

# Architectural design features of a programmable high throughput reconfigurable SHA-2 Processor

Medien Zeghid<sup>1,2</sup>, Belgacem Bouallegue<sup>1,2</sup>, Mohsen Machhout<sup>1</sup>, Adel baganne<sup>2</sup> and Rached Tourki<sup>1</sup>

<sup>1</sup> *Electronics and Micro-Electronics Laboratory, Faculty of Sciences of Monastir, Monastir 5000, Tunisia, medienzeghid@gmail.com*

<sup>2</sup> *Laboratoire d'Electronique des Systèmes TEmps Réel (LESTER), University of South Brittany, BP 92116 – 56321, Lorient, France, adel.baganne@univ-ubs.fr*

**Abstract:** The continued growth of both wired and wireless communications has triggered the revolution for the generation of new cryptographic algorithms. Hash functions are common and important cryptographic primitives, which are very critical for data integrity assurance and data origin authentication security services. Many hash algorithms have been investigated and developed in the last years. This work is related to hash functions FPGA implementation. SHA-2 hash family is a new standard in the widely used hash functions category. An architecture and the FPGA implementation of this standard are proposed in this work. We propose a reconfigurable SHA-2 processor in the sense that it performs the two hash functions (256, and 512) of the SHA-2 standard for extended signature authentication. This paper investigates optimizations techniques that have recently been proposed in the literature. These techniques consist mostly in operation rescheduling and hardware reutilization, allowing a significant reduction of the critical path while the required area also decreases. As several 64-bit adders are needed in SHA-512 hash value computation, investigations on the adders implementations on FPGAs are analyzed and developed, with a view to reduce the chip area. The proposed processor is compared with the implementation of each hash function in a separate FPGA device to demonstrate that our architecture achieves a performance comparable to separate implementations while requiring much less hardware. Two types of FPGAs are evaluated in order to produce realistic results. Speed/area results from this processor are analyzed and are shown to demonstrate that our designs achieves a favorably performances with other FPGA-based implementations. A fastest data throughput is achieved by our optimized design.

**Keywords:** Hash Function Standard, Signature, SHA-2 standard, FPGA, Reconfigurable Processor.

## 1. Introduction

Cryptography serves a great number of scopes and ensures different types of security due to alternative encryption schemes. Among them we can mention for instance, the bulk encryption, the message authentication and the data integrity. The symmetric ciphers, the asymmetric encryption algorithms and hash functions support each one of the above types respectively [1]. Hash functions operate at the root of many popular cryptographic methods in current use, such as the Digital Signature Standard (DSS), Transport Layer Security (TLS) and Internet Protocol Security (IPSec) protocols. They are also a basic building block of secret-key Message Authentication Codes (MACs), including the

American federal standard HMAC [2]. Other popular applications of hash functions include numerous random number generation, fast encryption, all-or-nothing transforms, and password storage and verification [3]–[4]. They are widely spread and many wireless protocols, such as WAP [5] and Hiperlan [6] have specified security layers and cryptographic schemes based on them.

The purpose of a hash function is to produce a “fingerprint” of a file, message, or other block of data. A hash value  $h$  is generated by a function  $H$  of the form  $h=H(M)$ , where  $M$  is a variable-length message and  $H(M)$  is the fixed-length hash value. In the cryptographic hash function, a message of arbitrary length padded and broken into blocks is fed sequentially to a compression function which converts a fixed-length input (current message block) to a fixed-length output (hash value). The hash values of individual blocks are used iteratively by the compression function to find the final hash value, referred to as message digest. A hash function provides a unique relationship between the input message and the hash value and hence, represents a longer message in a concise way. The current American Federal Information Processing Standard, FIPS 180-2, recommends the use of one of the four hash functions developed by National Security Agency (NSA) and approved by NIST (National Institute of Standard and Technology). By far the most widely used of these four functions is SHA-1 (Secure Hash Algorithm-1), a revised version of the standard algorithm introduced in 1993 [7]. The best attack against this algorithm is in the range of 280 operations, which makes its security equivalent to the security of Skipjack and the Digital Signature Standard (DSS). After introducing a new secret-key encryption standard, AES (Advanced Encryption Standard), with three key sizes, 128, 192, and 256 bits, the security of SHA-1 did not any longer match the security guaranteed by the encryption standard [8]. Therefore, an effort was initiated by NSA to develop three new hash functions, with the security equivalent to the security of AES with 128, 192, and 256 bit key respectively. This effort resulted in the development and standardization of three new hash functions referred to as SHA-256, SHA-384, and SHA-512 [9]. Since then, SHA-224 has been added to the standard, forming the ‘SHA-2’ family of hash functions. All four standardized algorithms

have a similar internal structure and operation. All of them are based on sequential processing of consecutive blocks of data, and therefore cannot be easily speed up by using pipelining or parallel processing (at least when only one stream of data is being processed).

In other hand, in our days, reconfigurable computing is a very attractive method for the hardware implementation of systems/algorithms [10]–[11]–[12]. Reconfigurable systems can change their “true” hardware configuration and can support multi-operations modes.

In this paper we presented ultra high speed architecture for the integration of SHA-256 and SHA-512 hash functions. The proposed system is reconfigurable in the sense that performs efficiently for both hash functions upon the user needs, it performs the two SHA-2 hash functions (256, 512). The architecture design is based on a pipelined methodology of 2-stages. This paper deals with three issues, namely, proposing architecture for reconfigurable implementation of a hash function on FPGA, optimizing the architecture and comparing the performance metrics of different FPGA, that implement a SHA-2 function and single chip implementation of SHA-2 family hash functions [13].

The remaining paper is organized in the following manner: in section 2 both SHA-256 and SHA-512 hash functions are described briefly. In the next section, the proposed system presented and the internal components of this architecture are described in detail. The synthesis results of the FPGA implementation are given in the next section. Comparisons with other related works are also presented in section 5. Finally, conclusions and observations are discussed in section 6.

## 2. Secure Hash Standard 2 (SHA-2) Functional Comparison

In 1993 the Secure Hash Standard (SHA) was first published by the NIST. In 1995 this algorithm was reviewed in order to eliminate some of the initial weakness and in 2001 new Hashing algorithms were proposed. This new family of hashing algorithms known as SHA-2, use larger digest messages, making them more resistant to possible attacks and allowing them to be used with larger blocks of data, up to 2128 bits, e.g. in the case of SHA-512. The SHA-2 hashing algorithm is the same for the SHA-256, SHA-224, SHA-384 and SHA-512 hashing functions, differing only in the size of the operands, the initialization vectors, and the size of the final digest message.

All descriptions of SHA-256, SHA-384 and SHA-512 algorithms can be found in the official NIST standard [9]. Table 1 shows a comparative study in terms of function characteristics of four hash functions.

The security of these hash functions is controlled by the size of their outputs, referred to as hash values,  $n$ . The definition of SHA-384 is almost identical to SHA-512, with the exception of a different choice of the initialization vector and a truncation of the final 512-bit result to 384 bits. All functions have a very similar internal structure and process each message block using multiple rounds. The number of rounds for SHA-384 and SHA-512 is the same and 20% smaller in SHA-256. 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 four hash functions

Hash functions	SHA1	SHA-2		
		256	384	512
Size of hash value ( $n$ )	160	256	384	512
Complexity of the best attack	$2^{80}$	$2^{128}$	$2^{192}$	$2^{256}$
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Message block size ( $m$ )	512	512	1024	1024
Word size	32	32	64	64
Numbers of words	5	8	8	8
Digest rounds number	80	64	80	80
Constants $K_t$ number	4	64	80	80

Each hash function operation can be divided in two stages: (1) pre-processing and (2) hash computation. Pre-processing involves padding the input message, parsing the padded data into a number of  $m$ -bit blocks ( $m = 512$  or  $1024$  bit) and setting the appropriate initial values, which are used in the hash computation. The hash computation uses functions applied to the padded data, constants and word logical and algebraic operations, to generate iteratively a series of hash values. After a specified number of transformation rounds the produced hash value is equal to the message digest. These latter ranges in length from 256 to 512 bits, depending each time on the selected hash function. The following describes the SHA-2 algorithm applied to the SHA-256 hash function, followed by the description of the SHA-512 hash function, which differs mostly in the size of the operands, using 64-bit words instead of 32-bit.

### 2.1 SHA-256 hash functions

SHA-256 calculates a 256-bit digest for an arbitrary  $b$ -bit message and it consists of the following steps.

- Pre-Processing:

The  $b$ -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  $b$  is appended to the result of the padding. Thus, the resulted message is a multiple of 512 bits. This message is denoted here as  $M(i)$ .  $M(i)$  message blocks are passed individually to the message expander. Which are used to generate the message schedule  $W_t$ 's.

- Message Expansion:

The functions in the SHA-256 algorithm operate on 32-bit words, so each 512-bit  $M(i)$  block from the Pre-Processing stage is viewed as 16 32-bit blocks denoted  $M_t(i)$ ,  $0 \leq t \leq 15$ . The message scheduler takes each  $M(i)$  and expands it into 64 32-bit  $W_t$  blocks, according to the equations:



### 3.1 Processor design

Since the SHA-256 and SHA-512 algorithms are very similar, they can be implemented on a single chip easily. A block based top-level implementation of our proposed processor is shown in figure 2. The proposed system supports four operation modes for reconfigurable SHA-2 processor.

- The Control Unit is designed to control the flow of data in the design, as well as the movement of data between the Padded Process Unit and Hash Computation Unit. A FSM is used for this purpose. During initialization phase, the user with the appropriate write commands selects the operation mode. The Control Unit coordinates all the system operations and processes. After the initialization phase, the control unit is totally responsible for the system operation. It defines the proper constants and operation word length, it manages the ROM blocks and it controls all the proper algebraic and digital logic functions for the operation of SHA-2 (256 and 512) hash functions.
- Padded Process Unit pads the input data messages and converts them to 512 or 1024 bit blocks (padded data).

This operation is characterized by simplicity and it is well defined by SHA specifications [9]. First, the Padded pads the input message and after that the hash function begins. The Padded Data is a multiple of 512-bit block for the SHA-256 and a multiple of 1024-bit block for the SHA-512. In every data transformation round, based on the padded data, in the Hash Computation Unit a new data block,  $W_t^{(i)}$ , is produced. In the ROM Blocks the specified constants set,  $K_t^{(i)}$ , of the SHA-2 standard are stored, in order to support the Hash Computation Unit process.

- The hash value generation is mainly centred in the Hash Computation Unit for both operations (SHA-256 and SHA-512). The Hash Computation Unit is the main data path component of the system architecture. The specified number of the data transformation rounds, for each one of the SHA-2 hash family functions, is performed in this component with the support of a rolling loop (feedback). In last transformation the message digest is produced, stored and beginning of the Bus Interface.
- A Bus Interface Unit has also been integrated in order for the proposed processor to communicate efficiently with the external environment.

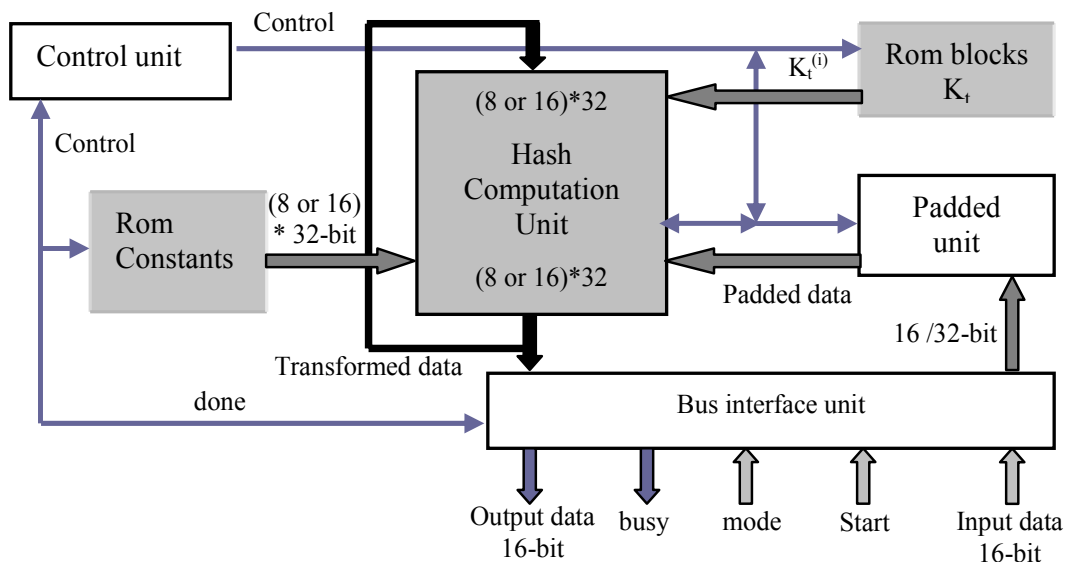


Figure 2. Integrated chip for SHA-256 and SHA-512

In the following, the system architecture's basic units are described.

### 3.2 Processor Implementation

Since the SHA processors are designed for implementation in a reconfigurable platform, these allowed one combination to be investigated with relative low complexity once unrolled (' $2x$ ') core is implemented [14]–[15]. Specifically, the goal is to permit next round's available inputs to calculate instantaneously an intermediate result. The aim is to obtain a resource efficient implementation, increasing the achievable data throughput.

#### 3.2.1 Padded Unit

The input message is padded, before the beginning of the Hash Computation Unit. So, the Padded ensures that the input message length is a multiple of 512-bit block in the

case of SHA-256, and multiple of 1024-bit block in the cases of SHA-512. Most message padding designs in the literature are proposed. However, here we aim to produce fast real time implementations of the hashing processor; therefore, a Padded block was also included to implement the padding described in Section 2. This was realised as a synchronous finite state machine (FSM). In our architecture, message blocks are not stored in Padded Unit. The Padded Unit passes input data to the hash Computation unit a word ( $N_b$ -bit) at a time, during the first  $N'$  clock cycles used to process each message block ( $N' = 32$  clock cycles for SHA-256 or 64 for SHA-512). In the Hash Computation Unit the Padded data are processed in order.

Figure 3 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  $N_b$ -bit word ( $N_b$

= 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. A signal mode selecting a counter, which counts is to  $N/2$ , is used to address the ROM and to select between the 512-bit and the shortened 256-bit message digests.

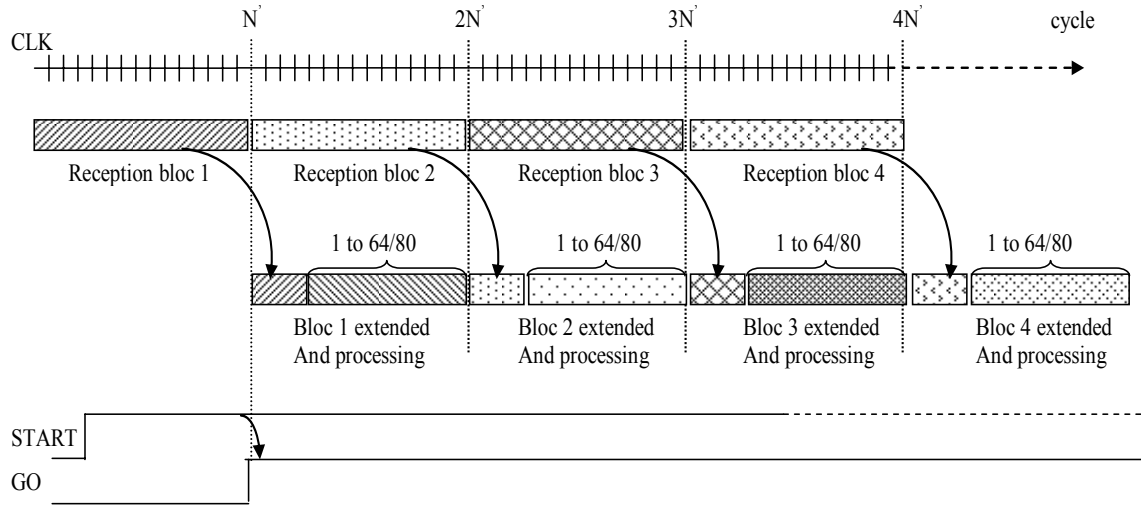


Figure 3. Timing diagram for hashing process sequence

3.2.2 ROM and Constants Units

For our proposed architecture, a memory elements configured as ROM are used to store the constants and the numbers of shifts (Figure 4).

$$\begin{aligned} \text{SHA-512 } K_0 &= \text{d728ae22428a2f98} \\ \text{SHA-512 } H_0^0 &= \text{f3bbc9086a09e667} \end{aligned} \rightarrow \text{SHA-256 } K_0, H_0^0$$

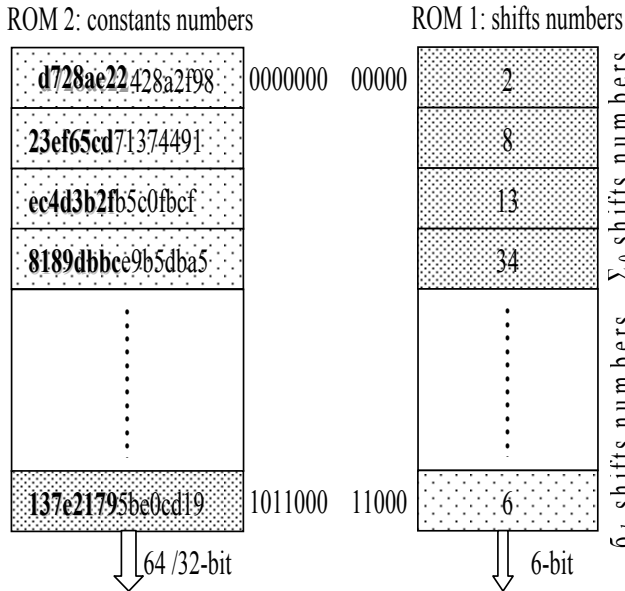


Figure 4. ROM Blocks

For the shift-number lookup, a ROM with 5-bit address must be used and 6-bit output used for the SHA-2 processor, since there are 24 steps in SHA-2. For constants lookup, a ROM with 7-bit input address and 64 or 32-bit output is used for the SHA algorithm as it specifies. The definition of the rounds constants are the same for equal word sizes and the value of  $K_t$  for  $W=32$  is identical to the right half of the corresponding  $W=64$  constant. Similarly, the right halves of the SHA-512 words of  $H_0$  are the words of  $H_0$  for SHA-256. For example:

This arrangement allows simple decoding logic to select the appropriate constants for the both algorithms. As LUTs are used in the aforementioned manner, a mode signal is used to index the appropriate values during each step. It simplifies the control logic design, resulting a compact implementation.

3.2.3 Hash Computation Unit

Synthesis results show that the design performance like critical path, area runs from the Hash Computation Unit. The data modification in the Hash Computation Unit are mainly performed by Ch, Maj,  $\Sigma_0$ ,  $\Sigma_1$  hash functions and the modulo adders. Especially, in the modulo adder components, modulo addition 232 is performed for the SHA-256, and modulo addition 264 for the SHA-512. Hence, reducing the size of these adders will reduce the number of logic elements required in the FPGA, thereby, reducing the overall area of the final circuits. We can trade-off a reduction in critical path and hence increase the throughput rate, as we discuss next.

• Adder Choice

The implementation of multi-operand 64-bit adders on FPGA demands selection of proper scheme for performing the addition, as both the speed and area are of concern. Several schemes consist for the implementation of multi-operand addition. In this work, four types of adders are implemented, optimized and tested, as known ripple carry (RCA), carry lookahead (CLA), carry save (CSA) and carry select adders (CSelA). In a CSA or CLA an array of full adders (FA) are used to perform addition of three binary

vectors without propagating the carries and two library vectors, pseudo-sum and carry are generated. Let  $a, b, c$  three  $n$ -bit numbers, the FA produces a partial sum PS and a shift-carry SC:

$$PS_i = a_i \oplus b_i \oplus c_i, SC_i = (a_i \wedge b_i) \text{ or } (a_i \wedge c_i) \text{ or } (b_i \wedge c_i).$$

The different proposed adders architectures were captured by using VHDL, with structural description logic. All main adder architectures were compared for word lengths of 8, 16, 32, 64 bits with carry input and output. The synthesis results for the proposed adders implementations are illustrated in Table 2. Two performance metrics are used: the area (slices) and delay (ns).

Table 2. Synthesis results of Adders ((a): Area requirements; (b): delay requirements)

bits	CSA	RCA	CselA	CLA
8	8	10	14	9
16	17	20	29	18
32	36	38	63	37
64	72	75	129	74

(a) Area (Slices) requirements

bits	CSA	RCA	CselA	CLA
8	5.80	13.34	10.06	12.37
16	5.95	21.14	14.41	20.17
32	6.03	36.74	22.10	35.77
64	6.15	67.94	37.295	66.97

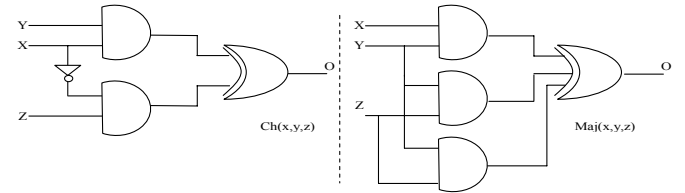
(b) delay (ns) requirements

It is apparent in table 2 that the CSA has almost the same allocated resources with the CLA adder. Furthermore, the carry save adder present the speed-delay for every word length. It is faster by at least two gate delays than all other adders with  $\log(n)$  time complexity because it works without the final sum-bit generation level built from XORs. So, our processor use the carry save representation of numbers to speed-up the multi operand addition and minimize the delay associated with carry propagation.

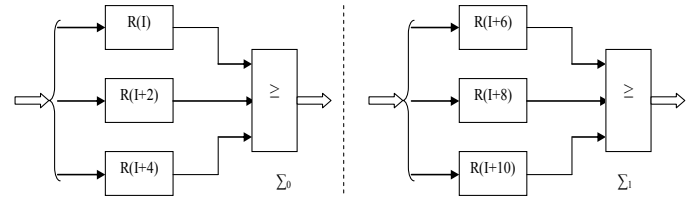
### • SHA-2 functions

As mentioned above, there are four nonlinear functions. The architectures of these functions are illustrated in Figure 5. The Ch and Maj (Figure 5.a) are common functions for SHA-256 and SHA-512 however  $\Sigma_0$  and  $\Sigma_1$  depend of the hash functions. The proposed architectures for the  $\Sigma_0$  and  $\Sigma_1$  functions are shown in Figure 5.b. Figure 5.c illustrates the  $\sigma_0$  and  $\sigma_1$  functions architectures for the generation of the  $W_t(i)$  data block.

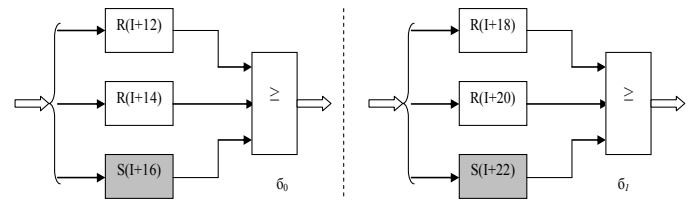
The functions of Figure 5 are similar but their implementations are done using different constants for each one of the SHA-2 family hash functions. The specified constants values for every hash function are given by the SHA-2 standard. Furthermore,  $R(I)$  indicates right rotation of the input data by  $I$ -bit, while the  $S(I)$  components indicates right shift of the input data by  $I$ -bits.



(a) : Ch and Maj functions architectures



(b) :  $\Sigma_0$  and  $\Sigma_1$  functions architectures  
( $I = 0$  for SHA-256,  $I = 1$  for SHA-512)



(c) :  $\sigma_0$  and  $\sigma_1$  functions architectures  
( $I = 0$  for SHA-256,  $I = 1$  for SHA-512)

Figure 5. SHA-2 functions architectures

### • Hash Computation design

In SHA-2, eight words ( $A, B, C, D, E, F, G$  and  $H$ ) remain almost unchanged by a single round. These words are only shifted by one position down. The words,  $A$  and  $E$ , undergo a complicated transformation equivalent to multi operand addition modulo 2word. These operands depend on which are the input words, the round-dependent constant  $K_t$  (which are stored in ROM to conserve space on hash function) and a message dependent word  $W_t$ . It should be noticed that in each round the computation is only required to calculate the values of  $A$  and  $E$ , since the remaining values are obtained directly from the values of the previous round, as depicted in section 2. So, the variables  $B, C, D, F, G$  and  $H$  are obtained directly from the values of the round, not requiring any computation, the values of  $A$  and  $E$  require computation and depend on all the values. In other words, the values  $A$  and  $E$  for round  $t$  can not be computed until the values for the same variables have been computed in the previous round have, as shown in (13).

$$\begin{cases} E_{t+1} = W_{t+1} + K_{t+1} + H_t + \Sigma_1(E_t) + Ch(E_t + F_t + G_t) + D_t \\ A_{t+1} = W_{t+1} + K_{t+1} + H_t + \Sigma_1(E_t) + Ch(E_t + F_t + G_t) + \Sigma_0(A_t) \\ \quad + Maj(A_t + B_t + C_t) \end{cases} \quad (13)$$

The proposed design approach is to condense two cycles ( $t+1, t+2$ ) in one and to reduce the numbers of adder's. Specifically, the goal is to permit next round's available inputs to calculate instantaneously an intermediate result. This process continues for  $N/2$  clock cycles ( $N = 80$  or  $64$ ). In both functions, the input registers are initialized with the constant initialization vector, and are updated with the new value in each round.  $2x$ -unrolled core requires two  $W_t$  words to be available simultaneously; both units generate  $N$

message dependent words,  $W_t, t = 0...(N-1)$ . The first 16 of these words,  $W_0.....W_{15}$ , are simply the first 16 words of the input message block. The remaining words are computed using a simple feedback function, shifts and XOR operations. After  $N/2$  cycles the values in registers A to H are added to the initial hash values to obtain new hash values. The expression to calculate outputs each cycle's, as shown in (14).

$$\begin{cases}
 C_{t+2} = A_t \\
 D_{t+2} = B_t \\
 G_{t+2} = E_t \\
 H_{t+2} = F_t \\
 Tmp = W_t + K_t + H_t + \Sigma_t(E_t) + Ch(E_t + F_t + G_t) \\
 F_{t+2} = Tmp + D_t \\
 A_{t+2} = Tmp + \Sigma_0(A_t) + Maj(A_t + B_t + C_t) \\
 B_{t+2} = A_{t+1} \\
 Tmp = W_{t+1} + K_{t+1} + G_t + \Sigma_1(F_{t+2}) + Ch(F_{t+2} + E_t + F_t) \\
 E_{t+2} = Tmp + C_t \\
 A_{t+2} = Tmp + \Sigma_0(A_{t+1}) + Maj(A_{t+1} + B_t + A_t)
 \end{cases} \quad (14)$$

The first observation, that input  $A_t$  and  $B_t$  feed directly outputs  $C_{t+2}$  and  $D_{t+2}$  respectively. Similarly,  $E_t$  and  $F_t$  feed directly outputs  $G_{t+2}$  and  $H_{t+2}$  respectively. In other hand, something that can be easily observed from the latter equation is that the most problematic characteristic of the design process is the need to add (modulo  $2w$ ) several numbers. If this addition were to be implemented in a straightforward manner, 14 adders would be required, one for each internal variable, of 32 or 64 bits depending if SHA-256 or SHA-512 is being implemented. However, some hardware reuse can be achieved.

In order to reduce the area design and optimize the critical path, we have implemented and tested two methods (see figure 6, table 3, figure 7 and table 4).

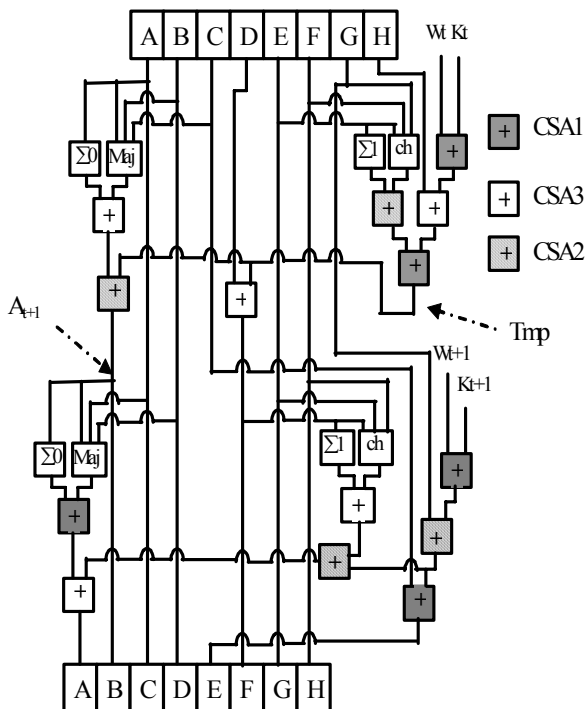


Figure 6. Internal structure of unrolled Hash Computation Unit-3 adders

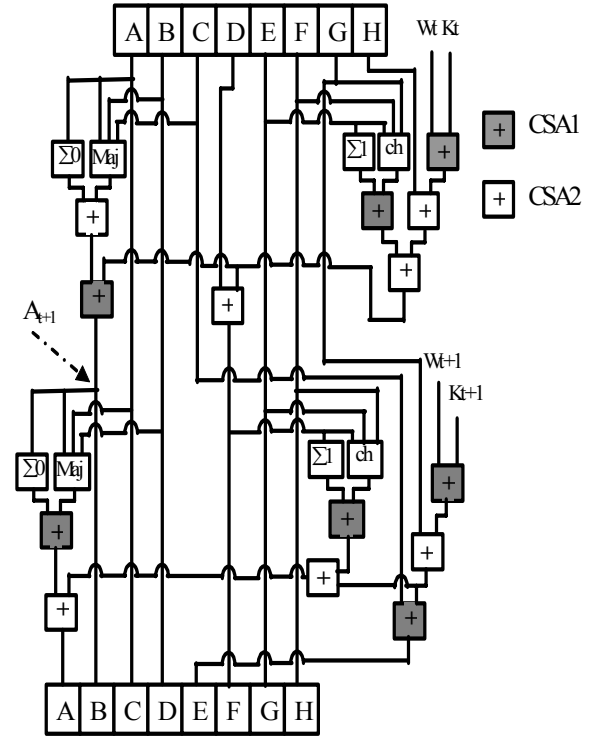


Figure 7. Internal structure of unrolled Hash Computation Unit-2 adders

Table 3. Methodology functions of three adders (while Ryx; Result of CSAy; step x)

	CSA1		CSA2		CSA3	
	Etat	Function	Etat	Function	Etat	Function
Step1	On	$W_t + K_t$	On	$Ch0 + S10$	On	$S00 + Maj0$
Step2	On	$W_{t+1} + K_{t+1}$	Off		On	$H + R11$
Step3	On	$R21 + R32$	On	$R12 + G$	Off	
Step4	On	$C + R23$	On	$R31 + R13$	On	$R13 + D$
Step5	On	$Maj1 + S01$	Off		On	$Ch1 + S11$
Step6	Off		On	$R35 + R23$	Off	
Step7	Off		Off		On	$R15 + R26$

Table 4. Methodology functions of two adders (while Ryx; Result of CSAy in step x)

	CSA1		CSA2	
	Etat	Function	Etat	Function
Step1	On	$W_t + K_t$	On	$S00 + Maj0$
Step2	On	$Ch0 + S10$	On	$H + R11$
Step3	On	$W_{t+1} + k_{t+1}$	On	$R12 + R22$
Step4	On	$R21 + R23$	On	$D + R23$
Step5	On	$Ch1 + S11$	On	$R13 + G$
Step6	On	$Maj1 + S01$	On	$R15 + R25$
Step7	On	$R16 + R26$	On	$R25 + C$

We assume the scheme of Figure 7 as a starting point for our discussion. As we will see, all the proposed optimizations will be applied to this canonical form of the circuit; this provides a good common ground to compare

implementation techniques and refinements. All additions operations lead to the figure 7 of the unrolled architecture of hash computation unit shown in table 4. At each step two additions are performed. We have 7 steps in totality. Furthermore, in the second method we have applied three adders. We notice, just in step 1 and step 4 we need three adders. However, in steps 2, 6 and 7 there are at least an adder that is not used. It apparent from table 3 and table 4, that method 1 and method 2 have the same performances in terms of rapidity, in terms of cost, there is a gain of an adder for the second method.

### 3.3 Proposed Optimization

Several optimizations have been proposed to improve the implementation of the SHA-2 algorithm. The proposed optimization is based on modifying the basic hardware structure of SHA processor with the following modifications and additions.

- Unrolling techniques that optimize the data dependency. An unrolled architecture implements multiple rounds of the core compression function in combinational logic, thereby reducing the number of clock cycles required to compute the hash. This technique allows for an improvement in the throughput. Our approach is to condense two round (t+1, t+2) in one.
- Using Carry-Save Addition (CSA). CSA accept 3 input operands, hence the designs in this paper use just 2 CSA to calculate all the additions operations.
- One look-up table (LUT) is used to store the constants and number of shifts.
- The SHA processor takes, Nb bits data as inputs (Nb = 16). The input data are not stored.
- A 1-stage shift register design approach is employed to implement the Padded Unit. The register is loaded with the Nb-bit input message per clock cycle.
- There are four different nonlinear functions: Every one is used for each hashing data process, as know SHA-256 and SHA-512. Instead, we have decided to use a common architecture of these functions, which reduce the area. For the Left and the Right shifted constants, it has to be mentioned with mode selected signal.

## 4. Experimental Results

We implemented an FPGA design that efficiently performs both SHA-256 and SHA-512. Figure 2 shows the top level schematic of the integrated chip. It can function as both an SHA-256 unit or as a SHA-512 unit depending up on the control signals start and mode. When mode = '0', it signals the data path unit that SHA-256 needs to be performed. Similarly, if mode = '1', SHA-512 will be performed. The described circuits have been implemented in VHDL using the Model Technology's ModelSim Simulator and synthesized, placed, and routed using target device of Xilinx (Xilinx Virtex XCV200pq240 FPGA). The architecture was simulated for verification of the correct functionality, by using the test vectors provided by the SHA-2 standard [9]. In order to have a fair and detailed evaluation, we implemented SHA-256 and SHA-512 separately. The proposed reconfigurable processor is compared with each

one the hash functions individual implementation. Five performance metrics such as cycles, clocking frequency (Mhz), the throughput (Mbps), the area (slices) and the power consumption were computed. The throughput (d) is computed as,

$$d = \text{message block size}/(\text{clock period} * \text{latency}).$$

The block size for SHA-256 is 512 bits while the block size for SHA-512 is 1024 bits. The designs were simulated for a block of 512 and 1024 bits of padded message for SHA-256 and SHA-512 respectively. Table 5 presents detailed results for these implementations.

Table 5. FPGA Synthesis results

Our Design	Cycles	Freq (Mhz)	Throughput (Mbps)	Area slices	Power (mW)
SHA-256	32	56	896	1480	39
SHA-512	42	53	1292	2385	43
SHA-2 processor	32 64	53	848	2530	47

It is apparent in table 5 that the reconfigurable processor has almost the same allocated resources with the separate SHA-512 implementation. The processor engine was able to operate at 53 Mhz, is the same as the frequency of SHA-512 implementation and about 3% less than the 56 MHz of the SHA-256 implementation.

The results show that unrolling the quasi-pipelined SHA-2 processor design provides data throughput advantage. Our circuit has a 2x-unrolled core. It is clear that the critical path inside the core of SHA-2 processor increases with the degree of unrolling. Although the unrolled designs process messages in fewer clock cycles than the basic designs, the longer critical path in the unrolled designs means that the maximum clock frequency decreases. Going from a basic quasi-pipelined SHA-512 or SHA-256 design to the 2x-unrolled designs reduces the number of clock cycles by a factor of 2. We effectively pushed the pipelining approach to the limit, in the sense that it is not possible to create more pipeline sections and increase the total amount of clock cycles only by a small further factor. This can be understood by considering the presence of the Maj and Ch functions, accessing simultaneously, the values stored in three different positions in the corresponding shift registers. Their output value is immediately inserted back into the shift register.

Furthermore, during self-test at 50 mhz, the reconfigurable processor consumed 47 mW. The separate implementations, SHA-256 and SHA-512, have estimated power dissipation 39, and 43 mW, respectively. In addition, SHA-2 processor requires only 32 clock cycles in the SHA-256 operation mode. In the case of SHA-512 operation mode, 64 clock cycles are required (because 64 cycles used to process 1024-bit for SHA-512).

## 5. Performance Comparison

### 5.1 Evaluation metrics

When evaluating a given implementation, the throughput of

the implementation and the hardware resources required to achieve this throughput are usually considered the most critical parameters. No established metric exists to measure the hardware resource costs associated with the measured throughput of an FPGA implementation. Two area measurements are readily apparent—logic gates and configurable logic blocks (CLBs) slices. It is important to note that the logic gate count does not yield a true measure of how much of the FPGA is actually being used. Hardware resources within CLB slices may not be fully utilized by the place-and-route software so as to relieve routing congestion. This results in an increase in the number of CLB slices without a corresponding increase in logic gates.

To achieve more accurate measure of chip utilization, CLB slice count as chosen as the most reliable area measurement. Therefore, to measure the hardware resource cost associated with an implementation's resultant throughput; the throughput per slice (TPS) metric is used. We defined it as

$$\text{TPS} = (\text{throughput rate} / \# \text{CLBslices used}).$$

### 5.2 Device used

We synthesised our design using two targets: theVirtex v200pq240 which is the most suitable for our architectures, the Virtex2 xc2v2000 which we used for providing accurate comparisons with existing schemes.

### 5.3 Comparisons with published implementations

Table 6 compare our implementation with several others very recently reported in the literature in terms of SHA-256 only, SHA-512 only, both SHA-256 and SHA-512 [14]–[15]–[16]–[17]–[18]. Furthermore comparisons with hash function standard (SHA-1) implementations [19]–[20]–[21] are also given. The performance is compared in terms of frequency, the area, the throughput, and the TPS.

The introduced system in [16] supports the three hash functions SHA-256, SHA-384 and SHA-512. The focus of [17] was to implement SHA-2 using the theVirtex v200pq240 as a target. Ref [15] presents a pipelined architecture for a single chip SHA-384/SHA-512. The SHA-2 implementation of [14], presented the highest throughput, where SHA-2 was implemented using the re-use and pipeline techniques simultaneously. Although, in [15]–[16]–[17] the SHA-256 and the SHA-512 have been implemented separately.

Using the TPS metric the proposed design is proved better about 24% by compared with [16], and better about 20 % by compared with [17]. When compared with [14] the results show higher throughput, from 17 % up to 26 %, while achieving a reduction in area above 25 % and up to 42 %. Figure 8 and Figure 9 detail the optimal implementation in terms of TPS for each SHA-2 implementations (SHA-256, SHA-512 and SHA-2 integrated chip).

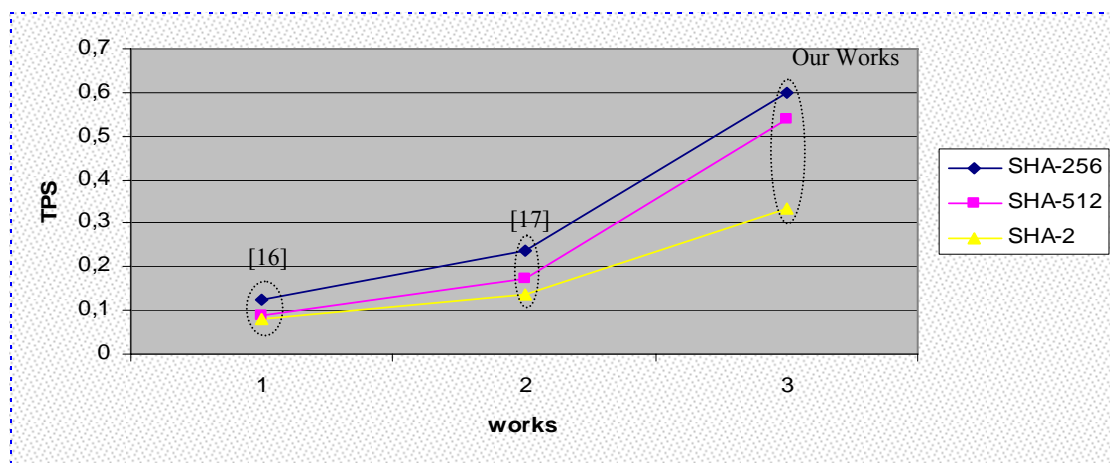


Figure 8. Representing TPS- Virtex v200pq240

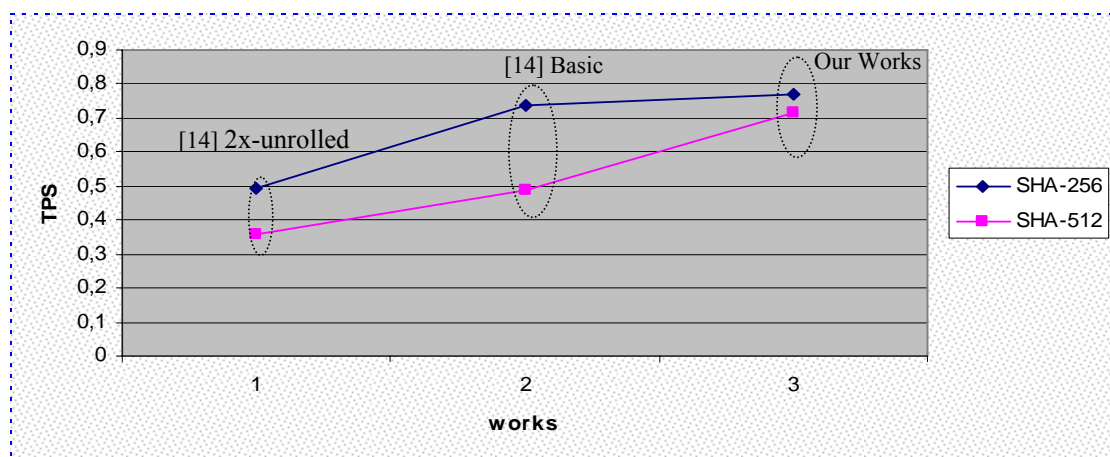


Figure 9. Representing TPS- Virtex xc2v2000-bf957

Also, the proposed system is proved to be better compared with the previous SHA-1 standard hardware implementations [19]–[20]–[21]. For instance, we cannot go on a detailed “fair” comparison with the previous standard, since these two standards (SHA-1 and SHA-2)

have major differences in their specifications. From the synthesis results of Table 6, it is also proven that the proposed implementation performs better in terms of operating frequency and throughput compared with the hardware design work of [9].

Table 6. Performance comparison results

Reference	Frequency (MHz)	Area	Throughput (Mbps)	TPS (Mbps/slice)
SHA-1 architectures				
[19] Virtex 2v500fg45	55	4490 CLBs	1339	0,298
[20] Virtex 2v500fg45	38	3100 CLBs	900	0,290
[21] Virtex v150bg352	55	NA	2816	NA
SHA-2 architectures				
[18] APEX II, Stratix,	41,97	3383 LEs	335, (256) 268,9 (512)	NA
[16] Virtex v200pq240	83	2120 slices	262,(SHA-256)	0,123
[16] Virtex v200pq240	75	4474 slices	396,( SHA-512)	0,089
[16] Virtex v200pq240	74	4768 slices	233,(SHA-256) 390,(SHA-512)	0,049/0,082
[17] Virtex v200pq240	77	1306 slices	308,( SHA-256)	0,236
[17] Virtex v200pq240	69	2545 slices	442,( SHA-512)	0,174
[17] Virtex v200pq240	69	2951 slices	200,(SHA-256) 320,(SHA-512)	0,108/0,136
[15] VirtexE xcv600E8	38	5828 slices	479,(SHA-384) 479,(SHA-512)	0,082
[14] (basic) xc2v2000-bf957	133	1373	1009,(SHA-256)	0,735
[14] (2x-unrolled) xc2v2000-bf957	73,975	2032	996,(SHA-256)	0,491
[14] (basic) xc2v2000-bf957	109,03	2726	1329,(SHA-512)	0,488
[14] (2x-unrolled) xc2v2000-bf957	65,893	4107	1466,(SHA-512)	0,357
Proposed architectures				
Virtex v200pq240	56	1480 slices	896,(SHA-256)	0,60
Virtex v200pq240	53	2385 slices	1292,(SHA-512)	0,541
Virtex v200pq240	53	2530 slices	848,(SHA-256) 848,(SHA-512)	0,335
Virtex xc2v2000- bf957	73	1520 slices	1168,(SHA-256)	0,768
Virtex xc2v2000- bf957	71	2410 slices	1731,(SHA-512)	0,718

## 6. Conclusion

In this paper, a reconfigurable implementation with multi-mode operation is proposed for the SHA-2 hash family. SHA-2 secure hash algorithm is the newest hash function standard. This hash functions family deployed in a broad range of applications with different area-performance requirements. The proposed implementations are compared in terms of operating frequency, throughput

and TPS product. From the comparative, with other related well known works, our system performs much better. They can substitute efficiently the previous SHA-1 implementations, in communication protocols and networks integrity units, with higher supported security level and better achieved performance. This work can be applied efficiently in the implementation of digital signature algorithms, keyed-hash message authentication codes and in random numbers generators architectures.

The introduced system performs efficiently for the two SHA-2 standard functions (256, 512). The allocated resources of the proposed system are almost the same with the covered area of the separate implementation SHA-512. The achieved performance is almost comparable to the separate implementations performance.

## References

- [1] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.
- [2] National Institute of Standards and Technology, "The keyed-hash message authentication code", Federal Information Processing Standards 198, March 2002.
- [3] W. Stallings, "Network and Internetwork Security: Principles and Practice", Prentice Hall International, 1995.
- [4] National Institute of Standards and Technology, "Digital Signature Standard", Federal Information Processing Standards 186-2, January 2002.
- [5] WAP Forum: WAP White Paper, www.wapforum.org, 2002.
- [6] HiperLan2 Global Forum, Hiperlan specifications, www.hiperlan2.com, 2002.
- [7] National Institute of Standards and Technology, "Secure Hash Standard", Federal Information Processing Standards 180-1, April 1995.
- [8] National Institute of Standards and Technology, "Advanced Encryption Standard", Federal Information Processing Standards 197, November 2001.
- [9] National Institute of Standards and Technology, "Secure Hash Standard", Federal Information Processing Standards 180-2, August 2002.
- [10] A. Stoica, R. Zebulum, D. Keymeulen, R. Tawel, T. Daud, A. Thakoor, "Reconfigurable VLSI architectures for evolvable hardware: From experimental field programmable transistor arrays to evolution-oriented chips", *IEEE Trans on VLSI*, 9(1), pp. 227–232, 2001.
- [11] N. Shirazi, W. Luk, P. Y. K. Cheung, "Framework and tools for run-time reconfigurable designs", *Computers and Digital Techniques*, IEE Proceedings, 147(3), pp.147-152, 2000.
- [12] P. James-Roxby, E. Cerro-Prada, S. Charlwood, "Core-based design methodology for reconfigurable computing applications", *Computers and Digital Techniques*, IEE Proceedings-Publication, 147(3), pp.142-146, 2000.
- [13] M.Zeghid, B.Bouallegue, A.Baganne, M.Machhout, R.Tourki. "Reconfigurable Implementation of the New Secure Hash Algorithm". In *Proceedings of the International Conference on Availability, Reliability and Security 2007 (ARES 2007)*, pp. 281–285, 2007.
- [14] R.P. McEvoy, F.M. Crowe, C.C. Murphy, W.P. Marnane. "Optimisation of the SHA-2 Family of Hash Functions on FPGAs". In *Proceedings of the Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, IEEE Computer Society, pp.317–322, 2006.
- [15] M. McLoone, J. V. McCanny, "Efficient single-chip implementation of SHA-384 & SHA-512". In *Proceedings of the International Conference on Field-Programmable Technology (FTP)*, pp. 311–314, 2002.
- [16] N. Sklavos, O. Koufopavlou. "Implementation of the SHA-2 Hash Family Standard Using FPGAs", *The Journal of Supercomputing*, 31(3), pp.227–248, 2005.
- [17] R.Glabb, L.Imbert, G.Julien, A.Tisserand, N.Veyrat-Charvillon. "Multi-mode operator for SHA-2 hash functions", *Journal of systems architecture*, 53(2-3), pp.127-138, 2007.
- [18] A.Imtiaz, A. Shoba Das. "Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs", *Computers and Electrical Engineering*, 31(6), pp. 345–360, 2005.
- [19] N.Sklavos, G.Dimitroulakos, O.Koufopavlou. "An Ultra High Speed Architecture for VLSI Implementation of Hash Functions". In *Proceedings of ICECS*, pp. 990–993, 2003.
- [20] J.M. Diez, S. Bojanic, Lj. Stanimirovic, C. Carreras, O. Nieto-Taladriz. "Hash Algorithms for Cryptographic Protocols: FPGA Implementations". In *Proceedings of the 10th Telecommunications Forum, TELFOR2002*, Belgrade, Yugoslavia, May 26–28, 2002.
- [21] M.Harris, A.P. Kakarountas, O.Koufopavlou, C.E. Goutis. "A Low-Power and High-Throughput Implementation of the SHA-1 Hash Function". In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'05)*, pp. 4086–4089, 2005.

## Author Biographies

**Medien Zeghid** received his M.S. degree in Electronic Materials and Dispositifs from the Science Faculty of Monastir, Tunisia, in 2005. Currently, he is a PhD student. His research interests include Security Networks, implementation of standard cryptography algorithm, Multimedia Application, Network on Chip: NoC. He is working in collaboration with LESTER Laboratory, Lorient, France.

**Belgacem Bouallegue** received his MSc in Physic Microelectronic and his DEA in Electronic Materials and Dispositifs from the Science Faculty of Monastir, Tunisia, in 1998 and 2000, respectively. Currently, he is a PhD student. His research interests include High Speed Networks, Multimedia Application, Network on Chip: NoC, flow and congestion control, interoperability and performance evaluation. He is working in collaboration with LESTER Laboratory, Lorient Cedex 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 Assistant Professor at University of Monastir, Tunisia. His research interests include implementation of standard cryptography algorithm, key stream generator and electronic signature on FPGA.

**Adel Baganne** born in 1968 is presently an Associate Professor at the UBS University and member of the LESTER Lab. He received his Ph.D. degree in Signal Processing and Telecommunications at the University of Rennes, France, in 1997 and the Engineer degree in Electronics from the National Superior Engineering School in Angers (ESEO), France, in 1993. His research interests include communication synthesis, codesign, co-simulation, computer architecture, VLSI design and CAD tools.

**Rached Tourki** was born in Tunis, on May 13 1948. He received the B.S. degree in Physics (Electronics option) from Tunis University, in 1970; the M.S. and the Doctorat de 3eme cycle in Electronics from

Institut d'Electronique d'Orsay, Paris south University in 1971 and 1973 respectively. From 1973 to 1974 he served as microelectronics engineer in Thomson CSF. He received the Doctorat d'etat in Physics from Nice University in 1979. Since this date he has been professor in

Microelectronics and Microprocessors with the physics department, Faculty of Sciences of Monastir. His current research interests include: Digital signal processing and hardware software codesign for rapid prototyping in telecommunications.