International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 1, No. 2, July 2012, pp. 67~74
ISSN: 2089-4864

AES Encryption Algorithm Hardware Implementation: Throughput and Area Comparison of 128, 192 and 256-bits Key

Samir El Adib and Naoufal Raissouni
National School for Applied Sciences of Tetuan, University Abdelmalek Essaadi (www.UAE.ma).
Innovation & Telecoms Engineering Research Group. Remote Sensing & Mobile GIS Unit.
Mhannech II, B.P 2121 Tetuan, Morocco

Article Info

Article history:

Received May 6, 2012
Revised Jun 20, 2012
Accepted Jun 26, 2012

Keyword:

AES
BRAM
Cryptography
FPGA
VHDL

ABSTRACT

Advanced Encryption Standard (AES) adopted by the National Institute of Standards and Technology (NIST) to replace existing Data Encryption Standard (DES), as the most widely used encryption algorithm in many security applications. Up to today, AES standard has key size variants of 128, 192, and 256-bit, where longer bit keys provide more secure ciphered text output. In the hardware perspective, bigger key size also means bigger area and small throughput. Some companies that employ ultra-high security in their systems may look for a key size bigger than 128-bit AES. In this paper, 128, 192 and 256-bit AES hardware are implemented and compared in terms of throughput and area. The target hardware used in this paper is Virtex XC5VLX50 FPGA from Xilinx. Total area and Throughput results are presented and graphically compared.

Corresponding Author:

Naoufal Raissouni, Ph.D.,
National School for Applied Sciences of Tetuan, University Abdelmalek Essaadi,
Innovation & Telecoms Engineering Research Group. Remote Sensing & Mobile GIS Unit,
Mhannech II, B.P 2121 Tetuan, Morocco.
Email: nraissouni@uae.ma

1. INTRODUCTION

Security of data is becoming an important factor for a wide spectrum of embedded applications. Resistance against known attacks is one of the main properties that an encryption algorithm needs to provide. When a new attack is demonstrated as effective, the update of the encryption system is a real necessity to guarantee the security of data. Advanced Encryption Standard (AES) [1] [2] has replaced its predecessor, Double Encryption Standard (DES) [3] [4], as the most widely used encryption algorithm in many security applications. It offers a good ‘‘combination of security, performance, efficiency, implementability and flexibility’’ [5]. Although key size determines the strength of security, area and the power consumption issue has risen recently, especially in embedded hardware planted in mobile devices where lower area and power consumption becomes crucial. The trade-off between the level of security, throughput and area consumption is left in the hands of the implementers depending on the need. The authors of [6] show this trade-off based on DES encryption in their first graphical figure. They use a number of rounds to determine vulnerability.

In this paper, we present hardware implementations of the AES encryption using an approach which includes modules memory and lookup tables for 128-bit, 192-bit and 256-bit key. The higher the key size, the more secure the ciphered data, but also the more rounds needed. We simulated and synthesized an AES encryption algorithm hardware implementation using Very High Speed Integrated Circuit Hardware Description (VHDL) language and Xilinx ISE 9.1i simulator to see and compare throughput and area of hardware implementations of three variants of AES key sizes: 128, 192 and 256.

2. AES RIJNDAEL ALGORITHM

The AES Rijndael is a block cipher, which operates on different keys and block lengths: 128 bits, 192 bits, or 256 bits. The input to each round consists of a block of message called the state and the round key. It has to be noted that the round key changes in every round. The state can be represented as a rectangular array of bytes. This array has four rows; the number of columns is denoted by *Nb* and is equal to the block length divided by 32. The same could be applied to the cipher key. The number of columns of the cipher key is denoted by *Nk* and is equal to the key length divided by 32. The cipher consists of a number of rounds - that is denoted by *Nr* - which depends on both block and key lengths. Each round of Rijndael encryption function consists mainly of four different transformations: SubByte, ShiftRow, MixColumn and key addition. On the other hand, each round of Rijndael decryption function consists mainly of four different transformations: InvSubByte, InvShiftRow, InvMixColumn, and key addition.

The 128-bit data block and key are considered as a byte array, respectively called “State” and “RoundKey”, with four rows and four columns. The description of the four transformations of the Rijndael cipher and their inverses will be given below.

$$\text{State} = \begin{bmatrix} d_{15} & d_{11} & d_7 & d_3 \\ d_{14} & d_{10} & d_6 & d_2 \\ d_{13} & d_9 & d_5 & d_1 \\ d_{12} & d_8 & d_4 & d_0 \end{bmatrix} \quad (1)$$

2.1. SubByte Transformation

The SubByte (SB) transformation is a non-linear byte substitution, operating on each of the state bytes independently. The SubByte transformation is done using a once-pre-calculated substitution table called S-box[9] [10] [11]. That S-box table contains 256 numbers (from 0 to 255) and their corresponding resulting values. The SubByte transformation applied to the State can be represented as follows:

$$\text{SB(State)} = \begin{bmatrix} SB(d_{15}) & SB(d_{11}) & SB(d_7) & SB(d_3) \\ SB(d_{14}) & SB(d_{10}) & SB(d_6) & SB(d_2) \\ SB(d_{13}) & SB(d_9) & SB(d_5) & SB(d_1) \\ SB(d_{12}) & SB(d_8) & SB(d_4) & SB(d_0) \end{bmatrix} \quad (2)$$

2.1.1. InvSubByte Transformation

The InvSubByte transformation is done using a once-pre-calculated substitution table called InvS-box[12]. That table (or InvS-box) contains 256 numbers (from 0 to 255) and their corresponding values.

2.1.2. ShiftRow Transformation

In ShiftRow (SR) transformation, the rows of the state are cyclically left shifted over different offsets. Row 0 is not shifted; row 1 is shifted over one byte; row 2 is shifted over two bytes and row 3 is shifted over three bytes. Thus, the ShiftRow transformation proceeds as follows:

$$\text{SR(SB (State))} = \begin{bmatrix} SB(d_{15}) & SB(d_{11}) & SB(d_7) & SB(d_3) \\ SB(d_{10}) & SB(d_6) & SB(d_2) & SB(d_{14}) \\ SB(d_5) & SB(d_1) & SB(d_{13}) & SB(d_9) \\ SB(d_0) & SB(d_{12}) & SB(d_8) & SB(d_4) \end{bmatrix} \quad (3)$$

2.1.3. InvShiftRow Transformation

In InvShiftRow transformation, the rows of the state are cyclically right shifted over different offsets. Row 0 is not shifted, row 1 is shifted over one byte, row 2 is shifted over two bytes and row 3 is shifted over three bytes.

2.1.4. MixColumn Transformation

In Mix-Column, the columns of the state are considered as polynomials multiplied by a fixed polynomial $c(x)$, given by:

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

The MixColumn (MC) transformation can be written in a matrix multiplication as follows:

$$R = MC(SR(SB\ (State))) = \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \otimes \begin{bmatrix} SB(d_{15}) & SB(d_{11}) & SB(d_7) & SB(d_3) \\ SB(d_{10}) & SB(d_6) & SB(d_2) & SB(d_{14}) \\ SB(d_5) & SB(d_1) & SB(d_{13}) & SB(d_9) \\ SB(d_0) & SB(d_{12}) & SB(d_8) & SB(d_4) \end{bmatrix} \quad (5)$$

2.1.5. InvMixColumn Transformation

In InvMixColumn, the columns of the state are considered as polynomials multiplied by a fixed polynomial $d(x)$, defined by:

$$c(x) \otimes d(x) = '01' \quad (6)$$
$$d(x) = '0B'x^3 + '0D'x^2 + '09'x + '0E' \quad (7)$$

2.1.6. AddRoundKey

AddRoundKey (AK) performs an addition (bitwise XOR) of the State with the RoundKey:

$$AK(R) = \begin{bmatrix} R_{15} & R_{11} & R_7 & R_3 \\ R_{14} & R_{10} & R_6 & R_2 \\ R_{13} & R_9 & R_5 & R_1 \\ R_{12} & R_8 & R_4 & R_0 \end{bmatrix} \oplus \begin{bmatrix} rk_{15} & rk_{11} & rk_7 & rk_3 \\ rk_{14} & rk_{10} & rk_6 & rk_2 \\ rk_{13} & rk_9 & rk_5 & rk_1 \\ rk_{12} & rk_8 & rk_4 & rk_0 \end{bmatrix} \quad (8)$$

The inverse operation (InvAddRoundKey (IAK)) is trivial.

Round Keys are calculated with the key schedule for every AddRoundKey transformation. In AES-128, the original cipher key is the first (rk^0) used in the additional AddRoundKey at the beginning of the first round.

rk^i, where $0 < i \leq 10$, is calculated from the previous rk^{i-1}. Let $q(j)(0 \leq j \leq 3)$ be the column j of the rk^{i-1} and let $w(j)$ be the column j of the rk^i. Then the new rk^i is calculated as follows:

$$w(0) = q(0) \oplus (Rot(SB(q(3)) \oplus rcon^i$$
$$w(1) = q(1) \oplus w(0)$$
$$w(2) = q(2) \oplus w(1)$$
$$w(3) = q(3) \oplus w(2)$$

The round constant $rcon^i$ contains values $['02'^{i-1}; '00'; '00'; '00']$. Rot is a function that takes a four byte input and shifted over one byte.

3. PROPOSED LOOK-UPS TABLES APPROACH

The approach we propose is based on the combination of MixColumn and SubByte transformation into a single table consisting of 256 1-bytes columns. Compared to the 8x32 bits wide T-box look up tables [7], the tables proposes are of the size of 8x8 bits. The description described below explains how tables look-ups and the corresponding AES round operations are obtained: As also mentioned in (5), the consecutive SubByte and MixColumn operations on the first quarter of the round can be expressed as:

$$R = MC\big(SB(SR\ (State))\big) = A(x) \otimes SB(SR\ (State)) \quad (9)$$

$$A(x) = \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \quad (10)$$

$State$ is the data transformed, and $A(x)$ is the matrix of multiplicative vectors. The above multiplication may be performed by using logarithm and anti-logarithm table (see Tables 1 and 2, respectively).

For example: $C = a \times b$
C can be computed by using logarithm tables in the following way:

$$C = Log'\big((Log\ a) + (Log\ b)\big) \quad (11)$$

The mix-columns transformation computes each row separately. In order to compute the matrix multiplication of expression (9) and exploiting the expression (11), all of the bytes are substituted by using the logarithm tables (addition rather than a multiplication). If we define four tables (T0 to T3) containing 256 numbers (from 0 to 255) data as:

Tables for encryption:

$$To(a) = Log'\Big(\big(Log\ (01)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$
$$T1(a) = Log'\Big(\big(Log\ (01)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$
$$T2(a) = Log'\Big(\big(Log\ (02)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$
$$T3(a) = Log'\Big(\big(Log\ (03)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$

Table 1. Logarithm Table

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

Table 2. Anti-Logarithm Table

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

The final result will obtain by XORing the output of four tables (T0 to T3) as given by the following expression:

$R_{15} = T_2(d_{15})\ xor\ T_3(d_{10})\ xor\ T_1(d_5)\ xor\ T_0(d_0)\ xor\ \mathrm{rk}_{15}$;
$R_{14} = T_0(d_{15})\ xor\ T_2(d_{10})\ xor\ T_3(d_5)\ xor\ T_1(d_0)\ xor\ \mathrm{rk}_{14}$;
$R_{13} = T_1(d_{15})\ xor\ T_0(d_{10})\ xor\ T_2(d_5)\ xor\ T_3(d_0)\ xor\ \mathrm{rk}_{13}$;
$R_{12} = T_3(d_{15})\ xor\ T_1(d_{10})\ xor\ T_0(d_5)\ xor\ T_2(d_0)\ xor\ \mathrm{rk}_{12}$;

$R_{11} = T_2(d_{11})\ xor\ T_3(d_6)\ xor\ T_1(d_1)\ xor\ T_0(d_{12})\ xor\ \mathrm{rk}_{11}$;

$R_{10} = T_0(d_{11})\ xor\ T_2(d_6)\ xor\ T_3(d_1)\ xor\ T_1(d_{12})\ xor\ \mathrm{rk}_{10}$;
$R_9 = T_1(d_{11})\ xor\ T_0(d_6)\ xor\ T_2(d_1)\ xor\ T_3(d_{12})\ \ xor\ \mathrm{rk}_9$;
$R_8 = T_3(d_{11})\ xor\ T_1(d_6)\ xor\ T_0(d_1)\ xor\ T_2(d_{12})\ \ xor\ \mathrm{rk}_8$;

$R_7 = T_2(d_7)\ xor\ T_3(d_2)\ xor\ T_1(d_{13})\ xor\ T_0(d_8)\ \ xor\ \mathrm{rk}_7$;
$R_6 = T_0(d_7)\ xor\ T_2(d_2)\ xor\ T_3(d_{13})\ xor\ T_1(d_8)\ \ xor\ \mathrm{rk}_6$;
$R_5 = T_1(d_7)\ xor\ T_0(d_2)\ xor\ T_2(d_{13})\ xor\ T_3(d_8)\ \ xor\ \mathrm{rk}_5$;
$R_4 = T_3(d_7)\ xor\ T_1(d_2)\ xor\ T_0(d_{13})\ xor\ T_2(d_8)\ \ xor\ \mathrm{rk}_4$;

$R_3 = T_2(d_3)\ xor\ T_3(d_{14})\ xor\ T_1(d_9)\ xor\ T_0(d_4)\ \ xor\ \mathrm{rk}_3$;
$R_2 = T_0(d_3)\ xor\ T_2(d_{14})\ xor\ T_3(d_9)\ xor\ T_1(d_4)\ \ xor\ \mathrm{rk}_2$;
$R_1 = T_1(d_3)\ xor\ T_0(d_{14})\ xor\ T_2(d_9)\ xor\ T_3(d_4)\ \ xor\ \mathrm{rk}_1$;
$R_0 = T_3(d_3)\ xor\ T_1(d_{14})\ xor\ T_0(d_9)\ xor\ T_2(d_4)\ xor\ \mathrm{rk}_0$;

In the last round Mixcolumn transformation is excluded, while SubByte operation has to be performed.

Key Generator: The purpose of key size expansion is to discover the relationship between key size and area for various key sizes and for some institutions, 128-bit AES is just not sufficient for their ultra-high security. The key size increases, but the ciphered text output would still be 128 bits long. The changes are made in terms of the number of rounds necessary to complete one encryption process and also the key expansion algorithm figure 1.

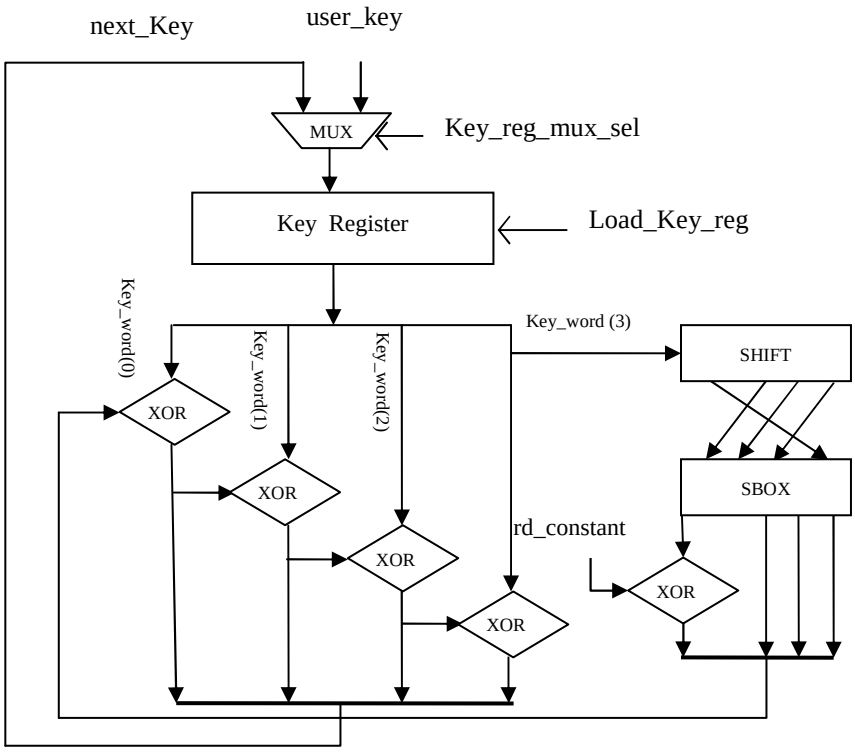

Figure 1. Block Diagram of Key_schedule Module

4. SYNTHESIS, THROUGHPUT, AREA RESULTS

This section shows the implementation results, consisting of functionality test results and Throughput and area results after synthesis. For 128/192/256-bit AES, test results are compared with FIPS 197 Documentation [8, Appendix C].

We use the Xilinx Virtex-5 XC5VLX50 FPGA which has advanced features that are useful for our application beyond traditional LUTs and registers. The results of mapping, for the key sizes of 128bits, 192 bits and 256 bits based on the 8x8bit look up tables are summarized in the Table 3.

Table 3: Implementation Results

Key size	Slices	BRAMs	Max. Freq. (MHz)	Clock Cycle Used	Throughput (Mbps)	Performance (Mbps)/Slices
128	587	2	346,194	104	426,08	0,73
192	746	2	315,348	126	320,35	0,43
256	1140	2	321,642	156	263,91	0,23

Figure 2. Shows the throughputs obtained for implementation of both the architectures of the combined encryption unit with three different key sizes. Figure 3. Shows the area results. Area is particularly important for those chip designers whose objective is to minimize area in chip fabrication or where large chip area is not desirable.

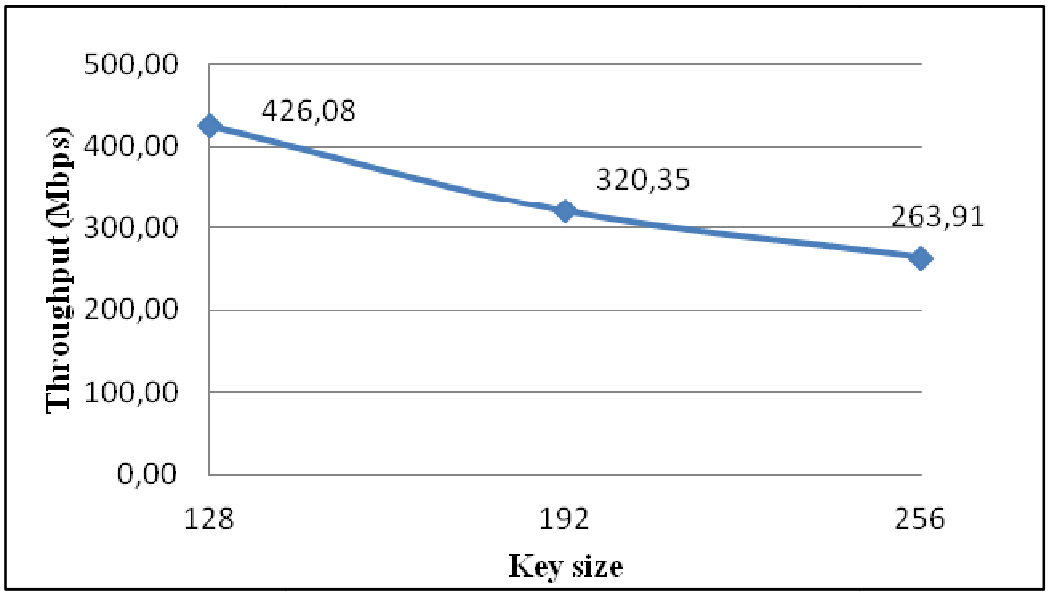

Figure 2. 128/192/256-bits key size AES throughput result

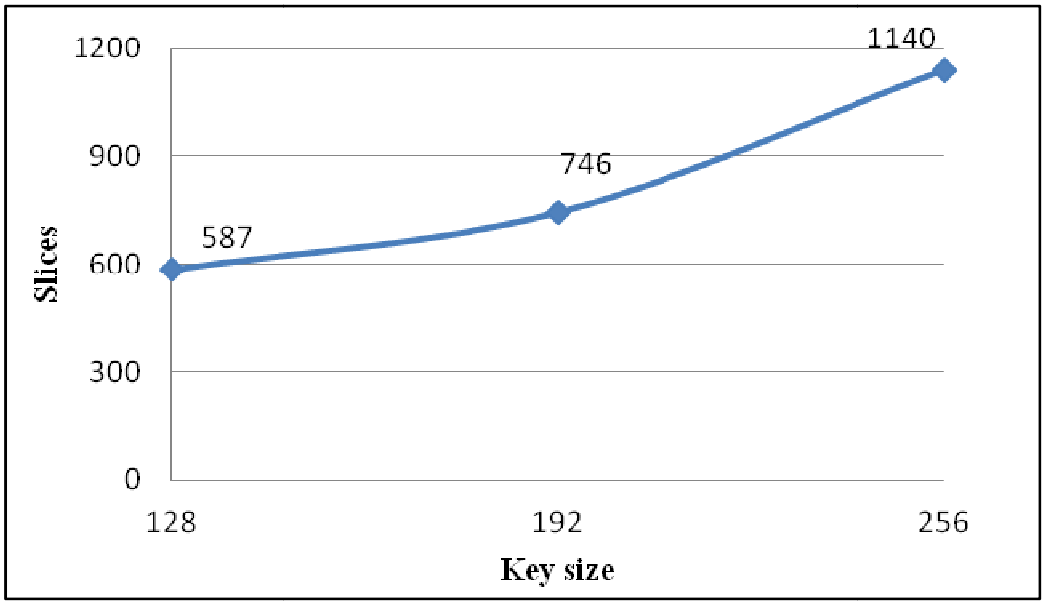

Figure 3. 128/192/256-bits key size AES area result

Back to Figure 1, one may find that increasing key size has almost linear impact on Throughput. On the other hand, key size versus area graph in Figure 2 shows exponential growth as key size increases.

5. CONCLUSION

In the present paper, throughput and area of 128, 192 and 256-bits AES have been measured in a hardware implementation. Results show that; key size has an almost-linear impact on throughput whereas it has an exponential positive relation with area. In terms of area 192-bits and 256-bits AES hardware design in this paper require about 21.31% and 48.51%, respectively, more area than 128-bits AES design. The tradeoff decision between level of security and power dissipation and area is left for designers or application engineers implementing the AES algorithm for their projects.

ACKNOWLEDGEMENTS

This work was supported in part by the Ministry for Higher Education, Management Training and Scientific Research under CSPT Grants for “Integration and application of GIS and GPS on mobile systems” and “Ad-hoc wireless sensor networks for remote sensing algorithm validation” projects. The authors would like to express gratitude to external anonymous referees whose comments and suggestions improved this manuscript.

REFERENCES

[1] J. Daemen and V. Rijmen, "AES Proposal: Rijndael. NIST AES Proposal," June 1998. Available at http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf.
[2] National Institute of Standards and Technology (U.S.), "Advanced Encryption Standard (AES)," Available at http://csrc.nist.gov/publications/drafts/dfips-AES.pdf.
[3] ANSI (American National Standards Institute), "Triple Data Encryption Algorithm Modes of Operation," 1998.
[4] National Institute of Standards and Technology (U.S.), "Data Encryption Standard (DES),"*FIPS Publication 46-3, NIST, 1999*. Available at http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
[5] A. Rudra, P.K. Dubey, C.S. Jutla, V. Kumar, J.R. Rao, P. Rohatgi, "Efficient Rijndael encryption implementation with composite field arithmetic," *Lecture Notes in Computer Science* 2162 (2001) 171–184.
[6] S. Bapatla, R. Chandramouli, "Battery power optimized encryption, " in: *IEEE International Conference on Comm.*, vol. 7, June 2004.
[7] J. Daemen, V. Rijmen, " AES proposal: The Rijndael Block Cipher, " Version 2 (Sept. 1999) pp. 1–45.
[8] National Institute of Standards and Technology (NIST), Specification for the Advanced Encryption Standard (AES), *FIPS* 197, November 26, 2001.
[9] A.Aziz and N. Ikram, “Memory efficient implementation of AES S-boxes on FPGA”, *Journal of Circuits, Systems, and Computers*, Vol. 16, No. 4, pp. 603-611, 2007.
[10] F. R-. Henriquez, N. A. Saqib and A. D-. Perez, “4.2 Gbits/s single chip FPGA implementation of AES algorithm”, *Electronics Letters*, Vol. 39, No. 15, pp. 1115-1116, 2003.
[11] I. A-. Badillo, C. F-. Uribe and R. C-. Para, “Design and implementation of an FPGA-based 1.452 Gbps non pipelined AES architecture”, in Proc. of the International Conference on Computational Science and its applications, *Lecture Notes in Computer Science, Springer-Verlag*, Vol. 3982, pp. 446-455, 2006.
[12] D. S. Kundi, S. Zaka, Q. Ain and A. Aziz, “A compact AES encryption core on Xilinx FPGA”, in Proc. of *2nd International Conference on Computer, Control and Communication*, pp.1-4, 2009.

BIOGRAPHY OF AUTHORS

Samir El Adib received the degree in Informatics, Electronics, Electrotechnics, and Automatics (IEEA) and M.S. degree in automatic and data processing from University Abdelmalek Essaadi (UAE), Tetuan, Morocco, in 2004 and 2006 respectively. Currently, he is a member of Remote-Sensing & Mobile-GIS Unit/Telecoms Innovation & Engineering Research group. His main research interests are FPGAs in custom-computing applications, and more concretely, applications of reconfigurable hardware to cryptography.

Naoufal Raissouni received the M.S., and Ph.D. degrees in physics from the University of Valencia, Spain, in 1997, and 1999, respectively. He has been a Professor of physics and remote sensing at the National Engineering School for Applied Sciences of the University Abdelmalek Essaadi (UAE) of Tetuan, since 2003. He is also heading the Innovation & Telecoms Engineering research group at the UAE, responsible of the Remote Sensing & Mobile GIS unit. His research interests include atmospheric correction in visible and infrared domains, the retrieval of emissivity and surface temperature from satellite image, huge remote sensing computations, Mobile GIS, Adhoc networks and the development of remote sensing methods for land cover dynamic monitoring.