Virtex 7 FPGA Implementation of 256 Bit Key AES Algorithm with Key Schedule and Sub Bytes Block Optimization

Mahendrakumar Gunasekaran
Xilinx India Pvt Ltd, Hyderabad
gmah@xilinx.com

Kumar Rahul
Xilinx India Pvt Ltd, Hyderabad
kumarr@xilinx.com

Santosh Yachareni
Xilinx India Pvt Ltd, Hyderabad
santoshy@xilinx.com

***Abstract*— Hardware Security plays a major role in most of the applications which include net banking, e-commerce, military, satellite, wireless communications, electronic gadgets, digital image processing, etc. Cryptography is associated with the process of converting ordinary plain text into unintelligible text and vice versa. There are three types of cryptographic techniques; Symmetric key cryptography, Hash functions and Public key cryptography. Symmetric key algorithms namely Advanced Encryption Standard (AES), and Data Encryption Standard use the same key for encryption and decryption. It is much faster, easy to implement and requires less processing power. The proposed 256-bit AES algorithm is highly optimized in Key schedule and Sub bytes blocks, for Area and Power. The optimization has been done by reusing the S-box block. We are optimizing the algorithm with a new approach where internal operations are 32-bit operations, as compared to 128-bit operations. The proposed implementation helps in re-using the same hardware in a pipelined fashion which results in an area reduction by 72% using slice registers, 62% using slice LUT's and 61% using LUT-FF Pairs. This in turn results in a power reduction by 78% in a FPGA implementation. The throughput (Mbps) of the proposed implementation using Virtex-7 (xc7vx485tffg1157) FPGA improved by 10%.**

Keywords*— *AES *(Advanced Encryption Standard),* ***FPGA*** *(field programmable gate array),* ***LUT*** *(Look up table),* ***Mbps*** *(megabit per second),* ***sub*** *(sub bytes),* ***shift*** *(shift rows),* ***mix*** *(mix column),* ***add*** *(add round key).*

I. Introduction

Cryptography is associated with the process of converting ordinary plain text into unintelligible text and vice versa. There are three types of cryptographic technique namely - Symmetric key cryptography, Hash functions and Public key cryptography.

Symmetric key algorithms namely Advanced Encryption Standard (AES), Data Encryption Standard use the same key for encryption and decryption. It is much faster, easy to implement and requires less processing power.

In this paper, we have discussed about implementation of area, power and performance-based architecture of 256-bit key AES algorithm. Also, we discussed about the PPA comparison of the conventional and proposed based implementation in FPGA.

II. Architecture of AES Algorithm

A. *Architecture of AES algorithm*

AES algorithm implementation is done using four operations namely Sub Bytes, Shift Rows, Mix Columns and Add Round Key. Fig. 1 shows the architecture of 256-bit AES algorithm. In total there are 14 rounds of operation for encryption and 14 rounds for decryption. The ciphertext after encryption will be transmitted across the channel. The receiver side will decrypt the message using same key which is used in encryption.

In 256-bit AES algorithm, the key size is 256 bits, but all the data size is 128 bits. Data include message to be encrypted, cipher text and the decrypted message.

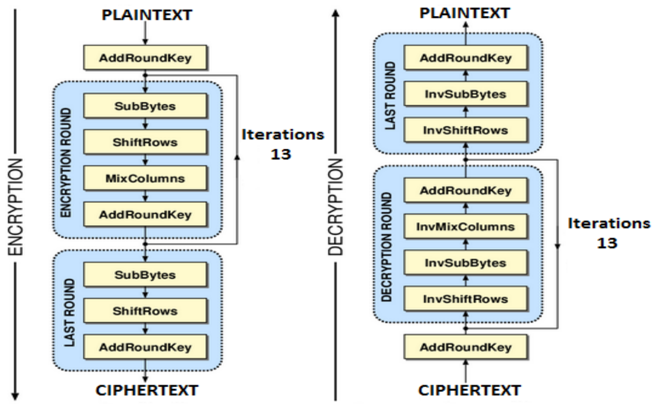

Fig. 1 Architecture of 256 AES Algorithm

Fig. 2 explains the internal data structure of 128-bit data. The 128-bit data is used as 4x4 matrix, where each elements of the matrix is of 8 bits. Since all the four operations are performed on columns basis, we convert the 128-bit data in 4x4 matrix with each element being 8 bits.

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

Fig. 2 Data Structure of 128-bit Message

III. Implementation of AES Algorithm

This paper explains about the implementation of both conventional and proposed architecture of 256 AES algorithm. This paper also compares the Power, Performance and Area number in FPGA implementation.

A. *Conventional Implementation of 256 bit AES Algorithm*

256-bit AES encryption block is implemented in 14 rounds. Each round consists of Add Round Key, Sub Bytes, Shift Rows, Mix column. Round 0 consists of only Add round Key operation as shown in Fig. 3. Round 14 consists of Sub Bytes, Shift Rows and Add Round Key operations, which need 3 clock cycles as shown in Fig. 3. Rounds 1 to 13 consists of all the four operations as shown in Fig. 13. We do a distinct operation in each clock cycle. Hence once the hardware has been implemented for Add Round Key, Sub Bytes, Shift Rows, Mix column, the same hardware can be used for all the 14 rounds [7] [10]. None of the four operations shares the

2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS) | 978-1-6654-4067-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/IEMTRONICS52119.2021.9422547

same clock cycle. Fig. 3 shows the sequence of round operation with specific sequence of 4 operations to complete the AES encryption. AES algorithm is serial process, i.e. output of first round is the input to the second round. Hence, we can use the same hardware for each round.

Round 0	01 cycle
Round 1 to 13	52 cycles
Round 14	03 cycles
Total	**56 cycles**

Table: 1 Cycles required in Each Round

		Cycle
Round 0	Add Round Key	1
	Sub bytes	2
	Shift Rows	3
	Mix column	4
Round 1	Add Round Key	5
	Sub bytes	6
	Shift Rows	7
	Mix column	8
Round 2	Add Round Key	9
	Sub bytes	10
	Shift Rows	11
	Mix column	12
Round 3	Add Round Key	13
	.	.
	.	.
	.	.
	Sub bytes	54
	Shift Rows	55
Round 14	Add Round Key	56

Fig. 3 Structure of conventional implementation

Fig. 3 shows the structure of conventional implementation of 256-bit key AES algorithm.

B. Data Structure

Fig. 4 shows the data structure of 128-bit matrix. Each column consists of 4 elements of 8 bits each, so in total we have 32 bits per word.

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Word 0 Word 1 Word 2 Word 3

Fig. 4 Word Format of 128-bit Data

Fig. 5 shows the number of S-box required and Mix Column required to implement conventional AES algorithm.

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

16 S box,
4 mix columns

Fig. 5 S-box required for conventional Method

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

4 S box,
1 mix column

Fig. 6 S box required for Proposed Method

Fig. 6 shows the proposed 32-bit AES implementation. We are doing operations per word (32-bits) in each cycle. Number of blocks required for conventional (128 bit) and proposed (32-bit) implementation are as follows

1) *S box - 16, 4 per clock cycle*
2) *Mix column block - 4, 1 per clock cycle*

In the proposed implementation, we are using the same S-box hardware for both encryption and decryption. Affine transform is the only difference between S-box (encryption) and inverse S-box (decryption). All the other logic (for encryption and decryption) is same for the S-box and inverse S-box. Hence, we are reusing every logic other than the Affine transform for encryption and decryption. Fig. 7 shows the mux selection between S-box and inverse S-box [1]. While doing AES encryption S-box path is chosen and while doing AES decryption inverse S-box path is chosen [1].

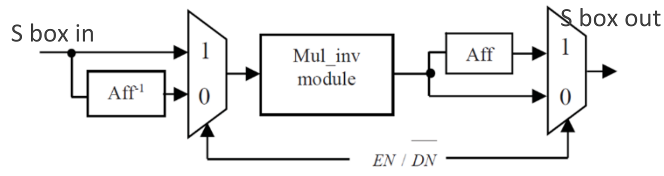

Fig. 7 Combined structure of S box and Inverse S box

C. Proposed 32 bit operations Implementation

In proposed 32-bit operation method, we are reusing S-box and Mix Column blocks. In proposed design "Mix Column" and "Add Round Key" together we called Mix block.

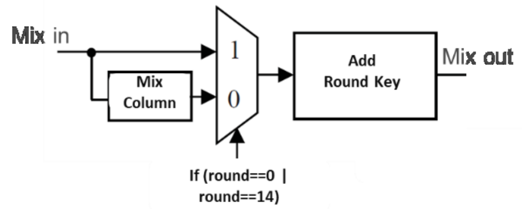

Fig. 8 Combined structure of Mix Column and Add Round Key - Mix

Fig. 9 shows the pipeline structure of the proposed design, where each color represents different round as follows,

Mix – round 0
Mix – round 1
Mix – round 2
Mix – round 3
Mix – round 14

S box	Shift	Mix	Cycle
		Mix_0	1
Sub_0	-	Mix_1	2
Sub_1	Shift_0	Mix_2	3
Sub_2	Shift_1	Mix_3	4
Sub_3	Shift_2		5
-	Shift_3	Mix_0	6
Sub_0	-	Mix_1	7
Sub_1	Shift_0	Mix_2	8
Sub_2	Shift_1	Mix_3	9
Sub_3	Shift_2	-	10
-	Shift_3	Mix_0	11
Sub_0	-	Mix_1	12
Sub_1	Shift_0	Mix_2	13
Sub_2	Shift_1	Mix_3	14
Sub_3	Shift_2	-	15
-	Shift_3	Mix_0	16
.	.	Mix_1	17
.	.	Mix_2	18
Sub_0	-	Mix_3	19
Sub_1	Shift_0	.	.
Sub_2	Shift_1	.	.
Sub_3	Shift_2	-	.
-	Shift_3	Mix_0	71
		Mix_1	72
		Mix_2	73
		Mix_3	74

This is round 0 and hence this mix is just add round key operation

Fig. 9 Pipelined structure of mix operation

From Fig. 9, each word having a size of 32 bits. In **cycle 1**, we are doing Mix operation of word 0 (mix_0). We can denote this as cycle1[round0(mix_0)].

Hence this 32-bit word is available to undergo 32-bit Sub operation. Hence in **cycle 2** we are doing sub operation of word 0 (sub_0) and Mix operation of word 1 (mix_1). We have valid input for "sub" block word 0 in clock cycle 2, and hence we don't need to wait for all the 4 words "mix" block operation to complete. We can denote this as cycle2[round1(sub_0), round0(mix_1)].

In clock **cycle 3**, we are doing sub operation for word 1 (sub_1), shift operation of word 0 (shift_0) and mix operation of word 2 (mix_2), and. We can denote this as cycle3[round1 (sub_1, shift_0), round0(mix_2)].

In clock **cycle 4**, we are doing sub operation for word 2 (sub_2) and shift operation for word 1 (shift_1) and mix operation for word 3 (mix_3). We can denote this as cycle4[round1(sub_2, shift_1), round0(mix_3)].

Since all 128-bit (4 words) round 0 mix operation completed, we don't have mix operation in cycle 5. In clock **cycle 5**, we are doing sub operation for word 3 (sub_3) and shift operation for word 2 (shift_2). We can denote this as cycle5[round1(sub_3, shift_2)].

Since all 128-bit (4 words) round 0 sub operation completed, we don't have sub operation in cycle 6. In clock **cycle 6**, we are doing round 1 shift operation of word 3 (shift_3) and mix operation of word 0 (mix_0) and. Since we already have the last byte value from sub_3, we are using that for mix_0. In this way we don't need to wait extra one cycle of shift operation to start the mix operation. We can denote this as cycle6[round1(shift_3, mix_0)].

In clock **cycle 7**, we are doing sub operation for word 0 (sub_0) and mix operation for word 1 (mix_1). We can denote this as cycle7[round2(sub_0), round1(mix_1)].

In clock **cycle 8**, we are doing sub operation for word 1 (sub_1) shift operation for word 0 (shift_0) and mix operation for word 2 (mix_2). We can denote this as cycle8[round2(sub_1, shift_0), round1(mix_2)].

In clock **cycle 9**, we are doing sub operation for word 2 (sub_2), shift operation for word 1 (shift_1) and mix operation for word 3 (mix_3). We can denote this as cycle9[round2 (sub_2, shift_1), round1 (mix_3)].

In clock **cycle 10**, we are doing sub operation for word 3 (sub_3) shift operation for word 2 (shift_2). The same order of execution repeats for all 14 rounds. We can denote this as cycle10[round2(sub_3, shift_2)]. The same sequence repeats for all 14 rounds.

S box	Shift	Mix	Cycle
		Mix_0	1
Sub_0	-	Mix_1	2
Sub_1	Shift_0	Mix_2	3
Sub_2	Shift_1	Mix_3	4
Sub_3	Shift_2	-	5
Key_2	Shift_3	Mix_0	6
Sub_0	-	Mix_1	7
Sub_1	Shift_0	Mix_2	8
Sub_2	Shift_1	Mix_3	9
Sub_3	Shift_2	-	10
Key_3	Shift_3	Mix_0	11
Sub_0	-	Mix_1	12
Sub_1	Shift_0	Mix_2	13
Sub_2	Shift_1	Mix_3	14
Sub_3	Shift_2	-	15
-	Shift_3	Mix_0	16
.	.	Mix_1	17
Key_14	.	Mix_2	18
Sub_0	-	Mix_3	19
Sub_1	Shift_0	.	.
Sub_2	Shift_1	.	.
Sub_3	Shift_2	-	.
-	Shift_3	Mix_0	71
		Mix_1	72
		Mix_2	73
		Mix_3	74

Key Generation for Round 2

Fig. 10 Pipelined structure of Key gen block

As shown in Fig. 10, in cycle 6 we are using sub bytes S box for key generation block, because of this we don't need extra S box for key generation block. We are generating 128-bit key for every 5 cycles, so that it requires only 4 S box in one cycle. In conventional method, we need 8 S box for key generation block. 128-bit key generated in cycle 6 will be used in cycle 14 mix operation. Similarly, key generated in cycle 11 used in mix operation of cycle 19.

As shown in Fig. 10, in cycle 9 we have valid output for round 1 (cycle 5 to 9), so we need 5 cycles to perform round 1. Total we need 74 clock cycles to complete AES encryption.

Round 0	04 cycles
Round 1 to 14	70 cycles
Total	**74 cycles**

Table: 2 Cycles required in each round

		Cycle
Round 0	Add Round Key	1
	Sub bytes	2
	Shift Rows	3
	Mix column	4
Round 1	Add Round Key	5
	Sub bytes	6
	Shift Rows	7
	Mix column	8
Round 2	Add Round Key	9
	Sub bytes	10
	Shift Rows	11
	Mix column	12
Round 3	Add Round Key	13
	.	.
	.	.
	.	.
	Sub bytes	54
	Shift Rows	55
Round 14	Add Round Key	56

Fig. 11 Conventional Method

S box	Shift	Mix	Cycle
		Mix_0	1
Sub_0	-	Mix_1	2
Sub_1	Shift_0	Mix_2	3
Sub_2	Shift_1	Mix_3	4
Sub_3	Shift_2	-	5
Key_2	Shift_3	Mix_0	6
Sub_0	-	Mix_1	7
Sub_1	Shift_0	Mix_2	8
Sub_2	Shift_1	Mix_3	9
Sub_3	Shift_2	-	10
Key_3	Shift_3	Mix_0	11
Sub_0	-	Mix_1	12
Sub_1	Shift_0	Mix_2	13
Sub_2	Shift_1	Mix_3	14
Sub_3	Shift_2	-	15
-	Shift_3	Mix_0	16
.	.	Mix_1	17
Key_14	.	Mix_2	18
Sub_0	-	Mix_3	19
Sub_1	Shift_0	.	.
Sub_2	Shift_1	.	.
Sub_3	Shift_2	-	.
-	Shift_3	Mix_0	71
		Mix_1	72
		Mix_2	73
		Mix_3	74

Fig. 12 Pipelined structure of proposed method

Fig. 11 and Fig. 12 shows the pipelined structure comparison between the implementation of conventional and proposed method.

D. *Results and Comparison*

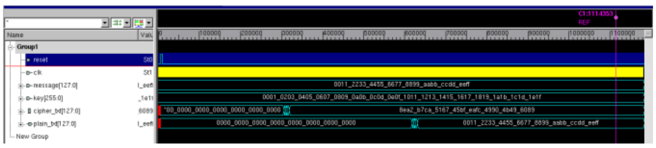

Fig. 13 Simulated waveform of standard 256-bit key example

Fig. 13 shows the VCS simulation of 256-bit key AES encryption and decryption. The cipher text is matching with standard 256 AES algorithm results. Also verified all the internal sub bytes, shift rows, mix columns and add round key output with standard test case for 256 key AES implementation.

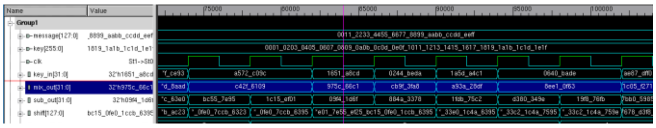

Fig. 14 Output of round 4 internal operations

```
round[ 3].k_sch    1651a8cd0244beda1a5da4c10640bade
round[ 4].start    975c66c1cb9f3fa8a93a28df8ee10f63
round[ 4].s_box    884a33781fdb75c2d380349e19f876fb
round[ 4].s_row    88db34fb1f807678d3f833c2194a759e
round[ 4].m_col    b2822d81abe6fb275faf103a078c0033
round[ 4].k_sch    ae87dff00ff11b68a68ed5fb03fc1567
round[ 5].start    1c05f271a417e04ff921c5c104701554
```

Fig. 15 Reference example from AES standard

Fig. 15 shows the round 4 internal operation outputs from AES standard example document "Federal Information Processing Standards Publication 197 November 26, 2001 Announcing the ADVANCED ENCRYPTION STANDARD (AES)"

Fig. 16 Simulated waveform of random inputs

Fig. 16 shows the VCS simulation of 256-bit key AES encryption and decryption with random message and 256-bit key. We are seeing plain text is matching with message.

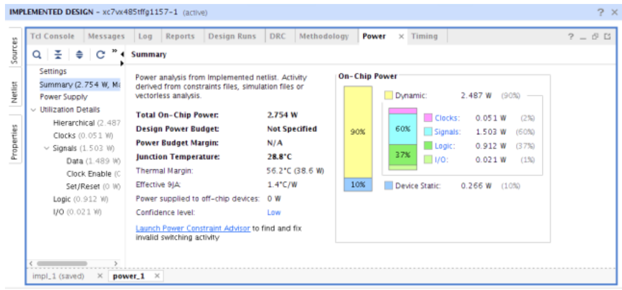

Fig. 17 Conventional implementation on-chip power

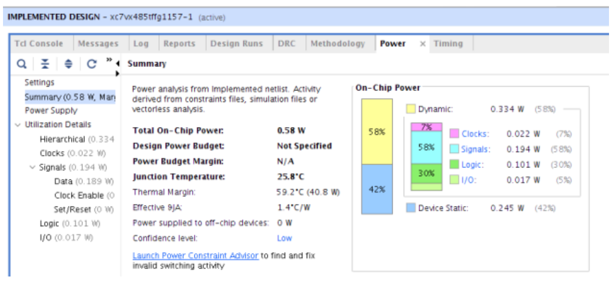

Fig. 18 Proposed implementation on-chip power

Fig. 17 & Fig. 18 shows the on-chip power in Vivado implementation for conventional and proposed methods.

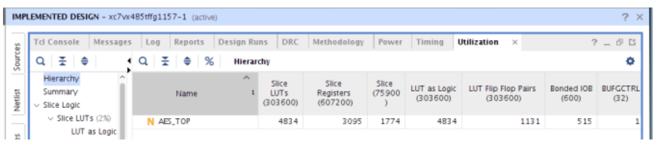

Fig. 19 Conventional implementation Area utilization

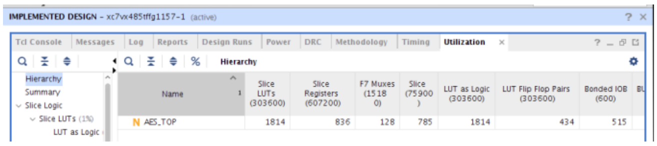

Fig. 20 Proposed implementation Area utilization

Fig. 19 & Fig. 20 shows the area utilization in Vivado implementation for conventional and proposed methods.

Block	Instance	Conventional	Proposed
Sub bytes	S box	16	4
Mix Column	Mix	4	1
Key Gen	S box	8	0

Table:3 Subblock utilization Comparison

Table:3 shows the comparison for modules used between final implementation of conventional and proposed 256-bit key AES algorithm.

Table: 3 shows that area reduced 4 times in sub bytes and mix column operations in conventional vs proposed methods. S box usage came down to 0 for Key Gen block, since we are reusing same S box of sub bytes.

	FPGA		
Method	Conventioı	Proposed	% Savings
Total cycles for encryption	56	74	
Frequency (MHz)	109	161	
Throughput (Mbps)	249	278	10.53
On chip power (W)	2.75	0.58	78.91
Slice LUT	4834	1814	62.47
Slice Register	3095	836	72.99
LUT Flip Flop Pairs	1131	434	61.63

Table: 4 PPA comparison Conventional vs Proposed

Table: 4 shows the PPA number comparison for conventional and proposed methods in FPGA implementation [6] [8].

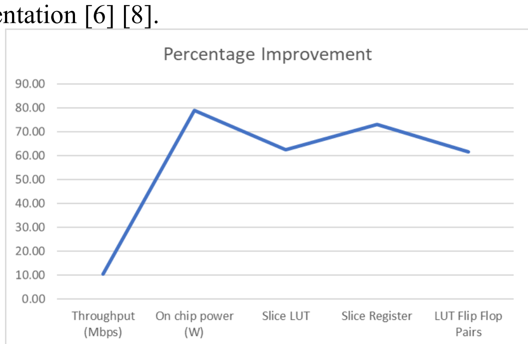

Fig. 23 PPA comparison Conventional vs Proposed

	FPGA		
Method	Proposed	[12]	[13]
Slice LUT	1814	3959	15376
Slice Register	836	1124	5356
LUT Flip Flop Pairs	434	973	2309

Table: 5 Area Comparison for Proposed vs Existing Methods

Table: 5 shows area utilization comparison for proposed vs existing methods [12] [13].

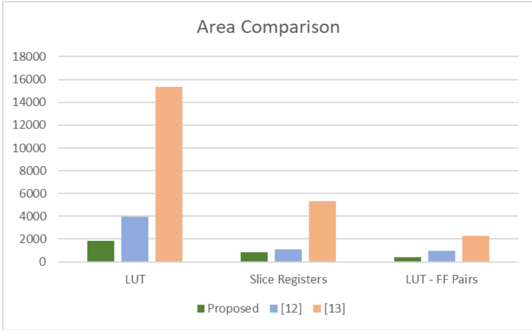

Fig. 24 Area comparison Proposed vs Existing Methods

Fig. 24 shows area utilization comparison in chart for proposed vs existing methods [12] [13].

IV. Conclusion

In this paper, we have compared the PPA numbers of conventional and proposed method in Virtex-7 (xc7vx485tffg1157) FPGA implementation of a 256-bit Key AES algorithm. The proposed implementation has an area reduction by 72% using slice registers, 62% using slice LUT's and 61% using LUT-FF Pairs. This results in a power reduction by 78%. The throughput (Mbps) of the proposed implementation improved by 10%. We proposed reusing the 32-bit Sub bytes and 32-bit Mix column blocks for 128-bit data, reusing the S-box for Sub Bytes and Key Schedule operations, reusing the same hardware for both encryption and decryption. The proposed method is generic and can be used for 128, 196 and 256-bit Key size. The proposed method is generic and can be used with word size operation of 16, 32, 64 bits.

V. References

[1] M. Rajeswara Rao, Dr.R.K.Sharma, SVE Department, NIT Kurushetra "FPGA Implementation of combined S box and Inv S box of AES" 2017 4th International conference on signal processing and integrated networks (SPIN).

[2] Nalini C. Iyer ; Deepa ; P.V. Anandmohan ; D.V. Poornaiah "Mix/InvMixColumn decomposition and resource sharing in AES".

[3] Xinmiao Zhang, Student Member, IEEE, and Keshab K. Parhi, Fellow, "High Speed VLSI architectures for the AES Algorithm", IEEE. VOL.12. No.9. September 2004

[4] Shrivathsa Bhargav, larry Chen, abhinandan Majumdar, Shiva Ramudit "128 bit AES Decryption", CSEE 4840 – Embedded system Design spring 2008, Columbia University.

[5] Atul M. Borkar ; R. V. Kshirsagar ; M. V. Vyawahare "FPGA implementation of AES algorithm".

[6] Announcing the ADVANCED ENCRYPTION STANDARD (AES), November 26 2001.

[7] Yulin Zhang ; Xinggang Wang; "Pipelined implementation of AES encryption based on FPGA" 2010 IEEE International Conference on Information Theory and Information Security.

[8] Yuwen Zhu ; Hongqi Zhang ; Yibao Bao ; "Study of the AES Realization Method on the Reconfigurable Hardware" 2013 International Conference on Computer Sciences and Applications.

[9] Tsung-Fu Lin ; Chih-Pin Su ; Chih-Tsun Huang ; Cheng-Wen Wu; "A high-throughput low-cost AES cipher chip" Proceedings. IEEE Asia-Pacific Conference on ASIC.

[10] C. Sivakumar ; A. Velmurugan ; "High Speed VLSI Design CCMP AES Cipher for WLAN (IEEE 802.11i)" 2007 International Conference on Signal Processing, Communications and Networking.

[11] Vatchara Saicheur ; Krerk Piromsopa ; "An implementation of AES-128 and AES-512 on Apple mobile processor" 2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)

[12] S.P Guruprasad ; B.S Chandrasekar ; "An evaluation framework for security algorithms performance realization on FPGA" 2018 IEEE

International Conference on Current Trends in Advanced Computing (ICCTAC)

[13] N. S. Sai Srinivas ; Md. Akramuddin; "FPGA based hardware implementation of AES Rijndael algorithm for Encryption and Decryption" 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT).

[14] P. S. Abhijith ; Mallika Srivastava ; Aparna Mishra ; Manish Goswami ; B. R. Singh ; "High performance hardware implementation of AES using minimal resources" 2013 International Conference on Intelligent Systems and Signal Processing (ISSP).

[15] Wei Wang ; Jie Chen ; Fei Xu ; "An implementation of AES algorithm Based on FPGA" 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery

[16] Ashwini M. Deshpande ; Mangesh S. Deshpande ; Devendra N. Kayatanavar; "FPGA implementation of AES encryption and decryption" 2009 International Conference on Control, Automation, Communication and Energy Conservation