FPGA IMPLEMENTATION OF THE AES ENCRYPTION AND DECRYPTION ALGORITHMS

Hamid BESSALAH, Fadila MOSTEFAI, Zahia BRAHIMI,

Centre de Développement des Technologies Avancées, BP 45, lotissement 20 Août 1956, Baba Hassen,- Alger, ALGERIE
bessalah@cdta.dz, f.mostefai@cdta.dz, zbrahimi@cdta.dz

ABSTRACT

In this paper, architecture for hardware implementation of the Advanced Encryption Standard (AES) Algorithm is presented. Where, encryption, decryption and key schedule are all implemented using small resources of only 3383 Slices and 8 Block RAMs. So our implementation fits easily in a Xilinx VirtexII XC2V2000-4FF896 FPGA. The proposed implementation can encrypt and decrypt data streams with a throughput of 235 Mbps, and a new way of implementing MixColumns and InvMixColumns transformations using shared logic resources is presented.

Keywords: *AES, Decryption, Encryption, FPGA, ECB,Images*

1. INTRODUCTION

Since the adoption of the Rijndael algorithm as the new Advanced Encryption Standard (AES) by the National Institute of Standards and Technology (NIST) in 2001 [1], numerous FPGA and implementations and evaluations of the AES were presented in literature [2-9].

The AES algorithm can use cryptographic keys of 128, 192, and 256 bits. In this work we implement the 128 bits standard on a Field-Programmable Gate Array (FPGA) using the VHDL as a hardware description language.

The main aim of this work is to produce a low area and fast clock speed FPGA device. This is done by the use of a mixed processing of 32 bits and 128 bits; so the result is a faster algorithm with more cycles.

The proposed architecture is an alternative of that presented in [10], and in order to implement encryption/decryption process in the same circuit, we took as a starting point this implementation.

This paper is organized as follows; a review of the basic structure of the AES is given in section II. Section III is dedicated to the description of our system architecture and its specifications. Section IV is dedicated to performances evaluation of our design and finally, in section V we conclude our paper and highlight the future work.

2. THE AES ALGORITHM

The AES algorithm consists mainly of a symmetric block cipher that can process data blocks of 128, 192 or 256 bits by using key lengths of 128, 196 and 256 bits. The algorithm is based on the round function, and different combinations of the algorithm are structured by repeating this round function different times. Each round function contains uniform and parallel 4 steps, Byte Substitution, Row Shifting, Column Mixing and Key Addition, and each step has its own particular functionality.

A full description of the AES is detailed in the Rijndael proposal [11]

3. THE PROPOSED ARCHITECTURE

The architecture of this implementation is based on the paper described by P. Chodowiec [12], the schematic diagram of circuit implementation of the encryption and decryption is depicted in Figure 1 and Figure 2:

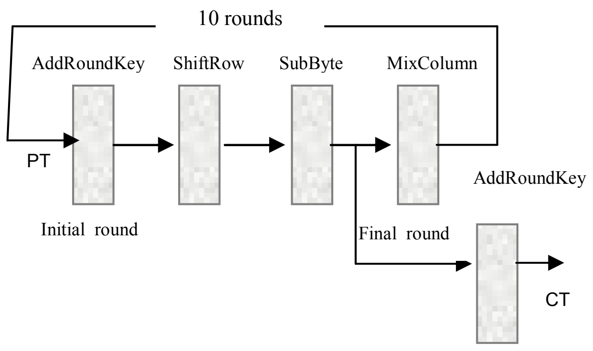

Figure 1. Encryption schematic

Circuit schematic

The choice of the implementation was based on the following criteria:

- The module of encryption and decryption in the same circuit
- A circuit with minimum possible FPGA resources
- Considerable Throughput

The solution adopted to implement the two modules of encryption and decryption in the same circuit is to reduce iterative architecture.

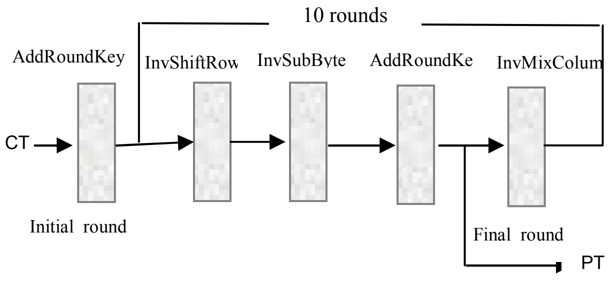

Figure 2. Decryption schematic

Then, we implement ¼ of iteration, and calculate the iteration on 4 cycles of clock instead of only one as illustrated in Figure 3.

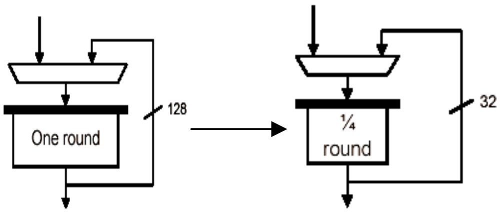

Figure 3. Iteration calculation

We began the design of the compact architecture by analyzing the basic architecture, as introduced in [12]. The basic architecture unrolls only one full cipher round, and iteratively loops data through this round until the entire encryption or decryption transformation is completed. Only one block of data is processed at a time. The structure of the AES round for encryption is shown in Figure 4.

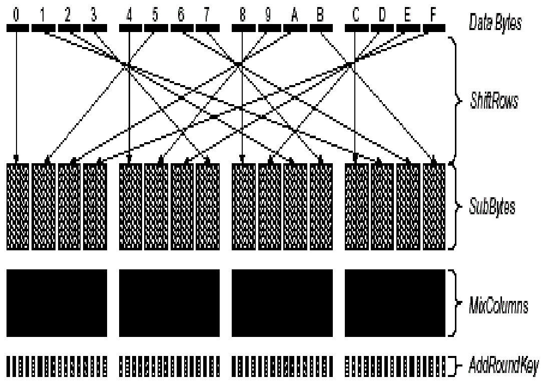

Figure 4. AES operations

The decryption round looks very similar, except it employs inverted operations in the following order: InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns. The SubBytes and ShiftRows operations in Figure 5 are reordered compared to the cipher round depicted in the standard.

Their order is not significant because SubBytes operates on single bytes, and ShiftRows reorders bytes without altering them. This feature was used in our implementation. The round is composed of sixteen 8 bits S-boxes computing SubBytes, and four 32-bit MixColumns operations, working independent of each other. The only operation that spans throughout the entire 128-bit block is ShiftRows.

In order to create a folded architecture with better parameters, the bytes of data were arranged in columns as shown in the Figure 5.

The execution is done in the following steps:

- Read input bytes: 0, 5, A, F; execute SubBytes, MixColumns and AddRound- Key on them; write results to the output at locations: 0, 1, 2, 3.

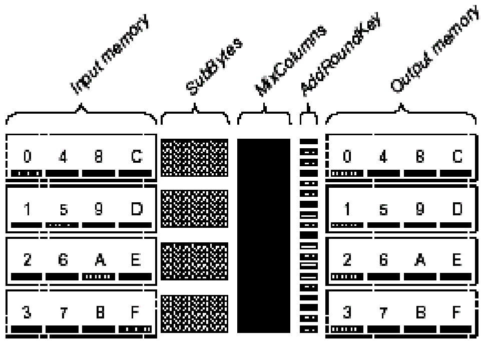

Figure 5. AES folded architecture

- Repeat above operations for input bytes: 4, 9, E, 3; write results at output locations: 4, 5, 6, 7.
- Repeat above operations for bytes: 8, D, 2, 7; write results at locations: 8, 9, A, B.
- Repeat above operations for bytes: C, 1, 6, B; write results at locations: C, D, E, F. Output now becomes input for the next step.

In these four steps the entire AES round was executed including ShiftRows operation. At each step only one byte was read from each input row and one byte was written to each output row.

A similar exercise with identical conclusions can be executed for decryption transformation. Each row can be viewed as an addressable 8-bits wide memory. The correct execution of ShiftRows and InvShiftRows is now resolved to the proper addressing of each of the memories at the consecutive clock cycles. At the fourth clock cycle output memories become input memories and vice versa.

3.2. SHIFT REGISTER IMPLEMENTATION

Bytes from the output of AddRoundKey are written into consecutive locations in the output memory in consecutive clock cycles. Therefore, we could use a simple shift-register to shift computed data in without generating any addresses. Fortunately, LUTs can also be configured as 16-bit shift registers with variable taps as shown in Figure 6.

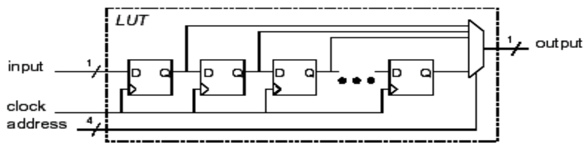

Figure 6. Look-Up Table (LUT) configured as a shift register

3.3. SUBBYTES AND INVSUBBYTES IMPLEMENTATION

A simple implementation of SubBytes or InvSubBytes requires a 256x8 read only memory ROM. A block RAM has enough space to implement SubBytes and InvSubBytes as shown in figure 7.. Each bus has access to the whole memory capacity, and can carry out a transformation of SubBytes or InvSubBytes independently one of each other. Folded architecture requires only 4K bytes block RAM to implement four operations of InvSubBytes or SubBytes. The RAM is an entirely synchronous memory.

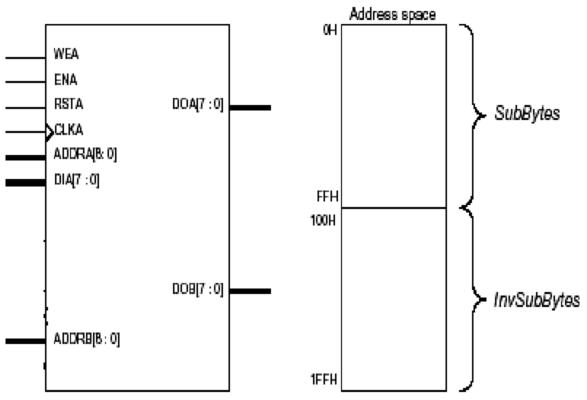

Figure 7. Block RAM for SubBytes et InvSubBytes.

3.4. MIXCOLUMNS AND INVMIXCOLUMNS IMPLEMENTATION

The 32-bit input to the MixColumns transformation is represented as a polynomial of the form a(x) = $a_3x^3+a_2x^2+a_1x+a_0$, with coefficients in the Galois field **(2^8).** The coefficients of a(x) are also polynomials of the form b(x) = $b_7x^7 +b_6x^6 +b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$, with their own coefficients in Galois field (2^8).

The MixColumns multiplies the input polynomial by a constant polynomial

$$c(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\} \quad (1)$$

modulo $x^4 +1$. The coefficients in **GF(8)** are multiplied modulo $x^8 +x^4 +x^3 +x+1$. The InvMixColumns multiplies the input polynomial by another constant polynomial:

$$d(x)=c^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\} \quad (2)$$

The implementation of the MixColumns is very simple because the coefficients of c(x) are small. On the other hand, the InvMixColumns is far more complex and occupies larger area.

P. Chodowiec [12] proposed an implementation based on the following idea:

$$d(x) = c(x) + e(x) + f(x) \quad (3)$$

where

$$e(x) = \{08\}x^3 +\{08\}x^2 +\{08\}x+\{08\} \quad (4)$$

and

$$f(x) = \{04\}x^2 + \{04\} \quad (5)$$

This implementation yields logic optimizations since InvMixColumns shares logic resources with MixColumns. Our implementation is derived as follows:

$$c(x).\bullet d(x) = \{01\} \quad (6)$$

If we multiply both sides of equation (6) by *d(x)* we obtain:

$$c(x).\bullet d^2(x) = d(x) \quad (7)$$

where

$$d^2(x) = \{04\}x^2 + \{05\} \quad (8)$$

The MixColumns and InvMixColumns can be implemented using shared logic resources as shown in Figure8. They are implemented in 4 LUTs input of the FPGA.

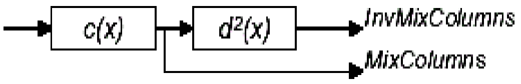

Figure 8. Implementation of MixColumns and InvMixColumns

3.5. ENCRYPTION/DECRYPTION UNIT

The AES encryption and decryption rounds substantially differ from the point of view of hardware implementations. One of the inconveniences arises from the fact that the AddRoundKey is executed after MixColumns in the case of encryption and before InvMixColumns in the case of decryption. Therefore, a switching logic is required to select appropriate data paths, which affects the performance, as shown in Figure 9.

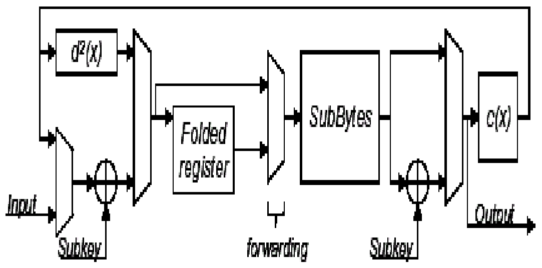

Figure 9. Implementation of the encryption/decryption unit

3.6. IMPLEMENTATION OF THE KEY SCHEDULE

Our AES implementation is designed to perform both encryption and decryption. Since we did not see any advantage in computing round keys on-the-fly, we choose to implement the key schedule that precomputes all round keys. The implementation of the key schedule is shown in Figure11. It computes 32-bits of the key material per clock cycle; therefore, full key schedule execution takes 44 clock cycles. The computed round keys are stored in a single Block RAM.

4. SYNTHESIS AND IMPLEMENTATION RESULTS

We implemented our solution on Xilinx Virtex XC2V2000-4ff896. Simulation is done by ModelSim 5.7c to verify the correctness of the design. The encryption and decryption unit has been verified using 128 test vectors verification provided by the NIST.

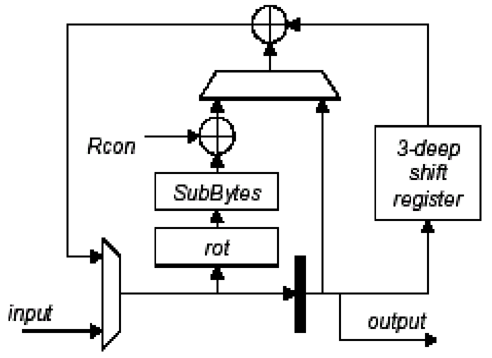

Figure 11. Implementation of the key schedule

This design has been synthesized using ISE Xilinx 6.3i, the summary of the area utilization in FPGA Xilinx is shown in table1.

Table 1. Synthesis results

	Throughput (Mbps)	BRAMs	Slices	4 input LUTs
AES Encrypt / Décrypt	235	8	3383	4157
AES Encrypt only [12]	443.33	0	2345	4601

5. CONCLUSION

In this paper, the feasibility of creating a very compact FPGA implementation of the AES was examined. The proposed folded architecture achieves good performance and occupies less area. This compact design was developed by the examination of each of the components of the AES algorithm using ECB mode and matching them into the architecture of the FPGA.

The demonstrated implementation fits in Xilinx XC2V2000-4FF896 FPGA. Only 34% of the logic resources available in this device were utilized. This implementation can encrypt and decrypt data streams up to 235 Mbps.

The Architecture has been tested and validated on the NIST vectors tests Benchmarks. However, in future work, we aim to test and validate our architecture on medical and satellite images. And adapt our design, in order to fit it in Xilinx Virtex II XC2V1000-4FG456C of the Memec Virtex II System Board .

6. REFERENCES

[1] National Institute of Standards and Technology: *FIPS 197: Advanced Encryption Standard,* November 2001

[2] A. J. Elbirt and C. Paar, "*An FPGA Implementation and Performance Evaluation of the Serpent Block Cipher*", Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA 2000, pp. 33-40, February 2000.

[3] Henry Kuo , Ingrid Verbauwhede, ”*Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rjindael Algorithm”*, in 3rd international workshop cryptographic Hardware and embedded systems (CHES 2001), LNCS2162, Paris, May 2001,pp 51-64.

[4] Ting Liu, Camel Tanougast, Philippe Brunet, Yves Berviller, Hassan Rabah et Serge Weber "Implantation FPGA optimisée de l’algorithme AES pour applications embarquées", Journées Francophones sur l'Adéquation Algorithme Architecture (JFAAA’05), *IEEE Signal Processing Society, IEEE CAS society*, 18-21 janvier 2005 à Dijon, France.

[5] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists", *Proceedings of the Third Advanced Encryption Conference*, pp. 13-27, April 2000

[6] X. Zhang and K. K. Parhi, “High-speed VLSI Architectures for the AES Algorithm,” *IEEE Trans. on VLSI Systems*, vol. 12(9), pp. 957-967, Sep. 2004.

[7] A. Hodjat, I. Verbauwhede, “Speed-Area Trade-off for 10 to 100 Gbits/s Throughput AES Processor’, *37th Asilomar Conference on Signals, Systems, and Computers,* November 2003.

[8] A. Hodjat, I. Verbauwhede,” Minimum area cost for a 30 to 70 Gbits/s AES processor” Dept. of Electr. Eng., California Univ., Los Angeles, CA, USA; 19-20 Feb. 2004.

[9] N.Sklavos, O.Koufopavlou “Architectures and VLSI Implementations of the AES-Proposal Rijndael**”,***IEEE Computer Society* Washington, DC, USA December 2002.

[10] H.Bessalah,,F. Mostefai, Z. Brahimi, *“ FPGA Implementation of the AES Algorithm Encryption* ”, *International Computer Systems & Information Technology conference ICSIT’05*, Algiers ALGERIA, July 19-21, 2005.

[11] J. Daemen and V. Rijmen, AES Proposal: Rijndael (Version 2).NIST AES Website; http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndaelammended.pdf

[12] Pawel Chodowiec and Kris Gaj, “Very Compact FPGA Implementation of the AES Algorithm” , *CHES 2003*, LNCS 2779, pp. 319-333