Implementation of a Cryptographic Co-processor

A.P. KAKAROUNTAS, H. MICHAIL
Dpt. of Electrical & Computer Engineering
University of Patras
University Campus, Patras, GR-26500
GREECE

Abstract: - In this paper a high-speed cryptographic co-processor, named HSSec, is presented. The core embeds two hash functions, SHA-1 and SHA-512, and the symmetric block cipher AES. The architecture of HSSec renders it suitable for widely spread applications with security demands. The presented co-processor can be used inevery system integrating standards such as IPSec or the upcoming JPSec and P1619. The main characteristic of the proposed implementation is common use of the available resources, to minimize further area requirements. Additionally the cryptographic primitives can operate in parallel, providing high throughput whenever needed. Finally the system can operate in ECB or CBC modes. The HSSec co-processor has relatively small area and its performance reaches 1 Gbps (AES, SHA-1 and SHA-512) for XILINX's Virtex II FPGA family.

Key-Words: - Cryptography, Block cipher, Hash function, Embedded computing, Processing core, Authentication

1 Introduction

Software implementations of cryptographic algorithms cannot provide the necessary performance when large amounts of data have to be moved over high speed channels or store data in high-capacitance storage devices. Thus, leading service providers turn to hardware solutions aiming to high performance and competitive characteristics for their products. However, available cryptographic cores hardly meet these specifications, especially for applications requiring throughput over the 1 Gbps. Protocols such as IPSec [1], JPSec [2] and the upcoming IEEE P1619 [3] require solutions that can serve almost real-time system demands.

This race for high-speed cryptographic implementations was the motivation for the implementation of a High-Speed Secure (HSSec) co-processor. The HSSec core is a crypto-system that provides encryption/decryption using the widely used symmetric block cipher Rijndael [4], described in [5]. The HSSec embeds AES-128 and SHA-1 and SHA-512, which are fully described in Secure Hash Standard [6].

Every attempt until now to implement a core combining the latter algorithms was based on the use of existing separate cores [7–12]. However, each core is optimized for use as stand-alone, resulting in bulky implementations with low performance. The proposed co-processor exploits characteristics of the cryptographic algorithms. Additionally, HSSec allows concurrent operation of AES and SHA, resulting in parallel processing of a packet (or image frame). The modes of operation [13] are EBC (Electronic Code Book) and CBC (Cipher Block Chaining), and CFB and OFB are supported. The core is fully programmable making it suitable for use with a general purpose processor. In section 2, the implementation of AES-128, SHA-1 and SHA-512 is prsented. In section 3, the architecture of the proposed co-processor is presented. In section 4, details of the implementation of HSSec are offered. Finally, conclusions are offered in section 5.

2 Exploration of SHA and AES

The HSSec, as it was mentioned until now, allows the parallel operation of three cryptographic primitives: AES-128, SHA-1 and SHA-512. The first step during development of the HSSec was to explore the three cryptographic primitives and find common characteristics that can help design and optimization process. Below the three algorithms are presented and at the end their characteristics are highlighted.

2.1 SHA-1 exploration

SHA-1 is the most commonly used hash function along with MD4-MD5 hash functions. SHA-1 may be used to hash a k-bits message, where $0 < k < 2^{64}$. During pre-processing phase the message is padded

and parsed into 512-bit message blocks, which are used to generate the message schedule W_t s. SHA-1 requires 80 cycles to produce the 160-bit message digest. Each cycle requires the previous rounds results, W_t, as well as constant values K_t. The rounds of SHA-1 differ by the applied non-linear function f_t. There are in total four non-linear functions, $f_1(0 \leq t \leq 19)$, $f_2(20 \leq t \leq 39)$, $f_3(40 \leq t \leq 59)$, $f_4(60 \leq t \leq 79)$.

2.2 SHA-512 exploration

SHA-512 may be used to hash a k-bits message, where $0 < k < 2^{128}$. During pre-processing phase the message is padded and parsed into 1024-bit message blocks, which are used to generate the message schedule W_t. SHA-512 requires 80 cycles to produce the 512-bit message digest.

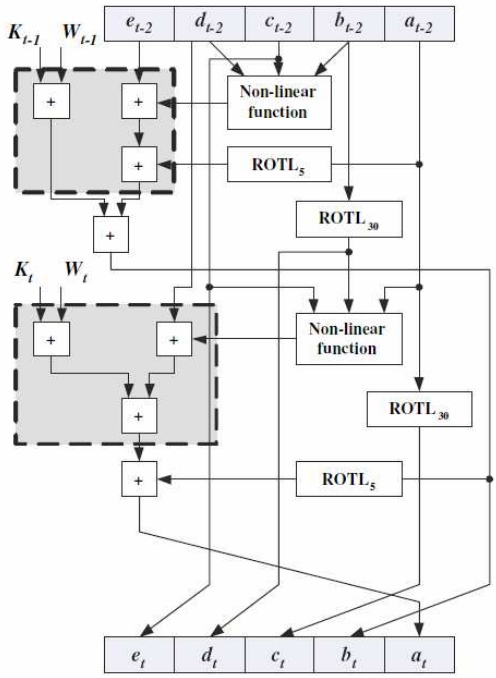

Fig. 1 Partially unrolled SHA-1 operation block (two operations per cycle).

Each cycle requires the previous rounds results, W_t, as well as constant values K_t. Contrary to SHA-1, all the operation blocks are identical. An interesting proposal to implement small-sized high-speed hash functions is presented in [15]. It is based on pre-computing values with no dependencies and use them later in the operation block. This design approach when applied to the SHA-512 operation block, increases its throughput by 30% with negligible area overhead. The final implementation requires a padding memory of 1024-bits, eight temporal registers (64-bit) to store the intermediate hash values and sixteen temporal registers (32-bit) to store the message schedule W_t.

2.3 AES-128 exploration

The AES algorithm is a round-based, symmetric block cipher. It processes data blocks of fixed size (128 bits) using cipher keys of length 128, 196 or 256 bits. HSSec embeds the AES-128 because it is the most popular variant of the algorithm. The algorithm can be split in three processes:

Padding The 128-bits of the plaintext are aligned in the States matrix.

Key expansion The initial 128-bit cipher key has to be expanded to eleven round keys of same length. The first round key is the cipher key ($RoundKey_0$) and all subsequent round keys are produced when a function is applied to the previously generated round key.

Encryption AES performs a number of transformations to the plaintext, and a 128-bit output block (called ciphertext) is produced as a result.

A variety of implementations can be found in the IP market and the international literature. Two works are of special value for the HSSec implementation. In [16] a highly regular architecture of AES-128 is given. However, the resulted throughput doesn't meet the need for high performance. Also, the architecture cannot be easily modified to co-exist with SHA-1 and SHA-512 cores. However, it served as the cornerstone of the HSSec architecture. In [17] a high-speed implementation of AES-128 is proposed, that requires 10+1 clock cycles to produce the ciphertext. This design approach was followed in order to implement AES for the HSSec. Combining the two works, it was possible to design a high-speed AES-128 with low design complexity and disjoint the resources (such as memory) that can be used from SHA variants. Summarizing, to implement the AES-128 there is the need of a padding memory of 128-bits and sixteen temporal registers (8-bit) to store the *State* matrix.

2.4 Exploration Results

From the analysis of the requirements of the three algorithms it was derived that in the case of AES-128, there is the need to process 128-bits in 10 clock cycles, in the case of a partially unrolled SHA-1, process 512-bits in at least 40 clock cycles and in the case of pre-computed SHA-512, process 1024-bits in at least 80 clock cycles. The latter presents a symmetry in the width of processing data, indicating

that there must be available at least 128-bits every 10 clock-cycles. This allows synchronized parallel processing of the available data in the padding memory. This means that instaed of requiring three padding units of totally 1664 bits, one padding unit of 1024 bits is more than sufficient.

Another remark that can be made, based on the analysis of the three algorithms, is the need of an extra block that provides data with time dependencies, such as the message schedules Wt and the constants for the SHA-1 and the SHA-512 and the keys for the rounds of AES-128. Finally, there is also the need for a sophisticated register file to store immediate and/or initialization values for the three algorithms.

3 System Architecture

The HSSEC co-processor is based on a typical architecture with central control and processing elements (PE) in its periphery. The data inputs are 32-bit wide, while output is offered through two 32-bit wide ports. The handshake signals READY and SEND allow synchronization of the core for data receive/send. Furthermore, it controls the flow of the data in order to successfully provide the calculated outputs. Especially, signal SEND is also used as a halt signal, when the core outputs the message digest of SHA-512. Through the *AES_en*, *SHA1_en*, *SHA2_en* inputs the co-processor is programmed to enable operation of the appropriate algorithm. In the case that the three latter signals bits are set to 0 HSSEC doesn't process data. The MODE input is responsible for indicating the operation mode of AES-128, selecting between the ECB (Electronic Code Book) and the CBC (Cipher Block Chaining). Also, signal Key indicates that a key is available. Finally, outputs *OUT hot*, *OUT AES*, *OUT SHA1/2* signal output of the processed data (ciphertext, message digest. Due to the limitation that only one of the three cryptographic blocks can take control of the data bus at every time instance, when *OUT_hot* is set to 1, then *OUT_AES* is set to 1 if it has the control of the output. In the case that *OUT_AES* is set to 0, then surely one of the SHA blocks drive the data bus, thus *OUT_SHA1/2* is sufficient to distinguish the originator of the message digest.

In Fig. 2 the architecture of HSSec is offered in detail. As it can be seen there is a central Control Unit (CU) to manage data processing and communication with the rest of the world. The processing blocks that implement the three algorithms are placed in parallel and share a common global data bus (64-bits). The same bus provides data to the memory block (considering memory hierarchy shown in Fig. 2 as one block and to the Key Scheduler. The Key Scheduler block is responsible for the key expansion of AES-128 and the generation of the message schedules. Additionally it provides the constants of the SHA hash functions. In order to produce a new round key, two transformations have to be performed in the Key Scheduler, *RotWord* and *SubWord*. The first one simply cyclically shifts the bytes of the first 32-bit word of the previous key by one position to the left. SubWord on the other hand performs the SubBytes transformation to each byte of the rotated word. Simple bitwise xor is needed in order to produce the final round key.

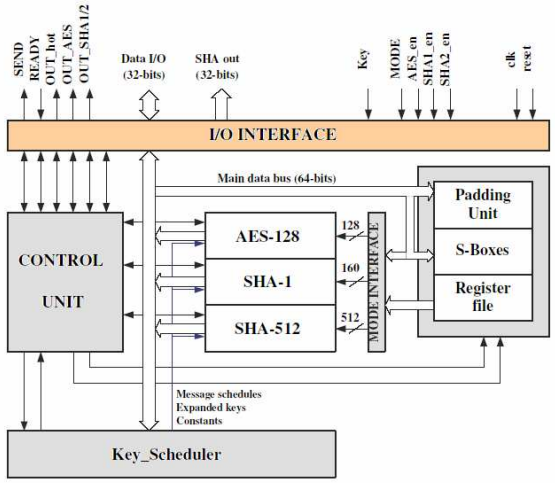

Fig. 2. HSSec core architecture

3.1 Area requirements and performance

The HSSec architecture has some key characteristics. HSSec's area is kept low due to the sharing of the memory block (including the padding unit and the temporal registers. Design complexity has kept also low in order to gain the regularity this architecture offers. In general the main benefit in terms of area requirements, is the reduction of each common but unshared resource (triplication) to one but fully shared. In the case of performance, the parallelism of the three cryptographic blocks and the selection of high-speed design approaches offered the noteworthy throughput of 1 Gbps for every cryptographic block. In fact, it is the critical path of the SHA-512 that limits performance of HSSec to this level. In the case of multiple clocks throughput could even exceed the 2 Gbps.

3.2 Memory organization

Successful selection of the memory organization is significant for the correct operation of HSSec. The Memory Block is organized in three main parts. The

first part is a collection of registers that store the initialization values for the three cryptographic algorithms. The second part is a general-purpose register file where temporal values are stored for quick access. The third part is the padding unit which is responsible for storing the fetched data.

This unit has been organized in 8 banks of 128-bits wide. Each bank corresponds to the minimum data required by one of the algorithms (AES-128) as input. Due to the regularity of the system, 4 consecutive banks form the input message for the SHA-1 algorithm, while 8 consecutive banks form the input message for the SHA-512 algorithm. Thus, 8 banks of memory (128-bits each) are forming the padding unit allowing high-speed processing of the data. In Fig. 3 the memory organization is illustrated.

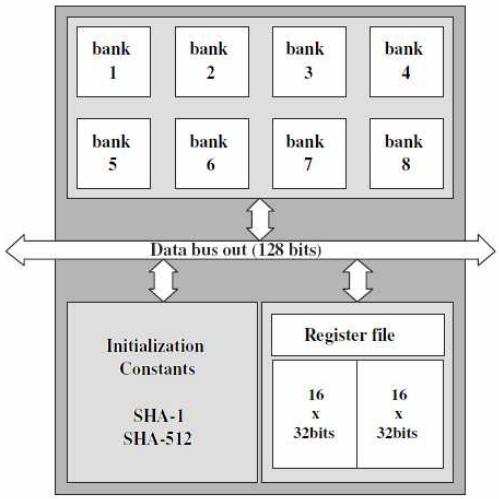

Fig. 3. Memory organization of HSSec.

3.2 Synchronization of HSSee

The main benefit of the HSSec is the synchronization of the three cryptographic blocks. The exploration of the three algorithms helped significantly before entering the system capture process. The selection of the appropriate design parameters and proposed approaches was the key for successful co-operation of the three cryptographic blocks without synchronization problems or tricky solutions.

Selecting an AES-128 design that offered high-speed ciphering in 10+1 clock cycles set the first limiting factor that had to be followed in all the design process and ensure that it was guaranteed. Thus, 128 bits of data had to be processed in at least 10 clock cycles. Considering that the 32-bit input of HSSec required 4 clock cycles to fetch the data in the padding unit, the first limitation seemed to be guaranteed. Furthermore, after the fetching of the first 128 bits, Control Unit was responsible to pre-fetch data, through the I/O interface, 4 clock cycles before the calculation of the ciphertext. The output of the ciphertext is performed through the Data I/O of the I/O Interface. Due to the width of the output port (32-bits) there is the need for 4 clock cycles (128 bits= 4 x 32 bits) to successfully send the whole ciphertext. The architecture of HSSec ensures that there is at least 10 clock cycles that interpose between successive calculations of two ciphertexts. Thus, there is no violation of the synchronization of the system (4 cycles to pre-fetch data and 4 cycles to output ciphertext is less than the 10 cycles available).

In the case of the two hash functions, synchronization is much simpler. Input of the two cryptographic blocks SHA-1 and SHA-512 is provided through the very same Data I/O port that is used for the AES-128. Data are padded in the memory block in banks of 128 bits. This size was selected to satisfy primarily the AES-128 algorithm. However, it was observed later on that this size was ideal for all the three algorithms. Thus, 4 banks are required for the input message of SHA-1 (512 bits input) and 8 banks are required for the corresponding input message of SHA-512 (1024 bits input). The regularity of the architecture appeared in this exact point. However, there was a significant issue that found its solution almost instantly. SHA-1 and SHA-512 require exactly the same number of clock cycles to generate the message digest. However, this is unacceptable because of the processed data. SHA-512 calculates its message digest for 1024 bits in 80 cycles and thus SHA-1 has to generate in the same time two message digests, one for each 512-bits block. The solution was instant and the initial limitation that was posed for the AES-128 was extremely handful. Thus, knowing that each bank (128-bits) are processed in at least 10 cycles, then there are 40 clock cycles to process the SHA-1 input message. In [14] there is a structure of a high-speed SHA-1 that performs message digest calculation in 40 cycles. Regarding SHA-512, time to process 8 banks (1024 bits) is at least 80 cycles, exactly the required number of cycles to generate the message digest.

4 Implementation Results

The HSSec cryptographic co-processor was captured 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 [18] does not infer dual-ported BlockRAMs. The Xilinx ISE

5.2 tools have been used for the implementation of the design and ModelSim was the simulation environment (for both logic and timing simulation).

Xilinx Virtex-II XC2V1000bg575 (speed grade -5) has been chosen as the target device. This device has 5120 CLB Slices which provide a total of 10240 LookUp Tables and 11224 Dffs or latches. Total memory available in terms of BlockRAM is 40 blocks of 16kbits each. Detailed analysis of the Virtex-II structure may be found in [19]. Table 1 presents detailed implementation results for the HSSec cryptographic co-processor. The implementation was tested in depth both functionally and operationally. There was no report for erroneous or generally unexpected behavior.

TABLE 1
Implementation Results

Device Utilization (XC2V1000bg575 -5)			
Resources	Used	Available	(%)
IOs	295	328	90
CLB Slices	3213	5120	63
BlockRAMs	9	40	23
Timing Report (XC2V1000bg575 -5)			
Clock Freq. (MHz)		80.21	
Clock cycle (ns)		12.47	
Throughput (Mbps)			
AES-128		1026.68	
SHA-1		1026.68	
SHA-512		1026.68	

5 Conclusion

HSSec is a cryptographic co-processor, specially designed to support security systems requiring AES-128 ciphering parallel to SHA-1 and SHA-512 hashing. The latter algorithms are commonly used in network applications and have been selected for the upcoming protocols JPSec [2] and P1619 [3]. The architecture of HSSec allows parallel operation of the three cryptographic primitives. Special design effort was given to benefit the system with regularity, which enabled low-area requirements. There are no synchronization issues due to the appropriate selection of the AES-128, SHA-1 and SHA-512 implementations.

Area requirements have been kept low. However this didn't had a bad effect to the coprocessors performance. HSSec was implemented for XILINX Virtex-II FPGA device family. Its operation and functionality have been validated and the implementation was evaluated. Area was kept low, as expected, while the three cryptographic primitives performed a 1Gbps throughput. The achieved throughput can be further increased using a multi-clock design strategy, which however would affect the characteristics of HSSec.

6 Acknowledgement

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] IP Security Protocol Charter (IPSEC). Internet Drafts for IPSec. http://www.ietf.org/html.charters/ipsec-charter.html
[2] ISO/IEC 15444-1/IUT-T T.800. International standard, 2000.
[3] IEEE standard P1619. Standard Architecture for Encrypted Shared Storage Media (draft). http://siswg.org/.
[4] J. Daemen, V. Rijmen, The Design of Rijndael, Springer-Verlag, 2002.
[5] National Institute of Standards and Technology, standard 197. The Advanced Encryption Standard (AES), 2001.
[6] National Institute of Standards and Technology, standard 180-2. Secure Hash Standard (SHA), 2002.
[7] ALMA Technologies. http://www.alma-tech.com
[8] Bisquare Systems Private Ltd. http://www.bisquare.com
[9] Helion Technology Ltd. http://www.heliontech.com
[10] Intron, Ltd. http://www.lviv.uar.net/ intron/
[11] Ocean Logic Ltd. http://www.ocean-logic.com
[12] Amphion. http://www.amphion.com/index.html
[13] National Institute of Standards and Technology, Special Publication 800-38. Recommendation for Block Cipher Modes of Operation, 2001.
[14] H. Michail, A.P. Kakarountas, O. Koufopavlou, C.E. Goutis, "A Low-Power and High-Throughput Implementation of the SHA-1 Hash Function", in Proc. of IEEE 2005 International Symposium on Circuits and Systems (ISCAS'05), Kobe, Japan, pp. 4086-4089, May 2005.
[15] I. Yiakoumis, M. Papadonikolakis, H. Michail, A.P. Kakarountas, C.E. Goutis,

"Efficient small-sized implementation of the keyed-hash message authentication code", in Proc. of IEEE 2005 EUROCON as finalist in the IEEE Region 8 Best Student Paper Contest, Belgrade, Yugoslavia, pp. 1875-1878, Nov. 21-24, 2005.

[16] S. Mangard, M. Aigner, S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture", IEEE Trans. on Computers 52(4) (2003) 483–491.

[17] A. Brokalakis, A.P. Kakarountas, C.E. Goutis, "A High-Throughput Area Efficient FPGA Implementation of AES-128 Encryption", in Proc. of IEEE 2005 International Workshop on Signal Processing Systems (SiPS'05), Athens, Greece, pp. 116-121, Nov. 2-4, 2005.

[18] Xilinx 5 Software Manuals. Synthesis and Simulation Design Guide. Xilinx, Inc, 2002.

[19] Xilinxs Virtex-II Platform FPGAs Datasheets.