AES 128 Bit Optimization: High-Speed and Area-Efficient through Loop Unrolling

Sandarbh Yadav, Gunin Girdhar,
sandarbh.yadav.ug20@nsut.ac.in , gunin.girdhar.ug20@nsut.ac.in
Department of Electronics and Communication Engineering, Netaji Subhas University of Technology (NSUT) *Azad Hind Fauj Marg, Sector-3,*
Dwarka, New Delhi, Delhi 110078, India

Abstract: This study introduces a high-throughput FPGA implementation of AES-128, prioritizing efficiency for robust security and fast data processing needs. AES-128 is renowned for its security and widespread use in various applications. Employing techniques like loop unrolling and pipelining, the implementation maximizes throughput and customizes AES for FPGA architectures. A novel optimization approach, "new-affine-transformation," reduces resource demands and latency for the Sub-Bytes function. The AES architecture is strategically modified for efficiency, with rearranged functions and streamlined processing. The implementation, in VHDL and utilizing Xilinx Virtex-5 FPGA, achieves remarkable performance: 37.9 Gbps (encryption) and 38.5 Gbps (decryption) throughput at frequencies of 296.789 MHz (encryption) and 300.806 MHz (decryption). Resource utilization is efficient, with 264 (encryption) and 260 (decryption) slice registers and 1044 (encryption) and 1581 (decryption) total slices.
Keywords: **AES, FPGA, cryptography, encryption, decryption, throughput, plain text, cipher text**

1. INTRODUCTION

Cryptography, a vital component of information security, focuses on employing mathematical techniques to safeguard privacy, ensure the authenticity of entities, maintain data integrity, and verify data origin. The primary objective of cryptography is to detect and prevent fraudulent or malicious activities. Symmetric-key cryptography involves the utilization of a shared key known only to the parties involved in the communication. One noteworthy attribute of this approach is its utilization of a solitary key for the purposes of encrypting and decrypting data. The Data Encryption Standard (DES) operates using the same key for encryption and decryption, facilitating rapid execution on general-purpose processors or specialized hardware, achieving throughputs exceeding 1 GByte/s. However, DES's reliance on a 56-bit key size raises concerns regarding vulnerability to brute-force attacks, considering the advancements in computer power. In response to these concerns, Triple Data Encryption Standard (3DES) emerged, enabling the use of larger keys to bolster security compared to the relatively modest 56-bit key size of DES. Despite its enhanced security, 3DES suffers from significant drawbacks, notably its sluggish performance in software implementations due to its hardware-oriented design and the use of three times the number of rounds compared to DES. Furthermore, both DES and 3DES share a common limitation of utilizing a 64-bit block size, which compromises efficiency and security. Recognizing the need for a more efficient and secure alternative, the selected algorithm, Rijndael, developed by Joan Daemen and Vincent Rijmen, underwent rigorous evaluation and was officially adopted as the standard in 2001. Advanced Encryption Standard (AES) operates on a substitution-permutation network, executing byte-level substitutions and word-level permutations in each processing cycle. Unlike DES, which employs the Feistel structure, AES's design facilitates swift software implementation owing to its substitution and permutation-based approach.

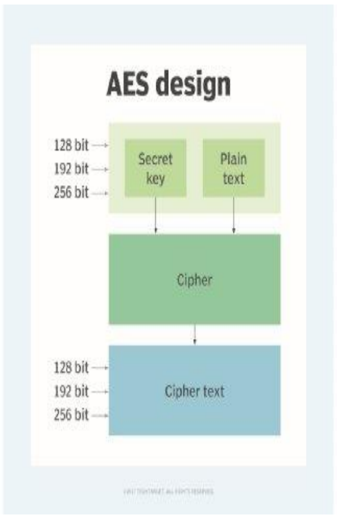


Fig. 1 AES DESIGN

Several researchers' efforts to construct the AES algorithm using field-programmable gate arrays (FPGAs) are highlighted in this section. A number of academics have zeroed down on either speed optimization or area optimization. **[6]** A highly optimized hardware implementation of the Rijndael AES Algorithm was realized on the Xilinx Virtex-5 XC5VLX50 FPGA device, using a modular VHDL approach. The design operates at 339.087 MHz, achieving a throughput of 4.34 Gbps with just 399 slices of the Virtex-5 FPGA, emphasizing both high performance and efficiency. **[7]** This project optimizes area for a masked AES with an unrolled structure, leveraging FPGA block RAM (BRAM) to enhance hardware efficiency. The implementation achieves a significant 36.2% reduction in area footprint, with the main method contributing to a 20.5% decrease and BRAM optimization providing an additional 15.7% reduction. It achieves 40.9 Gbps throughput at 4.5 Mbits/s per slice on the Xilinx XC6VLX240T platform, enhancing defense against DPA and glitch attacks. **[8]** This paper presents an FPGA implementation of the AES-Rinjdael cryptosystem with 128-bit blocks and keys. Synthesis results from the Virtex II Pro Kit FPGA using the Xilinx Synthesis Tool show a computation time of 6,922 nanoseconds for generating ciphertext with AES, utilizing four s-boxes and two dual port RAMs. A synthesizable and optimized VHDL code is developed for encryption and decryption of 128-bit data, validated using Xilinx's ISE 9.2i functional simulator. To reduce hardware usage, an iterative design approach simulates every algorithmic transformation. **[9]** This design, utilizing the XC3S50 0E-4FG320 FPGA device, achieves a remarkable frequency of 222.41 MHz and an outstanding throughput of 2.846 Gbps. With just 2439 slices, it optimizes resource utilization, achieving a high throughput per slice of 1.166 Mbps. Its superior balance among frequency, throughput, and slice efficiency sets it apart from other designs, emphasizing the importance of optimizing these aspects for improved hardware performance in FPGA-based design methodologies. **[10]** The architectures in this paper were executed on reconfigurable platform FPGAs. Successful implementation on Xilinx Virtex4 (device xc4vlx80, package 12ff1148) confirms that the proposed architectures require minimal hardware resources. The AES Encryption and Decryption designs each utilize only 9% of the chip resources at a clock frequency of 382.988 MHz. By employing pipeline techniques, throughput can be increased. **[12]** AES algorithm based on FPGA that is proposed and employs 1746 logic elements and 32768 memory bits. This design was synthesized using Altera on Cyclone-II. The algorithm attains a minimal latency, with encryption throughput measuring 465 Mbit/sec and decryption throughput measuring 189 Mbit/sec. **[13]**, utilizing the XC3S400-FG456 device, exhibited a throughput of 160.875 Mbits/s with 2059 Mbps, utilizing 1403 slices and achieving a high throughput/area ratio of 1.467 Mbps per slice. Although the overall throughput was slightly lower than other designs, the efficient use of available hardware resources, reflected in the high throughput per slice area, indicated a notable achievement in optimizing the design's efficiency. **[14]** The project aims to implement the pipelined AES algorithm with key sizes of 128, 192, and 256 bits for image encryption and decryption. It conducts a comparative analysis covering latency, efficiency, security, frequency, and throughput. The proposed architecture, realized through VHDL programming, utilizes ModelSim for simulation and Xilinx devices for synthesis, placement, and routing. Target devices include the Xilinx Virtex XCV600E-6BG560, Spartan XC6SLX25, and Spartan 3E starter kit FPGA. Achieved maximum frequencies are 385.239, 181.258, and 224.770 MHz, with throughputs of 1232.736, 580.02, and 719.264 Mbit/sec for Encryption and Decryption, respectively. **[15]** This paper presents a reconfigurable platform's rapid and secure AES algorithm implementation. Key generation is done in MATLAB, while design and simulation use Xilinx SysGen, Nexys4, and Simulink. Leveraging offline key generation and an enhanced Xilinx System Generator-based design, the system operates at a maximum frequency of 1102.536 MHz with only 121 slice registers in use. Additionally, it achieves a throughput of 14.1125 Gbps. **[16]** The paper discusses the development of a low-power, high-throughput VLSI architecture for the Advanced Encryption Standard (AES) algorithm, targeting cryptographic applications in high-speed network environments. Efficient implementation of AES in both hardware and software is explored, supporting encryption and decryption with 256-bit keys and achieving a throughput of 0.06 Gbps. VHDL is used for design simulation, and FPGA chips are employed for hardware implementations. The focus is on a reconfigurable hardware implementation of AES using a key expansion approach, emphasizing metrics such as throughput, critical path delay, and power consumption essential for FPGA performance analysis. The proposed implementation with a dual-stage scheme demonstrates a significant reduction in power requirements by up to 43.4% and a decrease in critical path time to 21.4% compared to existing schemes. **[17]** The paper presents an efficient implementation of the Advanced Encryption Standard (AES) algorithm on Field Programmable Gate Arrays (FPGAs) to enhance security and throughput, the implementation focuses on minimizing resource utilization and achieving high throughput through parallel-pipeline design and optimized S-box. Demonstrating a throughput of 97.11Gbps and efficiency of 85.18 Mbps/slice, with significant reduction in resource usage and increased throughput compared to existing work. **[19]** This research paper focuses on enhancing data security using the Advanced Encryption Standard (AES), a technique aimed at safeguarding information from theft or tampering. It centers on developing a specialized electronic chip to expedite and optimize the AES process. Through innovative approaches to mathematical operations and key generation, the researchers devised methods to accelerate the chip's performance while conserving space. Testing of the new chip design demonstrated significantly improved speed compared to traditional methods. **[20]** The paper emphasizes architectural optimization, exploiting pipelining, loop unrolling, and sub-pipelining to increase speed, with a trade-off of increased area. Various methods such as resource sharing and common sub-expression elimination are discussed to reduce critical path and area issues between encryptor and decryptor. The paper details the AES algorithm, key expansion, and various architectures for improving speed, and provides a comparison of pipelining, sub-pipelining, and loop unrolling. It involves the use of techniques such as unrolling, pipelining, and combinational logic for SubBytes/InvSubBytes to tailor performance and area requirements. VHDL and devices like Virtex-E Xilinx Foundation Series 7.li software were utilized for the implementation, with evaluations based on throughput, area cost, and efficiency. The paper also discusses various architectures for the Mix Columns transformation, including methods for efficient implementing Substitute byte operation, Inverse Mix Column Transformation, and detailed analysis of the performance measurements. **[21]** The research aims to develop a high-throughput, FPGA-efficient (FPGA-Eff) cryptosystem tailored for high-traffic applications. To handle substantial workloads effectively, loop-unrolling, inner and outer pipelining techniques are employed. Addressing the resource-intensive nature and latency issues of Substitution bytes (Sub-Bytes) in AES, a novel approach named new-affine-transformation is proposed, integrating inverse isomorphic and affine transformation. AES is further optimized according to the suggested architecture, with strategic modifications such as interchanging Shift-Rows and Sub-Bytes for the initial nine iterations and partitioning Mix-Columns into two stages to achieve stage latency parity. The implementation utilizes VHDL on the Xilinx Virtex-5 platform, achieving a throughput of 79.7 Gbps, FPGA-Eff of 13.3 Mbps/slice, and a frequency of 622.4 MHz. Notably, the proposed layout demonstrates a 22.63% improvement in FPGA-Eff and an 8.02% enhancement in data transmission compared to existing solutions.

2. BASIC AES ALGORITHM

An input block of a solitary 128 bits is utilized for both the encryption and decryption processes in AES. This specific input block is denoted by an octagonal matrix of bytes. A copy of this block is appended to the state array, which is modified during each encryption and decryption operation. Finally, the state is duplicated using an output matrix. Illustration 1 illustrates these procedures. Also represented by the square matrix or grid of data is the 128-bit key. Following this, the 128-bit key is expanded to a list of 44 key scheduling words, of which four bytes comprise each word. The columns of a matrix are utilized to arrange the bytes. To illustrate, suppose the initial column of the matrix comprises the first four bytes of unencrypted inputs amounting to 128 bits to the encryption cipher. The second column would contain the second four bytes, and so forth. The word is composed of the initial four byte values of the expanded key, which are located in the initial column of the w matrix. Its principal design objectives were to demonstrate the subsequent attributes: Protection against all recognized forms of assault - Optimal performance with minimal code footprint across multiple platforms "Simplify Plan Algorithm" AES Procedure.Each encryption or decryption process involves a set number of iterations, where sequential transformations are applied to the bits of a specific data block. The number of iterations is determined by the length of the key, denoted as Nr=10 for a 128-bit key, indicating ten iterations. Each of the initial Nr-1 rounds comprises four operations: Sub-bytes (), Shift Rows (), Mix Columns (), and Add Round Key.The sub-byte transformation process entails replacing each byte in the state independently using a substitution box, generating an invertible S-box through finite field GF (28) multiplication inversion with unresolved polynomials $m(x) = x^8 + x^4 + x^3 + x + 1$. An affine transformation over GF (2^8) is then applied.
Shift rows differ from offsets as they cycle through state rows. The decryption process remains consistent, albeit with unique values assigned to each shifting offset. The Mix Columns Transformation processes columns one by one on the State, treating each column as a four-term polynomial and multiplying it with constant polynomials $a(x) = <03\ x^3 + <01\ x^2 + \sim 02\ x$, within GF ($2^8$) modulo $x^4 + 1$.
During this transformation, a round key is XORed with the state to add it. Each round key contains Nb words in the key expansion, updating state columns with Nb terms. The decryption process similarly employs Key Addition.
Key expansion involves multiplying the 4-word array comprising the round key by the key from the previous round to expand the key. A variable constant is added to the key at each round, and a series of S-Box lookups are performed for each 32-bit word in the key. Nb represents the total words generated from the Key schedule expansion (Nr + 1).

2.1 RIJNDAEL ALGORITHM

A block cipher functions by encrypting and decrypting data through a series of iterative transformations performed across multiple rounds. This process involves the repetitive application of a specific transformation mechanism. These transformations are executed on data blocks, typically of fixed size, in a stepwise fashion. The encryption and decryption operations rely on encryption keys, which dictate the exact transformations applied at each step. These keys come in various lengths, commonly 128, 192, or 256 bits, and serve as crucial components in securing the data. Rijndael, a prominent example of a block cipher, follows this paradigm. It utilizes encryption keys of different lengths and operates on data blocks in increments of 128 bits. These data blocks are represented as one-dimensional arrays of 8-bit bytes, with characters often employed for textual mapping. The encryption key itself is held within a similar array structure. Throughout the encryption and decryption processes, the intermediate cipher results undergo numerous alterations, ensuring the security and integrity of the data. One notable aspect of Rijndael is its support for various key lengths, including 128, 192, or 256 bits, providing flexibility in choosing the level of security required for a given application. Notably, the number of possible AES 128-bit keys greatly exceeds that of DES 56-bit keys, highlighting the enhanced security offered by Rijndael. Additionally, Rijndael employs sophisticated key scheduling techniques to derive subkeys from the primary cipher key, thereby fortifying the encryption process against potential attacks. The byte-level operations within Rijndael, such as matrix manipulations and finite field arithmetic, play a crucial role in the encryption and decryption processes. By treating bytes as polynomials within a finite field, these operations become more manageable and straightforward to implement. This approach enhances both the efficiency and security of the algorithm. In comparison to older encryption standards like DES and Triple DES, Rijndael boasts several advantages. Its alignment of block and key sizes ensures compatibility with modern cryptographic requirements, while its computational efficiency outperforms that of DES. Furthermore, Rijndael offers flexibility in key length and block size, allowing for tailored security solutions to meet specific application needs. The robustness of Rijndael's underlying structure, coupled with its adaptability to different security requirements, makes it a preferred choice for various cryptographic applications. Its resilience against attacks and ability to accommodate evolving security needs contribute to its widespread adoption in both academic and practical settings. Rijndael is well-suited for securely exchanging keys and transmitting data of either 128 or 256 bits in length.

2.2 PIPELINING VS LOOP UNROLLING

There are two methods for implementing hardware: loop-unrolling and pipelining:
In order to process each input data block concurrently in each processing element, registers are placed between all combinational processing unit in a pipeline. The following form illustrates a pipelined version of the Advanced Encryption Standard (AES) algorithm, with each round representing an ith round of the process.

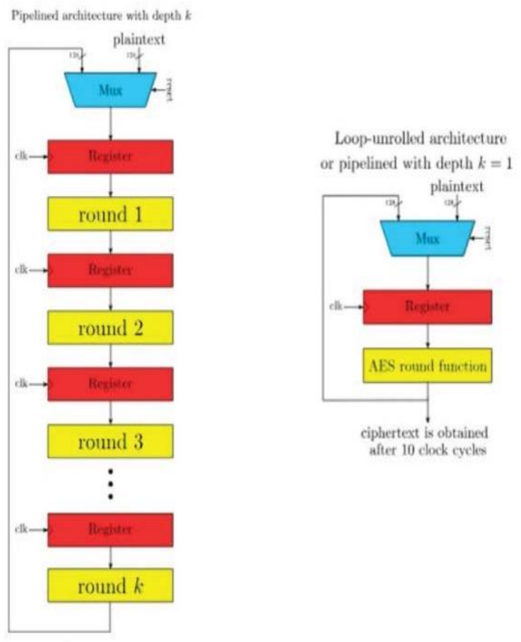

Pipelined architecture with depth k
plaintext
Mux
clk
Register
round 1
round 2
round 3
round k
Loop-unrolled architecture or pipelined with depth $k = 1$
AES round function
ciphertext is obtained after 10 clock cycles

Fig. 2 pipelining vs loop unrolling

A completely pipelined architecture is one that uses the AES-128 cypher and can process every one of the blocks of ten rounds at once. The cypher has 10 rounds. The AES-128 algorithm requires 10 128-bit data registers in a fully pipelined implementation. The greater the number of data blocks that can be processed concurrently, the greater the number of registers and, therefore, the space required for implementation. A loop-unrolling approach, in comparison with pipelining, processes one or more rounds of an algorithm in a single clock cycle. As shown in the accompanying design, the simplest form for a loop unrolled execution of AES uses a data register for storing the result from the preceding clock cycle and just a single iteration of the algorithm as a combinational circuitry processing element. In contrast to a completely pipelined design, which allows for the entry of fresh plaintext into the encryption process every clock cycle, this one requires 10 clock cycles. While loop-unrolled design uses less space, pipelined architecture uses more, while having higher throughput. For situations when space is not an issue, a completely pipelined design provides optimal performance. For applications with limited space, the simplest form of loop-unrolled—also known as the round-based implementation—is preferable than fully pipelined design since it utilizes the least amount of room. Although it is not too difficult to change this code to its pipelined equivalent, it is a loop-unrolled implementation.
The main difference between pipelining and loop unrolling lies in their purpose, application, and abstraction level. Pipelining is primarily employed at the hardware level, such as in CPU architectures, to increase throughput by executing multiple instructions concurrently in stages. In contrast, loop unrolling is a software optimization technique applied during compilation to enhance loop performance by minimizing overhead and potentially exposing optimization opportunities like instruction-level parallelism. While pipelining operates at a lower level of abstraction, involving hardware design and implementation, loop unrolling operates at a higher level, transforming source code to optimize loops. While both techniques aim to improve performance, pipelining focuses on maximizing throughput by overlapping instruction execution, while loop unrolling targets loop efficiency and reduction of overhead.
In summary, while both pipelining and loop unrolling aim to improve performance, they do so at different levels of abstraction and are applied in different contexts: pipelining in hardware design and loop unrolling in software optimization.

2.3 AES ENCRYPTION

2.3.1 ADD ROUND KEY

The state is augmented by one round key through the utilization of a bit-wise exclusive-OR (XOR) operation in the Add Round Key transformation. The Round Add Key is illustrated in the figure below. Decryption and encryption both utilize the identical transformation.

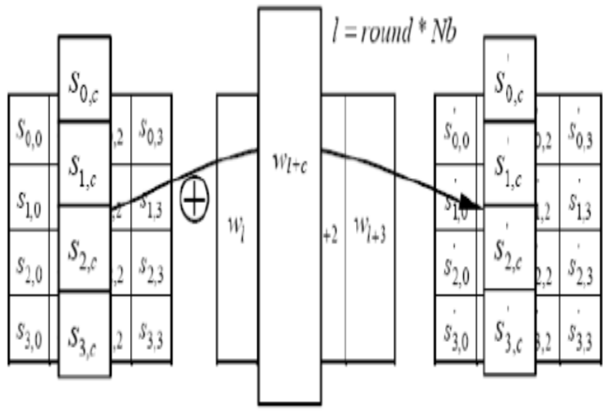
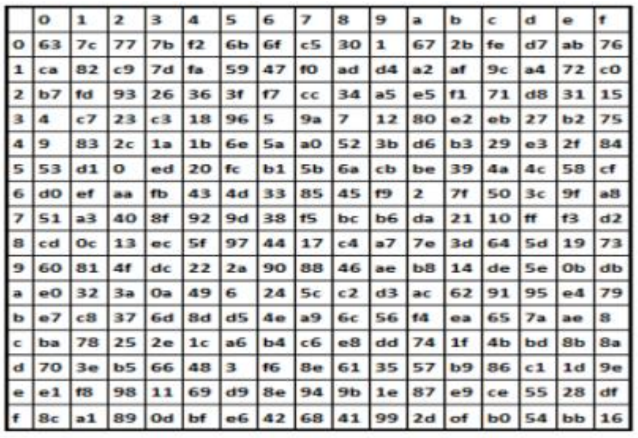


Fig. 3 Add round key

2.3.2 SUBSTITUTIVE BYTES

Sub Bytes is an operation that swaps bytes in a nonlinear fashion. In accordance with the substitution box (also known as the S-box), one byte is substituted for every byte in the input state. To calculate the S-box, a bit-wise affine transformation and the multiplicative inversion in the finite field GF (2^8) are used.



	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	1	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	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
4	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	2	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	6	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	8
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	3	f6	8e	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	of	b0	54	bb	16

Fig 4.1 S-Box

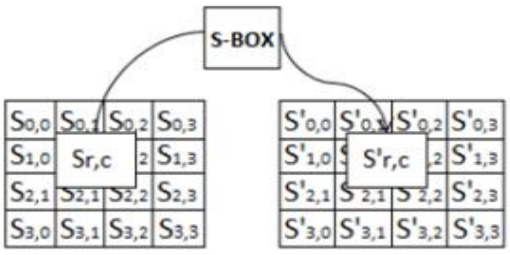

S-BOX
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{r,c}$ $S_{1,3}$
$S_{2,1}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S'_{0,0}$ $S'_{0,1}$ $S'_{0,2}$ $S'_{0,3}$
$S'_{1,0}$ $S'_{r,c}$ $S'_{1,3}$
$S'_{2,1}$ $S'_{2,1}$ $S'_{2,2}$ $S'_{2,3}$
$S'_{3,0}$ $S'_{3,1}$ $S'_{3,2}$ $S'_{3,3}$

Fig.4.2 S-Box

2.3.3 SHIFT ROWS TRANSFORMATION

In the Shift Rows, a cyclic shift operation is performed on each row of the state. The bytes in the initial row of the state remain unaltered throughout this procedure. The figure illustrates a cyclic progression of one byte to the left in the second row, two bytes in the third row, and three bytes in the fourth row. When inv Shift Row is implemented, the procedure is carried out in the opposite order of Shift Rows.

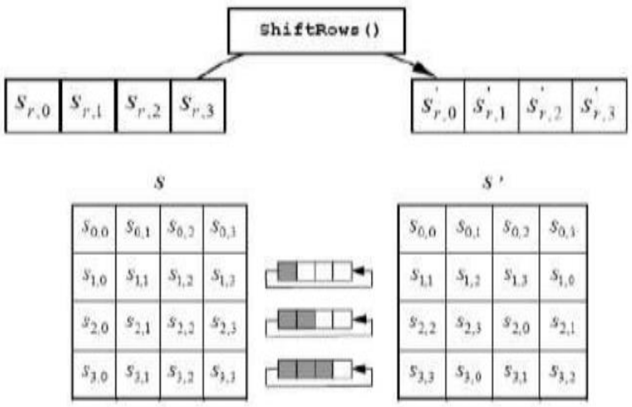

ShiftRows()
$S_{r,0}$ $S_{r,1}$ $S_{r,2}$ $S_{r,3}$
$S'_{r,0}$ $S'_{r,1}$ $S'_{r,2}$ $S'_{r,3}$
S
S'

Fig.5 Shift Row

2.3.4 MIX COLUMN

Each column of the state undergoes the Mix Column transformation separately. The four-term polynomial over GF (2^8) is multiplied by for each column.

a(x) modulo (x^4 + 1) where a(x) = {03}x^3 + {01}x^2 + {01}x + {02}

The expression for this transformation in matrix form is

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

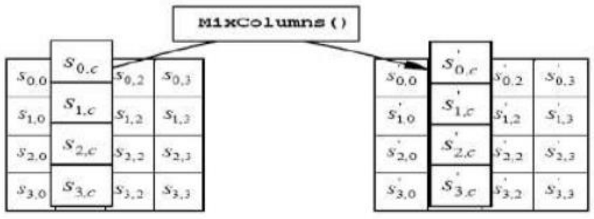
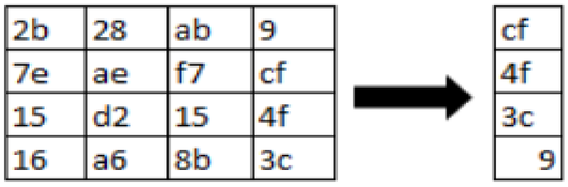

MixColumns()
$s_{0,0}$ $s_{0,c}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,2}$ $s'_{3,3}$

Fig. 6 Mix Column

2.3.5 KEY SCHEDULING FUNCTION

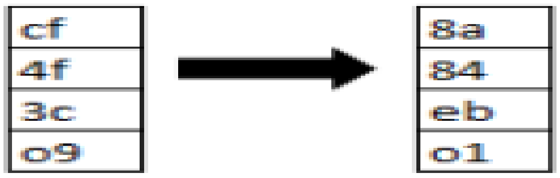
All of the Round Keys, used in every round, are derived from the initial secret input key, and the spawning manoeuvre is key expansion. An encryption key's first round is known as the original key. When decrypting, the original key is the last one produced via key expansion. Following the previous description, the plain text input will be followed by the secret round key before the repeated phases of encryption or decryption begin. Key sizes of 128 bits will generate 10 sets of 24-byte round keys. The procedure for producing ten iterations of the round key is detailed below: Swap the items in the fourth column of the (i-1) key so that they are all moved up one row. An example of this is shown below.



2b	28	ab	9
7e	ae	f7	cf
15	d2	15	4f
16	a6	8b	3c

cf
4f
3c
9

As the example below demonstrates, the column values are subsequently replaced with SBox table values in a manner analogous to the Sub Bytes Transformation round.



cf
4f
3c
09

8a
84
eb
01

A row constant denoted by Rcon is used to perform an XOR operation on the modified byte matrix. Rcon depends on every round. The ith key's ultimate state is described below.

Rcon =

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

The preceding output is XOR-ed with the initial column of the (i-1)th key.

The first column of the ith key is then XOR-ed with the second column of the (i-1)th key. As illustrated below.

$$\begin{matrix} 28 \\ ae \\ d2 \\ a6 \end{matrix} \quad \boxed{\oplus} \quad \begin{matrix} ao \\ fa \\ fe \\ 17 \end{matrix} \quad = \quad \begin{matrix} 88 \\ 54 \\ 2c \\ b1 \end{matrix}$$

Apply the identical process to the remaining columns (2).This will yield the comprehensive key for the current iteration.

$$\begin{matrix} 2b & 28 & ab & o9 \\ 7e & ae & f7 & cf \\ 15 & d2 & 15 & 4f \\ 16 & a6 & 88 & 3c \end{matrix} \quad \Rightarrow \quad \begin{matrix} a0 & 88 & 23 & 2a \\ fa & 54 & a3 & 6c \\ fe & 2c & 39 & 76 \\ 17 & b1 & 39 & 5 \end{matrix}$$

For every one of the ten keys, this same process is repeated. Since the (10-i)th round of decryption is equal to the ith round key of encryption, it is necessary to save the keys in advance.

2.4 INTRODUCTION TO AES DECRYPTION

To get the original plaintext from an encrypted cipher-text, decryption works in the opposite direction of encryption and uses inverse round transformations. Figure (b) depicts the decryption procedure, which involves using the key to perform a basic conversion from encrypted text to plain text. In order to decrypt data in a round fashion, the following functions are used: Add Round Key, Inv Mix Columns, Inv Shift Rows, and Inv Sub Bytes. decryption using AES, as it is shown here.

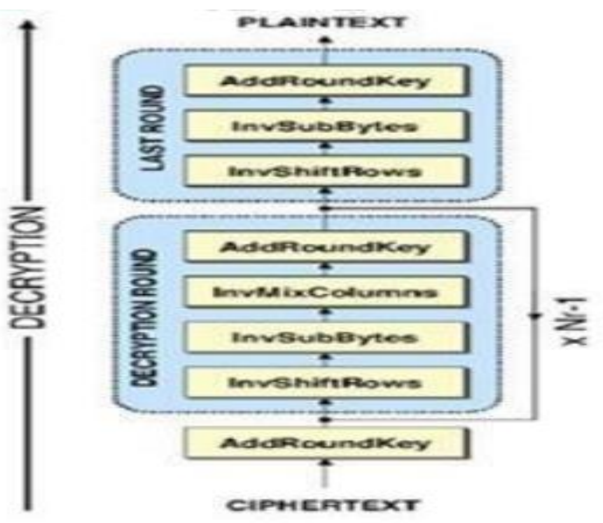


Fig. 7 AES Decryption

2.4.1 ADD ROUND KEY TRANSFORMATION

Similar to how the XOR operation has an inverse operation, the Add Round Key operation also has an inverse operation. To apply the round keys, it is necessary to select them in the opposite direction.

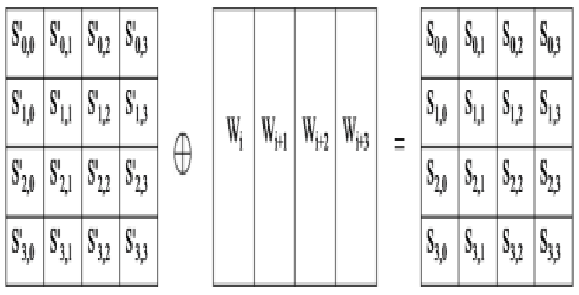


Fig. 8 Add round key

2.4.2 INVMIX-COLUMN TRANSFORMATION

The InvMix-Columns transformation involves the multiplication of coefficients, represented by elements within the state columns, by a predetermined polynomial modulo (x^4 + 1). This multiplication occurs within polynomials of degree less than 4 over the Galois Field GF (2^8). The fixed polynomial utilized for this operation is denoted as d(x) = {0b}x^3 + {0d}x^2 + {09}x + {0e}, where {0b}, {0d}, {09}, and {0e} signify hexadecimal values.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb.$$

Fig. 9 Inverse Mix Column

2.4.3 INVSHIFT ROWS TRANSFORMATION

Shift Rows and InvShift Rows operate identically, albeit in the opposite orientation. While the first row remains unchanged, the right-ward shifts of the second, third, as well as fourth rows are one, two, and three bytes, respectively. The figure illustrates the InvShift Rows Transformation.

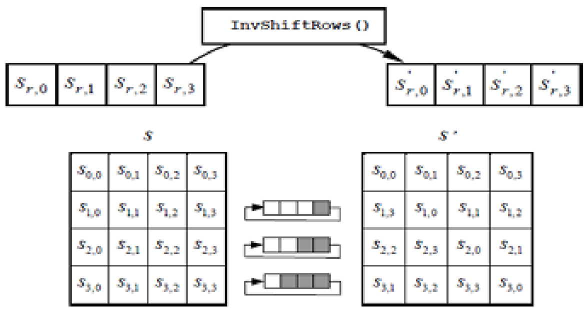
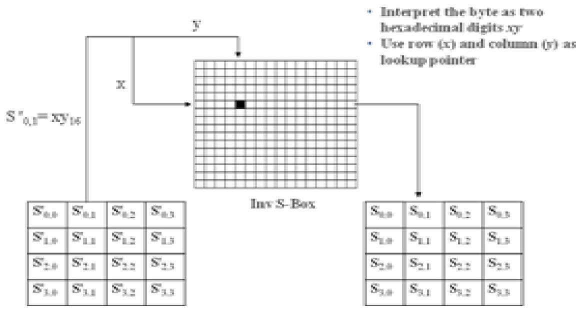
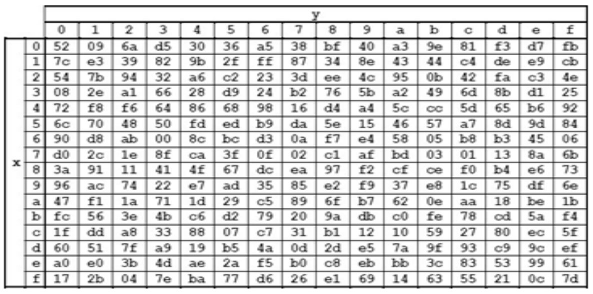


Fig. 10 Inverse Shift Rows

2.4.4 INVSUB-BYTES TRANSFORMATION

The Inv Sub-Bytes transformation is executed utilizing the Inv S-box, which is a substitution table that was previously calculated. The Inv S-box table comprises 256 numbers ranging from 0 to 255, with their respective values listed in the table.


Interpret the byte as two hexadecimal digits xy
Use row (x) and column (y) as lookup pointer
y
x
$S'_{0,1} = xy_{16}$
Inv S-Box
$S'_{0,0}$ $S'_{0,1}$ $S'_{0,2}$ $S'_{0,3}$
$S'_{1,0}$ $S'_{1,1}$ $S'_{1,2}$ $S'_{1,3}$
$S'_{2,0}$ $S'_{2,1}$ $S'_{2,2}$ $S'_{2,3}$
$S'_{3,0}$ $S'_{3,1}$ $S'_{3,2}$ $S'_{3,3}$
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$



		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. 11 Inverse Sub-Bytes

3. PROPOSED ARCHITECTURE

The primary goals of this research are to investigate existing AES acceleration techniques, identify their limitations, and explore innovative methods for optimizing AES performance. One of the methods we explore in this study is loop unrolling. Figure 12 illustrates the block diagram of an AES (Advanced Encryption Standard) encryption process. It begins with the plaintext and the key, each entering separate multiplexers (Mux) that select the input between the new data and a reset signal. These inputs are then fed into registers, synchronized by a clock signal (clk). The plaintext register's output is XORed with the subkey generated from the key schedule round function, producing the ciphertext.

The subsequent steps involve the core AES transformations: SubBytes, ShiftRows, and MixColumns. SubBytes performs a non-linear substitution, transforming each byte individually. ShiftRows shifts the rows of the state array cyclically. MixColumns mixes the columns of the state, providing diffusion. These steps are iterated in rounds controlled by a counter within the controller section, which also ensures the correct number of rounds.After the rounds, another Mux determines if the MixColumns operation should be bypassed for the final round, which differs slightly from the other rounds in the AES process. The key schedule round function generates the required subkeys for each round, orchestrated by the controller to align with the AES round transformations.

The controller also includes a register, multiplexer, and specific constants (0x36, 0x01, 0x6C) to manage the round operations and signal when the encryption process is complete. The final ciphertext is produced after the last round of transformations and exits the system. By implementing loop unrolling techniques in AES encryption and decryption routines, we aim to achieve significant speedups while maintaining the robust security properties of the algorithm. Figure 13 illustrates the block diagram the AES decryption process. The process starts with the ciphertext and the decryption key, which are fed into separate multiplexers (Mux) that select between the new data and a reset signal. These inputs are then loaded into registers, synchronized by a clock signal (clk). The ciphertext register's output is XORed with the subkey generated from the key schedule round function, producing the intermediate decryption result. The core AES decryption transformations follow, which include InvSubBytes, InvShiftRows, and InvMixColumns. InvSubBytes performs the inverse of the byte substitution step, transforming each byte individually back to its original form. InvShiftRows reverses the cyclic row shifts applied during encryption. InvMixColumns reverses the column mixing, undoing the diffusion applied during encryption. These steps are iterated over multiple rounds as controlled by the round counter within the controller section. The controller uses a lookup table and a 4-bit counter to manage the decryption rounds and ensure they occur the correct number of times. The controller also incorporates specific constants (0x36 and 0x00) to manage the initialization and completion of the decryption process. After the appropriate number of rounds, another Mux determines if the InvMixColumns operation should be bypassed for the first round, which differs slightly from the other rounds in the AES decryption process. The key schedule round function, in this case, generates the required subkeys for each decryption round, aligned with the AES decryption transformations.

The final plaintext is obtained after the last round of inverse transformations and is outputted from the system. The decryption process is completed when the controller signals that all rounds have been executed. . The area efficiency achieved by the proposed architecture through the use of loop unrolling makes it the optimal choice for implementing sequential data blocks in a small footprint.

3.1 MODIFIED AES ALGORITHM

The AES algorithm consists of a series of operations performed in nine rounds. The operations are done in the following order: add-round-key, sub-bytes, shift-rows, mix-columns, and add-round-key. After the first nine rounds, the operations sub-bytes, shift-rows, and add-round-key are executed.

In the modified version, which is shown in fig. 12 & 13. We have introduced a key scheduling function to enhance and facilitate the AES algorithm more efficiently and robustly.

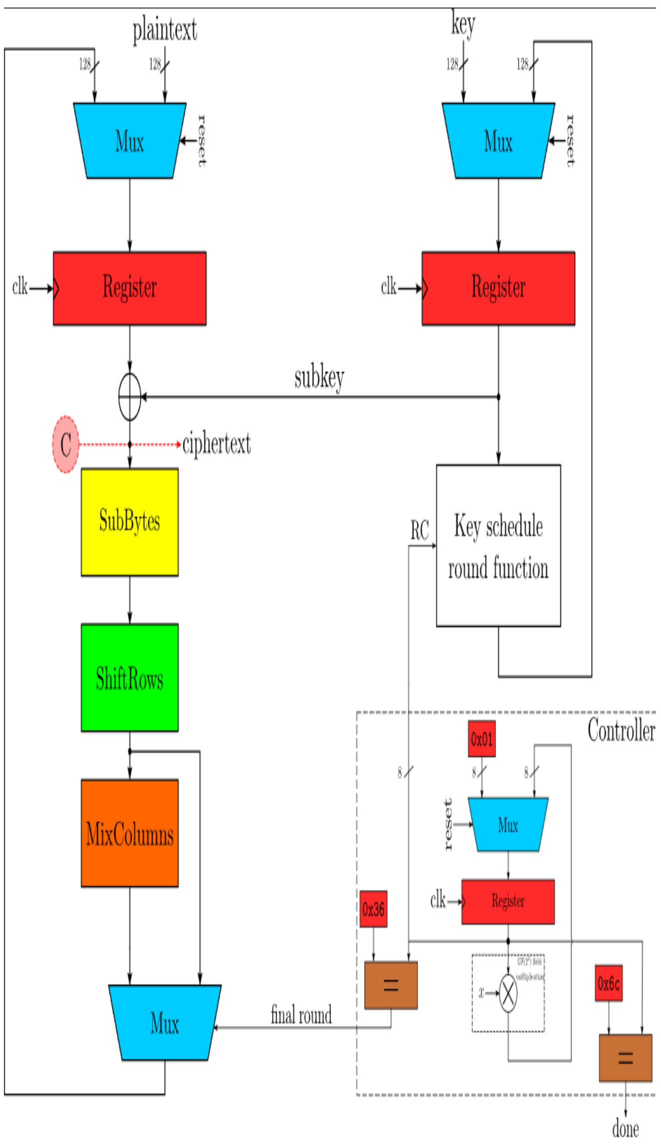


Fig. 12 Proposed Aes Encryption

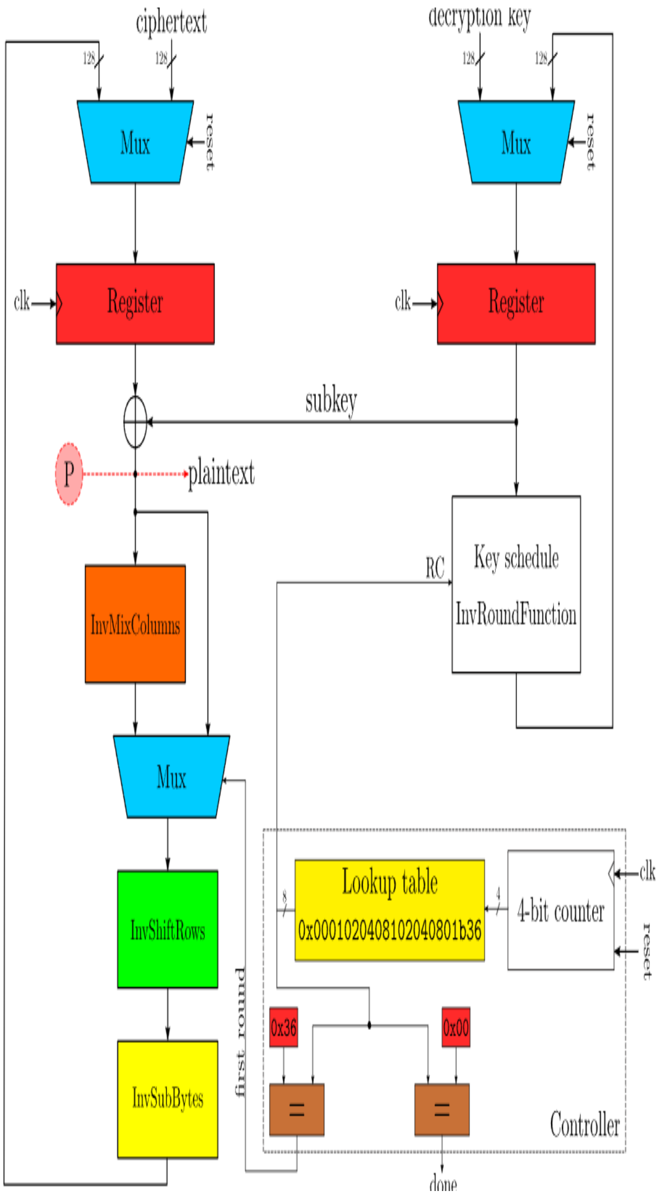


Fig. 13 Proposed Aes Decryption

3.1.1 KEY SCHEDULING FUNCTION

The key scheduling process is crucial in cryptographic operations such as encryption and decryption. Upon initialization signalled by reset assertion, the module loads the initial key into the register. Clock cycle processing entails two main operations. Firstly, during each cycle, the register's current state is passed through the round function. If its post-reset, the initial key is utilized; otherwise, the previous round key is employed. Secondly, the round function computes the next sub-key using the current sub-key and a round constant, storing it in the feedback signal. This newly generated sub-key becomes the input for the subsequent cycle, facilitating a feedback loop ensuring each round key is derived from the prior one. Consequently, the round key output continuously furnishes the current round key for cryptographic operations. The process initializes by loading the initial key, iteratively computes subsequent round keys, updates the register with each generated key, and provides the current round key for cryptographic functions, forming a robust and iterative key scheduling mechanism.

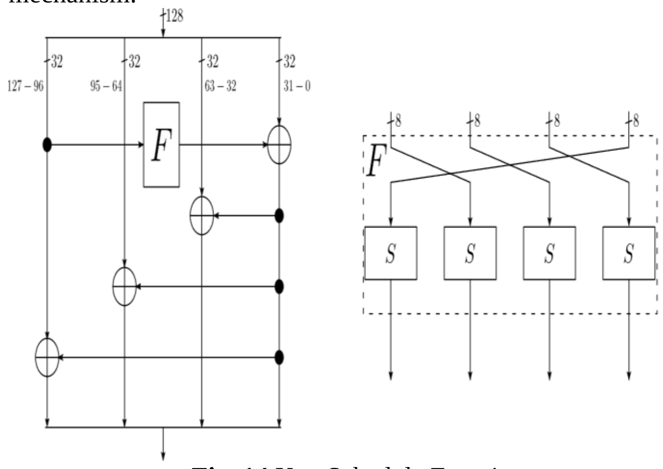


Fig. 14 Key Schedule Function

4. RESULTS AND COMPARISON

Ref. No.	Device Model	Frequency (MHz)	Slice Registers	Throughput (Gbps)
this work	XC7Z100FFG1156-2	296.789(enc) 300.806(Dec)	264(enc) 260(dec)	37.9(enc) 38.5(dec)
[7]	XC6VLX240T	319.5		40.9
[9]	XC3S500E4FG320	222.41	2439(enc) 2635(dec)	2.846
[15]	NEXYS 4	1102.536	121	14.1125
[16]	XC5VLX30	277.4	5493	3.5
[19]	XC5VLX110T	322.7	3012	41.31

In Our Research On Aes-128 Utilizing Loop Unrolling, We Have Achieved Notable Advancements In Throughput - 37.9 Gbps (Enc), 38.5 Gbps (Dec), Frequency - 296.789 Mhz (Enc), 300.806 Mhz (Dec), Slice Registers – 264 (Enc), 260 (Dec) And Slices – 1044 (Enc), 1581 (Dec) Compared To Some Existing Studies: -

[7] Throughput – 40.5 Gbps, Frequency –319.5 MHz.

[9] Throughput – 2.846 Gbps, Frequency – 222.41 MHz, Slices – 2439.

[15] Throughput –14.1125 Gbps, Frequency – 1102.536 MHz, Slice Registers –121.

[16] Throughput – 3.5 Gbps, Frequency – 277.4 MHz, Slices - 5493.

Our implementation showcases a substantial increase in throughput, thanks to the efficient reduction of loop overhead and optimized memory access patterns enabled by loop unrolling. This improvement in throughput translates to enhanced data processing speeds, making our aes-128 implementation well-suited for high-performance computing environments.

Furthermore, our research demonstrates superior frequency performance, indicating the capability of our implementation to operate at higher clock frequencies compared to some previous approaches. By minimizing redundant loop control operations and maximizing instruction-level parallelism through loop unrolling, we have effectively reduced critical path delays and improved overall frequency scalability.

In terms of resource utilization, our aes-128 implementation using loop unrolling exhibits efficient utilization of slice LUT (look-up table) resources, a crucial consideration in FPGA (field-programmable gate array) implementations. By carefully optimizing loop unrolling factors and exploiting hardware resources judiciously, we have achieved a balance between performance and resource efficiency, making our implementation highly competitive in resource-constrained environments.

Overall, our research presents a significant advancement in aes-128 implementations, offering improved throughput, frequency performance, and resource utilization compared to some prior studies. By leveraging the benefits of loop unrolling, we have developed a high-performance and resource-efficient aes-128 implementation that holds promise for various applications requiring secure and efficient data encryption.

4.1 RESULT DISCUSSION

Utilizing the Vivado 2018.2 tool, we conducted synthesis and timing analysis. Employing loop unrolling in our design, we achieved a clock cycle of 330.579MHz. Following verification of each module's functionality, integration becomes feasible. In support of this approach, we partitioned the AES algorithm into distinct encryption and decryption modules. The chip synthesis occurred within the Vivado 2018.2 environment, targeting ZYNQ technology (specifically the xc7z100ffg1156-2 target device), with detailed results depicted in Figures 2.4.2.2 and 3.6.2.2. Throughout the entirety of the process, Vivado 2018.2 remained the tool of choice. Subsequent to synthesizing in VHDL style, we derived individual Register Transfer Logic (RTL). Additionally, a timing simulation was conducted to validate the functional integrity of our design.

5. CONCLUSION

Our research has focused on optimizing AES encryption and decryption speeds through the exploration of loop unrolling techniques while considering various implementation strategies. Utilizing Vivado 2018 software, we successfully implemented AES 128-bit encryption and decryption algorithms, achieving notable throughputs of 38 Gbps for encryption and 38.5 Gbps for decryption with a fully pipelined design operating in Electronic Codebook (ECB) mode. Our findings demonstrate that manual loop unrolling effectively reduces loop iterations and enables better compiler optimizations, leading to significant performance improvements while maintaining robust security properties. Additionally, achieving minimum periods of 3.324ns for decryption and 3.369ns for encryption, corresponding to maximum frequencies of 300.806MHz and 296.789MHz respectively, underscores the efficiency and reliability of our AES 128-bit implementation. These results contribute valuable insights into AES acceleration methodologies, emphasizing the importance of optimizing performance without compromising security. Furthermore, they highlight the potential for rapid data processing in security-critical scenarios. Moving forward, further refinement and optimization of these methodologies hold promise for even greater improvements in throughput and efficiency, laying the groundwork for enhanced cryptographic systems in the future.

6. REFERENCES

[1] Alex Panato, Marcelo Barcelos, Ricardo Reis, “An IP of an Advanced Encryption Standard for Altera Devices”, SBCCI 2002, pp. 197-202, Porto Alegre, Brazil, 9 and 14 September 2002.

[2] Arshad Aziz and Nassar Ikram, “Memory efficient implementation of AES S-boxes on FPGA”, Journal of Circuits, Systems, and Computers, Vol. 16, No. 4, pp. 603-611, 2007.

[3] Yang Jun, Ding Jun, Li Na, Guo Yixiong “FPGA-based design and implementation of reduced AES algorithm,” 978-0-7695-3972-0/2010 IEEE DOI 10.1109/CESCE.2010.123.

[4] A. Amaar, I. Ashour and M. Shiple “Design and Implementation A Compact AES Architecture for FPGA Technology”, World Academy of Science, Engineering and Technology 59 2011.

[5] Ai-Wen Luo, Qing-Ming Yi, Min Shi, “Design and Implementation of Area-optimized AES Based on FPGA”, 978-1-61284-109-0/11/2011 IEEE.

[6] Muhammad H. Rais and Syed M. Qasim , “Efficient Hardware Realization of Advanced Encryption Standard Algorithm using Virtex-5 FPGA”, International Journal of Computer Science and Network Security (IJCSNS) , VOL.9 No.9, September 2009.

[7] Yi Wang and Yajun Ha, Senior Member, IEEE, “FPGA-Based 40.9-Gbits/s Masked AES with Area Optimization for Storage Area Network”, IEEE Transaction On Circuits And Systems—II: Express Brief, VOL. 60, No. 1, January 2013.

[8] Amandeep Kaur, Puneet Bhardwaj, Naveen Kumar, “FPGA Implementation of Efficient Hardware for the Advanced Encryption Standard”, International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-2, Issue-3, February 2013.

[9] Leelavathi.G, Prakasha S, Shaila K, Venugopal K R , L M Patnaik[9],“Design and Implementation of Advanced Encryption Algorithm with FPGA and ASIC”,IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 3, June-July, 2013ISSN: 2320 – 8791.

[10] Adnan Mohsin Abdulazeez, And Ari Shawkat Tahir, “Design and Implementation of Advanced Encryption Standard Security Algorithm using FPGA”, International Journal of Scientific & Engineering Research, Volume 4, Issue 9, September-2013 1988ISSN 2229-5518.

[11] Alia Arshad, Kanwal Aslam, Dur-e-Shahwar Kundi and Arshad Aziz, “FPGA Implementation of Advance Encryption Standard Using Xilinx System Generator”, Asian Journal of Applied Sciences (ISSN: 2321 – 0893) Volume 02 – Issue 02, April 2014.

[12] Sonali A. Varhade N. N. Kasat, “Implementation of AES Algorithm Using FPGA &Its Performance Analysis”, International Journal of Science and Research (IJSR)ISSN (Online): 2319-7064.

[13] YewaleMinal J, M. A. Sayyad, “Implementation of AES on FPGA”,IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)Volume 4, Issue 5, Ver. II (Sep-Oct. 2014), PP 65-69 e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197.

[14] Pravin V. Kinge, S.J. Honale, Prof. C.M. Bobade, “Design of AES Pipelined Architecture for Image Encryption/Decryption Module”, International Journal of Reconfigurable and Embedded Systems (IJRES)Vol. 3, No. 3, November 2014, pp. 114~118 ISSN: 2089-4864.

[15] P. B. Mane, A. O. Mulani, “High Speed Area Efficient FPGA Implementation of AES Algorithm”, International Journal of Reconfigurable and Embedded Systems Vol. 7, No. 3, November 2018, pp. 151~159 ISSN: 2089-4864, DOI: 10.11591/ijres.v7.i3.pp151-159.

[16] K.Kalaiselvi, H.Mangalam "Power efficient and high performance VLSI architecture for AES algorithm" Journal of Electrical Systems and Information Technology" vol 2, issue 2, 2015

[17] Sarita Sanap, Vijayshree More "An Ultra-High Throughput and Efficient Implementation of Advanced Encryption Standard" International Journal of Electrical and Electronic Engineering & Telecommunications"

[18] Naveen Kumar Dumpala, Shivu Kumar B.Patil, Daniel Holcomb, Russell Tessier "Loop Unrolling for Energy Efficiency in Low-Cost Field-Programmable Gate Arrays" University of Massachusetts Amherst

[19] Anuroop K.B, Neema M "Fully Pipelined-Loop unrolled AES with Enhanced Key Expansion" IEEE International Conference On Recent Trends In Electronics Information Communication Technology, May 20-21, 2016

[20] Nalini C, Nagaraj, Dr. Anandmohan P.V, Poornaiah D.V, V.D.kulkarni "An FPGA Based Performance Analysis of Pipelining and Unrolling of AES Algorithm"

[21] Karim Shahbazi, Seok-Bum Ko "High throughput and area-efficient FPGA implementation of AES for high-traffic applications" ,The Institution of Engineering and Technology ,1751-8601 15th May 2020

[22] Anup Gujar, “Image encryption using AES algorithm based on FPGA”, International Journal of Computer Science and Information Technologies (IJCSIT), 2014.

[23] M. Hasamnis, P. Jambhulkar and S. Limaye, “Implementation of AES as a Custom”, Advanced Computing: An International Journal (ACIJ), vol.3, No.4, July 2012.

[24] Wang Wei, Chen Jie, XuFei, "An Implementation of AES Algorithm Based on FPGA", 978-1-4673-0024-7/2012 IEEE.

[25] A. Amaar, I. Ashour and M. Shiple "Design and Implementation A Compact AES Architecture for FPGA Technology", World Academy of Science, Engineering and Technology 59 2011.

[26] Marko Mali, Franc Novak and Anton Biasizzo "Hardware Implementation of AES Algorithm" – Journal of ELECTRICAL ENGINEERING, Vol. 56, No. 9-10,2005, 265-269.

[27] Hamdan.O. Alanazi, B.B.Zaidan, A.A.Zaidan, Hamid A.Jalab, M.Shabbir and Y. Al-Nabhani, "New Comparative Study between DES, 3DES and AES within Nine Factors", Journal of Computing, Volume 2, Issue 3, March 2010, ISSN 2151-9617

[28] A.A. Zaidan, B.B. Zaidan, Anas Majeed, "High Securing Cover-File of Hidden Data Using Statistical Technique and AES Encryption Algorithm", World Academy of Science Engineering and Technology (WASET), Vol.54, ISSN: 2070-3724, P.P 468-479.

[29] National Institute of Standards and Technology, Advanced Encryption Standard (AES), Federal Information Processing Standards Publications – FIPS 197

[30] Gaj, Kris, Pawel Chodowiec, "FPGA and ASIC implementations of AES", Cryptographic Engineering, Springer US, 2009, 235-294.