I.J. Computer Network and Information Security, **2015, 1, 9-15**
Published Online December 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2015.01.02

Efficient FPGA Hardware Implementation of Secure Hash Function SHA-2

Hassen Mestiri
Electronics and Micro-Electronics Laboratory (E. µ. E. L) Faculty of Sciences of Monastir, Tunisia
Email: Hassen.mestiri@yahoo.fr

Fatma Kahri
Electronics and Micro-Electronics Laboratory (E. µ. E. L) Faculty of Sciences of Monastir, Tunisia
Email: kahrifatma@gmail.com

Belgacem Bouallegue
Electronics and Micro-Electronics Laboratory (E. µ. E. L) Faculty of Sciences of Monastir, Tunisia
Email: belgacem_bouallegue@yahoo.fr

Mohsen Machhout
Electronics and Micro-Electronics Laboratory (E. µ. E. L) Faculty of Sciences of Monastir, Tunisia
Email: mohsen.machhout@fsm.rnu.tn

Abstract—The Hash function has been studied by designers with the goal to improve its performances in terms of area, frequency and throughput. The Hash function is used in many embedded systems to provide security. It is become the default choice for security services in numerous applications. In this paper, we proposed a new design for the SHA-256 and SHA-512 functions. Moreover, the proposed design has been implemented on Xilinx Virtex-5 FPGA. Its area, frequency and throughput have been compared and it is shown that the proposed design achieves good performance in term of area, frequency and throughput.

Index Terms—Security, SHA-256, SHA-512, FPGA Implementation.

I. Introduction

The SHA-2 was finalized in 2009 by the National Institute of Standards and Technology (NIST) [1]. The SHA-2 algorithm replaced the SHA-1, which had been in use since 1995. Until now, many architectures, for efficient VLSI realization of SHA-2 algorithm, have been proposed and their performance have been evaluated by using ASIC libraries and FPGA [2-11].

Cryptographic algorithm SHA-2 is currently used in a very large variety of scenarios. The most common examples: e-commerce and financial transactions, which have strong security requirements. Thus, the necessity to protect the hardware implementation against the Cryptographic attacks [12-14].

In this paper, we propose a reconfigurable design for the SHA-256 and SHA-512 algorithms. We present its details implementation. Experimental synthesis results show that the proposed architecture achieves a high performance in term of area, frequency and throughput.

The organization of this paper is as follows. Section II describes the related background knowledge. Section III presents describes the SHA-256 and SHA-512 algorithms. Section IV presents the proposed a reconfigurable design for the SHA-256 and SHA-512 algorithms. Their experimental synthesis results and performances are discussed and compared in terms of area, frequency, throughput and efficiency are presented in section V. Section VI concludes the paper.

II. Background

Some descriptions of SHA-1, SHA-256 and SHA-512 algorithms can be found in the official NIST standard [1]. Table 1 shows a comparative study of three hash functions characteristics. The security of these hash functions is controlled by the size of their outputs, referred to as hash values. All functions have a similar internal structure and process each message block using multiple rounds. These hash functions enable the determination of a message's integrity: any change to the message will result in a different produced message digest, with a very high probability.

Table 1: Functional characteristics of two family hash functions

Hash Function	SHA-1	SHA-2	
		SHA-256	SHA-512
Size of hash value (n)	160	256	512
Message size	2^{64}	2^{64}	2^{128}
Message block size (m)	512	512	1024
Word size	32	32	64
Numbers of words	5	8	8
Digest rounds number	80	64	80

III. SHA-2 Description

A. General

SHA-256 accepts messages with arbitrary lengths up to 264-bit. The SHA-256 Hash function produces a final digest message of 256 bits that is dependent of the input message, composed by multiple blocks of 512-bit each. This input block is expanded and fed to the 64 cycles of the SHA-256 function in words of 32-bit each (denoted by K_t,W_t). Intermediate hash values are rerouted back into the compression loop.

B. SHA-256 Description

B.1. Preprocessing

As with other popular hashing functions, with SHA-256 the message to be hashed is first padded so that its final length is a multiple of 512-bit. The n-bit message is padded so that a single 1-bit is added into the end of the message. Then, 0 bits are added until the length of the message is congruent to 448 modulo 512. A 64-bit representation of n is appended to the result of the padding. Thus, the result message is a multiplicity of 512-bit. This message is denoted here as M(i). M(i) message blocks are passed individually to the message expander. Padding can be represented as:

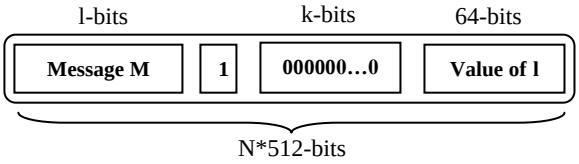

Fig. 1: Message preprocessing (SHA-256)

B.2. Definitions

In the SHA-256 algorithm, six logical functions which operate on 32-bit values are used:

$$Ch(x, y, z) = (x \wedge y) \oplus (\sim x \wedge y) \quad (1)$$
$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (2)$$
$$\sum_0(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \quad (3)$$
$$\sum_1(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \quad (4)$$
$$\sigma_0(x) = ROTR^7(x) \oplus ROTR^1(x) \oplus SHR^3(x) \quad (5)$$
$$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \quad (6)$$

Where $\wedge$, $\sim$ and $\oplus$ are the bitwise AND, NOT and XOR operations.

ROTR and SHR are the rotate right and shift right functions respectively.

B.3. Algorithm

The message, M is expended by a message Scheduler according to the following function:

For j = 0 to 15: $W = M_j^{(i)}$ and
For J = 16 to 63{
$W_j \leftarrow \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$ }

For i=1 to N
{Initialize registers a, b, c, d, e, f, g, h with the $(i-1)^{st}$ intermediate hash value.
Apply the following compression function to registers a-h:

For j= 0 to 63{
$T_1 \leftarrow h + \sum_1(e) + Ch(a, b, c) + K_j + W_j$
$T_2 \leftarrow \sum_0(a) + Maj(a, b, c)$
$h \leftarrow g$, $g \leftarrow f$, $f \leftarrow e$, $e \leftarrow d + T_1$
$d \leftarrow c$, $c \leftarrow a$, $b \leftarrow a$, $a \leftarrow T_1 + T_2$}
i^{th} intermediate hash:
$H_1^{(i)} \leftarrow a + H_1^{(i-1)}, \ldots, H_8^{(i)} \leftarrow h + H_8^{(i-1)}$}
The hash of M: $H^{(N)} = (H_1^{(N)}, H_2^{(N)}, \ldots, H_8^{(N)})$ [1]

C. SHA-512 Description

In this section, the architecture SHA-512 is described. Internal structures of these units and sizes of their operands differ in the four versions of SHA-2.

Full descriptions of the SHA-224, -256, -384 and -512 algorithms can be found in the official NIST standard [1]. SHA-512 produces a 512-bit message hash; SHA-384 a 384-bit message hash etc. An overview of SHA-512 is given here, and then the differences between SHA-512 and the other members of the SHA-2 family are outlined. The SHA-512 algorithm essentially consists of 3 stages:

- Message padding unit
- Block expansion
- Round computation unit.

C.1. Preprocessing

The SHA-2 input block size depends on the algorithm used. The input block size is 1024-bit. The pre-processing stage first splits the original message into N blocks, namely M(1), M(2), …,M(N). Each block has 1024 bits. Then, if the message length is not a multiple of the underlying block size, message padding must be performed. Next, eight initial hash values, $H_0^{(0)}, \ldots, H_7^{(0)}$, are set as listed in the specifications [1]. Each algorithm uses a distinct set of initial hash values. (See Fig.2).

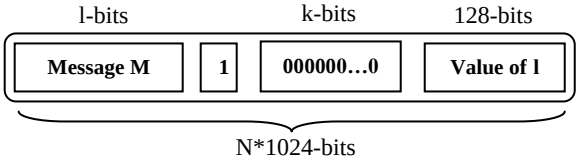

Fig. 2: Message preprocessing (SHA-512)

C.2. Hash Computation

The hash computation is based on operations over 1024-bit words. The number of iterations performed by the algorithm is given by j = 80. Actually, j can be considered to represent the number of 1024-bit words processed by the algorithm. More specially, the SHA-512 algorithms comprise j message schedule words

($W_0,\ldots,W_{80}$), eight working variables (a, b, c, d, e, f, g, h), and eight intermediate hash values ($H_0^{(i)},...,H_7^{(i)}$). As specified in [15], SHA-512 utilize 80 constants ($K_0,\ldots,K_{80}$), where each of them is 1024-bit wide. Additionally, six logical functions are employed, as shown below. ROR n(x) and SHR n(x) correspond to, respectively, a rotation and a shift of x by n bits to the right. Besides, ⊕ represents the bitwise XOR operation, ^ the bitwise AND operation, and ~x the bitwise complement of x.

SHA-512: The hash computation step uses four logical functions: Ch, Maj, Σ_0, and Σ_1. The result of each new function is either a new 64-bit:

$$Ch(x, y, z) = (x \wedge y) \oplus (\sim x \wedge y) \quad (7)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (z \wedge y) \quad (8)$$
$$\sigma_0(x) = ROTR_1(x) \oplus ROTR_8(x) \oplus SHR_7(x) \quad (9)$$
$$\sigma_1(x) = ROTR_{19}(x) \oplus ROTR_{61}(x) \oplus SHR_6(x) \quad (10)$$
$$\Sigma_0 = ROTR_2(x) \oplus ROTR_{34} \oplus ROTR_{39}(x) \quad (11)$$
$$\Sigma_1 = ROTR_{14}(x) \oplus ROTR_{18} \oplus ROTR_{41}(x) \quad (12)$$

For each message block i, 1< i <N, a four-step digest round is performed as follows:

- Initialize the eight working variables:

$$a = H_0^{(i-1)},\ b = H_1^{(i-1)},\ c = H_2^{(i-1)},\ d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)},\ f = H_5^{(i-1)},\ g = H_6^{(i-1)},\ h = H_7^{(i-1)}$$

- Prepare the message schedule:

$$W_j = ; M_0^{(i-1)}; 0 \le j \le 15 \quad (13)$$
$$W_j = ROTL_1 (W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16}); \quad (14)$$
$$16 \le j \le 79$$

- The SHA-512 algorithm is given in listing 1.

$For\ t = 0\ to\ 79:$
{
$T_1 = h + \Sigma_1^{256}(e) + Ch(e, f, g) + K_j^{256} + W_j$
$T_2 = \Sigma_0^{256}(a) + Maj(a, b, c)$
$h = g$
$g = f$
$f = e$
$e = d + T_1$
$d = c$
$c = b$
$b = a$
$a = T_1 + T_2$
}

Listing 1. SHA-512 algorithm

- Compute the i^{th} intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)},\ H_1^{(i)} = b + H_1^{(i-1)},$$
$$H_2^{(i)} = c + H_2^{(i-1)},\ H_3^{(i)} = d + H_3^{(i-1)},$$
$$H_4^{(i)} = e + H_4^{(i-1)},\ H_5^{(i)} = f + H_5^{(i-1)},$$
$$H_6^{(i)} = g + H_6^{(i-1)},\ H_7^{(i)} = h + H_7^{(i-1)},$$

After all N blocks of message M are processed, the final message digest is obtained by concatenating the hash values ( ) More precisely, the message digest for each algorithm is given by the concatenations shown below. The concatenation of words is represented by the symbol ||.

$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_0^{(N)}$ [1]

IV. Proposed Design

This section presents the proposed reconfigurable SHA-256 and SHA-512 design. Fig. 3 shows the block diagram of our reconfigurable AES. The given architecture supports four operation modes for reconfigurable SHA processor.

- Input interface and output interface: the input data is arbitrary length, but the outputs are 256 or 512 bits for SHA-256 and SHA-512, respectively. So input interface and output interface have to buffer the input and output data during the loading process.
- The Control Unit is designed to control the flow of data in the design, as well as data exchange between the Padded procedure Unit and Hash Computation Unit.

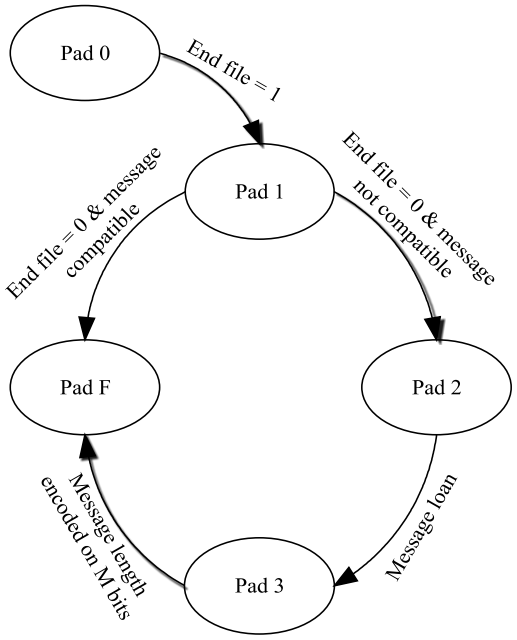

Fig. 4: Finite State Machine of the Padded process unit

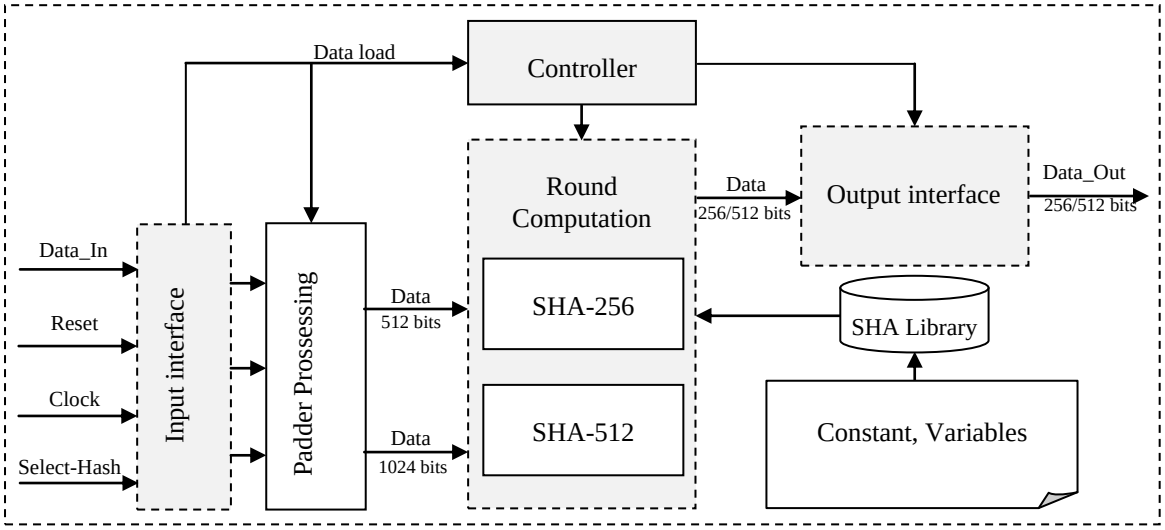

Fig. 3: Block Diagram of SHA-256 and SHA-512

- The Hash Computation Unit is the principal data path component of the system architecture. SHA-256 requires 64 cycles to produce the 256-bit message digests, however, the SHA-512 performs 80 cycles to produce 512-bit. Each cycle of SHA-512 algorithm requires the previous rounds, as well as the constant value Ki, the core utilize eight 64-bit words: a-d, wish are initialized to predefined values, at the start of each call to the hash function. The padder receives its input words via the Data-in, and the hash value can be read on port Data-out.
- Padded Process Unit pads the input data messages and converts them to 1024-bit realizes the message pre-processing, handling all message data to be hashed. A Finite state machine (FSM) is used for this function (See Fig. 4).

The FSM performs the five states

- Pad 0: in which data recovery takes place in the form of packets of 8 bits each.
- Pad 1 and Pad F: in the course of this transmission a 512 bits a packet counting takes place followed by a compatibility checking
- Pad 2: this state consists in finding out the number of bits '0' to be added.
- Pad 3: at this level the last packet is loaded up by the binary coding length of the message.

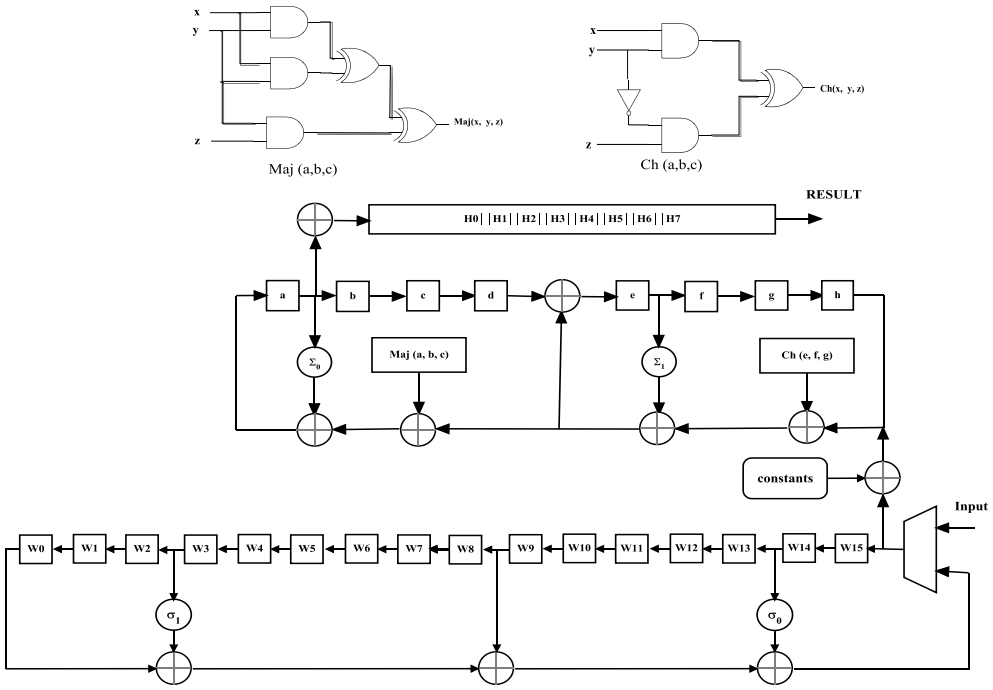

Fig. 5: Proposed compressor scheme

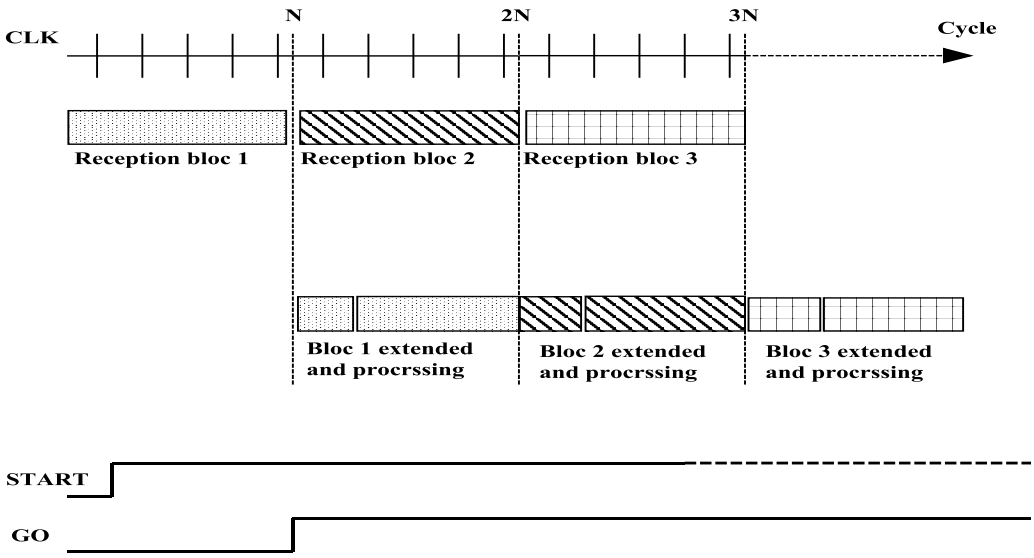

Fig. 6: Timing diagram for hashing process sequence

Several hardware-based SHA-512 designs have appeared in the literature [7-8]. The longest data path (critical path) in the SHA-512 core is the calculation of working variable A, which involves addition modulo2^{64} for SHA-512 of 7 operands (see listing 1). The following techniques have been proposed to speed up calculations in the SHA-512 core.

The constants memory provides the initialization values for the working variables (a,…, h). Initial hashes (H_0,…, H_7) are also set within this period of time. The initialization of working variables and initial hashes takes one clock cycle.

When initialization is complete, the compressor employs registers a,…, h, as well as Wj and Kjto determine the new values of a,…,h. As described in Step 3 of SHA-2, the algorithm takes j iterations and is controlled internally by an iteration counter. Precisely, SHA-512 utilizes 80 iterations. In each of these iterations, registers W_0,…, W_{15} and a,…,h are shifted in the direction of the arrows shown in Fig. 5. After j iterations, the intermediate hash computation is performed. It would be possible to execute this operation in a single clock cycle. In order to save implementation area, only two adders are utilized.

Fig. 6 shows the first message blocks of N words being stored into the core. The START signal is asserted at the start of each message. The SHA-2 processor is ready to accept data when START is asserted. Each Nb-bit word (Nb = 16) is clocked into the core on the rising edge of CLK when START is asserted. The START signal is used to acknowledge a data request from the core. The end of the message is indicated by a low-state of the START signal. After a feeding of a block of N words at the input, the signal GO is asserted as the SHA-2 core which computes the message digest. After, the next N/2 clock cycles (N = 64 for SHA-256, N = 80 for SHA-512), the message digest for the previous N word block is computed. Finally when the final message block has been processed, the hash value outputs are concatenated to produce the 256 or 512-bit message digest.

V. FPGA Implementation

The SHA-256 and SHA-512 have been described using VHDL and simulated by ModelSim 6.6 and synthesized, placed, and routed with Xilinx ISE 10.1.03. The FPGA target was XC5VFX70T from Xilinx Virtex-5 family. The architecture was simulated for verification of the correct functionality, by using the test vectors provided by the SHA-512 standard [1].

As seen in table 2, the number of occupied slices, the frequency (in megahertz), the throughput (in megabits per second) and the efficiency (in Gegabits per second/slices) for the SHA-256 and SHA-512 are presented.

The throughput is computed as:

$$Throughput = \frac{\#bit \times frequency}{\#clock\ cycles} \quad (15)$$

The Efficiency is obtained by using the following equation:

$$Efficiency = \frac{Throughput}{Area} \quad (16)$$

Table 2: FPGA implementation of the proposed design: Results

SHA Algorithm	Area (slice)	Frequency (MHz)	Throughput (Gbps)	Efficiency (Mbps/Slices)
SHA-256	387	202.54	1.58	4.19
SHA-512	874	176.06	2.20	2.58

The implementation of the SHA-256 takes 387 slices for 202.54 MHz frequency. The SHA-512 occupies 874 slices for 176.06 MHz Frequency. The number of slices for the SHA-512 is more than that of the SHA-256. This is mainly because of the constants, variables and logical operations are more complex than SHA-512.

Table 3 shows a comparison between our proposed design with some previous works for FPGA implementation.

Table 3: FPGA implementation of the proposed SHA-256: Comparison

SHA Algorithm	Area (slice)	Frequency (MHz)	Throughput (Gbps)	Efficiency (Mbps/Slices)
Proposed	387	202.54	1.58	4.19
[5]	424	179.5	1.4	3.38
[9]	-	80	0.61	-

Compared to [5] and [9], our proposed design has the minimum area and the highest frequency and throughput. In terms of hardware resources, the proposed design takes 387 slices for 202.54 MHz frequency while the SHA-256 in [5] occupied 424 slices with 179.5 MHz operating frequency. This means that the proposed SHA-256 is the most efficient.

VI. Conclusions

In this paper, we proposed a new design for the SHA-256 and SHA-512 functions. Moreover, the proposed design has been implemented on Xilinx Virtex-5 FPGA. Its area, frequency, throughput and Efficiency have been compared and it is shown that the proposed design outperform the previously reported ones.

References

[1] National Institute of Standards and Technology, "Secure Hash Standard", Federal Information Processing Standards 180-4, 2012.

[2] 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.

[3] I. Algredo-Badillo, C. Feregrino-Uribe, R. Cumplido , M. Morales-Sandoval, FPGA-based implementation alternatives for the inner loop of the Secure Hash Algorithm SHA-256, Microprocessors and Microsystems, 2012, Vol. 37, pp. 750-757.

[4] Kris Gaj, Ekawat Homsirikamol, Marcin Rogawski, Rabia Shahid, and Malik Umar Sharif, Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs, IACR Cryptology ePrint Archive 2012: 368 (2012).

[5] Yaser Jararweh, Lo'ai Tawalbeh, Hala Tawalbeh, Abidalrahman Moh'd "Hardware Performance Evaluation of SHA-3 Candidate Algorithms" Journal of Information Security, 2012, Vol(3), pp. 69-76.

[6] Kris Gaj, Ekawat Homsirikamol, Marcin Rogawaski "Fair and comprehensive Methodologiy for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidats using FPGAs" In Proceedings of Cryptographic Hardware and Embedded Systems workshop, CHES 2010, pp. 264-278.

[7] R.Chaves, G.Kuzmanov L. Sousa, S. Vassiliadis "Improving SHA-2 Hardware Implementations" Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006.

[8] McEvoy, R.P., Crowe, F.M., Murphy, C.C., Marnane, W.P "Optimisation of the SHA-2 family of hash functions on FPGAs" IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, pp. 317-322, 2006.

[9] 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.

[10] Fatma Kahri, Belgacem Bouallegue, Mohsen Machhout, Rached Tourki "An FPGA implementation of the SHA-3: The BLAKE hash function", In 10th International Multi-Conference on Systems, Signals & Devices (SSD), 2013.

[11] Fatma Kahri, Belgacem Bouallegue, Mohsen Machhout, Rached Tourki, "An FPGA implementation and comparison of the SHA-256 and Blake-256", In 14th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), 2013, pp. 152-157.

[12] Hassen Mestiri, Noura Benhadjyoussef, Mohsen Machhout and Rached Tourki, "A Comparative Study of Power Consumption Models for CPA Attack," International Journal of Computer Network and Information Security, Vol. 5, No. 3, pp. 25-31, 2013.

[13] Hassen Mestiri, Noura Benhadjyoussef, Mohsen Machhout and Rached Tourki, "A robust fault detection scheme for the advanced encryption standard", International Journal of Computer Network and Information Security, Vol. 5, No. 6, pp. 49-55, 2013.

[14] H. Mestiri, N. Benhadjyoussef, M. Machhout and R. Tourki, High performance and reliable fault detection scheme for the advanced encryption standard, International Review on Computers & Software (IRECOS), Vol 8, No 3, pp. 730–746.

[15] J Philippe Aumasson, L Henzen W. Meier, SHA-3 proposal BLAKE varsion 1.3, decembre 16, 2010.

Authors' information

Hassen Mestiri received his M.S. degree in Microelectronic Systems from the Faculty of Sciences of Monastir, Tunisia, in 2011. Currently, he is a PhD student. His research interests include implementation of standard cryptography algorithm, security of embedded system and Hardware/Software Codesig.

Fatma kahri received here M.S. degree in Microelectronic Systems from the Faculty of Sciences of Monastir, Tunisia, in 2012. She is a PhD student. Her research interests include implementation of standard hash algorithm and security of embedded system on FPGA.

Belgacem Bouallegue received his MSc in Physic Microelectronic, his DEA in Electronic Materials and Dispositifs and and the Doctorat de 3ème cycle in Electronics from the Science Faculty of Monastir, Tunisia, in 1998, 2000

and 2005, respectively. Currently, he is a Hdr student. His research interests include High Speed Networks, Multimedia Application, Network on Chip: NoC, flow and congestion control, interoperability, Security Networks implementation of standard cryptography algorithm, key stream generator and electronic signature on FPGA and performance evaluation. He is working in collaboration with Lab-STICC à Lorient Laboratory, Lorient Cedex France and LIP6, Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie, UPMC - CNRS UA 7606, Département SoC, Systèmes Embarqués sur Puce, 4, place Jussieu; 75252 PARIS Cedex 05, France.

Mohsen Machhout was born in Jerba, on January 31 1966. He received MS and PhD degrees in electrical engineering from University of Tunis II, Tunisia, in 1994 and 2000 respectively. Dr Machhout is currently Associate Professor at University of Monastir, Tunisia. His research interests include implementation of standard cryptography algorithm, key stream generator and electronic signature on FPGA.

