International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Implementation of Reconfigurable Modified Advanced Encryption Algorithm with Area Optimization

[1]Priyank Patel, [2]Nitin Bathani, [3]Ketu Patel, [4]Nirav Patel, [5]Sameer Mansuri, [6] Bhavik Brahmbhatt

Submitted: 07/02/2024 **Revised**: 15/03/2024 **Accepted**: 21/03/2024

Abstract*:* With the rise of computer technology and the Internet in the era of IoT and Industry 4.0, ensuring information security has become paramount. The Advanced Encryption Standard (AES) has emerged as a leading cryptography standard, garnering significant attention from hardware designers. IOT end devices requires low power and fast processing hence modification in AES is needed. This paper presents a novel implementation of AES, focusing on minimizing hardware area. Utilizing Vivado 2015.4, we conduct simulations and compare results, exploring coding methodologies. Our design demonstrates superior resource efficiency compared to others, addressing the critical need for secure communication in modern interconnected systems. The papers also showcases the various methods that can further increase the effectiveness and minimizes the area utilization.

Keywords: Advanced Encryption System, Galois field, Mix column, key expansion, Area Optimization, IOT application

I. INTRODUCTION

The Internet and communication technology have rapidly advanced in recent years, leading to the widespread integration of intelligent gadgets into all aspects of human existence [1]. The Internet of Things (IoT) enables the connectivity of things located in various places by utilizing high-speed computer network processing and wireless data communication. This further promotes the widespread adoption and advancement of the information age. Currently, the Internet of Things (IoT) is extensively employed in industrial monitoring, urban management, smart home systems, and intelligent transportation [2,3,4]. Moreover, it is projected that by 2020, the number of devices connected by IoT would range from 2 to 5 billion [5,6]. Ensuring information security has been a prominent subject for researchers and technical firms due to the need to prevent information leakage in intricate and dynamic application environments. Cryptographic methods are widely recognized as the most effective method for ensuring data security during the transmission of IoT data [7,8,9]. The increasing data volumes and the commoditization of the Internet of Things (IoT) necessitate computer technology to meet increased demands in terms of speed, power consumption, and cost. Furthermore, in order to augment the level of difficulty in breaking cryptographic algorithms, the complexity of these algorithms surely places a greater strain on the limited resources of hardware implementation. When comparing the Field Programmable Gate Array (FPGA) with the Application Specific Integrated Circuit (ASIC) and the digital signal processor (DSP), the FPGA, which is a semi-custom circuit in the ASIC field, not only shortens the development time for custom circuits, but also addresses the limitations of the original programmable device's gate capacity. It offers the benefits of rapid processing speed, extensive functionality, great adaptability, and extremely low power consumption. Recently, FPGA has emerged as a suitable platform for Internet of Things (IoT) applications to handle large amounts of data and perform cloud computing [9].

Data encryption in the Internet of Things (IoT) primarily involves the encryption of information and the authentication of messages. This serves to safeguard the security of information and identify any attempts to tamper with it [10]. Various cryptographic algorithms fulfil distinct encryption criteria as a result of their individual characteristics. Thus, the encryption and decryption system must possess a diverse range of algorithms to enable users to use them in a flexible manner. Nevertheless, the encryption and decryption system implemented on the FPGA encounters three challenges: Initially, the incorporation of several algorithms will also augment the utilization of hardware logic resources, potentially surpassing the aggregate quantity of on-chip resources. Furthermore, just a single algorithm is employed for data processing concurrently, although other algorithms continue to consume a significant portion of logical resources, hence diminishing resource utilization. Furthermore, in order to modify and update the cryptographic method, the system is required to halt all ongoing work processes and carry out maintenance activities, resulting in a reduction in

[1,2,3,4,5,6] *Electronics and Communication Engineering Department, Government Engineering College, Modasa, Arvalli, Modasa 383315, India*
[1]Pvp.fetr@gmailcom;[2]nitinbathani@gmail.com,[3]ketu.patel@gecmodasa.ac.in,[4]nirav2009ec@gmail.com,[5]mansurisameer@gmail.com,
[6]bhavik0072009@gmail.com
Corresponding author: Bhavik Brahmbhatt ; bhavik0072009@gmail.com

flexibility. A solution to improve the security of information and the performance of hardware implementation in IoT is offered by the encryption and decryption system that relies on dynamically reconfigurable technology. In our previous study [11], we developed a dynamic reconfigurable control platform that allows the host computer user to choose from multiple cryptographic algorithms. This platform enables dynamic switching of algorithms within the allocated area, without occupying all the hardware resources with all the cryptographic algorithms. Thus, this work enhances the hardware implementation and finalizes the architecture of the Dynamic Reconfigurable Encryption and Decryption System (DREDSMoreover, this study improves the use of AES and 3DES algorithms, as well as the Static Encryption and Decryption System (SEDS), while also enhancing the data on the usage of hardware resources. Due to the rapid advancement of computer technology and the widespread availability of the internet, effortless communication among millions of people has become the standard. The requirement for strong information security has greatly increased, especially in the areas of e-commerce, e-government, and the burgeoning field of IoT and edge devices. The Data Encryption Standard (DES) was previously considered sufficient, but its key length of 56 bits became inadequate. The Advanced Encryption Standard (AES) was developed as a solution to provide heightened security by utilising key lengths of either 128, 192, or 256 bits. AES, invented by Belgian cryptographers, received significant recognition and popularity, particularly after 2006, when it was officially recognised by NIST in 2001. The performance of FPGA technology has been enhanced through optimisation, making it suitable for a wide range of applications, including lower-end devices such as PDAs and embedded systems. There is a strong need for AES solutions that are specifically designed for IoT and edge devices. This has led to the investigation of several architectural approaches. It includes a discussion on an 8-bit AES programme designed for less powerful devices.The remaining content is structured in the following fashion. After the introduction in Section 1.Section 2 introduces the methodology of the advanced encryption standard. Section 3 presents the test system that has been implemented in a real-world setting to compare the algorithms discussed. The simulation results are reported in Section 4. Section 5 finishes by providing a concise overview of the main discoveries and suggestions for further research.

II. ADVANCE ENCRYPTION STANDARD

Typically, symmetric key methods are classified as either block cyphers or stream cyphers. The AES (Advanced Encryption Standard) has become increasingly popular due to the widespread usage of the internet and the need for strong security measures. The AES utilises the "Rijndael" algorithm. The AES encryption method utilises the Rijndael algorithm and functions as a block cypher, processing data in 4x4 matrices as depicted in Figure 1.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Figure.1 State matrix

In AES, the basic unit of processing is a byte. The state matrix is formed by organising a 128-bit plaintext word. The AES algorithm is largely composed of two primary processes: key scheduling and round transformation. Key scheduling is the process of transforming the original key into an extended key, which produces a different number of round keys based on the length of the input key. For example, when using a 128-bit input key, the algorithm generates 10 round keys to be used in 10 rounds. On the other hand, if the input key is 192 or 256 bits, the algorithm generates 12 or 14 round keys, respectively, to be used in the corresponding number of rounds. The mix column state is excluded in the final round. The AES algorithm consists of four distinct operations: Sub-byte transformation, Shift row, Mix columns, and Add round key. These operations are sequentially applied to the state matrix. All of these modifications are carried out on this state matrix. The Sub-byte transform operation replaces the input plaintext with corresponding elements from the Galois field GF(2^8), with each byte as the unit of operation. Byte rotation is the process of rotating individual bytes inside a specific row, using a 32-bit operational unit. The mix column operation involves multiplying the columns of the state matrix by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ in GF(2^8), while also acting on 32 bits. The Add round key operation consists of performing an XOR operation between the round key that was initially calculated and

the state matrix. In this operation, the basic unit used is a byte [2].

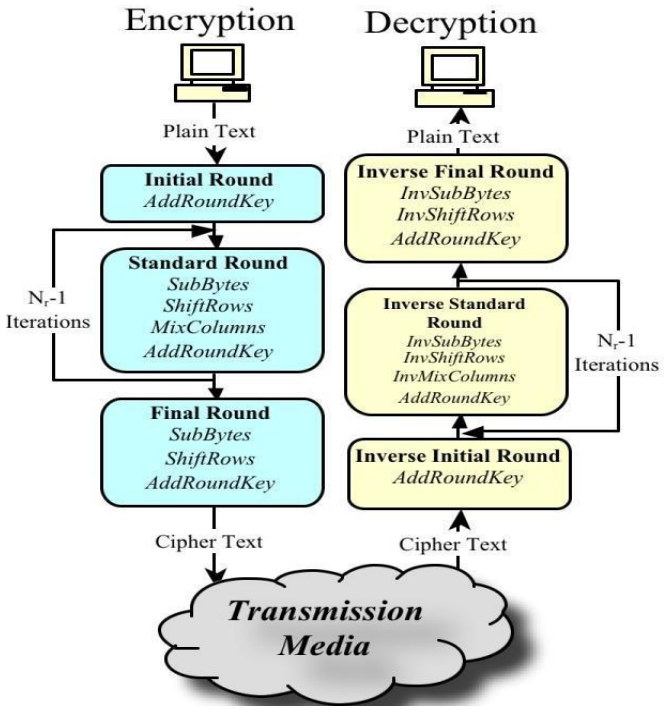


Figure.2 Overall flow encryption and decryption in communication

After calculating the risk-measuring parameters, the aggregator assesses the risk aversion factor and renders a judgement. In other words, the aggregator chooses the best technique by considering the value of the OF. FPGA implementation is not well-suited for classical AES. Several approaches and diverse methodologies have been utilised. AI-WEN LUO, QING-MING YI, and MIN SHI have proposed a paper that utilises pipelining technology. Their proposed structure involves partitioning the plaintext into four 32-bit units. The initial key is also distributed to two distinct components, Key Expansion and Key Selection. The primary purpose of a round transformation is to perform Sub Bytes and Mix Columns operations on 32-bit columns. Each of these four blocks is executed autonomously. The XOR operation is used to combine the output and key source with the 64-bit input port. The Sub Byte function is implemented using a Look-Up Table (LUT). It signifies that the Find and Replace process is finalised once all replacement units have been put in memory. The ultimate stage features a 128-bit CPU. After completing nine rounds of operations, including Shift Rows, Sub Byte, and Mix Columns, the 128-bit intermediate encrypted data will be subjected to an XOR operation with the final enlarged key. This expanded key is obtained via the key expansion module [2].

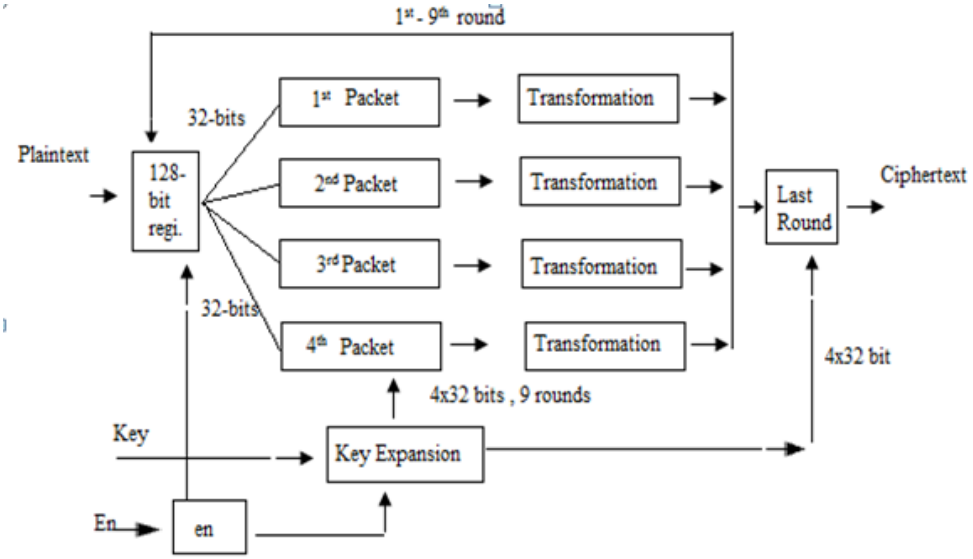


Figure.3 The data flow diagram of Advanced Encryption Standard (AES) implementation. [4]

In 2003, Pawel Chodowiec and Kris Gaj released a study that introduces a concise FPGA architecture designed for the AES algorithm with a 128-bit key. The architecture is specifically optimised for low-cost embedded applications. The AES algorithm exhibits a significant level of parallelism, which can be leveraged for improved performance. The procedure can be implemented using 4 sub bytes, 1 mix column, and 4 add round keys instead of the original 16 sub bytes, 4 mix columns, and 16 add round keys. This latest round is referred to as the folded round. This study not only discusses the fact, but also proposes the implementation of the s-box, mix columns, and shift rows. Several approaches for the implementation have also been deliberated. For instance, the diagram below illustrates a comprehensive configuration of a folded round, excluding the shift row action, which has been directly incorporated into the circuit.

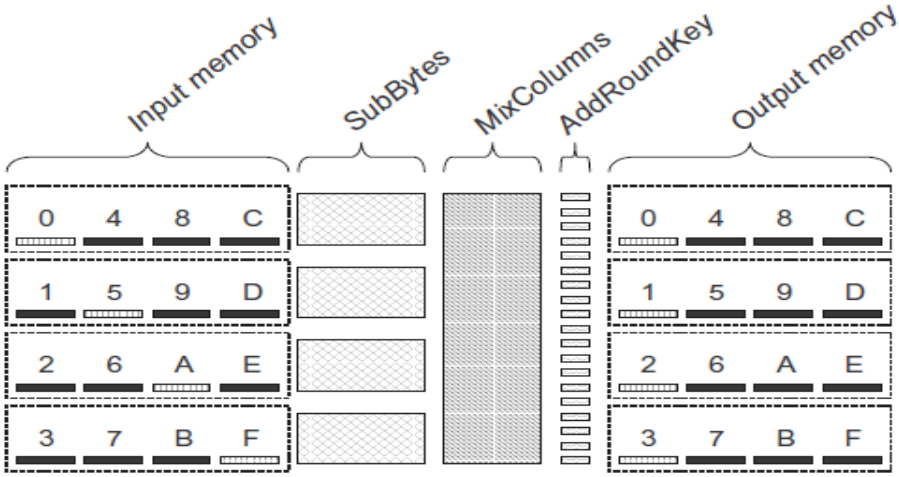


Figure.4 Arrangment exlude the shift row

This above mentioned structure can be implemented in dual port RAM ,here as explained above the exlusion is done with the shift row and inverse shift row. Two LUT's in the same slice can be configured as a 16x1 dual port ram .8-bit wide dual port ram can be implemented using the 8 consecutive CLB slices[3].

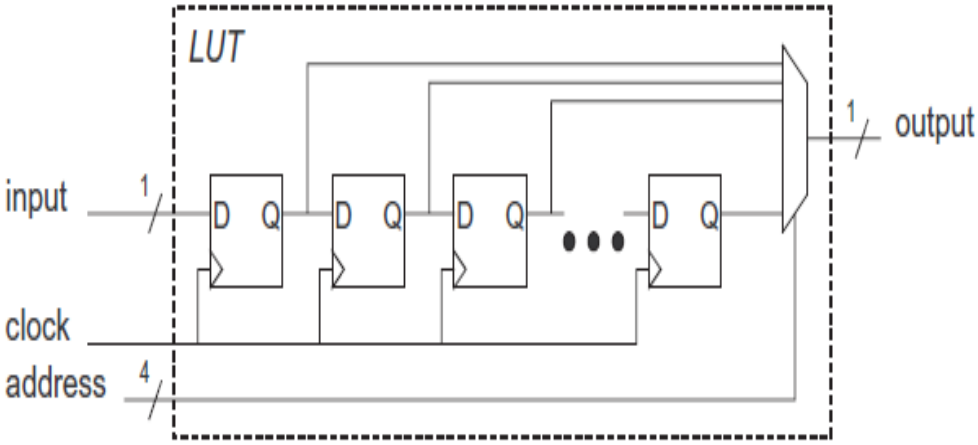


Figure.5 LUT configuration

Figure 5 illustrates the using the shift register have been discussed. The fact that, all bytes from the output of AddRoundKey are written into consecutive locations in the output memory in consecutive clock cycles. A shift register can be, taken considered & the fact that the CLB's can be configured as shift registers[3]. We know that, for mix columns and inverse mix columns,

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (1)$$

$$d(x) = c^{-1}(x) = \{0b\}\, x^3 + \{0d\}\, x^2 + \{09\}\, x + \{0e\} \quad (2)$$

It is evident from above two equations that,

$$c(x) \bullet d^2(x) = d(x) \quad (3)$$

The above discussion can be implemented and to compact the architecture. This shared logic

implementation can minimize the area and better utilization of the resources. It shows that same resources can be used for both mix-columns as well as for Inv-Mix-Columns.

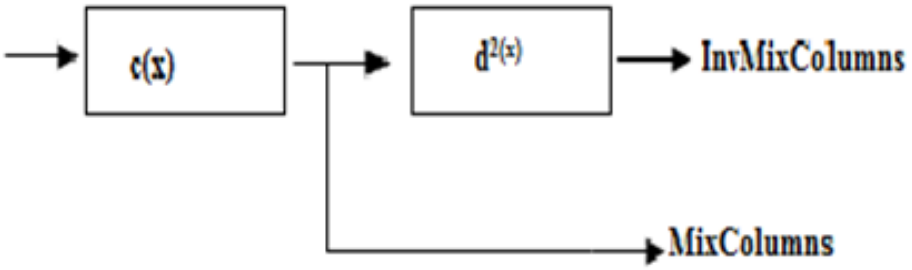
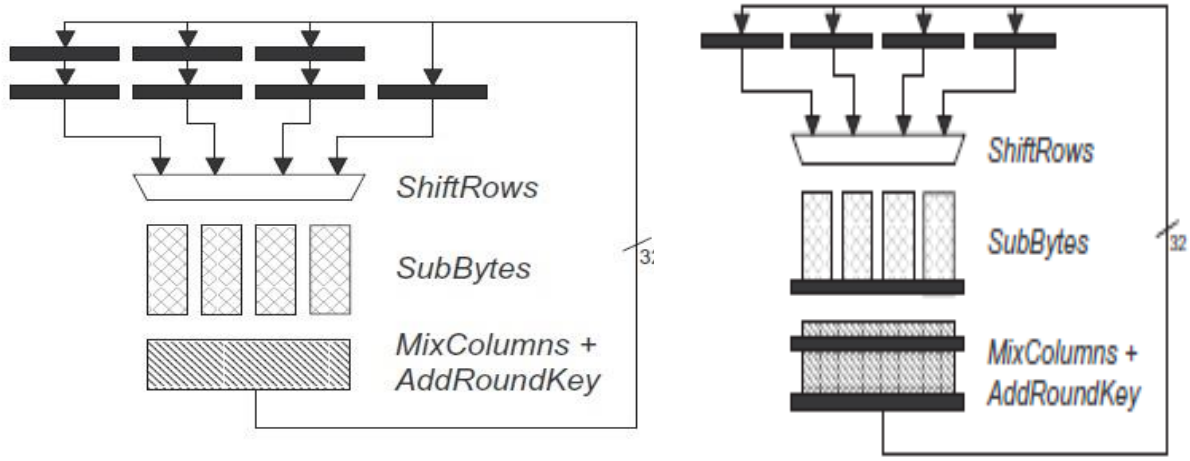


Figure.6 Implementation of Sharedlogic

Another simple method for the implementation is to employ folded architecture. The great deal of parallelism shown by the traditional AES implementation is a key for this method. This fact can be taken in considering the architecture of AES on VHDL language in a way FPGA implementation. Based on these two architectures can be derived and are also shown in below figure 7a.



(a) (b)

Figure.7 Folded architecture in two different flow

This architecture requires one 128-bit register, one 96-bit register and one 32-bit wide 4-to-1 multiplexer on top of the main cipher operations. The multiplexer becomes even bigger when both Shift Rows and Inv Shift Rows are implemented using same logic resources. The execution of one round takes four clock cycles. Their results show that the 4-cycle round takes 50% of the resources required by the 1-cycle round, and yields four times lower throughput. Another possible architecture is shown in figure 7 b The 96-bit register is implemented as three 32-bit registers inserted into round operations creating a pipeline. In the case of FPGAs, those 32-bit registers will most likely be placed in the same Slices as logic operations yielding better resource utilization. The critical path is also shortened which permits the execution at a higher clock rate; however, the execution of the entire round requires seven, instead of four, clock cycles [4].

For the low end applications such as PDA's , Chi-Jeng Chang, Chi-Wu Huang , Hung-Yun Tai, Mao-Yuan Lin' and Teng-Kuei Hu, proposed a paper of 8-bit data path in 2007. Initially for the better utilization of the resources and better throughput the pipelining had been implemented. But with the advent of the small scale instrument need arose for the compact AES

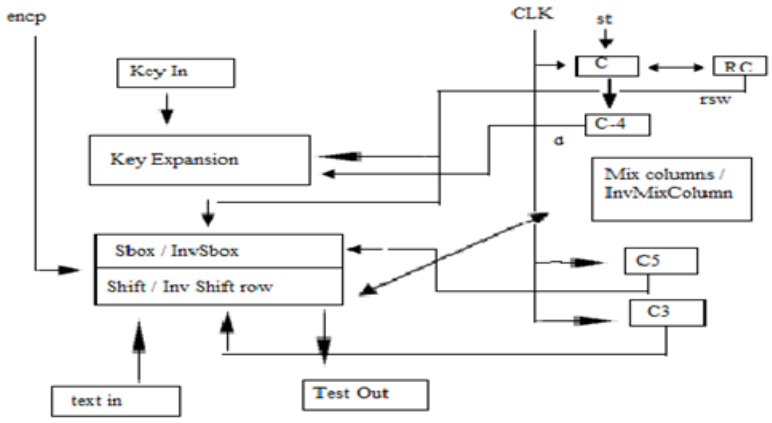


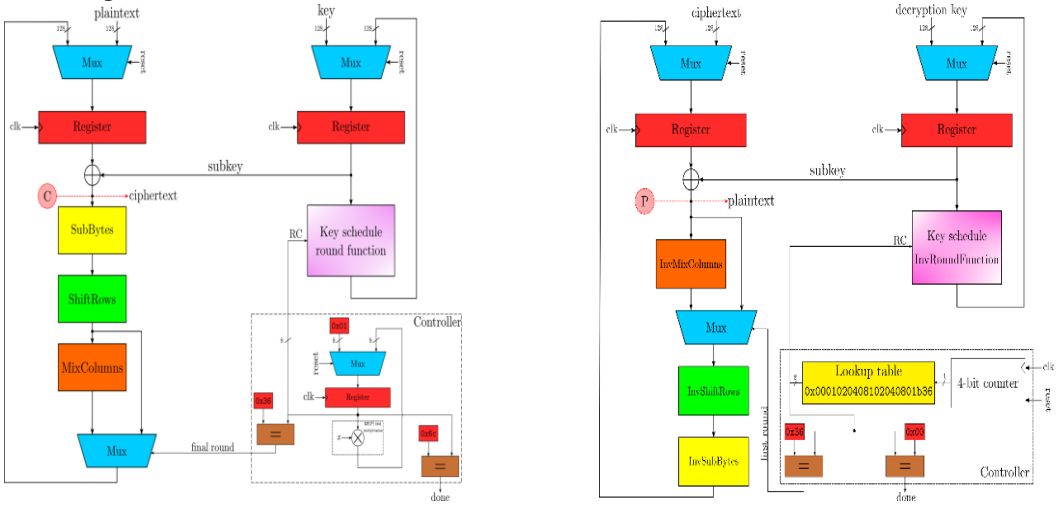
Figure.8 The data flow diagram of 8-bit Advanced Encryption Standard

Above shown idea multiplexes the 128-bit architecture to 8-bit data-path for saving resource area and uses BRAM to decrease the multiplexer/de-multiplexer circuit overhead of the Mix Column operation of AES. This direct 8-bit AES hardware implementation has moved S box, shift Row and most key Expansion circuit in BRAM, a total of 6 BRAM are needed. Two for shift Row (2x16x8) /Inv S box (2x256x8) in Mix Column one for S box in Key expansion (256x8) [4].

III. TEST SYSTEM

For our simulation results, Vivado 2015.4 software was utilized, a versatile tool allowing coding in both VHDL and Verilog. AES VHDL coding follows a mixed-mode modeling approach, incorporating component initialization. Initially, we generated the necessary rounds for AES, accommodating various key lengths: 128-bit, 192-bit, or 256-bit. An examination of the AES algorithm reveals that round keys can either be pre-generated or dynamically generated during runtime. In our implementation, we opted for on-the-fly key generation, along with the round transforms for each AES round. Furthermore, multidimensional arrays are employed for both AES plaintext and key generation to facilitate state generation.

The design of code is reconfigurable. Pipelining is a CPU optimization technique where the execution of instructions is overlapped, breaking down the instruction execution into several stages. This allows multiple instructions to be processed simultaneously, improving overall throughput and performance. Each stage in the pipeline handles a different part of the instruction execution process, such as fetching, decoding, executing, and storing results. Loop unrolling is a loop optimization technique aimed at reducing loop overhead by executing multiple iterations of a loop in parallel or consecutive sequence. Instead of executing the loop one iteration at a time, loop unrolling processes multiple iterations at once, reducing loop control overhead and improving instruction-level parallelism. The code employes pipelining concept as illustrated below. The concept of loop unrolling is also to further achieve the degree of optimization.

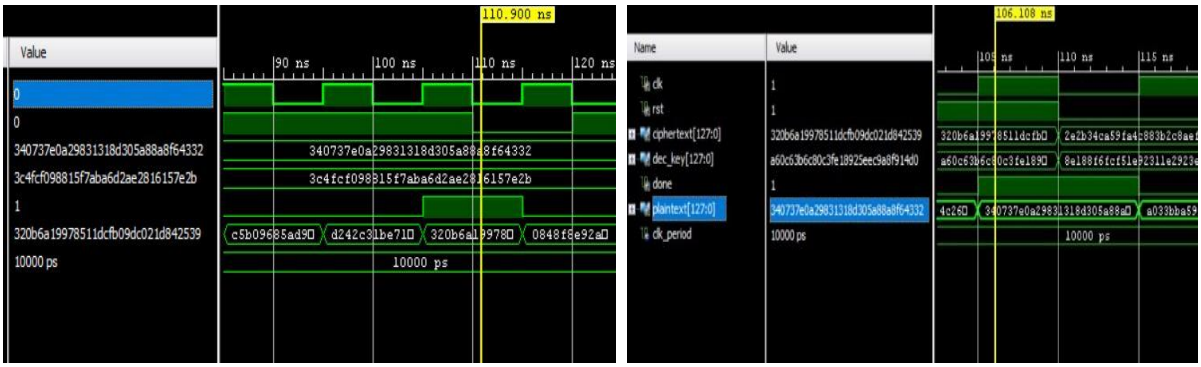


(a) encryption (b) Decryption

Figure .9 AES pipelined (a) encryption (b) Decryption

IV. Simulation Result Analysis

Simulation for same is done and studied. Study shows resource utilization for each stage and different transforms. For AES code the generated synthesis report shows how much resource utilization is of various stages as well as of various transforms. It followed from this that mix-columns and key expansion utilizes the higher resources compared to add round key. Much of the area utilization done by sub byte transforms. Any further efficient and compactness in this region will reduce the size of code considerably [14]. Below is a chart showing encryption decryption with know plaintext and key and utilization of resources stage wise.



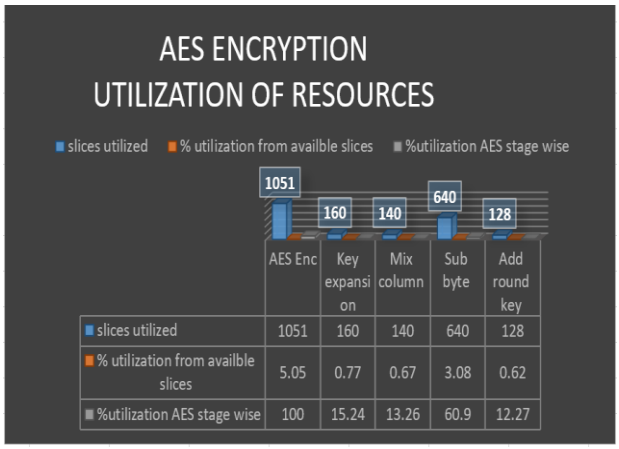
(a) encryption (b)Decryption Results

Figure.10 AES (a) encryption and b) Decryption simulation results

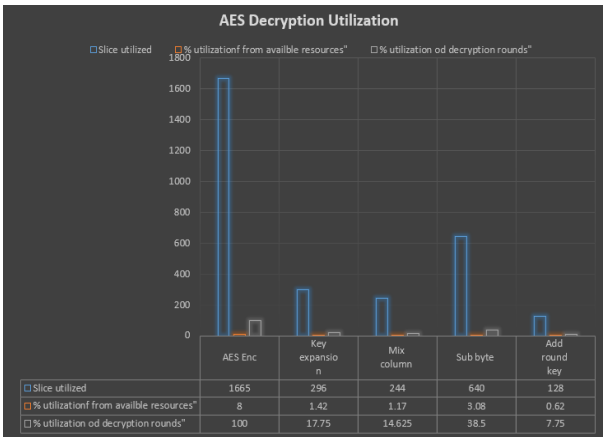
plaintext: X"2a179373117e3de9969f402ee2bec16b"

key: X"3c4fcf098815f7aba6d2ae2816157e2b"

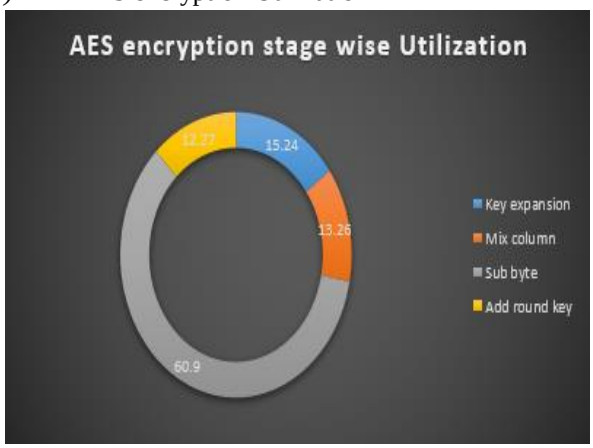
cipher : X"97ef6624f3ca9ea860367a0db47bd73a"



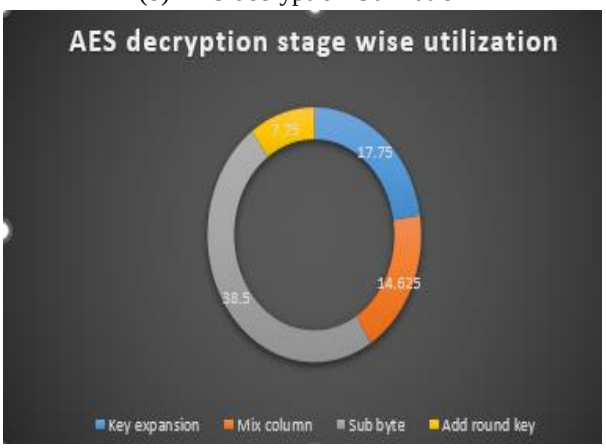
a) AES encryption Utilization



(b) AES decryption Utilization



(c) AES Encryption Stage wise utilization



(d)AES Encryption Stage wise utilization

Figure.11 Utilization Chart of AES

Figure 11 chart confirms our analysis that key expansion also utilizes the maximum resources. For the Decryption part we can use the same generated key from encryption part, instead of generating round keys from key input. Inv-sub bytes and sub-bytes are directly inputted as a ROM, so we only have to retrieve values from locations hence utilizes least resources. The Shift-row and Inv-Shift-row are only shifts in a position so not much utilization is there.

For the Galois field multiplication, we have used a method described federal information processing standards publications 197 announced in November 26, 2001. It states that multiplication in GF corresponds to with multiplication of polynomials modulo an irreducible polynomial of degree 8. For the AES algorithm, this irreducible polynomial is,

$$M(x) = x^8 + x^4 + x^3 + x + 1 \qquad (4)$$

Also a fact that multiplication by x (i.e.,{00000010} or {02}) can be implemented at the byte level as a left shift and a subsequent conditional bitwise XOR with {1b} is used. This operation on bytes is denoted by xtime (). So to multiply {57} · {13} can be implemented in parts as,

$$\{57\} \cdot \{02\} = \text{xtime}(\{57\}) = \{ae\}$$

$$\{57\} \cdot \{04\} = \text{xtime}(\{ae\}) = \{47\}$$

$$\{57\} \cdot \{08\} = \text{xtime}(\{47\}) = \{8e\}$$

$$\{57\} \cdot \{10\} = \text{xtime}(\{8e\}) = \{07\}$$

And thus,{57} · {13} = {57} · ({01} xor {02} xor {10})

While using this algorithm the number of xor operations will increase in mix columns states. To cope with it we may implement a Galois field multiplier. This multiplier can be implemented as per the flowchart shown below.

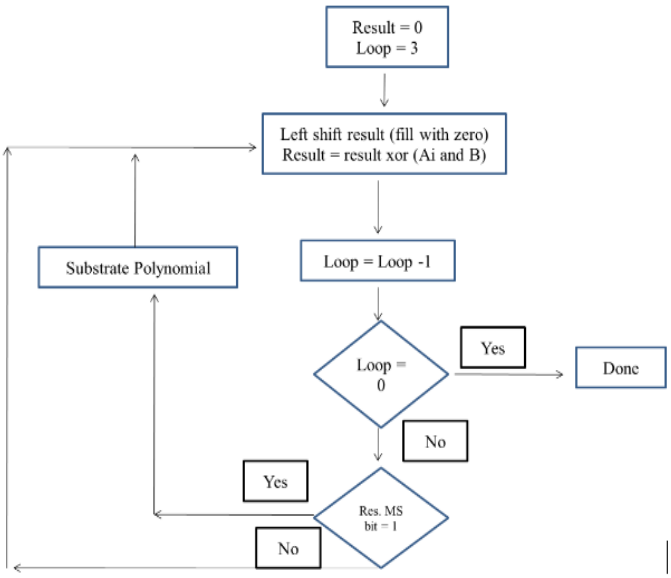


Figure. 12 flow chart for Galois Field multiplier

Here the loop is checked for 4 times for 4-bit multiplier. Where each MSB is checked, if found '1' xor-ing with irreducible polynomial is performed. To generate 8-bit multiplier we substitute 8 bits in entire flow diagram. So, instead 4 loops will now run for 8 loops [8]. It is evident from figure number 10 that key expansion utilizes the maximum resources. Studying key expansion scheduler reveals that, to generate a round key xor operations are performed. To generate 10 round keys out of 128 bit key we need to do 320 xor operations. If somehow usage of these many xoring can be minimized, better area and hence resource utilization can be achieved.

V. CONCLUSION

The structural style simplifies execution but consumes more resources. Optimizing the key expansion scheduler's code can significantly save resources. The shift row transform, involving only cyclic shifts, can be omitted if needed. The s-box is substitution of values but traditional 2-D S-box and the prior 1-D S-box utilizes more resources , a novel one-dimensional substitution Box (S-box) can be used to make code even more compact increasing the latency, throughput, transmission time.[15] This analysis reveals clear areas for improvement in the current approach.

REFERENCES

[1] Rady, Ahmed, Ehab EL Sehely, and AM EL Hennawy. "Design and implementation of area optimized AES algorithm on reconfigurable FPGA." In 2007 Internatonal Conference on Microelectronics, pp. 35-38. IEEE, 2007.

[2] Standaert, Francois-Xavier, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. "Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs." In Cryptographic Hardware and Embedded Systems-CHES 2003: 5th International Workshop, Cologne, Germany, September 8–10, 2003. Proceedings 5, pp. 334-350. Springer Berlin Heidelberg, 2003.

[3] Mulani, Altaf O., and Pradeep B. Mane. "High-Speed area-efficient implementation of AES algorithm on reconfigurable platform." Computer and Network Security 119 (2019).

[4] Wang, Zhu, Yan Yao, Xiaojun Tong, Qinghua Luo, and Xiangyu Chen. "Dynamically reconfigurable encryption and decryption system design for the internet of things information security." Sensors 19, no. 1 (2019): 143.

[5] Zhang, Xinmiao, and Keshab K. Parhi. "Implementation approaches for the advanced encryption standard algorithm." IEEE Circuits and systems Magazine 2, no. 4 (2002): 24-46.

[6] Gandh, D. Rahul, V. Kamalakannan, R. Balamurugan, and S. Tamilselvan. "FPGA implementation of enhanced key expansion algorithm for Advanced Encryption Standard." In 2014 International Conference on Contemporary Computing and Informatics (IC3I), pp. 409-413. IEEE, 2014.

[7] Alkamil, Arkan, and Darshika G. Perera. "Towards dynamic and partial reconfigurable hardware architectures for cryptographic algorithms on embedded devices." IEEE Access 8 (2020): 221720-221742.

[8] Granado-Criado, Jose M., Miguel A. Vega-Rodríguez, Juan M. Sánchez-Pérez, and Juan A. Gómez-Pulido. "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration." Integration 43, no. 1 (2010): 72-80.

[9] Masoumi, Massoud. "A highly efficient and secure hardware implementation of the advanced encryption standard." Journal of Information Security and Applications 48 (2019): 102371.

[10] Jamuna, S., P. Dinesha, K. PShashikala, and Kumar K. Kishore. "Design and Implementation of Runtime Reconfigurable Encryption Algorithms using Custom ICAP Processor." International Journal of Computer Network and Information Security 11, no. 12 (2019): 10-16.

[11] Chowdhury, A.R., Mahmud, J., Kamal, A.R.M. and Hamid, M.A., 2018, March. MAES: Modified advanced encryption standard for resource constraint environments. In 2018 IEEE Sensors Applications Symposium (SAS) (pp. 1-6). IEEE.

[12] Mane, P. B., and A. O. Mulani. "High speed area efficient FPGA implementation of AES algorithm." International Journal of Reconfigurable and Embedded Systems 7, no. 3 (2018): 157-165.

[13] Sireesha, K., and S. R. Madhava. "A novel approach of area optimized and pipelined FPGA implementation of AES encryption and decryption." International Journal Scientific and Research Publications 3, no. 9 (2013): 1-5.

[14] Fathy, Ahmed, Ibrahim F. Tarrad, Hesham FA Hamed, and Ali Ismail Awad. "Advanced encryption standard algorithm: Issues and implementation aspects." In Advanced Machine Learning Technologies and Applications: First International Conference, AMLTA 2012, Cairo, Egypt, December 8-10, 2012. Proceedings 1, pp. 516-523. Springer Berlin Heidelberg, 2012.

[15] Rajasekar, P., and H. Mangalam. "Design and implementation of power and area optimized AES architecture on FPGA for IoT application." Circuit World 47, no. 2 (2020): 153-163.

[16] Standaert, François-Xavier, Gilles Piret, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. "ICEBERG: An involutional cipher efficient for block encryption in reconfigurable hardware." In Fast Software Encryption: 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004. Revised Papers 11, pp. 279-298. Springer Berlin Heidelberg, 2004.