ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.14, No.11, 1–9

A lightweight implementation of the Tav-128 hash function

Honorio Martin[1a)], **Pedro Peris Lopez**[2,3b)], **Enrique San Millan**[1], **and Juan E. Tapiador**[2]

[1] *Department of Electronic Technology, University Carlos III of Madrid, Spain*

[2] *Department of Computer Science, University Carlos III of Madrid, Spain*

[3] *Department of Computer Science, Aalto University, Finland*

a) *hmartin@ing.uc3m.es*

b) *pperis@inf.uc3m.es*

Abstract: In this article we discuss the hardware implementation of a lightweight hash function, named Tav-128 [1], which was purposely designed for constrained devices such as low-cost RFID tags. In the original paper, the authors only provide an estimation of the hardware complexity. Motivated for this, we describe both an ASIC and an FPGA-based implementation of the aforementioned cryptographic primitive, and examine the performance of three architectures optimizing different criteria: area, throughput, and a trade-off between both of them.

Keywords: hardware implementation, hash function, ASIC, FPGA

Classification: Integrated circuits

References

[1] P. Peris-Lopez, *et al.*: in *Emerging Directions in Embedded and Ubiquitous Computing*, LNCS, vol. 4809 (Springer, 2007) 781–794.

[2] M. Feldhofer and C. Rechberger: in *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, LNCS, vol. 4277 (Springer, Berlin, Heidelberg, 2006) 372–381.

[3] X. Guo, *et al.*: "Silicon implementation of sha-3 finalists: BLAKE, Grøstl, JH, Keccak and Skein," ECRYPT II Hash Workshop (2011).

[4] A. Shamir: in *Fast Software Encryption*, LNCS, vol. 5086 (Springer, Berlin, Heidelberg, 2008) 144–157.

[5] S. Badel, *et al.*: in *Cryptographic Hardware and Embedded Systems, CHES*, LNCS, vol. 6225 (Springer, Berlin, Heidelberg, 2010) 398–412.

[6] M. A. Abdelraheem, *et al.*: in *Advances in Cryptology - ASIACRYPT*, LNCS, vol. 7073 (Springer, Berlin, Heidelberg, 2011) 308–326.

[7] P. M. Mukundan, *et al.*: "Hash-one: A lightweight cryptographic hash function," IET Inf. Secur. **10** (2016) 225 (DOI: 10.1049/iet-ifs.2015.0385).

[8] J. Guo, *et al.*: in *Advances in Cryptology - CRYPTO*, LNCS, vol. 6841 (Springer, Berlin, Heidelberg, 2011) 222–239.

[9] J.-P. Aumasson, *et al.*: "Quark: A lightweight hash," J. Cryptol. **26** (2013) 313 (DOI: 10.1007/s00145-012-9125-6).

[10] A. Bogdanov, *et al.*: "Spongent: The design space of lightweight cryptographic hashing," IEEE Trans. Comput. **62** (2013) 2041 (DOI: 10.1109/TC.2012.196).

[11] S. Mikami, *et al.*: "Fully integrated passive uhf rfid tag for hash-based mutual authentication protocol," Sci. World J. **2015** (2015) 498610 (DOI: 10.1155/2015/498610).

© IEICE 2017

DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

[12] N.-W. Lo, *et al.*, ed.: *RFIDsec'14 Asia Workshop Proceedings*, Cryptology and Information Security Series, vol. 12 (IOS Press, 2014).
[13] J. C. Hernandez-Castro, *et al.*: "Wheedham: An automatically designed block cipher by means of genetic programming," IEEE Congress on Evolutionary Computation (2006) 192 (DOI: 10.1109/CEC.2006.1688308).
[14] A. Akhshani, *et al.*: "Pseudo random number generator based on quantum chaotic map," Commun. Nonlinear Sci. Numer. Simul. **19** (2014) 101 (DOI: 10.1016/j.cnsns.2013.06.017).
[15] A. Kumar, *et al.*: in *Progress in Cryptology - INDOCRYPT*, LNCS, vol. 6498 (Springer, Berlin, Heidelberg, 2010) 118–130.
[16] Mentor Graphics: ModelSim SE User's Manual. Software Version 6.5c, August 2009.
[17] Synopsys: 90 nm Generic Core Cell Library. Data Book. Rev.: 0.3., February 2009.
[18] D. Brenk, *et al.*: "Energy-efficient wireless sensing using a generic adc sensor interface within a passive multi-standard RFID transponder," IEEE Sensors J. **11** (2011) 2698 (DOI: 10.1109/JSEN.2011.2156782).
[19] M. O'Neill: "Low-cost sha-1 hash function architecture for rfid tags," Proc. of Workshop on RFID Security (2008) 41.
[20] Xilinx: ISE In-Depth Tutorial, April 2012.
[21] Xilinx: Spartan-3E FPGA Family Data Sheet, July 2013.
[22] Opencores: SHA cores: Overview, December 2012.

1 Introduction

There is a great variety of hash functions. For a general context, MD5 and SHA family are commonly employed — although the use of MD5 is not currently recommended. In detail, the cost of implementing SHA-256, SHA-1 and MD5 is around 10.9 K, 8.1 K and 8.4 K Gates Equivalents (GE) [2] respectively. In 2008, NIST SHA-3 competition was launched to develop a new general-purpose hash function and the proposals focused on software efficiency. In fact, any SHA-3 finalists (BLAKE, Grøstl, JH, Keccak and Skein) consume more than 30 K GEs [3].

The widely use of limited devices is behind the new lightweight hash functions, in which design the hardware restrictions play a key role. In consonance with this, Shamir proposed SQUASH, inspired by the Rabin encryption scheme, and is expected to offer a tiny footprint [4]. ARMADILLO hash function is another interesting proposal (2,9 K GEs with a fully serial architecture [5]), but unfortunately it present serious security weaknesses [6]. Another step towards the design of compact hash function are those based on sponge functions [7]. Quark [8], Photon [9] and SPONGENT [10] are example of these constructions, and the circuit area demanded for its implementation is extremely tiny. In detail, for a 64-bit collision resistance, U-Quark, Photon-128 and SPONGENT-128 consume around 1.5 K GE [10].

Contribution: The Tav-128 lightweight hash function was proposed in [1] as a design suitable for low-cost RFID tags. It follows a classical Merkle-Damgård structure similar to those used in the MD and SHA families. The authors analyzed the statistical properties of its output and provided an estimation of the hardware footprint required, stating that around 2.6 K GEs would be needed. In this article,

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

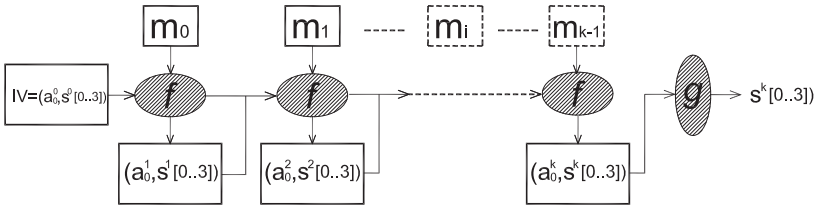

Fig. 1. Merkle-Damgård structure of Tav-128.

we consider three different ASIC architectures for Tav-128 and discuss the results (chip area, power consumption, and throughput) obtained both with an ASIC and an FPGA implementation. Finally, it is worth noting that for the integration of this crypto module within a RFID tag some extra and crucial components like the analogue module would be necessary—the reader is urged to consult [11] or [12] where the authors present a fully integrated passive UHF RFID tag with a hash function on-board.

2 The Tav-128 hash function

Tav-128 follows a Merkle-Damgård structure (see Fig. 1), where the input message is split into 32-bit blocks and a 128-bit output is generated. The compression function $f : \{0,1\}^{32} \times \{0,1\}^{160} \rightarrow \{0,1\}^{160}$ makes use of two filter functions (called A and B) and two expansion functions (called C and D). The internal state is composed of five 32-bit words (register a_0^k plus the four states $S^k[0,\ldots,3]$), and the final output consists of the four 32-bit state registers $S^k[0,\ldots,3]$. The finalization function g truncates the state and outputs its 128 least significant bits, i.e., $g(a_0^k, S^k[0,\ldots,3]) = S^k[0,\ldots,3]$. The structure of Tav-128 is shown in Fig. 2.

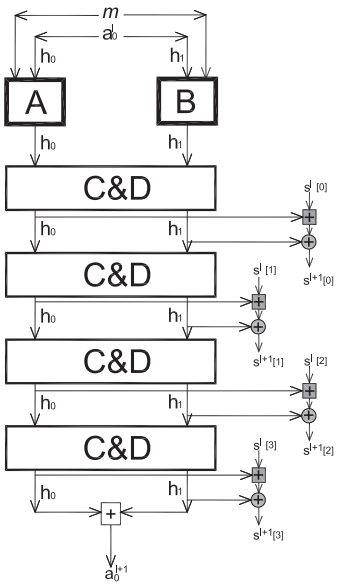

Fig. 2. Compression function f in Tav-128.

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

Algorithm 1 Filter functions A and B of Tav-128

function A$((h_0, m)$
  for $i \leftarrow 0, 31$ **do**
    $h_0 \leftarrow (h_0 \ll 1) + (h_0 + m) \gg 1)$
  end for
end function
function B(h_1, m)
  for $i \leftarrow 0, 31$ **do**
    $h_1 \leftarrow (h_1 \gg 1) + (h_1 \ll 1) + h_1 + m$
  end for
end function

The filter functions A and B were designed to make it difficult for an attacker to have access to the internal state. Otherwise, the construction would have a fundamental flaw that has been used in the past to attack other cryptographic primitives. A pseudocode description of functions A and B is provided by Algorithm 1.

The expansion function is built as a sequential-iterative execution of functions C and D, whose pseudocode is provided in Algorithm 2. These functions were designed with a twofold objective. On the one hand, the expansion have to be efficient both in terms of circuit size (GEs) and throughput. On the other hand, the functions must be highly non-linear. This multiobjective design problem was approached from a genetic programming perspective in [13]. Thus, a search algorithm was used to explore potential functions composed of lightweight operators, and taking into account both the circuit size and the resulting non-linearity of the ouput. The design of Tav-128 reported in [1] was carried out following the same principles.

Algorithm 2 Expansion functions C and D of Tav-128

for $j \leftarrow 0, 3$ **do**
  for $i \leftarrow 0, 7$ **do**
    function $C(h_0, h_1)$
      $h_0 \leftarrow h_0 \oplus ((h_1 + h_0) \gg 3)$;
      $h_0 \leftarrow ((((h_0 \gg 2) + h_0) \gg 2) + (h_0 \ll 3) + (h_0 \ll 1)) \oplus$
$0x736B83DC$
    end function
    function $D(h_0, h_1)$
      $h_1 \leftarrow h_1 \oplus ((h_1 \oplus h_0) \gg 1)$
      $h_1 \leftarrow (h_1 \gg 4) + (h_1 \gg 3) + (h_1 \ll 3) + h_1$
    end function
  end for
  $S[j] \leftarrow S[j] + h_0$
  $S[j] \leftarrow S[j] \oplus h_1$
end for
  $a_0 = h_1 + h_0$

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

In terms of security, the output of Tav-128 was assessed against a suite of standard and cryptographic randomness test, including ENT, DIEHARD, and the NIST suites [14]. Although the obtained results do not show any evidence of weaknessess, a more exhaustive analysis conducted by Kumar et al. [15] demonstrates that the security level of Tav-128 is lower than the maximum achievable. Despite this, the design is still attractive for a number of reasons. For example, the study of the constituent elements carried out in [15] show that the concatenation of functions A and B produces a 64-bit permutation from 32-bit messages, which could be a useful cryptographic component for future designs.

3 Hardware architectures for Tav-128

In this section, we present three architectures for the hardware implementation of Tav-128. Since this is a hash function intended for constrained devices (e.g., low-cost RFID tags or sensor nodes), the proposed architectures are aimed at optimizing some of the critical parameters found in this technology: footprint, power consumption, and throughput. All the studied architectures consist of at least two 32-bit registers (h_0 and h_1) plus a state register of 128 bits ($S^k[0,\ldots,3]$). As previously shown in Algorithms 1 and 2, three counters are used in the hash function: one for the top-level loop in the filter function and two for the nested loops in the expansion function. In order to reduce the circuit area, two hardware counters are employed in the proposed implementation. Finally, all the associated control logic is implemented by a Finite State Machine (FSM).

In the first proposed architecture, called τ-Tav-128, the main goal is to achieve a high throughput. The second architecture, named μ-Tav-128, aims at reducing the circuit area measured in GEs. Finally, the third architecture, called υ-Tav-128, attempts to reach a trade-off between area and throughput. Fig. 3 shows a high-level architectural view of the main blocks of the design. The building block at the bottom represents the operations supported and will be different for each architecture: between one and five 32-bit adders depending on the architecture. We next describe each one of them in more detail.

Architecture I: τ-Tav-128 In this architecture, all operations are computed within the minimum possible number of clock cycles in order to maximize throughput. This is achieved by using five 32-bit adders, which allows computing both filter functions A and B in parallel. Furthermore, these adders are also employed in the expansion functions C and D.

Architecture II: μ-Tav-128 This architecture attempts to optimize the chip area by using only one adder rather than the five required by the first design. This implies that the filter functions A and B are executed sequentially. As a consequence, the area is optimized at the expense of decreasing throughput.

Architecture III: υ-Tav-128 Finally, in this architecture we try to reach a trade-off between minimizing the circuit area while maximizing throughput. The design can be seen as a midpoint between τ-Tav-128 and μ-Tav-128. In particular, we used two adders, as this is the minimum number required to computer the filter functions A and B in parallel.

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

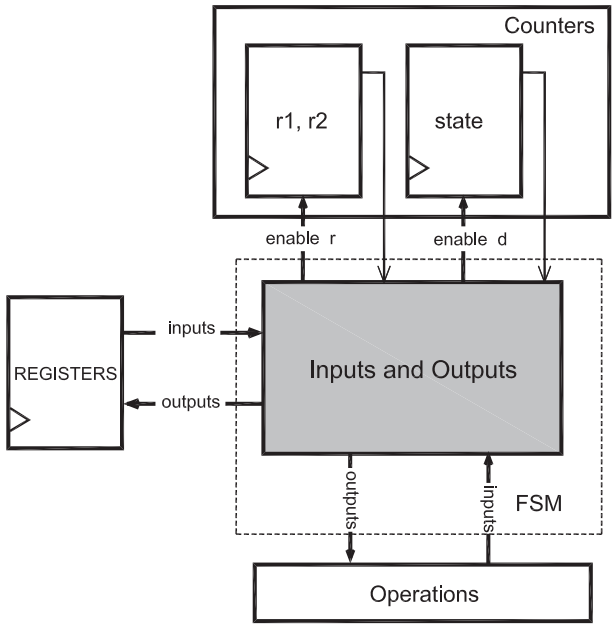

Fig. 3. Main architectural blocks for Tav-128.

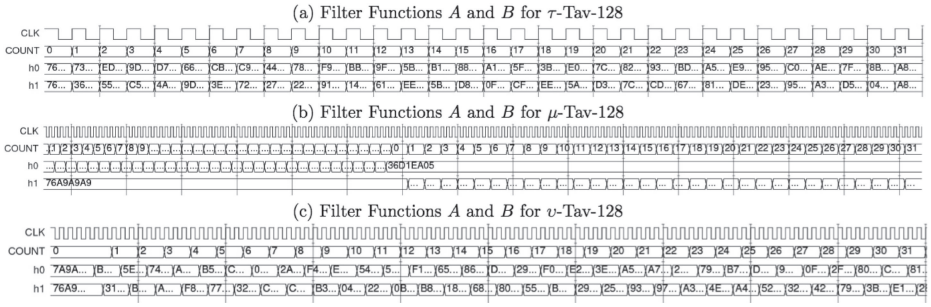

Fig. 4. Timing diagrams for the filter functions A and B in the three proposed architectures.

4 Experimental results

We next present the results obtained from the implementation of the three architectures decribed above. For each one of them, we provide the resulting circuit area in GEs, the number of clock cycles consumed, its throughput in Kbps, and an efficiency ratio measured as the throughput divided by the chip area.

4.1 Simulation results

The execution of the filter functions A and B differs considerably on each of the proposed architectures. Recall that function A updates h_0, while h_1 is updated by the B. As these functions are not nested and, therefore, can be executed independently, we use them to show the functioning and differences between the three proposed architectures.

An execution example is provided in Fig. 4. These simulations have been obtained with the Modelsim HDL Simulator 6.5.c. [16]. We can observe in Fig. 4(a) how τ-Tav-128 offers the higher throughput, consuming only 32 clock cycles for the parallel execution of functions A and B. Contrarily, Fig. 4(b) shows the sequential behavior of μ-Tav-128: function A is first executed, consuming 64

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

Table I. ASIC implementation results

Architec.	GE	Power (nW)	Clock cycles	Throughput (Kbps)	Tput /GE
τ-Tav-128	5251	431.9	320	40.0	7.6
μ-Tav-128	3194	329.8	608	21.0	6.6
υ-Tav-128	4106	407.6	448	28.6	6.9

clock cycles, and then function B takes 96 additional clock cycles for its computation. Finally, the use of two adders in υ-Tav-128 allows the parallel computation of A and B, but each update of h_0 or h_1 must be carried out sequentially. Overall, it takes 96 clock cycles to complete one execution in this case.

4.2 ASIC results

The results presented in this section have been obtained using the Synopsys software and the Faraday 90 nm of UMC library [17]. This library was selected for the implementation because it provides information about the layout of the basic cells. Therefore, the obtained results can be considered as a good estimation of the result that would be obtained if the circuit were manufactured. All the tests have been performed for a clock operation frequency of 100 KHz, which is one of the most common values for low-cost RFID tags [2] and, in general, for lightweight implementations [7, 10]. In addition, the employed library sets a low-power supply of 1.25 V.

Table I presents the synthesis results for the three architectures evaluated. In order to facilitate comparisons with other proposals and make the results as much independent as possible from the used technology, the chip area is provided in GEs. Normalization to GEs is obtained by dividing the obtained circuit area by the area of a NAND gate—in our particular case, a NAND gate occupies 3.16 μm^2. As shown in Table I, τ-Tav-128 is the less efficient in terms of circuit area. Conversely, μ-Tav-128 represents the most efficient implementation, resulting in a reduction of around 39% and 22% of the chip area in comparison to τ-Tav-128 and υ-Tav-128, respectively.

The synthesis results also show that the three evaluated architectures offer a low power consumption. Specifically, the obtained measures are well below 18 μW (1.2 V $\times$ 15 μA), which is the limit commonly assumed for RFID implementations [18]. We emphasize that the figures provided represent the sum of the static and dynamic consumptions, and the value is directly facilitated by the synthesis tool.

The third column in Table I shows the number of clock cycles consumed to compute one output. In the worst case, these values are at least three time less than the limit of 1800 clock cycles suggested by Feldhofer [2] to compute an interleaved challenge-response protocol suitable for RFID systems. As expected, τ-Tav-128 is the most efficient architecture in this regard, offering a throughput improvement of around 47% and 26%, respectively, compared to μ-Tav-128 and υ-Tav-128.

Finally, as an efficiency measure, we have computed the ratio between the throughput (kbps) and the chip area (GEs). The overall result is that τ-Tav-128 is the most efficient solution, which makes this architecture suitable for applications

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

Table II. Comparison between Tav-160 and a representative gropup of 160-bit hash functions

Architecture	GE	Clock cycles	Throughput (Kbps)	Tput /GE
τ-Tav-160	6443	320	50.0	7.7
μ-Tav-160	3930	608	26.3	6.7
υ-Tav-160	5030	448	35.7	7.1
SHA-1 [19]	6122	344	46.5	7.6
SHA-1 [2]	8120	1274	12.6	1.5
SPONGENT-160 [10]	2406	90	17.8	7.1
PHOTON-160 [10]	2849	180	20.0	7.0
D-QUARK (144) [10]	3695	88	18.2	4.9

where there are no severe restrictions concerning the circuit area. Both μ-Tav-128 and υ-Tav-128 offer similar efficiency, and choosing one or the other depends on whether restrictions come from footprint area or throughput, respectively.

Finally, in Table II we have compared Tav-128 to SHA-1 and to some recently proposed lightweight hash functions. Note that SHA-1 and Tav-128 were designed following a Merkle-Damgård structure. In order to make a fair comparison, we have implemented Tav-160, which is identical to Tav-128 except that five 32-bit registers are used for the state ($S^k[0,\ldots,4]$). Power consumption is omitted in this comparison since it highly depends on the technology and supply voltage. We can observe that the three proposed architectures for Tav-160 offer an efficiency four times higher than the implementation of SHA-1 presented in [2]. In comparison with the implementation presented in [19], the efficiency is similar but υ-Tav-160 and μ-Tav-160 require smaller footprints. On the other hand, the sponge functions SPONGENT-160 and PHOTON-160 do not offer advantage in terms of efficiency (i.e., Throughput/GE) but these primitives consume much less circuit area [10]. On the other hand, D-QUARK requires a footprint close to the one demanded by μ-Tav-160 but with a slight degradation in the offered throughput [7, 10].

4.3 FPGA results

Apart from the ASIC implementation discussed above, we have explored various implementatios of Tav-128 in a Field-programmable Gate Array (FPGA). In this case we used the Xilinx ISE Design suite 13.4 [20] and the experimentation was conducted with the Spartan3E XC3S500E board [21]. We used an efficient FPGA implementation of SHA-1 which is freely available in [22]. Considering that we do not have severe hardware restrictions on the FPGA, τ-Tav-128 is a priori the most suitable architecture for this sort of environments. In particular, we implemented τ-Tav-128 and τ-Tav-160 and compared them with SHA-1. In Fig. 5, we show the final state for a full execution of Tav-128 hash function in Spartan3E.

Table III summarizes the results obtained for both primitives. This stresses the importance of each building block in the hash function design. Note that, as a consequence of using a high volume of combinational logic, τ-Tav-128 and τ-Tav-160 both demand a slightly higher number of slices than SHA-1. Contrarily, in

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017

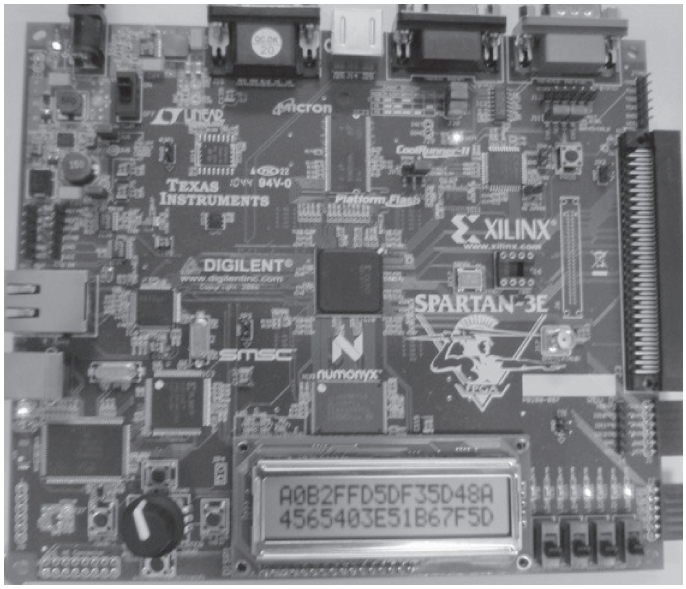

Fig. 5. FPGA Implementation of Tav-128

Table III. Comparison between FPGA implementations of Tav-128, Tav-160, and SHA-1

Hash	Slice FF	4 Inputs LUTs	Number of Slices	Max Frequency (MHz)
τ-Tav-128	251 (2%)	1187 (12%)	605 (12%)	90.25
τ-Tav-160	316 (3%)	1438 (15%)	766 (16%)	89.7
SHA-1 [22]	664 (7%)	867 (9%)	554 (11%)	80.86

terms of storage needs SHA-1 is three and two times more demanding than τ-Tav-128 τ-Tav-160, respectively. Finally, it can be observed that both implementations of Tav support a maximum operating frequency around 10 MHz higher than that of SHA-1.

5 Conclusions

In [1], the authors proposed a new ligthweight hash function called Tav-128 and provided an estimation of the hardware complexity using high-level arguments. In this paper, we have reported our results with a hardware implementation of Tav-128, both in ASIC and in an FPGA. We have explored various architectures focusing either on minimizing the footprint area, the throughput, or both. The most efficient proposal in terms of GEs, called μ-Tav-128, consumes a slightly bigger area than the estimation presented in [1]; i.e., 3194 GEs versus 2578 GEs originally suggested. This difference is mainly due to the complexity of the control logic, which is often underestimated.

Finally, it is worth mentioning that modern designs of lightweight hash functions have been proposed in the last years. Those based on sponge functions (e.g., SPONGENT, QUARK or PHOTON) are promising and offer a small footprint [7, 10]. Tav-128 is not as efficient as these proposals, but could fit well on limited devices like low-cost RFID tags or sensor nodes as it takes between 3 K GEs and 5 K GEs, depending on the desired throughput.

© IEICE 2017
DOI: 10.1587/elex.14.20161255
Received December 21, 2016
Accepted February 28, 2017
Publicized May 18, 2017
Copyedited June 10, 2017