FPGA-Based Design Approaches of Keccak Hash Function

George Provelengios
National and Kapodistrian University of Athens
Athens, Greece
e-mail: georprob@gmail.com

Paris Kitsos
Computer Science
Hellenic Open University
Patras, Greece/
KNOSSOSnet Research Group
Patras, Greece
e-mail: pkitsos@ieee.org

Nicolas Sklavos
KNOSSOSnet Research Group
Technological Educational Institute of Patras
Patras, Greece
e-mail: nsklavos@ieee.org

Christos Koulamas
Industrial Systems Institute
Patras, Greece
e-mail: koulamas@isi.gr

***Abstract*—Keccak hash function has been submitted to SHA-3 competition and it belongs to the final five candidate functions. In this paper FPGA implementations of Keccak function are presented. The designs were coded using HDL language and for the hardware implementation, a XILINX Virtex-5 FPGA was used. Some of the proposed implementations use DSP48E blocks in order to accelerate the designs execution. So, comparisons between the proposed designs in terms of time performance and FPGA resources are given in order to examine the efficiency of the using DSP48E blocks. Also, comparisons with previous published works are provided.**

Keywords- SHA-3 competition, Keccak hash function, DSP48E, VIRTEX-5 FPGA

I. INTRODUCTION

It is well known that the old fashion hash functions standards are candidates for attacking using up to date cryptanalysis methods. A good example of this assumption is Secure Hash Algorithm-1 (SHA-1) [1] that it has attacked some years ago [2]. As well SHA-2 [3] standard functions are designed with similar philosophy as SHA-1 function, so is a possible subject for an attack. The National Institute of Standard and Technology (NIST) have been organized a public competition to develop a new cryptographic hash function standard [4] (call SHA-3) that will replace the SHA-2 standard.

Keccak [5] hash function has been submitted to SHA-3 competition and it belongs to the final five candidate functions. In this paper FPGA implementations of Keccak hash function are presented. Many FPGA vendors have further increased the proposition of utilizing FPGAs for DSP applications by providing specialized programmable logic blocks for DSP operations. Xilinx has developed FPGA devices such as Virtex-4, Virtex-5, Virtex-6 and Virtex-7 devices, with logic blocks named Xtreme DSP appropriate for DSP applications. This block provides improved flexibility and utilization, application efficiency and overall low power consumption. In simpler terms, a DSP48 block is a multiplier and multiply-accumulator with several specialized features such us multiply add, three-input add, barrel shift, wide-bus multiplexing, magnitude comparator, bit-wise logic functions, pattern detect, and wide counter provided in a dedicated circuit. In this work FPGA-design considerations were given in the sense of using or not using the DSP48 [6] (DSP48E is the name of Xtreme DSP in Virtex-5 FPGA) slice in a VIRTEX-5 [7] device. This work can easily extend to any device that uses similar slices such as Virtex-6 and Virtex-7 family devices.

The rest of this paper is organized as follows: Section II briefly presents the Keccak specifications. Sections III and IV introduces the architectures and the FPGA-based considerations of the Keccak hash function. The next section the FPGA synthesis results and comparisons are given, while the paper conclusions are discussed in the last section.

II. KECCAK SPECIFICATIONS

Keccak is a hash function that based on the sponge construction [8].

Firstly, for the I/O interface, NIST requires the candidate algorithms to support at least four different output lengths $n \in$ (224, 256, 384 and 512). According to the desired output length, the algorithm uses two parameters for the sponge construction. These two input parameters are the bitrate r and capacity c.

In order to obtain the sponge the input data are padded according to "*pad10*1*" rule and then follows some inversions per byte. More specifically, initially attached two

bytes (0x01 and 0x80) to the input message. The position of this concatenation it depends on the two parameters bitrate and capacity and from the input length of data. Then, the entire message is divided into 64-bit words. In each word, each byte changes positions where the last byte comes first and the first byte becomes last. Note that this procedure is not a standard rotation procedure. The above operation gives the Keccak sponge which consists of 1600-bit. The sponge consists of a 2-D array which has five rows and five columns. Each array position consists from 64-bit which also called "*Lane*". So, the twenty five lanes of 64-bit give the total number of 1600-bit. Let the expression α[x][y][z] to get a single bit from each position of the sponge with *x*, *y* and *z* corresponds to rows, columns and lanes respectively.

The main part of Keccak consists of twenty four rounds r where in each round take place five permutations. Bellow these permutations are given:

- $\boldsymbol{\theta}$: α[x][y][z] = α[x][y][z] + ∑α[x-1][y'][z] + ∑α[x+1][y'][z-1], for y'=0:4
- $\boldsymbol{\rho}$: α[x][y][z] = α[x][y][z - (t+1)(t+2)/2] with *t* satisfying $0 \le t \le 24$ and $(0\ 1 : 2\ 3)^t (1 : 0) =$ (x : y) or *t* = -1 if x = y = 0 with (:) denoted as 2x2 array or 2x1 array.
- $\boldsymbol{\pi}$: α[x][y] = : α[x']['y'], with (x : y) = (0 1 : 2 3) (x' : y')
- $\boldsymbol{\chi}$: α[x] = α[x] + (α[x+1] + 1) α[x+2]
- $\boldsymbol{\iota}$: α = α + RC[i_r]

In Iota expression the RC correspond to a round constant value. All the additions and multiplications between the terms are in *GF*(2). All permutations have been occurred only on lanes and not in separated bits. Furthermore, the sequence of the procedures is arbitrarily except from θ expression which should take place at the beginning of each round.

Finally, after twenty four rounds the output of last expression is simply truncated to give the desired hash function output. Note that in the output message should applied the inversion procedure from padding in order to get the right output. This means that again last bytes should come first and the first bytes should become last.

III. KECCAK CORE ARCHITECTURE

The hardware architecture of the Keccak hash function is shown in Figure 1. The design has a 128-bit data input and 2-bit input that define the desired output length. The parameters bitrate and capacity are fixed for the four output lengths so are not necessary extra inputs for these values. Below are given the values for bitrate and capacity depending on output length:

- **224:** Bitrate = 1152, Capacity = 448
- **256:** Bitrate = 1088, Capacity = 512
- **384:** Bitrate = 832, Capacity = 768
- **512:** Bitrate = 576, Capacity = 1024

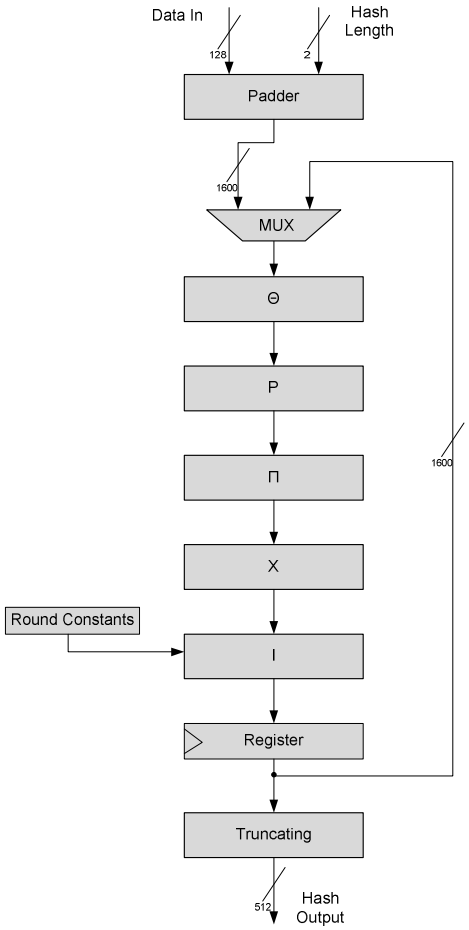

Figure 1. Core architecture of the Keccak hash function.

The Padder component implements the padding and the inversions per byte procedures, as described previously and has an output of 1600-bit which is the sponge of Keccak. Each 64-bit in sequence considered as a single lane beginning from bit in position 1599 of the vector. The following mapping was used. The first vector's lane (1599-1536) is the lane α[0][0], the second vector's lane is the lane α[0][1] and so on. Note that if it is denoted the sponge as a 2D array with lanes, the x parameter points at x-axis while the y parameter points at y-axis so the α[0][0] element is located at the left bottom corner of array.

Then a 2x1 multiplexer drives the data output from Padder to the main components of Keccak. After the padding procedure this multiplexer operates as a feedback multiplexer in order to be operated the twenty four rounds.

Θ component takes the input bits and applies XOR operations between the lanes at each column. As a result

there are five XORed columns. Then, those five columns are left-rotated for one bit and XORed again with the results from previous XOR operations. Finally, the data from the last XOR operations are driven to a finally XOR stage with the component Θ input lanes. In Θ component 50 XOR of 64-bit logic gates are used in total. The hardware architecture of the Θ component is shown in Figure 2.

P component simple rotate left each lane. Those rotations are different for each lane. The number of rotations per lane resulting from the remainder of the division between some fixes values and the length of the lanes. These fix values are given from Keccak specifications. Note that the mapping in P component is different from the standard one. Lane α[0][0] is depicted at the center of the 2D array since x, y parameters takes the values with the sequence (3,4,0,1 and 2).

The same mapping is used in Π component. Simple wiring was used instead of logic operations in order to changes the position between the lanes according to the specifications. Also, logic operations (NOT, AND and XOR) between the lanes are used at X component. These operations applied to entire rows of lanes for each row. The standard mapping was used.

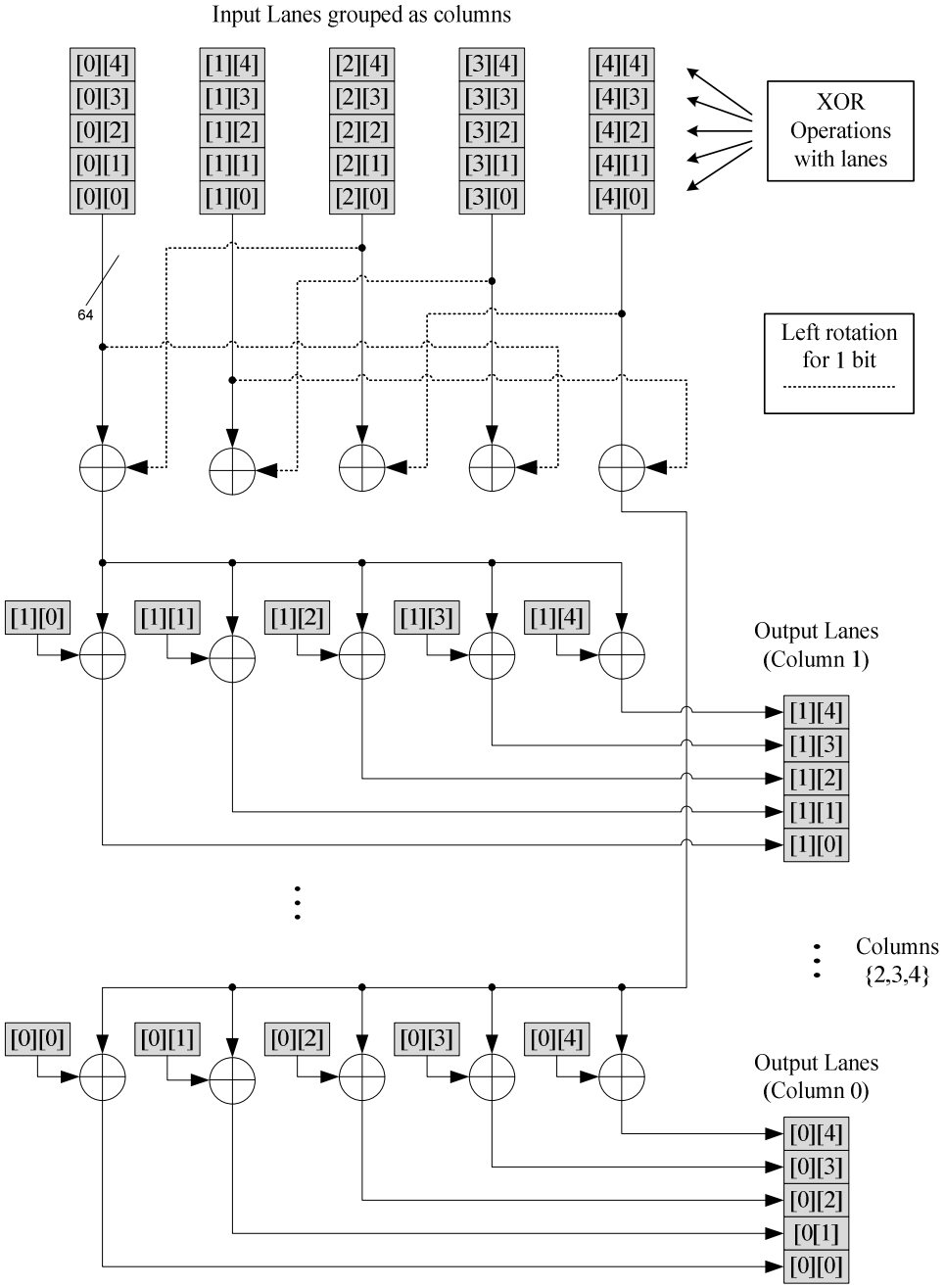

Figure 2. Implementation of Θ component.

In Figure 3 the implementation of X component for a single row is given. Since there are five rows of five lanes, the X component uses 25 NOT, 25 AND and 25 XOR of 64-bit logic gates. The next component, I, implement a XOR operation between the first lane (1599 1536) and the round constant value. These values are fixed and are given from the function specifications. At the end of the round a register is used in order to synchronize the data path. The design needs one clock cycle to operate one round. This means that the hash value will be ready after twenty five clock cycles. The last component is the Truncating component which simply truncates its input data to the hash output desired length. Also applies the inverse procedure of padding, as described previously.

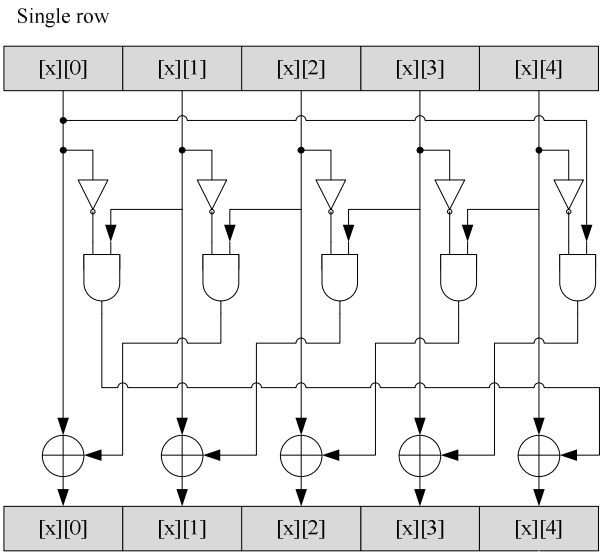

Figure 3. X component for a single row.

IV. FPGA-BASED DESIGN CONSIDATIONS

Some design techniques were proposed in the previous architecture in order to achieve better time performance. So, in this chapter the different implementations according to the proposed techniques are presented. The basic objective was a comparison between the different implementations in order to examine their efficiency in Keccak implementation. For the implementations a device from Virtex-5 family was used.

The first technique uses DSPs slices (the implementation is symbolized as Kec_DSP). With this technique better time performance of the FPGA implementations were achieved. The major scope was to examine if this method is appropriate for high throughputs in Keccak implementation. The component's name is DSP48E [6] and it has an I/O interface with two inputs of 48-bit each of them, one output and also some control signals. The DSP48E could be useful for many digital signal procedures. However, in the proposed implementation the DSP48E was used in order to implement the logic operations which are used by the algorithm. As it is mentioned in previous, the logic operations (NOT, AND and XOR operations) are used in Θ and X components. So, the VHDL code was modified appropriately in order to implement these functions through DSPs components.

In addition, the DSP48E provide the capability for internal pipelining stages. So, it is possible to choose the pipelining mode from one to four pipelining levels. After some experiments three pipelining levels were used. This is because the best tradeoff between area and time performance was achieved. This implementation is symbolized as Kec_DSP2.

The next technique which is used is inner pipelining between the rounds of the hash function. This technique is utilized in order to reduce the critical path; so a better time performance it will have achieved by the design. After some measurements a positive edge triggered register between Π and X components is used. Now the design has two pipelining levels. This implementation is symbolized as Kec_Pip.

In the last design some instructions by the Xilinx's [9] are adopted for high performance designs. According to Xilinx rules a FPGA design should used only synchronized control signals. In addition, the control signals such as reset signals should avoided (when is possible). Resets signals were used only were really necessary and also all control signals into counters and registers were synchronized. The implementation in this case is symbolized as Kec_Sync.

V. SYNTHESIS RESULTS

The designs were implemented and synthesized with Xilinx ISE 11 tool. The target FPGA device was XC5VSX240T-2FF1738. The measurements were focused in design output throughput and the consumed FPGA area resources. In Table 1 the measurements results from the proposed architectures are shown. In DSP implementations were used 201 DSP48E components. For the throughput measurement a 512-bit data output was used.

TABLE I. EXPERIMENTAL RESULTS

Device: XC5VSX240T - 2FF1738				
Architectures	Slices	Clock Cycles	Freq (MHz)	Throughput (Gbps)
Keccak	2573	25	285	5,70
Kec_DSP	3176	25	58	1,16
Kec_DSP2	3009	384	334	0,43
Kec_Pip	2326	49	306	3,12
Kec_Sync	2517	25	278	5,56

In first row the measurements the FPGA resources of the hardware implementation (Figure 1) are shown. This design achieves a maximum operating frequency at 258 MHz. With output of 512-bit per 25 clock cycles gives a throughput of

5,70 Gbps which is the highest of all proposed architectures. The second row shows the results for the time performance for the Kec_DSP implementation. The operating frequency is decreased up to 58 MHz. In this case a reduction of throughput up to 1,16 Gbps was measured. Kec_DSP2 is the implementation were used the inner pipelining technique of three levels through the DSP components. This increases the operating frequency at 334 MHz but also increases the clock cycles of execution algorithm. With 512-bit output per 384 clock cycles we have the lowest throughput from all architectures at 0.43 Gbps. The Kec_pip implementation achieves a high operational frequency of 306 MHz but also doubles the execution clock cycles. As a result a reduction of throughput at 3,12 Gbps was achieved. Finally, the Kec_Sync architecture achieves a frequency at 278 MHz however reduce the execution clock cycles. So in synchronous implementation has been achieved the second best throughput at 5,56 Gbps. As a result for algorithms with very low complexity such as Keccak neither the technique with DSP component nor the technique with synchronous control signals achieves high time performance.

In terms of area consumption Keccak design requires 2573 slices. On architectures that use DSP components the hardware requirements are increased at 3176 slices for Kec_DSP design and 3009 slices for Kec_DSP2 design. This is explained by the fact that DSP48E component has 48-bit inputs length and each of lane has a size of 64-bit. So to use DSP48E components each of lane should be split into two smaller component that will fit in DSP48E component. As a result it is increased the circuit complexity and the hardware requirements are increased. The other two architectures have small reductions in FPGA resources. The Kec_Pip design uses 2326 slices. Finally, the Kec_Sync design also achieves a low area reduction. This decrement is explained by the fact that the circuit complexity is reduced since less control signals are used.

In the Table II, the measurements results from other hardware implementation [10-15] in comparison with the better proposed architecture are given. For the throughput measurements some of the implementations use the input throughput. For the calculation of internal throughput the length of input Block in bits is used instead of the data output length that used in the proposed implementations.

The design in [10] achieves an operating frequency at 265 MHz with low area consumption. But needs 2573 clock cycles so, the throughput is very low at 0,07 Gbps. Similarly, the implementation in [11] has a small throughput at 0,08 Gbps because needs 1896 clock cycles. In terms of hardware resources has the lowest value with 188 slices and with 285 MHz operating frequency. Then in [12] use a block size of 1088-bit for the internal data rate computation. So this implementation achieves an input throughput at 10,25 Gbps in 24 clock cycles with 231 MHz operating frequency. With the same block size the implementation [13] has an input throughput of 6,07 Gbps with 143 MHz operating frequency. Also, this design has hardware requirements of 2043 slices. The next design [14] is an ASIC implementation that uses a 0,13 μm CMOS technology with CMR8SF-RTV standard cell library. This design achieves an input throughput up to 10,67 Gbps with block size of 1024-bit and 250 MHz operating frequency. The last design [15] is also an ASIC implementation on a standard 0,13 μm CMOS technology. The goal of this implementation was a lightweight version of Keccak because of it has a low input throughput at 4,4 Mbps with block size of 1088-bit and 0,1 MHz operating frequency.

TABLE II. HARDWARE PERFORMANCE COMPARISONS

Architectures	Technology	Slices	Clock Cycles	Freq (MHz)	Throughput (Gbps)
Proposed	V5	2573	25	285	5,70
[10]	V5	444	5160	265	0,07
[11]	V6	188	1896	285	0,08
[12]	V6	1043	24	231	10,25
[13]	V4	2024	25	143	6,07
[14]	0,13 μm	-	24	250	10,67
[15]	0,13 μm	-	24	0,1	4,4 Mbps

As a result two are the main results of the above measurements and comparisons. Firstly, that the proposed implementation (Keccak) outperforms all the previous published works in term of throughput and secondly, in the designs with low complexity (like Keccak hash function) the usage of the DSP block has negative effect in terms of throughput and FPGA resources.

VI. CONCLUSION

Keccak hash function hardware implementations are described in this paper which has been implemented by means of a Virtex-5 FPGA device. The design approaches include the usage of DSP blocks which result in the minimal use of traditional user logic such as look-up tables (LUTs), pipelining which result the increase of time performance and a combination of the two techniques. The experimental results prove that the Keccak implementations is a very good solution for applications with high throughput demands and also the use of DSP block is a inefficient method in design with low complexity demands.

REFERENCES

[1] SHA-1 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-1, 1995, available on line at www.itl.nist.gov/fipspubs/ fip180-1.htm

[2] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, "Finding collisions in the full SHA-1," 25th Annual Int. Cryptology Conference-CRYPTO 2005, Santa Barbara, California, USA, August 14-18, 2005, LNCS 3621, 2005.

[3] Secure Hash Standard (SHS), National Institute of Standards and Technology (NIST), FIPS PUB 180-3, 2008, available on line at http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[4] National Institute of Standard and Technology (NIST), "Cryptographic hash algorithm competition", 2007, available on line at http://www.nist.gov/itl/csd/ct/hash_competition.cfm

[5] Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche, "The Keccak SHA-3 submission", Submission to NIST (Round 3), 2011, available on line at http://keccak.noekeon.org/Keccak-submission-3.pdf

[6] Virtex-5 FPGA XtremeDSP Design Considerations, available on line at http://www.xilinx.com/support/documentation/user_guides/ug193.pdf

[7] Xilinx, Virtex-5 FPGAs, available on line at http://www.xilinx.com/support/documentation/virtex-5.htm

[8] Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche, "On the indifferentiability of the sponge construction", Advances in Cryptology − Eurocrypt 2008, Lecture Notes in Computer Science, vol. 4965, Springer, 2008.

[9] Brian Philofsky, "HDL Coding and design practices for improving Virtex-5 utilization, performance, and power", Xcell Jounal, Issue 59, Four Quarter 2006, Xilinx.

[10] Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. "Keccak sponge function family main document", version 1.2, April 2009, available on line at http://keccak.noekeon.org/.

[11] Stéphanie Kerckhof, François Durvaux, Nicolas Veyrat-Charvillon, Francesco Regazzoni, Guerric Meurice de Dormale, François-Xavier Standaert, "Compact FPGA implementations of the five SHA-3 finalists", 10th IFIP Smart Card Research and Advanced Applications 2011 (CARDIS 2011), Leuven, Belgium, pp. 217-233, September 14-16, 2011.

[12] Kashif Latif, M Muzaffar Rao, Arshad Aziz and Athar Mahboob, "Efficient hardware implementations and hardware performance evaluation of SHA-3 finalists", NIST Third SHA-3 Candidate Conference, Washington D.C., March 22-23, 2012.

[13] Abdulkadir Akin, Aydin Aysu, Onur Can Ulusel, Erkay Savaş, "Efficient hardware implementations of high throughput SHA-3 candidates Keccak, Luffa and Blue Midnight Wish for single- and multi-message hashing", NIST 2nd SHA-3 Candidate Conference, Santa Barbara, August 23-24, 2010.

[14] Xu Guo, Meeta Srivastav, Sinan Huang, Dinesh Ganta, Michael B. Henry, Leyla Nazhandali and Patrick Schaumont, "Silicon implementation of SHA-3 finalists: BLAKE, Grostl, JH, Keccak and Skein", ECRYPT II Hash Workshop 2011, Tallinn, Estonia, 19-20 May 2011.

[15] Elif Bilge Kavun and Tolga Yalcin, "A lightweight implementation of Keccak hash function for radio-frequency identification applications", Radio Frequency Identification: Security and Privacy Issues, Lecture Notes in Computer Science, 2010, Volume 6370/2010, 258-269.