Effective Pipelined FPGA Implementation for AES-256 Algorithm

Mona Sayed Abdul-Karim[1], Kamel Hussien Rahouma[2], and Khalid Nasr[3]

[1, 2, 3] Communications and Electronics Section, Elec. Eng. Dept., Faculty of Engineering, Minia University, Egypt.
[2] Faculty of Computer Science, Nahdah University in BeniSuef, BeniSuef, Egypt
[3] CS Department, College of Computer Science and Engineering, Taibah University, Saudi Arabia.
[1]mona.sayed.abdelkarim@mu.edu.eg,[2] kamel_rahouma@yahoo.com,[3] knasr@taibahu.edu.sa

Abstract

Advanced Encryption Standard (AES) is one of the most common and secured cryptographic techniques. In recent years, AES has received significant attention from scientists owing to its wide spectrum of applications such as communication, network, military, electronic banking, Internet of Things (IoT), etc. AES implementation can be executed using software and hardware tools. Using hardware tools a higher data rate can be accomplished compared to software ones. Field Programmable Gate Array (FPGA) is considered one of the most common and efficient tools for hardware implementation. In this paper, we have developed an FPGA implementation for AES-256, which is considered the most secure one among AES categories. We applied fully pipelining, sub-pipelining, loop-unrolling techniques and other effective solutions for the most complex operations of AES-256 such as Mix-columns, and Sub-byte transformation. Our AES-256 implementation is carried out using Virtex-7(XC7VX485T-FFG1157-1) FPGA and accomplishes a maximum operating frequency of **345.095 MHZ** and throughput of **44.17 Gbps**.

Keywords: *AES-256, high-throughput, fully pipelining, sub-pipelining, FPGA.*

1. Introduction

Cryptography is a significant component for secure communication and transmission of information because it provides security services like confidentiality, data integrity, authentication access control and non-repudiation [1]. It converts sensitive information to unreadable format and only the authorized parties have the ability to access this information by converting it back into the original text [2]. Cryptography can be categorized into three main types; Asymmetric and symmetric key systems [3], [4]. In symmetric systems, the encryption and decryption processes use the same private keys [5]. In asymmetric systems, encryption and decryption use different keys which are related by a certain function. AES is a familiar popular symmetric key cryptographic algorithm [6].

Recently, AES has received significant attention from researchers owing to its broad range of applications in communication, military, network, electronic banking, IoT, etc [7]. AES can be implemented in either software or hardware structure. Hardware structure can be implemented using reconfigurable devices such as FPGAs which give high-performance requirements. Hardware implementation can use the loop-unrolling [7], [8] and partial rolling [9] techniques to increase the throughput to area ratio and decrease the area cost. Furthermore, to increase running frequency and throughput, pipelining and sub-pipelining techniques can be applied. This paper aims to introduce an FPGA implementation of AES-256. To do that, we apply the loop-unrolling, fully pipelining, and sub-pipelining techniques. Moreover, other efficient methods are utilized for the most complex parts of AES-256 such as Mix-columns, and S-boxes. Such implementation is highly fast and thus it can be utilized in high-speed networks. The rest of this paper is arranged as follows: **Section 2** presents an overview of the AES algorithm. **Section 3** presents the proposed high throughput implementation for the AES-256 algorithm. **Section 4** presents results and a comparison with previous work. **Section 5** gives a conclusion.

2. Overview of Advanced Encryption Standard

AES was released by the National Institute of Standards and Technology (NIST) in 2002 [8] to replace the old Data Encryption Standard (DES) and 3DES as the approved and strongest standard for a broad range of applications [10]. AES is not built on Feistel Structure such as DES and 3DES therefore it can process the full block of data at once in a single array during each round [11]. AES is a symmetric block cipher that takes two inputs, the plaintext message and the key and produces a ciphertext message as shown in **Figure 1**. In AES as shown in **Figure 1**, the plaintext message is segmented into blocks each of 16 bytes (128 bits) in length. The length of the key is taken as 16, 24, or 32 bytes (128, 192, or 256 bits). Hence, the algorithm is called AES-128, AES-192, or AES-256, based on the key length [12][13]. AES encryption/decryption process consists of three main stages [14] [15]; adding the initial key in the stage of the Add-Round key, *n* rounds are based on the key size and key expansion unit. In the first part, the initial key is added to the plaintext. In the second part, each round includes four transformations except the final one has only three, called, substitute bytes (S-Box), Shift-Rows, Mix-Columns and Add-Round Key [7], [16]. Mix-Columns operation is removed from the final round. All AES operations are executed on 8-bits over the finite field $GF(2^8)$ by the following polynomial given in **Eq. 1** [17], [12]:

$$m(X) = X^8 + X^4 + X^3 + X + 1 \quad (1)$$

Each time, a single 128-bit block is processed. The block is organized as a 4*4 square array of bytes named the **State** array [12]. The **State** array is filled with the 128-bit block. The array is updated at each stage of AES. When the final stage is ended, the **State** array is copied to an output one [12], [14]. In the following sub-sections, we will give an overview of the transformations of each round.

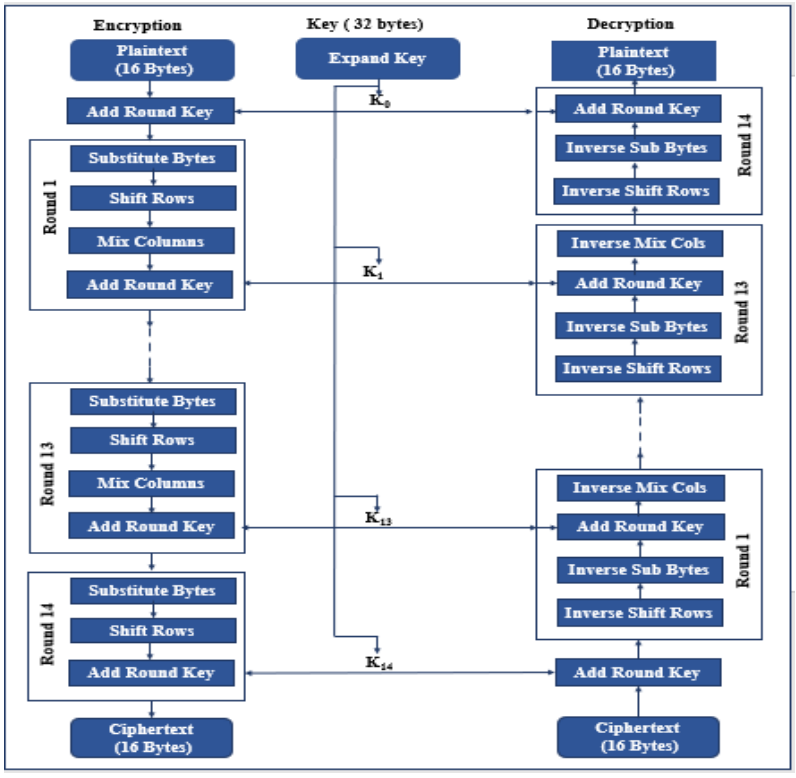

Figure 1. The structure of AES-256.

2.1. Substitute Bytes Transformation (S-Box)

S-box is an invertible transformation that replaces independently each byte of the State array with its corresponding byte value using a fixed size look-up table (256 bytes) [16]. The S-box offers non-linearity and confusion depends on multiplicative inverse and affine transformation as shown in **Eq.2** and **3** [16], [18].

$$S(X) = Affine\ transform\ (X^{-1}) \quad (2)$$

$$Affine\ transform = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} i_7 \\ i_6 \\ i_5 \\ i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \quad (3)$$

Where $i = X^{-1}$.

2.2. Shift Rows Transformation

In this transformation, the rows of the State array are circularly left-shifted with different values as follows [12], [18]; The first row is not changed. The second row is circularly left-shifted by one byte. The third row is by two bytes the fourth row is by three bytes [15], [19].

2.3. Mix-Columns Transformation

Mix-columns operation is processed individually according to the elements of each column of the State array. Each byte of a column is interchanged by a new one [19], [20]. The new byte is a function of all four bytes of the same column. Each column is expressed by a four-term polynomial over $GF(2^8)$ [21]. The column is multiplied by a constant polynomial $a(x)$ specified in **Eq. 4** and then modulo $(X^4 + 1)$ is calculated. **Eq. 5** presents the calculations of the Mix-columns transformation in matrix multiplication [22], [23]. Where i is the old column and d is the new one.

$$a(x) = (03)x^3 + (01)x^2 + (01)x + (02) \quad (4)$$

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} i_0 \\ i_5 \\ i_{10} \\ i_{15} \end{bmatrix} \quad (5)$$

2.4. Add Round Key Transformation

The round key array is added to the State array over $GF(2^8)$. The addition in $GF(2^8)$ is executed using a bitwise Exclusive-OR (*XOR)* operation for each byte of the sub-key with the corresponding byte of the State array. For each round, the sub-key is generated from the main key using the key expansion unit[24], [25]. The same operation is executed in the decryption process.

2.5. Key expansion

Each round has its key which is generated from the master key (of length *Nk*) using key expansion. The AES algorithm carries out *Nr* rounds and each round needs an initial group of *Nb* words of the key. A total number of *Nb(Nr+ 1)* words are produced from the key expansion [26], [27]. The resultant key schedule consists of an array of 4-byte words, defined *w[i],* with i in the range $0 \leq t < Nb(Nr + 1)$ [21]. A pseudo-code representation [26] of the key expansion is given in **Table 1**. For AES-256, *Nk* is 8, *Nb* is 4, *Nr* is 14, ***SubWord()*** is a function that applies the S-box to each byte of the four bytes and produces an output word, ***RotWord()*** is a function that does a cyclic rotation by one byte, and ***Rcon[i/Nk]*** is the round constant and *i* is the word index.

3. Proposed High Throughput Implementation for AES-256

First, we choose the AES-256 since it is more secure than AES-128 and AES-192 because of its longer key size and more rounds. Hence, we tried to propose an efficient implementation for it. Our implementation of the AES-256 is performed using loop-unrolling techniques to eliminate all the required loops in the algorithm which results in changing the critical path. This modification in the critical path allows inserting some pipelining registers which in turn raises the operating frequency, speeds up the algorithm and enhances the quality of service for the applications of AES-256. We apply both sub and full-pipelining techniques. The pipelining registers (called Pip. Reg. in **Figure 2** and **Figure 3**) are added between the rounds in the whole algorithm and between the operations in each round. **Figure 2** shows the overall block diagram of loop-unrolled and pipelined AES-256. **Figure 3** shows the overall block diagram of the sub-pipelined round of AES-256. We also employ efficient methods for the most complex operations in AES-256, Mix-columns and S-box. They will be discussed in the following sub-sections.

Table 1. Pseudocode for key expansion.

Algorithm 1: Key expansion

```
KeyExpansion (byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
word temp
i = 0
while (i < Nk)
  w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
  i = i+1
end while
i = Nk
while (i < Nb * (Nr+1))
   temp = w[i-1]
   if (i mod Nk = 0)
     temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
 else if (Nk > 6 and i mod Nk = 4)
  temp = SubWord(temp)
  end if
  w[i] = w[i-Nk] xor temp
  i = i + 1
 end while
end
```

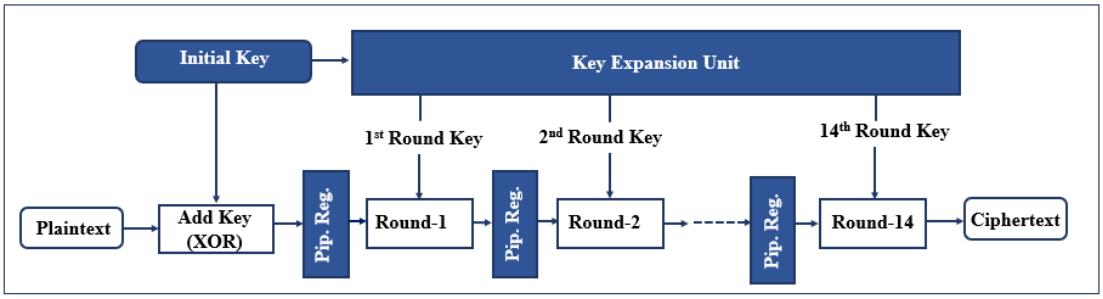

Figure 2. The block diagram of the loop-unrolled and pipelined AES-256

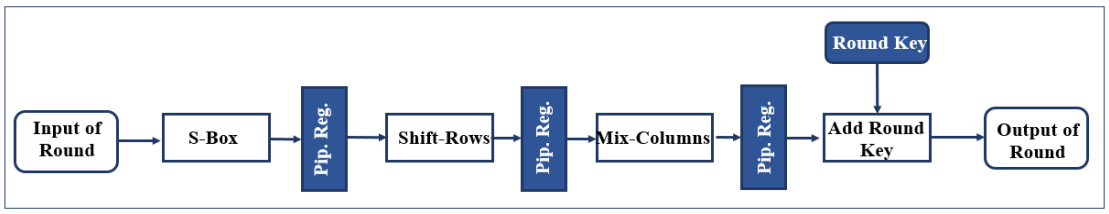

Figure 3. The block diagram of the sub-pipelined round of AES-256.

3.1. Efficient Mix-Columns Transformation

Mix-Columns transformation is calculated by applying **Eq. (4)** [14]. The Mix-Columns matrix includes numbers 00;01;02, and 03 only. Multiplication with 00 and 01 does not require much processing time. The multiplication with 03 can be implemented using **Eq. (6)** where a is a GF(2^8) element.

$$3 \times a = 2 \times a + a \tag{6}$$

We can write multiplication with 02 as shown in **Eq. (7):**

$$2 \times a = (a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0) \times (00000010) mod\,(X^8 + X^4 + X^3 + X + 1) \tag{7}$$

Consequently, we have:

$$2 \times a = (a_7X^8 + a_6X^7 + a_5X^6 + a_4X^5 + a_3X^4 + a_2X^3 + a_1X^2 + a_0X) \; mod(X^8 + X^4 + X^3 + X + 1) \quad (8)$$

By substituting **Eq. (8)** in **Eq. (7)** and then doing some simplification, we obtain **Eq. (9)**. This equation provides an efficient implementation of multiplication by 02 which does not require significant processing time.

$$\begin{aligned}(a_7X^8 + a_6X^7 + a_5X^6 + a_4X^5 + a_3X^4 + a_2X^3 + a_1X^2 + a_0X) \\ = a_7 \times (X^8 + X^4 + X^3 + X + 1) + a_6X^7 + a_5X^6 + a_4X^5 + (a_3 + a_7)X^4 \\ + (a_2 + a_7)X^3 + a_1X^2 + (a_0 + a_7)X + a_7\end{aligned} \quad (9)$$

3.2. Efficient S-box method using logic optimization based on the truth table

We apply an efficient pipelined S-box implementation. It uses combinational logic to solve the unbreakable delay caused by the look-up table. Moreover, It decreases the critical path delay incurred by complex field arithmetic. The transformation of the S-box has a 16 ×16 bytes table. So, its truth table has 128 rows. This truth table produces an output with an 8-bit length. Thus, it is very hard to minimize this complex and large table. The solution is to partition the main truth table of the S-box into smaller 16 sub-truth tables according to the least (or the most) significant 4 bits of the main truth table. These 4 bits will be the input of the 16 modules logic functions (from M1 to M16) as shown in **Figure 4**. Minimization of these functions is gotten using the Karnaugh map which is implemented using the Sum of Products approach by the basic gates. After the minimization of the sub-truth tables, sixteen 8-bit logic output functions are produced. Another four bits of data of least significant bits are chosen to be the input of a 16 to 1 multiplexer that will generate the S-box final output. An effective 16 to 1 multiplexer is executed using five small (4 to 1) multiplexers as shown in **Figure 4**. This method permits placing some pipelining registers (Pip. Reg. in **Figure 4**) between these multiplexers to carry out the sub-pipelining which results in a decrease of the critical delay path and improves the S-box speed and in its turn the whole AES-256 algorithm.

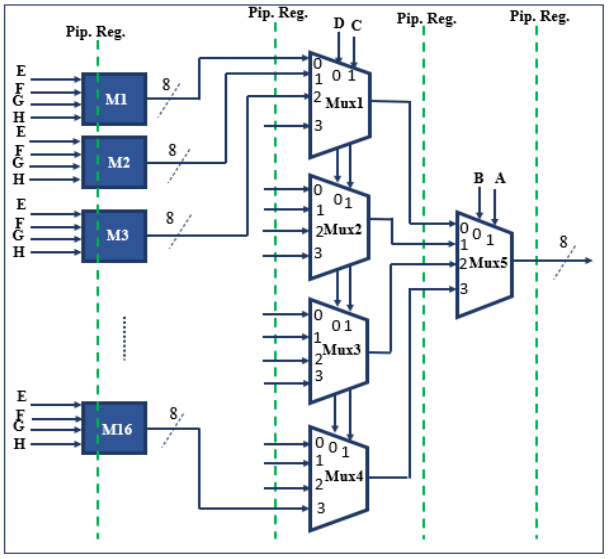

Figure 4. The architecture of the S-box using logic circuits.

4. Results and Comparison

4.1. Simulation Results

The AES-256 is coded using VHDL language and simulation results are taken using Vivado Design Suite 2019.1 from XILINX as a simulation tool. In the following figures, we simulated our design to create waveforms that represent the functionality AES-256. **Figure 5** shows the timing simulation of the S-box. Where signal *SI* represents S-box input, signal *So* represents the output, and signals *M1out: M6out* are the outputs of the 16 modules' logic functions. **Figure 6** shows the timing simulation of the key expansion unit. Where signal *Key* represents the original key of size 256 bit and signals K_1:K_{14} represent the sub-keys of size 128 bit. **Figure 7** shows the timing simulation of the whole AES-256. Where the input signal *CLK* is used to trigger the design, the input signal *RST* is used to reset the

design. Signal *Plaintext* represents the original message to be encrypted (clear text), signal *Key* represents the master key, signal *CipherTxt* represents the final encrypted message after 14 rounds. Signals K_1:K_{14} represent the sub-keys of size 128 bit and signals, CRs, are the ciphertexts after each round.

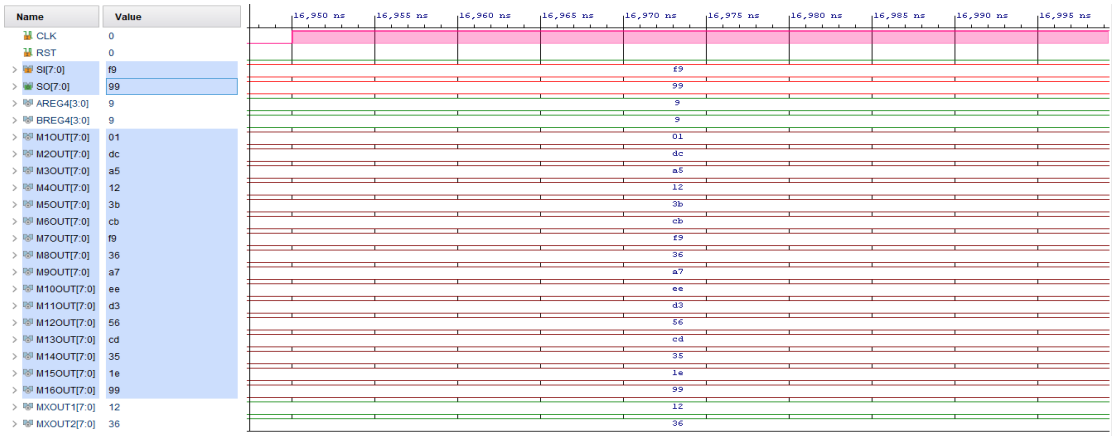

Figure 5. Timing simulation of the S-box transformation.

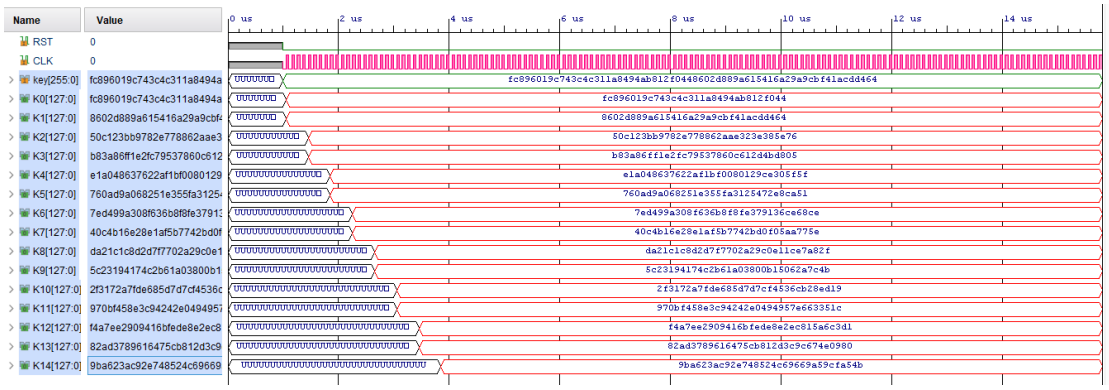

Figure 6. Timing simulation of the key expansion unit.

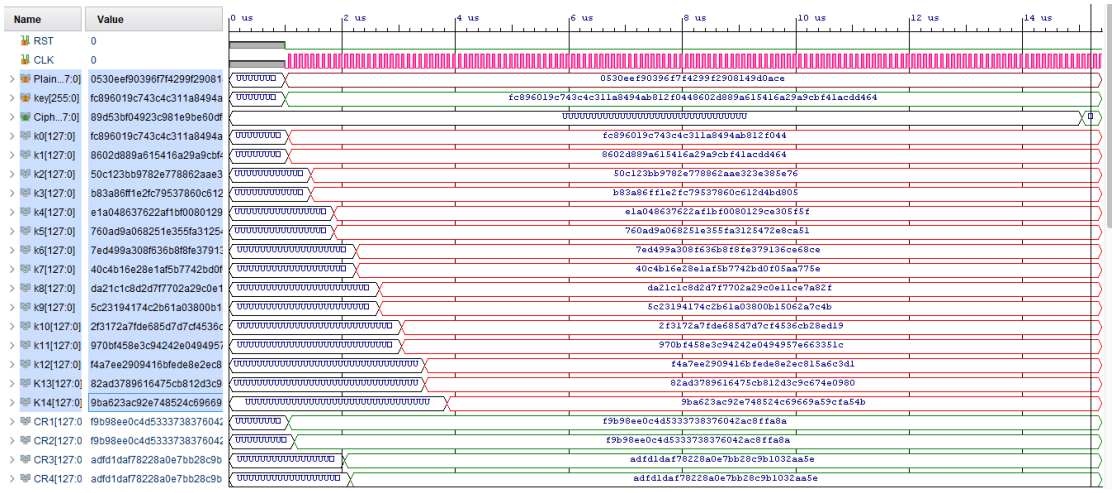

Figure 7. Timing Simulation of the whole AES-256 design.

4.2. Implementation Results

The FPGA implementation of the proposed AES-256 implementation is carried out using **Virtex-7 (**XC7VX485T-FFG1157-1**)** with Vivado Design Suite 2019.1 (with aid of Xilinx ISE 14.7) from Xilinx

as a synthesis tool. The top-level RTL schematics are given in **Figure 8, Figure 9, and Figure 10** to establish the fact that the HDL (Hardware Description Language) codes of the AES-256 are synthesizable. **Figure 8** shows the RTL schematic of S-box operation, which includes 16 logic functions (M1, M2, M3… M16), 5 multiplexers and pipelining registers as explained earlier in **Figure 4**. **Figure 9** shows the RTL schematic of a single round of the overall algorithm, which includes the main operations of each round (Add round key, Mix-columns, Shift rows and Sub bytes) and some pipelining registers as explained earlier in **Figure 2**. **Figure 10** shows the RTL schematic of the whole AES-256 algorithm, which includes 14 rounds, a key expansion unit and as explained earlier in **Figure 3**.

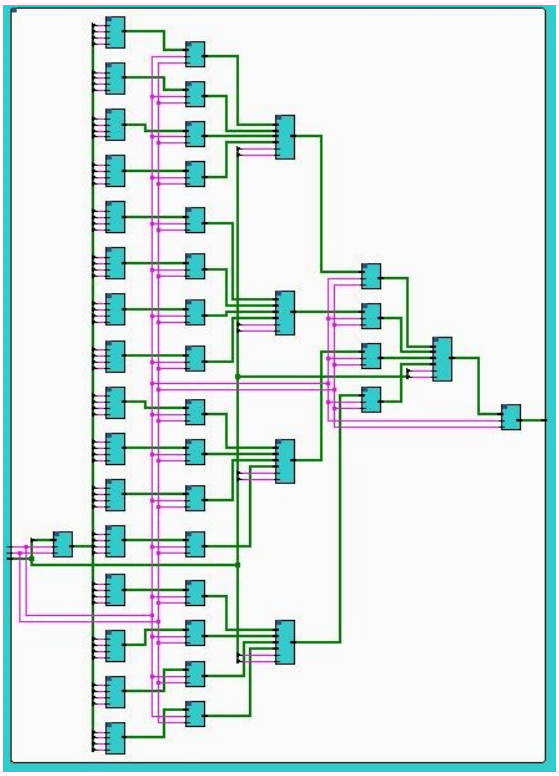

Figure 8. RTL schematic of S-Box operation.

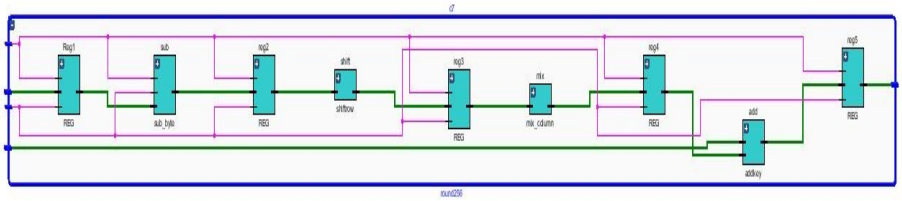

Figure 9. RTL schematic of one round of AES-256.

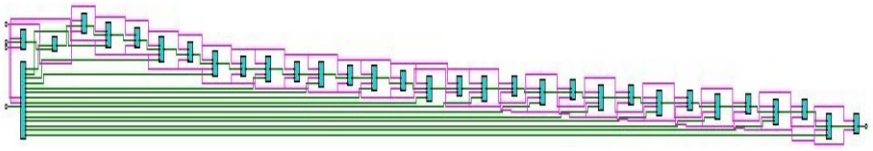

Figure 10. RTL schematic of AES-256 algorithm.

Performance metrics for our design include the maximum operational frequency, area or device utilization, and power consumption. The area is calculated in terms of FPGA look-up tables (LUT), flip-flops (FF), and general buffers (BUFG) to know the percentage of the used logic resources from the available resources as shown in **Table 2**. The total on-chip power consumption in FPGA circuits is estimated by summing static and dynamic power. Static power is due to the leakage current of transistors during steady-state. Dynamic power dissipation is due to components and clocks. **Figure 11** shows the power analysis summary of the design at the maximum clock frequency (**345.095 MHZ**).

Table 2. Device utilization summary of AES-256 implementation.

Logic Utilization	**U**tilization	**Available**	**Utilization**
LUT	31964	303600	10.53%
LUTRAM	832	130800	0.64%
FF	54377	607200	8.96%
IO	514	600	85.67%
BUFG	2	32	6.25%

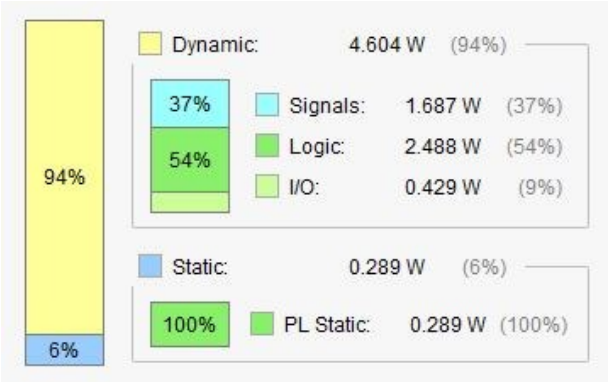

Figure 11. Power analysis of AES-256 implementation at maximum freq.

4.3. Comparison with Previous Work

Hardware implementation of encryption algorithms can be characterized by multiple performance parameters such as latency and throughput [28]. Encryption/decryption throughput is described as the number of encrypted or decrypted bits in a time unit. Typically, the encryption and decryption throughputs are equal. Consequently, only one parameter is reported. A typical unit of throughput is Mbit/s (megabit per second) or Gbit/s (gigabit per second) [28]. Encryption/decryption latency is described as the time consumed to encrypt a single block of plaintext or to decrypt its ciphertext. The typical unit of latency is ns (nanosecond). Latency and throughput are related by **Eq. 11** [28]. A comparison between our implementation and previous work in terms of throughput and the maximum frequency is given **in Table 3**. Our implementation achieves a very high operating frequency and throughput in comparison with some previous work.

$$Throughput = \frac{Block\ Size \times Number\ of\ Blocks\ Processed\ Simultaneously}{Latency} \quad (10)$$

Table 3. Comparison with previous work.

	Device	**Throughput**	**Max. Freq. (MHZ)**
Our work	Virtex-7(XC7VX485T-FFG1157-1)	**44.17 Gbps**	**345.095**
[29]	Virtex-5 (XC5VLX50)	829.99 Mbps	---
[30]	Spartan (XC3S500)	352 Mbps	---
[31]	Virtex-7 (XC7VX485T-FFG1157)	278 Mbps	161

5. Conclusions

In this paper, we have proposed an effective FPGA implementation for AES-256 using some efficient techniques such as loop-unrolling, and sub/fully-pipelining techniques. Moreover, we have applied other efficient solutions for the most complex parts of AES-256 such as Mix-columns and S-box. Our implementation of AES-256 is carried out using Virtex-7(XC7VX485T-FFG1157-1) FPGA**.** We have achieved high throughput of **44.17 Gbps** and a maximum operating frequency of **345.095 MHZ**.

References

[1] M. Faheem, S. Jamel, A. Hassan, Z. A., N. Shafinaz, and M. Mat, "A Survey on the Cryptographic Encryption Algorithms," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 11, pp. 333–344, 2017, doi: 10.14569/ijacsa.2017.081141.

[2] R. Yegireddi and R. K. Kumar, "A survey on conventional encryption algorithms of Cryptography," *Proc. 2016 Int. Conf. ICT Business, Ind. Gov. ICTBIG 2016*, 2017, doi: 10.1109/ICTBIG.2016.7892684.

[3] M. A. Siddiqui, K. Aslam, A. Afroz, and F. H. Rizvi, "Implementation of Advanced Encryption Standard (AES) on FPGA," *J. Comput. Sci. Newports Inst. Commun. Econ.*, vol. 7, no. 5, pp. 35–46, 2017.

[4] K. H. Rahouma, F. M. Abdelghany, L. N. Mahdy, and Y. B. E. Hassan, "Design and Implementation of a New DNA Based Stream Cipher Algorithm using Python," *Egypt. Comput. Sci. J.*, vol. 44, no. 1, pp. 102–113, 2020.

[5] M. T. Saleh, M. A. W. Shalaby, H. N. Elmahdy, and S. Engineering, "Modified Arnold ' s Cat Map -RC4 Encryption Technique for Medical Images," *Egypt. Comput. Sci. J.*, vol. 44, no. 2, pp. 11–23, 2020.

[6] A. M. Qadir and N. Varol, "A review paper on cryptography," *7th Int. Symp. Digit. Forensics Secur. ISDFS 2019*, pp. 1–6, 2019, doi: 10.1109/ISDFS.2019.8757514.

[7] M. S. Abdul-Karim, K. H. Rahouma, and K. Nasr, "High Throughput and Fully Pipelined FPGA Implementation of AES-192 Algorithm," in *Proceedings of 2020 International Conference on Innovative Trends in Communication and Computer Engineering, ITCE 2020*, 2020, pp. 137–142, doi: 10.1109/ITCE48509.2020.9047815.

[8] G. P. Saggese, A. Mazzeo, N. Mazzocca, and A. G. M. Strollo, "An FPGA-Based Performance Analysis of the Unrolling, Tiling, and Pipelining of the AES Algorithm BT - Field Programmable Logic and Application," 2003, pp. 292–302.

[9] H. Qin, T. Sasao, and Y. Iguchi, "A design of AES encryption circuit with 128-bit keys using look-up table ring on FPGA," *IEICE Trans. Inf. Syst.*, vol. E89-D, no. 3, pp. 1139–1147, 2006, doi: 10.1093/IETISY/E89-D.3.1139.

[10] U. Arom-Oon, "An AES cryptosystem for small scale network," in *Proceedings - ACDT 2017: 3rd Asian Conference on Defence Technology: Advance Research Collaboration on Defence Technology*, 2017, pp. 49–53, doi: 10.1109/ACDT.2017.7886156.

[11] Ritambhara, A. Gupta, and M. Jaiswal, "An enhanced AES algorithm using cascading method on 400 bits key size used in enhancing the safety of next generation internet of things (IOT)," in *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017*, 2017, pp. 422–427, doi: 10.1109/CCAA.2017.8229877.

[12] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 6th ed. USA: Prentice Hall Press, 2013.

[13] Y. Yuan, Y. Yang, L. Wu, and X. Zhang, "A High Performance Encryption System Based on AES Algorithm with Novel Hardware Implementation," 2018, doi: 10.1109/EDSSC.2018.8487056.

[14] A. Soltani and S. Sharifian, "An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA," *Microprocess. Microsyst.*, vol. 39, no. 7, pp. 480–493, 2015, doi: https://doi.org/10.1016/j.micpro.2015.07.005.

[15] R. Santhosh Kumar, R. Shashidhar, A. M. Mahalingaswamy, M. S. Praveen Kumar, and M. Roopa, "Design of High Speed AES System for Efficient Data Encryption and Decryption System using FPGA," in *3rd International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques, ICEECCOT 2018*, 2018, pp. 1279–1282, doi: 10.1109/ICEECCOT43722.2018.9001535.

[16] L. Yu, D. Zhang, L. Wu, S. Xie, D. Su, and X. Wang, "AES Design Improvements Towards Information Security Considering Scan Attack," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International*

Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 322–326, doi: 10.1109/TrustCom/BigDataSE.2018.00056.

[17] V. Gopi and E. Logashanmugam, "Design and analysis of nonlinear AES S-box and mix-column transformation with the pipelined architecture," 2013, pp. 235–238, doi: 10.1109/ICCTET.2013.6675955.

[18] G. Manjula and H. S. Mohan, "Improved Dynamic S-Box generation using Hash function for AES and its Performance Analysis," in *Proceedings of the 2nd International Conference on Green Computing and Internet of Things, ICGCIoT 2018*, 2018, pp. 109–115, doi: 10.1109/ICGCIOT.2018.8753074.

[19] P. Parikh and S. Narkhede, "High performance implementation of mixing of column and inv mixing of column for AES on FPGA," in *2016 International Conference on Computation of Power, Energy, Information and Communication, ICCPEIC 2016*, 2016, pp. 174–179, doi: 10.1109/ICCPEIC.2016.7557244.

[20] S. Sridevi Sathya Priya, M. Junias, S. Sarah Jenifer, and A. Lavanya, "Implementation of Efficient Mix Column Transformation for AES encryption," in *Proceedings of the 4th International Conference on Devices, Circuits and Systems, ICDCS 2018*, 2019, pp. 95–100, doi: 10.1109/ICDCSYST.2018.8605077.

[21] M. Sasikumar, K. N. Sreehari, and R. Bhakthavatchalu, "Systolic array implementation of mix column and inverse mix column of AES," in *Proceedings of the 2019 IEEE International Conference on Communication and Signal Processing, ICCSP 2019*, 2019, pp. 730–734, doi: 10.1109/ICCSP.2019.8697927.

[22] R. Riyaldhi, Rojali, and A. Kurniawan, "Improvement of Advanced Encryption Standard Algorithm With Shift Row and S.Box Modification Mapping in Mix Column," *Procedia Comput. Sci.*, vol. 116, pp. 401–407, 2017, doi: https://doi.org/10.1016/j.procs.2017.10.079.

[23] A. Barrera, C. W. Cheng, and S. Kumar, "Improved Mix Column Computation of Cryptographic AES," in *Proceedings - 2019 2nd International Conference on Data Intelligence and Security, ICDIS 2019*, 2019, pp. 229–232, doi: 10.1109/ICDIS.2019.00042.

[24] R. Andriani, S. E. Wijayanti, and F. W. Wibowo, "Comparision of AES 128, 192 and 256 bit algorithm for encryption and description file," in *Proceedings - 2018 3rd International Conference on Information Technology, Information Systems and Electrical Engineering, ICITISEE 2018*, 2018, pp. 120–124, doi: 10.1109/ICITISEE.2018.8720983.

[25] K. B. Jithendra and T. K. Shahana, "New Results in Related Key Impossible Differential Cryptanalysis on Reduced Round AES-192," *2018 Int. Conf. Adv. Commun. Comput. Technol. ICACCT 2018*, pp. 291–295, 2018, doi: 10.1109/ICACCT.2018.8529666.

[26] NIST, "Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)," *Natl. Inst. Stand. Technol.*, 2001, [Online]. Available: http://csrc.nist.gov/csor/.

[27] A. Murtaza, S. J. H. Pirzada, M. N. Hasan, T. Xu, and L. Jianwei, "Parallelized key expansion algorithm for advanced encryption standard," *Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS*, pp. 609–612, 2019, doi: 10.1109/ICSESS47205.2019.9040825.

[28] K. Gaj and P. Chodowiec, "FPGA and ASIC Implementations of AES," *Cryptogr. Eng.*, pp. 235–294, 2009, doi: 10.1007/978-0-387-71817-0_10.

[29] S. El Adib *et al.*, "Implementation of the AES-128 , AES-192 , and AES-256 On Virtex-5 FPGAs : Device Resources and Execution Time Reduction," vol. 1, no. 2, pp. 8–12, 2013.

[30] G. S. S. Venkateswarlu, Deepa G.M, "Implementation of AES-256 Encryption Algorithm on FPGA," *Int. J. Emerg. Eng. Res. Technol.*, vol. 3, no. 4, pp. 104–108, 2015.

[31] M. Gunasekaran, K. Rahul, and S. Yachareni, "Virtex 7 FPGA implementation of 256 Bit Key AES algorithm with key schedule and sub bytes block optimization," *2021 IEEE Int. IOT, Electron. Mechatronics Conf. IEMTRONICS 2021 - Proc.*, Apr. 2021, doi: 10.1109/IEMTRONICS52119.2021.9422547.