IMPLEMENTATION OF SHA-256 ALGORITHM IN FPGA BASED PROCESSOR

SOUNDARYA.J PG SCHOLAR,[1]
SRI SHAKTHI INSTITUTEOF ENG AND TECH, COIMBATORE-641062.

K.VENKATESA..,ME [2,] *Assistent professor.*
SRI SHAKTHI INSTITUTE OF ENG AND TECH, COIMBATORE-641062.

Abstract:

Hash functions play a significant role in today's cryptographic applications. SHA (Secure Hash Algorithm) is famous message compress standard used in computer cryptography, it can compress a long message to become a short message abstract. In this paper, SHA- 256 hash algorithm has been implemented using Verilog HDL (Hardware Description Language). The SHA-256 source code is divided into three modules, namely Data path, Memory and Top module. The Verilog code is synthesized using Xilinx software tool. The test vectors have been applied to verify the correctness of the SHA-256 functionality. A comparison between the proposed SHA-256 hash function implementation with other related works shows that it achieves the introduced system can working on reuse data, minimize critical paths and reduce the memory access by using cache memory, reducing clock cycles and needs less silicon area resources. The achieved performance in the term of throughput of the proposed system/architecture is much higher than the other hardware implementations. The proposed system could be used for the implementation of integrity units, and in many other sensitive cryptographic applications, such as, digital signatures, message authentication codes and random number generators.

1.Introduction:

cryptographic algorithms have been considered slow, demanding high computational resources and inefficiently implemented in conventional general purpose processors. 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. The processor approach is more oriented to use less amount of area resource, the co-processor approach is more oriented to perform the algorithmic operations

ISSN: 2278 – 909X

faster. 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 SHA algorithm. A hash function is a sort of operation that takes an input and produces a fixed- size string which is called the hash value. The input string can be of any length depending on the algorithm used. The produced output is a condensed representation of the input message or document and usually called as a message digest, a digital fingerprint or a checksum. The size of the message digest is fixed depending on the particular algorithm being used.

This means that for a particular algorithm, all input streams yield an output of same length. Furthermore a very small change in the input results with a completely different hash value. Hash functions can be classified as keyed and un keyed hash functions. The keyed hash functions take a secret key as an additional input parameter. In this case, the above defined characteristics of hash functions are satisfied for any value of the secret key. Keyed hash functions are also named as Message Authentication Codes or MACs. In this study, we only deal with un keyed hash functions.

- The hash value of the document is calculated.
- The calculated hash value is encrypted with the private key, thereby the document is signed
- The document and the signed hash value are send to the recipient
- The recipient calculates the one way hash value of the document and decrypts the signed hash value by using the public key. If the signed hash value is the same with the calculated hash value, then the signature is valid.

There are two brute-force attacks to a hash function. In a brute force, random inputs are tried and the results of the computations are stored until a collision is found. The first attack can be described as follows: Suppose that the hash of a specific message is given, an adversary can try to find another message which has the same hash value. On the other hand, the second attack can be explained as follows: suppose that an adversary tries to find to messages that have the same hash value. This attack is easier than the first one and known as *birthday attack*

2.SHA algorithm:

In order to catch security levels offered by other

ISSN: 2278 – 909X

cryptographic algorithms, NIST developed the three new hash functions: SHA-256, SHA-384 and SHA-512. These hash functions are standardized with SHA-1 as SHS (Secure Hash Standard). A 224-bit hash function SHA-224, based on SHA-256, has been added to SHS in 2004. Hash calculations are mainly composed of three sections. In the first part the incoming message is padded and fixed sized message blocks are prepared according to the particular hash function being applied. After these padding operations, the message schedule is prepared. In this state, message block is further divided into sub blocks to be used in each round of the hash calculation process. In the hash calculation process message digest is computed after some specific number of iterations related to the algorithm by using

- Algorithm specific constants
- Message words prepared by the message scheduler
- The chaining variables

Hash functions can be implemented in hardware or software. However, as security and throughput requirements of the systems increase, it is found that software implementations cannot provide desired security and throughput values As a result, it is preferred to implement the hash functions in hardware. There are several hash function implementations in the literature and commercially available in the market.

2.1SHA -256:

The SHA-2 family was published in 2002 by the National Institute of Standards and Technology (NIST). 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. 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)... D(N). Each block of data D(i) is processed sequentially by a main function during 64 rounds. A partial 256-bit hash value Hi 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.

2.2SHA main loop:

Require: D(i) a 512-bit block

ISSN: 2278 – 909X

Ensure: The HASH value H(i)
corresponding to D(i) from H(i_1)
1: for t from 0 to 63 do
2: Prepare Wt
3: Compute∑(a)
4: Compute Maj(a; b; c)
5: Compute T2
6: Compute∑1(e)
7: Compute Ch(e; f ; g)
8: Compute T1
9: h← g
10: g← f
11: f ←e
12: e ←d +T1
13: d ←c
14: c← b
15: b← a
16: a ←T1 + T2
17: end for
18: H(temp) = a|b|c|d|e|f |g|h
19: H(i) ←H(i_1) + H(temp)
20: return H(i)

3.Proposed system:

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.

Lets consider again the main operations computed in the internal rounds of the SHA-256 algorithm. The equations for computing Wt ; T1; T2; 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

ISSN: 2278 – 909X

that can be efficiently computed in a specialized 4-input ALU.

The datapath for the proposed SHA-256 processor is shown in A customized ALU named HASH_ALU computes Eqs. 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.

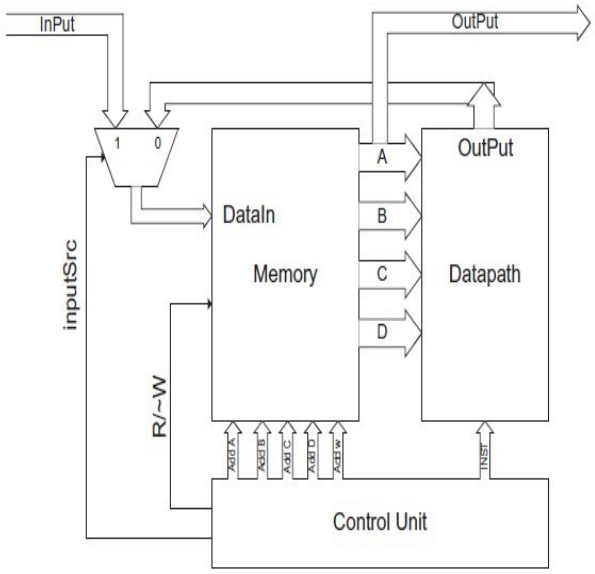

Fig: 3.1 block diagram of SHA- 256 processor

The R module is a forwarding unit that 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 Ty _ 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 and the subtraction between the incoming data R1 and the internal register R2. The R2 register stores the incoming operand C from the previous iteration. The block diagrams of the HASH_ALU, AU and R modules are depicted.

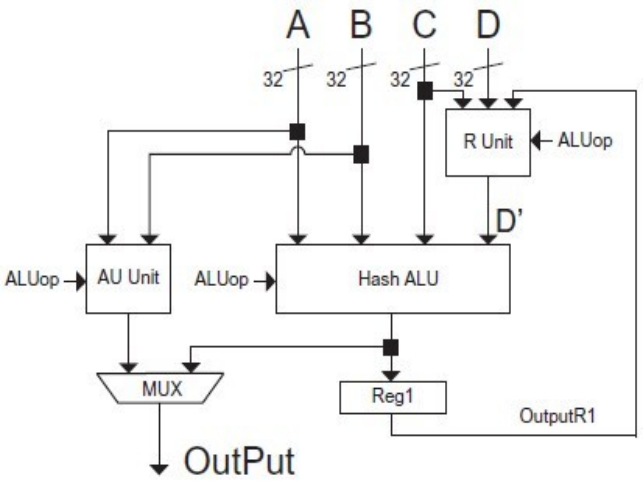

Fig: 3.2data path for computing basic operation in SHA 256 algorithm

During the execution of an internal round of the SHA-256 algorithm in the interval 0 6 t < 16, the data path contains the values shown. When 16 6 t < 64, the data flow in the data path is as shown. 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.

ISSN: 2278 – 909X

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:

S0 – The initial state waits for new message to be hashed.

S1 – Loads the 16 words of the input message, one at each clock cycle.

S2 – Loads the initial hash H(0) or update the intermediate hash value H(i).

S3 – Computes Wt , being Wt =Mt for the first 16 rounds. After that, Wt is computed using the HASH_ALU.

S4 – Computes T(x) the result is not stored in memory

S5 – Computes Ty. First computes d + T1 (see equation) 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.

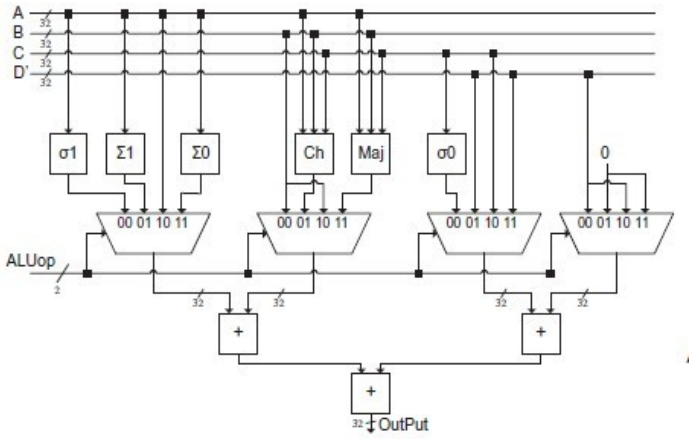

Fig 3.3 Internal structure of main modules in the data path for the proposed SHA-256 processor.

S6 – Computes T(z) and stores the result in the variable h. Again, the same reasoning applied in the state S5 is used to avoid unnecessary assignments and save time.

S7 – Computes the intermediate hash value H(i).

S8 – All data blocks are processed. The final value in H(N) is taken and placed in the data out 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,

ISSN: 2278 – 909X

useful parameters for measuring the Throughput and Efficiency of Performance.

- Throughput = Data block size / Clock time _ Clock cycles
- Efficiency =Throughput / Number of slices

The FPGA implementation results are shown ,

Implementation results.

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

Fig:4.1 fpga implementation

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 Hi to the working variables a–h.

The internal loop (64 rounds) are computed in two sequences:

- During the first 16 rounds Wt is taken from the incoming 512-bit data block and only 3 cycles are used.
- For the next 48 rounds, Wt 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.

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 (i) Simplicity favours 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]

ISSN: 2278 – 909X

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 2 among representative FPGA implementations of SHA algorithm under the same conditions.

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,

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

Fig 4.2 comparison results

490 clock cycles with a clock frequency of 71.5MHz 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 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.

ISSN: 2278 – 909X

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 VERILOG 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-3 algorithm to derive the customized 4-input ALU and its associated datapath to get a more compact SHA processor. Our design can be easily extended to other hash algorithms of the SHA-3 family since all of them exhibit a similar functionality.

6.REFERNCE:

[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.

ISSN: 2278 – 909X

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.

ISSN: 2278 – 909X

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.

R. García et al. / Computers and Electrical Engineering 40 (2014) 194–202 201

ISSN: 2278 – 909X