i-ETC: ISEL Academic Journal of Electronics,
Telecommunications and Computers
Vol. 8 , n. 1 (2022) ID-3

A New Lightweight and Efficient FPGA based Architecture for AES Algorithm Targeting IoT Security

Abiy Tadesse Abebe[a]
[a]School of Electrical and Computer Engineering, Addis Ababa Institute of Technology, Addis Ababa University, Addis Ababa, Ethiopia
abiy.tadesse@aait.edu.et

***Abstract* — Different platforms such as resource limited devices and high-performance processors are used in IoT networks each with its own set of resource, performance, and security needs. It is critical to optimize existing standard cryptographic algorithms to meet the needs of today's networks, yet this is a difficult undertaking. In this paper, a compact and efficient architecture for the Advanced Encryption Standard (AES) is developed and implemented using several FPGA devices, with the goal of addressing both constrained and high-performance platforms in IoT networks. To implement a compact and efficient FPGA-based AES architecture, a hybrid optimization technique is applied. The implementation takes advantage of FPGA embedded resources such as BRAMs and DSP slices. To synthesize and implement it on the Xilinx Virtex-7 device, the Vivado HLS tool 2019.2 is used. Similarly, as devices older than the Xilinx 7-series platforms are not directly supported by Vivado HLS tool, Xilinx 14.5 ISE tool is used to synthesize and implement it. Compared to existing research results found in the literature, reductions of 78.33 %, 63.12% and 78.14% of the number of slices utilized on Virtex-5, Virtex-6 and Virtex-7, respectively, are obtained. Also, improvements of 0.33%, 0.37% and 7.42% of throughput on Virtex-5, Virtex-6 and Virtex-7, respectively are achieved.**

Keywords: AES algorithm, cryptography, FPGA based implementation, IoT security, parallel pipelining architecture

I. INTRODUCTION

There are a number of well-known standard cryptographic algorithms with proven security capabilities, such as AES [1]. Although the benefits of standard algorithms have long been known, the contemporary network requires lightweight and efficient architectures since it includes constrained as well as high-performance platforms that differ in resource, performance and security needs. Optimizing existing standard cryptographic algorithms and improving their performance based on the needs of today's networks are on going research but are challenging.

After the Data Encryption Standard (DES) was broken due to its small key size, AES was developed as a standard symmetric key technique [1]. The AES algorithm is a strong symmetric key block cipher that has been used to secure a variety of applications.

Although it is not an authenticated encryption technique in and of itself, it serves as the foundation for several authenticated encryption algorithms [2], [3]. It is a well-organized standard symmetric key algorithm that can be implemented in both hardware and software.

Hardware implementation of AES provides stronger physical security and higher speed compared to the software implementation. For this reason, implementing AES in hardware is vital [4].

In this paper, a compact and efficient architecture for AES is developed and implemented using several FPGA platforms, with the goal of addressing the security of both constrained and high-performance platforms in IoT networks. To create compact and efficient FPGA-based architecture for AES, a hybrid optimization technique is applied. For the implementation, FPGA embedded resources such as BRAMs and DSP slices are combined with a reasonable number of typical FPGA logic elements.

The contribution of this research work is summarized as follows: considering the resource, performance and security requirements of the contemporary IoT network that incorporates high performance platforms and constrained devices, an FPGA-based AES architecture is proposed and implemented on different FPGA devices and optimized to achieve reduced area and improved throughput. The hardware-based small footprint architecture (in terms of small number of FPGA slices and embedded hard-cores) with a good throughput performance achieved using Virtex-5 device is intended for constrained devices' security in IoT application environment. Conversely, the Virtex 6 and Virtex 7 implementations of the proposed architecture with higher throughputs are intended for high performance platforms in current IoT networks.

The rest of the paper is organized as follows: Section II reviews the AES algorithm. FPGA based implementation approaches for AES are discussed in Section III. The proposed architecture is presented in Section IV. Section V presents the discussion and analysis of the achieved results in comparison to existing research results found in the literature. Finally, Section VI concludes the paper.

II. OVERVIEW OF AES ALGORITHM

AES is a 128-bit block cipher algorithm with three possible key sizes: 128-bit, 192-bit, and 256-bit, respectively, with 10, 12, and 14 round operations. As shown in Figure 1, the AES encryption process executes four essential operations to provide data confidentiality: SubBytes (byte substitution), ShiftRows (rows shifting), MixColumns (column mixing), and AddRoundKey (adding round keys). As illustrated in Figure 1, there is no MixColumns transformation for the last round.

The 128 bit block data (Plaintext) is placed in a 4x4 matrix known as state before the execution of these four basic operations can begin. State elements are elements with an 8-bit size that are found in each cell of a state. Before the round operations begin, the state is XORed with the initial round key.

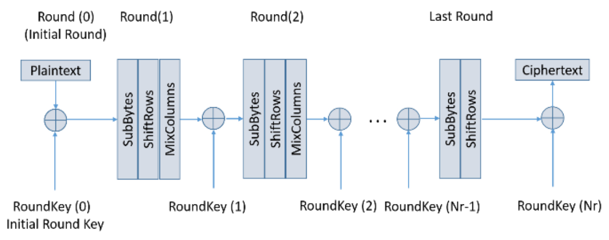


Fig. 1. The structure of AES Encryption Rounds

The first round key (128 bits) is combined with 128 bits plaintext to complete the initialization process. Then, until the desired round number is reached, all four operations are done in order, on the resulting state outputs (after each round step). The ciphertext is the end product of the encryption procedure. A separate key is utilized for each round operation, which is generated using a key generator function [1].

The SubBytes function is a nonlinear byte replacement that uses a substitution table (S-box) to act independently on each byte of the state. S-boxes are produced in two stages: Finding the multiplicative inverse in the GF(2^8) for the numbers 00H-FFH (Zero has no reciprocal, thus it is replaced by itself); then, performing the affine transformation to them. This entails multiplying a finite field by a matrix $\boldsymbol{A}$, then adding a finite field (Exclusive OR) to a vector $\boldsymbol{X}$ of hexadecimal value 63 as: [$\boldsymbol{A}$] * b + $\boldsymbol{X}$, which is extended as indicated in Eq. (1). Thus, applying all 256 possible bytes to the matrix in Eq. (1) yields a lookup table that implements the SubBytes transformation.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad (1)$$

ShiftRows is a transformation that cyclically shifts rows to the left. In this scenario, the first row remains unchanged, but the second row shifts one byte, the third row shifts two bytes, and the fourth row shifts three bytes to the left [1]. The MixColumns operation is processed by multiplying each state column by a matrix of constant numbers to produce an updated column [1] as shown in Eq. (2) and Eq. (3) for encryption and decryption processes, respectively [1].

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

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \qquad (3)$$

The MixColumns transform is composed of multiplication and addition operations, with 16 multiplications and 12 additions. When the input is multiplied by one, it can be directly taken. AddRoundKey is a function that adds the round key word to each column of the state matrix. One column at a time is processed by AddRoundKey. To generate fresh state, the state column is XORed with the key generated by the key generator.

InverseSubBytes, InverseShiftRows, AddRoundKey (which is the inverse of itself), and InverseMixColumns are the inverse operations used in AES decryption process. The construction of AES is described in detail in [1] and [5].

III. FPGA BASED IMPLEMENTATION APPROACHES FOR AES

Different studies have mostly concentrated on the AES S-box for implementing the AES algorithm on FPGA [4], [6]. This is because the S-box is the only non-linear component of the algorithm that has a significant impact on its performance. The ShiftRows operation is a permutation of bytes that does not require any hardware. The MixColumns operation is a linear column mixing transformation. AddRoundKey is a 128-bit word that is XORed with another 128-bit word. However, the AES S-box is a nonlinear byte replacement with a block length of 128 bits and a key length of 128 bits, considering AES-128.

The substitution box is the most expensive portion of AES in terms of hardware resources, necessitating effective hardware optimization for efficient implementation [4]. As a result, new approaches of reconstructing it (the S-box) for high speed or compact area optimization targets have been presented in [4], [6], [7], without compromising the algorithm's basic purpose.

RAM-based (using pre-stored S-Box values) [8], composite field-based (using combinational logic circuits rather than pre-stored values) [9], and LUT-based (using FPGA logic elements) [8], [10], are some of the implementation options for AES on FPGA. To build substitution boxes, the RAM-based technique makes use of the block RAMs found inside FPGAs. SubBytes and InverseSubBytes are thus stored in BRAMs. This approach can save logic elements because current FPGAs include BRAMs [8], [10]. However, it necessitates a large area. The composite field technique is the computation of SubBytes to directly implement the multiplicative inverse and affine transforms [9]. Small-area architecture can be achieved using this method. However, it does require some transformation logic [4], [11]. The LUT-based implementation method employs generic FPGA logic elements [12]. The design might not be accommodated due to a lack of resources if it is sophisticated and requires more such logic parts and the FPGA does not give as much of these resources as needed.

The T-Box (T-table) [13] is another implementation alternative that can be used instead of S-box implementations. A T-Box is a lookup table technique that includes SubBytes, ShiftRows, and MixColumns in addition to SubBytes. A T-Box can be expressed algebraically as demonstrated in Eq. (5):

$$\begin{bmatrix} T_0 = S'_{0,c} \\ T_1 = S'_{1,c} \\ T_2 = S'_{2,c} \\ T_3 = S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} SubBytes(S_{0,c}) \\ SubBytes(S_{1,c+1}) \\ SubBytes(S_{2,c+2}) \\ SubBytes(S_{3,c+3}) \end{bmatrix} \quad (5)$$

A. Optimization Techniques for Small Area AES on FPGA

Area optimization refers to the use of a small amount of FPGA resources for compact implementation [6] results, especially for use in constrained devices. There are many methods of area optimization for FPGA-based AES implementations [4], in addition to the composite field method [9]. The looping (iterative) design is the most basic way to implement AES on FPGA since it takes numerous computing cycles (as it is basically an iterative algorithm). It allows the use of the same FPGA hardware resource resulting in a decreased area utilization while requiring more clock cycles [4]. An iterative design on reconfigurable hardware for AES implementation is described in [12]. For AES-128, this method used a single round and processed it ten times iteratively. As demonstrated in Eq. (6), the throughput may be calculated by multiplying the maximum frequency reached by the data block size (128 bits) and dividing the result by ten (number of rounds) [12].

$$\mathit{Throughput\ (Mbps)} = (F_{max}(MHz) \times 128)/10 \quad (6)$$

Therefore, looping architecture necessitates the repetition of the same operation for a large number of computation cycles. As shown in Figure 2 [4], this architecture employs a feedback loop in which the data is iteratively modified by round functions.

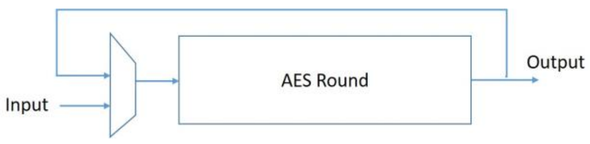


Fig. 2. Iterative (Looping) Architecture

Resource sharing is another way for small-area optimization. This optimization technique allows different functions and operations of the same algorithm to handle equivalent jobs on the same hardware. This method aids in the reduction of hardware requirements for various components of the algorithm that would otherwise necessitate independent hardware for each [4], [6].

In general, the applicability of the algorithm for area optimization and the coding styles associated to the structure of the FPGA device might influence optimization techniques for reduced resource consumption.

B. Optimization Techniques for High-Speed AES

Using pipelined architectures, it is possible to boost the throughput of the AES architecture at the cost of additional area. As shown in Figure 3, registers are inserted at each cycle of AES to form the pipeline for concurrent processing [4], [14]. The depth of the pipeline can be determined limiting the amount of data blocks that can be processed at the same time. If full pipeline is required, the total number of rounds of the AES is chosen as the pipeline depth to obtain higher throughput [4], [9], [15]. Pipeline architecture improves the performance of the encryption process as numerous blocks of data are executed at the same time.

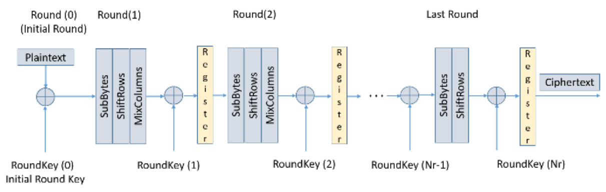


Fig. 3. Pipelined AES

Sub-pipeline architecture is created by inserting registers within the round functions of AES, as shown in Figure 4 [4], [9], [10]. Registers are also inserted within each round unit in this situation. If each round unit has x stages with equal delay, an N-round sub-pipelined design can reach x times the speed of an N-round pipelined architecture, with some additional registers and control logic causing some area increase [4].

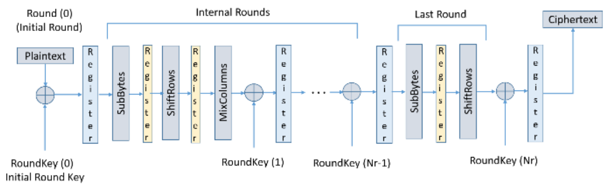


Fig. 4. Sub-pipeline AES

All rounds are implemented independently in hardware in a loop unrolling architecture [4], [16], as illustrated in Figure 5. In this case, the registers placed at each round in the AES pipelined architecture are eliminated, and multiple AES rounds are processed in the same clock cycle. Each round has the same delay, which is determined by the combinational logic employed. However, the aggregate of many delays produces the total delay, which makes these systems slow. Unrolled architectures can increase hardware complexity by allowing a large number of rounds to be implemented independently in hardware [4].

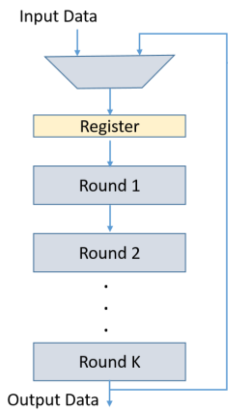


Fig. 5. Loop Unrolling Architecture

C. Area Versus Speed Trade-offs

In the design and implementation of FPGA-based AES, there is always a trade-off between throughput and hardware resource utilization. Designing a better architecture for a small-area and high-throughput application is demanding and challenging. In general, some applications require extremely high throughput, such as e-commerce servers, a wide range of throughput, such as cell phone design, and very tiny area and low power implementations, such as RFID cards [4]. Despite the fact that many studies have provided diverse implementation strategies for FPGA-based AES, efficiency could be significantly improved by using efficient architectures and optimization techniques linked to devices and algorithms [4].

When AES needs to be smaller in size while still performing at a high speed, balancing area and timing efficiency is essential [12], [13], [17]-[19]. However, the area and throughput trade-offs are dependent on the application's specific requirements [17]-[19].

IV. THE PROPOSED ARCHITECTURE

Using a hybrid approach, an efficient FPGA-based AES architecture is proposed as shown in Figure 6. Figure 6 depicts the proposed architecture, which contains two AES-128 cores running in parallel with full outer pipelining stages of rounds, including the initial round, intermediate rounds, and final round. It also depicts inner partial sub-pipelining and round-by-round processes. Except for the last round, which eliminates the MixColumn operation, a round consists of SubBytes, ShiftRows, MixColumns, and AddRound-Key operations. SubBytes and ShiftRows actions are made to function in tandem with MixColumns and AddRoundKey operations, as illustrated in Figure 6. These round activities are also partially pipelined.

To provide high throughput, the two AES-128 cores are designed to run in parallel with their respective full outer pipelining and parallel sub-pipelining modes. A pre-stored S-box BRAM and a hardware key scheduling module are also included in the proposed design for data access and key expansion duties, respectively. The plaintext (P_t) is provided in parallel mode using pre-stored BRAMs, as shown in Figure 6. The two concurrent AES-128 cores each accept 128-bit input data and process it in parallel.

In both cores, the initial round is first conducted by XORing the plaintext (P_t) with the initial round key in parallel. The intermediate rounds are then processed in the same way, with each round's processed state output XORed with the AddRoundKey of its associated round in both AES-cores in parallel. After the intermediate rounds are finished, the final rounds are run in parallel on both AES-cores, yielding the ciphertext (C_t). For both AES cores, the same round keys are utilized. To speed up the procedure, pipelined registers are employed. The key schedule generates the appropriate keys and stores them in BRAMs, which are then utilized by the two AES-cores during encryption. All of these procedures continue until all of the input messages have been processed.

To balance hardware resource use and improve throughput efficiency, this design uses both the FPGA general fabrics and specialized hard-cores such as DSP48E1 and BRAMs. The activities of these components and the parallel operations of the modules are synchronized using a control block. The proposed method varies from other analogous designs in that it combines several methodologies to provide a compact and efficient AES-128 architecture based on a hybrid approach, which is implemented utilizing DSP and BRAM hard-cores, as well as balanced resources from generic FPGA fabrics. Rather than employing all traditional logic parts or all FPGA hard-cores entirely, it balances the usage of both hardware resources. This enhances implementation flexibility while utilizing the modern FPGA resources for appropriate sections of the algorithm. The proposed architecture's combined impact is intended to generate a balanced trade-off between throughput and area, with more emphasis on throughput.

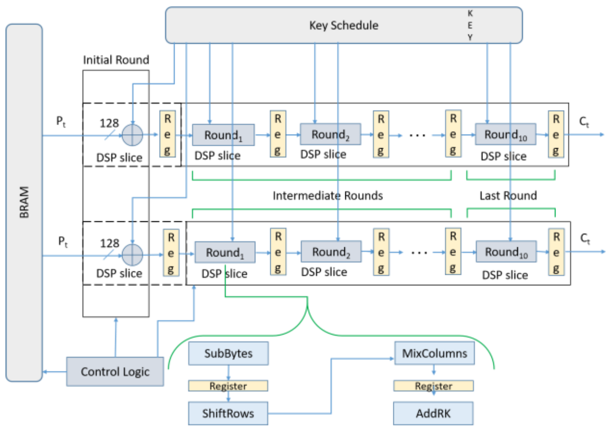


Fig. 6. A two core parallel pipelined AES architecture

Techniques from [4], [16], and [20] were employed and merged in the proposed method. In [4], generic FPGA fabrics were used to build fully pipelined AES. They didn't employ parallel AES cores or DSP hard-cores. DSP slices and BRAMs were employed in [16] to construct a high-throughput round-based and unrolled pipelined architecture. They didn't employ parallel AES-128 cores in their method. In [19], a similar technique to [16] was adopted for implementing AES and extending the method to include GCM and optimize AES-GCM utilizing DSP and BRAM. They followed the architecture used in [16], which included round-based and unrolled architectures. Unlike the method proposed in [20], where four parallel AES-cores were employed for AES-GCM implementation, only two parallel AES-128 cores are used based on hybrid optimization technique in the present proposed architecture.

The area needed by two parallel AES-128 cores is half that of four parallel AES-cores. The usage of DSP and BRAM hard-cores proposed by [16] for implementation of unrolled pipelined architecture is followed; however, it is only utilized to create parallel AES-128 cores in the current study, not for a single round based and unrolled pipelined AES architecture. The proposed work uses the same fully pipelining approach as [4] to fully pipeline the two parallel AES-128 cores; however, this time it is also based on DSP slices and BRAMs, rather than using only generic FPGA logic elements as in [4]. In addition, the partial sub-pipelining strategy uses simultaneous SubBytes and ShiftRows operations with MixColumns and AddRoundKeys operations.

The key schedule shown in Figure 7 is implemented in hardware, but the required round constants shown in Table 1 are stored in BRAMs. The process of creating all round keys from the original input key is known as key-expansion in AES-128. The key-expansion method develops *Nr* + 1, 128-bit round keys from a single 128-bit key if the number of rounds is given by *Nr*. Before beginning the encryption or decryption operations, an initial round key is added to the input. The circular keys are made in a word-by-word fashion. For the 128-bit key size, ten round keys of 16 bytes are created. SubWord applies the S-box to the 32-bit input word, RotWord rotates the word one byte to the left, and the round constant RC_i is an 8-bit constant associated with each round, as illustrated in Figure 7.

TABLE I
AES ROUND CONSTANTS

Round	Constant
1	(**01** 00 00 00)$_{16}$
2	(**02** 00 00 00)$_{16}$
3	(**04** 00 00 00)$_{16}$
4	(**08** 00 00 00)$_{16}$
5	(**10** 00 00 00)$_{16}$
6	(**20** 00 00 00)$_{16}$
7	(**40** 00 00 00)$_{16}$
8	(**80** 00 00 00)$_{16}$
9	(**1B** 00 00 00)$_{16}$
10	(**36** 00 00 00)$_{16}$

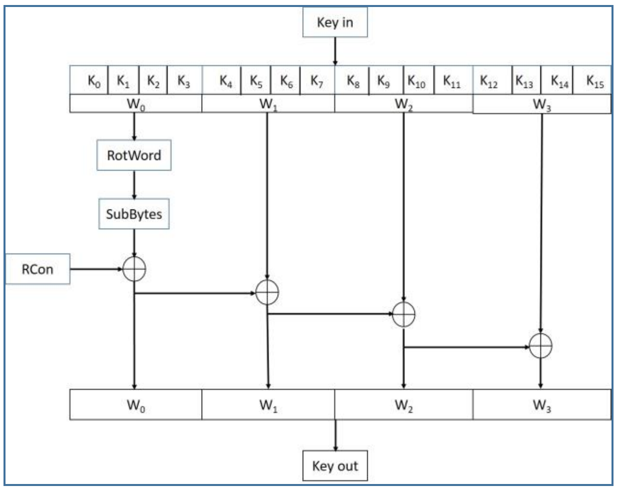
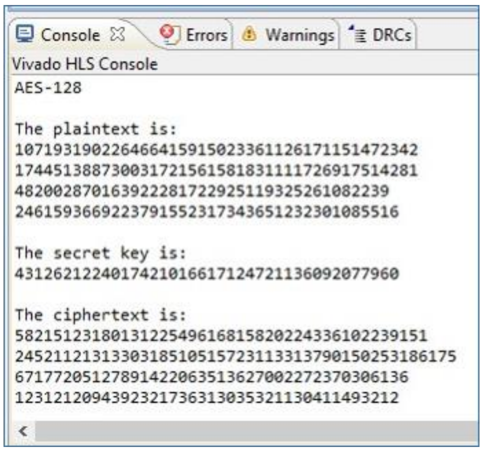


Fig. 7. Key expansion in AES

V. IMPLEMENTATION APPROACHES

The proposed architecture was first implemented on Xilinx Virtex-7 (Part: XC7VX690T, speed-grade -3) platform. It was tested using the Xilinx Vivado 2019.2 High-Level Synthesis (HLS) tool. A functional test for the encryption section of AES-128 is shown in Figure 8. The RTL output was then synthesized on Xilinx Virtex-5 (XC5VLX155T, speed-grade -3) and Virtex-6 (XC6VLX75T, speed-grade -3) FPGAs.



```
AES-128

The plaintext is:
107193190226466415915023361126171151472342
174451388730031721561581831111726917514281
482002870163922281722925119325261082239
2461593669223791552317343651232301085516

The secret key is:
4312621224017421016617124721136092077960

The ciphertext is:
5821512318013122549616815820224336102239151
2452112131330318510515723113313790150253186175
67177205127891422063513627002272370306136
1231212094392321736313035321130411493212
```

Fig. 8. Functional test for AES-128 Encryption

Xilinx Vivado HLS is a cutting-edge EDA tool that allows to specify a design in software, synthesize it, and generate RTL for the specified design. It gives the freedom to optimize the design implementation using a variety of optimization directives to get the results needed. The synthesized VHDL code was implemented on Xilinx Virtex-7 device. Xilinx ISE 14.5 was used to implement and analyse the performance of the proposed architecture on earlier generations of Xilinx FPGA devices that are not directly supported by the Vivado HLS tool, including Virtex-5 and Virtex-6. A simple Finite State Machine (FSM) was used for synchronization of the activities of the different components of the proposed architecture including the pipeline registers found in DSP slices. Exclusive-or (XOR) operations are employed to accomplish binary addition operations in hardware, hence DSP slices were used to conduct the XOR operations. BRAMs were used to hold the S-box values and constants. As a result, the BRAMs are read whenever the data and keys for processing the AES states are required. To execute 128 bit operations, cascaded DSP slices were employed.

VI. RESULTS COMPARISONS AND ANALYSIS

Xilinx Virtex-5, Virtex-6, and Virtex-7 FPGA platforms were used to synthesize the proposed architecture. Table II, Table III and Table IV show comparisons of the synthesis results of proposed architecture on Virtex-5, Virtex-6 and Virtex-7 devices, respectively, including resource utilization and throughput performance, against existing research findings found in the literature. It's difficult to obtain a balance between throughput performance and resource consumption as there is always a tradeoff between them. Nonetheless, the goal of this research is to achieve better throughput while using relatively less hardware resources.

In comparison to existing research outcomes found in the open literature, a lesser number of LUT slices are utilized for synthesizing the proposed AES architecture on Xilinx Virtex-5, Virtex-6, and Virtex-7 FPGA devices, while achieving improved throughput performance. Thus, reductions of 78.33 %, 63.12% and 78.14% of the number of slices utilized on Virtex-5, Virtex-6 and Virtex-7, respectively, are obtained. Similarly, improvements of 0.33%, 0.37% and 7.42% of throughput on Virtex-5, Virtex-6 and Virtex-7, respectively, are achieved compared to the outcomes found in the open literature for the related works. These results are achieved at the cost of smaller number of FPGA hard-cores.

From the results shown in Table II, Table III and Table IV, it is noted that the achieved frequencies of the present work are higher. This is because the reported results in this paper are Post-synthesis Timing analysis and not post place-and-rout (PAR) simulation analysis that considers full design routing. Moreover, only the AES encryption part is considered for the Post-synthesis timing analysis.

VII. CONCLUSIONS

It's critical to improve the performance of existing standard cryptographic algorithms to meet today's security standards. Despite the fact that the benefits of such standard algorithms have long been recognized, the contemporary network's resource, performance, and security needs demand lightweight and efficient designs since it includes constrained and high-performance platforms. An efficient architecture for FPGA-based AES algorithm is proposed which takes into account the IoT security. A hybrid technique is employed, and the proposed AES architecture is implemented on Xilinx Virtex-5, Virtex-6, and Virtex-7 FPGA devices. In comparison to existing research outcomes found in the open literature, lower number of LUT slices are utilized for implementation of the proposed AES architecture on Xilinx Virtex-5, Virtex-6, and Virtex-7 FPGA devices, while achieving improved throughput at the cost of smaller number of FPGA hard-cores. As a result, 78% reduction of the number of slices on Virtex-5 device and 7.42% improvement of throughput on Virtex-7 device are achieved. In the future, Post-implementation Timing analysis will be performed considering full routing of the design.

TABLE II
RESULTS FOR IMPLEMENTATION OF THE PROPOSED AES ARCHITECTURE ON VIRTEX-5 DEVICE

Methods	Device	Slices	BRAMs	DSPs	Freq. (MHz)	Throughput (Gbps)
[21]	Virtex-5	3420	-	-	199.18	25.50
	Virtex-5	3788	-	-	232.30	29.73
[22]	Virtex-5	7385	-	-	638.16	81.68
[23]	Virtex-5	2940	-	-	704.70	90.40
This work	Virtex-5	637	8	32	708.60	90.70

TABLE III
RESULTS FOR IMPLEMENTATION OF THE PROPOSED AES ARCHITECTURE ON VIRTEX-6 DEVICE

Methods	Device	Slices	BRAMs	DSPs	Freq. (MHz)	Throughput (Gbps)
[21]	Virtex-6	4095	-	-	463.42	5.93
[17]	Virtex-6	423	-	-	191.98	2.50
[21]	Virtex-6	5566	-	-	237.45	3.03
	Virtex-6	4095	-	-	463.42	59.31
[22]	Virtex-6	6361	-	-	849.18	108.69
[23]	Virtex-6	2537	-	-	740.70	94.81
[17]	Virtex-6	5759	-	-	732.28	93.73
	Virtex-6	9531	-	-	457.58	58.57
	Virtex-6	5759	-	-	849.18	108.69
[24]	Virtex-6	1626	-	-	166.66	0.24
[6]	Virtex-6	1551	-	-	190.66	0.56
This work	Virtex-6	572	8	32	828.60	109.0

TABLE IV
RESULTS FOR IMPLEMENTATION OF THE PROPOSED AES ARCHITECTURE ON VIRTEX-7 DEVICE

Methods	Device	Slices	BRAMs	DSPs	Freq. (MHz)	Throughput (Gbps)
[21]	Virtex-7	4089	-	-	495.32	6.34
[7]	Virtex-7	126040	288	-	-	31.29
[25]	Kintex-7	4493	-	-	202.26	21.92
[23]	Virtex-7	2617	-	-	813.0	104.06
This work	Virtex-7	572	8	32	878.0	112.40

REFERENCES

[1] J. Daemen and V. Rijmen, The Design of Rijndael: AES - The Advanced Encryption Standard, In Information Security and Cryptography. springer, 2002.

[2] D. McGrew, J. Viega, The Galois/Counter Mode of operation (GCM), Submission to NIST, May 2005.

[3] H. Wu and B. Preneel, AEGIS: A Fast Authenticated Encryption Algorithm (v1. 1), Submission to CAESAR, 2016.

[4] A. Tadesse and P.S. Kumar, Effective Implementations Techniques for FPGA Based AES Algorithm, 2016 KICS Korea and Ethiopia ICT International Conference, 2016.

[5] PUB, NIST FIPS. 197. Specification for the advanced encryption standard (AES), 2001-11-26). ht-tp://csrc. nist. gov/publications/fips/fips197/fips-197. pdf 2001.

[6] P. Rajasekar and H. Mangalam, Design and implementation of power and area optimized AES architecture on FPGA for IoT application, Circuit World, 2020.

[7] S. Chen, W. Hu, Z. Li, High Performance Data Encryption with AES Implementation on FPGA, In 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE

Intl Conference on Intelligent Data and Security (IDS), 149-153. IEEE, 2019.

[8] C.J. Chang, C.W. Huang, H.Y. Tai, M.Y. Lin, and T.K. Hu, 8-bit AES FPGA implementation using block RAM, In IECON 2007-33rd Annual Conference of the IEEE Industrial Electronics Society, 2654-2659. IEEE, 2007.

[9] X. Zhang and K. K. Parhi. High-Speed VLSI Architectures for the AES Algorithm, IEEE transactions on Very Large Scale Integration (VLSI) Systems, 12(9):957-967, 2004.

[10] F. Wu, L. Wang, J. Wan, A low cost and inner-round pipelined design of ECB-AES-256 crypto engine for Solid State Disk, In 2010 Fifth IEEE International Conference on Networking, Architecture, and Storage, IEEE, 485-491, 2010.

[11] C. Arul Murugan, P. Karthigaikumar and Sridevi Sathya Priya, FPGA implementation of hardware architecture with AES encryptor using sub-pipelined S-box techniques for compact applications, Automatika, 61(4), 682-693, 2020.

[12] F. X. Standaert, G. Rouvroy, J. J. Quisquater, and J. D. Legat, Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs, In International Workshop on Cryptographic Hardware and Embedded Systems-CHES, 334-350, Springer, Berlin, Heidelberg, 2003.

[13] V. Fischer, and M. Drutarovský, Two methods of Rijndael implementation in reconfigurable hardware, In Proceedings of the Conference on Cryptographic Hardware and Embedded Systems (CHES'01). 2160, 77-92, Springer, Berlin, Heidelberg, 2001.

[14] C. P. Fan and J. K. Hwang, Implementations of high throughput sequential and fully pipelined AES processors on FPGA, In 2007 International Symposium on Intelligent Signal Processing and Communication Systems, 353-356, IEEE, November, 2007.

[15] A. Hodjat and I. Verbauwhede, A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA, In 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 308-309, IEEE, 2004.

[16] S. Drimer, T. Güneysu, and C. Paar, C., DSPs, BRAMs, and a pinch of logic: Extended recipes for AES on FPGAs, ACM Transactions on Reconfigurable Technology and Systems (TRETS), ACM, 2010.

[17] A.Q. Al-Khafaji, M.F. Al-Gailani and H.N. Abdullah, FPGA Design and Implementation of an AES Algorithm based on Iterative Looping Architecture, In 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin), 1-5. IEEE, September 2019.

[18] A. Soltani, S. Sharifian, An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA, Microprocess. Microsyst. 39(7), 480–493, 2015.

[19] E.B. Kavun, N. Mentens, J. Vliegen and T. Yalçın, Efficient Utilization of DSPs and BRAMs Revisited: New AES-GCM Recipes on FPGAs, In 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 1-2. IEEE, December 2019.

[20] L. Henzen and W. Fichtner, FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications, In 2010 Proceedings of ESSCIRC, IEEE, 202-205, 2010.

[21] H. Zodpe, and A. Sapkal, An efficient AES implementation using FPGA with enhanced security features,Journal of King Saud University-Engineering Sciences, 32(2) 115-122, 2018.

[22] S. Oukili, S. Bri, High throughput FPGA implementation of Advanced Encryption Standard algorithm, Telkomnika. 15(1), 494–503, 2017.

[23] M. B. Chellam, R. Natarajan, AES Hardware Accelerator on FPGA with Improved Throughput and Resource Efficiency, Arabian Journal for Science and Engineering, 43(12), 6873-6890, 2018.

[24] S. Oukili, S. Bri, Hardware implementation of AES algorithm with logic Sbox, Journal of Circuits, Systems and Computers. 26(9), 1750141, 2017.

[25] S.S. Rekha and P. Saravanan, Low-Cost AES-128 implementation for edge devices in IoT applications, Journal of Circuits, Systems and Computers, 28(4) 1950062, 2019.

[26] Shengiian, L., Ximing, Y., Senzhan, J. and Yu, P., A fast hybrid data encryption for FPGA based edge computing, In 2019 14th IEEE International Conference on Electronic Measurement & Instruments (ICEMI) 1820-1827. IEEE, 2019.