IJARSCT

International Journal of Advanced Research in Science, Communication and Technology

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

ISSN: 2581-9429

Volume 5, Issue 10, June 2025

Impact Factor: 7.67

Efficient Implementation of 128-Bit AES Algorithm on FPGA for Minimal Resource Utilization

Vinaya Kumar K[1] and Prof. Raji C[2]

School of Electronics and communication Engineering Reva University, Bangaluru, Karnataka, India[1,2]

kalasapuravinaykumar@gmail.com, raji.c@reva.edu.in

Abstract: *Cryptography plays a crucial role in ensuring the security and integrity of data, especially in the era of increasing cyber threats. The Advanced Encryption Standard (AES) is commonly used for secure communications, but its hardware implementations must effectively balance performance, resource use and power efficiency, particularly in FPGA-based settings. This paper presents an optimized AES-128 implementation in Verilog, crafted to minimize resource demands and power consumption while preserving encryption performance. In contrast to standard AES designs, our approach leverages optimized S-Box computations, efficient key expansion, and resource-sharing techniques to lower logic complexity. The design has been synthesized and implemented on an FPGA, achieving significant reductions in resource utilization compared to conventional AES architectures. Simulation results reveal improvements in resource utilization, area efficiency and energy savings, making our design well-suited for low-power embedded systems*

Keywords: Cyber threats, Advance encryption standard, Verilog, Logic complexity, resource utilization, energy savings

I. INTRODUCTION

Cryptography, an ancient practice with origins spanning thousands of years, has evolved into an indispensable component of contemporary life, addressing key properties such as secrecy, authentication, integrity, authenticity, non-repudiation, and availability. The foundational principle governing encrypted messages' security is not solely the encryption algorithm but the inclusion of a secret key during the procedures for encryption and decryption.

In this study, we introduce an optimized AES-128 encryption implementation tailored for FPGA-based systems that operate under strict resource and power limitations. Unlike standard AES architectures, our design incorporates efficient S-Box transformations, minimizes key expansion overhead, and employs low-power combinational logic optimizations to enhance performance while decreasing power consumption. We evaluate our results against existing AES implementations, focusing on metrics such as lookup table (LUT) usage, flip-flop count, power consumption, and encryption latency.

The key contributions of this paper include:

1) Optimized S-Box Computation – Implementing a memory-efficient S-Box lookup to decrease resource usage.

2) Low-Power Design Approach – Applying resource-
sharing techniques to lower FPGA logic consumption.

A secure cryptographic processor, specifically designed for tasks such as modular exponentiation and DES encryption, integrates various physical security features to enhance resistance against tampering.

These secure processors can take various forms depending on usage profiles, including FPGAs, or field-programmable gate arrays [3], [8], application-specific integrated circuits [15] or microcontrollers[14]. FPGAs offer a unique advantage with their reconfiguration capabilities, allowing optimization of operations and corrections if necessary.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 10, June 2025 Impact Factor: 7.67

II. RELATED WORKS

Busam Swayamprakash et. al. [1] In comparison to DES, which had a fixed 56-bit key, AES is more secure since it employs 16-byte blocks and supports key sizes ranging from 128 to 256 bits. For AES implementation, Verilog is favored over VHDL because of its fewer clock cycles and propagation delay, which increase efficiency and minimize power consumption.

Deguang Le et.al. [2] The research built and constructed a fast GPU-based data encryption system and proposed a new AES parallel encryption algorithm. The test demonstrates that, in comparison to CPU, our method may greatly speed up AES encryption.

M.Natheera Banu et.al [3] In this paper, a hardware implementation of AES for safe text and image encryption using an FPGA and a constant 128-bit key is presented. AES may be successfully executed on FPGA, as the study shows, with encryption and decryption times of 2.595 ns and 2.606 ns for a 2048-byte text file and 374.656 ns and 361.856 ns for a 2048-byte image file. The outcomes demonstrate the viability of using FPGA-based AES for safe data transfer.

. M.Pitchaiah et.al. [4] This paper presents a synthesizable 128-bit Rijndael algorithm-based AES encryption and decryption, implemented in Verilog for FPGA deployment. The technique efficiently transforms data into secure plaintext by utilizing cipher, inverse cipher, and key expansion.

Ganesh Gopal Shet et.al. [5] AES encryption has been implemented in this work utilizing Verilog with key lengths of 128, 192, and 256 bits. Because it works at a greater frequency and minimizes encryption latency, a parallel AES design is ideal for applications requiring speed.

Zabina Kouser et.al.[6] this study explores FPGA-based AES implementation, highlighting the advantages of hardware acceleration for faster encryption and improved security. FPGA architectures enable parallel processing, reducing encryption time while enhancing performance. A comparative analysis of different FPGA families assessed performance metrics, power consumption, and energy efficiency optimizations. The findings offer valuable insights into FPGA design best practices for achieving high throughput and robust encryption in hardware-based security systems.

Hrushikesh S et.al [7] explored the implementation of AES on FPGA, emphasizing efficiency improvements. Their study provided a detailed analysis of different FPGA architectures and their impact on AES performance. They examined the role of pipelining, parallelism, and memory optimizations in enhancing encryption speed. The research also addressed power consumption concerns, proposing solutions to balance security with energy efficiency. Their findings reinforce the growing importance of FPGA-based cryptographic solutions in modern security applications.

Selvakumar et.al. [8] this study explores AES encryption and decryption implementation using Verilog HDL for FPGA-based applications, aiming to enhance speed, efficiency, and security. It highlights the benefits of parallel processing and hardware acceleration, analyzing optimization techniques like pipelining and resource sharing. Key performance metrics such as throughput, area utilization, and latency are evaluated, demonstrating FPGA's advantages for real-time security applications. The study also explores power-saving techniques for low-power embedded systems, reinforcing FPGA-based encryption's role in IoT, communication systems, and secure data transmission.

Chaitanya Kumar et.al [9] present an optimized FPGA-based AES implementation aimed at high-speed encryption. Their study focuses on efficient hardware resource utilization to enhance performance while maintaining security. By leveraging FPGA parallelism, the modified AES design improves throughput and reduces latency, making it ideal for real-time encryption in secure communications and data protection.

A. Desai et.al. [10] this study presents an optimized AES key expansion technique for FPGA implementations, improving encryption efficiency by reducing memory usage and computational overhead. By modifying the key schedule structure, the proposed method minimizes latency, making AES more suitable for real-time applications. Experimental results show enhanced processing speed without compromising security, making it ideal for embedded systems and secure communications. The findings highlight the importance of hardware-aware optimizations in AES, offering valuable insights for high-speed encryption in network security and secure data transfer.

Vikram R et.al. [11] propose a hybrid cryptographic model combining AES and RSA for enhanced secure communication. AES ensures fast data encryption, while RSA handles secure key exchange, balancing efficiency and robustness. This approach mitigates vulnerabilities of standalone AES or RSA, improving security and computational

feasibility. Performance evaluation shows increased resistance to attacks and optimized resource utilization, making it ideal for applications requiring confidentiality and authentication, such as financial transactions and secure data sharing.
S. Narayan et.al.[12] analyze fault injection attacks on AES hardware and propose countermeasures to enhance security. These attacks exploit vulnerabilities by introducing faults to extract secret keys. The study identifies weak points in AES circuits and suggests error detection and redundancy-based protection strategies to mitigate risks. Their approach strengthens AES resilience without significant performance overhead. This research is crucial for secure hardware design in critical applications like financial systems and embedded security, ensuring robust protection against physical attacks.
Ako Muhammad Abdullah [13] The symmetric block encryption known as the Advanced Encryption Standard (AES) is popular. known for its strong security and efficiency. It encrypts and decrypts sensitive data in both hardware and software applications worldwide. AES provides three different key sizes: 256, 192, and 128 bits. —while maintaining a 128-bit block size. Research indicates that AES provides superior security and faster performance compared to older algorithms like DES and 3DES. So far, no vulnerabilities have been found that require modifications to the algorithm
Yehya A. Nasser et al. [14] presented an AES implementation tailored for low-cost 8-bit microcontrollers. Their research focused on optimizing AES for resource-constrained environments, demonstrating how lightweight encryption can be achieved while maintaining security standards. The study highlighted the trade-offs between computational efficiency and encryption strength, showcasing how microcontroller-based implementations can support secure communication in embedded systems. The authors evaluated different key scheduling techniques to improve execution speed and reduce memory consumption, making AES more feasible for low-power applications.
T. Kumar et.al [15] present a low-power AES implementation on ASIC, aimed at enhancing energy efficiency in secure data communication, especially for battery-operated and embedded devices. Their study introduces architectural modifications to reduce power consumption without compromising encryption strength. Key optimizations include efficient AES round transformations, key expansion, and S-box implementations, leading to substantial energy savings while maintaining cryptographic robustness. The proposed ASIC-based design outperforms conventional AES architectures in power efficiency, making it suitable for IoT devices, smart cards, and portable security systems. The research highlights trade-offs in low-power cryptographic design and underscores the need for power-efficient encryption hardware in modern secure applications.

III. METHODOLOGY

For the AES-128, 192, and 256-bit versions, the encryption process requires 10, 12, and 14 rounds, respectively. The encryption key and the input data are both 128 bits wide in the AES-128 bit variation, and each round takes one cycle to finish its work.. An example of the architectural flow of AES can be found below.

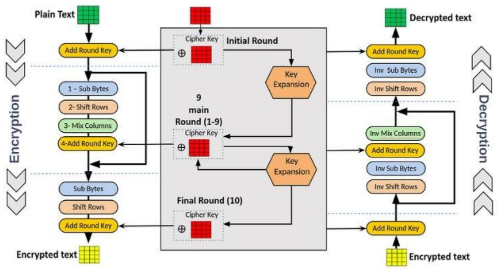


Fig.1 AES algorithm

An array of two-dimensional bytes identified by the State is utilized internally by the AES algorithm to carry out its operations. According to the following technique, the input array, "in," is copied to the State array at the point of beginning of the Cipher or Inverse Cipher.: In[r + 4c] = s[r, c]. 32-bit words are composed of four bytes in each column of the State array, where the row number r acts as an index for the four bytes. As a result, the state can be represented as a one-dimensional string of words, w0 … w3, with a bit size of 32 (columns), where the column number serves as an index.

A state description in an array of four words is presented underneath.
w0 = s0,0 s1,0 s2,0 s3,0 w1 = s0,1 s1,1 s2,1 s3,1
w2 = s0,2 s1,2 s2,2 s3,2 w3 = s0,3 s1,3 s2,3 s3,3

ENCRYPTION :
Every AES algorithm round (except from round 10) includes the few stages listed below:

- Add round key
- Substitute bytes
- Shift rows
- Mix columns

1. ADD ROUND KEY: The AddRoundKey() transformation, a straightforward bitwise XOR operation, adds a Round Key to the State. The AES algorithm starts with a straightforward XOR instruction. Since our plaintext and key are both 128 bits long, XOR works bit by bit, as seen below.

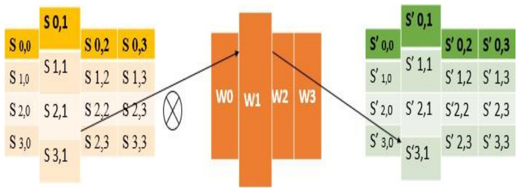


Fig.2 Add round key

The 16-byte matrix is treated as 128 bits and x-ord to the 128 bits of the round key. Should this be the last round, the results is encrypted and contains 128 bits. If not, the same round involving 16 bytes will be applied to these 128 bits once more.

2. SUBSTITUTE BYTES :
The byte has been modified in this non-linear transformation by a value in the S-box, which has been preset for the algorithm's use. Data is substituted using S-box. S-box can be thought of as a lookup table. If every block has 8-bit data, we may utilize the first 4-bit as a row index and the last 4-bit as a column index when swapping out the block for bytes. The value can then be retrieved from the S-box using these row and column indices.

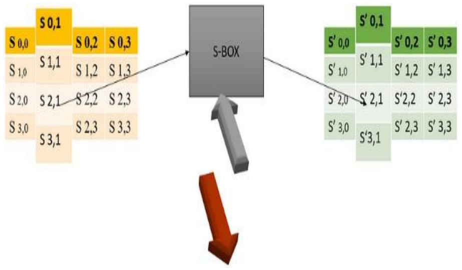


Fig.3 Sub bytes transformation

3. SHIFT ROWS: In this process, the rows of the state matrix undergo a leftward cyclic shift based on their row index. The upper row persists as unaltered, while the second, third, and fourth rows all shift one, two, and three positions to the left, respectively.

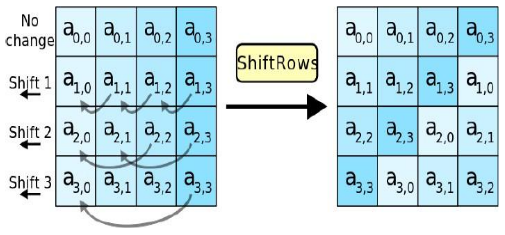


Fig.4 Shift rows

4. MIX COLUMN:

The Mix Columns transformation treats each column as a four-term and works on the State column by column. the polynomial. utilizing a fixed polynomial a(x), which has the following definition, the columns are multiplied modulo x4 + 1 as polynomials over GF (2^8):

a(x) = (02)x3+ (03)x2 + (01)x1 + (02)

S'(x)=a(x)⊕s(x)

The above equation can be described in the matrix form as below:

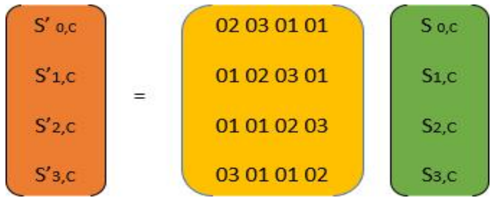


Fig5. State matrix

The polynomial equation above matrix is

*S'(0)=[(02)*S 0,]) ⊕ [(03) *S 1,c] ⊕ S 2,c ⊕ S 3,c S'(1)=[(02)*S 1,c] ⊕ [(03) *S 2,c] ⊕ S 0,c ⊕ S 3,c S'(2)=[(02)*S 2,c] ⊕ [(03) *S 3,c] ⊕ S 0,c ⊕ S 1,c S'(0)=(02)*S 3,c]⊕[(03) *S 0,c]⊕ S 2,c ⊕ S 1,c.*

KEY GENERATION: Using a 4-word key as input, the AES key expansion algorithm creates a sequential array of 44 words. There are four of these terms used in each encryption cycle. Each sub-key has a length of 128 bits because each word has 32 bytes. The enlarged key is expanded from the original key in a predefined order, along with the first four words instantly repeated from the original key.

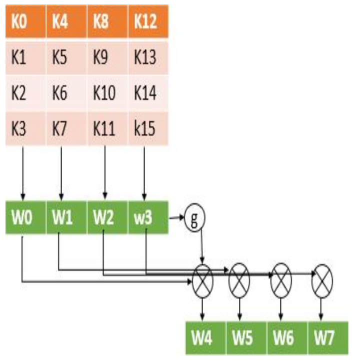


Fig 6. Key generation

The remaining part of the extended key is generated in group of four words at a time. The word immediately following it (w[i-1]) and the word four positions earlier (w[i-4]) are the sources of each new word (w[i]). In most cases the process involve a straight forward XOR operation. However when the position of a word is a multiple of four a more complex transformation is applied.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 10, June 2025 Impact Factor: 7.67

That complex function is denoted by the letter g. The following subfunctions make up the function g:
RotWord applies a circular left shift of one byte to a word. The input word [b0, b1, b2, b3] ultimately yields [b1, b2, b3, b0].
Each byte of its input word is substituted by SubWord using the s-box previously indicated**.**
The round constant, Rcon[j], is XORed with the outcome of steps 1 and 2.

DESCRIPTION:
ADD ROUND KEY: Using a bitwise XOR operation, the Round Key and the State are combined in the AddRoundKey() transformation. The AES algorithm starts by performing this step, which is executed as a straightforward XOR operation.. Given that both the plaintext and the key are 128 bits in length, the XOR operation is performed bit by bit. The 16-byte matrix, representing 128 bits, is XORed with the 128-bit round key. If this is the final round, the result is a 128-bit encrypted output. Otherwise, these 128 bits proceed to the next round, continuing the transformation withŸˆ another 16-byte block.

INVERSE SHIFT ROW: In this step, the bytes in each row of the state array are shifted to the right by a varying number of positions, essentially undoing the shifts applied during encryption. The first row is not shifted, the The second, third, and fourth rows are moved to the right by one, two, and three positions, respectively.

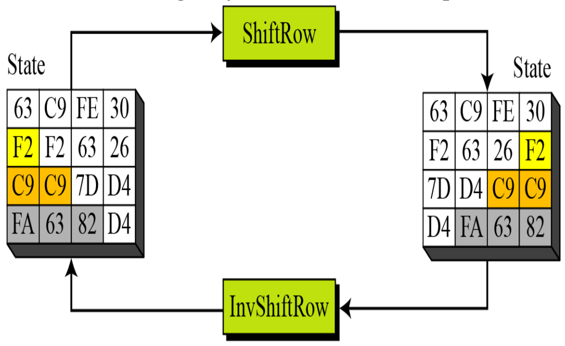


Fig 7 Inverse shift rows

INVERSE SUB BYTES:
In the Inverse SubBytes step during decryption, the inverse of the SubBytes operation is performed. Instead of looking up values from the regular S-box, the Inverse SubBytes step uses an inverse S-box, which contains the inverse of the substitution values used in step SubBytes.. The state array's individual bytes are swapped out for their corresponding value from the inverse S-box. This ormed during encryption, helping to recover the original plaintext.

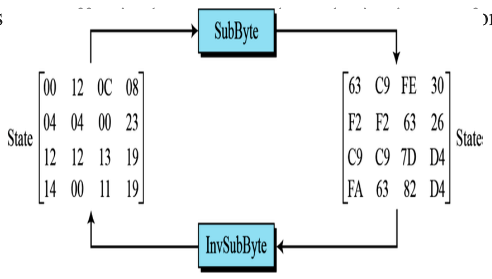
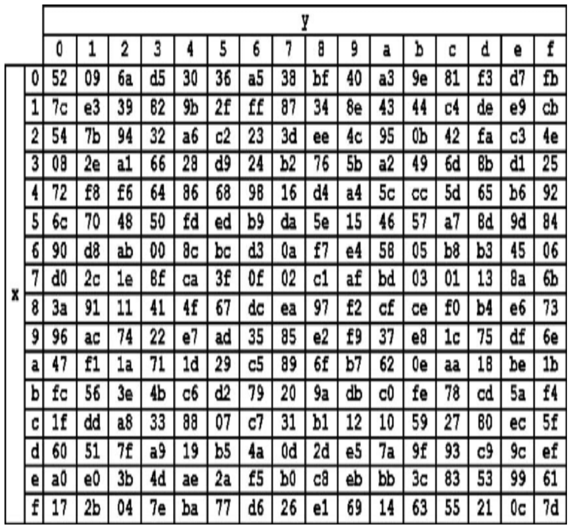


Fig8 Inverse sub bites



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

Fig 9 Inverse S- box

INVERSE MIX COLUMN : The MixColumns step's opposite operation is called the Inverse MixColumns step. A particular inverse matrix multiplication operation is used during decryption to return each column of the state array to its initial values.

.This transformation undoes the mixing applied during encryption, ensuring that the original plaintext can be accurately recovered. The inverse matrix used for this operation is designed such that applying it to the mixed columns results in the original columns before encryption

$$\underset{C}{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}} \xleftrightarrow{\text{Inverse}} \underset{C^{-1}}{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}$$

Fig10 Inverse mix column

IV. IMPLEMENTATION USING VERILOG

The implementation of AES-128 begins with writing the design in Verilog. The system undergoes synthesis and verification after being constructed in full 128-bit mode. The top module houses the encryption module and the key scheduler, according to the design. Before proceeding with the implementation, it is essential to understand certain data manipulations. The 128-bit data is divided into 32 words, with each word consisting of 8 bits. The total number of words generated during key expansion is 44, resulting in 11 keys in total.

1. S-box: The S-box requires significant hardware space since it utilizes multiple registers to store the predefined values specified in the S-box table for AES. The input and output signals are called data and dout, respectively, and are both 8-bit in size.

2. Sub-bytes: To substitute a byte with a corresponding value from the S-box, the S-box must be instantiated within the sub-byte module. Because 128-bit data is processed in each cycle, the S-box is instantiated 16 times. within this module. An internal wire, tmp_out, is used to collect the output from each S-box, and the final 128-bit result is assigned to data_out.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 10, June 2025
Impact Factor: 7.67

3. Shift Rows: The rows of In accordance with their row index, the states are periodically moved to the left.. The data is then fed into the mix column block, while 128-bit data is retrieved from the sub-byte block.

4. Mix column: Every byte in the state matrix is multiplied by a predetermined modulo number in the mix column block. The first step in doing this is to define a multiplier. The system must produce outputs for 2x and 3x, for example, if xxx is an 8-bit input. These outputs are then merged in a single module to create the necessary polynomial.

$a(x) = (03)*x3 + (01)*x2 + (01)*x + (02)*x1$

Simply shifting the bits to the left is how we define multiply in hardware; for instance, to create 2x, we must shift the bit by 1.The next step involves instantiating both modules within the mul_32 module to enable operations on 32-bit data. Initially, the 32-bit input is segmented into 8-bit temporary variables to facilitate processing. The outputs from the mul_2 and mul_3 modules are represented by m2_tmp_out and m3_tmp_out, respectively, and are XORed in accordance with the polynomial equation described in the Mix Columns block. Ultimately, the complete operation is executed on 128-bit data by instantiating the mul_32 module four times for full integration.

Key generation: In addition to the Encryption and Decryption modules, the Key Expansion Schedule plays a critical role in the AES architecture. This component is essential to the overall strength and security of the AES algorithm. According to the AES standard, the user-provided key is XORed with the initial Plaintext or Ciphertext during the first round. For subsequent rounds, keys generated by the Key Expansion process are used in XOR operations with the data. The expansion procedure follows a predefined algorithm, which includes internal elements such as Rcon and SubWord—also referred to as the G matrix—and a specific key expansion method. To begin, we'll focus on the roles of Rcon and SubWord.

Rcon: For every word generation, the round constant, also known as Rcon, is generated.

Subword: In order to create a changed output word, the SubWord() function applies the S-box substitution to each byte of a four-bit input word. Notably, the generation of SubWords relies on a sequential dependency, where each SubWord is derived using the S-box transformation of a given word, while subsequent words incorporate the previously generated SubWord. This iterative process ensures continuity and structured key expansion within the AES algorithm..

Key expansion: The key expansion module accepts a 128-bit input key and is designed to generate a total of 11 round keys, resulting in an output of 1408 bits (11 × 128 bits). These expanded keys are utilized as follows: the first key is applied in the initial AddRoundKey operation, while the remaining ten keys are used across the ten main encryption or decryption rounds. The generated round keys are labeled sequentially from key_s0 to key_s10.

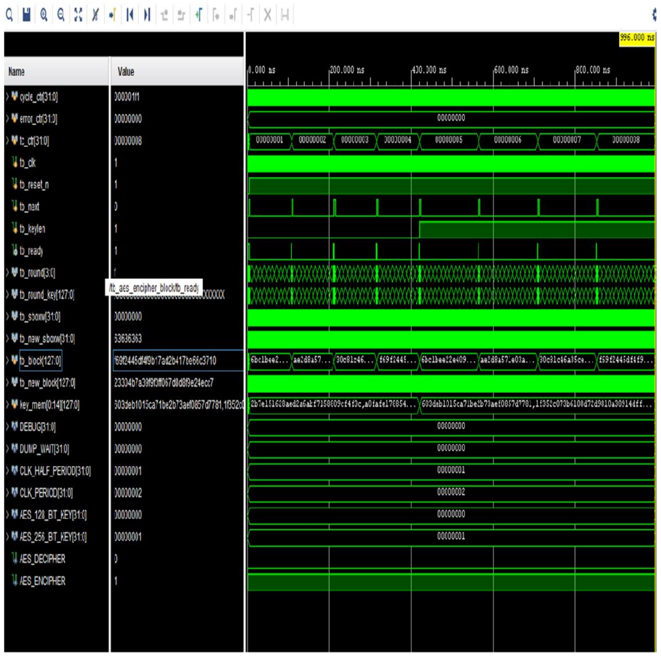
In the AES key expansion process, all operations are performed on 32-bit words. A total of 44 words are generated from the initial 128-bit key. The generation of each new word relies on a combination of previously computed words, the SubWord transformation (derived from the S-box), and the round constant (Rcon). Initially, words **w0** through **w3** are taken directly from the input key and used in the initial AddRoundKey step prior to the first round of encryption.The word **w3** plays a key role in generating the first SubWord, which is subsequently used in computing words **w4** through **w7**. These words are generated by XORing the SubWord (produced from the S-box and denoted by **u0, u1, u2, u3**) with the appropriate round constant from Rcon, and then with the corresponding previous word. This process repeats iteratively: for instance, **w7** contributes to the creation of the next SubWord, enabling the generation of **w8** through **w11**, and so on, until all 44 words are produced. These words are then grouped to form the 11 round keys required by AES-128.

Round Module: The **Round** module integrates the core AES transformations— SubBytes, AddRoundKey, MixColumns, and ShiftRows—into a single encryption cycle. There are ten such rounds in AES-128. All four transformations are used in every round, with the exception of the last one. In accordance with the AES specification, the MixColumns step is skipped in the last round.

AES_Main Module: The AES_Main module represents the top-level integration of the encryption system. It instantiates both the **Key Expansion** and **Round** modules to produce the final AES encryption output. The decryption process is handled similarly, with modules arranged in reverse order to accurately invert the encryption steps.

V. RESULT

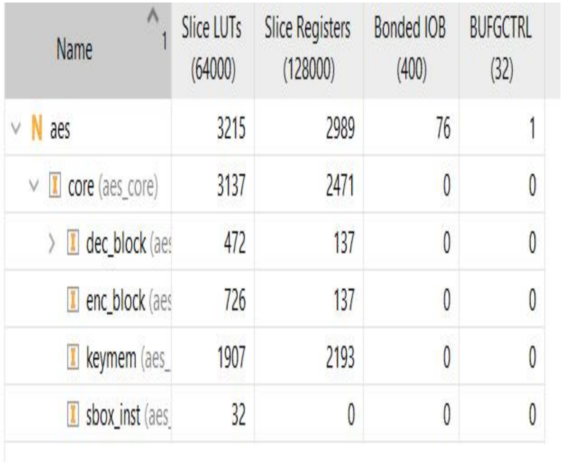
The simulation result displays outputs based on input signals such as clock, key, enable, reset, done and plaintext. The algorithm initializes input and based on the operations made on the input cipher text out is determined. This can be verified in the below image where the cipher text is successfully converted into plain text after Decryption.



Name	Value
cycle_ctr[31:0]	00000111
error_ctr[31:0]	00000000
tc_ctr[31:0]	00000008
tb_clk	1
tb_reset_n	1
tb_next	0
tb_keylen	1
tb_ready	1
tb_round[3:0]	f
tb_round_key[127:0]	XXXXXX
tb_sboxw[31:0]	00000000
tb_new_sboxw[31:0]	63636363
tb_block[127:0]	f69f2445df4f9b17ad2b417be66c3710
tb_new_block[127:0]	23304b7a39f9f3ff067d8d8f9e24ecc7
key_mem[0:14][127:0]	603deb1015ca71be2b73aef0857d7781,1f352c0
DEBUG[31:0]	00000000
DUMP_WAIT[31:0]	00000000
CLK_HALF_PERIOD[31:0]	00000001
CLK_PERIOD[31:0]	00000002
AES_128_BIT_KEY[31:0]	00000000
AES_256_BIT_KEY[31:0]	00000001
AES_DECIPHER	0
AES_ENCIPHER	1

Fig 11 Result of Encryption

The figure below depicts the the resource utilization and power consumption by the design synthesized and implemented on Xilinx Vivado, using **Spartan-7 FPGA** with a **-1Q speed grade** which is from the **Xilinx Spartan-7 family**, designed **for low-power, high-performance applications** with **optimized logic utilization**.



Name	Slice LUTs (64000)	Slice Registers (128000)	Bonded IOB (400)	BUFGCTRL (32)
aes	3215	2989	76	1
core (aes_core)	3137	2471	0	0
dec_block (aes	472	137	0	0
enc_block (aes	726	137	0	0
keymem (aes_	1907	2193	0	0
sbox_inst (aes_	32	0	0	0

Fig. 12 Resource Utilization

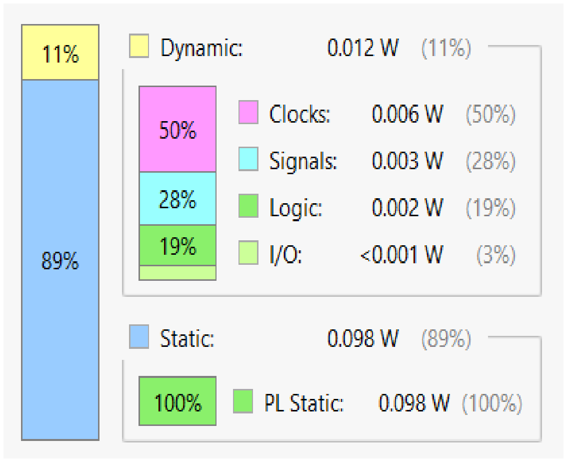


Fig 13 Power Consumption

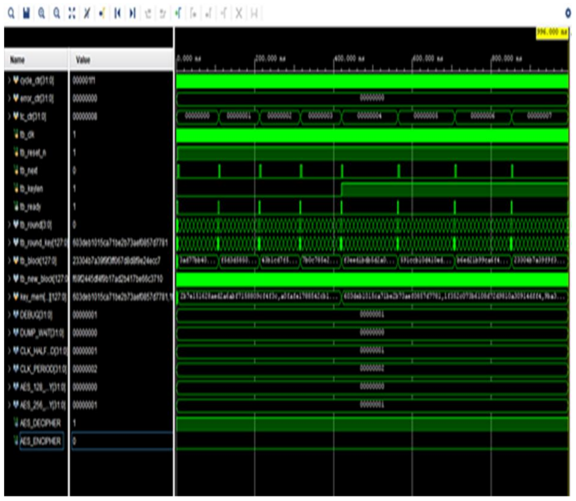


Fig13 Result of Description

VI. CONCLUSION

The implementation of the Advanced Encryption Standard (AES) with a 256-bit key on a Field-Programmable Gate Array (FPGA) board offers a highly secure and efficient solution for cryptographic applications. FPGA-based implementations leverage the parallel processing capabilities of these devices, allowing for accelerated encryption and decryption processes. The AES 256-bit algorithm provides a robust level of security, meeting stringent requirements for data protection in various domains such as communication, finance, and sensitive information storage. Also, it exhibits lower power consumption compared to general-purpose processors, making them suitable for power-constrained environments.

REFERENCES

[1] Busam Swayamprakash, Panditi Jeevana Shreshta, Chittapu Maheshwar Reddy, K.Jamal, Kiran Mannem, M.Suneetha, "Design of advanced encryption standard using Verilog HDL"Proceedings of the 7th International Conference on Trends in Electronics and Informatics (ICOEI 2023) IEEE Xplore Part Number: CFP23J32-ART; ISBN: 979-8-3503-9728-4
[2] Deguang Le, Jinyi Chang, Xingdou Gou, Ankang Zhang, Conglan Lu, "Parallel AES Algorithm for Fast data encryption on GPU" V6-2 2010 2nd international conference on computer engineering and technology.

[3] M.Natheera Banu " FPGA Based Hardware implementation of encryption algorithm " International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249-8958 (Online), Volume-3 Issue-4, April 2014
[4] M.Pitchaiah, Philemon Daniel, Praveen, "Implementation of advanced encryption standard algorithm" International Journal of Scientific & Engineering Research Volume 3, Issue 3, March -2012 1 ISSN 2229-5518
[5] Ganesh Gopal Shet, Jamuna V, Shravani S, Nayana H G, Pramod Kumar S, " Implementation of Advanced encryption standard algorithm ", 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT) | 978-1-6654-0118-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSSIT53264.2022.971641
[6] Zabina Kouser,Manish Singhal and AmitM Joshi "FPGA implementation of Advanced encryption standard algorithm " IEEE international conference on recent advances and innovation in engineering
[7] Hrushikesh S Deshpande, Kailash J karanda Altaaf O Mulani, "Efficient implementation of AES algorithm on FPGA" International conference on communication and signal processing 2014
[8] Selvakumar, K., & Vijayakumar, R. (2017). Design and implementation of AES encryption and decryption algorithm in FPGA using Verilog HDL. International Journal of Engineering and Technology, 9(1), 228-236
[9] Chaitanya Kumar M, K Sangeeta " optimized AES algorithm implementation on FPGA for high speed encryption" IEEE 2020
[10] A Desai M Naik " Efficient expantion Technique for AES on FPGA" IEEE international symposium of circuit system 2020
[11] Vikran R, M, Chaturvedi " Hybrid cryptographic model combining AES and RSA for secure Communication" International Journal of Infromation security Springer 2022
[12]S. Narayan, A. Roy, "Fault Injection Attack Analysis and Countermeasures for AES in Hardware," IEEE International Conference on VLSI Design, 2020.
[13] Ako Muhammad Abdullah, "Advanced encryption (AES) algorithm to encrypt and decrypt data" Cryptography and Network Security 2017
[14] Yehya A. Nasser, Mohammad A. Bazzoun, Samih Abdul Nabi, "AES algorithm implementation for a simple low cost portable 8- bit microcontroller", IEEE 2016.
[15] T. Kumar, V. Gupta, "Low-Power AES Implementation on ASIC for Secure Data Communication," Microelectronics Journal, Elsevier, 2019