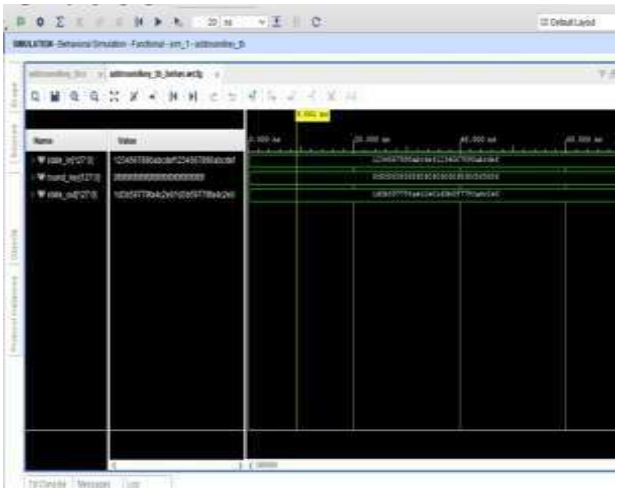


Fig -4 AddrooundKey waveform

The waveform illustrates the functional verification of the AddRoundKey module using Verilog HDL in Xilinx Vivado. The 128-bit input state is combined with the corresponding 128-bit round key through a bitwise XOR operation to generate the output state. Each bit of the input state is XORed with the corresponding bit of the round key. The simulation waveform demonstrates the resulting state after key

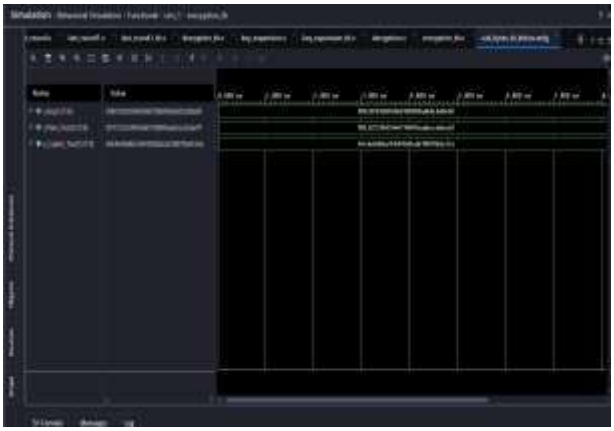


Fig-5 Vivado Behavioral Simulation Waveform

The waveform represents the simulation of the AES-128 encryption module implemented using Verilog HDL in Xilinx Vivado. The module takes a 128-bit plaintext and a 128-bit secret key as inputs and performs the initial AddRoundKey operation followed by ten encryption rounds to generate the ciphertext.

DECRYPTION

3.6 Initial AddRoundKey

The Initial AddRoundKey operation is the first step in the AES-128 decryption process. In this operation, the 128-bit ciphertext is combined with the 128-bit round key generated for Round 10 during the AES key expansion process.The operation is performed using the bitwise XOR (Exclusive-OR) function. Each bit of the ciphertext is XORed with the corresponding bit of the Round 10 key. This produces an intermediate 128-bit state that is passed to the subsequent inverse transformation stages.The AddRoundKey operation is reversible because XORing the same key twice restores the original value. The initial decryption operation uses the last encryption round key rather than the original cipher key.

Mathematical expression:

$*S' = C \oplus K_{10}$

* **C** represents the 128-bit ciphertext.

* K10 represents the Round 10 key.

* S` represents the resulting intermediate state.

*⊕represents the bitwise XOR operation.

3.7 InvShiftRows

InvShiftRows is the inverse of the ShiftRows transformation used during AES encryption. TheAES state is arranged as a 4 × 4 matrix of bytes. During decryption, the rows are cyclically shifted to the right by different numbers of byte positions.

* First row: no shift.

* Second row: shift right by one byte.

* Third row: shift right by two bytes.

* Fourth row: shift right by three bytes.

This operation restores the byte positions that were modified by the ShiftRows transformation during encryption. It does not change the values of the individual bytes; it only changes their positions within the state matrix.

Purpose: To reverse the row-shifting operation performed during AES encryption

3.8 InvSubBytes

InvSubBytes is the inverse of the SubBytes transformation. During encryption, SubBytes replacesevery byte in the state with a corresponding value from the AES substitution box (S-Box). During decryption,InvSubBytes uses the inverse substitution box (Inverse S-Box) to recover the original byte values.Each of the 16 bytes in the AES state is independently substituted using the inverse S-Box.

3.9 InvMixColumns

InvMixColumns is the inverse of the MixColumns transformation used in AES encryption. It operates on each column of the 4 × 4 AES state matrix independently.During encryption, MixColumns combines the four bytes of each column using arithmetic in the finite field . During decryption, InvMixColumns applies the inverse transformation to recover the state before the original MixColumns operation.

Each column is multiplied by a fixed inverse matrix:

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

4.0 AddRoundKey

AddRoundKey is a fundamental operation used in every AES encryption and decryption round. Itcombines the current 128-bit state with the appropriate 128-bit round key using a bitwise XOR operation.During decryption, the round keys are applied in reverse order, beginning with and proceeding towards . After the initial AddRoundKey operation, the regular decryption rounds use the corresponding round keys in descending order.The round keys are generated from the original 128-bit cipher key using the AES key expansion algorithm. They do not need to be independently entered by the user for every round.

Mathematical expression:

$$\boldsymbol{Sout = Sin \oplus Kr}$$

4.1 Inverse Round

The Inverse Round module combines the inverse transformations required for the main rounds ofAES-128 decryption. AES-128 uses nine regular inverse rounds after the initial AddRoundKey operation.In the standard inverse cipher, each regular decryption round performs the following operations**:**

Input and output status

Processing Module	Testbench Input State Data (Hexadecimal)	Simulated Vector Output Data (Hexadecimal)	Verification Status
Inv_Round	bd6e7c3df2b5779e0b61216e8b10b689	f4df52424d0af9898baf731af4dca25e	Functional Match
Inv_ShiftRows	8855ddcc6611aa770033ecbb9944ff22	8844ee776655ffbb0011dd229933aa0es	Functional Match
Inv_Mixcolumns	e488a0208580330203a2d80c60746d0	9fd202aed2031fad58b2207db02d9e54	Functional Match
AES-128 Decrypted Output	69c4e0d86a7b0430d8cdb78070b4c55a	00112233445566778899aabbccddeeff	Functional Match

The behavioral verification waveforms confirm strict adherence to standard test vectors. Stable signal states during simulation verify that the modular hardware blocks maintain consistent data pipelines without logic hazards or execution faults.

RESULT

Fig - 4 Inv_Round

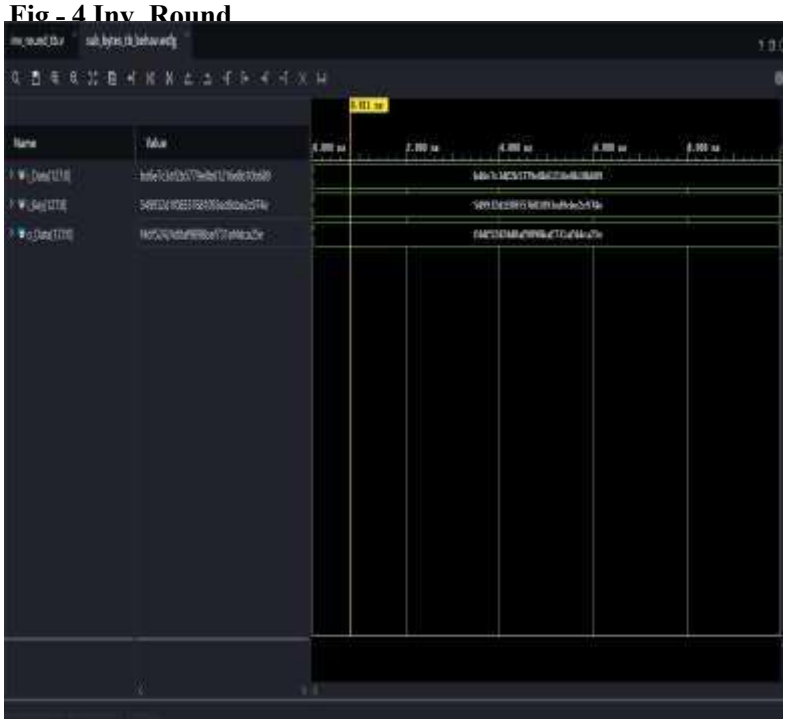


Fig -4 **Inv_Mixcolumns**



The MixColumns output is obtained by multiplying each column of the input state by a fixed AES matrix using finite-field arithmetic in . Each output byte is calculated using multiplication and XOR operations on the input bytes. For example, the first output byte is calculated as: **B0=(02*04)⊕(03*6B)⊕5D⊕30**

The resulting bytes form the transformed 128-bit state, as observed in the Vivado simulation waveform

.

CONCLUSIONS

AES (Advanced Encryption Standard) is a secure and efficient symmetric key encryption algorithm widelyused for protecting digital information. It operates on 128-bit data blocks and uses secret keys of 128, 192, or 256 bits for encryption and decryption. In this project, AESencryption algorithm was implemented and analyzed using Verilog HDL. The main operations of AES such as SubBytes, ShiftRows, MixColumns, and AddRoundKey were designed and simulated successfully. SubBytes provides substitution using an S-box .

ShiftRows, MixColumns, and AddRoundKey were designed and simulated successfully. SubBytes provides substitution using an S-box, ShiftRows performs cyclic shifting rows,MixColumnsincreases diffusion through matrixmultiplication, and AddRoundKey combines the data with the round key using XOR operation.The simulation waveforms confirmed the correct functionality of each module. The project provided a clear understanding of cryptographic techniques, digital hardware implementation, and functional verification using simulation tools.

REFERENCES

1. Prithwi Narayana Bhat M. G. and Hemanth Kumar A. R., "Application of the Advanced Encryption Standard for Image Processing and Performance Analysis," TIJER – International Research Journal, vol. 10, issue 9, pp. a604–a609, September 2023. Available: TIJER - INTERNATIONAL RESEARCH JOURNAL.
2. Nitin R. Chavan and Suresh A. Annadate, "VHDL Implementation of AES-128 on FPGA," International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering, 2015. DOI: IJIREEICE.2015.3108?utm_source=chatgpt.com). Peer-reviewed Journal.
3. "FPGA Implementation of a High Throughput Low Power Advanced Encryption Standard (AES-128) Cipher," 2023 10th International Conference on Signal Processing and Integrated Networks (SPIN), 2023. Available through IEEE Xplore. IEEE Xplore.
4. "High-Speed VLSI Architectures for the AES Algorithm," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 9, pp. 957–967, September 2004. DOI:IEEE Explore.
5. Tsung-Fu Lin, Chih-Pin Su, Chih-Tsun Huang, and Cheng-Wen Wu, "A High-Throughput Low-Cost AES Cipher Chip," 2002 IEEE Asia-Pacific Conference on ASIC, pp. 85–88, 2002. DOI:.National Tsing Hua University.
6. Chih-Pin Su, Tsung-Fu Lin, Chih-Tsun Huang, and Cheng-Wen Wu, "A Highly Efficient AES Cipher Chip," Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), National Tsing Hua University.
7. "Area-Efficient VLSI Architecture for Advanced Encryption Standard," 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT),2024.DOI: IEEE Xplore