Efficient Small-Sized Implementation of the Keyed-Hash Message Authentication Code

Ioannis Yiakoumis, Markos Papadonikolakis, Harris Michail, *Student Members, IEEE*, Athanasios P. Kakarountas, *Member, IEEE* and Costas E. Goutis, *Member, IEEE*

***Abstract* — A design approach to create small-sized high-speed implementations of the Keyed-Hash Message Authentication Code (HMAC) is presented. The proposed implementation can either operate in HMAC-MD5 and/or in HMAC-SHA1 mode. The proposed implementations do not introduce significant area penalty. However the achieved throughput presents an increase compared to commercially available IP cores that range from 30%-390%. The main contribution of the paper is the increase of the HMAC throughput to the required level to be used in modern telecommunication applications, such as VPN and the oncoming 802.11n.**

***Keywords* — Security, message authentication, hash function, high-speed HMAC, VLSI implementation.**

I. Introduction

HASH functions are common and critical cryptographic primitives. Their primary application is combined use with public-key cryptosystems in digital signature schemes. The most widespread functions are SHA-1 (Secure Hash Algorithm- 1) [1], and MD5 (Message Digest) [2]. These two hash functions are widely known for being used in the Keyed-Hash Message Authentication Code (HMAC) [3], which is met in numerous communication applications, to address authentication issues.

The SHA-1 hash function was selected for the Digital Signature Algorithm (DSA), as specified in the Digital Signature Standard (DSS) [4], and whenever a secure hash algorithm is required for federal applications. The latter hash functions are used widely in the field of communications, where until nowadays throughput of the cryptographic systems was not required to be high. However, since the use of the HMAC in the IPSec [5], e-payment and VPN applications, the throughput of the cryptographic system, especially the server, has to reach the highest degree of throughput. Especially in applications that transmission and reception rates are high, any latency or delay on calculating the digital signature of the data packet leads to degradation of the network's quality of service. Software implementations are presenting unacceptable performance for high-speed applications. Also poor performing and bulk implementations of HMAC IP cores are met even nowadays in the market.

The latter facts were a strong motivation to propose a novel hardware implementation of the HMAC. Thus, this paper aims to provide a low-cost design approach, compared to the proposed solutions from both academia and industry, in order to satisfy the requirements of the new communication applications. It introduces a negligible area penalty; increasing the throughput and keeping the area small enough as required by most portable communication devices. The main contribution of this work is the design approach to optimize performance without introducing extra area.

The rest of this paper is organized as follows. In Section II, the proposed HMAC architecture is presented. Section III presents the proposed implementation in depth along with the two hash functions, providing details regarding the architecture, the logic and the modifications to decrease the critical path, and the characteristics of the circuit that are expected. In Section IV the proposed HMAC is implemented for an FPGA technology and it is compared to other implementations. Finally, in Section V the paper concludes.

II. The HMAC Algorithm

The HMAC standard [3] defines a special mechanism that guarantees message authentication for transmission through a non-secure communication channel. The main idea is the use of a cryptographic hash function (usually the MD5 or SHA-1). The purpose of the HMAC is to authenticate both the source of a message and its integrity. The main parameters of the HMAC are the message input and the secret key, which is known only to the message originator and the intended receiver(s). The main function of the HMAC is the generation of a value (the MAC), formed by condensing the message input and the secret key. The MAC value is sent along with the message and the receiver has to evaluate that the received message generates the received MAC value, using the secret key which is agreed between the message originator and the receiver. The final MAC value is given by the expression shown in (1), where text is the plain text of the message, *K*

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

I. Yiakoumis, M. Papadonikolakis, and H. Michail are students. They are with the Dpt. of Electrical & Computer Engineering, University of Patras, Greece (phone: +30 2610-997-324; fax: +30 2610-994-798; e-mail: {yiakoumi, markos, michail} @ee.upatras.gr).

A. P. Kakarountas and C.E. Goutis are with the Dpt. of Electrical & Computer Engineering, University of Patras, GR-26500, Patras, Greece (phone: +30 2610-997-283; fax: +30 2610-994-798; e-mail: {kakaruda, goutis} @ee.upatras.gr).

is the secret key and K_0 is K appended with zeros to form a $\mathrm{mod}_{32}(n)$ byte key, i_{pad} and o_{pad} are predefined constants, and $\oplus$ is bitwise XOR.

$$HMAC(K,text) = H\big((K_0 \oplus ipad)\|H((K_0 \oplus opad)\|text)\big) \quad (1)$$

III. Proposed HMAC Implementation

The architecture of the proposed HMAC offers a significant benefit concerning the maximum achieved operation frequency. The critical path is observed to the Hash Core block, where the hash functions are implemented. This allows design effort to be focused on the Hash Core and the optimization of the hash functions' critical path. Following, the two hash functions are presented, along with some critical optimizations on the critical path. Solutions are offered for applications that require either sole HMAC-MD5 or -SHA1, or a combined HMAC-MD5-SHA1.

A. SHA-1 Hash Function

The SHA-1 hash function is an iterative algorithm that requires 80 transformation steps to generate the final hash value (Message Digest – MD). In each transformation step, a hash operation is performed that takes as inputs five 32-bit variables (a,b,c,d,e), and two extra 32-bit words (one is the message schedule, W_t, which is provided by the Padding Unit, and the other word is a constant, K_t, predefined by the standard). The calculations that take place in each operation (clock cycle t) are described below in (2), where $\mathrm{ROTL}_x(y)$ represents rotation of word y to the left by x bits and $f_t(z,w,v)$ represents the non-linear function associated to clock cycle t.

$$\begin{aligned} e_t &= d_{t-1} \\ d_t &= c_{t-1} \\ c_t &= ROTL_{30}(b_{t-1}) \\ b_t &= a_{t-1} \\ a_t &= ROTL_5(a_{t-1}) + f_t(b_{t-1},c_{t-1},d_{t-1}) + K_t + W_t \end{aligned} \quad (2)$$

The linear function f_t is changing every 20 cycles. Thus, the SHA-1 is divided in four rounds of 20 identical operations, based on the used non-linear function. The hash value resulted from the 80 iterations is a 160-bit MD.

From (2), it is easily extracted that the critical path is located in the calculation of at, which is equal to the delay of three Carry-Propagate Adders (CPA). Several implementations that have been proposed for the SHA-1 hash function [6],[7], do not pay the appropriate design effort on this critical notice, considering that the synthesis tool will find the optimal solution. However in [8] there is a design approach that tries to exploit the characteristics of the Carry Save Adder (CAS) in order to minimize the critical path.

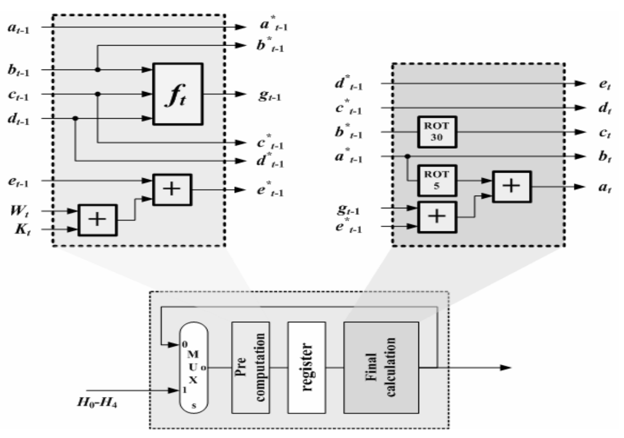

Fig. 1. The modified SHA-1 operation block, separated in two calculation phases.

The proposed design approach to optimize the critical path, exploits the fact that a_t is calculated using the inputs of cycle t-1. Thus, some intermediate values can be pre-computed, stored in a register and used without introducing any delay. So, (2) is transformed in (3) in order to reduce the critical path. Some observations can be made analyzing (3). First, the introduced area penalty is a single register that stores the intermediate value g_{t-1}. Second, power dissipation is kept low and almost the same to that of the initial implementation. The extra power dissipation is that of the read/write operations of the introduced register. On the other hand, the paths are shortened and balanced, reducing the glitches and the dynamic power dissipation on the circuit's wires. The new operation block of the SHA-1, as resulted from the application of the pre-computation stage is illustrated in Fig. 1, and presents a delay of two adders, synthesized as a CSA and a CPA. The introduction of this pre-computational stage is a novel design approach.

$$\begin{aligned} &e'_{t-1} = e_{t-1} + K_t + W &&, e_t = d'_{t-1} \\ &d'_{t-1} = d_{t-1} &&, d_t = c'_{t-1} \\ &c'_{t-1} = c_{t-1} &&, c_t = ROTL_{30}(b'_{t-1}) \\ &b'_{t-1} = b_{t-1} &&, b_t = a'_{t-1} \\ &a'_{t-1} = a_{t-1} &&, a_t = ROTL_5(a'_{t-1}) + e'_{t-1} + g_{t-1} \\ &g_{t-1} = f_t(b_{t-1},c_{t-1},d_{t-1}) \end{aligned} \quad .(3)$$

B. MD5 Hash Function

MD5 is an improved version of MD4, which addresses several known successful attacks on MD4 [9]. As in SHA-1, MD5 focuses on the transformation of an initial input, through iterative operations. MD5 produces a 128-bit MD, instead of the 160-bit hash value of SHA-1. Additionally, there are still four rounds, consisting however of 16 operations each. There are four 32-bit (a,b,c,d) inputs and two extra 32-bit values (one is the message schedule, M_t, which is provided by the Padding Unit, and the other word is a constant, L_t, predefined by the standard) that are transformed iteratively to produce the final MD. The calculations that take place in each operation (clock cycle t) are described below in (4), where $fn_t(z,w,v)$ represents the non-linear function associated to clock cycle *t*. Rotation in (3) is performed for *s* positions, which varies from cycle to cycle and is pre-defined by the standard [2].

$$
\begin{aligned}
d_t &= c_{t-1} \\
c_t &= b_{t-1} \\
b_t &= b_{t-1} + ROTL_s\left(a_{t-1} + fn_t\left(b_{t-1}, c_{t-1}, d_{t-1}\right) + M_t + L_t\right) \\
a_t &= d_{t-1}
\end{aligned} \quad (4)
$$

The critical path is located on the calculation of a sole output, b_t. A pre-computational stage can be applied also to this hash function to reduce the critical path.

C. HMAC Implementation Scenarios

As already mentioned, HMAC can be implemented either including one sole hash function, or two hash functions combined to operate when selected. Also, both SHA-1 and MD5 hash functions have an identical parameter; they both have four discreet rounds. The above offer a wide range of characteristics of the HMAC implementation that if exploited wisely, can give solutions depending on the nature of the application.

1) Rolling Loop Technique: If the critical parameter is small area, a rolling loop technique can be applied. As illustrated in Fig. 3, the output of the operational block is fed back to the input through pre-computation stage. Notice that the main benefit of the insertion of the pre-computation stage is that a_t, which is the output of the final calculation block, enters the pre-computation stage as the new a_{t-1}, which is a wire directly connected to the register. This technique allows small-sized implementations through re-use of the same configurable operation block. Configurability issues have to address correct selection of the non-linear function for both hash functions and the rotate positions for the case of MD5.

2) Pipeline Technique: If the critical design parameter is performance, with a more relaxed area constraint, then pipeline can be applied. As already mentioned a common characteristic of the two hash functions is the four rounds. Thus, applying a pipeline stage to every round, result in quadruplicating of the achieved throughput. This technique exploits small-sized implementations based on rolling loop and the characteristic of the four rounds to result in relatively small sized implementations, achieving throughput four times higher than the limit imposed by the design of the operation block of the hash function.

3) Sole Hash Function: In the case of implementing HMAC-MD5 or HMAC-SHA1, the throughput is directly associated to the maximum operating frequency of the hash function's operation block. The proposed modifications of the two hash function reduce significantly the critical path making. As shown in the next section, the implementation of HMAC-MD5 or HMAC-SHA1, using the pre-computational stage, scores a 30% increase of throughput, if no pipeline is applied.

4) Co-Existence of the Two Hash Functions: In many applications there is the need for the selective use of SHA-1 or MD5. There are two design approaches for co-existence of the two hash functions. The first is the implementation of the two hash functions as separate cores and selection through a multiplexer. Although this approach presents low design complexity it is not optimal for small-area requirements and power dissipation is also considerably high. The second design approach is the exploration of the two hash functions to locate resources that can be used by both functions. In this case area requirements are reduced and extra power dissipation is only a factor of the latter approach.

IV. Implementation And Results

Considering the latter mentioned implementation scenarios, several HMAC designs were implemented to verify and evaluate the value of the presented design approach. The designs were captured in VHDL and were fully simulated and verified using the Model Technology's ModelSim Simulator. A large number of test vectors were used to verify the designs' functionality, either adopted from the standarts [1],[2],[3], or randomly created. The XILINX FPGA technologies were selected as the targeted technologies, synthesizing the designs for the VIRTEX, VIRTEX-II, VIRTEX-E and SPARTAN3 device families. The selection of the technologies was based on the available information from commercial IP cores [10],[11],[12],[13],[14],[15] and research works [6],[7],[8]. The synthesis tool used to port VHDL to the targeted technologies was Synplicity's Synplify Pro Synthesis Tool. Simulation of the designs was also performed after synthesis, exploiting the back annotated information that was extracted from the synthesis tool. Further evaluation of the designs was performed using the prototype board for the Xilinx Virtex-E device family. The FPGA device on the board is an XCV1600EBG560.

A. Results of the Implementations

In Table I, the characteristics of the HMAC implementations are offered. Only implementations of the Virtex-E FPGA family were fully verified and numbers reflect experimental results. The results of the rest FPGA technologies are provided as reported from the Synplicity's synthesis tool. The implementation of the combined hash functions is considered for two target design parameters, performance optimized which uses implementation of two separate cores and selection through a multiplexer, and area optimized which exploits commonly re-used primitives. The reported throughput corresponds to a design approach with rolling loop technique applied but without pipeline. If pipeline technique is applied then throughput is quadrupled and the area is increased by 3.21 times in average. Notice that is the first time that an implementation without pipeline exceeds 1 Gbps in Virtex-II FPGA technology.

TABLE I
CHARACTERISTICS OF THE PROPOSED HMAC IMPLEMENTATIONS FOR THE TARGETED FPGA TECHNOLOGIES

HMAC	Slices	Op.Frequency (MHz)	Throughput (Mbps)
Xilinx Virtex (-6)			
SHA-1	686	91	582.4

HMAC	Slices	Op.Frequency (MHz)	Throughput (Mbps)
MD5	612	55	440.0
SHA-1 & MD5 (perf.)	1100	55	352.0 440.0
SHA-1 & MD5 (area)	780	53	339.2 424.0
Xilinx Virtex-II (-6)			
SHA-1	854	162	1036.8
MD5	797	96	768.0
SHA-1 & MD5 (perf.)	1357	96	614.4 768.0
SHA-1 & MD5 (area)	982	81	518.4 648.0
Xilinx Virtex-E (-8)			
SHA-1	686	111	710.4
MD5	612	65	520.0
SHA-1 & MD5 (perf.)	1100	65	416.0 520.0
SHA-1 & MD5 (area)	780	61	390.4 424.0
Xilinx Spartan-3 (-4)			
SHA-1	750	87	543.0
MD5	665	62	480.0
SHA-1 & MD5 (perf.)	1090	62	387.0 480.0
SHA-1 & MD5 (area)	882	43	275.2 344.0

In Table II the characteristics of the commercial HMAC IP cores are reported in order to make comparison Additionally, the characteristics of [6],[7],[8] SHA-1 implementations (not HMAC-SHA1) are reported in order to make a fair comparison. Recall that the results in Table I are for implementations with no pipeline stages. The work in [7] reports four pipeline stages, so the appropriate anagogic function has to be performed in order to calculate the characteristics of the proposed implementation. Every single SHA-1 implementation (not an HMAC-SHA1) is marked with a * at the start. These designs are offered as a reference, due to the explicit dependency of the maximum operating frequency of the HMAC from the critical path of the used hash function. Analyzing the performance of the implementations presented in Table II, it can be observed that throughput of the proposed HMAC implementations exceeds those of the available commercial IP cores by 30%- 390%.

TABLE II
CHARACTERISTICS OF THE PROPOSED HMAC IMPLEMENTATIONS FOR THE TARGETED FPGA TECHNOLOGIES

HMAC	Slices	Op.Frequency (MHz)	Throughput (Mbps)
Xilinx Virtex (-6)			
*SHA-1 [6]	1004	43	119.0
*SHA-1 [6]	1004	43	119.0
*SHA-1 [7]	2245	55	1339.0 (334.8)
*SHA-1 [8]	-	86	530.0
SHA-1 [10]	686	70	442.0
Xilinx Virtex-II (-6)			
*SHA-1 [12]	573	140	874.0
*SHA-1 [14]	612	79	498.1
*MD5 [12]	613	96	744.0
*MD5 [14]	614	62	488.3
*MD5 [15]	844	60	472.0
SHA1 & MD5 [12]	888	95	593.0 736.0
Xilinx Virtex-E (-8)			
*SHA-1 [13]	716	71	449.0
*SHA-1 [14]	612	72	451.9
*MD5 [14]	605	50	393.8
SHA-1 [11]	579	66	422.4
MD5 [11]	324	50	400.0
Xilinx Spartan-3 (-4)			
*SHA-1 [12]	677	87	543.0
*MD5 [12]	630	63	488.0
MD5 [11]	630	38	304.0
SHA-1 & MD5 [12]	1010	62	387.0 480.0

The designs that are marked with a '*' are indicating implementations of the described hash functions, not HMAC implementations.

V. CONCLUSION

A novel design approach for the development of small sized and high-speed HMACs was presented in this paper. It was showed that the critical path can be further reduced, by exploiting special properties of the included hash functions. The proposed implementations were expected to present at least 30% higher throughput than any other available implementation. Significant design effort was paid to keep area low. The experimental results showed that a negligible area penalty was introduced for achieving an increase in throughput that ranged from 30%-100% compared to the competing implementations. Finally the design was fully tested and verified for the Xilinx Virtex-E FPGA family using a prototype board.

REFERENCES

[1] *Secure Hash Standard (SHS)* (Standard). National Institute of Standards and Technology (NIST). FIPS PUB 180-2 Standard, 2002.

[2] R.L. Rivest, "The MD5 Message Digest Algorithm," in IETF Network Working Group, RFC 1321, 1992.

[3] *The Keyed-Hash Message Authentication Code (HMAC)* (Standard). National Institute of Standards and Technology (NIST). FIPS PUB 198 Standard, 2002.

[4] *Digital Signature Standard (DSS)* (Standard). National Institute of Standards and Technology (NIST). FIPS PUB 186-2, 2000.

[5] IP Security Protocol Charter (IPSEC). Internet Drafts for IPSec. Available: http://www.ietf.org/html.charters/ipsec-charter.html

[6] S. Dominikus, "A Hardware Implementation of MD-4 Family Hash Algorithms," in *Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS'02)*, 2002, pp. 1143–1146.

[7] N. Sklavos, G. Dimitroulakos, and O. Koufopavlou, "An Ultra High Speed Architecture for VLSI Implementation of Hash Functions," in *Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS'03)*, 2003, pp. 990–993.

[8] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott, "Comparative Analysis of the Hardware Implementations of Hash Functions SHA-1 and SHA-512," in *Proc. Information Security Conference (ISC'02)*, Springer-Verlag, Heidelberg, 2002, pp. 75–89.

[9] B. den Boer, and A. Bosselaers, "An attack on the last two rounds of MD4," in *Proc. of CRYPTO, Advances in Cryptology (CRYPTO '91)*, Springer Verlag, Berlin, Heidelberg, 1992, pp. 194-203.

[10] ALMA Technologies. Available: http://www.alma-tech.com

[11] Bisquare Systems Private Ltd. Available: http://www.bisquare.com

[12] Helion Technology Ltd. Available: http://www.heliontech.com

[13] Intron, Ltd. Available: http://www.lviv.uar.net/~intron/

[14] Ocean Logic Ltd. Available: http://www.ocean-logic.com

[15] Amphion. Available: http://www.amphion.com/index.html