Proceedings of the 5th WSEAS Int. Conf. on MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES, Corfu, Greece, August 17-19, 2005 (pp125-129)

A High-Speed and Area Efficient Hardware Implementation of AES-128 Encryption Standard

A. BROKALAKIS[1], H.MICHAIL[2], A.KAKAROUNTAS[2], E.FOTOPOULOU[2],
A.MILIDONIS[2],G.THEODORIDIS[3] ,C.GOUTIS[2]
[1]Computer Engineering & Informatics Department
[2]Electrical & Computer Engineering Department
University of Patras
GR-26500 Patra
[3] Department. Of Physics, Aristotle
University of Thesalloniki,
GR-54006 Thesalloniki
GREECE

Abstract: - The Advanced Encryption Standard (AES) is used nowadays extensively in many network and multimedia applications to address security issues. In this paper, a high throughput area efficient FPGA implementation of the latter cryptographic primitive is proposed. It presents the highest performance (in terms of throughput) among competitive academic and commercial implementations. Using a Virtex-II device, a 1.94Gbps throughput is achieved, while the memory usage remains low (8 BlockRAMs) and the CLB coverage moderate.

Key-Words: - Security, AES-128 , Area-Efficient, High-Throughput, Hardware Implementation, ASIC, FPGA.

1 Introduction

Transmission and storage of sensitive data in open networked environments is rapidly growing. Along with it, grows the need of efficient and fast data encryption. Software implementations of cryptographic algorithms cannot provide the necessary performance when large amounts of data have to be moved over high speed network links that reach the Gbps range. Therefore hardware implementations have to be considered for these applications, either in the form of ASIC or FPGA designs.

FPGAs are very well suited for high speed cryptography, as they can provide the required performance without the excessive cost of initialing an ASIC manufacturing process. AES (Advanced Encryption Standard) [1] has become the algorithm of choice among the cryptographic protocols that are based on symmetric cipher

Since then, numerous FPGA implementations of the AES algorithm have been reported in literature, while commercial IP cores are also offered. Designs that focus on performance are typically divided on two groups. The first group involves fast AES encryption / decryption cores which provide high throughput while requiring a reasonable amount of resources ([2], [3], [5], [8], [9], [10]). The second group targets only ultra fast implementations which achieve throughputs of an order of magnitude greater by fully unrolling the algorithm ([2], [3], [7], [10]). The latter are extremely demanding in terms of area, and only the largest FPGA devices can accommodate them.

This paper describes a high throughput implementation of the AES encryption algorithm with 128-bit cipher key (AES-128) on a Xilinx Virtex-II FPGA. This design has moderate area demands in terms of combinational logic and memory, while its performance sets it on top of its class. The low resource usage and high throughput achieved, position this implementation as an ideal candidate for integration in SoCs or ASIP designs that target applications where the AES algorithm is needed. Two most representative examples of these applications are IPSec [12] and JPSec[13].

Section 2 details the architecture and overall design of the implementation, while area and timing results are presented in Section 3, in comparison with other published FPGA-implementations. Conclusions are drawn in Section 4.

2 Proposed AES-128 Architecture

The high-level architectural organization of the AES encryption core is presented in Fig. 1. As it may be seen from Fig.1, five logic blocks compose the overall system. The Input Interface unit is responsible for feeding the Key Logic and the Processing Core with the correct data from the input lines. Key Logic handles any cipher key related operation. Processing Core is the unit that performs the main AES encryption process. SBox is a ROM that is used for the SubBytes (SubWord) transformation and finally the System Control Unit is responsible for the system's overall synchronization and communication with external logic. In the following subsections the functionality of each logic block will be further explained.

2.1 Input Interface

When a new data block has to be loaded to the system, the external logic has to notify the system that the data present on the Key/Plaintext input lines are valid and to define the type (key or plaintext) of these data. Two signals are used for this purpose, LoadKey and LoadData.

The activation of one of them indicates the validity of the data in the Key/Plaintext lines as well as the data type. Upon loading a new key, the Input Interface will latch the value from the input lines and notify the Key Logic that a new cipher key is available which is also stored in a local register for future usage.

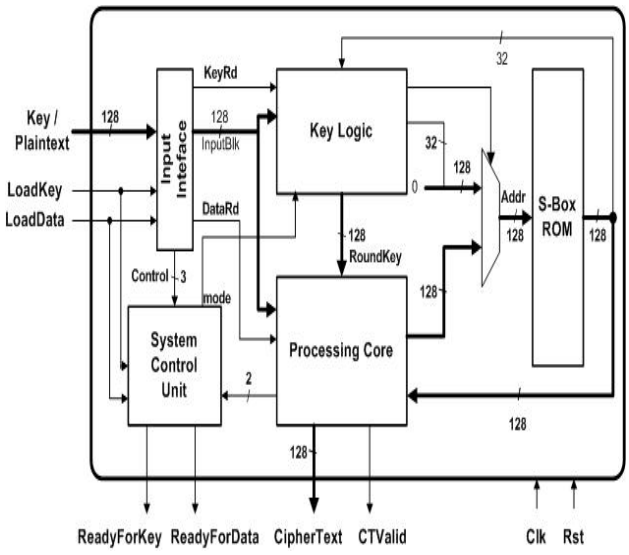

Fig. 1: Block diagram of the AES encryption core

When new plaintext is loaded, the Input Interface latches the result of a bitwise xor operation between the value of the input lines and the value of the cipher key that is stored in the local register. The Processing Core unit is then notified that a new state is available for processing. This way, the initial round of the encryption process is carried out together with the loading of data and processing time is saved.

2.2 Key Logic

The AES algorithm requires that a new round key is used in every encryption round. There are three different approaches to the time and manner that these round keys are calculated. The "online" approach (followed by the majority of the referenced implementations) will calculate a new round key at every encryption round using the previous round key. Another approach (which may be considered as "offline"), generates all round keys upon the reception of the initial cipher key and stores them in a small memory.

This memory is accessed at every encryption round in order to provide the necessary round key. The final method is a pure offline approach where the round keys are precalculated by an external source, say an external key generator or a program running on a processor, and are loaded to the AES system sequentially as in [3]. The implementation presented in this paper is based on the second approach for a number of reasons.

The calculation of the round keys is totally independent of the plaintext and the round keys that are produced from a certain cipher key will remain the same as long as the initial cipher key does not change. Since a single cipher key is used for the encryption of data collections (such as files or packets) of typical sizes significantly larger than 128-bits, the same round keys are going to be used for a many plaintext blocks. Therefore the online approach wastes both resources and power. Additionally, the online approach may prove more complex to design.

However, the cost to be paid for not using an online key expansion, comes in terms of latency, but as it will be shown this is not actually high.The pure offline approach, on the other hand, is better suited to systems that are resource-limited and do not target high performance. The power saved due to lack of specific hardware to generate the round keys is invested in increased bus activity to load the keys to the core. If the bus is external power dissipation grows significantly.

To sum up the Key Logic unit has two main functions a) to perform the key expansion process whenever a new cipher key is inserted to the system

and b) to access the local memory in order to provide the right round keys during an encryption process. The operation mode of the Key Logic unit is controlled by a signal from the system's Control unit.

For producing a new round key, two transformations have to be performed, RotWord and SubWord. RotWord cyclically shifts the bytes of the first 32-bit word of the previous key by one position to the left whereas. SubWord performs the SubBytes transformation to each byte of the rotated word. Simple bitwise xors are then needed in order to produce the final round key.

The SubWord (SubBytes) transformation is implemented with a ROM, called S-Box. This ROM is a synchronous memory and thus it requires one clock cycle to produce an output. Therefore, the process to generate a round key is logically divided into two processing cycles. During the first one, the RotWord transformation is performed and an address is generated for the S-Box ROM. At the second cycle, the ROM's output is xored with the proper Rcon value and subsequent xors of this result produce the remaining 3 words of the round key. All four words are then stored at the unit's register file. The whole key expansion process requires 20 cycles to be completed (10 round keys to be generated in 2 cycles each).

An 11x128 register file is needed in order to store all round keys. Since only one access (read or write) is required in a cycle, the register file is organized as a single-ported RAM with synchronous read and write.

2.3 Processing Core & S-Box

The Processing Core unit performs the actual AES encryption process. A block diagram of the unit is presented in Fig. 2.

In order to complete the encryption process of a plaintext block, 10 encryption rounds are needed. SubBytes, ShiftRows, MixColumns and AddRoundKey transformations have to be performed during each round. The SubBytes transformation is applied on each byte in the state matrix, altering its value by a non-linear manner. The ShiftRows transformation then acts upon each row of the state and changes the position of each byte in that row. ShiftRows transformation does not alter in any way the value of a byte in the state. Therefore, these two transformations can be merged in one, which operates on both the value and the position of each byte in the state. The S-Box byte substitution function (which underlies the SubBytes transformation) can be implemented either by using

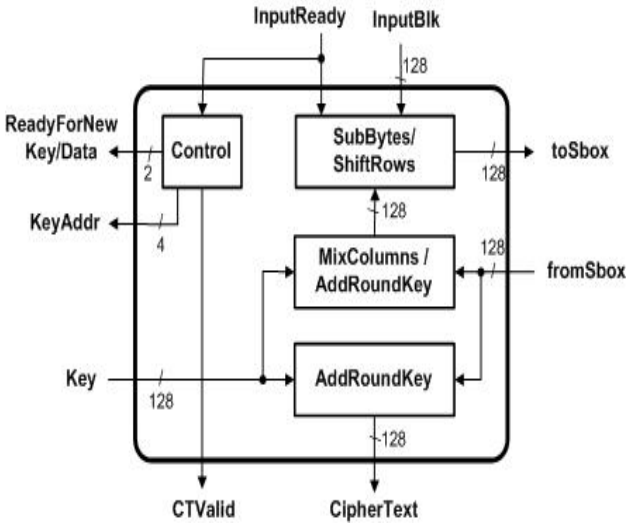

Fig. 2: The Processing Core Logic Unit

combinational logic or a ROM containing all 256 possible precalculated outcomes. The most appealing implementation - in terms of area/performance - on an FPGA device is the usage of a ROM. Reports from implementations that have used combinational logic for the S-Box function ([5], [6], [7]) confirm this approach. Xilinx Virtex-II FPGAs provide fast on-chip memories, (BlockRAMs), which are ideal for this kind of design. A state is comprised of 128-bits or 16 bytes. Since the S-Box byte substitution function must be applied to each byte of the state, 16 ROMs have to be used.

As BlockRAMs can be configured as dual-ported ROMs, the total amount of ROMs required is reduced to 8. The values of the bytes in the initial state are used as addresses to these ROMs and the data read from them is the transformed state. The whole process – generating an address for the S-Box and getting the results – requires one cycle, because BlockRAMs are synchronous.

The outcome of the combined SubBytes/ShiftRows transformation has to undergo two more transformations. The MixColumns transformation is quite simple and can be effectively implemented by a network of xor gates (2 levels).One more level of xor gates has to be used for the AddRoundKey transformation. This delay is very low, does not affect the system's critical path and therefore these two transformations may be carried out in a single cycle.

From the above, it is concluded that the completion of an encryption round requires 2 cycles. Since 10 rounds have to be executed, a total of 20 cycles is needed to produce the final state (the ciphertext). What is interesting in this

implementation is the fact that without replicating any logic unit, two blocks of data may be processed in parallel and a new ciphertext may be produced at the next cycle. At any given time during the computation, a block of data may lie either at the first stage of processing (the SubBytes/ShiftRows part) or at the second (the MixColumns/AddRoundKey part). Therefore, another block of data may be processed by the logic of the stage that it is not currently used. As a result, this design behaves equivalently with implementations that require 1 cycle per round (such as [2] that requires 11 cycles to produce an outcome), but since the combinational part is broken to two parts, greater frequencies may be achieved.

In order for this scheme to work, careful scheduling has to be exercised. The Processing Core includes a dedicated control unit that supervises the whole process. This subunit (implemented as a Finite State Machine) is responsible for the generation of all the control signals that shape the data flow in the Processing Core. Above that, it is the logic block that computes the addresses of the round keys to be used in each round and generates a number of signals through which communication with other units and the system's outputs is realized. For example, before the last encryption round, the ReadyForNewData signal is generated, so that the system may indicate that new plaintext data may be loaded and thus the processing of new data may start without delay.

2.4 System Control Unit

This unit has the overall control of the system. Every logic block of the system has been designed so that it can operate autonomously without external supervision. Therefore the central control unit is fairly simple (just a FSM with very few states) and it is mainly responsible for some I/O signaling and the selection of mode of operation of the logic blocks.

3 Implementation Results

The AES-128 encryption system has been designed using Verilog HDL. XST and Leonardo Spectrum have been used for synthesis. XST has been used to synthesize only the modules that infer the BlockRAMs, because Leonardo Spectrum does not infer dual-ported BlockRAMs. Xilinx Virtex-II XC2V1000bg575 (speed grade -5) has been chosen as the target device.

Tables 1 and 2 provide a comparison with competitive academic and commercial implementations such as [2], [3], [6], [8] and [10]. For reasons of thoroughness, the best software implementation found (Lipmaa [4]) is also included in the comparison.

Tables 1 and 2 provide a comparison in terms of absolute numbers related to frequency, throughput, memory usage (indicated as number of BlockRAMs used) and area (CLB usage). The marks BlockRAMs per Gbps and CLB Slices per Gbps illustrate the relationship between throughput and area requirements in a more comprehensive way.

DESIGN	Frequency (MHz)	Throughput (Mbps)
[2]	113.7	1323
[3]	146.0	1699
[4]	-	1538
[6]	115.0	1330
[8]	29.9	1911
[10]	119.0	1450.0
Proposed	**159.2**	**1940.9**

Table 1. Frequency and Throughput Comparisons of AES-128 Implementations

DESIGN	BlockRAMs	CLB Slices
[2]	10	573
[3]	10	450
[4]	-	-
[6]	10	800
[8]	4	8767
[10]	10	542
Proposed	**8**	**1122**

Table 2. Frequency and Throughput Comparisons of AES-128 Implementations

The results demonstrate that our proposed implementation is the best of its class in terms of throughput, by a considerable margin as it is shown in Table 1. Only [8] is close enough, but one has to take into consideration that this implementation is a 5-pipelined version of their basic architecture meaning that all encryption resources are replicated 5 times. In terms of memory usage, our implementation has the lowest requirements, except for [8]. It should be stated, though, that these designs use two to 4-times more CLB resources. As far as CLB resources are concerned, our implementation seems to be on an average level.

DESIGN	BlockRAMs/Gbps	Slices/Gbps
[2]	7.56	758.05
[3]	5.88	264.86
[4]	-	-
[6]	7.69	615.38
[8]	1.05	4587.65
[10]	6.90	373.79
Proposed	**4.12**	**578.05**

Table 3. Area and Memory Usage Comparisons of AES-128 Implementations

This picture changes significantly if the CLB Slices per Gbps mark is to be considered, since only [3] and [6] perform better in this metric. The relevant comparisons are shown in Table 3 where the ratio BlockRAMs to Throughput and CLB Slices to Throughput have been estimated for athe proposed and all alternatives AES-128 implementations.

4 Conclusion

A highly efficient FPGA implementation of the AES encryption algorithm has been presented. Offline key expansion is used in order to reduce memory requirements and save power. Combined data load and execution of the initial round of the encryption algorithm reduces the number of rounds required, by one. Inner-round pipelining and thorough scheduling allow high frequencies to be achieved and efficient usage of resources. The resulting implementation has moderate area demands in terms of CLB slices, low memory requirements and achieves throughputs in the range of 2 Gbps. Compared to other academic and commercial implementations, the presented design demonstrated the highest throughput and one of the smallest memory/area to performance ratios.

Acknowledgments

We thank European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the program PYTHAGORAS, for funding the above work.

References:

[1] FIPS. Advanced Encryption Standard (AES). FIPS PUB-197, November 26 2001.

[2] Amphion Web page, available at http://www.amphion.com/index.html

[3] Helion Technology Ltd. Web page, available at http://www.heliontech.com.

[4] H. Lipmaa. AES/Rijndael speed comparison. www.tcs.hut.fi/~helger/aes/rijndael.html

[5] Christopher Caltagirone and Kasi AnanthaI. High Throughput, Parallelized 128-bit AES Encryption in a Resource-Limited FPGA, in SPAA'03, June 2003.

[6] Francois-Xavier Standaert, Gael Rouvroy, JeanJacques Quisquater and JeanDidier Legat. A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL, in FPGA'03, February 2003.

[7] Kimmo U. Jarvinen, Matti T. Tommiska and Jorma O. Skytta. A Fully Pipelined Memoryless 17.8 Gbps AES128 Encryptor, in FPGA'03, February 2003.

[8] Anna Labbe and Annie Perez. AES Implementation on FPGA: Time - Flexibility Tradeoff, in FPL 2002, FPL 2002, LNCS 2438, pp. 836-844, 2002.

[9] Nicholas Weaver. Rijndael core. www.cs.berkeley.edu/~nweaver/rijndael.

[10] George Mason University. Hardware IP Cores of Advanced Encryption Standard AES-Rijndael. ece.gmu.edu/crypto/rijndael.htm.

[11] S.Mangard, M.Aigner and S.Dominikus. A Highly Regular and Scalable AES Hardware Architecture, in IEEE Transactions on Computers, vol. 52, no. 4, pp. 483-491, April 2003.

[12] Renee Esposito and Richard Graveman, Protocol Security : IPSec and IKE, IPv6 Security Workshop, June 2004

[13] Frederic Dufaux, Susie Wee, John Apostolopoulos and Touradj Ebrahimi. JPSEC for Secure Imaging in JPEG 2000, in Photonic Devices and Algorithms for Computing VI, Proceedings of the SPIE, Volume 5558, pp. 319-330 (2004)