High Throughput Hardware/Software Co-Design Approach for SHA-256 Hashing Cryptographic Module In IPSec/IPv6

H. E. Michail[1] G. S. Athanasiou[2] A. A. Gregoriades[3]

Ch. L. Panagiotou[2] C.E. Goutis[2]

GJCST Classification
D.4.6, C.2.2, K.6.5

***Abstract*- Nowadays, more than ever, security is considered to be critical issue for all electronic transactions. This is the reason why security services like those described in IPSec are mandatory to IPV6 which will be adopted as the new IP standard the next years. In fact E.U. has set the target of moving to IPv6 for about 25% of European e-infrastructures in 2010. However the need for security services in every data packet that is transmitted via IPv6, illustrates the need for designing security products able to achieve higher throughput rates for the incorporated security schemes. In this paper a top-down methodology is presented which manages to increase throughput of SHA-256 hash function hardware design. The higher degree of throughput with limited area penalty and cost is achieved through appropriate Software/Hardware partitioning and design.**

Keywords- Hash-Functions, Hardware design, VLSI, High-Throughput, IPSec, SHA-256.

I. Introduction

Security is now considered as a must-have service for almost all kind of e-applications. This is the reason why in IPv6 which is bound to be adopted worldwide, IPSec [1] is a mandatory protocol. IPSec (Internet Protocol Security) is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a data stream. IPSec also includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPSec can be used to protect data flows between a pair of hosts (e.g. computer users or servers), between a pair of security gateways (e.g. routers or firewalls), or between a security gateway and a host. In IPSec and in other applications like keyed-hash message authentication codes (HMACs) [2], the Secure Electronic Transactions (SET), and the 802.16 standard for Local and Metropolitan Area Networks incorporate authenticating services, an authenticating module that includes a hash function is nested in the implementation of the application. Moreover, digital signature algorithms are used for authenticating services in electronic mail, electronic funds transfer, electronic data interchange, software distribution, data storage etc are based on using a critical cryptographic primitive like hash functions.

Hashes are used also in SSL [3], which is a Web protocol for establishing authenticated and encrypted sessions between Web servers and Web clients.

However, in these specific applications there is an urgent need to increase their throughput, especially of the corresponding server of these applications and this is why, as time goes by, many leading companies improve their implementations of hash functions. This is also true for IPv6/IPSec since corresponding designs and implementations should be able to achieve such a high throughput so as to be able to provide cryptographic services to all data packet that are transmitted via internet.

Although software encryption is becoming more prevalent today, hardware is the embodiment of choice for military and many commercial applications [4]. The NSA, for example, authorizes only encryption in hardware. This is because hardware designs are much faster than the corresponding software implementations [5], and because hardware implementations offer a higher level of security since they also provide physical protection [6].

The security scheme of these throughput-demanding applications like HMAC in IPSEC and SSL\TLS incorporate encryption and authenticating modules. Lately many implementations of the AES encryption module have been designed that exceed or approach 20 Gbps of throughput [7], so it is crucial to design hash functions that also achieve high throughput, and increase throughput of the whole IPSec and SSL\TLS security scheme.

The latter mentioned facts were strong motivation to propose a novel methodology for hardware design and implementation applicable to SHA-256 hash function [8] which will dominate in the near future. However, with minor modifications, the proposed methodology can also be applied to other hash functions leading also to much higher throughput designs with small area penalty.

As a case study, the efficient design and mapping of IPSec components in a reconfigurable platform is illustrated. This way, in abstract level, the generic formulation of a platform

*About[1]-H. E. Michail is with: a) Department of Electrical and Computer Engineering and b) Department of Computer Engineering and Informatics, University of Patras, Rio Campus, Patras, Greece, c) Department of Mechanical Engineering, Technological and Educational Institute of Patras, Patras, Greece(telephone:302610997321,
(email: michail@ece.upatras.gr)
About[2]- G. S. Athanasiou, Ch. L. Panagiotou and C. E. Goutis are with the Department of Electrical and Computer Engineering, University of Patras, Rio Campus, Patras, GR-26500, Greece
(email: gathanas, chpanag, goutis @ece.upatras.gr)
About[3]-A. A. Gregoriades is with the Department of Computer Science and Engineering, European University of Cyprus, Nicosia, Cyprus,
(email: A.Gregoriades@euc.ac.cy)*

aiming to boost performance of IPSec with low cost is illustrated. Only the critical kernels/components of IPSec are mapped for execution on the (expensive) reconfigurable logic.

II. Hashing and Related Work

Hash functions are iterative algorithms and their operation block (in fact the "hashing machine" of the algorithm), is responsible to process the message schedule. Usually it consists of simple functions like additions, rotations and/or Boolean logic functions. In SHA-256 the operation block is repeated 64 times, feeding its output as input to the consecutive operation block and then the final hash value is ready.

The need for high throughput is widely recognized and thus various design approaches have been proposed in order to introduce to the market high-speed and small-sized hashing cores such as loop unrolling, pipeline, re-use resource and usage of newer and faster FPGAs [9].

Nevertheless the performance of all hardware implementations is degraded because not much effort has been paid on optimizing the inner logic of the transformation rounds. In our work we propose a methodology to optimize the inner logic of SHA-256 hash function so as to reach the highest level of throughput, with minor area penalty which in turn will lead to achieving a higher throughput for the whole security scheme (i.e. in IPSec).

III. Hardware/Software Co-Design

In Fig.1, an overview of the reconfigurable system-on-chip (SoC) architecture considered in this work is shown. The platform is composed by a Reconfigurable Functional Unit (RFU) like an FPGA and an embedded CPU.

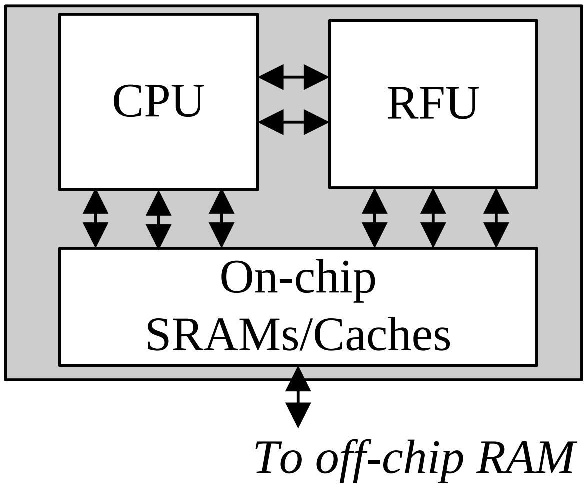

Fig.1. Reconfigurable SoC Architecture.

The RFU is a Coarse-grained Reconfigurable Array (CRA). On-chip memories (SRAMs, caches or combination of them) store program code, CRA configurations and data. Local data and instruction (configuration) memories are located in both the CPU and in the CRA. The CRA acts as a coprocessor to the CPU and accelerates computational intensive software parts of the application. The embedded CPU, typically a RISC like an ARM or MIPS, executes control-dominant sequential parts.

The programming (execution) model of the reconfigurable platform considers that the data communication between the CRA and the CPU uses shared-memory mechanism. The shared memory is comprised of the system's on-chip data RAM and coprocessor data registers inside the RFU. The communication process used by the CPU and the CRA preserves data coherency by requiring their execution to be mutually exclusive. The mutual exclusive execution simplifies the programming since complicated analysis and synchronization procedures are not required.

If we consider the design of IPSec, as we have already mentioned the nested hash function is the limiting factor of its performance. So, the design and implementation of this hash function must be selected to be mapped on the RFU so as to be speeded-up, whereas the rest components can be executed on the CPU illustrated in Fig.1. Moreover certain blocks of SHA-256 hash function, pictured in Fig.2, like padding unit, control unit, message digest extraction etc. can also be assigned for execution on the CPU and not on the FPGA (CRA).

As long as the other basic component of IPSec is concerned (that is AES), from [7], it is derived that AES designs implementations present higher throughputs but also higher operating frequencies. Thus from all points of view SHA-256 is the limiting factor of the performance of the design and implementation of IPSec/IPv6 in reconfigurable Hardware.

Obviously the blocks assigned for execution on the CRA, thus the FPGA, is those which determine the critical path of the incorporated hash function. The critical path of the illustrated architecture is located between the pipeline stages and they are going to be mapped on the FPGA. However in order to boost performance of IPSec, we focus on reducing the critical path of the design mapped on the FPGA, so as to increase performance of the whole system. The optimization of the critical path is solely focused on the operation block, in order to reduce the delay and thus increase the operating frequency.

IV. Proposed Methodology

The generic architecture of a hash function is shown in Fig. 2. Due to the blocks' logic variation from round to round numerous implementations [10, 11, 12], are based on four pipeline stages of single operation blocks. Also from a heuristic survey [11] to hash functions it is clear enough that the best compromise is to apply four pipeline stages so as to quadruple throughput and keep the hash core small as well. This selection was made in the presented methodology as it is shown in Fig.2.

Exploring the generic architecture of Fig. 2 it is easily extracted that the critical path is located between the pipeline stages. The other units, MS RAM and the array of constants, do not contribute due to their nature (memory and hardwired logic respectively), while control unit is a block containing very small counters which also don't contribute to the overall maximum delay. Thus, optimization of the critical path should be solely focused on the operation block.

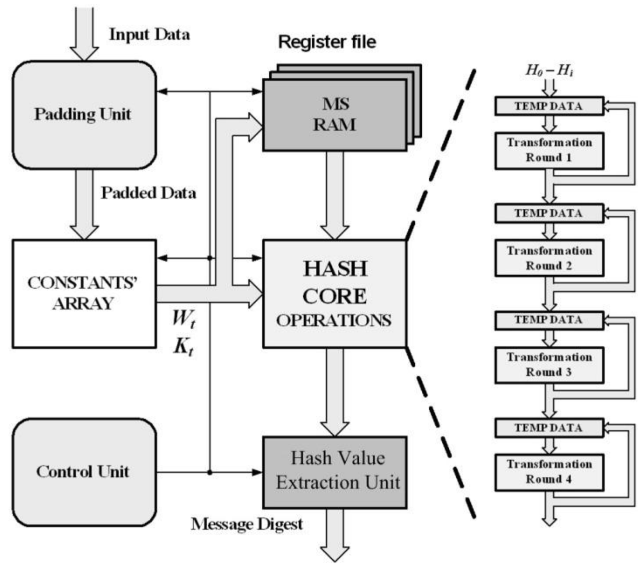

Fig.2. SHA-256 hash core architecture with 4 pipeline stages.

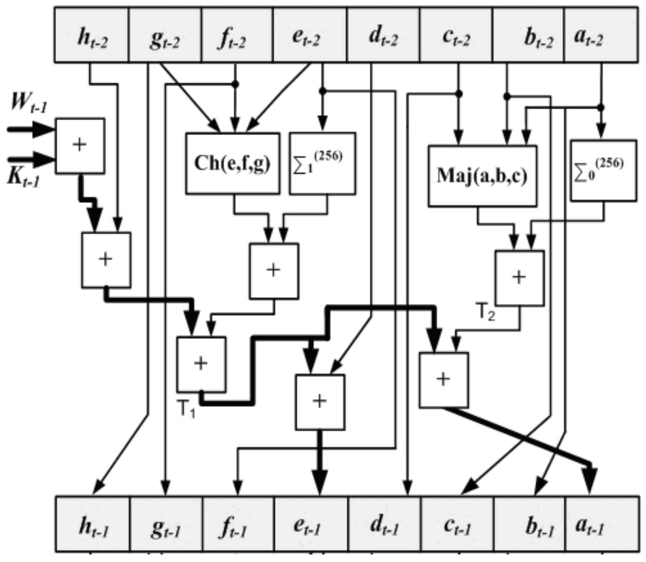
h_{t-2} g_{t-2} f_{t-2} e_{t-2} d_{t-2} c_{t-2} b_{t-2} a_{t-2}
W_{t-1}
K_{t-1}
Ch(e,f,g)
$\Sigma_1^{(256)}$
Maj(a,b,c)
$\Sigma_0^{(256)}$
T_2
T_1
h_{t-1} g_{t-1} f_{t-1} e_{t-1} d_{t-1} c_{t-1} b_{t-1} a_{t-1}

Fig.3. SHA-256 operational block

The operation block of SHA-256 is shown in Fig.3. The critical path (darker line) is located on the computation of at and et values that requires four addition stages and a multiplexer for feeding back the output data.

At the first step of our methodology, a number of operations are partially unrolled. That number is determined by a separate analysis on SHA-256 hash function. This analysis compares variations of partially unrolled operations, their corresponding throughput, the required area and then calculating the proper ratio (cost function). In Fig. 4, the results of a cost function analysis for SHA-256 algorithm, performed in Virtex-II FPGA family, are illustrated. As it is shown, selecting to partially unroll two operations results in the best achieved Throughput/Area ratio (ratio > 2).

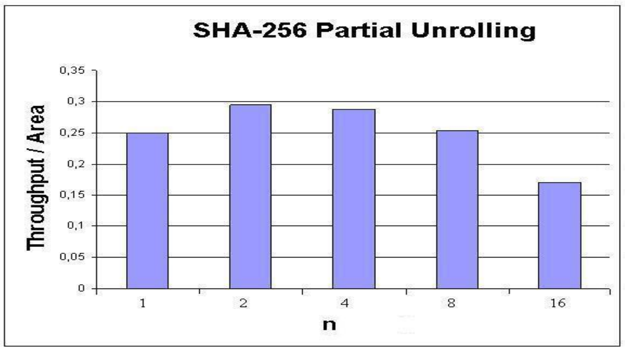

Fig.4. Effect of unrolling the operation blocks of SHA-256

In Fig. 5, the consecutive SHA-256 operation blocks of Fig. 3, have been modified so as to exploit parallel calculations. The gray marked areas on Fig. 5 indicate the parts of the proposed SHA-256 operation block that operate in parallel.

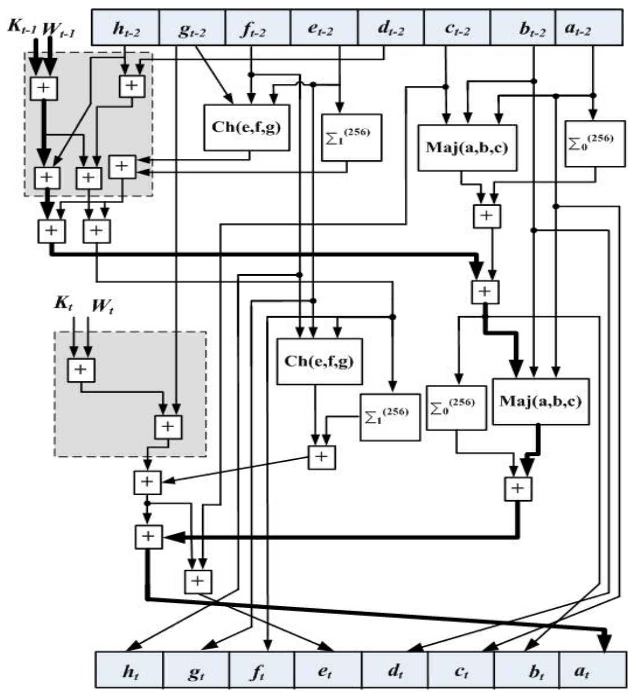
K_{t-1} W_{t-1}
h_{t-2} g_{t-2} f_{t-2} e_{t-2} d_{t-2} c_{t-2} b_{t-2} a_{t-2}
Ch(e,f,g)
$\Sigma_1^{(256)}$
Maj(a,b,c)
$\Sigma_0^{(256)}$
K_t W_t
h_t g_t f_t e_t d_t c_t b_t a_t

Fig.5. Two unrolled SHA-256 operation blocks.

It is noticed that two single addition levels have been introduced to the critical path that now consists of six addition stages needed for the computation of at and et values. Although, this reduces the maximum operation frequency, the throughput is increased significantly since the message digest is now computed in only 32 clock cycles (instead of 64). The area requirements are increased since more adders have been used in order to achieve the partial unrolling.

The next step of the proposed methodology has to do with the spatial pre-computation technique. Taking into consideration the fact that some outputs are derived directly from some inputs values respectively we can assume that it is possible during one operation to pre-calculate some intermediate values that will be used in the next operation. These pre-calculations are related only with those output values that derive directly from the latter mentioned input values. This pre-computation technique is applied on the partially unrolled operation block in Fig. 5 and the new modified operation block is shown in Fig. 6.

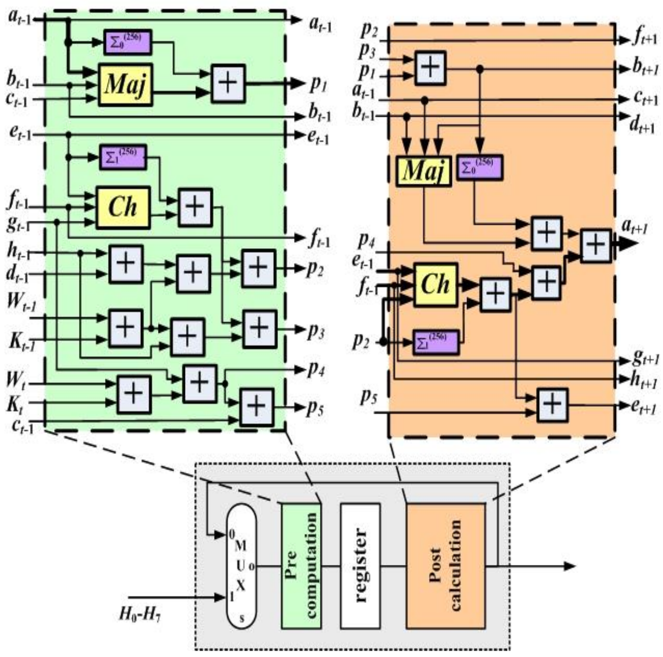

Fig.6. Partially unrolled operation block with pre-computed values.

Observing Fig. 6 it is noticed that the critical path is now located on the computation of the peripheral value p1 that is introduced in Fig. 6. The critical path has been reduced from six addition stages and a multiplexer to four addition stages, two non-linear functions (noted as Maj and Ch in Fig. 6) and a multiplexer. Comparing to the conventional implementation of the single operation block shown in Fig. 3, theoretically throughput has been in-creased by 80%-90%.

This has been achieved by pre-calculating some intermediate values and moving the pipeline registers to an appropriate intermediate point to store them. The new operation block now consists of two units, the “Pre-Computation” unit which is responsible for the pre-computation of the values that are needed in the next operation and the “Post-Computation” unit which is responsible for the final computations of each operation.

The third step of the proposed technique is to apply the system-level pre-computation so as to achieve data pre-fetching. It was noticed that all Wt values can be computed and be available for adequate time before they are really needed in each operation t since they are computed through some XOR bitwise operations. Also the values of the constants Kt are known a priori. These two facts give us the potential of pre-computing the sum Wt + Kt outside of the operation block. The sum is then saved into a register that feeds the operation block and thus the externally (regarding the operational block) pre-computed sum Wt + Kt is available at the beginning of each operation. So at the operational block, from now on it will be assumed that this sum available at the beginning of each operation and its computational time is excluded from the critical path. The new operational block is illustrated in Fig. 7.

Inspecting Fig. 7, we observe that the critical path is located on the computation of the peripheral value p1, and consists of four addition stages and two non-linear functions. However we notice that at the beginning of this path there is the value p4 that is pending to be added to a sum that at the same time is being calculated.

So for this case, a CSA can be used in order to add the three values in advance compared to the necessary time in case we used two adders as in Fig.7. The Carry Save Adder is applied on the “Post-Computation” unit as it is depicted in Fig.8 where we have also used a Carry Save Adder in the “Pre-Computation” unit. This way the critical path inside the operation block has been reduced to one Addition stage, two Non-linear functions and two Carry Save Adders that are required in order to compute the value p1.

The final proposed operation block for SHA-256 is illustrated in Fig. 8. It processes two operations in a single clock cycle, and the critical path is shorter than that of the conventional implementation, resulting in an increase of through-put of more than 110% (theoretical). The introduced area penalty is 3 adders, 4 Carry Save Adders, two 32-bit registers and 2 non-linear functions. The introduced area penalty is about 35% for the whole SHA-256 core compared to the conventional pipelined implementation. This corresponds to an area penalty of about 9% for the whole security scheme. This area penalty is worth paying for about 110% increase of throughput.

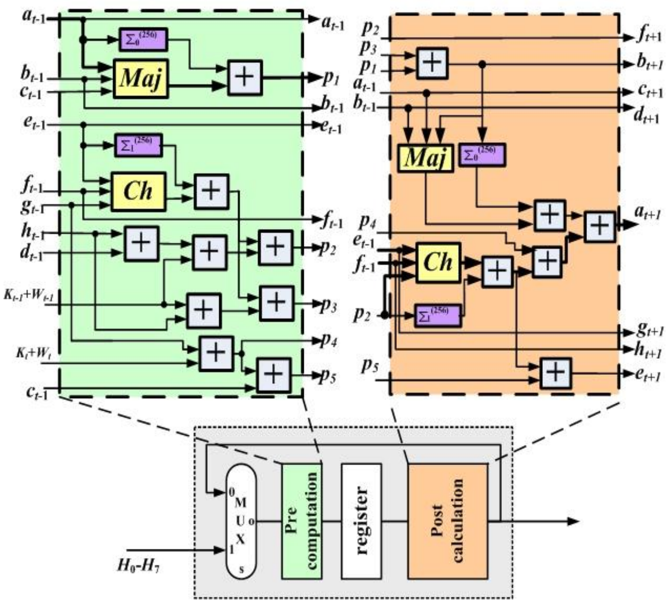

Fig.7. Partially unrolled operation block with pre-computed values for SHA-256 with pre-fetching of W+K values.

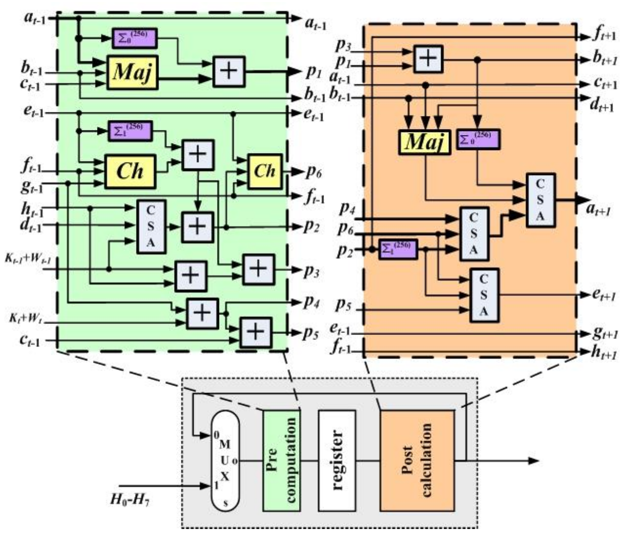

Fig.8. Proposed SHA-256 operation block.

V. Results and Comparisons

In order to evaluate the proposed methodology, SHA-256 hash function was captured in VHDL and was fully simulated and verified.

Table 1: Performance Characteristics and comparisons

SHA-256				
Implementation	Op. Freq. (MHz)	Throughput (Mbps)		Area CLBs
		Post-synthesis	Post Place & Route	
[13] [a]	42.9	77	-	1004
[14] [a]	88.0	87	-	1261
[11] [a]	83	326	-	1060
[15] [a]	82	646	-	653
[16] [a]	77	308	-	1480
[17] [a]	53	848	-	2530
Proposed [a]	**35.1**	**2210**	**2077**	**1534**
[15] [b]	150	1184	-	797
[18] [b]	133	1009	-	1373
[19] [b]	81	1296	-	1938
Proposed [b]	**52.1**	**3334**	**3100**	**1708**
[20 [c]	64	2052	-	1528
Proposed [c]	**36.4**	**2330**	**2190**	**1655**
[21] [d](Commercial IP)	96	-	756	945
[22] [e](Commercial IP)	-	-	1900	1614 (LUTs)
[23](Commercial IP)	133	-	971	asic

[a] *Virtex FPGA family*
[b] *Virtex II FPGA family*
[c] *Virtex-E FPGA family*
[d] *Virtex 4 FPGA family*
e Virtex 5 FPGA family

The XILINX FPGA technologies were selected as the targeted technologies, synthesizing the designs for the Virtex FPGA family.
To exhibit the benefits of applying the proposed design methodology, SHA-256 hash function was implemented following the steps of the proposed methodology and is compared with other existing implementations proposed either by academia or industry.
The results from the latter implementations are shown in Table 1, for a variety of FPGA families. There are reported both post-synthesis and post-place & route results. The reported operating frequencies for the proposed implementations are related to the corresponding post-synthesis results.
As it can be easily seen, the increase observed for SHA-256, is about 110% gain in throughput and 30% area penalty compared to a non-optimized implementation with four pipeline stages (implemented in the same technology).
This way the improvement that arises from the proposed methodology is confirmed and evaluated fairly, verifying the theoretical analysis in the previous section. Furthermore, comparing the implementations of other researchers to those that were resulted from the proposed methodology, it can be observed that all of them fall short in throughput, in a range that varies from 0.75 – 26.4 times less than the proposed implementation.

VI. Conclusions

In this paper a new methodology was proposed for achieving high throughputs for SHA-256 and other hash functions with a small area penalty. The presented methodology is generic and can be used to a wide range of existing hash functions that are currently used or will be deployed in the future and call for high throughputs.
The methodology led to significant increase of throughput (about 110% for SHA-256), compared to corresponding conventional implementations, with a small area penalty. The results derived from their implementation in FPGA technologies confirm the theoretical results of the proposed implementation.

VII. Acknowledgements

This work was supported by action “Young Researchers from Abroad” which is funded by the Cypriot state-Research Promotion Foundation (RPF/IPE).

VIII. References

1) SP800-77, “Guide to IPSec VPN’s”, NIST, US Dept of Commerce, 2005.
2) FIPS 198-1, “The Keyed-Hash Message Authentication Code (HMAC)”, FIPS Publication 180-1, NIST, US Dept of Commerce, 2007.
3) Thomas, S. (2000). “SSL & TLS Essentials: Securing the Web”, John Wiley and sons Publications.

4) Schneier, B. (1996). “Applied Cryptography – Protocols, Algorithms and Source Code in C” , Second Edition, John Wiley and Sons.
5) Nakajima and M.Matsui, M. (2002). “Performance Analysis and Parallel Implementation of Dedicated Hash Functions”, in LNCS, vol. 2332, pp. 165–180, Springer.
6) Oorschot van, P.C. and Somayaji, A. and Wurster, G. (2005). “Hardware-Assisted Circumvention of Self-Hashing Software Tamper Resistance”, IEEE Transactions on Dependable and Secure Computing, vol. 02, no. 2, pp. 82-92 April-June.
7) Hodjat, A. and Verbauwhede, I. (2004) “A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA IEEE Symposium on Field-Programmable Custom Computing Machines Systems”, (FCCM '04) pp. 308-309.
8) FIPS 180-2, (2002) “Secure Hash Standard”, FIPS Publication 180-1, NIST, US Dept of Commerce.
9) Hoare, R., Menon, P. and Ramos, M. (2002) “427 Mbits/sec Hardware Implementation of the SHA-1 Algorithm in an FPGA”, IASTED International Conference on Communications and Computer Networks, pp.188 – 193.
10) Diez, J.M. and Bojanic, S. and Carreras, C. and Nieto-Taladriz, O. (2002) “Hash Algorithms for Cryptographic Protocols: FPGA Implementations”, TELEFOR.
11) Sklavos, N. and Koufopavlou, O. (2005) “Implementation of the SHA-2 Hash Family Standard Using FPGAs”, Journal of Supercomputing, Kluwer Academic Publishers, vol. 31, pp. 227-248.
12) Lee, Y. K., Chan, H. and Verbauwhede, I.. (2006) “Throughput Optimized SHA-1 Architecture Using Unfolding Transformation”, In Proceedings of the IEEE 17th international Conference on Application-Specific Systems, Architectures and Processors (ASAP), IEEE Computer Society, Washington, DC, 354-359., September 11 – 13.
13) Dominikus, S. (2002) “A Hardware Implementation of MD4-Family Hash Algorithms”, IEEE International Conference on Electronics Circuits and Systems (ICECS'02), pp.1143-1146.
14) Ting, K. K. and Yuen, S. C. L. and Lee, K.-H. and Leong, P. H. W. (2002)“An FPGA based SHA-256 processor”, Lecture Notes in Computer Science (LNCS), vol. 2438, pp. 577–585. Springer.
15) Chaves, R. and Kuzmanov, G.K. and Sousa, L. A. and Vassiliadis, S. (2006). “Improving SHA-2 Hardware Implementations”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2006), pp. 298-310.
16) Glabb, R. And Imbertb, L. and Julliena, G. and Tisserandb, A. and Charvillon, N.V. (2007) “Multi-mode operator for SHA-2 hash functions”, Journal of Systems Architecture, Elsevier Publishing, vol. 53, is. 2-3B, pp. 127–138.
17) Zeghid, M. and Bouallegue, B. and Bagagne, A. Machhoot, M. and Tourki, R. (2007) “A Reconfigurable Implementation of the new Hash Algorithm”, Availability, Reliavility and Security, (ARES 2007), pp.281-285.
18) McEvoy, R.P. and Crowe, F.M. and Murphy, C.C. and William, P. (2006) “Optimisation of the SHA-2 Family of Hash Functions on FPGAs”, Emerging VLSI Technologies and Architectures (ISVLSI’06), pp.317-322.
19) Zeghid, M. and Bouallegue, B. and Machhoot, M. and Bagagne, A. and Tourki, R. (2008) “Architectural Design Features of a Programmable Hgh Throughput Reconfigurable SHA-256 Processor”, Journal of Information Assurance and Security, pp.147-158.
20) Michail, H. and Milidonis, A. and Kakarountas, A.P. and Goutis, C.E. (2005) “Novel High Throughput Implementation of SHA-256 Hash Function Through Pre-Computation Technique”, IEEE International Conference on Electronics, Circuits and Systems (ICECS’05).
21) CAST Inc., Web page, available at http://www.cast-inc.com/cores.
22) Helion Technology Ltd, Data Security Products, Web page, available at http://www.heliontech.com/auth.htm.
23) Cadence, “Hashing Algorithm Generator SHA-256: Technical Data Sheet”, Web page available at http://www.cadence.com/datasheets/SHA256_Datasheet.pdf.