Computers and Electrical Engineering 40 (2014) 194–202

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

A compact FPGA-based processor for the Secure Hash Algorithm SHA-256 ☆

Rommel García [a], Ignacio Algredo-Badillo [a], Miguel Morales-Sandoval [b], Claudia Feregrino-Uribe [c], René Cumplido [c,*]

[a] Universidad del Istmo, Tehuantepec, Oaxaca, Mexico
[b] CINVESTAV – Tamaulipas, Laboratorio de Tecnologías de la Información, Mexico
[c] Instituto Nacional de Astrofísica, Óptica, y Elctrónica, Coordinación de Ciencias Computacionales, Mexico

ARTICLE INFO

Article history:
Available online 7 December 2013

ABSTRACT

This work reports an efficient and compact FPGA processor for the SHA-256 algorithm. The novel processor architecture is based on a custom datapath that exploits the reusing of modules, having as main component a 4-input Arithmetic-Logic Unit not previously reported. This ALU is designed as a result of studying the type of operations in the SHA algorithm, their execution sequence and the associated dataflow. The processor hardware architecture was modeled in VHDL and implemented in FPGAs. The results obtained from the implementation in a Virtex5 device demonstrate that the proposed design uses fewer resources achieving higher performance and efficiency, outperforming previous approaches in the literature focused on compact designs, saving around 60% FPGA slices with an increased throughput (Mbps) and efficiency (Mbps/Slice). The proposed SHA processor is well suited for applications like Wi-Fi, TMP (Trusted Mobile Platform), and MTM (Mobile Trusted Module), where the data transfer speed is around 50 Mbps.

1. Introduction

Traditionally, cryptographic algorithms have been considered slow, demanding high computational resources and inefficiently implemented in conventional general purpose processors [1,2]. That fact has motivated the design and implementation of dedicated computing architectures that allow to accelerate the processing time and increase the performance expressed as mega bits per second (Mbps). These custom architectures in general can be classified according to two designing approaches: processor and co-processor. In the former approach, the aim is to provide the minimum hardware that can be used to execute a finite set of machine instructions that, according to a program executes the cryptographic algorithm. On the contrary, in the later approach the aim is to exploit the parallelism in data and execute most of the involved operations in the algorithm directly in hardware. So, while the processor approach is more oriented to use less amount of area resources, the co-processor approach is more oriented to perform the algorithmic operations faster.

Nowadays, with the explosion in the use of mobile devices, such as cellular phones, PDAs, smartphones, and tablets, new applications have emerged but also, several risks and threats to the security of such systems have arisen. In this context, the mobile applications demand security building blocks implemented inside the application itself, which is known as

☆ Reviews processed and approved for publication by Editor-in-Chief Dr. Manu Malek.

* Corresponding author. Tel.: +52 222 2663100x8225; fax: +52 222 2663152.
E-mail addresses: rommelgh@gmail.com (R. García), mmorales@tamps.cinvestav.mx (M. Morales-Sandoval), cferegrino@inaoep.mx (C. Feregrino-Uribe), rcumplido@inaoep.mx (R. Cumplido).

0045-7906/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compeleceng.2013.11.014

embedded security [3,4]. Due to the fact that mobile devices are computationally constrained if compared against desktop computers or workstations, it is necessary that the implementation of the security services in the mobile applications compromise amount of area resources, performance, power consumption, and clock frequency.

This work describes a hardware architecture based on the processor approach, previously mentioned, to execute the SHA-256 algorithm [5], which is a state of the art algorithm to compute a hash or digest for a piece of binary information, allowing to offer the security services of integrity and authentication by implementing digital signature schemes or message authentication code algorithms (MAC) [6]. Despite currently new hash algorithms are being evaluated to select the next standard SHA-3 for hashing [7,8], the SHA-2 family, and particularly the SHA-256 algorithm, provides enough security level to be considered in the next years mainly for security on constrained devices. The main target application of the proposed processor is for the mobile applications Wi-Fi, TMP (Trusted Mobile Platform), and MTM (Mobile Trusted Module), where the required performance is up to 50 Mbps.

The methodology used in this work is focused in the analysis of each operation involved in the SHA-256 algorithm and its associated data dependency, that allows to design a customized datapath that favors data reusing, minimizes memory access, and increases the amount of processed data per clock cycle. From this analysis, a reduced number of basic operations was determined, and the corresponding datapath and arithmetic and logic units were designed. An iterative design methodology was applied, allowing to refine the designs at each iteration in order to decrease the critical path and reduce hardware resources. The main contributions of this work are:

1. A novel architectural design of a processor for the SHA-256 algorithm, having as a core module in its datapath a 4-input arithmetic and logic unit not previously reported.
2. A novel compact SHA-256 processor, occupying the least amount of area resources reported for FPGA implementations, consuming only 139 slices in Virtex5 FPGA and saving around 60% slices compared to related works.
3. A novel compact SHA-256 processor with the best efficiency compared to previous approaches in the literature of compact designs, reaching 0.84 Mbps/Slice.

In the literature, FPGA-based implementations [9–14] have focused in executing the SHA-256 algorithm as fast as possible, computing a SHA-round during a clock cycle and using techniques such as pipelining, unrolling, operation reordering, retiming and unfolding [15]. On the contrary, the approach in this work is to design a compact implementation suitable for mobile applications. Although in the literature exist reported compact FPGA design for the SHA algorithm [16–20], the novel processor architecture we propose based on a 4-input custom ALU allows to obtain designs using still fewer FPGA resources while achieving both higher performance and efficiency. Our design is 64% and 66% more compact than the designs presented in [16,14] respectively.

The rest of this document is organized as follows. Section 2 overviews the SHA-256 algorithm, Section 3 describes the hardware architecture of the proposed processor. Section 4 discusses the results and comparisons against related works. Finally, Section 5 points out the conclusion of this work.

2. SHA-256 algorithm

The SHA-2 family was published in 2002 by the National Institute of Standards and Technology (NIST) [5]. This family is a more robust version than its predecessors SHA-0 and SHA-1. SHA-256 is an algorithm specified in the SHA-2 family, sharing similar functionality with other versions with higher security such as SHA-384 and SHA-512 (see Table 1). It computes the digest of an arbitrary length message in the following way. The input message m is padded with one '1' and leading '0's until the message length (in bits) becomes a multiple of 512. The last 64 bits in the padded message are used to store the length of the original message as a 64-bit number. After the padding, the resulting message is divided into blocks of length 512-bits $D^{(1)}, D^{(2)}, \ldots, D^{(N)}$. Each block of data $D^{(i)}$ is processed sequentially by a main function during 64 rounds. A partial 256-bit hash value H_i is obtained as the current $D^{(i)}$ data block is totally processed. After computing the last data block $D^{(N)}$, the final hash H_N is computed and delivered. Algorithm 1 lists the general functioning of SHA-256 algorithm.

Table 1
Characteristics of SHA-2 family algorithms.

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

Algorithm 1. SHA-256 main loop

Require: $D^{(i)}$ a 512-bit block
Ensure: The HASH value H_i corresponding to $D^{(i)}$ from $H_{(i-1)}$
1: **for** t from 0 to 63 **do**
2: Prepare W_t
3: Compute $\sum_0(a)$
4: Compute $Maj(a,b,c)$
5: Compute T_2
6: Compute $\sum_1(e)$
7: Compute $Ch(e,f,g)$
8: Compute T_1
9: $h \leftarrow g$
10: $g \leftarrow f$
11: $f \leftarrow e$
12: $e \leftarrow d + T_1$
13: $d \leftarrow c$
14: $c \leftarrow b$
15: $b \leftarrow a$
16: $a \leftarrow T_1 + T_2$
17: **end for**
18: $H_{temp} = a|b|c|d|e|f|g|h$
19: $H^{(i)} \leftarrow H^{(i-1)} + H_{temp}$
20: return $H^{(i)}$

All the functions involved in the SHA-256 algorithms are computed according to the Eqs. (1)–(4).

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{1}$$

$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{2}$$

$$\sum_0(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \tag{3}$$

$$\sum_1(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \tag{4}$$

The main loop processing the $D^{(i)}$ data block uses eight 32-bit working variables a,b,c,d,e,f,h and g, forming a buffer state, that after the 64 rounds becomes the partial hash value H_{temp}. This hash value together with the hash computed for the previous data block is added to get the hash value corresponding to the message until the data block $D^{(i)}$. When the first data block is processed, an initial hash value H_0 is used. At the beginning of the computation of $H_{(i+1)}$, the value of H_i needs to be loaded into the buffer state.

At each iteration t in the hash algorithm a constant K_t is required, which is well defined in the SHA-256 specification. The temporal variables used in Algorithm 1 named T_0 and T_1 are computed as specified in Eqs. (5) and (6) respectively.

$$T_1 = h + \sum_1^{256}(e) + Ch(e,f,g) + K_t^{256} + W_t \tag{5}$$

$$T_2 = \sum_0^{256}(a) + Maj(a,b,c) \tag{6}$$

The 32-bit value called W_t, that is used at each iteration in Algorithm 1, is computed as follows. The first 16 values for W_t during the first 16 iterations are taken directly from the 512-bit message. For the remaining iterations, the values for W_t are computed using functions σ_0 and σ_1 (see Eqs. (7)–(9)).

$$Wt = \begin{cases} M_t^{(i)} & 0 \leqslant t \leqslant 15 \\ \sigma_1(W_{t-2}) \oplus W_{t-7} \oplus \sigma_0(W_{t-15}) \oplus W_{t-16} & 16 \leqslant t \leqslant 63 \end{cases} \tag{7}$$

$$\sigma_0(x) = \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) \tag{8}$$

$$\sigma_1(x) = \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x) \tag{9}$$

3. Proposed architecture

The key idea in the hardware organization of the proposed processor for the SHA-256 algorithm was to reuse data, minimize critical paths and reduce the memory access by using cache memory. Initially, the SHA-256 processor was designed containing a simplified 2-input ALU with the main objective to reduce area consumption. The expectation was that a small ALU considering only two operands would lead to a compact design of the SHA processor. However, during the design process and based on a study on the type of operations in the SHA algorithm, their execution sequence and the associated dataflow, we realize that a more compact design could be achieved by designing a 4-input ALU, reducing clock cycles and memory resources for intermediate results. A 4-input ALU is well suited for implementation in FPGAs, mapping the logic to the 4-in LUTs (Look up tables) included in them.

The proposed architecture for the SHA-256 processor consists of the following main modules: a control unit, a datapath, a bank of 32-bit registers, and two ROM memory blocks. The architectural hardware design is based on an analysis of the operations involved in the algorithm itself. Such analysis allowed to identify the kind and sequence of operations to design a specialized 4-input ALU, as well as internal cache memory to speed up the data access and reduce read/write cycles to the register bank. The block diagram of the SHA processor is shown in Fig. 1.

Lets consider again the main operations computed in the internal rounds of the SHA-256 algorithm. The equations for computing W_t, T_1, T_2, a, and e can be conveniently rewritten, leading to a new set of equations of the form $f = w + x + y + z$, without altering the functioning of the SHA algorithm. The main goal in this new representation is to define basic instructions that can be efficiently computed in a specialized 4-input ALU. The rewritten equations are (10)–(14).

$$T_x = \sum_1(e) + Ch(e,f,g) + W_t + 0 \tag{10}$$

$$T_y = h + K_t + d + T_x \tag{11}$$

$$T_z = \sum_0(a) + Maj(a,b,c) + (T_y - 0) + 0 \tag{12}$$

$$e = T_y \tag{13}$$

$$a = T_z \tag{14}$$

The datapath for the proposed SHA-256 processor is shown in Fig. 2. A customized ALU named HASH_ALU computes Eqs. (10)–(14). Additionally, a smaller arithmetic unit named AU performs sums of two operands and transfers the intermediate hash value H_i to the buffer state (the working variables a–h). The AU module is composed of an adder and a multiplexer that allows to select the result between the incoming data (a transfer) or the adder result. The R module is a forwarding unit that

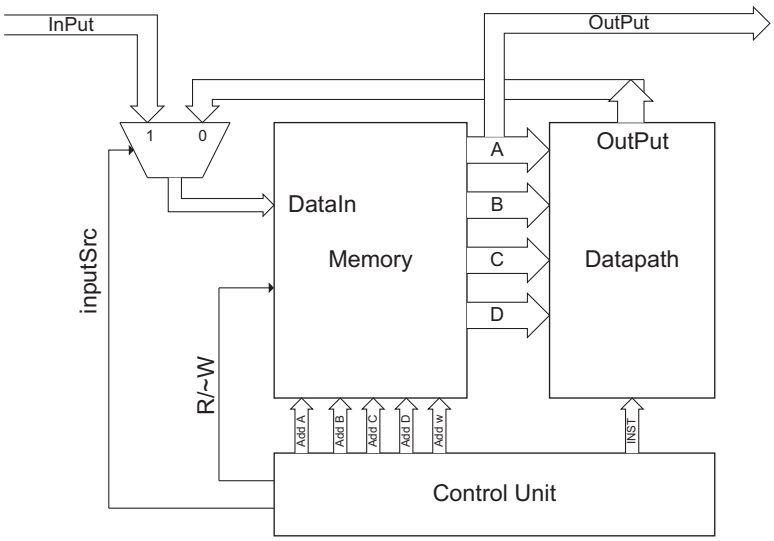

Fig. 1. Block diagram of the SHA-256 processor.

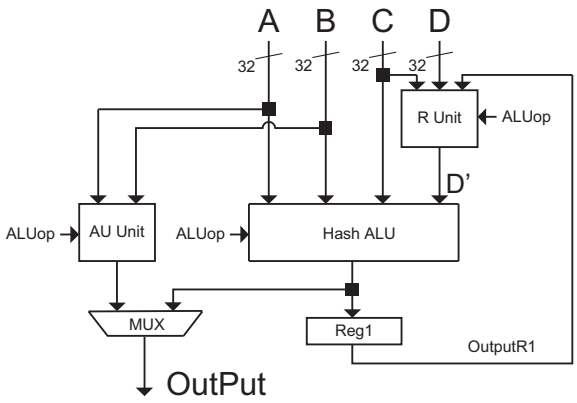

Fig. 2. Datapath for computing basic operations in the SHA-256 algorithm.

allows to feedback data obtained one previous iteration to the HASH_ALU. This functionality reduces the number of memory access and avoids the unnecessary memory addressing and all the related logic. Another responsibility for this module is to compute the operation $T_y - d$. So, the R module is composed of a 32-bit register, a subtracter, and a multiplexer. The values that can be propagated by the R unit are: the incoming operand D (see Fig. 2) and the subtraction between the incoming data R_1 and the internal register R_2. The R_2 register stores the incoming operand C from the previous iteration. The block diagrams of the HASH_ALU, AU and R modules are depicted in Fig. 3.

During the execution of an internal round of the SHA-256 algorithm in the interval $0 \leqslant t < 16$, the datapath contains the values shown in Table 2. When $16 \leqslant t < 64$, the data flow in the datapath is as shown in Table 3.

The control unit generates the four addresses for the operands incoming to the HASH_ALU unit as well as the address for the destination register, where the result obtained from the ALU is stored. In addition, the control unit orchestrates the data source to the ALU. During the first 16 cycles the data is taken from the exterior, that is, the first 16 32-bit words are taken from the 512 data block $D^{(i)}$ to be hashed. Once the 512-bit data block is loaded, the control unit transfers the content of the intermediate hash H_i to the buffer state (registers a–h). For the first block, such intermediate hash value is the initial hash value H_0, which is taken from a memory that stores constants. For the next rounds and data blocks, H_i is computed from the registers a to h and the previous $H_{(i-1)}$.

The control unit is a finite state machine composed of 9 states:

S_0 – The initial state waits for new message to be hashed.
S_1 – Loads the 16 words of the input message, one at each clock cycle.
S_2 – Loads the initial hash H_0 or update the intermediate hash value H_i.
S_3 – Computes W_t, being $W_t = M_t$ for the first 16 rounds. After that, W_t is computed using the HASH_ALU (see Fig. 3).
S_4 – Computes T_x the result is not stored in memory.

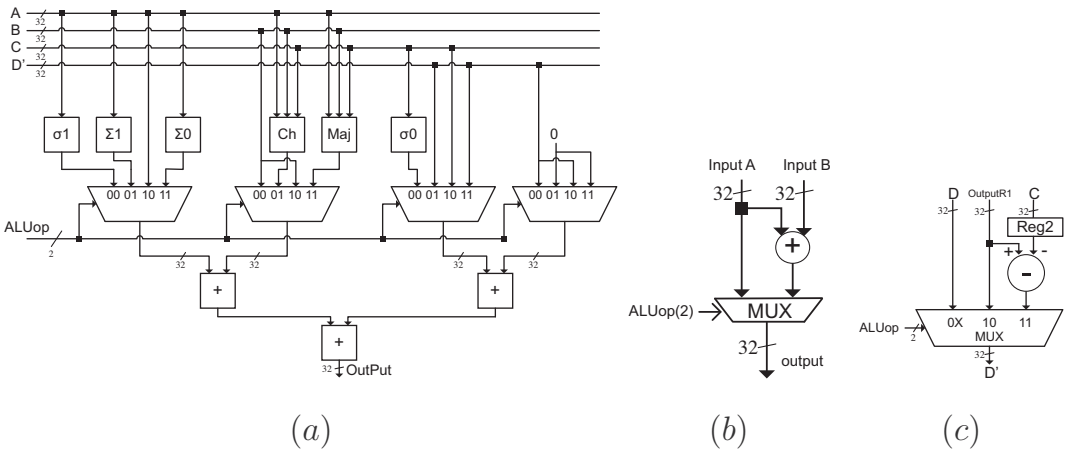

Fig. 3. Internal structure of main modules in the datapath for the proposed SHA-256 processor. (a) the HASH_ALU unit, (b) the AU unit, and (c) the R unit.

Table 2
Operations to compute the first 16 rounds in the SHA-256 algorithm.

Step	Operation	R1	R2
1	$\sum_1(e) + Ch(e,f,g) + M_t + 0$	"xxxxxxxx"	"xxxxxxxx"
2	$h + k_t + d + T_x$	T_x	g
3	$\sum_0(a) + Maj(a,b,c) + (T_y - d) + 0$	T_y	d

Table 3
Operations to compute the last 48 rounds in the SHA-256 algorithm.

Step	Operation	R1	R2
1	$\sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$	"xxxxxxxx"	"xxxxxxxx"
2	$\sum_1(e) + Ch(e,f,g) + W_t + 0$	W_t	W_{t-15}
3	$h + k_t + d + T_x$	T_x	g
4	$\sum_0(a) + Maj(a,b,c) + (T_y - d) + 0$	T_y	d

S_5 – Computes T_y. First computes $d + T_1$ (see equation in Section 2) and stores the result in the address for d, not for e as the algorithm indicates. This is done because the algorithm indicates that each variable will contain the value from the variable that precedes it. The control unit rotate the addresses of the variables in order to avoid unnecessary assignments and save time.
S_6 – Computes T_z and stores the result in the variable h. Again, the same reasoning applied in the state S_5 is used to avoid unnecessary assignments and save time.
S_7 – Computes the intermediate hash value H_i.
S_8 – All data blocks are processed. The final value in H_N is taken and placed in the dataout bus.

4. Results

The proposed SHA-256 compact processor was described using VHDL. The Xilinx ISE 13.2 tools were used for implementing the VHDL design in a Xilinx Virtex-5 FPGA. ISE tools allow to calculate the utilized hardware resources and to determine the maximum clock frequency, useful parameters for measuring the Throughput (see Eq. (15)) and Efficiency of Performance (see Eq. (16)).

$$\text{Throughput} = \frac{\text{Data block size}}{\text{Clock time} \times \text{Clock cycles}} \tag{15}$$

$$\text{Efficiency} = \frac{\text{Throughput}}{\text{Number of slices}} \tag{16}$$

The FPGA implementation results are shown in Table 4. The SHA-256 processor processes a 512-bit data block with a latency of 280 clock cycles:

- 16 Cycles are used to load the 512-bit data block.
- 8 Cycles are used to transfer the intermediate hash value H_i to the working variables a–h.
- The internal loop (64 rounds) are computed in two sequences:
  - During the first 16 rounds W_t is taken from the incoming 512-bit data block and only 3 cycles are used.
  - For the next 48 rounds, W_t is computed using the HASH_ALU, requiring 4 cycles per round.
- After the internal loop, the intermediate hash value is computed. This process takes 8 cycles.
- Finally, it takes 8 cycles to output the final hash value.

Table 4
Implementation results.

FPGA	xc5vlx50t-3ff1136
Frequency (MHz)	64.45
Slices	139
LUTs	527
Latency	280
Throughput (Mbps)	117.85
Efficiency (Mbps/Slice)	0.84

The VHDL design was debugged analyzing the data flow and checking the state of every variable at each round of the algorithm against values taken from a software implementation. Also, the implemented processor was validated applying testbeds for computing hash values of different files and comparing the results against the ones obtained from the software counterpart.

Our proposed architecture compromises performance and area usage, reusing the hardware as much as possible to keep a high efficiency. The hardware architecture proposed in this work is based on the design principles for computer architecture [21]: (i) Simplicity favors regularity, (ii) smaller and simpler is faster, (iii) make the common case fast, (iv) good design demands good compromises.

From the architectural point of view, our proposal is quite different from previous approaches to construct compact SHA processors. The hardware architecture presented in [17] (which is almost the same presented in [19]) uses a 2-input ALU containing only one adder. This module is re-used, taking the input data from two 5-1 multiplexers. Their design uses three RAM memories where variables and constants are stored. Also, other combinatorial independent modules are required to compute the required operations in the SHA algorithm. On the contrary, our design uses only one memory and a single ALU computing all the required operations in the SHA algorithm. The hardware architectures presented in [16,18] also contain independent specialized modules (generator, compressor and controller separately). In that design, there is not memory for storing the buffer state containing the current hash value. Instead, a set of registers are used. The module for computing the hash value uses only one adder, which is re-used by multiplexing its inputs with 2-1 and 7-1 multiplexers. The generator module uses two adders, one memory and one register.

The architecture presented in [19] also re-uses only one adder and keeps independent modules for computing the operations in the SHA algorithm. That design requires three memories for storing data being processed at each round, buffer state and constants. The adder takes its arguments from two sources selected by 5-1 multiplexers. In [20], the reported architecture is more complex, keeping the data at each round stored in registers, and the figure of an integrated ALU is not present.

Although it is hard to compare different FPGA implementations due to the different technologies used, we attempt to provide a comparison as fair as possible in Table 5 among representative FPGA implementations of SHA algorithm under the same conditions.

In Table 5, comparison against non-campact architectures [9,11,10,22,12–14] is just a reference to show the saving in area resources that can be achieved. However, since [9,10] are implemented in CMOS technology it is hard to compare those implementation results against our FPGA results. Also, [11] is implemented in a different FPGA family so area usage (GE vs. Slice) cannot be directly compared. The results presented in [22] are from a Intel general purpose processor (software implementation) so comparisons are also not possible. The amount of area reported in [13] is quite big, our design being more compact requires 60% less FPGA resources compared to that work. Compared to [14], our approach requires 66% less area resources, although this saving comes with a penalty in the performance. However, considering both metrics, area and performance, our design is 2.18 more efficient.

Regarding the compact designs [17,16,18–20] previously reported, our design uses fewer area resources with a better compromise area/performance. Compared to all those works our design requires less clock cycles and the operational frequency is lower, which can lead to reduced power consumption, a well desired characteristic in compact implementations. The results presented in [17,19] are from ASIC implementations. In practice, an ASIC implementation is faster than an FPGA implementation. However, compared to [17,19] our design achieves a performance three times better operating at a lower frequency. The SHA processor presented in [18] and implemented in a Virtex-II XC2V2000 requires 779 slices, 490 clock cycles with a clock frequency of 71.5 MHz achieving 74.7 Mbps. In contrast, we require 45% less area resources, 42% less clock cycles, with a lower clock frequency of 35.5 MHz. In terms of efficiency, our design is better obtaining 0.150 Mbps/Slice

Table 5
Comparison results.

Work	Platform	Campact?	Hardware resources	Clock freq. (MHz)	Clock cycles	Performance (Mbps)
[9]	CMOS 0.13 mm	No	22025 Gates	793.6	68	5975
[10]	CMOS 0.13 mm	No	22025 Gates	793.6	68	5975
[11]	Stratix EP1S10F484C5	No	104760 GEs	74	65	595
[12]	Virtex XCV200	No	1306 Slices	77	66	308
[13]	Virtex XCV200	No	1060 Slices 1 BRAM	83	–	326
[14]	Virtex XCV300	No	1261 Slices	88	73	87
[22]	Intel core	No	–	–	–	18.62 – 15.31 Cycles/byte
[16]	Virtex2 XC2VP20	Yes	1210 Slices	85	355	122.6
[17]	CMOS 0.35 mm	Yes	10868 GEs	50	1128	22.5
[18]	Virtex-II XC2V2000	Yes	779 Slices	71.5	490	74.7
[19]	CMOS 0.35 mm	Yes	10868 GEs	50	1128	22.5
[20]	Virtex-II	Yes	639 Slices	85	1120	–
	Virtex-4		615 Slices	102	1120	–
This work	Virtex XC2VP	Yes	431 Slices	35.50	280	64.91
	VIrtex-4 LX		422 Slices	50.06	280	91.53
	Virtex-5 VLX		139 Slices	64.45	280	117.8

compared to 0.096 Mbps/Slice obtained by [18]. Similarly, our design uses 64% less area resources than [16] although with the cost of a decrease in the performance but with better efficiency (0.1 vs. 0.15). Compared to [20], we use over 30% less area resources under the same FPGA technology. In terms of performance, we achieve almost double of the throughput reported in [20]. The basic idea in [20] is the design of an unrolled architecture, reusing hardware and restricting operations to only 8 bits. However, that hardware reusing approach increases the area usage and latency, which affects the overall performance.

The performance achieved by the SHA processor described in this work, which is higher than the one obtained by an ASIC implementation [17,19], is enough for mobile applications such as Wi-Fi, TMP (Trusted Mobile Platform), MTM (Mobile Trusted Module), where the maximum throughput is about 50 Mbps.

5. Conclusion

This work presented and discussed the hardware design of a customized processor for executing the SHA-256 algorithm. The main module is a 4-input ALU and a customized datapath that reuses data, avoids unnecessary access to memory and its FPGA implementation leads to short critical paths and reduced amount of area, around 60% compared to related works. The VHDL description of the processor is well mapped to the 4-in LUTs contained in common FPGAs, making an efficient use of the available area. The resulting design uses fewer area resources than other approaches while keeping a performance suitable for mobile applications like Wi-Fi or IEEE 802.11 networks, where the performance is around 50 Mbps. Our first approach was to design a simple datapath consisting in a small 2-input ALU. However, we needed to rewrite the original equations in the SHA-256 algorithm to derive the customized 4-input ALU and its associated datapath to get a more compact SHA-256 processor. Our design can be easily extended to other hash algorithms of the SHA-2 family since all of them exhibit a similar functionality.

References

[1] Kim H, Lee M-K, Kim D-K, Chung S-K, Chung K. Design and implementation of crypto co-processor and its application to security systems. In: Hao Y, Liu J, Wang Y-P, Cheung Y-m, Yin H, Jiao L, et al., editors. Computational intelligence and security. Lecture notes in computer science, vol. 3802. Berlin Heidelberg: Springer; 2005. p. 1104–9.

[2] Reddy SK, Sakthivel R, Praneet P. VLSI implementation of AES crypto processor for high throughput. In: (IJAEST) International journal of advanced engineering sciences and technologies, vol. 6; 2011. p. 022–6.

[3] Huffmire T, Brotherton B, Sherwood T, Kastner R, Levin T, Nguyen TD, et al. Managing security in FPGA-based embedded systems. IEEE Des Test Comput 2008;25(6):590–8. http://dx.doi.org/10.1109/MDT.2008.166.

[4] Crenne J, Cotret P, Gogniat G, Tessier R, Diguet J-P. Efficient key-dependent message authentication in reconfigurable hardware. In: 2011 International conference on field-programmable technology (FPT'11); 2011. p. 1–6.

[5] National Technical Information Service. FIPS 180-2 – secure hash standard, U.S. Department of Commerce/NIST, Springfield, VA; 2002 <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>.

[6] Sklavos N, Koufopavlou OG. Implementation of the SHA-2 hash family standard using FPGAs. J Supercomput 2005;31(3):227–48.

[7] Regenscheid A, Perlner R, Chang S-j, Kelsey J, Nandi M, Paul S. Status report on the first round of the SHA-3 cryptographic hash algorithm competition. Tech rep, NIST; 2009.

[8] Andreeva E, Mennink B, Preneel B. Security reductions of the second round SHA-3 candidates. In: Proceedings of the 13th international conference on information security, ISC'10. Berlin, Heidelberg: Springer-Verlag; 2011. p. 39–53.

[9] Lee YK, Chan H, Verbauwhede I. Verbauwhede: iteration bound analysis and throughput optimum architecture of SHA-256 (384,512) for hardware implementations. In: Information security applications, 8th international workshop, WISA 2007. Lecture notes in computer science, vol. 4867. Springer-Verlag; 2007. p. 102–14.

[10] Lee Y, Knezevic M, Verbauwhede I. Hardware design for hash functions. In: Verbauwhede IM, editor. Secure integrated circuits and systems, integrated circuits and systems. US: Springer; 2010. p. 79–104. http://dx.doi.org/10.1007/978-0-387-71829-3_5.

[11] Mladenov T, Nooshabadi S. Implementation of reconfigurable SHA-2 hardware core. In: IEEE Asia Pacific conference on circuits and systems, 2008. APCCAS 2008; 2008. p. 1802–5.

[12] Glabb R, Imbert L, Jullien G, Tisserand A, Veyrat-Charvillon N. Multi-mode operator for SHA-2 hash functions. J Syst Archit 2007;53(2–3):127–38. http://dx.doi.org/10.1016/j.sysarc.2006.09.006.

[13] Sklavos N, Koufopavlou O. On the hardware implementations of the SHA-2 (256, 384, 512) hash functions. In: Proceedings of the 2003 international symposium on circuits and systems, 2003. ISCAS '03. vol. 5; 2003. p. V-153–V-156. http://dx.doi.org/10.1109/ISCAS.2003.1206214.

[14] Ting K, Yuen S, Lee K, Leong P. An FPGA based SHA-256 processor. In: Glesner M, Zipf P, Renovell M, editors. Field-programmable logic and applications: reconfigurable computing is going mainstream. Lecture notes in computer science, vol. 2438. Berlin/Heidelberg: Springer; 2002. p. 449–71. http://dx.doi.org/10.1007/3-540-46117-5_60.

[15] Shi Z, Ma C, Cote J, Wang B. Hardware implementation of hash functions. In: Tehranipoor M, Wang C, editors. Introduction to hardware security and trust. New York: Springer; 2012. p. 27–50. http://dx.doi.org/10.1007/978-1-4419-8080-9_2.

[16] Kim M, Lee D, Ryou J. Compact and unified hardware architecture for SHA-1 and SHA-256 of trusted mobile computing. Pers Ubiquit Comput 2012:1–12. http://dx.doi.org/10.1007/s00779-012-0543-0.

[17] Feldhofer M, Wolkerstorf er J. Hardware implementation of symmetric algorithms for RFID security. In: K itsos P, Zhang Y, editors. RFID security. US: Springer; 2009. p. 373–415. http://dx.doi.org/10.1007/978-0-387-76481-8_15.

[18] Kim M, Ryou J, Jun S. Efficient hardware architecture of SHA-256 algorithm for trusted mobile computing. In: Yung M, Liu P, Lin D, editors. Information security and cryptology. Lecture notes in computer science, vol. 5487. Berlin Heidelberg: Springer; 2009. p. 240–52. http://dx.doi.org/10.1007/978-3-642-01440-6_19.

[19] Feldhofer M, Rechberger C. A case against currently used hash functions in RFID protocols. In: Meersman R, Tari Z, Herrero P, editors. On the move to meaningful internet systems 2006 – OTM 2006. Lecture notes in computer science, vol. 4277. Montpellier, France: Springer; 2006. p. 372–81.

[20] Cao X, Lu L, O'Neill M. A compact SHA-256 architecture for RFID tags. In: Proceedings of the 22nd IET irish signals and systems conference, ISSC, Trinity College Dublin; 2011.

[21] Patterson DA, Hennessy JL. Computer organization and design – the hardware/software interface. The Morgan Kaufmann series in computer architecture and design. Academic Press; 2012.

[22] Gueron S. Speeding up SHA-1, SHA-256 and SHA-512 on the 2nd generation Intel core processors. In: Ninth international conference on Information Technology: New Generations (ITNG), 2012; 2012. p. 824–6.

Rommel García Hernández was born in Oaxaca, México. He has a Bachelors degree in Computer Engineering from the University of Isthmus, Oaxaca, México, in 2013. His current research interests include Information Security, Reconfigurable Computing and high performance cryptography processor design.

Ignacio Algredo-Badillo received the B.Eng. in Electronic Engineering from Technologic Institute of Puebla and the M.Sc and Ph.D degrees in Computer Science from National Institute for Astrophysics, Optics and Electronics. Since 2009, he has been professor of Computer Engineering at University of Istmo and involved in the design of high-performance digital systems, reconfigurable architectures, SDR, cryptographic systems, FPGAs, and microcontrollers.

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 with the Information Technology Lab at Cinvestav-Tamaulipas, Mexico. His research areas include FPGA-based hardware architectures for public key cryptography.

Claudia Feregrino-Uribe received the M.Sc. degree from CINVESTAV, Mexico in 1997 and PhD degree from Loughborough University, UK, in 2001. Since 2002 she is a researcher at Computer Science Department at INAOE. Her research interests include cryptography and watermarking. She has been PC member for several conferences and associate editor of international journals and has published more than 85 articles.

René Cumplido is a Professor at the Computer Science Department at INAOE. He holds a B.Sc. degree in Computer Systems, a M.Sc. in Electrical Engineering and a Ph.D. Electrical Engineering from Loughborough University, UK. His research interests are Reconfigurable Computing for DSP and Digital Communications, FPGA Technologies, Custom Architectures for Scientific Computing.