Architectures and VLSI Implementations of the AES-Proposal Rijndael

N. Sklavos and O. Koufopavlou, Member, IEEE

Abstract—Two architectures and VLSI implementations of the AES Proposal, Rijndael, are presented in this paper. These alternative architectures are operated both for encryption and decryption process. They reduce the required hardware resources and achieve high-speed performance. Their design philosophy is completely different. The first uses feedback logic and reaches a throughput value equal to 259 Mbit/sec. It performs efficiently in applications with low covered area resources. The second architecture is optimized for high-speed performance using pipelined technique. Its throughput can reach 3.65 Gbit/sec.

Index Terms—AES, Rijndael, secret key ciphers, security, pipelining architectures.



IN our days, the need for secure transport protocols seems to be one of the most important issues in the communication standards. Of course, many encryption algorithms support the defense of private communications. However, the implementation of these algorithms is a complicated and difficult process and sometimes results in intolerant performance and allocated resources in hardware terms. The explanation for this fact is because these encryption algorithms were designed some years ago and for general cryptography reasons. In recent years, new flexible algorithms specially designed for the new protocols and applications have been introduced to face the increasing demand for cryptography.

In October of 2000, the National Institute of Standards and Technology (NIST) announced the cipher Rijndael as the Advanced Encryption Standard (AES) [1] in order to replace the aging Data Encryption Standard (DES) [2]. The new algorithm is expected to be a standard by the summer of 2001 [3].

In the Third Advanced Encryption Standard (AES) Candidate Conference [4], papers from different research groups were presented [5], [6], [7], [8], [9]. The main purpose of these works was the evaluation of the AES finalist algorithms in terms of hardware implementation performance. In order to achieve this, all the authors used general purpose architectures and not specialized designs for each algorithm implementation. This is a fair methodology for comparison of different algorithms. On the other hand, this way is not well-suited to the implementation of each algorithm separately. In addition, in two of these works ([5], [6]), only the encryption mode of operation was implemented and not the decryption. References [6], [7], and [9] do not support the onchip-generation of the necessary for the algorithm encryption/ decryption keys. In other words, the proposed designs do not support the completed operation of the algorithms and perform inefficiently in terms of both the encryption and decryption mode of data transformation.

Especially for the Rijndael algorithm, other works [10], [11], [12] have been published. The proposed work in [10] is an uncompleted implementation of the algorithm's total operation. It supports only the encryption process. In [12], two different designs

are introduced, one for encryption and one for decryption. They have been implemented in two separate FPGA devices. This is not the right way for the implementation of a block cipher. It is not efficient for the implementation of communications protocols, especially in integrated circuits with low allocation resource specifications. The proposed implementation in [12] needs two different FPGA devices in order to ensure the complete operation of the algorithm.

In this paper, two architectures and VLSI implementations of the AES proposal are presented. These alternative designs operate both for encryption and decryption process in the same device. They are proposed in order to reduce the required hardware resources and to achieve high-speed performance. In the first design, the appropriate key expansion unit is integrated with the encryption/decryption core.

The paper is organized as follows: In Section 2, the cipher Rijndael is described. In Section 3, the two different architectures are presented in detail. Performance analysis and comparison results with other works are reported in Section 4. Finally, concluding remarks are made in Section 5.

2 THE RIJNDAEL ENCRYPTION/DECRYPTION ALGORITHM

A new block encryption algorithm called Rijndael has been developed and published by Daemen and Rijmen [13]. This algorithm is an iterated block cipher with variable block length and a variable key length. The block and the key length can be independently specified to 128, 192, or 256 bits. The number of algorithm rounds depends on the block and key length.

The different transformations of the algorithm architecture operate on the intermediate result, called State. The State can be pictured as a rectangular array of bytes. This array has four rows. The number of columns is called Nb and it is equal to block length divided by 32. The Key is also considered as a rectangular array with the same number of rows as State. The number of columns is equal to the key length divided by 32. This number is denoted as Nk. The number of rounds, Nr, depends on the values Nb and Nk. For block and key length equal to 128 bits, both values of Nb and Nk are equal to four and the number of rounds Nr is defined as 10. These specifications are served by the proposed implementations, which will be analyzed in detail in the next paragraphs.

A basic round transformation relies on combining operations from four fundamental algebraic functions that operate on arrays of bytes. These transformations are:

- **SubBytes:** Operates in each byte of the State independently. This mathematical substitution is constructed of the compositions of two transformations: multiplicative inverse in GF(2⁸) and an affine mapping over GF(2) transformation. The decryption needs the multiplicative inverse in GF(2⁸), too, and the inverse of the affine mapping transformation over GF(2).
- **ShiftRow:** Cyclically shifts the rows of the State over different offsets. The operation is almost the same in the decryption process except for the fact that the shifting offsets have different values.
- **MixColumn:** In this transformation, the columns of the State are considered as polynomials over $GF(2^8)$ and are multiplied with a fixed polynomial $c(x) = '03'x^3 + '01'x^2 + '01'x + '02'$ for encryption and with the polynomial $d(x) = '0B'x^3 + '0D'x^2 + '09'x + '0E'$ for the decryption process. Both polynomial multiplications are modulo $(x^4 + 1)$.

The authors are with the Electrical and Computer Engineering Department, University of Patras, Patras, Greece.
 E-mail: nsklavos@ee.upatras.gr.

Manuscript received 30 July 2001; revised 12 Nov. 2001; accepted 25 Apr. 2002.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114619.

мих

Multiplexer Block

XOB Block

128 bit

Affine Mapping

128 bit

128 bit

Control Port

128 bi

128 bit

MixCol

Select

Round Key

128 bit

Input

Data

SUBBYTE

Affine Affine Mappin

DATA SHIFT

MIX COLUMN UNIT

Basic

KEY ADDITION

Transformation

Data



1.

inverse of each byte of the State in the finite field $GF(2^8)$ and b) an affine mapping transformation over GF(2). The most well-known VLSI architectures for the multiplicative inverse in $GF(2^m)$ use arrays of basic inversion block cells ([14], [15], [16]). This method has time and area requirements with a complexity which varies from $O(m^2)$ ([6]) to $O(m^4)$ ([13], [14]). In terms of execution time, such architectures need a number of cycles per inversion with range values between m ([15], [16]) and 3m + 2 ([16]) in order to achieve multiplicative inverse in $GF(2^m)$. These values are unacceptable for a high-speed implementation of a cryptographic algorithm.

In order to overcome the above performance bottleneck, it is proposed that the integration of the multiplicative inverse be produced by the use of an LUT (look up table). In this way, each byte of the State is replaced with its reciprocal in the same $GF(2^8)$. The use of the LUT needs one time step and this execution time is significantly less than the other proposed implementations in [14], [15], [16]. Although the covered area of the LUT is greater than the conventional architectures ([14], [15], [16]), this is not a problem for the current FPGA technology.

The affine mapping transformation for the algorithm encryption process is defined as:

$$\begin{aligned} & \text{Out}[i] = \text{In}[i] \text{ XOR In}[(i+4) \mod 8] \text{ XOR In}[(i+5) \mod 8] \\ & \text{ XOR In}[(i+6) \mod 8] \text{ XOR In}[(i+7) \mod 8] \\ & \text{ XOR C}(i), \end{aligned} \tag{1}$$

where In[i] is the ith bit of the input byte and C(i) is the ith bit of a byte constant C with the value $C = \{01100011\}$, as the algorithm specifications defines.

The inverse affine mapping transformation for the decryption process is defined by:

$$\begin{aligned} \operatorname{Out}[i] &= \operatorname{In}[(i+2) \bmod 8] \text{ XOR } \operatorname{In}[(i+5) \bmod 8] \\ & \operatorname{XOR } \operatorname{In}[(i+7) \bmod 8] \text{ XOR } \operatorname{C}(i). \end{aligned} \tag{2}$$

In this case, constant C has the value $C = \{00000101\}$.

It's important to mention that, in the encryption process, the SubByte transformation takes the multiplicative inverse over $GF(2^8)$ of a byte first and then the affine mapping process is applied. In the decryption process, that is also called inverse cipher, the inverse of the affine mapping is followed by taking the multiplicative inverse in GF(2). In order to achieve the implementation of these two different operating modes in the SubByte component, a multiplexer block (Fig. 1) has been added.

- 2. *ShiftRow:* The ShiftRow operation is achieved with simple use of four multiplexers.
- 3. MixColumn: This operation is applied over the State column. Every column S of the State consists of four bytes S = {S0, S1, S2, S3}. In both encryption and decryption, the state column is multiplied by a different specified polynomial, as is described in Section 2, and, finally, a transformed column T = {T0, T1, T2, T3} is generated. The MixColumn component does not operate in the last round of the algorithm. An appropriate select signal determinates



 KeyAddition: In this operation, the round key is applied to the State by simple bit by bit XOR. KeyAddition is the same for the decryption process.

Before the first round, a key addition layer is applied to the cipher data. This transformation is stated as the algorithm initial round key addition. The final round of the cipher is equal to the basic round with the MixColumn step removed. A key expansion unit is defined in order to generate the appropriate key, for every round, from the initial key value. When all rounds of transformation are completed, a cipher data block with the same length as the plain data has been generated.

The decryption process has the same structure as the encryption architecture. The only main difference is that for every function that is used in the basic round, the mathematical inverse of it is taken. The key expansion unit performs almost the same operation with the encryption process. The only difference is that the decryption of the round keys is obtained by applying the inverse MixColumn to the corresponding round keys. The initial value of the key for the decryption operation is changed. The appropriate basic decryption key must be loaded in the key buffer before the decryption beginning (for more details, see [13]).

3 HARDWARE ARCHITECTURES AND VLSI IMPLEMENTATIONS

Two alternative architectures are proposed for the Rijndael algorithm in order to reduce the required hardware resources and to achieve high-speed performance. Both architectures serve the encryption and decryption process in the same hardware device.

3.1 Basic Block Round

The architecture of the basic block round is shown in Fig. 1. As was already mentioned in the previous section, each basic round of the algorithm is composed of basic building blocks: SubBytes, ShiftRow, MixColumn, and KeyAddition. The structure of SubBytes and MixColumn turned out to be challenging.



Fig. 2. Architecture with implemented key expansion unit (AKE).

when the input data would be transformed. In the case of the last round, the appropriate value of the MixCol Select signal forces the input to the output of the component with no data transformation.

In order to achieve the appropriate polynomial multiplications, in the proposed hardware implementation of the MixColumn, two basic components were designed. The first component is named ControlPort. It accepts as input the four bytes of the transformed State column, {In0, In1, In2, In3}, and provides as output two bytes, Y and Z. In the encryption process, Y and Z are defined as:

$$Y = In0 XOR In1 XOR In2 XOR In3,$$
(3)

$$Z = Y.$$
 (4)

In the decryption process, Y and Z are defined as:

$$\Gamma 0 = \text{In0 XOR In1 XOR In2 XOR In3}, \tag{5}$$

T1 = T0 XOR [In2Trans (In2Trans(T0)],(6)

 $Y = T1 \text{ XOR [In2Trans (In2Trans(In0 \text{ XOR In2}))]}, (7)$

$$Z = T1 \text{ XOR [In2Trans (In1 XOR In3))]}, \quad (8)$$

where In2Trans(K) is the multiplication of the byte K by X (hexadecimal "02") over $GF(2^8)$. The second component of the MixColumn, called BasicPort, accepts as input the In0, In1, In2, In3, Y, and Z and provides as output the Out0,

Out1, Out2, and Out3. The appropriate operation of BasicPort architecture is described as:

 $Out0 = In0 XOR [Y XOR In2Trans (In0 XOR In1)], \quad (9)$

Out1 = In1 XOR [Z XOR In2Trans (In1 XOR In2)], (10)

Out2 = In2 XOR [Y XOR In2Trans (In2 XOR In3)], (11)

Out3 = In3 XOR [Z XOR In2 Trans (In3 XOR In0)]. (12)

In hardware, this can be implemented in byte level as a shift left of one bit and a subsequent conditional bitwise XOR with "1B." In a dedicated hardware, this operation needs four 2-input XORs. The input bytes In0, In1, In2, and In3 are the same four bytes of the transformed State column, while input Y and Z are provided by the ControlPort component.

4. *KeyAddition:* The KeyAddition component consists of eight 2-input XORs for every byte of the State column. Every bit of the round key is XORed with the appropriate bit of the transformed data byte.

The two alternative proposed architectures use the same basic block round architecture.

3.2 First Architecture with Implemented Key Expansion Unit (AKE)

The first proposed Architecture (AKE) is shown in detail in Fig. 2. This architecture performs both the encryption and the decryption process, with input plaintext and key vector equal to 128 bit. The



Fig. 3. Architecture using RAM for key loading (ARKL).

algorithm specifies 10 rounds for the State transformation and an extra initial round key addition.

A key buffer of 128-bit width is used for the key storage. In the initial round key addition transformation, the input state is XORed with the input key. In the first step, the initial round key addition is executed and the key for the first round is calculated. In a clock cycle, one transformation round is executed and, at the same time, the appropriate key for the next round is calculated. The whole process reaches the end when 10 rounds of transformation are completed. The Input Register is used to keep the transformed State after every round of operation. The State is forced to this register with the use of a feedback technique. The Basic Block Round architecture is shown in Fig. 1 and has been described in detail in Section 3.1.

The Key Expansion Unit for the AKE architecture is illustrated in Fig. 2. The round keys are derived from the initial key. Two are the basic component of this unit, the Key Transformation and the Round Key selection. The total number of the round key bits is equal to the block length, multiplied by the number of rounds plus one. The proposed implementation with 128 bit block length and 10 rounds generates 10 128 bit round keys. The round keys are taken from the initial key in a complicated way, defined in detail in the algorithm specifications [13].

The algorithm demands a different operation mode of the key expansion unit, between encryption and decryption processes. The basic difference is that, in decryption, the round keys are obtained by applying the inverse MixColumn to the corresponding round keys. The total execution time is one clock cycle for every round, plus one clock cycle for the initial round key addition. So, the system needs 11 clock cycles in order to completely transform a 128 bit data block.

3.3 Second Architecture Using RAM for Key Storage (ARKL)

The second proposed architecture is shown in Fig. 3. The main characteristics of this are: 1) the pipelining used technique and 2) the usage of a RAM for the key storage and loading. It is not possible to apply pipelining in many cryptographic applications. However, the Rijndael cryptographic algorithm internal architecture provides the possibility of being implemented with pipelining technique.

The pipelining architecture offers the benefit of high-speed performance. The implementation can be applied in applications with hard throughput needs. This goal is achieved by using a number of operating blocks with a final cost to the covered area. The proposed architecture uses 10 basic round blocks, which are cascaded by using pipeline registers.

In this architecture, 10 blocks of data can be transformed at the same time. The main disadvantage of the second proposed design is the increased required effective area. In order to face this problem, RAM was used for the key storage. Many FPGAs provide embedded RAM, which may be used to replace the Key Expansion Unit and the internal buffer of the AKE architecture for the initial key. In this way, the appropriate key for each round can be addressed from the RAM. External RAM blocks can also be used.

ARCHITECTURE	PROCESS	FPGA DEVICE	CLB SLICES	FREQUENCY (MHz)	THROUGHPUT (MBPS)
[6]	Encr	XCV1000 BG560	5302/10992	14,1/31,8	300/1940
[5]	Encr	XILINX (no specified)	5673	-	353
[12]	Decr	Altera EPF10K 250AG	2885	41,5	248
[11]	Encr/Decr	Altera APEX1K4001	845 LE	-	750 (best)
[8]	Encr/Decr	ASIC Approach	$\frac{35 \times 10^6 \text{ um}^2}{380 \times 10^6} \text{ um}^2$	-	265/4500
[7]	Encr/Decr	Xilinx Virtex	2902	25,9	331
[10]	Encr	ASIC Approach	3.96 mm ²	100	910
AKE	Encr/Decr	XCV300 BG432	2358	22	259
ARKL	Encr/Decr	XCV1000 BG560	17314	28,5	3650

TABLE 1 Performance Analysis Measurements

The size of RAM megacells can be customizable to fit the application demands in terms of the key length. In such an architecture, the switching time of the RAM is a factor that has to be considered in the total performance timing measurements.

4 PERFORMANCE ANALYSIS

Each one of the proposed architectures was implemented by using VHDL, with structural description logic. Both implementations were simulated for the correct encryption and decryption operation using the test vectors provided by the AES submission package [4]. The VHDL codes of the two designs are synthesized, placed, and routed using FPGA devices of Xilinx (Virtex) [17]. The two architectures then were simulated again for the verification of the correct functionality in real time operating conditions. The measurements of the performance analysis are shown in Table 1. Measurements from other designs are added in the same table.

The AKE architecture was optimized with covered area constraints. Xilinx Virtex XCV300BG432 was selected for this architecture implementation. The throughput reaches the value of 259 Mbit/sec for both encryption and decryption process. This architecture operates with an external clock with frequency of 22MHz. In the proposed architecture, the critical path is 45ns.

The throughput is calculated with the following formula:

$$Throughput = block_size * frequency/total clock cycles.$$
(13)

The transformed block size is 128 bit and the frequency is 22 MHz. The necessary clock cycles for one block encryption or decryption are 11.

For the pipelining ARKL architecture, the device Xilinx Virtex XCV1000BG560 was selected. This device has 128K bits of embedded RAM, divide in 32 RAM blocks, that are separate from the main body of the FPGA [17]. The FPGA device may be configured to provide a maximum of 384K bits of RAM independent of the supported embedded RAM. The Virtex block RAM also includes dedicated routing to provide an efficient interface with both Configurable Logic Blocks (CLBs) and other block RAMs.

The throughput in the pipelining architecture is given by:

$$Throughput = block_size/Tclkbasic,$$
(14)

where Tclkbasic is the delay of a single round, including register delay. Tclkbasic is 35 ns. The width of the transformed block size is 128 bit. The ARKL architecture achieves throughput 3.65 Gbit/sec. The external clock frequency is 28.5 MHz.

All the compared architectures operate with data and key block width of 128 bits. Someone could claim that the proposed AKE architecture has a little bit slower performance at about 10-15 percent compared with the other architectures. Nevertheless, this is a physical result of the algorithm philosophy and not a tradeoff. In this cryptographic algorithm, the key expansion unit is partially modified in the case of decryption process. Especially, as the Rijndael introducers clarify in their AES-Proposal specifications [13], the InvMixColumn has to be applied to all round keys except the first and the last one, during the decryption process. In our proposed architecture (AKE), the critical path is specified of the key expansion unit. In order to have a hardware implementation that supports both encryption and decryption, the critical path of the key expansion unit for the slower process (decryption) defines the critical path of the total system.

The two proposed architectures support encryption and decryption in the same dedicated hardware device. So, in a comparison attempt, in hardware performance with other architectures that support only encryption ([5], [6]), this special algorithm characteristic must be considered. Some other designs ([6], [7]) do not support the appropriate key scheduling unit in the implemented device. In the proposed AKE architecture, the appropriate key expansion unit has been integrated in the same FPGA device. This extra feature of the AKE architecture adds, of course, more allocated hardware recourses and decreases the algorithm core performance.

5 CONCLUSIONS

Two different philosophies of VLSI architectures for the design and implementation of the Rijndael encryption/decryption algorithm have been presented. The first uses feedback logic and reaches throughput value equal to 259 Mbit/sec. This architecture supports key expansion unit in the same device and performs efficiently in applications with low covered area resources. The second is optimized for high-speed performance using pipelining technique with high data throughput of 3.65 Gbit/sec. The resulting VLSI circuits achieve data rates significantly high, supporting both operation process (encryption/decryption) of Rijndael algorithm. They can be applied to online encryption/decryption needs of high speed networking protocols like Asynchronous Transfer Mode (ATM) or Fiber Distributed Data Interface (FDDI).

REFERENCES

- "Advanced Encryption Standard Development Effort," http://www.nist. gov/aes, 2000.
- [2] B. Schneier, Applied Cryptography Protocols, Algorithms and Source Code in C, second ed. New York: John Wiley & Sons, 1996.
- [3] "Advanced Encryption Standard Home Page," http://csrc.nist.gov/ encryption/aes, 2001.
- [4] "Third Advanced Encryption Standard (AES) Candidate Conf.," Apr. 2000. http://crscr.nist.gov/encryption/aes//round2/conf3/aes3conf.htm.
- [5] A. Dandalis, V.K. Prasanna, and J.D.P. Rolim, "A Comparative Study of Performance of AES Final Candidates Using FPGAs," *Proc. Third Advanced Encryption Standard (AES) Candidate Conf.*, Apr. 2000. (This work has also been published in the *Proc. CHES 2000*, Aug. 2000).
 [6] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA Based
- [6] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," Proc. Third Advanced Encryption Standard (AES) Candidate Conf., Apr. 2000.
- [7] K. Gaj and P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," *Proc. Third Advanced Encryption Standard (AES) Candidate Conf.*, Apr. 2000.
 [8] B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke, "Hardware Performance
- [8] B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke, "Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms," Proc. Third Advanced Encryption Standard (AES) Candidate Conf., Apr. 2000.
- [9] K. Gaj and P. Chodowiec, "Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays," *Proc. RSA Security Conf.*, Apr. 2001.
 [10] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82 Gbits/
- [10] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82 Gbits/ sec VLSI Implementation of the AES Rijndael Algorithm," *Proc. CHESS* 2001, May 2001.
- V. Fischer and M. Drutarovsky, "Two Methods of Rijndael Implementation in Reconfigurable Hardware," Proc. CHESS 2001, May 2001.
- [12] P. Mroczkowski, "Implementation of the Block Cipher Rijndael Using Altera FPGA," http://csrc.nist.gov/encryption/aes/round2/ pubcmnts.htm, 2001.
- [13] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," http://www.esat. kuleuven.ac.be/~rijmen/rijndael, 2001.
- [14] H. Brunner, A. Curiger, and M. Hofstetter, "On Computing Multiplicative Inverses in GF(2^m)," *IEEE Trans. Computers*, vol. 42, no. 8, pp. 1010-1015, Aug. 1993.
- [15] C.Č. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in GF(2^m)," *IEEE Trans. Computers*, vol. 34, no. 8, pp. 709-717, Aug. 1985.
- GF(2^m)," IEEE Trans. Computers, vol. 34, no. 8, pp. 709-717, Aug. 1985.
 K. Araki, I. Fujita, and M. Morisue, "Fast Inverters over Finite Field Based on Euclid's Algorithm," Trans IEICE, vol. E-72, no. 11, pp. 1230-1234, Nov. 1989.
- [17] Xilinx Inc., San Jose, Calif., "Virtex, 2.5 V Field Programmable Gate Arrays," 2001, www.xilinx.com.