International Journal of Information & Network Security (IJINS)
Vol.1, No.2, June 2012, pp. 110~118
ISSN: 2089-3299

AES Encryption Algorithm Hardware Implementation Architecture: Resource and Execution Time Optimization

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 Jun 6th, 2012
Revised Jun 20th, 2012
Accepted Jun 26th, 2012

Keyword:

Cryptography
AES/Rijndael
FPGA
BRAM
Digital Clock Manager
Look-up Tables

ABSTRACT

In the present paper we present an architecture to implement Advanced Encryption Standard (AES) Rijndael algorithm in reconfigurable hardware. Rijndael algorithm is the new AES adopted by the National Institute of Standards and Technology (NIST) to replace existing Data Encryption Standard (DES). Compared to software implementation, hardware implementation of Rijndael algorithm provides more physical security as well as higher speed. The first factor to be considered on implementing AES is the application. High-speed designs are not always desired solutions. In some applications, such as mobile computing and wireless communications, smaller throughput is demanded. Architecture presented uses memory modules (i.e., Dual-Port RAMs) of Field-Programmable Gate Array (FPGAs) for storing all the results of the fixed operations (i.e., Look-Up Table), and Digital Clock Manager (DCM) that we used effectively to optimize the execution time, reduce design area and facilitates implementation in FPGA. The architecture consumes only 326 slices plus 3 Block Random Access Memory (BRAMs). The throughput obtained was of 270 Mbits/s. The target hardware used in this paper is Spartan XC3S500E FPGA from Xilinx. Results are presented and compared with other reference implementations, as known from the technical literature. The presented architecture can be used in a wide range of embedded applications.

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 applications, including communication systems, wireless devices, and many other 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 (also in term of computation time), the update of the encryption system is a real necessity to guarantee the security of data.

In October 2000, National Institute of Standards and Technology (NIST) selected Rijndael [1] as the new Advanced Encryption Standard (AES) [2], in order to replace the old Data Encryption Standard (DES) [3] [4]. It offers a good ‘‘combination of security, performance, efficiency, implementability and flexibility’’ [5]. AES specifies a Federal Information Processing Standards (FIPS) approved cryptographic algorithm that is used to safely protect electronic data [6]. The selection process included performance evaluation on both software and hardware platforms and many hardware architectures were proposed. However, most of these architectures simply transcript the algorithm into hardware designs without relevant optimizations and tradeoffs. Moreover, the throughput and area

constraints considered are often unrealistic as shown by the recently published results. In this paper, we present an architecture for the AES Rijndael algorithm based on three techniques to improve the implementation of the AES Rijndael Algorithm:

- Look-Up Table to facilitate implementation.
- Digital Clock Manager (DCM) to optimize execution time.
- Memory modules (Dual-Port RAMs) to reduce design area.

The proposed architecture uses only a relatively small area and lower execution time and it can be used for a wide range of applications. However, most of publications on implementations of AES only provide performance and area figures without interfaces and registers.

2. DESCRIPTION OF AES RIJNDAEL ALGORITHM AND PREVIOUS WORK

2.1. Description of AES Rijndael Algorithm

The 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 output of the above transformations is called the 'State'. The state consists of the same byte length as each block of the message. The description of the four transformations of the Rijndael cipher and their inverses will be given below.

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

2.1.1. SubByte Transformation

The SubByte 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. 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:

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

2.1.2. InvSubByte Transformation

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

2.1.3. ShiftRow Transformation

In ShiftRow 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:

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

2.1.4. 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.

2.1.5. 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 transformation can be written in a matrix multiplication as follows:

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

2.1.6. InvMixColumn Transformation

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

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

2.1.7. AddRoundKey

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

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

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

RoundKeys 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:

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

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.

2.2. Previous work

There exist many presentations of hardware implementations of Rijndael AES algorithms in literature. In 2001, Elbirt et al., [7] compared five candidate algorithms (including Rijndael algorithm) for AES using Field-Programmable Gate Array (FPGA) implementations. Later FPGA implementations demonstrate better utilization of FPGA resources. Several architectures using dedicated on-chip memories implementing S-boxes and T-boxes were developed [8] [9] [10] [11] [12]. Recent research focused on fast pipelined implementations in both FPGA [13] [14] [15] [16] [17] [18]. Unfortunately, most of those implementations are too costly for practical applications.

3. EXISTING AES ARCHITECTURES

Various architectures exist to realize the AES encryption. Among them, the rolling architecture and the unrolling architecture (see Figures 1 and 2, respectively).

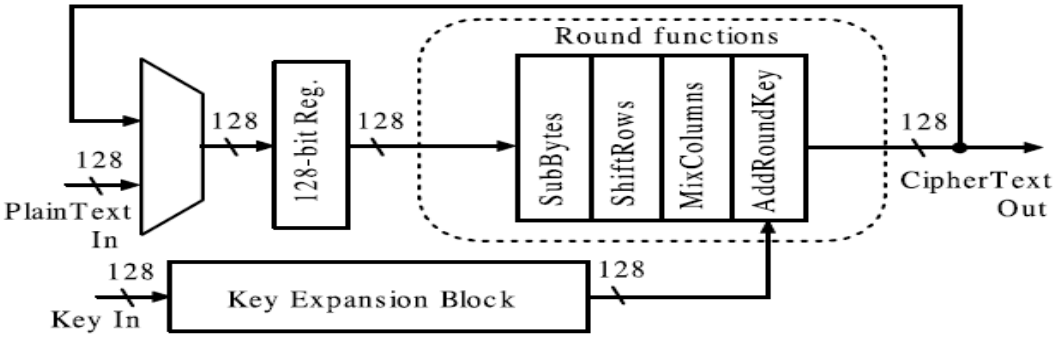

Figure 1. Rolling architecture of AES Encryption with 128 bit key

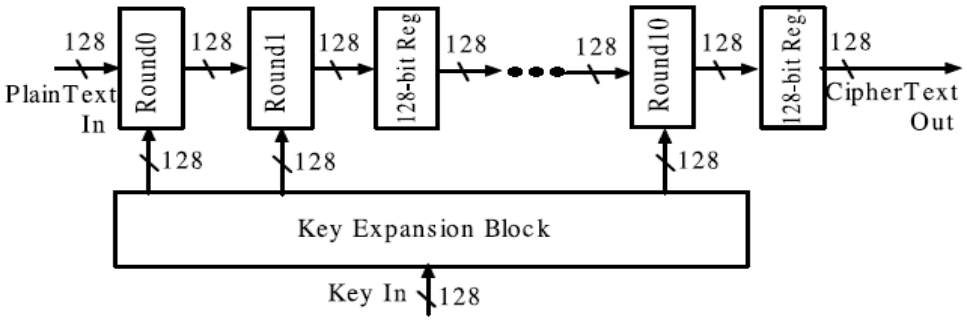

Figure 2. Unrolling architecture of AES Encryption with 128 bit key

The rolling architecture shown in Figure 1 uses a feedback structure where the data are iteratively transformed by the round function. This approach occupies small area, but achieves low throughput [19]. In the unrolling architecture shown in Figure 2 the rounds blocks are pipelined and the inserted pipeline registers allow simultaneous operation of all 11 round blocks. Due to the pipeline, this approach achieves a high throughput, but requires large area [20] [21].

4. PROPOSED AES HARDWARE ARCHITECTURE

The approach we propose allows the computation of the entire iteration of AES using look-ups tables and XOR operations. These precomputed look up tables represent the combined operation of the Subbytes and the Mixcolumn transformations. Compared to the 8x32 bits wide T-box look up tables, 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:

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

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

$State$ 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).

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

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

$$C = Log'\big((Log\ a) + (Log\ b)\big) \tag{11}$$

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:

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

Tables for Decryption:

$$ITo(a) = Log'\Big(\big(Log\ (0E)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$
$$IT1(a) = Log'\Big(\big(Log\ (09)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$
$$IT2(a) = Log'\Big(\big(Log\ (0D)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$
$$IT3(a) = Log'\Big(\big(Log\ (0B)\big) + \big(Log\ \big(SB(a)\big)\big)\Big)$$

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

$R_{15} = T_2(d_{15})\ xor\ T_3(d_{10})\ xor\ T_1(d_5)\ xor\ T_0(d_0)\ \ xor\ \mathrm{rk}_{15}$;
$R_{14} = T_0(d_{15})\ xor\ T_2(d_{10})\ xor\ T_3(d_5)\ xor\ T_1(d_0)\ \ xor\ \mathrm{rk}_{14}$;
$R_{13} = T_1(d_{15})\ xor\ T_0(d_{10})\ xor\ T_2(d_5)\ xor\ T_3(d_0)\ \ xor\ \mathrm{rk}_{13}$;
$R_{12} = T_3(d_{15})\ xor\ T_1(d_{10})\ xor\ T_0(d_5)\ xor\ T_2(d_0)\ \ xor\ \mathrm{rk}_{12}$;
$R_{11} = T_2(d_{11})\ xor\ T_3(d_6)\ xor\ T_1(d_1)\ xor\ T_0(d_{12})\ xor\ \mathrm{rk}_{11}$;
$R_{10} = T_0(d_{11})\ xor\ T_2(d_6)\ xor\ T_3(d_1)\ xor\ T_1(d_{12})\ xor\ \mathrm{rk}_{10}$;
$R_9 = T_1(d_{11})\ xor\ T_0(d_6)\ xor\ T_2(d_1)\ xor\ T_3(d_{12})\ \ xor\ \mathrm{rk}_9$;
$R_8 = T_3(d_{11})\ xor\ T_1(d_6)\ xor\ T_0(d_1)\ xor\ T_2(d_{12})\ \ xor\ \mathrm{rk}_8$;
$R_7 = T_2(d_7)\ xor\ T_3(d_2)\ xor\ T_1(d_{13})\ xor\ T_0(d_8)\ \ xor\ \mathrm{rk}_7$;
$R_6 = T_0(d_7)\ xor\ T_2(d_2)\ xor\ T_3(d_{13})\ xor\ T_1(d_8)\ \ xor\ \mathrm{rk}_6$;
$R_5 = T_1(d_7)\ xor\ T_0(d_2)\ xor\ T_2(d_{13})\ xor\ T_3(d_8)\ \ xor\ \mathrm{rk}_5$;
$R_4 = T_3(d_7)\ xor\ T_1(d_2)\ xor\ T_0(d_{13})\ xor\ T_2(d_8)\ \ xor\ \mathrm{rk}_4$;
$R_3 = T_2(d_3)\ xor\ T_3(d_{14})\ xor\ T_1(d_9)\ xor\ T_0(d_4)\ \ xor\ \mathrm{rk}_3$;
$R_2 = T_0(d_3)\ xor\ T_2(d_{14})\ xor\ T_3(d_9)\ xor\ T_1(d_4)\ \ xor\ \mathrm{rk}_2$;
$R_1 = T_1(d_3)\ xor\ T_0(d_{14})\ xor\ T_2(d_9)\ xor\ T_3(d_4)\ \ xor\ \mathrm{rk}_1$;
$R_0 = T_3(d_3)\ xor\ T_1(d_{14})\ xor\ T_0(d_9)\ xor\ T_2(d_4)\ xor\ \mathrm{rk}_0$;

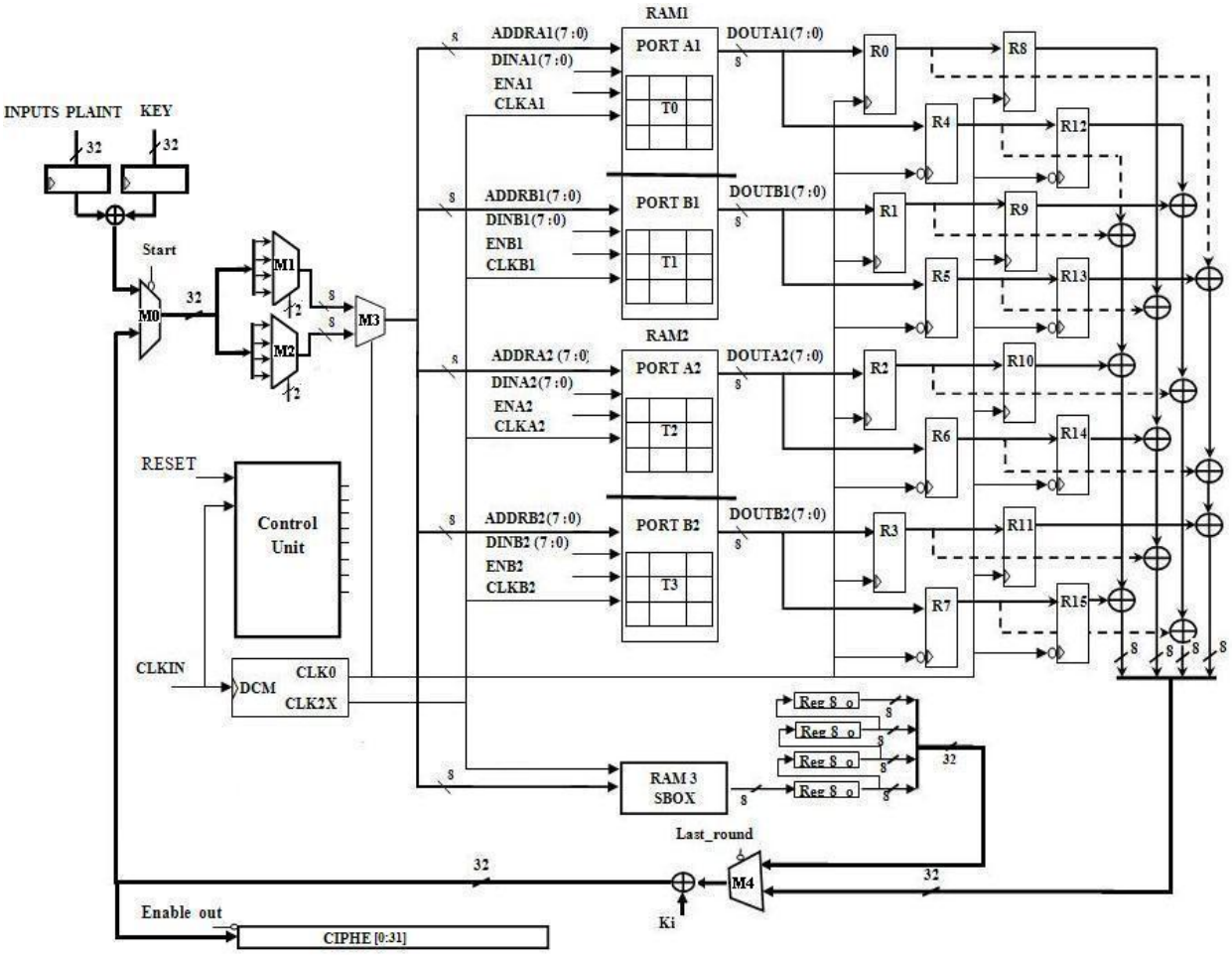

Figure 5. AES Encryption Architecture

The size of one Ti table is 2 Kbits for encryption. A Block Random Access Memory (BRAMs) has enough space to implement both tables. The architecture requires only 2 Block RAMs to implement of the combination of SubByte followed by MixColumn. This is implemented by using the existing Dual-Port RAMs by adding a doubled clock and some extra logic.

The detail of our proposed rolling architecture is shown in Figure 5. The BRAM is configured as Dual-Port ROM (Read Only Mode) to access the 8-bit lookup values corresponding to the 8-bit input addresses. Read

operation uses one clock edge and the data in the memory location selected by the addresses appear on the output ports after the BRAM access time.

DCM is used to generate two clocks; CLK0 (same frequency as that of input source clock) and CLK2X (double to the input source frequency) from the input clock source CLKIN. The CLK2X will enable the BRAM to output twice during one complete cycle of CLK0. At the input side three 2×1 multiplexers M1, M2 and M3 are being used to select the corresponding input data. The controlling signal for the multiplexer M3 is CLK0. 16 8-bit registers R0–R15 are being used to store the BRAM's output at both rising and falling edge of CLK0; R0–R3 and R8–R11 are positive edge triggered while R4–R7 and R12–R15 are negative edge triggered. All the 16 8-bit register's output are XORed to get the final output as shown in Figure 5.

Control unit based FSM (Finite State Machine), used to generate control signals of all multiplexers for the correct execution of ShiftRows transformation.

Key Generator: This block is responsible for creating the keys for each round as shown in Figure 6.

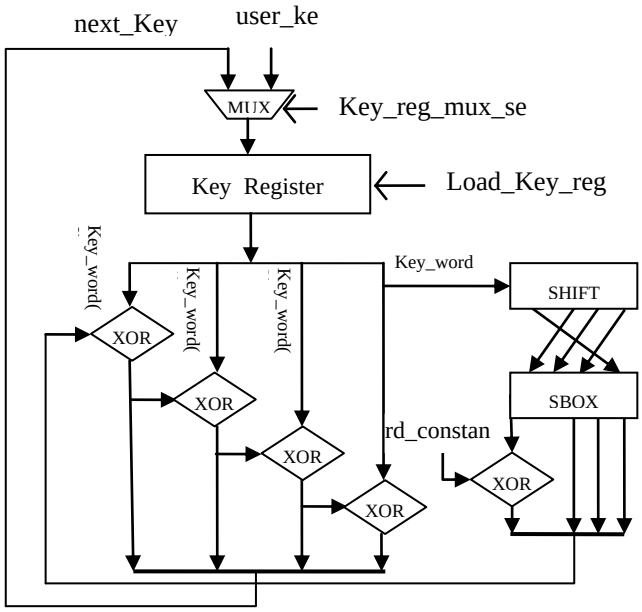

Figure 6. Block Diagram of Key_schedule Module

5. SYNTHESIS AND IMPLEMENTATION RESULTS

The architecture of AES using Look-Up Tables, Memory modules (Dual-Port RAMs) and Digital Clock Manager (DCM) is done using VHDL and implemented in a Xilinx Spartan3E -4 XC3S500E FPGA. Table 3 shows the synthesis and mapping results of AES design for encryption of 128- bit data with AES-128 key. It describes the selected target Xilinx FPGA device, encryption throughput achieved and the overall device utilization.

Table 3. Implementation Results

Target FPGA device	Spartan3E XC3S500E -4 FT256
Max. clock frequency	168.765MHz
Number of Slices	326
Block RAMS	3
Cycles	80
Encryption throughput	270 Mbps

6. PERFORMANCE AND COMPARISONS

The design was implemented on a Xilinx Spartan3E XC3S500E-4 FT256 FPGA devices. It occupies only 3 BRAMs and 326 Slices. Our design is operating at a clock frequency of 168.765 MHz which is still high enough for real time cryptographic applications and offers a throughput of 270 Mbps.

Table 4 details the comparison results with previous FPGA implementations using look up tables. The results clearly show that our proposed implementation achieves a good balance between hardware area and design performance.

Table 4. Performance comparisons in encryption

Author	Device	Slices	BRAMs	Throughput (Mbps)	Performance (Mbps/Slices)
Labbe et al. [24]	XCV1000-4	2151	4	390	0.18
Saggesse et al. [25]	XCVE2000-8	446	10	1000	2.3
Chodwiec et al. [26]	XC2530-5	222	3	139	0.62
Chodwiec et al. [26]	XC2530-6	222	3	166	0.74
Standaert et al.[27]	VIRTEX2300E	542	10	1450	2.6
Segreo et al. [28]	XCV-100-4	496	10	417	1.49
Segreo et al. [28]	XCV-600E-8	496	10	743	0.49
Saqib [23]	XCV812E	2744	0	258.5	0.09
Zambreno[22]	Virtex-2	1780	0	1000	0.56
Our	**XC3S500E -4**	**326**	**3**	**270**	**0.83**

7. CONCLUSION

The new architecture presented allows the implementation of the AES Rijndael Algorithm using an approach which includes modules memory and lookup tables. Specific applications for this circuit are in wireless LANs, cellular phones and smart cards. For instance, this circuit can be successfully used for wireless LAN in which the maximum throughput required is of 128 Mbit/s, lower than the throughput obtained from measurements. Our implementation achieves a throughput of 270 Mbps and uses a total of 326 slices of a Spartan3E FPGA. The results clearly show that our proposed implementation achieves a good balance between hardware area and design performance compared with other researchers.

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] M. McLoone, J.V. McCanny, "Rijndael FPGA implementations utilising look-up tables," *The Journal of VLSI Signal Processing* 34 (3) (2003) 261–275.
[7] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists, "*IEEE Trans*. VLSI Syst. 9 (4) (2001) 545–557.
[8] P. Chodowiec, K. Gaj, P. Bellows, B. Schott, "Experimental Testing of the Gigabit IPSec-Compliant Implementations of Rijndael and Triple DES Using SLAAC-1V FPGA Accelerator Board," *Information Security Conference (ISC 2001),* Malaga, Spain, 2001.
[9] V. Fischer and M. Drutarovsk´y, "Two Methods of Rijndael Implementation in Reconfigurable Hardware," *Cryptographic Hardware and Embedded Systems (CHES 2001),* Paris, France, 2001.
[10] M. McLoone and J.V. McCanny, "High Performance Single-Chip FPGA Rijndael Algorithm Implementations," *Cryptographic Hardware and Embedded Systems (CHES 2001),* Paris, France, 2001.
[11] M. McLoone and J.V. McCanny, "Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm," *Field-Programmable Logic and Applications (FPL 2001),* Belfast, Northern Ireland, UK, 2001.
[12] M. McLoone and J.V. McCanny, "Rijndael FPGA implementation utilizing look-up tables," *IEEE Workshop on Signal Processing Systems*, 2001.
[13] F. Standaert, G. Rouvroy, J. Quisquater, J. Legat, "Efficient implementation of Rijndael encryption in reconfigurable hardware: improvements and design tradeoffs, " *In: Proceedings of the CHES 2003*, LNCS 2779; 2003. p. 334–50.
[14] GP. Saggese, A. Mazzeo, N. Mazzocca, AGM. Strollo, "An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm, " *In: Proceedings of the FPL 2003*, LNCS 2778; 2003. p. 292–302.
[15] A. Hodjat, I. Verbauwhede, " A 21.54 Gbits/s fully pipelined AES processor on FPGA, " In: *Proceedings of the 12th IEEE symposium on field-programmable custom computing machines*; 2005. p. 308–9.

[16] X. Zhang, K. Parhi, "High-speed VLSI architectures for the AES algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2004; 12:957–67.
[17] J.M. Granado-Criado, A. Vega-Rodriguez Miguel, M. Sanchez Perz Juan, A. Gomez- Pulido Juan, "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration," *Department Technologies of Computers and Communications*, University of Extremadura, Spain; 2009.
[18] S. M. Yoo, D. Kotturi, D.W. Pan, J. Blizzard, "An AES crypto chip using a high-speed parallel pipelined architecture," *Microprocessors and Microsystems* 2005;29:317–26.
[19] Amphion Semiconductor, " CS5210-40: High performance AES encryption cores, " 2003, available at http://www.amphion.com/cs5210.
[20] G.P. Saggese, A. Mazzeo, N. Mazzoca, and A.G.M. Strollo, "An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES aogorithme, " *FPL 2003*, LNCS 2778, pp.292-302,2003.
[21] F.X. Standaert, G. Rouvroy, J.J. Quisquater, and J.D. Legat, " Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs, " *Proc. CHES 2003*, LN CS2523, pp.334-350, cologne, Germany, Springer-Verlag, Sept. 2003.
[22] J. Zambreno, D. Nguyen, and A. Choudhary, "Exploring area/delay tradeoffs in an AES FPGA implementation," *FPL 2004*, LNCS3203, pp.575-585, 2004.
[23] F. Rodriguez-Henriquez, N. A. Saqib and A. D. Diaz-Perez, "4.2 Gbit/s Single-Chip FPGA Implementation of AES Algorithm, " In *IEE Electronic Letters*, volume 39 (15), pages 1115–1116, July 2003.
[24] A. Labbe, Annie Perez, "AES Implementation on FPGA: Time and Flexibility Tradeoff, " in *Proceedings of FPL*, pp. 836-844, 2002.
[25] 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, " In *Lecture Notes in Computer Science*, volume 2778, pages 292–302, January 2003.
[26] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 319–333, 2003.
[27] F. X. Standaert, G. Rouvroy, J.J. Quisquart and J.D. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 334–350, 2003.
[28] A. Segredo, E. Zabala and G. Bello, "Diseno de un procesador criptografico Rijndael en FPGA," In *X Workshop IBERCHIP*, pages 64–65, 2004.

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.