AES-128 CIPHER. MINIMUM AREA, LOW COST FPGA IMPLEMENTATION

M. C. LIBERATORI† and J. C. BONADERO‡

† *Laboratorio de Comunicaciones, Facultad de Ingeniería, UNMDP, Mar del Plata, Argentina*
mlibera@fi.mdp.edu.ar.
‡ *Laboratorio de Comunicaciones, Facultad de Ingeniería, UNMDP, Mar del Plata, Argentina*
jbona@fi.mdp.edu.ar

***Abstract*— The Rijndael cipher, designed by Joan Daemen and Vincent Rijmen and recently selected as the official Advanced Encryption Standard (AES) is well suited for hardware use. This implementation can be carried out through several trade-offs between area and speed. This paper presents an 8-bit FPGA implementation of the 128-bit block and 128 bit-key AES cipher. Selected FPGA Family is Altera Flex 10K. The cipher operates at 25 MHz and consumes 286 clock cycles for algorithm encryption or decryption, resulting in a throughput of 11 Mbps. Synthesis results in the use of 957 logic cells and 6528 memory bits. The design target was optimization of area and cost.**

***Keywords*— FPGA, cryptography, AES, cipher, VHDL.**

I. INTRODUCTION

The Rijndael Algorithm, developed by Joan Daemen and Vincent Rijmen, has been approved by the U. S. National Institute of Standards and Technology (NIST) as the new Advanced Encryption Standard (AES). It became official in October 2000, replacing DES (FIPS 197, 2001). As this block cipher is expected to be widely used in an extensive variety of products, its efficient implementation becomes a significant priority. Hardware implementation is, by nature, more physically secure than software implementation.

The three major design targets with respect to hardware realization are: optimization for area or cost, low latency that minimizes time to encrypt a single block and high throughput to encrypt multiple blocks in parallel. All these design criteria involve a trade off between area and speed. There are a wide range of equipment where encryption is needed for authentication and security but throughput is not the principal concern. A low cost, small area design could be used in smart card applications as well as in other storage devices and low speed communication channels.

This paper presents an architecture for the 10 rounds AES Algorithm implemented on an Altera FPGA device. The goal of this design is to produce, in a low cost FPGA, a minimum area core cipher that exploits the symmetry between encryption and decryption operations.

The final architecture is based on previous work on the cipher design. In this work a decryption core is added, the number of clock cycles required to encrypt a single block has been reduced and the amount of hardware resources has been optimized with respect to the original design (Liberatori and Bonadero, 2005). A hierarchical design was adopted so that a collection of components were identified. Specified functions related with AES internal transformations were developed independently and composed to create the core cipher. The resulting design requires 957 LE´s, 6528 memory bits, operates at 11 Mbps and is compared with other known implementations.

II. THE AES ALGORITHM

AES is an iterative private-key symmetric block cipher (Stalling, 1999), operating on a block size of 128 bits. It comprises 10, 12 or 14 rounds when the key size is 128, 192 or 256 bits respectively. In an iterative block cipher, the block of information (*plaintext*) is transformed into another block of the same length (*ciphertext*) by repeated application of a *round* function. The intermediate cipher result is called the *state* in the AES proposal (Daemen and Rijmen, 1999). Each *round* involves an addition or bitwise *EXOR* of the plaintext and the key, so the original key must be expanded into a number of *Round Keys* and this transformation is known as the *Key Schedule*. A *Round Key* consists of a N_c word sub-array from the *Key Schedule* (Bonadero *et al*, 2005). In general the length of the cipher input, the cipher output and the cipher state is also N_c, and is measured in multiples of 32 bits. Rijndael Algorithm allows N_c to take values 4, 6 or 8 but the AES standard only allows a length of 4. The length of the cipher key, N_k, again measured in multiples of 32 bits, is also 4, 6 or 8, all of which are allowed by both Rijndael and the AES standard.

Each encryption round is composed of four operations: *SubBytes()*, *ShiftRows()*, *MixColumns()* and *AddRoundKey()*. The last round is slightly different because *MixColumns()* is not present.

SubBytes() transformation is a non-linear byte substitution as depicted in Fig. 1. Each byte of the *state* is inverted over $GF(2^8)$ followed by an affine transformation (Murphy and Robshaw, 2001). The overall operation is known as the *S-Box* (Rijmen, 2003) and can be performed by using a look up table.

ShiftRows() transformation operates on a whole state (128 bits). Rows of the state are cyclically shifted to the left as shown in Fig. 2.

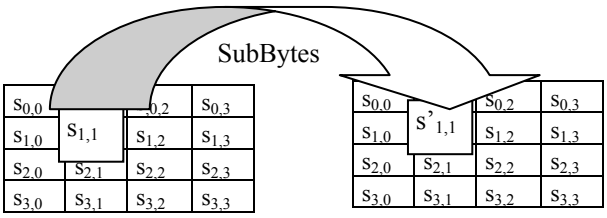

Figure 1 – *SubBytes()* Transformation

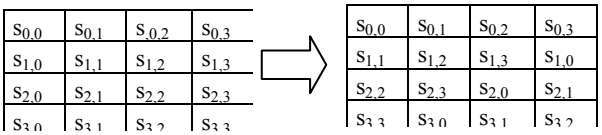

Figure 2 – *ShiftRows()* Transformation

MixColumns() transformation acts independently on every column of the state and treats each column as a four-term polynomial (Kerins *et al.*, 2002). The columns of the *state* are considered as polynomials over $GF(2^8)$ (Rijmen, 2003) and multiplied modulo (x^4+1) with a fixed polynomial $c(x)$, coprime to (x^4+1) and therefore invertible (Murphy and Robshaw, 2001). This operation can be written as a matrix multiplication.

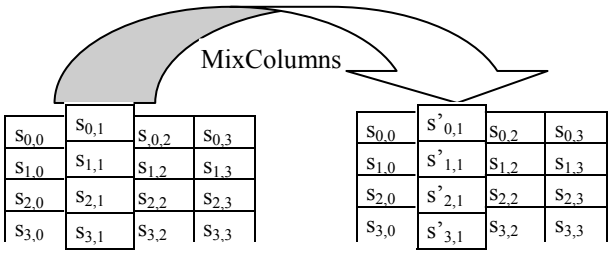

Figure 3 – *MixColumns()* Transformation

The cipher transformations can be inverted and then implemented in reverse order to produce an *Inverse Cipher* for the AES Algorithm. The individual transformations used in the *Inverse Cipher* are *InvShiftRows()*, *InvSubBytes()*, *AddRoundKey()* and *InvMixColumns()*. The last round is slightly different because *InvMixColumns()* is not present.

In the Inverse Cipher the sequence of transformations differs from that of the Cipher while the form of the *Key Schedule* for encryption and decryption remains the same.

However, several properties of the AES Algorithm allow for an *Equivalent Inverse Cipher* (Daemen and Rijmen, 1999) that has the same sequence of transformations as the cipher. This is accomplished with a change in the *Key Schedule*. There are two properties that allow for this *Equivalent Inverse Cipher*. On one hand, *SubBytes()* and *ShiftRows()* transformations commute and the same is true for their inverses. On the other hand, the column mixing operation and its inverse are linear with respect to the column input. Then the order of *AddRoundKey()* and *InvMixColumns()* transformations can be reversed, provided that the columns of the decryption *Key Schedule* are modified using the *InvMixColumns()* transformation:

$$InvMixColumns(state \oplus Roundkey) = InvMixColumns(state) \oplus InvMixColumns(Roundkey) \quad (1)$$

Given these changes, the resultant *EquivalentInverse Cipher* offers a more efficient structure than the *Inverse Cipher*. In this way the amount of extra resources that must be added to the original cipher perform both operations, encryption and decryption, is reduced. (Liberatori and Bonadero, 2005).

III. ARCHITECTURE OPTIONS

Rijndael is a block cipher with a basic looping architecture whereby data is iteratively passed through a round function. There are several architectural options to yield optimized implementations (Biham, 1999; Elbirt *et al*, 2001).

Basic Achitecture: When examining AES principal aspects, it is obvious that an implementation of the fully 128 data path stream could encrypt 128 bits per cycle (Gaj and Chodowiec, 2001) as shown in Fig. 4. One round of the cipher is implemented using combinational logic, one register and an input multiplexer. Such an implementation will encrypt 128 bits per clock cycle and consume many resources in terms of area. It will require a large amount of I/O pins and will not fit on low target FPGA.

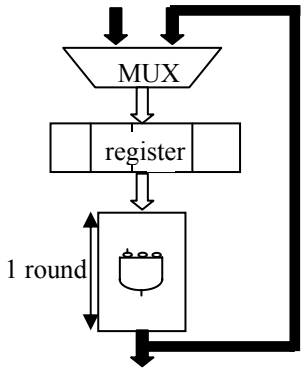

Figure 4 – Basic Architecture

Inner pipelining: In this architecture extra registers are added in the middle of the combinational logic, so that several blocks of data can be processed by the cipher at the same time, as depicted in Fig. 5. Circuit throughput is improved and area is increased.

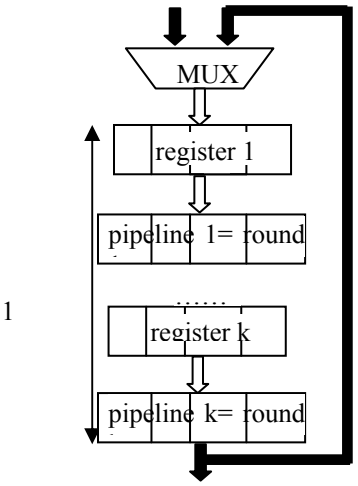

Figure 5 – Inner pipelining.

Loop Unrolling (LU-k): In this case the combinational part of the basic architecture implements *k* rounds of the cipher instead of a single round. An increase in speed is obtained at the cost of circuit area.

Proposed Architecture: In a minimum configuration (Fischer, 2000), the cipher should use as few memory blocks as possible and a basic interface with a host system as to allow its adaptation to a wide range of proposals. The provided embedded RAM is used to replace the round key and S-Box hardware. As a result, there is no key scheduling unit; instead a memory for storing the internal keys and the circuitry necessary to distribute these keys is included in the encryption/decryption unit.

As the circuit uses one Embedded Array Block (EAB) for key and sub-keys storage, only 5 EAB´s are free inside the selected FPGA. If we choose an internal bus of 16 bits we will need 4 EAB´s to implement S-boxes. Since one byte is the basic data unit for the Rijndael operations, the architecture selected to implement the cipher is an 8 bit basic one (Liberatori and Bonadero, 2005). In this way, only 2 EAB´s are used to implement *SubBytes()* operation.

On the other hand, *MixColumns()* operation is implemented over 4 bytes at a time. In this case we have to leave the 8 bit internal bus and introduce extra internal registers such that 4 blocks of the original internal data unit can be processed together. This is internal pipelining.

A first analysis of the algorithm identifies primary operations, which leads to the development of the functional units needed. This design strategy is a hierarchical one, where the basic blocks are implemented and then composed to obtain the cipher (Pardo and Boluda, 1999).

In the encryption/decryption core, only one round is implemented and the cipher must iterate ten rounds to perform an encryption/decryption. Iterative looping (LU-1) is a subset of loop unrolling (LU-k) where only one round is unrolled (Gaj and Chodowiec, 2001). This approach usually minimizes the hardware required for the implementation and an effort is made to maximize the speed. Thus, one round is implemented with combinatorial logic supplemented with registers, memories and multiplexers. First, input block of data is fed to the circuit via the 8-bit input interface and the initial round is executed. In this round the input data is XORed with the Cipher Key. Then the encryption/decryption unit evaluates ten rounds of the algorithm and the result is temporarily stored in the RAM. A control unit generates control signals for the other units, solving the problem of the separation between control and data path logic.

The basic architecture in conjunction with the non-feedback mode of operation (Stalling, 1999) is easy to implement and will likely result in smaller circuit area (Gaj and Chodowiec, 2001). The cipher was designed keeping in mind the amount of resources shared between encryption and decryption.

IV. FPGA IMPLENTATION

The FPGA family selected for the present implementation is Altera Flex 10K, in particular EPF10K20. It is a low volume device which has only 1152 logic elements (LEs) and 6 embedded array blocks (EABs). In this design, the device is part of the UP1 Educational Board of the University Program Design Laboratory Package from Altera. MAX+PLUS II (1996) Version 7.21 Student Edition is the software used to synthesize a VHDL (Terés *et al.*, 1997) implementation of the AES algorithm. This tool is also used to perform behavioral and timing simulations. Although it is tempting to generate cipher blocks with this automated tool, it has limitations. The high modular language that is integrated into the MAX+PLUS II System is AHDL. However, the cipher design is described in VHDL to ensure portability. On the other hand, our experience suggests that the best implementation results are achieved when hand mapping is used. And this is particularly true in the case of block cipher circuits that are used to implement algorithms. For this reason we have identified basic elements and followed a hierarchical design strategy where basic blocks are combined to create the desired cipher.

The round transformation data path is shown in Fig. 6.

It consists of one 16x8 RAM (U1), two 256x8 ROM (U2 and U3), one set of blocks to perform *MixColumns()/InvMixColumns()* operations with interfaces for the 8 bit data bus (*reg32mix8x8*/*invreg32mix8x8*), another two components to perform the *ShiftRows()*/*Inv ShiftRows()* operations (*shift*/*invshift*), one 8-bit *EXOR* and six multiplexers. Round Keys for encryption/decryption are stored in the 256x8 RAM (*ramkey*). All keys are loaded before the process begins. The *control* unit is a finite state machine with only three states to manage one initial round, nine similar intermediate rounds and a final round. It provides multiplexers select signals and generates control signals for the previously mentioned round components. The operation mode (encryption/decryption) is indicated to the *control* unit via an external signal: *encrypt/ decrypt.*

In either mode of operation, the input multiplexer, *mux4x8* U0, sequentially receives the 128-bit input data block, through an 8 bit bus (*datain*). The circuit processes this block through the successive rounds and the final result (plaintext or ciphertext) is stored in the RAM U1.

In the first round, the *AddRoundKey()* function is common to both modes of operation and the original key (encrypt mode) or the last sub-key (decrypt mode) is *EXORed* byte to byte with the input data.

Within each intermediate round, each byte is passed through the S-Box U2(encrypt) or InvS-Box U3 decrypt). These ROMs perform byte substitution, storing the overall transformation needed in 8 x 256 bits. They are implemented using two EAB blocks. An 8-bit address is the data input and an 8-bit data value is the output. The *control* unit selects the appropriate bus of the multiplexer U4.

The results from the *SubBytes()/InvSubBytes()* operation are temporarily stored in RAM U5 when the round number is even or in RAM U15 when the round

number is odd. These memories are written so that *SubBytes()* / *InvSubBytes()* and *ShiftRows()* /*InvShiftRows()* operations are combined. The components U13 and Z13 present the appropriate address value on the RAM memories address bus. Components U5 and U15 are read or written in this way.

In the case of encryption, the execution of these

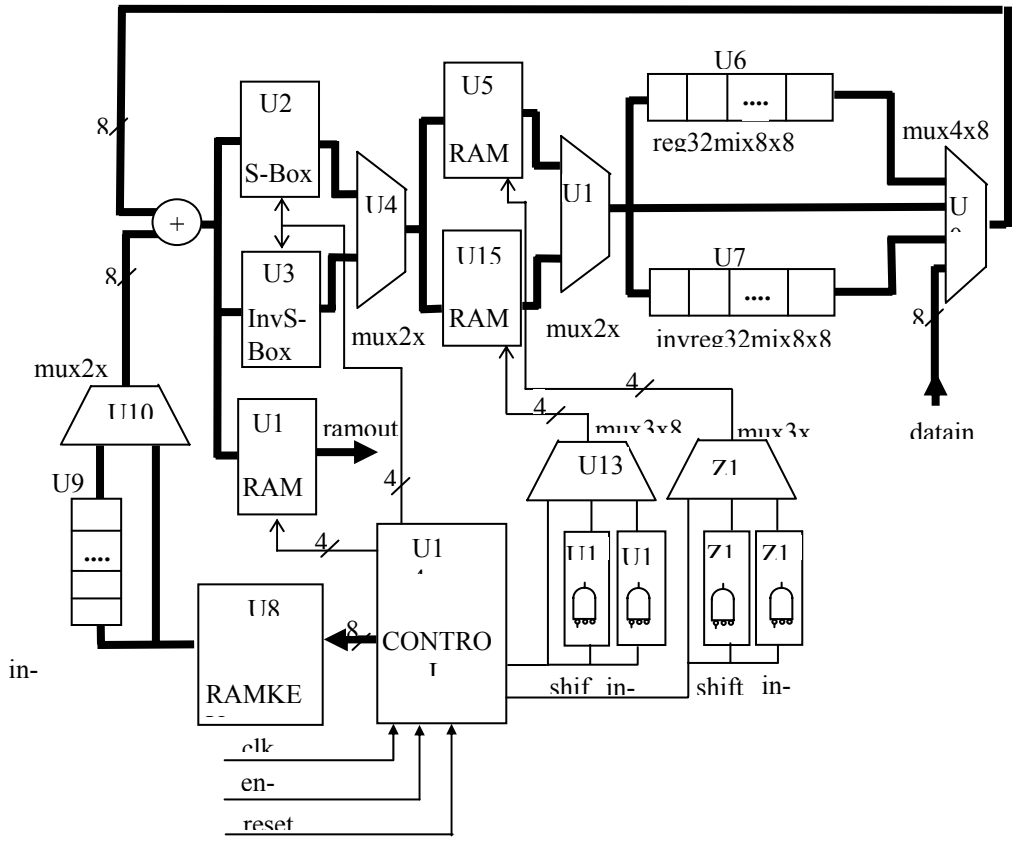

Figure 6 - Round Transformation Data Path for Encryption/Decryption

transformations modifies the address coming from the *control unit* via the *shift* component as it is depicted in Fig. 7. In the case of decryption, the writing order is imposed by the *invshift* component, shown in Fig. 8.

The *shift* or the *invshift* component generates the addresses in either RAM so that it can be written in the order that is presented in Table 1 (Karri and Kim, 1999). The *control* unit generates the addresses to read the memory. The reading process is a direct one, advancing from address 0 to address F.

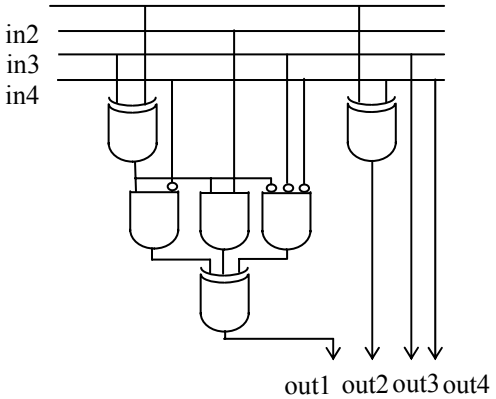

Figure 7 – *shift* component.

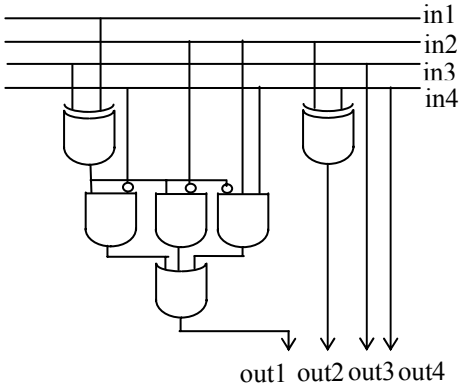

Figure 8 – *invshift* component

Table 1. *ShiftRows()* Transformation.

Reading from SROM	Writing RAM1	Reading from SROM	Writing RAM1
Byte 0	Address 0	Byte 8	Address 8
Byte 1	Address D	Byte 9	Address 5
Byte 2	Address A	Byte 10	Address 2
Byte 3	Address 7	Byte 11	Address F
Byte 4	Address 4	Byte 12	Address C
Byte 5	Address 1	Byte 13	Address 9
Byte 6	Address E	Byte 14	Address 6
Byte 7	Address B	Byte 15	Address 3

The RAM memory used in this transformation cannot be reused in the next round because it would be written at the same time it is being read.

This is the reason of the duplication of the components U5 and U15 and its accessories: U11, U13, Z11 and Z13. The multiplexer U16 allows the selection of the right memory output between even and odd round numbers.

The *MixColumns()/InvMixColumns()* transformations previously described can be written as a circular matrix multiplication. As a result, in the case of the encryption mode, the four bytes in the column can be replaced by the following expressions:

$$s'_{o,c} = (\{02\}.s_{o,c}) \oplus (\{03\}.s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \quad (2)$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\}.s_{1,c}) \oplus (\{03\}.s_{2,c}) \oplus s_{3,c} \quad (3)$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\}.s_{2,c}) \oplus (\{03\}.s_{3,c}) \quad (4)$$

$$s'_{3,c} = (\{03\}.s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\}.s_{3,c}) \quad (5)$$

In the case of decryption mode these expressions are replaced by:

$$s'_{o,c} = (\{0e\}.s_{o,c}) \oplus (\{0b\}.s_{1,c}) \oplus (\{0d\}.s_{2,c}) \oplus (\{09\}.s_{3,c}) \quad (6)$$

$$s'_{1,c} = (\{09\}.s_{0,c}) \oplus (\{0e\}.s_{1,c}) \oplus (\{0b\}.s_{2,c}) \oplus (\{0d\}.s_{3,c}) \quad (7)$$

$$s'_{2,c} = (\{0d\}.s_{0,c}) \oplus (\{09\}.s_{1,c}) \oplus (\{0e\}.s_{2,c}) \oplus (\{0b\}.s_{3,c}) \quad (8)$$

$$s'_{3,c} = (\{0b\}.s_{0,c}) \oplus (\{0d\}.s_{1,c}) \oplus (\{09\}.s_{2,c}) \oplus (\{0e\}.s_{3,c}) \quad (9)$$

A development of these equations allows us to identify the basic components to perform the mentioned transformations. Multiplication by *x* (i.e. 00000010 or {02}) can be implemented at the byte level as a left shift and a subsequent conditional *EXOR* with {1B}. This operation on bytes is denoted by *xtime()* (Karri and Kim, 1999). Multiplication by higher powers of x can be implemented by repeated application of *xtime()*. Multiplication by *(1+x)* (i.e. 00000011 or {03}) can be thought of as multiplication by ({01} ⊕ {02}). Fig. 9 and Fig 10 shows one basic block to perform *MixColumns()*operation and its inverse *InvMixColumnns()*.

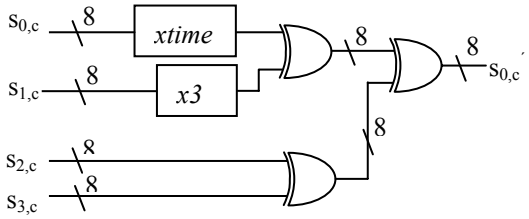

Figure 9. Basic block to perform *MixColumns()* Transformation

Four components like the one shown in Fig. 9, each with its entries consistent with Eq. (2) to Eq. (5), are needed to process 32-bit data simultaneously (Liberatori and Bonadero, 2005). In order to generate four bytes in one operation, *reg32mix8x8* accepts four bytes from input via a serial to parallel converter register. The result, one column of the state generated from each input column, must be converted to the serial form to fit in the original 8-bit data path (Shim *et al.*, 2002).

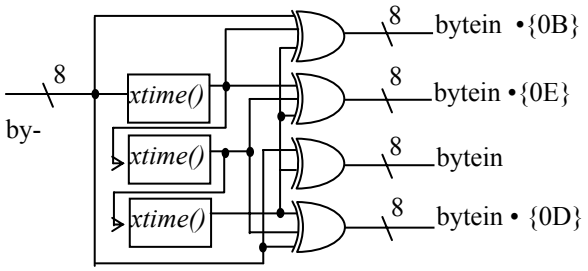

Figure 10. Basic block to perform *InvMixColumns* Transformation

The same reasoning applies to the inverse transformation in the decryption mode. Fig 10 shows the basic block to perform multiplication by constants in the Eq. (6) to Eq. (9). Four blocks like the one shown in Fig. 10 and a set of four *EXORs* are needed to perform the *InvMixColumns()* transformation.

V. RESULTS FOR THE ALTERA 10K FAMILY

The parameters used to evaluate the quality of the implementation are logic cells, bits of memory, cipher speed and Throughput Per LE (TPL).

The results of the implementation in terms of area and speed are summarized in Table 2. This table also presents the results obtained with other hardware implementations of the AES-128, targeted on different devices manufactured by Altera.

Altera synthesis tools measure the amount of used resources in terms of logic cells (LC´s) or logic elements (LE´s), because they are the basic constructive block inside any Altera device. Other FPGA manufacturers have similar tools that generate the same kind of reports. The principal difference between the reports from different manufacturers is the basic element definition and its interconnection with others of the same kind. For this reason, the results of the synthesis are compared with other implementations that have been targeted on chips from the same manufacturer.

Another metric used to compare different implementations is the Throughput Per LE´s (TPL) = Speed / Area (LE´s). When comparing implementations using TPL , it is required that the architectures are implemented on the same FPGA. Different FPGAs within the same family could yield different timing results as a function of available logic and routing resources, (Elbirt, *et al,* 2001).

Panato main design proposal is to produce a small area device with good performance and internal sub-key generation. The architecture is implemented in a high volume FPGA. To guarantee small area the Panato´s decision is to mix processes of 32 bits and 128 bits. The design with better TPL uses 66% of the memory resources.

Mroczkowski´s design contains an internal sub-key generator, 16 parallel working S-boxes for encryption or decryption and used an external clock with minimal period 22 ns ((45,45 kHz) for the encryption chip and 24 ns. (41, 46 kHz) for the decryption chip. The shift transformation is done by interconnections.

Table 2. Performance Results for comparison between different hardware implementations.

Design	Bits Memo	LE´s	Speed Mbps	TPL x 10^3	FPGA
Panato et al. 2003 (1)	32768 66%	3222 64%	150	46,55.	Acex1K
*Panato et al.*2003 (1)	0 0%	7034 35%	197	28	Cyclone
Mroczkowski (2001) (2)	40960	1032	268	259,68	Flex 10K250A
Mroczkowski (2001) (3)	40960	2885	248	85,96	Flex 10K250A
Fischer (2000) (4)	24EAB	3348	179	53,46	Flex 10KE
Fischer (2000) (5)	12EAB	3320	93.8	28,25	Flex 10KE
Fischer (2000) (6)	3 EAB	3324	24.3	7,34	Flex 10KE
Our design	**6528 53%**	**957 83%**	**11**	**11,49**	Flex 10K20

(1) Internal 32-bit /128-bit data path
(2) 128-bit data path for Encryption
(3) 128-bit data path for Decryption
(4) Fast configuration. 128-bit data path
(5) Fair configuration. 64-bit data path
(6) Minimum configuration. 16-bit data path

Fischer fast configuration uses as much S-boxes as possible to increase speed and also stores sub-keys in EAB. Fair configuration processes 64-bit data words. Minimum configuration uses as few memory blocks as possible with a 16-bit internal bus.

From Table 2, the most comparable VHDL implementation is the Fischer´s Minimum 16-bit data path (Fischer, 2000). Although it achieves more than the double of speed when is compared with our design, it requires almost twice the memory bits and three times more area in terms of logical cells. Our design offers better throughput per area, probably as a consequence of the hand placement and internal bus selection. In our approach, a small amount of memory is used as a register file to store intermediate results, with the data path performing the basic operations of the cipher/decipher. A state machine controls the basic 8 bit data path. Such a sequential approach is usually limited in performance but offers complete functionality in a small space.

On the other hand, none of the implementations presented in Table 2 can be synthesized on a device of low volume, just as Altera Flex 10K20.

In terms of complexity, the operation that requires more hardware resources as well as computation time is the *InvMixColumns()* multiplication. The design decision of working with an internal 8-bit data path implies two conversions: 8-bit serial to 32-bit parallel to perform the *MixColumns()/InvMixColumns()* transformation and 32-bit parallel to 8-bit serial to fit in the original 8-bit data path. This is the main limitation of the cipher performance in terms of speed.

VI. CONCLUSIONS

This paper presents a low area, cost-effective Rijndael cipher for encryption and decryption using a basic 8-bit iterative architecture, targeted towards the Altera Flex 10 K family of FPGAs. This architecture is based on previous work on the cipher design. In this work a decryption core is added, the number of clock cycles required to encrypt a single block has been reduced and the amount of hardware resources has been optimized with respect to the original design (Liberatori and Bonadero, 2005). The cipher has been synthesized using Altera MAX+PLUS II Version 7.21 Student Edition. The algorithm is implemented in VHDL, which led to the use of bottom-up design and test methodology. This choice also insures portability of the code to the devices of other vendors.

The architecture needs fewer logic cells than other ciphers and uses as few memory blocks as possible. It has 11 Mbps throughput. The minimum clock period depends on the access time to memories used and the frequency of the external clock.

Future work should concentrate on speed performance.

REFERENCES

Biham, E.: ¨A note on Comparing the AES Candidates¨. *Second AES conference* (1999).

Bonadero, J.C., M. Liberatori and H. Villagarcía Wanza,¨Expansión de la Clave en Rijndael. Diseño y Optimización en VHDL¨. *XI Reunión de Trabajo en Procesamiento de la Información y Control*, Rio Cuarto, Argentina, 115-120 (2995)

Daemen, J. and V. Rijmen, AES Proposal: Rijndael¨. Document version 2. *NIST´s AES home page*, http://www.nist.gov/aes. Date: 03/09/99. (1999)

Elbirt, A., W. Yip, B. Chetwynd and C. Paar, ¨An FPGA Implementation and Performance Evaluation of the AES block cipher candidates algorithm finalists¨. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 545-557 (2001).

FIPS 197, Federal Information Processing Standards Publication (FIPS) 197: ¨*Specification for the Advanced Encryption Standard (AES)*" NIST´s AES home page, http://www.nist.gov/aes. November 26 (2001).

FISCHER, V., ¨Realization of the round 2 AES candidates using Altera FPGA¨. http://csrc.nist.gov/ encryption/aes/round2/conf3/aes3papers.html. March 2000.

Gaj, K. and P. Chodowiec, "Comparison of the hardware performance of the AES candidates using reconfigurable hardware". *Proceedings of RSA Security Conference - Cryptographer's Track*, San Francisco, CA, 84-99 (2001).

Karri, R. and Y. Kim, ¨Field Programmable Gate Array implementation of Advanced Encryption Standard¨. http://www.eeweb.poly.edu/dream-it/publications/Rijndael.pdf. (2001).

Kerins, T., A Popovici, A. Daly and W. Marnane, "Hardware encryption engines for e-commerce". *Proceedings of Irish Signals and Systems Conference*, 89-94 (2002).

Liberatori, M. and J.C. Bonadero, "Minimum Area, low cost FPGA implementation of AES". *VIII International Symposium on Communications Theory and Applications*, UK, 461-466 (2005).

MAX+PLUS II. *Programmable Logic Development System. VHDL*. Altera Corporation. (1996)

Mroczkowski, P., "Implementation of the block cipher Rijndael using Altera FPGA". http://csrc.nist.gov/encryption/aes/round2/comments/20000510-pmroczkowski.pdf (2001).

Murphy, S. and M. Robshaw, "Essential Algebraic Structure within AES". *Second NESSIE. New European Schemes for Signature, Integrity and Encryption Workshop*. September (2001).

Panato, A., M. Barcelos and R. Reis, "A Low Device Occupation IP to Implement Rijndael Algorithm¨. http://www.inf.ufrgs.br/%7Epanato/artigos/designforum03.pdf. *Designer´s Forum. Munich, Germany*. (2003)

Pardo, F. and J. Boluda, *VHDL. Lenguaje para síntesis y modelado de circuitos*. Editorial RaMa (1999).

Rijmen, V.: ¨Efficient Implementation of the Rijndael S-Box¨. *CHES 2003*, LNCS 2779, 334-350 (2003).

Shim, J., D. Kim, Y. Kang, T. Kwon and J. Choi, "Inner-pipelining Rijndael cryptoprocessor with on-the-fly key scheduler". http://www.apsic.org/2002/proceedings/2B/2B-3.PDF (2002).

Stallings W.: *Cryptography and Network Security*, 2nd Edition, Prentice Hall.(1999).

Terés, L., Y. Torroja, S. Olcoz and E. Villar, *VHDL. Lenguaje Estándar de Diseño Electrónico*. Editorial Mc Graw Hill/Interamericana de España, S.A.U. (1997).

Received: April 14, 2006.
Accepted: September 8, 2006.
Recommended by Special Issue Editors Hilda Larrondo, Gustavo Sutter.