Microprocessors and Microsystems 37 (2013) 750–757

Contents lists available at SciVerse ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

FPGA-based implementation alternatives for the inner loop of the Secure Hash Algorithm SHA-256

I. Algredo-Badillo [a], C. Feregrino-Uribe [b], R. Cumplido [b,*], M. Morales-Sandoval [c]

[a] Computer Engineering, University of Istmo, Tehuantepec, Oaxaca 70760, Mexico
[b] National Institute for Astrophysics, Optics and Electronics, Santa. Ma. Tonantzintla, Puebla 72840, Mexico
[c] Polytechnic University of Victoria, Information Technology Department, Tamaulipas 87138, Mexico

ARTICLE INFO

Article history:
Available online 6 July 2012

Keywords:
SHA-2
Hash function
FPGA

ABSTRACT

Hash function algorithms are widely used to provide security services of integrity and authentication, being SHA-2 the latest set of hash algorithms standardized by the US Federal Government. The main computation block in SHA-2 algorithms is governed by a loop with high data dependence for which several implementation strategies are explored in this work as well as designs efficiently mapped to hardware architectures. Four new different hardware architectures are proposed to improve the performance of SHA-256 algorithms, reducing the critical path by reordering some operations required at each iteration of the algorithm and computing some values in advance, as possible as data dependence allows. The proposed designs were implemented and validated in the FPGA Virtex-2 XC2VP-7. The achieved results show a significant improvement on the performance of the SHA-256 algorithm compared to similar previously proposed approaches, obtaining a throughput of 909 Mbps and an improved efficiency of 0.713 Mbps/slice.

1. Introduction

Digital communication systems are widely used for executing a wide variety of electronic operations such as: electronic transfers, mobile communications, multimedia, electronic commerce, document transfers, and videoconferences, among others. The information being transmitted by these systems can potentially be at risk if no security measures are taken. The most common risks found in these systems include: non-authorized accesses, denial of service, data corruption, leakage, monitoring attacks, authentication, and trashing [1]. Several measures can be taken to reduce such risks; among the most common are the use of firewalls, traffic filtering, and detection systems. Additionally, security can also be provided by using cryptographic algorithms [1].

Hash functions are cryptographic primitives widely used to provide services of data integrity and authentication issues. These functions allow to maintain a high level of security by performing complex operations, usually in an iterative fashion, which require a significant amount of computing resources. A hash function maps binary strings of arbitrary length to binary strings of some fixed length, called hash-value or *digest*. Hash functions are widely spread and mainly used for the implementation of digital signature algorithms [2,3]. Several algorithms for performing hash functions [1] have been proposed: MD5, SHA-1, SHA-2, Whirlpool, Haval, and RipeMD-160, being the SHA family the most widely used, in particular SHA-2 that offers higher security and solved the insecurity problems of SHA-1 [4] and other popular algorithms as MD5 and SHA-0 [5,6] (already broken). Currently, several hash function algorithms are being evaluated to select SHA-3 [7,8].

The efficient implementation of hash functions has been an active research topic in industry and academia in recent years. There are several research papers and commercial products that offer hardware or software implementations of one or more algorithms from the SHA-2 family. The hardware architectures previously reported aim to achieve better performance by customizing hardware elements that efficiently compute specific functions. Main techniques previously used include the use of well balanced Carry Save Adders (CSA) [9], unrolling techniques [10,11], the usage of embedded memories [12], the use of pipelining techniques [13] among others. In general, these techniques resulted in more area requirements or more complex control logic that affect the critical path and decrease the performance. This work describes two strategies to improve the performance of hardware implementation of the main operations in the SHA-2 algorithms. The critical path is reduced by predicting the result of some computations in the inner loop of SHA-2 algorithms up to two operations in advance. The critical path is also decreased by reducing latency of the calculations by re-arranging operations in the core of SHA-2 algorithms. The architectures proposed in this work are optimized to execute

* Corresponding author. Tel.: +52 222 2663100x8225; fax: +52 222 2663152.
E-mail addresses: algredobadillo@sandunga.unistmo.edu.mx (I. Algredo-Badillo), cferegrino@ccc.inaoep.mx (C. Feregrino-Uribe), rcumplido@inaoep.mx (R. Cumplido), mmoraless@upv.edu.mx (M. Morales-Sandoval).

0141-9331/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2012.06.007

the entire algorithm of hashing, not only for processing a single block as in [13]. The four proposed architectures use carry save adders (CSAs), balancing of datapaths, and a 256-bit state buffer to store the eight registers $A - H$ to compute the hashing. This state buffer is considered as a single signal that is feed backed to process each new data block. In [13], feed backing process is based on the use of multiplexers and single 32-bit registers but that approach may result in larger data paths and more complex control logic that could increase the delay of the critical path and decrease the performance. The proposed hardware designs were evaluated for the SHA-256 algorithm implemented on the Virtex-2 XC2VP-7 FPGA device. However, these designs can be easily extended for other hash algorithms in the SHA-2 family, which present a common structure. Compared to similar approaches such as [13], the throughput per slice efficiency achieved in this work is the highest, reaching around 909 Mbps with an area usage of 1150 FPGA slices. The work presented here is an extended version of [14], where two hardware architectures for computing the inner loop of SHA-2 algorithms were reported. The main idea in [14] is the computation of some additions in the inner loop one iteration in advance to reduce the critical path delay in the next iteration. While the first architecture is based in the use of a subtraction, the second one is only based on precomputed additions. In the present work, Carry Save Adders are used to perform the additions demanded in the inner loop of SHA-2 algorithms instead of traditional adders, which allows to reduce the critical path delay. In addition, a new evaluation platform is designed to execute the entire SHA-2 algorithm, where the datapath is shortened by adding 2 new registers without increasing the latency. This platform allows to evaluate the two hardware proposals for the inner loop, executing the entire SHA-2 algorithm and achieving improved results compared to the ones reported in [14].

The rest of this paper is organized as follows: Section 2 describes the SHA-2 family, Section 3 shows the details of data dependence in SHA-256 algorithm, Section 4 presents the proposed architectures for computing the main operations of SHA-2, Section 5 describes the implementation details and discusses the results, and finally Section 6 concludes.

2. Secure Hash Algorithms

Hash functions are mainly used to provide the security service of integrity. They also provide the security service of authentication when they are used in combination with digital signature and message authentication code (MAC) algorithms. Among the most important hash functions is the SHA-2 family, which share the same functional structure with some variation in the internal operations, message size, message block size, word size, number of security bits and message hash size, see Table 1 [15].

These algorithms are iterative and one-way functions that input a message and output a message digest. They process the input data in two stages: preprocessing and digest computing. The preprocessing warranties that the message has a size that is multiple of a particular value, allowing to divide the message into predefined block sizes and to provide an initial hash value. In the second stage, each message block is utilized during a fixed number of iterations, where at each iteration the algorithm defines functions, constants and word operations to generate a series of hash values. After all blocks are processed, the value of the final hash is used as the message digest. In particular, the second stage of the SHA-256 algorithm performs 64 iterations over blocks of 512-bit messages and hash values of 256 bits described as eight 32-bit words $(A, B, \ldots, H)$. The hash message is 256-bit long, see Fig. 1.

Table 1
Secure Hash Algorithms (sizes and security are specified in bits).

Algorithm	SHA-1	SHA-256	SHA-384	SHA-512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Message digest size	160	256	384	512
Security	80	128	192	256

3. Related works

Implementations of hash functions require great amounts of spatial and temporal resources that may result in low performance and efficiency. This has motivated the research on several approaches for speeding up the computation or for improving the efficiency in terms of throughput/area ratio.

The pseudocode shown in Fig. 2 describes the SHA-256 algorithm; more details are given in [15]. The general structure of the pseudocode is very similar to the other algorithms of the SHA-2 family. The need to efficiently implement the computations of the inner loop, that represents the iterations performed by the algorithm, has resulted in a number of implementation approaches. In general, depending on the system organization, hardware implementations report better performance than software only implementations [16].

An important factor that limits the performance in both hardware and software implementations is the high data dependence of the computations within the inner loop. Because of this, fully parallel implementations are not possible, however partial loop unrolling has been explored by several authors [10,11]. When partial unrolling is used, data dependence is present when the iteration $k+1$ requires the results obtained for the 8 variables A, B, …, H by iteration k, see Fig. 3.

In order to take advantage of specialized cores or embedded systems, several architectures have been reported to iteratively operate without using parallelization or unrolling of the inner loop [17]. Other approaches consist on introducing a pre-computing stage in order to reduce the critical path. By adding some operations to the initialization process, pre-computing allows to reduce the number of adders on the critical path (reducing the data dependence), and increases the performance [9,13]. These architectures will be analyzed in Section 6.

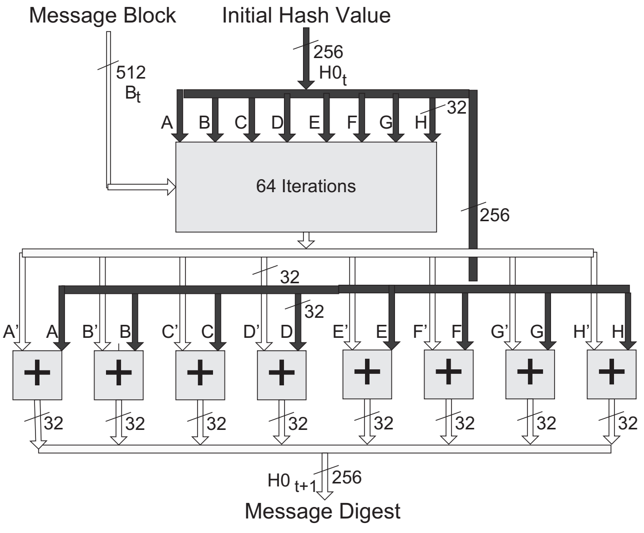


Fig. 1. Block diagram of the SHA-256 algorithm.

```
For i = 1 to N:
  Prepare the message Schedule $W_t$
 Initialize the eight working variables A,…,H
  For t = 0 to 63:
       Compute Temporal $T_1$
       Compute Temporal $T_2$
       Compute new H, G, F
       Compute new E = D + $T_1$
       Compute new D, C, B
       Compute new A = $T_1$ + $T_2$
  Compute the $i^{th}$ intermediate hash value $H^{(i)}$
Generate the message digest with $H^{(N)}$
```

Fig. 2. Pseudocode for the SHA-256 algorithm.

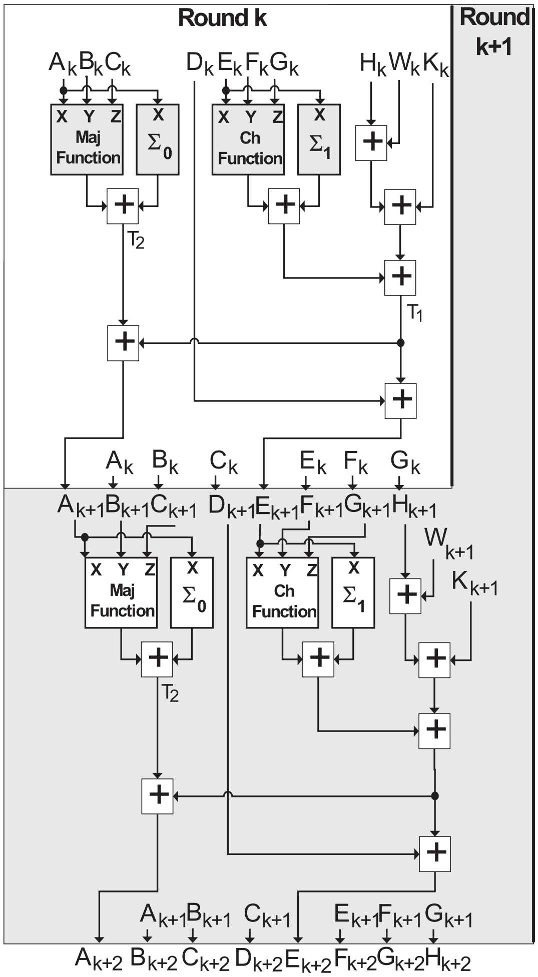

Round k
Round k+1
$A_k B_k C_k$
$D_k E_k F_k G_k$
$H_k W_k K_k$
X Y Z
Maj Function
X
Σ_0
X Y Z
Ch Function
X
Σ_1
+
T_2
T_1
A_k B_k C_k E_k F_k G_k
$A_{k+1} B_{k+1} C_{k+1} D_{k+1} E_{k+1} F_{k+1} G_{k+1} H_{k+1}$
W_{k+1}
K_{k+1}
$A_{k+1} B_{k+1} C_{k+1} E_{k+1} F_{k+1} G_{k+1}$
$A_{k+2} B_{k+2} C_{k+2} D_{k+2} E_{k+2} F_{k+2} G_{k+2} H_{k+2}$

Fig. 3. Data dependence in the iterative process of the SHA-256 algorithm.

4. Proposed architectures

In a straightforward hardware implementation of the SHA-256 algorithm, there are several ways for designing the inner part of the loop because of the number of additions required. It is possible to rearrange the order of the computation with the aim of increasing the performance taking into account the data dependencies.

The operations in the inner loop of the algorithm compute new values for the variables *A* and *E* taking into account six from the eight previous values of the same variables, see Eqs. (1) and (2). Additional processes are used for computing the remaining variables (*B*, *C*, *D*, *F*, *G*, *H*), which are updated by setting values of the previous value, more details in [15].

$$E_{k+1} = D_k + \Sigma_1(E_k) + CH(E_k, F_k, G_k) + H_k + K_k + W_k \tag{1}$$

$$A_{k+1} = \Sigma_0(A_k) + MAJ(A_k, B_k, C_k) + \Sigma_1(E_k) + CH(E_k, F_k, G_k) + H_k + K_k + W_k \tag{2}$$

In [13], authors propose the pre-computing of δ_k to save the calculation of a sum during the run time of the iteration, that is, to calculate δ_k in a previous iteration *k-1* by using G_{k-1} as described by Eq. (3).

$$\begin{aligned}\delta_k &= H_k + K_k + W_k \\ &= G_{k-1} + K_k + W_k\end{aligned} \tag{3}$$

In Eq. (3), the values H_k, K_k and W_k must be pre-computed or must be present to obtain δ_k at time *k* and to perform the calculation of iteration $k+1$. In this last iteration, the computing of the new values *A* and *E* is based on Eqs. (4) and (5).

$$E_{k+1} = D_k + \Sigma_1(E_k) + CH(E_k, F_k, G_k) + \delta_k \tag{4}$$

$$A_{k+1} = \Sigma_0(A_k) + MAJ(A_k, B_k, C_k) + \Sigma_1(E_k) + CH(E_k, F_k, G_k) + \delta_k \tag{5}$$

Additionally, two new forms of reordering the operations of the inner loop are proposed: one based on the subtraction of D_k and the other one employs two previous calculations. In the first proposal of this work, the data flow is rearranged to obtain the operations of the sum described by Eqs. (1) and (2). It calculates a variable τ_k, according to Eq. (6).

$$\tau_k = H_k + K_k + W_k + D_k \tag{6}$$

The calculation at run time *k* is based on Eqs. (7) and (8), where no pre-computing is executed.

$$E_{k+1} = \Sigma_1(E_k) + CH(E_k, F_k, G_k) + \tau_k \tag{7}$$

$$A_{k+1} = \Sigma_0(A_k) + MAJ(A_k, B_k, C_k) + \Sigma_1(E_k) + CH(E_k, F_k, G_k) + \tau_k - D_k \tag{8}$$

The second proposal of this work is based on the pre-computing of Eqs. (3) and (9) at time *k* using a previous value at time *k-1*. This last one thing is the difference between τ_k and δ_k, Eqs. (3) and (9), respectively. The main idea is to calculate other additions in a previous time *k* in order to reduce the critical path of the inner loop operations at the time $k+1$. It is important to mention that τ_k cannot be defined in terms of δ'_k because τ_k uses values computed at iteration *k* while δ'_k uses values obtained at iterations $k-1$ and *k*.

$$\delta'_k = \delta_k + D_k \tag{9}$$

The calculation at run time $k+1$ is based on the Eqs. (10) and (11).

$$E_{k+1} = \Sigma_1(E_k) + CH(E_k, F_k, G_k) + \delta'_k \tag{10}$$

$$A_{k+1} = \Sigma_0(A_k) + MAJ(A_k, B_k, C_k) + \Sigma_1(E_k) + CH(E_k, F_k, G_k) + \delta_k \tag{11}$$

Based on the four previous equations to compute $E_k + 1$ and $A_k + 1$, four corresponding hardware architectures are proposed to perform the operation of the inner loop in the SHA-2 algorithms, see Fig. 4. Fig. 4a shows the architecture in forward form derived from Eqs. (1) and (2). The implementation shown in Fig. 4b saves the calculation of two adders because they are pre-computed in a previous time *k*. This implementation calculates the inner loop operation according to Eqs. (3)–(5).

The architectures for the new algorithmic proposals of this work are shown in Fig. 4c and 4d. Fig. 4c shows the first proposed architecture that is based on rearranging the dataflow as described by Eqs. (6)–(8), whereas the second proposed architecture based

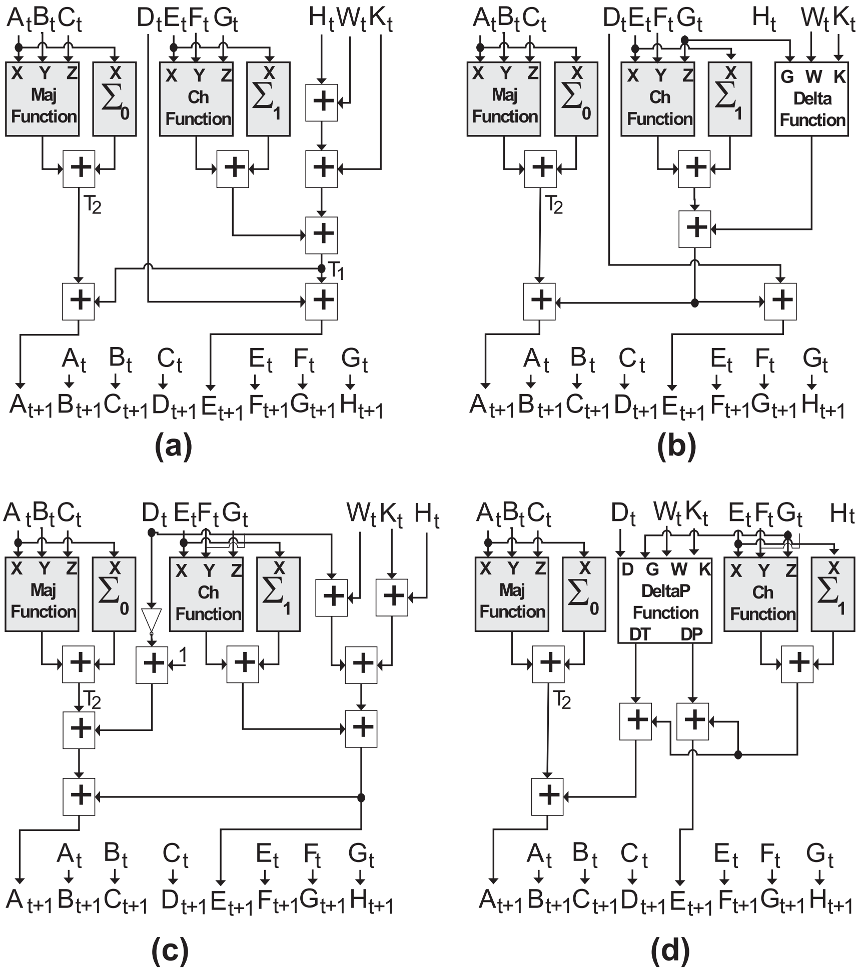


Fig. 4. Block diagrams for computing the inner loop operations of the algorithm SHA-256: (a) straightforward architecture, (b) architecture with basic pre-computing, (c) proposed architecture with reordering of the data flow and (d) proposed architecture with two pre-computations.

on Eqs. (9)–(11) is shown in Fig. 4d. The calculations of the hardware architectures shown in the Fig. 4b and 4d require an additional clock cycle to initialize the system, with the advantage of decreasing the data dependence.

The datapaths in the two proposals exhibit distinct levels of combinational logic. For the first proposal, its critical path is reduced by including CSAs and distributing the combinational elements along the X axis, see Fig. 5. The process is iterative, storing the new variables A_{k+1} and E_{k+1} in two registers which are feed backed to compute the next 64 rounds.

In the second proposal there are two independent sets of combinational logic, see Fig. 6. Set 1 involves the pre-computation of δ_k and δ'_k at time $k-1$. Set 2 involves all computations at time k to update the registers for variables A_{k+1} and E_{k+1}. The proposed partition allows to simplify the operations in the main process and reduce the critical path. Each set and datapath is divided by

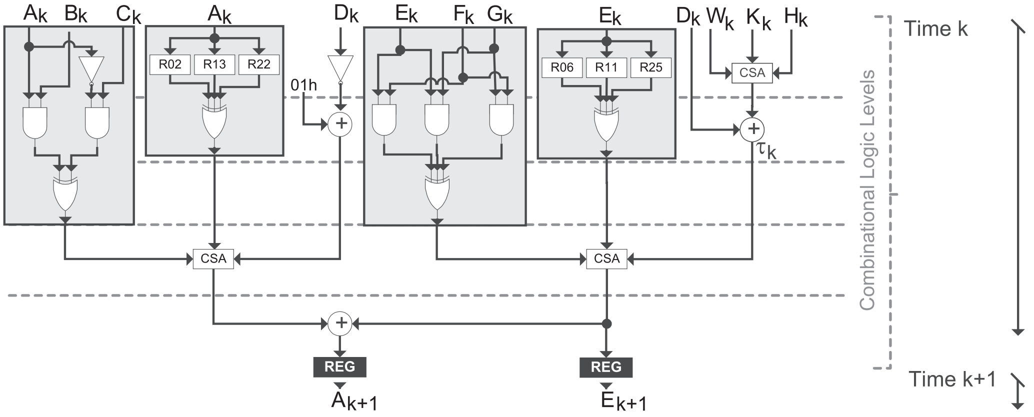


Fig. 5. Structure for the first proposal of the inner loop.

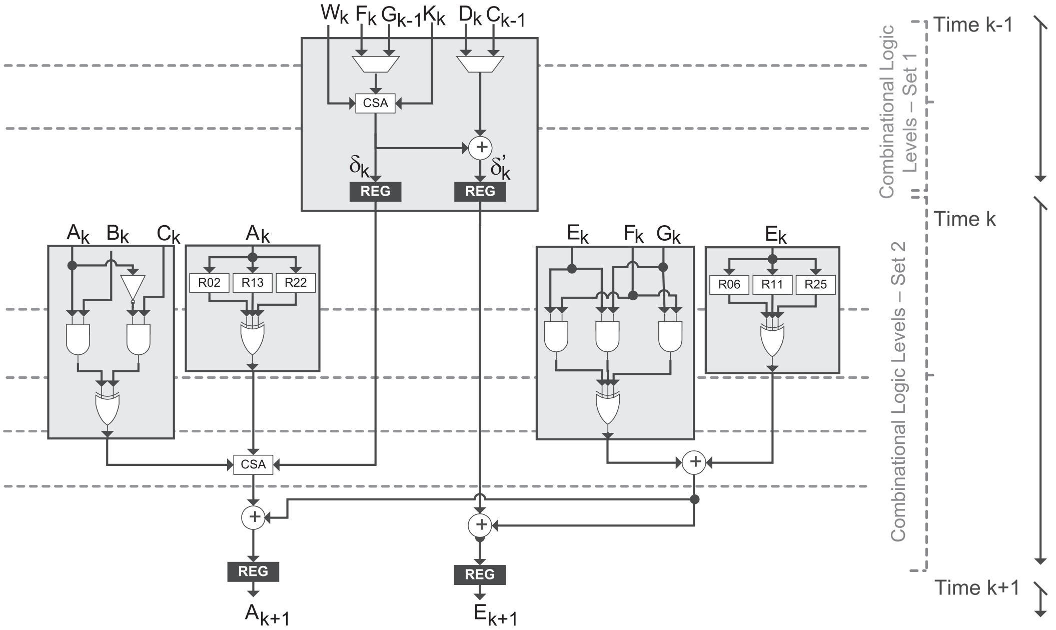


Fig. 6. Structure for the second proposal of the inner loop.

sequential elements (registers), enabling the pre-computing to be calculated at time $k-1$ and use it in the computations at time k which produces new values for the variables A and E at time $k+1$. It is important to highlight that the values at time k are necessary for the pre-computation at time $k-1$, such as round word W_k and round constant K_k. Also, an initial digest message is necessary when the first round is computed. For this reason the values F_k y D_k must be present from the beginning. For the other 63 rounds, the values G_{k-1} and C_{k-1} are used. These last four values allow the precomputation from an intermediate state buffer, considering that $H_t = G_{t-1}$ and $D_t = C_{t-1}$, see Fig. 3.

The proposed structures shown in Figs. 5 and 6 for the inner loops are implemented and evaluated on two different platforms that incorporate processes to compute the round word, round constant, state buffer and to control the dataflow of the entire SHA-2 algorithm.

5. Implementation results and discussion

In order to evaluate the different inner loops under fair conditions, four relative architectures are proposed from the structures described in Section 4 and they are initially implemented on a common platform. Later, two optimizations are included leading to a second platform. The first platform, see Fig. 7, allows to compare the proposed architectures in terms of hardware resources, throughput and efficiency. This common platform is a straightforward implementation of the SHA-256 algorithm that can be easily modified to accommodate the four architectures to perform the inner loop operations. The white blocks of each architecture in Fig. 4 were implemented in the white block named Main_Function of the basic platform, see Fig. 7.

The first platform shown in Fig. 7 [14] explores parallelization of necessary resources within an iterative module, which focuses on computing 64 rounds by using 64 clock cycles. Two important paths of this platform appears when a register is placed after the main function block: (a) one path is generated by the 6-bit bus used to produce the round constant K_t and (b) one path generated by the selection bit *SEL_H*0 of the multiplexer. The first path appears when the ROM *Memory*64×32 is controlled by a 6-bit counter. This counter generates addresses to select the constants K_t for each round, which are inputs to the adders and/or multiplexers when τ_k, δ_k and δ'_k are computed, see Figs. 5 and 6. This routine has combinational elements that generate a large path. The delay associated to this path is decreased by inserting a register after the ROM. This new register does not require any new clock cycle, because the control logic for that ROM is synchronized in a such way that the control signals in the current state are computed in a previous one. The values of the counter are properly displaced to have the correct values when they leave the register. For the second path a similar solution is applied. In the same way, the control signal of the multiplexer is displaced without increasing the clock cycles. The second platform incorporates two optimizations based on dividing large path that generate critical paths at different times, adding two registers, see Fig. 8.

The four architectures proposed in this work use the four different inner blocks previously revised, placed and routed on the second platform; they are implemented in VHDL language and synthesized for a Virtex-2 XC2VP-7 FPGA device using Xilinx's ISE 10.1 tool. The implementation results are shown in Table 2. Instead of using simple adders and designing an iterative architecture [14], these new architectures are balanced, considering combinational paths and sequential elements. Additionally, CSAs (carry save adder) are used to implement the integer additions required in the computation of A_{k+1} and E_{k+1}. The four architectures process a 512-bit input block.

The throughput was calculated using Eq. (12), whereas the efficiency with Eq. (13).

$$Throughput = \frac{Data\ block\ size}{Clock\ time \times Clock\ cycles} \tag{12}$$

$$Efficiency = \frac{Throughput}{Number\ of\ Slices} \tag{13}$$

The forward architecture (Fig. 4a) requires less area and reports the best efficiency, although it achieved the worst throughput. The first proposed architecture named SHA-256c (Fig. 4c) reports an improved throughput, requiring more hardware resources, which in turn decreases the efficiency. The two architectures that execute pre-computations report poorer efficiencies, but with the highest throughput. The second proposed architecture, named SHA-256d (Fig. 4d), requires more hardware resources because of the pre-

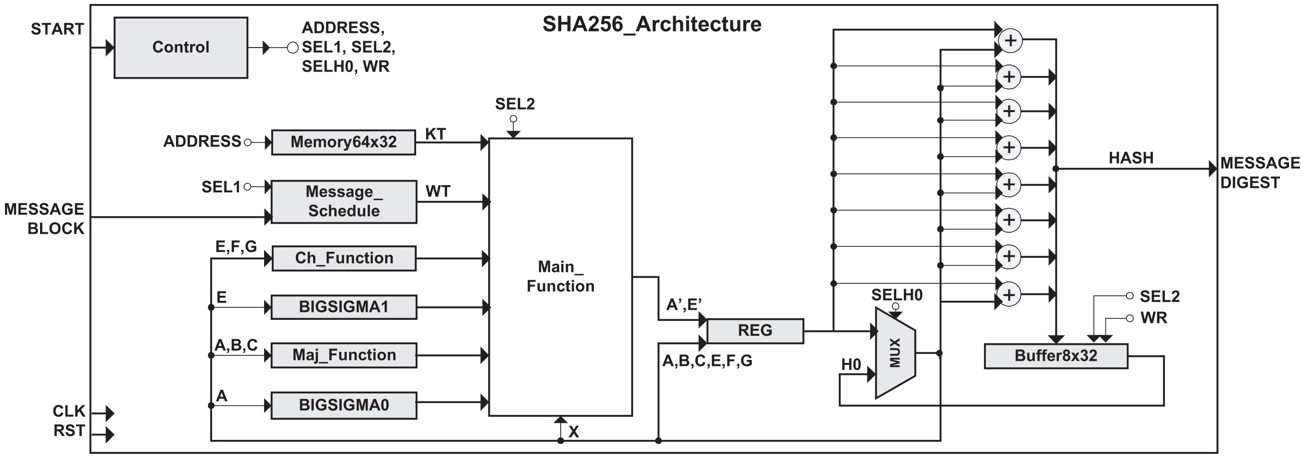


Fig. 7. Architecture of the SHA-256 algorithm used to evaluate the proposed approaches.

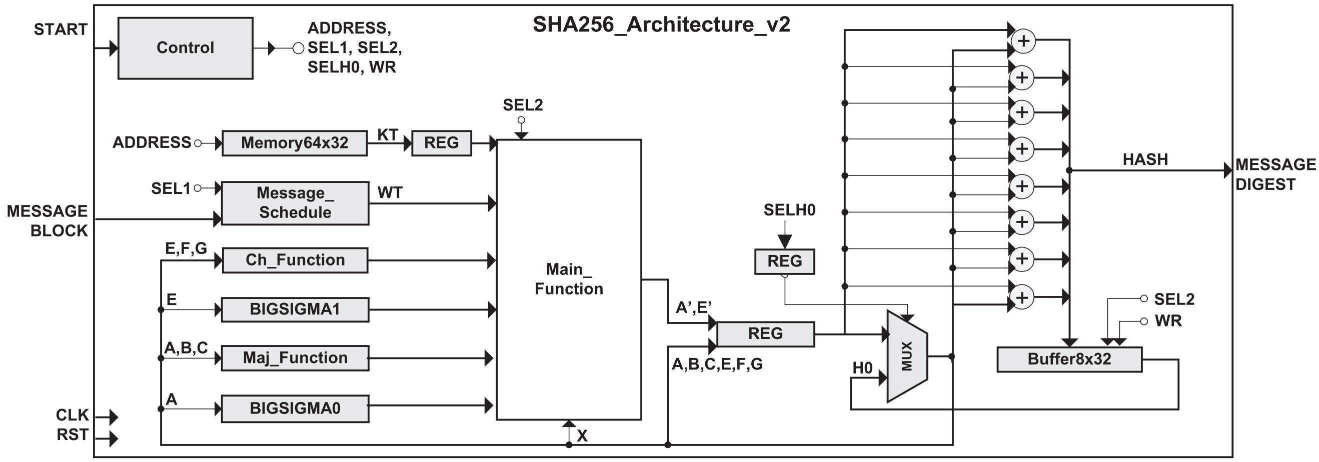


Fig. 8. Optimized platform of the SHA-256 algorithm, which balances large paths.

Table 2
Implementation results of the proposed architectures.

Architecture	Hardware resources (Slices)	Clock frequency (MHz)	Throughput (Mbps)	Efficiency (Mbps/Slice)
SHA-256a	1125 S, 1333 LUTs, 64 cycles	104.02	819.20	0.728
SHA-256b	1149 S, 1403 LUTs, 65 cycles	114.55	902.23	0.785
SHA-256c	1187 S, 1411 LUTs, 64 cycles	110.10	867.72	0.731
SHA-256d	1274 S, 1478 LUTs, 65 cycles	115.46	909.48	0.713

Table 3
Relevant hardware architectures for the algorithm SHA-256.

Work	Device	Hardware Eesources and clock frequency	Throughput (Mbps)	Efficiency (Mbps/Slice)
[13]	XC2VP30	100 MHz, 65 cycles	785	–
[17]	V200/ 400XCV	1306 Slices, 77 MHz, 66 cycles	308	0.236
[9]	XCV-1000	1038 LUTs, 39.5 MHz	316	–
[12]	XV200-6	2384 CLBs, 1 BRAM, 74 MHz	291	–
[21]	Stratix II EP2S60	1380 Slices	772	0.516
[22]	Virtex-5	80 MHz, 65 cycles	630	–

computations that it performs, however it achieves the highest throughput. The proposed architectures can be seen as single unrolled designs, thus they can be used as basis to explore architectures with several unrolled rounds.

There are several commercial and academic implementations of the SHA-256 algorithm reported in the literature [17,18]. For the commercial implementations reported in [19] and [20], internal details of the architectures are not reported, thus they will not be included in the discussion.

Academic implementations are based on several approaches such as straightforward design, unrolled/pipelined designs, hardware/software implementations and designs using pre-computations. The SHA-256 algorithm is defined for an indeterminate number of blocks from a message. To process a data block it is necessary the processing result of the previous data block. Because of this, works previously reported implementing the main core of SHA algorithms and processing just a single block are not considered, neither segmented architectures processing more than one block at the same time. For comparison, works reporting pipelined or iterative architectures to process a single block at a time are only considered [17,13,9,12,21,22] (see Table 3). However, a fair comparison is still not feasible because of variations in the device used as well as the software tools. Thus the information in Table 3 is provided only as a reference.

Considering implementation results from Table 3, the hardware architectures proposed in this work report the highest throughput and efficiency, using fewer area resources.

In [12,23], forward architectures that calculate three hash functions, including the SHA-256, are reported; the implementation in [12] only computes the first stage of the algorithm. The works [17,9,21,22] report architectures with basic iterative architectures, where optimizations are done to decrease the critical path. In [9], authors propose to use three operands adders. It is important to highlight that [13] proposes several optimizations for the inner loop using the function δ_k. The throughput reported in that work is 1.4 Gbps but this corresponds only to the architecture for computing the inner loop of SHA-2, not for the entire SHA-2 algorithm, for which the reached throughput is only 785 Mbps. The authors use circular buffer and save six adders to compute the final message digest. In this work, the results reported are for the complete algorithm with message schedule, 256-bit state buffer and control unit, parallelizing different modules, large buses and eight final adders. In [21], the reported architecture calculates three different hash functions: MD5, SHA-1 and SHA 224/256, whereas [22] reports an architecture to perform the ciphering algorithm AES with different key sizes and the hash functions SHA-1 and SHA-256. Some other works are implemented using different approaches such as hardware/software co-design [24–26], or use of parallelization techniques, pipelining and loop unrolling [13,10,11,18].

6. Conclusion

Four new different schemes to improve the performance of the hardware implementation of the SHA-2 family were proposed. These architectures exhibit high performance, throughput and efficiency. The hardware designs are based on the rearrangement of computation in the inner loop of SHA-2 algorithms, computing some values in advance, balancing of paths, adding registers to reduce paths by modifying control module without to increase clock cycles, and taking advantage of hardware architecture (parallelization, large buses and replicated components).

The proposed architectures use carry save adders, balancing of data paths and a state buffer to feedback the eight 32-bit registers that keep the partial hashing needed to process each next data block. The use of the state buffer helps to simplify the data paths in the hardware architectures reducing the maximum delay in the synthesized circuit. As a result, higher frequencies are achieved, minor area resources are used, and throughput is increased. The proposed architectures are improved by exploring the algorithmic proposals for computing A_{k+1} and E_{k+1} and taking advantage of hardware architectures. These modifications result in hardware implementations with an improved efficiency and higher throughput, while they slightly maintain a smaller use of hardware resources.

Although the results were focused on the SHA-256 algorithm, these approaches can be extended to the other algorithms of the SHA-2 family due to their common structure, as well as to explore other hardware techniques like the unrolling and pipelining.

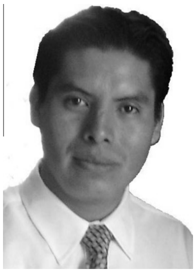
References

[1] W. Stallings, Cryptography and Network Security, Prentice Hall, NJ, 2003.
[2] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120126.
[3] American Bankers Association, ANSI X9.62-1998: public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA), 1998.
[4] X. Wang, Y. Yin, H. Yu, Finding collisions in the full SHA-1, in: V. Shoup (Ed.), Advances in Cryptology CRYPTO 2005, Lecture Notes in Computer Science, vol. 3621, Springer, Berlin, Heidelberg, 2005, pp. 17–36 (Chapter 2).
[5] P. Gauravaram, A. McCullagh, E. Dawson, Collision attacks on md5 and sha-1: is this the sword of damocles for electronic commerce? in: Proccedings of AusCERT Asia Pacific Information Technology Security Conference (AUSCERT2006): Refereed R& D STREAM, 2006, pp. 73–88.
[6] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, W. Jalby, Collisions of SHA-0 and reduced SHA-1, in: R. Cramer (Ed.), Advances in Cryptology EUROCRYPT 2005, Lecture Notes in Computer Science, vol. 3494, Springer, Berlin/Heidelberg, 2005, p. 577.
[7] A. Regenscheid, R. Perlner, S. Jen Chang, J. Kelsey, M. Nandi, S. Paul, Status report on the first round of the SHA-3 cryptographic hash algorithm competition, Tech. rep., NIST, September 2009.
[8] E. Andreeva, B. Mennink, B. Preneel, Security reductions of the second round SHA-3 candidates, in: ISC, 2010, pp. 39–53.
[9] D. Fedoryka, Fast Implementation of the Secure Hash Algorithm SHA-256 in Field Programmable Gate Arrays, George Mason University, 2004. <http://books.google.com/books?id=f009GwAACAAJ>.
[10] M. Zeghid, B. Bouallegue, M. Machhout, A. Baganne, R. Tourki, Architectural design features of a programmable high throughput reconfigurable SHA-2 processor, Journal of Information Assurance and Security 2 (2008) (2007) 147–158.
[11] H.E. Michail, G.S. Athanasiou, A.A. Gregoriades, C.L. Panagiotou, C.E. Goutis, High throughput hardware/software co-design approach for SHA-256 hashing cryptographic module in IPSec/IPv6, Global Journal of Computer Science and Technology 10 (4) (2010) 54–59.
[12] N. Sklavos, O. Koufopavlou, Implementation of the SHA-2 hash family standard using FPGAs, The Journal of Supercomputing 31 (3) (2005) 227–248.
[13] R. Chaves, G. Kuzmanov, L. Sousa, S. Vassiliadis, Improving SHA-2 hardware implementations, in: CHES, 2006, pp. 298–310.
[14] I. Algredo-Badillo, C. Feregrino-Uribe, R. Cumplido, M. Morales-Sandoval, Novel hardware architecture for implementing the inner loop of the SHA-2 algorithms, in: 14th Euromicro Conference on Digital System Design (DSD), IEEE Computer Society, Oulu, Finland, 2011, pp. 543–549.
[15] Federal Information Processing Standards Publication 180-2, Announcing the Secure Hash Standard. US DoC/NIST, August 2002.
[16] T. Wollinger, J. Guajardo, C. Paar, Security on FPGAs: state-of-the-art implementations and attacks, ACM Transactions in Embedded Computing Systems 3 (3) (2004) 534–574. http://dx.doi.org/10.1145/1015047.1015052.
[17] R. Glabb, L. Imbert, G. Jullien, A. Tisserand, N. Veyrat-Charvillon, Journal of Systems Architecture 53 (2–3) (2007) 127–138. http://dx.doi.org/10.1016/j.sysarc.2006.09.006.
[18] R.P. McEvoy, F.M. Crowe, C.C. Murphy, W.P. Marnane, Optimisation of the SHA-2 family of hash functions on FPGAs, in: IEEE Computer Society Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures (ISVLSI'06), 2006, pp. 317–322.
[19] Helion Technology Limited, Fast SHA-256 hash core for Xilinx FPGA, datasheet. Revision 1.4, 2008.
[20] Helion Technology Limited, Tiny hash core family for Xilinx FPGA, datasheet. Revision 2.0, 2010.
[21] S. Ducloyer, R. Vaslin, G. Gogniat, E. Wanderley, Hardware implementation of a multi-mode hash architecture for MD5, SHA-1 and SHA-2, in: Workshop on Design and Architectures for Signal and Image Processing, 2007.
[22] M. Togan, A. Floarea, G. Budariu, Design and implementation of cryptographic modules on FPGA, in: Proceedings of the Applied Mathematics and Informatics, 2010, pp. 149–154.
[23] N. Sklavos, O. Koufopavlou, On the hardware implementations of the SHA-2 (256, 384, 512) hash functions, in: Proceedings of IEEE International Symposium on Circuits & Systems (IEEE ISCAS'03), 2003, pp. 153–156.
[24] M. Aşkar, T. Şiltu Çelebi, Design and FPGA implementation of hash processor, in: Proceedings of the Information Security & Cryptology Conference with International Participation, 2007, pp. 85–89.
[25] K. Ting, S. Yuen, K. Lee, P. Leong, An FPGA based SHA-256 processor, in: M. Glesner, P. Zipf, M. Renovell (Eds.), Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream, Lecture Notes in Computer Science, vol. 24, Springer, Berlin/Heidelberg, 2002, pp. 449–471.
[26] M. Juliato, C. Gebotys, Tailoring a reconfigurable platform to SHA-256 and HMAC through custom instructions and peripherals, in: International Conference on Reconfigurable Computing and FPGAs, IEEE Computer Society, Los Alamitos, CA, USA, 2009, pp. 195–200. http://doi.ieeecomputersociety.org/10.1109/ReConFig.2009.40.

Ignacio Algredo-Badillo received the B.Eng. from the Instituto Tecnologico de Puebla, Mexico, in 2002. He received the M.Sc. degree from National Institute for Astrophysics, Optics, and Electronics (INAOE), Puebla, Mexico, in 2004. In 2008, he received the Ph.D. degree at INAOE. Currently, he is a professor-researcher at the University of Istmo in Oaxaca, Mexico. He has involved in the design and development of cryptographic systems, reconfigurable architectures, software radio platforms, FPGA implementations, and application specific hardware acceleration.

Claudia Feregrino-Uribe is a researcher at the Computer Science Department at the National Institute for Astrophysics, Optics and Electronics (INAOE) in Puebla, Mexico. She received the M.Sc. degree from Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV), Guadalajara and the Ph.D. degree from Loughborough University in the United Kingdom. Her research areas are Data Compression, Cryptography and Steganography (Watermarking), Digital Systems Design, applications of FPGAs. She has served as a PC member for several conferences/workshops and as associate editor of the International Journal of Reconfigurable Computing. She is member of the Researchers National System in Mexico and is founder member for the IEEE Puebla Computer Society Chapter.

Rene Cumplido received the B.Eng. from the Instituto Tecnologico de Queretaro, Mexico, in 1995. He received the M.Sc. degree from CINVESTAV Guadalajara, Mexico, in 1997 and the Ph.D. degree from Loughborough Univerity, UK in 2001. Since 2002 he is a professor at the Computer Science Department at INAOE in Puebla, Mexico. His research interests include the use of FPGA technologies, custom architectures and digital reconfigurable computing applications. He is co-founder and Chair of the ReConFig international conference and founder editor-in-chief of the International Journal of Reconfigurable Computing.



Miguel Morales-Sandoval received the B.Sc. degree in Computer Science from the University of Puebla and the M.Sc. and Ph.D. degree in Computer Science from the National Institute for Astrophysics, Optics and Electronics (INAOE) in 2004 and 2008 respectively. Currently he is a professor researcher at Information Technology department in the Polytechnic University of Victoria, Mexico. His research areas include Public Key Cryptography, Elliptic Curve Cryptography, Data Compression, Reconfigurable Computing, and HDL Hardware Design.