AES-128 CIPHER. HIGH SPEED, LOW COST FPGA IMPLEMENTATION

Mónica Liberatori, Fernando Otero

Electronics Departament
UNMDP
Juan B. Justo 4302. Mar del Plata. Argentina
email: mlibera@fi.mdp.edu.ar,
fotero@yahoo.com

J. C. Bonadero, Jorge Castiñeira

Electronics Departament
UNMDP
Juan B. Justo 4302. Mar del Plata. Argentina
email: jbona@fi.mdp.edu.ar
casti@fi.mdp.edu.ar,

ABSTRACT

The Rijndael cipher, designed by Joan Daemen and Vincent Rijmen, has been selected as the official Advanced Encryption Standard (AES) and it is well suited for hardware use. This implementation can be carried out through several trade-offs between area and speed. This paper presents an 64-bit FPGA implementation of the 128-bit block and 128 bit-key AES cipher. Selected FPGA Family is Spartan 3. The cipher consumes 52 clock cycles for algorithm encryption, resulting in a throughput of 120 Mbps. Synthesis results in the use of 1643 slices, 975 flip flops, 3055 4-input Look Up Tables and operates at 224 Mbps (maximum throughput). The design target was optimization of speed and cost.
Keywords: FPGA, cryptography, AES, cipher, VHDL.

1. INTRODUCTION

The Rijndael Algorithm became official in October 2000, replacing DES [1]. 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 is a wide range of equipment where encryption is needed for authentication and security. In fact, AES is at the core of 802.11i standard and is typically considered the most robust encryption strategy available. The trade-off is that AES requires additional processing power and may not be supported by older hardware used to generate ciphertext. A high speed, small area FPGA implementation could be used in these networks applications to alleviate the computing power from the main processor.

This paper presents an architecture for the 10 rounds AES Algorithm implemented on a Xilinx FPGA device. The goal of this design is to produce a high speed and low area core cipher. The final architecture is based on previous work on the cipher design [2], [3]. In this work, 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. A hierarchical design was adopted. Specified functions related with AES internal transformations were developed independently and composed to create the core cipher.

2. THE AES ALGORITHM

AES is an iterative private-key symmetric block cipher [4], operating on a block size of 128 bits. It comprises N_r rounds, with N_r = 10, 12 or 14 when the key size is N_k = 128, 192 or 256 bits respectively. At the end of each *round*, the intermediate cipher result is called the *state* in the AES proposal [4]. The *state* consists of four rows of bytes, each containing N_b bytes, where N_b is the block length divided by 32. 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* [5], [6].

Each encryption round is composed of four operations : *SubBytes()* [7], [8], *ShiftRows()*, *MixColumns()* [9] and *AddRoundKey()*. The last round is slightly different because *MixColumns()* is not present. A *Round Key* consists of a word sub-array from the *Key Schedule*. This key expansion generates a total of $N_b(N_r+1)$ words or sub keys. The algorithm takes an initial set of N_b words and each of the N_r rounds require N_b words of key data . The resulting key schedule consists of a linear array of *4*-byte words, denoted $w[_i]$, with i is in the range $0 \le i < N_b(N_r + 1)$. With a *128*-bit block and *128*-bit key AES cipher, $N_b = 4$ and $N_r = 10$ and *11* sub keys or *44 32*-bit words are obtained.

3. ARCHITECTURE OPTIONS

There are several architectural options to yield optimized implementations [10], [11]. In a fast configuration [12], the cipher should be capable to process 128 bits at once. In this case, the *ShiftRows()* transformation is just interconnection. *SubBytes()* can be implemented with Look Up Tables (LUT´s). *MixColumns ()* operation needs four identical circuits to process 128 bits at the same time. *AddRoundKey()* is a simple operation in either case.

The device selected is Spartan3 [13], in particular XC3S200FT256, with a maximum of 249 I/O pins. A fast

1-4244-0606-4/07/$20.00 ©2007 IEEE.

configuration would need 3 x 128 (384) I/O pins and many resources in terms of area. This important restriction leads to think in another architecture. A cipher implementation with an internal 64-bit bus would need 3 x 64 (192) I/O pins but the selected device offers only 173 single ended I/O. Thus, if we select two buses of 64 bits to process input and output data and one bus of 32 bit to enter the key, a total of 160 I/O pins would be used and 13 I/O pins would remain available for control signals. On the other hand, the selected FPGA has 1920 slices and an internal clock of 50 MHz. The total number of the available slices restricts the architecture selection to a single round loop unrolling (LU-1) with some degree of internal pipelining to improve throughput. For example, 64 clock cycles are needed for achieving a minimum throughput of 100 Mbps. This amount of internal latency could offer possibilities for internal pipelining. For these reasons, the architecture selected to implement the cipher is a 64 bit basic one with internal pipelining.

In this configuration, *ShiftRows()* is performed by wiring during two clock cycles. *SubBytes()* can be implemented with 8 LUT´s. *MixColumns()* operation is implemented over 4 bytes at a time. We have to turn the 64 bit internal bus into a 128 bit bus, in order to perform the operation in only one clock cycle. To achieve this objective, one extra internal register is introduced so that 2 blocks of the original internal data unit can be processed together. This is internal pipelining.

The design strategy is hierarchical, where the basic blocks are implemented and then composed to obtain the cipher [14]. In the encryption core, only one round is implemented and the cipher must iterate ten rounds to perform encryption. This approach usually minimizes the latency required for the implementation and an effort is made to improve the speed. In the first round the input data is *XOR*ed with the Cipher Key. Then the encryption unit evaluates ten rounds of the algorithm and the result is presented on the output bus. A *Control Unit* generates control signals for the other units, synchronizing internal operation and solving the problem of the separation between control and data path logic. A Key Scheduling unit performs the round key generation on the fly.

4. FPGA IMPLENTATION

XC3S200FT256 has 1920 Slices (480 CLB Slices) and is part of a Xilinx Educational Kit. Xilinx ISE version 6.3i and Modelsim 5.8 / Modelsim 6.0 SE Plus are the programs used to synthesize the VHDL [15] implementation and perform behavioral and timing simulations. Although it is tempting to generate cipher blocks with these automated tools, it has limitations. Thus, the cipher design is described in VHDL to ensure portability. Our experience suggests that the best implementation results are achieved when hand mapping is used. This is particularly true in the case of block cipher circuits that are used to implement algorithms. For this reason we have followed a hierarchical design strategy where basic blocks are combined to create the desired cipher. The round transformation data path is shown in Fig. 1.

In the first round, the first 64 bits of both, original key and plaintext, are EXORed. The output is directed to the *SubBytes* component. Every byte of the data stream is substituted with another byte. Direct implementation, using LUT´s, is the selected option to synthesize the component. This choice results in faster operation and reasonable use of area resources. Two clock cycles are needed to process the entire 128 bit input block. The first 64 bit block from the *SubBytes* component are temporarily stored in a internal register of 128 bits. This register is loaded so that *SubBytes()* and *ShiftRows()* operations are combined. This way, the *ShiftRows* component is just wiring. After the 128 bits are stored in the register, they are sent to the *MixColumns* component when the round is an intermediate one. In the case of the last round, this operation is not performed and the output is sent to the *AddRoundKey* component. The component U2 is in charge of this selection.

MixColumns() transformation acts independently on every column of the state array. The last byte from the *SuBytes()* operation is necessary to apply the *MixColumns()* transformation over the first column of the state. This is one important reason to work with an internal bus of 128 bits in this step of the algorithm. This way, the *MixColumns()* transformation requires four replied basic elements to replace four columns of the state at once.

The next component U3 must convert the internal bus of 128 bits to fit in the original data path of 64 bits. Additionally it must distinguish data coming from initial round, intermediate rounds and final round. Two internal registers allow to divide the 128 bits incoming data into two successive output blocks of 64 bits.

The *AddRoundKey()* function is the last operation of every round. U1 component distinguishes between the last round, where the ciphertext is obtained, and intermediate rounds, where the output from the *AddRoundKey* component is fed back onto the cipher.

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 can communicate with external host by means of two signals: idle/busy and valid_output. It provides multiplexers select signals and generates control signals for the previously mentioned round components and the *KeyScheduling* component. The on-the-fly architecture eliminates pre-processing overhead and does not require memory for storing the round keys which are generated dynamically. Only the cipher key, is serially loaded as a block of four words of 32 bits. The input multiplexer of the *KeyScheduling* component is controlled by the *Control Unit*. It can select between the original cipher key and the generated sub-keys for the

intermediate rounds. The output of this component is fed into a 4 x 32 bits shift register. This register stores each round key that is needed by the cipher core. Another register is used to store the present sub-key while processing the next one. The total number of cycles consumes to generate a round key is one less than that of processing one round of the ciphering. The *Control Unit* component is in charge of synchronizing the difference.

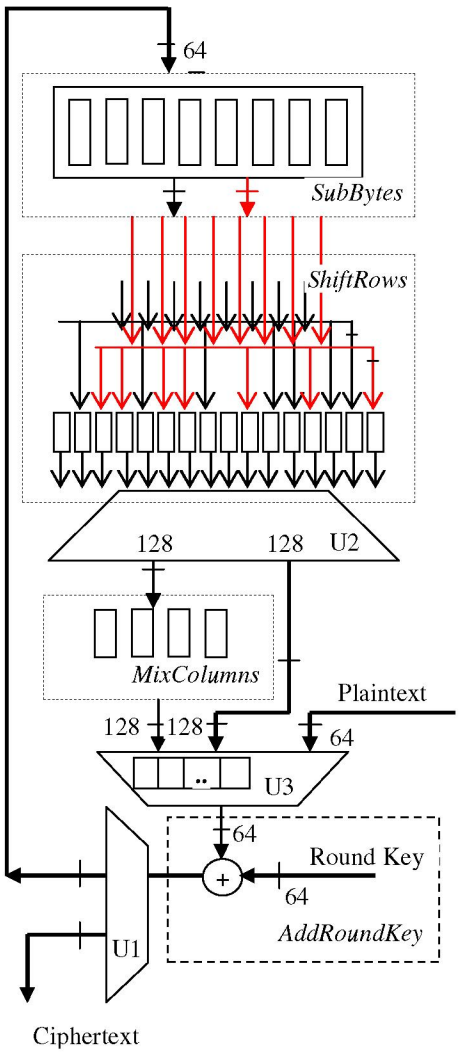

Figure 1 – Encryption Core

5. RESULTS

The parameters used to evaluate the quality of the implementation are slices, bits of memory, cipher speed and Throughput Per Slice (TPS). Xilinx synthesis tools measure the amount of used resources in terms of Configurable Logic Blocks (CLB´s) or slices, where one CLB is equivalent to four slices. 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.

When comparing implementations using TPS, 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 [11].

Number of Slices	1643 out of 1920 / 85%
Number of Slice FF	975 out of 3840 / 25%
Number of 4 input LUTs	3055 out of 3840 / 79%
Number of bonded IOBs:	168 out of 173 / 97%

Table 1 – Synthesis Results. Device :XC3s200FT256-5

The results of the implementation in terms of area and speed are summarized in Table 1. Table 2 presents the results obtained with other hardware implementations. The entire ciphering process previously explained consumes 52 clock cycles to complete one round of encryption and results in a throughput of 123.07 Mbps. Since the maximum frequency is 91.049MHz, the maximum possible throughput is 224.12 Mbps. Only up to 85% of resources of this FPGA are required by all the cipher components. The cipher core and the *Control Unit* component require 58% of the resources and the *RoundKey* component has an area cost of 546 slices.

Gaj and Chodowiec [16] main design proposal is to produce a small area device with good performance. The implementation performs both operations, encrypt and decrypt, but does not support the on-chip generation of internal keys. The architecture selected is a basic one with maximal resource sharing between encryption and decryption.

Elbirt [11] developed a round partially pipeline cipher design resulting in an efficient implementation in terms of area resources. Round keys are stored in internal registers and all keys must be loaded before encryption.

Shim [17] implemented an inner-pipelining cipher/decipher unit with on the fly key scheduler. In this case, the *SubBytes()* transformation and its inverse are store in ROM memory.

Chitu [18] implementation has capability to handle encryption and decryption with on the fly key scheduler.

All the mentioned implementations have an internal data path of 128 bits. From Table 2, Elbirt´s architecture is the the only one without Block RAM's. Although it achieves more than the double of speed when is compared with our design, it requires almost four times more area in

terms of CLB Slices. A CLB Slice is ½ CLB in the Virtex Family. Our design offers better throughput per area, probably as a consequence of the hand placement. On the other hand, none of the implementations presented in Table 2 can be synthesized on a device of low volume, just as Spartan3 XC3S200FT256.

Design	Bits Memo	CLB Slices	Speed Mbps	TPS	FPGA
Gaj et al	2 x 32kbits	2902	331.5	0.114	VirtexXC V1000
Elbirt et al.	-----	3061	491.4	0.160	VirtexXC V1000
Shim et al	2 x 32kbits	2580	451.5	0.175	VirtexXC V1000
Chitu et al	37 block	4325	739	0.170	Virtex II XC2V10 00-4
Our design	-----	**822**	**224.12**	**0.272**	Spartan3

Table 2. Performance Results for comparison.

5. CONCLUSIONS

This paper presents a high speed, low area, cost-effective Rijndael cipher for encryption using a basic 64-bit iterative architecture, targeted towards the Spartan family of FPGAs. This architecture is based on previous work on the cipher design. In this work a key scheduling unit 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. The original goal of this work was to compute the AES algorithm to generate the ciphertext in the range of 100 Mbps or higher, covering the challenge represented by the emergence of new protocols of the 802.11 family.

The architecture needs fewer logic cells than other ciphers and uses as few memory blocks as possible. It has 224 Mbps maximum throughput. The minimum clock period depends on the access time to memories used and the frequency of the external clock.

6. REFERENCES

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

[2] Liberatori, M., Bonadero, J.C.:¨AES-128 Cipher. Minimum Area, low cost FPGA implementation ¨. FPGA-Based Systems. Selected papers at the SPL 2006. II Southern Conference on Programmable Logic. March 8-10, 2006 Mar del Plata, Argentina. Chapter 2, Arithmetic and Cryptography, pp. 51-58.

[3] Liberatori, M., Bonadero, J.C.:¨Minimum Area, low cost FPGA implementation of AES¨. *VIII International Symposium on Communications Theory and Applications*, UK, Julio 2005. pp. 461-466.

[4] Stallings W.: *Cryptography and Network Security*, 2nd Edition, 1999. Prentice Hall.

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

[6] Bonadero, J. C., Liberatori, M., Villagarcía Wanza, H.: ¨Expansión de la Clave en Rijndael. Diseño y Optimización en VHDL¨. *XI RPIC XI Reunión de Trabajo en Procesamiento de la Información y Control*, Septiembre 2005. pp. 115-120.

[7] Murphy, S., Robshaw, M.: ¨Essential Algebraic Structure within AES¨. *Second NESSIE. New European Schemes for Signature, Integrity and Encryption Workshop*. September 2001.

[8] Rijmen, V.: ¨Efficient Implementation of the Rijndael S-Box¨. *CHES 2003*, LNCS 2779, pp 334-350.

[9] Kerins, T., Popovici, A., Daly, A., Marnane, W. : ¨Hardware encryption engines for e-commerce¨. *Proceedings of Irish Signals and Systems Conference*, 2002, pp 89-94.

[10] Biham, E. : ¨A note on Comparing the AES Candidates¨. *Second AES conference*, 1999.

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

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

[13] Xilinx. Spartan3 FPGA Family: Complete Data Sheet. www.xilinx.org.

[14] Pardo, F., Boluda, J.: *VHDL. Lenguaje para síntesis y modelado de circuitos*. Editorial RaMa. Edición 1999.

[15] Terés, L., Torroja, Y., Olcoz, S., Villar, E.: *VHDL. Lenguaje Estándar de Diseño Electrónico*. Editorial Mc Graw Hill/Interamericana de España, S.A.U. 1997.

[16] Gaj, K., Chodowiec, P. : ¨Comparison of the hardware performance of the AES candidates using reconfigurable hardware¨. *Proceedings of RSA Security Conference - Cryptographer's Track*, San Francisco, CA, 2001 April 8-12, pp. 84-99.

[17] Shim, J., Kim, D., Kang, Y., Kwon, T., Choi, J.: ¨Inner-pipelining Rijndael cryptoprocessor with on-the-fly key scheduler¨. http://www.ap-sic.org/2002/proceedings/2B/2B-3.PDF.

[18] Chitu C., Chien D, Chien C., Verbauwhede I., Chang F.: ¨ A hardware Implementation in FPGA of the Rijndael Algorithm¨. Circuit and Systems, vol 1, pp. 507-10. 2002.