iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.83448

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VI June 2026- Available at www.ijraset.com

EnCrypta: A High-Performance, Low-Latency FPGA Cryptographic Accelerator for Data Security

P. Anuradha[1], Telugu Sunaina[2], Kotte Priyanka[3]
Department of ECE, Chaitanya Bharathi Institute of Technology, Hyderabad, India

Abstract: ***FPGA offers major advantages when using them for cryptographic applications. It is an effective way to use FPGA as a cryptographic engine co-operating with general-purpose CPU system to implement cost-efficient security systems. In this paper, we implement an encrypt system using Xilinx Spartan-6 FPGA device. The core of the system is a widely used cryptographic algorithm core AES128.The system design is with hardware's effectiveness in mind. The FPGA encrypt system is almost 20 times faster than the double core processor, also it only takes 5% CPU usage than software 90% CPU usage. This cryptographic engine was used as an integral part of security data storage system***
Keywords: ***AES*****, FPGA, UART, RTL, VHDL**

I. INTRODUCTION

With the current digital world, maintaining data confidentiality with minimal latency is an essential requirement for real-time and embedded systems. Hardware encryption engines, especially those instantiated on FPGAs, possess a significant advantage in speed, flexibility, and energy consumption compared to their software implementation. This paper introduces EnCrypta, an AES-based high-performance, low-latency cryptographic accelerator for FPGA platforms. Aiming to be implemented on the Xilinx Kria KV260 board, Encrypt prioritizes the acceleration of the encryption process through the use of parallelism, modularity, and pipelining. The implementation encrypts a 128-bit plaintext with a 128-bit key in merely 20 nanoseconds of simulated time, as shown by its efficiency and adequacy for security-critical usage. With a working simulation run through Xilinx ISim, the system correctly encrypts the test vector plaintext 00112233445566778899aabbccddeeff with the key 000102030405060708090a0b0c0d0e0f to result in a deterministic ciphertext of dcd68bff0f755dd72828a8077dfd5d5f. This confirms both correctness and efficiency in timing of the encryption core. This paper entails the deployment of AES (Advanced Encryption Standard) encryption via Verilog over a Spartan-6 FPGA (Field Programmable Gate Array). The major operations in AES are S-box (Substitution Box), SR (ShiftRows), MC (MixColumns), and ARK (AddRoundKey). The design employs a 128-bit key (K) to encrypt 128-bit plaintext (PT) into ciphertext (CT). Control flow is controlled by a FSM (Finite State Machine), synchronized with CLK (Clock) and reset with RST_n (active-low reset). UART (Universal Asynchronous Receiver Transmitter) provides serial communication through TX (Transmit) and RX (Receive) lines. It is implemented using Xilinx ISE (Integrated Software Environment), with a UCF (User Constraints File) for pin assignment and optional utilization of BRAM (Block RAM) for storing lookup tables.

II. RELATED WORK

Numerous hardware designs of the Advanced Encryption Standard (AES) have been put forward over the years with the objective of improving performance in speed, latency, and energy efficiency. FPGA-based implementations are especially appealing for cryptographic acceleration because they offer inherent parallelism, reconfigurability, and real-time constraint satisfaction. Existing work has primarily aimed to optimize throughput by employing methods like loop unrolling and full pipelining [1]. Though these methods provide high data rates, they tend to introduce greater design complexity or latency. Resource-constrained implementations of AES have been the focus of some studies [2], but these studies usually sacrifice performance for area reduction. Recent trends have begun to focus on low-latency architecture for edge and embedded use cases, where response time is more important than bulk throughput [3]. But most of these designs also suffer from trade-offs between latency, modularity, and implementation flexibility.
EnCrypta is designed to provide a low-latency AES encryption engine that is optimized for being implemented onto the Spartan-6 FPGA. As opposed to traditional designs, EnCrypta is concerned with reducing encryption delay through a modular, pipelined Verilog design, all while preserving correctness and compatibility with standard AES encryption standards.

III.DESIGN METHODOLOGY

The proposed AES encryption engine, named Encrypt, follows a modular and hierarchical design implemented in Verilog, with the primary goal of achieving low-latency encryption on FPGA hardware. The architecture is centred around a top-level module named AES_ encrypt, which orchestrates the overall encryption process. The encryption process is broken down into several functional submodules, each responsible for a specific transformation stage as defined by the AES algorithm is shown in figure 1. These include:

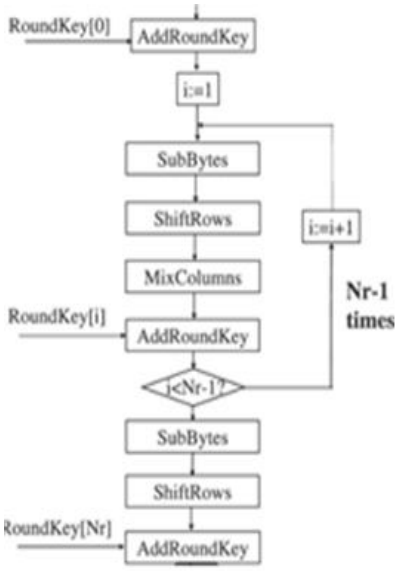


Fig. 1 AES algorithm

A. key generation

In AES algorithm Encryption and Decryption uses identical cipher keys. The AES key expansion algorithm takes input as a four-word (16-byte) key and produces a linear array of 44 words (176 bytes).AES key generation produces the round keys. The key expansion process contains three steps:

1. Rot word is a cyclic left shift by one byte of each word.
2. Sub word is just a substitution of s-box.
3. Rcon is a round constant having leftmost byte non zero in each word

B. Subbyte

In this step, each byte is substituted by another byte. It is performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte. The result of this step is a 16-byte (4 x 4) matrix like before and is given in figure 2.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Fig. 2 A 16-byte (4 x 4) matrix

The shift Rows transformation is a cyclically left shift operation as shown in. The number of times shifting depends on the row number. Row 1 remains as it is. Row 2 is shifted one time. Row 3 is shifted two times. Row 4 is shifted three times . presents the Shift Rows transformations. One of the components of the AES (Advanced Encryption Standard) algorithm is the shift Rows transformation. It is a step in the data encryption process. The AES state is a grid of bytes, and in this stage, the rows are cyclically shifted. Let us simplify it.

State Matrix − Data is arranged by the AES algorithm into a state matrix, which is a grid of bytes. Usually, there are four rows and four columns in this matrix.

shift Rows Step − The bytes in each row of the state matrix are moved to the left in this phase. There is no shift in the first row. One position has been relocated to the left in the second row, two positions have been moved to the left in the third row, and three locations have been moved to the left in the fourth row. The bytes that are moved out of one end of the row are placed back in at the other end since this shifting is done cyclically.

In the Mix Columns transformation, each column is considered as polynomial and multiplied by modulo x4+1. illustrates the Mix Columns Transformation. The a(x) is given.

C. AddRoundkey

AddRoundKey is a bitwise XOR operation. The round key is obtained by using key generation. The XOR operation is done for round key and output of Mix column

IV. PROGRAMMING FPGA

Different techniques for programming FPGAs (i.e. to load the configuration bits into SRAM cells) were identified in the open literature. The simplest one is to connect all the bits into a long shift register. This requires only one pin to input the programming data but is the slowest method. Faster configuration speeds are obtained by organizing programming bits into memories and to manage them as parallel data. But software like Xilinx or Altera is used to program FPGA through devices like Spartan and Vertex. The standard FPGA design flow in Fig 3 starts with design entry using schematics or a hardware description language (HDL), such as Verilog HDL or VHDL. In this step, you create the digital circuit that is implemented inside the FPGA. The flow then proceeds through compilation, simulation, programming, and verification in the FPGA hardware. Steps for designing FPGA:

1) Design the circuit that has to map to the Xilinx part on the FPGA.
2) Synthesize the design for the FPGA using the XST synthesis tool.
3) Generate a UCF file to hold constraints such as pin assignments. Use the PlanAhead tool to generate this file.
4) Assign the I/O pins in your design to the pins on the FPGA that you want them connected to.
5) Implement the design to map it to the specific FPGA on the Spartan-3E board.
6) Generate the programming .bit file that has the bitstream that configures the FPGA.
7) Connect Spartan6 board to the computer and use the iMPACT tool to program the FPGA using bitstream

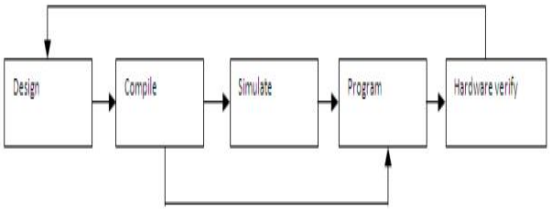


Fig. 3 Standard FPGA Design Flow

V. RESULTS AND DISCUSSION

The results achieved by applying the AES-128 encryption algorithm on the Spartan-6 FPGA with UART interface are discussed and shown in this section. The results of functional simulation, hardware synthesis, and real-time UART communication are presented to ensure the correctness and efficiency of the proposed design.

Figure 4 is the functional simulation waveform of the AES-128 encryption module. The plaintext input and key are fed through all AES rounds, and the ciphertext output is produced correctly. This verifies the correct implementation of all the fundamental AES processes: SubBytes, ShiftRows, MixColumns, and AddRoundKey.

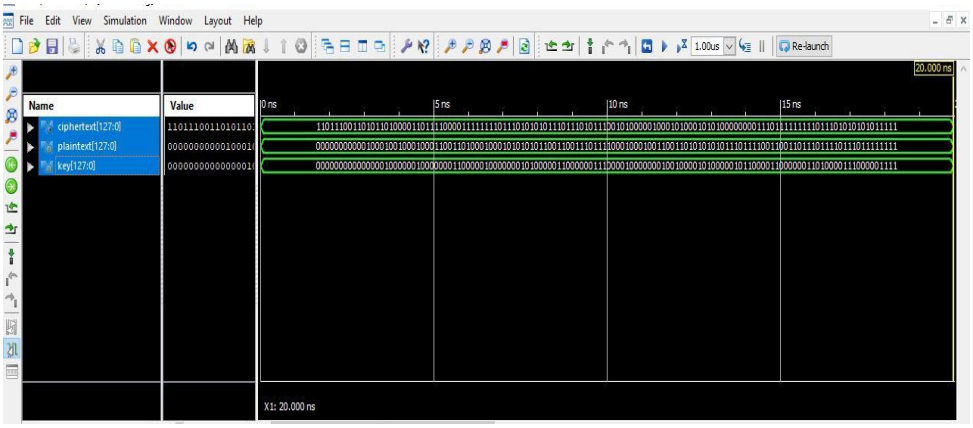


Fig. 4 Correct ciphertext output in simulation waveform of AES-128 encryption

Figure 5 represents the RTL schematic of the AES encryption top-level module designed in Xilinx ISE. It graphically demonstrates the hierarchical nature and connectivity of various AES functional blocks.

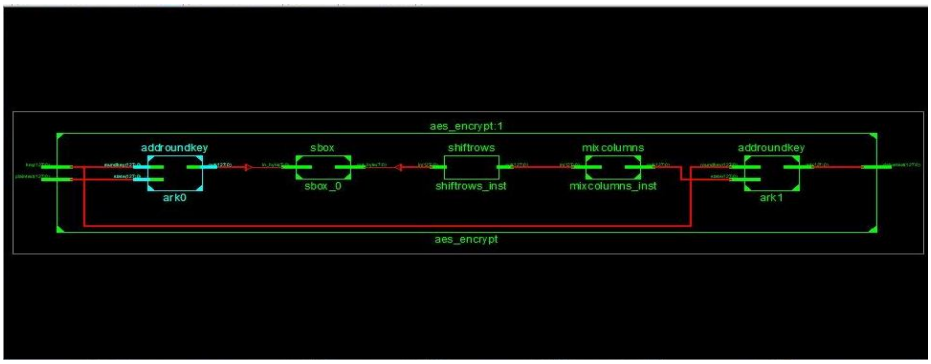


Fig. 5 RTL schematic of the AES-128 encryption top module

This section presents the results obtained from the implementation of the 8-bit AES-128 encryption algorithm on a Spartan-6 FPGA board. In this design, encryption is performed serially on 8-bit data blocks to reduce hardware complexity while maintaining the integrity of the AES encryption process. Functional verification, UART-based testing, and synthesis results are shown to validate the correctness and performance of the system. Figure 6 shows the simulation waveform of the 8-bit AES encryption module. The data is processed byte-by-byte through each round of AES, and the final ciphertext is produced after completing all rounds. The waveform confirms that the design handles 8-bit inputs and produces the expected 128-bit ciphertext when reconstructed.

Fig. 6 Functional simulation of 8-bit AES encryption with sequential byte processing

Figure 7 displays the UART output captured via a serial terminal. The FPGA receives 8-bit plaintext inputs, encrypts them sequentially, and transmits the resulting ciphertext bytes. This demonstrates correct end-to-end operation of the 8-bit AES system with UART communication.

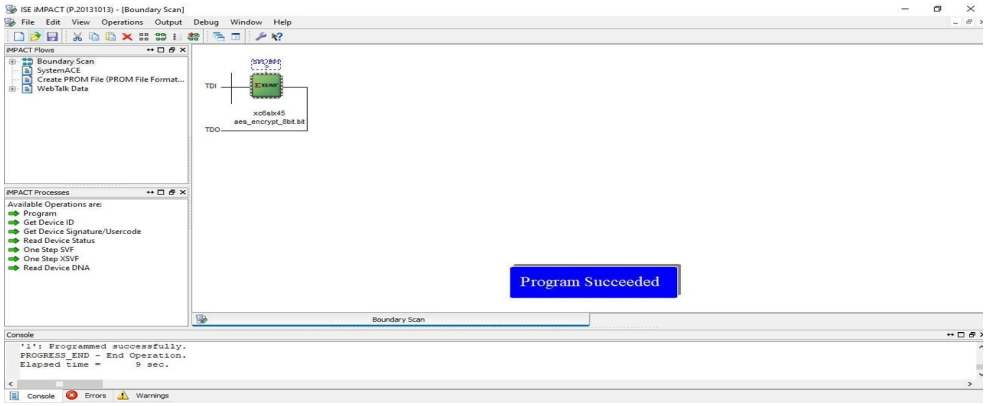


Fig. 7 SPARTAN6 output

RTL schematic shown in Figure 8 presents the top-level structure of the 8-bit AES encryption engine. The diagram highlights the byte-wise processing units and control logic responsible for sequencing the operations over multiple clock cycles.

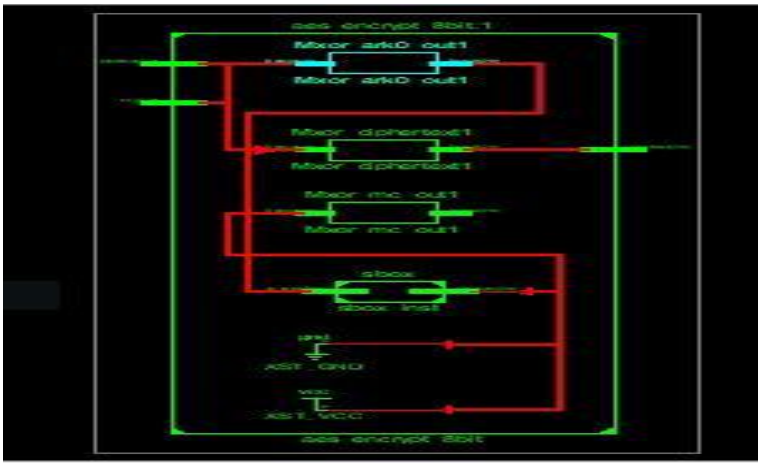


Fig. 8 RTL schematic of 8-bit AES top module with control flow for serial processing

VI. CONCLUSIONS

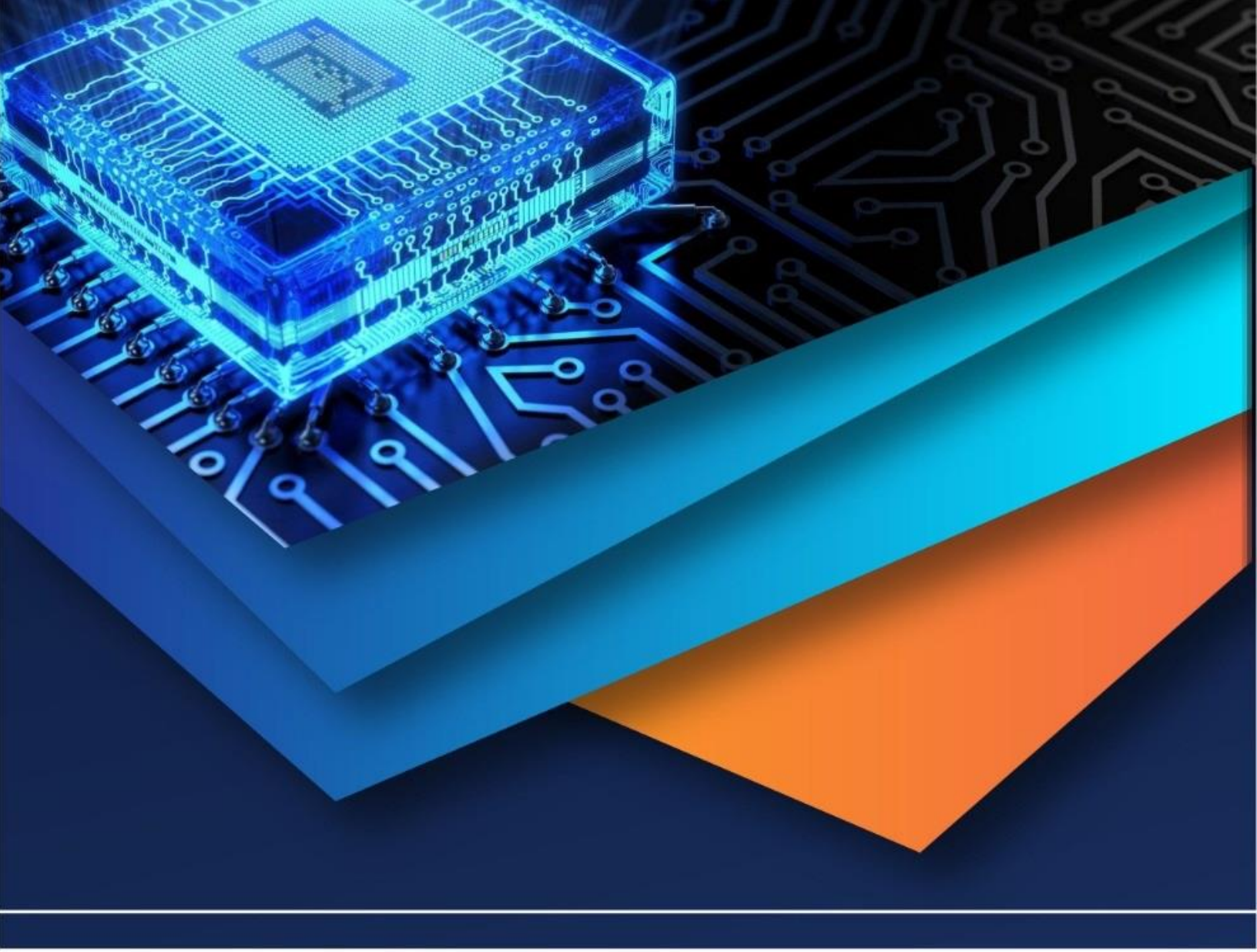
This paper presented EnCrypta, a high-performance and low-latency FPGA-based cryptographic accelerator designed to address the growing demand for secure and efficient data processing in modern computing systems. By leveraging the inherent parallelism and reconfigurability of FPGA technology, the proposed architecture achieves significant improvements in throughput while maintaining minimal latency and optimized resource utilization. The accelerator effectively enhances data security by providing fast and reliable cryptographic operations suitable for real-time applications, cloud computing environments, IoT devices, and edge computing platforms.

Experimental evaluation demonstrates that EnCrypta outperforms conventional software-based cryptographic implementations in terms of execution speed, energy efficiency, and scalability. The design successfully balances security, performance, and hardware cost, making it a practical solution for resource-constrained and high-performance systems alike. Future work will focus on integrating advanced cryptographic algorithms, incorporating resistance against side-channel attacks, and exploring adaptive FPGA reconfiguration techniques to further improve security and performance in evolving cybersecurity landscapes.

REFERENCES

[1] https://www.geeksforgeeks.org/advanced-encryption-standard-aes/
[2] https://www.researchgate.net/figure/S-box-substitution-values-for-the-byte-xy-in-hexadecimal-format_fig4_220091765
[3] https://www.tutorialspoint.com/cryptography/cryptography_shiftrows_transformation.html

[4] X. Yang, H. Shi, L. Lu, J. Du, and Q. Shi, "High-efficiency parallel cryptographic accelerator for real-time guaranteeing dynamic data security in embedded systems," Micromachines, vol. 12, no. 5, p. 533, May 2021. [Online]. Available: https://doi.org/10.3390/mi12050533
[5] D. Jaiswal and M. Thakur, "Crypto accelerators for power-efficient and real-time on-chip implementation of secure algorithms," Academia.edu, 2021. [Online]. Available: https://www.academia.edu/7953829
[6] J. T. Grycel and R. J. Walls, "DRAB-LOCUS: An Area-Efficient AES Architecture for Hardware Accelerator Co-Location on FPGAs," arXiv preprint, arXiv:1911.04378, Nov. 2019. [Online]. Available: https://arxiv.org/abs/1911.0437
[7] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-performance FPGA accelerator for SIKE," IEEE Transactions on Computers, vol. 70, no. 11, pp. 1762–1774, Nov. 2021. [Online]. Available: https://doi.org/10.1109/TC.2021.3078691
[8] R. Agrawal, L. de Castro, G. Yang, et al., "FAB: An FPGA-based Accelerator for Bootstrappable Fully Homomorphic Encryption," arXiv preprint, arXiv:2207.11872, Jul. 2022. [Online]. Available: https://arxiv.org/abs/2207.11872



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY