Implementation of AES-128 Encryption Algorithm on FPGA

Shreyas Ravishankar, Ritika Ramchandani

***Abstract*— Security is of paramount importance in data communication system, where more randomization in secret keys increases the security and complexity of the cryptography algorithms. This results in compensating the algorithms with enormous memory spaces and large execution time on hardware platform. Field programmable gate arrays (FPGAs), provide one of the major alternative in hardware platform scenario due to its reconfiguration nature, low price and marketing speed. In FPGA based embedded system, the embedded processor can be used to execute particular algorithm with the inclusion of a real time operating System (RTOS), where threads may reduce resource utilization and time consumption.**

Symmetric key or single key encryption was the earliest known form of encryption. The Advanced Encryption Standard was published in 2001 and has been implemented using various hardware since then. A low power, portable and simple implementation of AES for devices with low computational power has also been developed. This paper studies the implementation of the AES-128 algorithm in detail and compares two of its methodologies with respect to the Substitute Byte operation preformed in the process of the algorithm. The comparison is made in terms of hardware utilization and power consumption.

***Index Terms*— AES algorithm, bytes, FPGA, plaintext, S-Box, transformation, Verilog.**

I. Introduction

Advanced Encryption Standard (AES), is also known by its original name Rijndael and is a cryptographic standard for encryption, now used transfer of electronic data over the internet or otherwise. It was established by the U.S. National Institute of Standards and Technology in 2001 and is now used worldwide. It succeeded its predecessor, the Data Encryption Standard, which was established in 1977, to become the first publicly usable algorithm for top secret data and was approved by the National Security Agency (NSA) in 2002.

Shreyas Ravishankar is now with the Department of Electronics and Communication Engineering, Birla Institute of Technology and Science (BITS) Pilani, Hyderabad, India (e-mail: f20160180@hyderabad.bits-pilani.ac.in).

Ritika Ramchandani is now with the Department of Electrical and Electronics Engineering, Birla Institute of Technology and Science (BITS) Pilani, Hyderabad, India (e-mail:f20150186@hyderabad.bits-pilani.ac.in).

The AES is a symmetric-key algorithm, meaning that the same key is used for both, encryption and decryption. Three algorithms of the Rijndael family were chosen for encryption. All three algorithms have a block size of 128 bits but vary in key-lengths. AES-128 has a 128-bit key. Similarly, AES-192 and AES-256 have key-lengths of 192 and 256 bits, respectively.

The paper has been organized as follows. Sections II and III give as general overview about the background of the algorithm and other similar algorithms that are used. Section IV explains the idea of the project and section V goes on to give a detailed description of AES-128. The hardware aspect and the results are covered by sections VI and VII.

II. Network Security

The AES-128 algorithm, though very algebraically simple, would take 1.02×10^{18} years to break, against a brute-force attack which would render it secure against all such attacks since the output is a non-linear mathematical function of the input. Although practical side-channel attacks exist for AES, no empirical cryptanalytic attack against it has been discovered. The best reasonable attack against the cipher is a *biclique attack*, which knocks off a few bits from the key but would still take a million years to obtain the plaintext since it takes $2^{126.2}$ operations to obtain the 128-bit key.

The computational security of an encryption cipher depends on the length of the key used. Longer keys cost exponentially more, in terms of computation cost as compared to shorter keys. To attack AES it would take 2^{88} bits of data, which amounts to 38 trillion terabytes. The memory requirement of such an attack is greater the sum of all data stored on all electronic resources in the world as of 2018. The space complexity of AES has recently been increased to 2^{53} bytes, which adds up to 9007 terabytes of memory power.

III. Related Work

Significant amount of work has been done lately regarding efficient AES implementation on FPGA. For example, the authors in [4] present a sequential implementation of AES algorithm on FPGA that results in a design that is well suited for small embedded applications. Implementation in [7] and [9] focusses on the design of low power FPGA-based encryption schemes. These schemes try to achieve best power results without compromising the throughput of the design. Three implementation schemes are presented which are compared in terms of area, and power consumption rates. Similarly authors in [5] present an efficient implementation of

AES algorithm on FPGA that results in high throughput and it is well suited for applications requiring high speed and performance. Furthermore, authors in [11] explore pipelining, sub-pipelining, and loop unrolling techniques to increase the frequency and throughput of AES implementation on FPGA.

[8] has presented a compact hardware-software co-design of Advanced Encryption Standard (AES) on the field programmable gate arrays (FPGA) designed for low-cost embedded systems. The design uses MicroBlaze, a soft-core processor from Xilinx. The computationally intensive operations of the AES are implemented in hardware for better speed.

IV. Proposed Idea

To do a comparative study of two methods of configuring the *Substitute Byte* module which are:

1. If-else implementation
2. The distributed memory generator IP (Intellectual Property Core) block

Within every configurable logic block, there are look-up tables. This data can also be stored in the form of bits on the RAM. A few of them can be consolidated on a bigger RAM. This is called distributed RAM. It can be utilized as a ROM/ RAM. The need for memory is to store the values of the S-box in a ROM and reduce the amount of MUX's (combinational logic) which would be used in case of the earlier kind of implementation using if-else statements. Vivado provides IPs for such an implementation.

V. The AES Algorithm

AES is based on a design principle known as a substitution–permutation network, and is efficient in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network.The algorithm implemented in this paper takes 128-bit plaintext and a 128-bit key as inputs to give 128-bit cipher text as output. The number of rounds varies depending on the key-size. For a 128-bit key, we require 10 rounds of computation.

TABLE I
AES ROUNDS FOR KEY LENGTH

Key size (in bits)	Number of Rounds
128	10
192	12
256	14

The first round consists of the *Add Round Key* transformation, followed by *N* rounds of four distinct transformations which are *Substitute bytes*, *Shift Row*, *Mix columns* and *Add Round Key*, which will be described subsequently. The last round, which is round 10, consists of the transformations *Substitute Bytes*, *Shift Row* and *Add Round Key*. Each of these transformations takes 4 x 4 matrices and produces 4 x 4 matrices as outputs. The key generation module is used for expansion of the 128-bit key into 11 128-bit keys, for each of the rounds computed in the algorithm. A detailed description of each of the operations to be performed is given below.

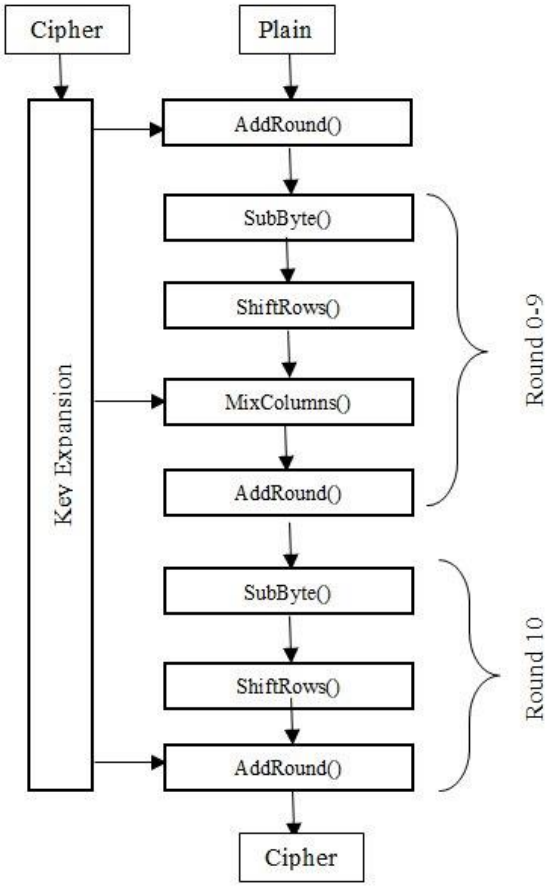

Fig. 1. The order of transformations to be performed in the AES Encryption Algorithm [5].

A. Substitute Bytes Transformation

This transformation, also known as the *SubBytes*, can be implemented either as a simple lookup table, or as a series of complex polynomial field arithmetic. This module takes a byte as input and produces a byte of data as output. The lookup location if determined by using the first four bits of the input byte as the row number and the last four bits as the column number. The row and column numbers serve as indexes to an 8-bit value that is unique, from a square matrix of 256 elements, consisting of all possible combinations of bytes.

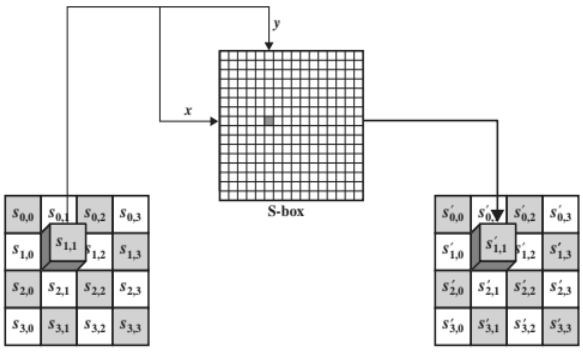

Fig. 2. Substitute byte transformation [2]

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Fig. 3. S-Box loop-up table for AES implementation [2]

The method of deriving the S-box using field polynomial involves initializing the 16 x 16 matrix cells with the byte worked out by concatenating the indexes of each cell. Each byte from the S-box array is then mapped to its multiplicative inverse in the finite field GF(2^8). The following transformation is applied to each bit in the S-Box to calculate the final entry.

$$b_i' = b_i \oplus b_{(i+4) mod\ 8} \oplus b_{(i+5) mod\ 8} \oplus b_{(i+6) mod\ 8} \oplus b_{(i+7) mod\ 8} \oplus c_i \quad (1)$$

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Fig. 4. Equivalent matrix transformation for the above equation

Ordinarily, matrix multiplication involves working out the products and finding their sums for every row against column. In field multiplication, the product is calculated as the bitwise XOR of the elements of every row against every column.

B. Shift Row Transformation

A state matrix is constructed using the input text by arranging it into a 4 x 4 matrix. Each element of the matrix is a byte. Each byte of the plaintext is arranged in a vertical manner before moving on to the next column. Once the matrix has been formed, the shift row transformation is performed. The first row of the matrix is not altered. The second row undergoes a 1-byte circular left shift. The third row is shifted to the left by 2 bytes. Finally, the last row of the matrix goes through a 1-byte right circular shift. This has been shown in Fig. 6.

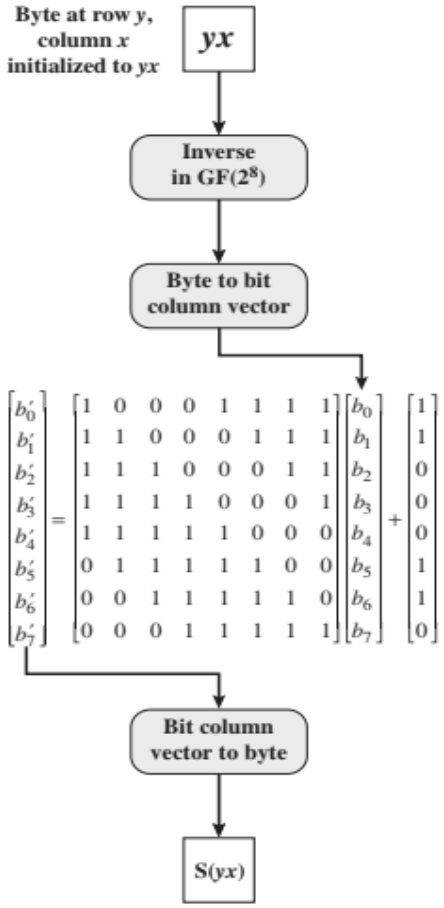

Fig. 5. Calculation of individual bytes of the S-Box. [2]

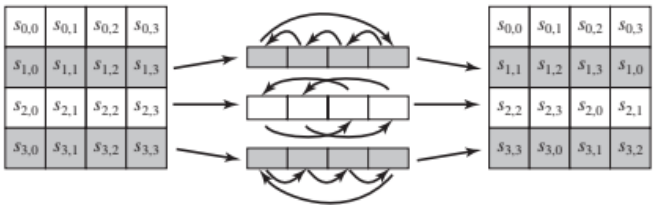

Fig. 6. Shift of each row bytewise. [2]

C. Mix Column Transformation

In this operation, each element of the state matrix is tranformed to a new byte which if a function of all the elements in that corresponding column. Such a transformation can be specified by the matrix multiplication shown in Fig. 7. The additions and multiplication operations pertaining to the matrices have been performed in the arithmetic field GF(2^8).

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Fig. 7. Matrix multiplication defining the Mix Column transformation

This operation can also be expressed in the form of equations. The new bytes are thus:

$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \quad (2)$$

$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \quad (3)$$

$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \quad (4)$$

$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \quad (5)$$

D. Add Round Key Transformation

Since the finite field of GF(2^8) identifies the addition operation as XOR, the 128-bits of the state matrix are bitwise XORed with the 128-bits of the key generated by the key expansion algorithm.

E. Key Expansion Algorithm

The key generation module takes the original key as input and provides 11 128-bits keys for each round of the algorithm. The last word (4 bytes) of the 16-byte long key is sent to a function g. This function performs a 1-byte left circular shift. Each byte of the word is then substituted using the S-Box. After substitution, the word is bitwise XORed with a round constant from an array of constants. The constant is chosen on the basis of the number of the round in process. The output of the g function is XORed with the first word of the previous key, to determine the first word of the key for the next round. Each of the three remaining words of the key is a result of the XOR operation performed between the previous word of the same key and the corresponding words of the key in the previous round.

TABLE II
ROUND CONSTANTS FOR KEY EXPANSION

Round	Hexadecimal Constant
1	01000000
2	02000000
3	04000000
4	08000000
5	10000000
6	20000000
7	40000000
8	80000000
9	18000000
10	36000000

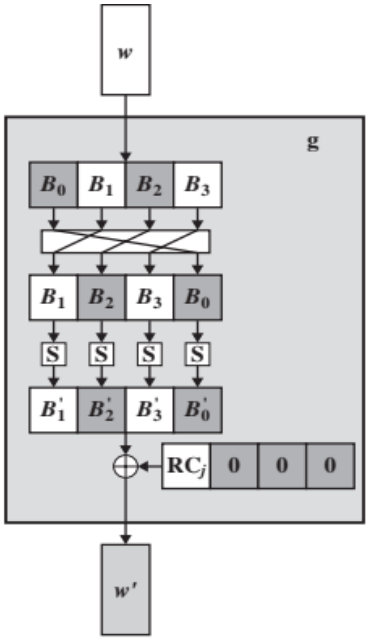

Fig. 8. Function g: The bytes are shifted left bytewise in a circular manner. The substitute byte operation is performed, followed by the XOR operation with a round key constant.

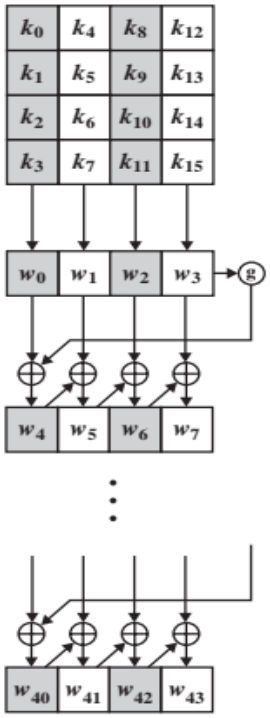

Fig. 9. Overall key expansion algorithm [2]

VI. HARDWARE IMPLEMENTATION

Verilog has been used as the hardware description language and has been synthesized and implemented using Vivado 2014.4. Simulation tools present of this software have been used to repeatedly test and debug the codes. The Xilinx Zynq-7000 FPGA board (Zedboard) has been used to realize this design.

Fig. 10. Zedboard Zynq-7000 ARM/FPGA SoC Development Board

The presence of 8 switches on the FPGA board allows an input of only 8 bits at a time. Similarly, only 8 bits of output can be diplayed on the board at a particular instance. Thus, the Virtual Input/Output (VIO) block were used to force the plaintext and 128-bit key as inputs. The outputs were also observed using the VIO feature.

The Substitute Byte Transformation has been implemented using two various methods:

1. Using if-else statements
2. Using the distributed memory generator IP block.

VII. Results

128-bit plaintext and a 128-bit key were given as inputs on the VIO. Hardware utilization after synthesis as well as after implementation were noted using the if-else model as well as the on-chip memory approach.

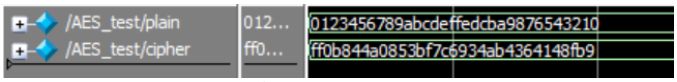

Fig. 11. Simulation results fo the AES 128-bit algorithm, implemented in Vivado.

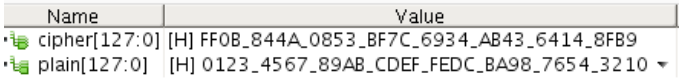

Fig. 12. Hexadecimal values of the given plaintext and corresponding ciphertext obtained after the AES-128 algorithm as shown on the VIO output screen.

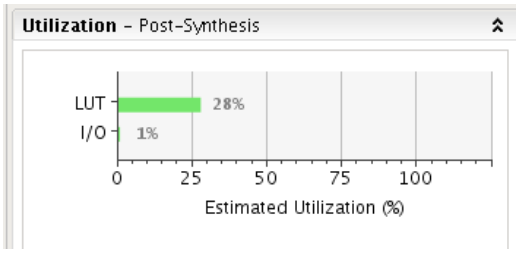

Fig. 13. Utilization summary post synthesis using if-else statements.

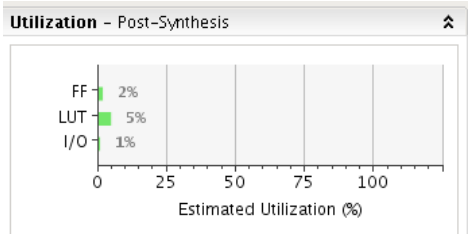

Fig. 14. Utilization summary post synthesis using distributed memory generator IP block

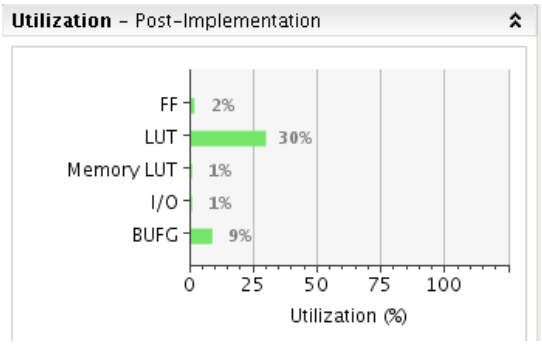

Fig. 15. Utilization summary post implementation using if-else statements

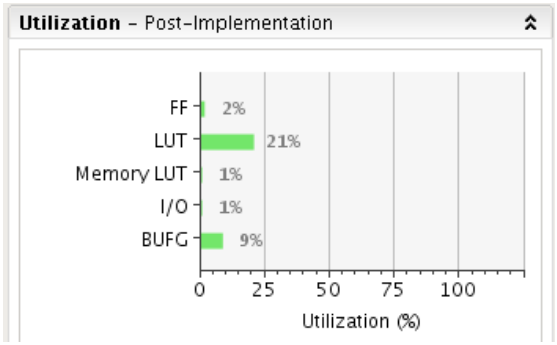

Fig.16. Utilization summary post implementation using distributed memory generator IP block

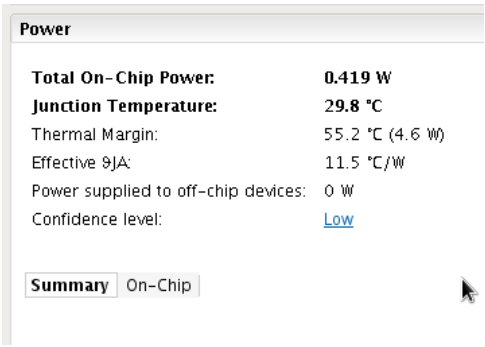

Fig. 17. Power used by the if-else design of S-Box

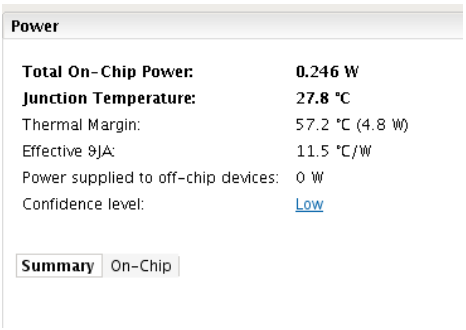

Fig. 18. Power used by the distributed on-chip memory approach.

As seen from the utilization reports, usage of FPGA LUTs is significantly lower while using the distributed memory IP block as compared to the if-else statements. The difference in post-synthesis usage of LUTs is almost 23% while post-implementation usage is 9%, while all other hardware usage remains the same. There is also difference between the two methodologies in terms of the number of flip-flops used. There is no use of flip-flop using the first method, while there is 2% use while using the on-chip RAM. While the usage of power in the first method adds up to 0.419W, the amount of power required in the second approach if almost half this value (0.246W).

VIII. Future Scope

This algorithm is used for secure electronic transfers over the internet. Thus large chunks of data are sent and received at any given time. For that reason, the algorithm can be extrapolated and implemented on the FPGA board for higher length plain text sequences, i.e implementing different modes of operation for AES. A Pipelined version of the algorithm can be implemented and subsequent comparison with its sequential counterpart can be studied. The weakening of the code using hardware Trojans and reverse engineering bitstreams can also be analyzed.

IX. Conclusion

The AES-128 algorithm provides low power and easily synthesizable implementation. Various approaches to the algorithm can help reduce power and hardware utilization. In this respect, optimality and efficiency was attained when S-Box was implemented using the distibuted memory IP core as compared to the if-else logic. Approximately 4% reduction in hardware utilization has been observed whereas redcution in power is 0.173 W.

X. Acknowledgement

It has been a pleasure to study FPGA Design and get to know its application in wide range of fields. We would specifically like to thank Prof. Sayan Kanungo, faculty of Electrical and Electronics department in BITS Pilani Hyderabad campus, for teaching the subject and guiding throughout the semester, making sure that progress is being made and helping us through any difficulties we faced. We also owe our gratitude to the PhD scholar Mr. Hariprasad who has assisted us in the lab work of this course.

XI. References

[1] M. Liberatori, F. Otero, J.C. Bonadero, "AES-128 Cipher. High Speed Low Cost FPGA Implementation", *Programmable Logic 2007. SPL '07. 2007 3rd Southern Conference on* , pp.195-198, 28–26 Feb. 2007.

[2] Willaim Stallings, “Advanced Encryption Standard,” in *Cryptography and Network Security*, 6th ed., U.S.A: Pearson, 2014, pp. 111-156.

[3] J. S. Grabowski and A. Youssef, "An FPGA implementation of AES with support for counter and feedback modes," 2007 International Conference on Microelectronics, Cairo, 2007, pp. 39-42.

[4] S. Ahmed, K. Samsudin, A. R. Ramli and F. Z. Rokhani, "Effective implementation of AES-XTS on FPGA," TENCON 2011 - 2011 IEEE Region 10 Conference, Bali, 2011, pp. 184-186.

[5] A. Aziz and N. Ikram, "An Efficient Fpga Based Sequential Implementation of Advanced Encryption Standard," *2005 International Conference on Information and Communication Technology* , Cairo, 2005, pp. 875-882.

[6] P. Swierczynski, M. Fyrbiak, P. Koppe and C. Paar, "FPGA Trojans Through Detecting and Weakening of Cryptographic Primitives," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 8, pp. 1236-1249, Aug. 2015.

[7] Xinmiao Zhang and Keshab K. Parhi, “Implementation Approaches for the Advanced Encryption Standard Algorithm,” IEEE Circuits and systems Magazine, vol. 2, no. 4, pp. 24–46, 2003.

[8] M. Biglari, E. Qasemi, B. Pourmohseni, "Maestro: A high performance AES encryption/decryption system", Computer Architecture and Digital Systems (CADS), 17th CSI International Symposium, pp.145-148, 30-31 Oct. 2013

[9] Xinmiao Zhang and Keshab K. Parhi, “High-Speed VLSI Architectures for the AES Algorithm.”, IEEE Transactions on Very Large Scale Integration(VLSI) Systems, Vol.12, No. 9, Septemper 2004.

[10] N. S. Sai Srinivas and MD. Akramuddin, “FPGA Based Hardware Implementation of AES Rijndael Algorithm for Encryption and Decryption”, International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) – 2016.

[11] C. Nalini P. Anandmohan Nagaraj "An FPGA Based Performance Analysis of Pipelining and Unrolling of AES Algorithm" Advanced Computing and Communications 2006 pp. 477-482 20–23 Dec. 2006.