2017 IEEE 23rd International Symposium for Design and Technology in Electronic Packaging (SIITME)

Technological solutions for throughput improvement of a Secure Hash Algorithm-256 Engine

Flavius Opritoiu, Sorin Liviu Jurj, Mircea Vladutiu
Advanced Computing Systems and Architectures (ACSA) Laboratory,
Computer Science and Engineering Department, "Politehnica" University of Timisoara,
2 V. Parvan Blvd, 300223 Timisoara, Romania
flavius.opritoiu@cs.upt.ro; jurjsorinliviu@yahoo.de; mircea.vladutiu@cs.upt.ro

***Abstract*—This article describes a set of techniques for improving the performance of an Secure Hash Algorithm 256 (SHA-256) hardware implementation. The proposed solution reduces the latency incurred for updating the intermediate hash values and relies on using combinational tree structures of CSAs interconnected in a Wallace tree manner for multi-operand addition. Furthermore, the paper investigates the throughout improvement provided by a combined implementation of architecture's binary adders with the round functions used by the hash computation process. The proposed acceleration techniques can be adapted to the other members of the SHA-2 family of algorithms. The architecture represents a case study for hardware optimization based on different combinational structures for binary addition and the effect of the carry propagate layer on the overall performance. The synthesis results of the proposed designs are provided as support for the performance analysis presented in this work.**

Keywords— Iterative Hardware Design, Secure Hash Algorithm 256, Combinational Structures for Addition

I. INTRODUCTION

The ever increasing demand for secure data storage and communication is amplified by the pervasive nature of computing, in general, and of the mobile computing, in particular. More so, by the sensitive nature of the services offered in the online space (banking, medical or educational assistance), the security mechanisms are expected not only to provide confidence in protecting the sensitive data but also to exhibit speed of operation. Hash functions represent an important instrument in the secure computing paradigm operating at the core of many of today's most popular cryptographic protocols and services such as Public Key Infrastructure (PKI), Transport Layer Security (TLS) and Internet Protocol Security (IPSec). Other applications relying on cryptographic hash functions are authenticated access to Virtual Private Networks, file integrity verification and electronic voting systems [1], to name only a few.

A hash function maps a message of arbitrary length to a binary sequence of a fixed length, known as the hash value or message digest, thus assuring the integrity of original message. Integrity, besides confidentiality and availability define the security attribute of a computing system, as presented in the pilot article that introduced the IEEE Transactions on Dependable and Secure Computing journal [2]. In the present era of biometric systems, the security attribute plays a pivotal role in implementation of authentication mechanisms, together with the dependable attribute [2].

The security of hash functions rely on their collision resistance, meaning that given a message, it must be computationally infeasible [3] to find a different one generating the same hash value. Hash functions are also used for generating random numbers and for storing passwords. In conjunction with a signing scheme, they can be used for assuring the authenticity of a message [4]. They make for the perfect binding between many of the cryptographic primitives currently used, due to their one-way property, isolating different parts of a cryptosystem and assuring the compromise of one secret does not reveal others [4].

The hardware implementations of the SHA-2 hash functions, in general and of SHA-256 algorithm, in particular, are benefiting of a higher throughput and of a comparatively more secure computing platform, when compared to their software counterparts. The literature contains references of dedicated hardware architectures for offloading the computational-intensive operations of hash calculation in order to obtain higher throughput [5], [6]. The effect of hash computation over system's throughput is in particular relevant for the case of servers offering services based around IPSec and SSL/TLS, for which the hash calculation latencies becomes a limiting factor in servicing all received requests. In this context, constructing a customized hashing accelerator relieves the Central Processing Unit (CPU) in such a server from computing hashes, allowing it to attend other tasks and thus, optimizing the clock cycles usage.

Another reason for implementing the hash computation in hardware relates to security. Software implementations of a hash function, running on a general purpose processor, oftentimes lack the physical protection found in hardware implementations due to the relative ease with which an attacker is capable of inspecting and even modify the software implementation. Moreover, the intrusion can even be set up concurrent with system's utilization, while the cryptographic application is operational, this being achieved by using debugging software. In the same class of attacks against software implementations of cryptographic functions can be mentioned the timing attacks and the cache attacks customized

for breaking the security of other cryptographic services as well [7].

The paper is organized as follows. Section II presents an overview of the SHA-256 algorithm while the next section offers a perspective on related work that has been done for improving the throughput of SHA-256 hardware realizations. The design considerations and proposed architectural improvements are described in Section IV and the experimental results conducted for evaluating the presented throughput enhancement approaches are described in Section V. The final section concludes the work.

II. SHA-256 ALGORITHM

The SHA-256 is formally presented in [8] and operates with words on 32 bits. The hash value or message digest of a message is a 256-bit vector. Message processing by SHA-256 involves three stages: padding and parsing, message schedule and hash computation or data compression. The padded message is parsed in blocks of 512 bits, each block being processed individually in order to obtain the final hash. The hash value is a vector of 8 words, defined as $H_j^{(i)}$, with j being the index of the hash word, $0 \leq j < 8$, and i being a counter for the currently processing 512-bit block. The initial values of the hash words, $H_j^{(0)}$, are given in [8].

The message schedule expands the 16 words of the 512-bit block into 64 words, denoted from W_0 to W_{63}. The first 16 words of the message scheduler are the very input words forming the 512-bit block. The remaining 48 words are generated by the message scheduler by using two dedicated functions, $\sigma_0^{\{256\}}$ and $\sigma_1^{\{256\}}$, and a binary adder modulo 2^{32}. The data compression phase makes use of 8 working variables, a to h, initialized with the current hash value at the beginning of each 512-bit block processing. This stage involves 64 iterations, each one updating the 8 working variables and using one of the 64 words generated by the message schedule.

The working variables' processing makes use of 4 functions, $\Sigma_0^{\{256\}}$, $\Sigma_1^{\{256\}}$, *Ch* and *Maj*. The variables' processing utilizes a lookup table for storing 64 word constants and generates the result using a modulo 2^{32} adder [8]. Finally, after the 64-th iteration, the hash values, $H_j^{(i)}$, are updated by adding to each of them the content of the corresponding working variables. After processing the last 512-bit block, the 8 hash words are delivered at the output as the message's digest.

III. RELATED WORK

There are several references in the literature concerning efficient hardware implementation for the SHA-256 algorithm. Among the existing works we selected those presenting several acceleration techniques and those having a state-of-the-art approach of summarizing existing solution.

In [1], the authors investigate a number of acceleration techniques that are expected to improve the performance of hash functions, in general, and of SHA-1 and SHA-256 hash operations, in particular. The proposed techniques can be applied in any combination in order to attain the targeted performance. The first solution for improving hash computation efficiency is loop unrolling for which several rounds of the data compression stage are instantiated in hardware. The article presents the design space exploration for determining the optimum number of unrolled iterations. At the expense of a, somewhat, slower clock signal, the hash calculation is performed in fewer cycles in this case.

The next acceleration technique relies on spatial precomputation which is tasked with selecting those output lines that are to be used in the next iteration and which are directly derived from the current iteration's inputs. Instead of deferring the respective calculation for the next round, action is taken to precompute the values used in the next round in order to speed up computation in advance. Often, this technique can be applied only by reordering architecture's pipeline registers.

The next improvement technique relies on prefetching of data directly accessed from architecture's storage layer or from lookup tables. The fourth approach is to design an iterative architecture, for which a limited number of rounds are included in hardware. We say that the rounds that were not instantiated in hardware will unfold in time. The final optimization approach is performed at the circuit-level and involves speeding up the round transformations by solutions such as using Carry Save Adder (CSA) structure for reducing the critical path.

In [9], authors investigate additional acceleration techniques applicable to SHA-256 and, by extension, to other functions from the SHA-2 family. Starting from the critical path of the algorithm, the authors first replace all binary adders by CSAs and include a final lookahead addition stage for computation of the sum in non-redundant representation.

Apart from duplication and operand reordering for computation of the multi-operand addition result, the article investigates two supplementary optimization approaches: pipelining and delay balancing. For delay balancing, the critical path is reduced by displacing elements from this path and moving them on the other side of the path's destination register. The binary adder, generating the non-redundant sum, is moved on the other side of the destination register, thus requiring two registers for storing the redundant form of addition's result.

The use of delay balancing techniques only for calculation of the working variable a is motivated by the limited improvement offered by applying the same technique for calculation of variable e. The pipelining solution introduces intermediate registers for reducing the critical path, placed in carefully selected points of the architecture for improved operation latency.

In [10], authors construct hardware architectures for SHA-1 and for SHA-512 standards for high throughput. The hash acceleration techniques include loop unrolling and precomputation for part of the values used for generating the next working variables. The sum precomputation techniques reduces the latency incurred by addition of 7 words when calculating the next value for working variable a. As a result, the proposed architecture performs addition of the next round constant, the next value of working variable h (which is current

variable *g*), the next message scheduler word and the current hash word, in advance, and stores the result to be used in the next round. The authors investigate the loop unrolling technique in order to determine the number of iterations to be unrolled for increased performance, followed by structural optimizations for the obtained architecture.

IV. Throughput Improvement Solutions for SHA-256

The SHA-256 architecture constructed in this paper is an iterative design, instantiating one round of the message scheduler and one round of the data compression stage in hardware. The reason for including a single iteration in hardware was to provide a rigorous comparison framework for other acceleration techniques, presented in the literature.

The message padding and parsing steps can be implemented either in hardware or in software and are not considered in this paper. The basic design is depicted in Fig. 1. The first architectural optimization that can be applied to a hardware realization of SHA-256, and the first design decision in our proposed architecture targets the message scheduler. Similarly to other hardware designs for SHA-256 found in the literature, the architecture proposed in this paper stores, at any given moment, only 16 words of the message scheduler, instead of providing storage space for all 64 words. This reduction in storage requirements can be realized because the relation used for calculating the next word of the message scheduler makes use of 4 words, all 4 being calculated no later than 16 iterations ahead [8].

Moreover, because the first 16 words of the message scheduler are the very 16 words of the 512-bit input block, the hardware design for the message scheduler stores the input block into the 16 words, at the beginning of a block processing. After delivering word W_0, the message scheduler calculates the next one and shifts the least significant 15 words to the left in order to append the newly calculated value.

The message scheduler consists of the modules in the top part of Fig. 1 in which, for clarity, some of the 16 registers storing the message scheduler words were omitted. In Fig. 1, the "*COUNTER"* unit is a 6-bit iteration counter keeping track of the current algorithm's round. By means of the multiplexing layer connected on the inputs of the message scheduler registers, either the initial 512-bit block or the shifted content is delivered. The word generated by the scheduler each iteration is stored in the least significant position and its calculation delimits the critical path for the message scheduler. The middle register layer of Fig. 1 is made up of the 8 registers storing the working variables, each having the associated variable symbolized next to its output.

One advantage of the SHA-256's hardware realizations over software is the straightforward use of concurrent processing. To this avail, the message scheduler and the data compression stages can be run in parallel because they both are iterated 64 times. The first iteration of the data compression stage makes use of the first word of the message scheduler. By

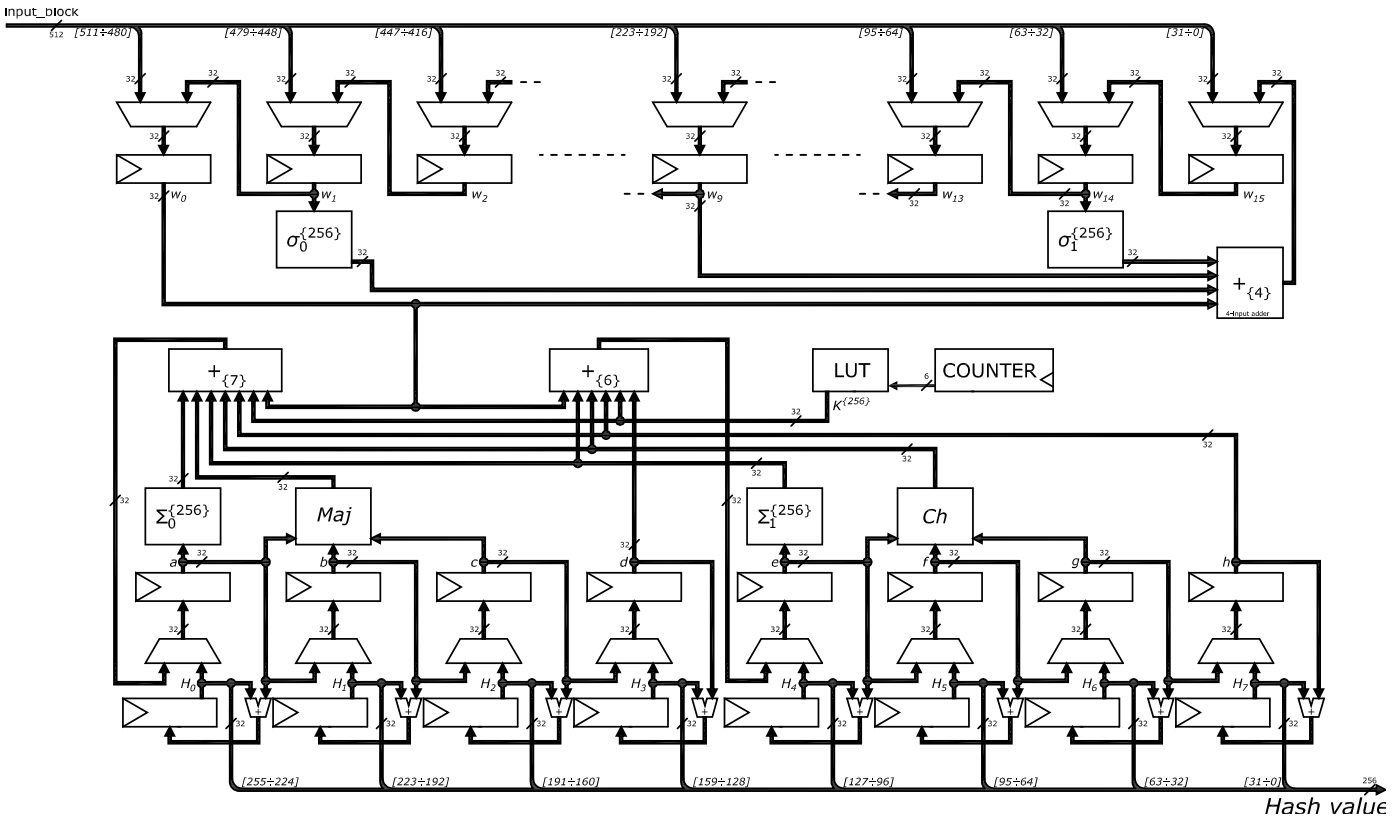

Fig. 1. Basic architecture for SHA-256

the time the scheduler delivers the first word it has already stored the first 16 words and, concurrently with the first iteration of the data compression loop, the scheduler starts calculating the 17th one. It is for this reason that both register sets, storing the message scheduler words and the 8 working variables, are controlled by the same loading signal.

The current word of the message scheduler to be used by the data compression engine is in the most significant position, as depicted in Fig. 1 and because of the content shifting of the scheduler's data words, by the time the last data compression iteration is executed, the scheduler will have generated 15 additional words, besides the last one, W_{63}. This 15 words will not be used by the hash computation. The computation of these words can be avoided; however, it would require additional hardware investment in selecting the current data to be delivered to the hash computation stage. Additional investment would be needed for disabling the load signal for the message scheduling unit. For an increased throughput architecture, the decision to have a common control line for loading the registers of the message scheduler and those storing the working variables, and being able to directly deliver word W_0 to the data compression unit are preferred, to the expense of computing 15 unused words.

The content of the 8 registers storing the working variables is updated either with the current hash value or with the new values generated by the data compression round. The multiplexors selecting the input of the working variable registers use the same selection line as those selecting the input for the scheduler registers because the two sets of registers are initially updated from alternate sources and they are initially updated at the same moment, at the beginning of the 64 iterations. The critical path of the hash engine in Fig. 1 contains the components used for calculating the next value for working variable *a* and includes the modules evaluating the 4 round functions ($\Sigma_0^{\{256\}}$, $\Sigma_1^{\{256\}}$, *Ch* and *Maj*), which operate in parallel, followed by the 7-operand, modulo 2^{32} adder calculating the next value for *a* and followed by the multiplexer delivering the new value to the corresponding working variable register.

The final storage layer is made up of the 8 registers keeping the current hash value. The 8 registers are updated at the end of the 64 data compression iterations by adding the value of the working variables to their current content. Eight modulo 2^{32} adders are required for this operation. In addition, the hash value update demands one supplementary clock cycle, thus directly affecting the latency and the throughput of the SHA-256 unit.

The first throughput acceleration technique proposed in this paper for the SHA-256 architecture reduces the number of cycles used for processing a 512-bit block by eliminating the previously mentioned hash update operation, performed at the end of data compression's loop. To achieve this, the last of the 64 iterations will have to update not only the working variables but the hash registers as well. In consequence, the final round of the data compression phase will have to additionally include the current hash value among the operands added together, in order to be able to generate the next hash value.

In addition to this computation strategy, the current hash value does not need to be loaded into the working variable registers at the start of a new block's processing. This is because the working variables were already updated with the same data as the hash registers in the last round of data compression's stage. However, this observation does not facilitate further reduction of clock cycles because, typically, the loading of the new 512-bit block is performed concurrently with the update of the hash registers.

The manner in which the proposed architecture is updating the working variable registers and the hash registers is depicted in Fig. 2, in which only the modules pertaining to the data compression phase were included. The message scheduler remains unmodified as in Fig. 1, together with the control signals commanding the scheduler's operation.

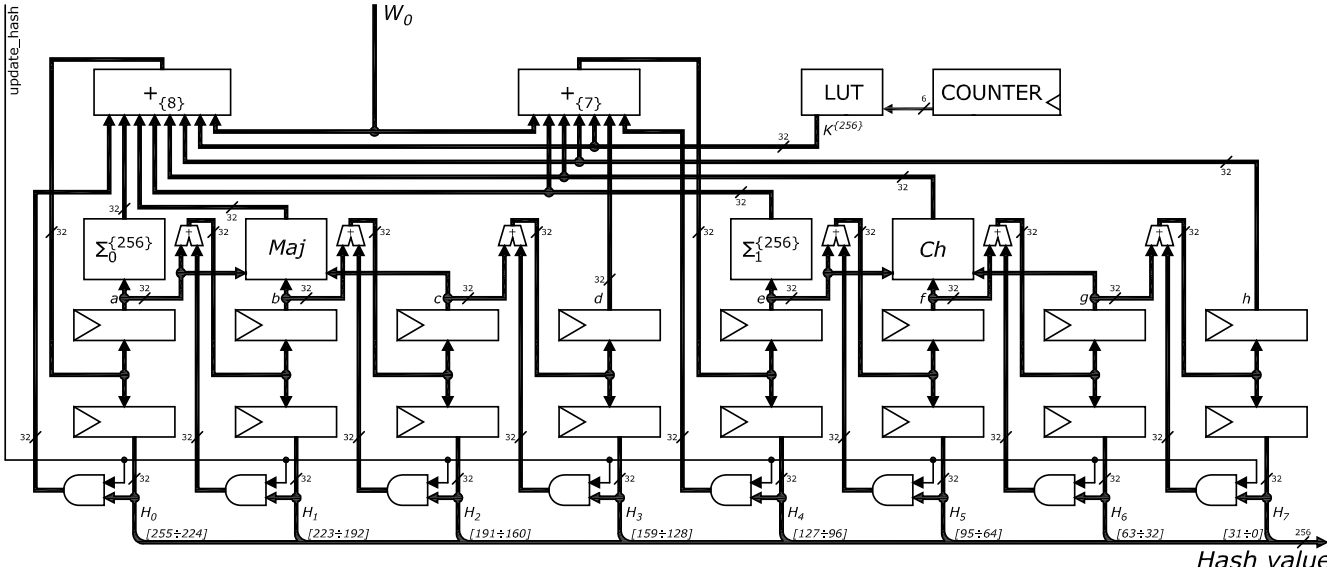

Fig. 2. Proposed architecture for SHA-256 hash calculation

Because the addition of the current hash value is performed only in the last iteration, the content of the hash registers is enabled only once by means of an AND-gate layer, commanded by the gating signal *update_hash*, in Fig. 2. The gating signal is, in fact, the signal enabling loading of the new hash value into the hash registers after the data compression finished. The critical path for this new design follows the signals' propagation through the 4 round functions, followed by the 8-operand modulo 2^{32} adder. The multi-operand adder architecture used throughout this article is a CSA based tree structure, organized in a Wallace manner [3] that generates the non-redundant sum by a final Carry Propagate Adder (CPA).

Because the addition of the operands is performed, for SHA-256, modulo 2^{32}, all CSAs are on 32 bits and, since the carry vector is one position more significant than the sum vector, the most significant carry generated by a CSA level is omitted. As a result, the CPA module is also on 32 bits. However, due to the manner in which latency is propagated through the Full Adder Cells (FACs) of a CSA, the critical path of a CSA tree structure is considerably reduced only if a fast adder is used for CPA, avoiding serial propagation of the carry [11]. Because of this, a fast 32-bit Kogge-Stone [12] adder is used for the CPA level throughout this paper. The replacement of the 7-operand adder in Fig. 1 by the 8-operand addition structure from Fig. 2 affects the adder's critical path only marginally, as the experimental results reveal.

The technological factor has an important influence on the synthesis result. With respect to this aspect, another latency incurring element in the architecture of Fig. 2 is the fan-out of the gating signal controlling the AND-gate layer. The fan-out of the *update_hash* signal is large due to its controlling the hash registers' load line and the gating layer. The large fan-out has an adverse effect on the critical path, in that, depending on the standard cell library used for synthesis, the *update_hash* signal's distribution tree can have a delay larger than the latency of the structures operating the 4 functions and the 8-operand adder together.

Further investigations evaluated alternative multi-operand addition structures and the effect of ordering addition's operands over the critical path. More precisely, in order to further improve performance of the hash core, we investigated the possibility to implement the $\Sigma_0^{\{256\}}$, $\Sigma_1^{\{256\}}$, *Ch* and *Maj* functions in a combined manner with the multi-operand adders present in the architecture. The 4 functions are part of the critical path and reducing theirs' number of logic level improves the overall latency.

Fusing any of the 4 functions with the 32-bit CSAs is limited in outcome by the relatively simple structure of the FACs. However, considering the delay balancing technique introduced in [9], the fused implementation of the 4 functions with the CPA has a larger potential for latency improvement. Fig. 3 illustrates a detail of the fused design combining variable *a*, in redundant form, with hash word H_0^i, also in redundant form, with the hash word H_1^i, the working variable *b* and with the working variable *c*. The fused module calculates $\Sigma_0^{\{256\}}$ and *Maj* functions together with the next working variable *b*.

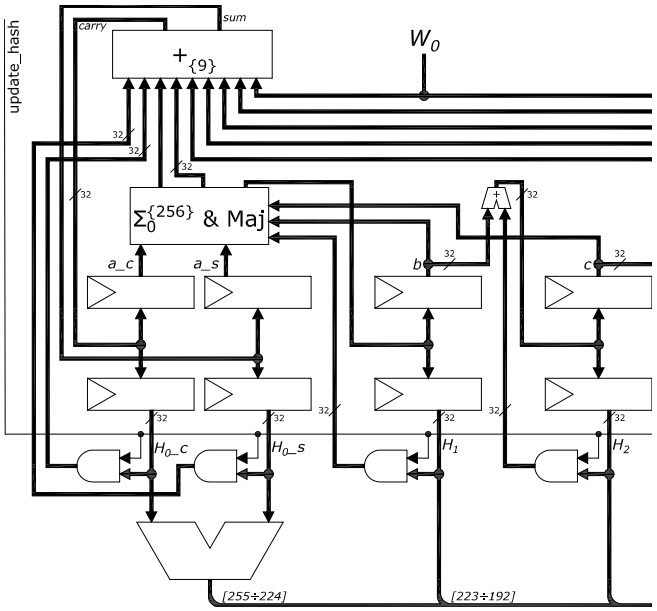

Fig. 3. Detail of the fused architecture

Regarding the delay balancing implementation used in our proposal, it is applied for both working variables *a* and *e*, whereas Fig. 3 illustrates this approach only for *a*. The method requires doubling the registers storing the two variables into sum-carry pairs of registers. The corresponding hash value registers, H_0^i and H_4^i, need to be doubled as well in order to preserve the critical path reduction (otherwise, if the hash registers would not be doubled, dedicated CPAs would be needed for generating the non-redundant hash words which would defeat the purpose of delay balancing). As a result of using two registers for storing the redundant form of hash word H_0^i, the multi-operand adder with 8 inputs from Fig. 2 is replaced by a 9-operand redundant adder which will store the sum in the register pair (*a_c*, *a_s*), for *a*'s carry and sum vectors. Another consequence of storing H_0^i in redundant form is that it requires a CPA for calculating the word of the final digest, adder that is visible in Fig. 3 as well.

Concerning the fused implementation, the *sigma* functions, $\Sigma_0^{\{256\}}$ and $\Sigma_1^{\{256\}}$, are composed of three rotations of the input words, each, followed by Exclusive-OR (EXOR) on the rotated vectors. The functions inputs are generated by the engine's CPAs and for achieving a combined implementation the functions evaluation will be embedded in the CPAs. More precisely, for a Kogge-Stone adder, the carry bits are calculated by a tree structure and they are EXOR-ed with the half sum bits (EXOR result of input operands). Because the *sigma* functions employ the same EXOR operator, their results can be speeded up by balancing the EXOR tree. This solution removes one level of EXOR gates compared to the unfused approach. The *Maj* and *Ch* functions can be expressed in terms of the faster AND/OR primitives as: *Maj*(*a*,*b*,*c*)=*a* AND (*b* OR *c*) OR (*b* AND *c*) and *Ch*(*e*,*f*,*g*)=(*e* AND *f*) OR ($\overline{e}$ AND *g*). The negation of *e* is constructed by negating the half sum bits and EXOR-ing the result with the carry bits inside the Kogge-Stone

CPA, for a rapid output generation. A final approach towards throughput improvement is to reorder the multi-operand adders' inputs in order to connect the signals generated at a later time on positions affected by smaller latencies. By means of synthesis results, the latest operands are identified and are correspondingly connected to the minimum delay inputs of the adder tree.

V. Experimental Results

All presented designs were modeled in Verilog and synthesized using the Design Compiler with the IIT Standard Cell Library for TSMC 0.18μm [13]. The synthesis results are presented in Table I for the basic architecture, together with the proposed and the fused designs. The basic architecture uses as little of the throughput enhancing techniques as possible to prove that the analyzed techniques can be applied in conjunction with most of the other existing acceleration methods.

TABLE I. SHA-256 Architectures Comparison

Architecture	Max Frequency [MHz]	Area [μm^2]	Throughput (Mbps)
Basic	326.80	480625	1287.08
Proposed	357.14	529690	1428.57
Fused	380.23	562704	1520.91

The basic architecture of Fig. 1 uses a multi-operand CSA adder tree, a Ripple Carry Adder for the final CPA layer and process a 512-bit block in 65 cycles. Although it requires the smallest area, the basic design is also the slowest one. The proposed design refers to the architecture described in Fig. 2. It makes use of the hash registers gating technique, uses 8-operand and 7-operand adders for generating variables *a* and *e*, respectively and requires only 64 iterations for a 512-bit block processing. The 10% performance improvement is obtained at the expense of increased area. Finally, the fused architecture takes advantage of the combined implementation of the 4 SHA-256 round functions with the existing CPAs yielding a performance increase of about 18% at the expense of a larger area overhead.

VI. Conclusions

This article presented several acceleration techniques for improving the throughput of a SHA-256 engine. The first acceleration technique eliminates the clock cycle used for hash value update and allows delivering a higher throughput due to the marginal performance loss associated with using an 8-operand adder instead of a 7-operand one. The second technique for improving performance implements the CPAs of the multi-operand adders in a fused manner to speedup generation of the round functions. The proposed, fused design further increase hash engine's performance while the synthesis driven approach for arranging the operands' order in the carry save adder tree further reduce the critical path. In addition to their performance improvements, the presented techniques can be applied in conjunction with other methods presented in the literature, such as loop unrolling, data precomputation in previous round, to name only a few.

The performance improvements of hash implementations presented in this article resonate with the current research aimed towards realizing a more secure and dependable biometric solutions.

References

[1] H. Michail, A. Kakarountas, A. Milidonis, and C. Goutis, "A Top-Down Design Methodology for Ultrahigh-Performance Hashing Cores," IEEE Transactions on Dependable and Secure Computing, vol. 6, pp. 255-268, 2009.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Trans. Dependable Secur. Comput., vol. 1, pp. 11-33, 2004.

[3] M. Vladutiu, Computer Arithmetic: Algorithms and Hardware Implementations: Springer Publishing Company, Incorporated, 2012.

[4] N. Ferguson, B. Schneier, and T. Kohno, Cryptography Engineering: Design Principles and Practical Applications: Wiley Publishing, 2010.

[5] Y. Niu, L. Wu, L. Wang, X. Zhang, and J. Xu, "A Configurable IPSec Processor for High Performance In-Line Security Network Processor," in Computational Intelligence and Security (CIS), 2011 Seventh International Conference on, 2011, pp. 674-678.

[6] A. Thiruneelakandan and T. Thirumurugan, "An approach towards improved cyber security by hardware acceleration of OpenSSL cryptographic functions," in Electronics, Communication and Computing Technologies (ICECCT), 2011 International Conference on, 2011, pp. 13-16.

[7] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: A Timing Attack on OpenSSL Constant Time RSA," in Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings, B. Gierlichs and Y. A. Poschmann, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 346-367.

[8] National Institute of Standards and Technology, "FIPS 180-4, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-4," 2015.

[9] L. Dadda, M. Macchetti, and J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," in Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, 2004, pp. 70-75 Vol.3.

[10] R. Lien, T. Grembowski, and K. Gaj, "A 1 Gbit/s Partially Unrolled Architecture of Hash Functions SHA-1 and SHA-512," in Topics in Cryptology – CT-RSA 2004: The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings, T. Okamoto, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 324-338.

[11] R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI and Their Synthesis," PhD Thesis, Swiss Federal Institute of Technology, Zurich, 1997.

[12] S. Roy, M. Choudhury, R. Puri, and D. Z. Pan, "Polynomial Time Algorithm for Area and Power Efficient Adder Synthesis in High-Performance Designs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, pp. 820-831, 2016.

[13] J. E. Stine, J. Grad, I. Castellanos, J. Blank, V. Dave, M. Prakash, et al., "A Framework for High-Level Synthesis of System-on-Chip Designs," presented at the Proceedings of the 2005 IEEE International Conference on Microelectronic Systems Education, 2005.