Journal of Cryptographic Engineering (2024) 14:193–205
https://doi.org/10.1007/s13389-023-00334-0

REGULAR PAPER

Hardware acceleration design of the SHA-3 for high throughput and low area on FPGA

Argyrios Sideris[1] · Theodora Sanida[1] · Minas Dasygenis[1]

Received: 7 May 2023 / Accepted: 4 August 2023 / Published online: 30 August 2023

Abstract
In sensitive communications, the cryptographic hash function plays a crucial role, including in the military, healthcare, and banking, ensuring secure transmission by verifying data integrity and carrying out other vital tasks. Compared to other cryptographic hash algorithms, such as SHA-1 and SHA-2, the Keccak hash function (SHA-3) boasts superior hardware performance and is more resilient to modern cryptanalysis techniques. Nonetheless, hardware performance enhancements, such as boosting speed or reducing area usage, are constantly required. This research focuses on increasing the Keccak hash algorithm's throughput rate by introducing a novel architecture that reduces the total number of clock cycles required to obtain the result of a hash function. Additionally, the new simplified structure of the round constant (RC) generator design assures a reasonably low area and achieves the highest throughput and efficiency. Thus, when implemented, it achieved the highest throughput of 19.515 Gbps, 24.428 Gbps, 33.393 Gbps, and 36.358 Gbps on FPGA devices with the Virtex-5, Artix-7, Virtex-6, and Virtex-7, respectively. Finally, our approach is compared to recently published designs.

Keywords Hardware acceleration · Hardware evaluation · Secured hash algorithm-3 (SHA-3) · Cryptography · Field-programmable gate array (FPGA) · Round constant (RC) generator

1 Introduction

In the digital era, a cryptographic protocol is a collection of rules and processes for securing communication between two or more parties. It is necessary to guarantee that data transported across a network or saved in a database is secured against unauthorized access, alteration, or theft. Without secure communication, sensitive information may be compromised, resulting in monetary loss or reputational harm. Encryption, digital signatures, and cryptographic hash functions are among the technologies that assure communication security [1, 2].

✉ Argyrios Sideris
asideris@uowm.gr

Theodora Sanida
thsanida@uowm.gr

Minas Dasygenis
mdasyg@ieee.org

1 Department of Electrical and Computer Engineering, University of Western Macedonia, ZEP Campus, 50100 Kozani, Greece

In the domain of cryptography, a hash function is a mathematical procedure that accepts input data and produces an output of a fixed size (mentioned as a hash). The hash function is constructed, so it is almost impossible to identify authentic information from the hash result. This characteristic is helpful for data integrity verification, authentication, the calculation of digital signatures, and password storage. The capacity to maintain data integrity is one of the primary benefits of cryptographic hash algorithms. A hash value is unique to a single input; any change to the information will result in a new hash value. By comparing the hash values of the original and updated data, it is feasible to discover any illegal changes. This characteristic makes hash functions suitable for assuring data integrity in electronic transactions, such as online banking, e-commerce, and government activities [3, 4].

Keccak [5] architecture of the Secure Hash Algorithm-3 (SHA-3) has become popular among hash techniques and has replaced the previously used SHA-1 [6] and SHA-2 [7–9] techniques. SHA-3 has better benefits than its predecessors, such as diversity and reusability; methods have been explored since its introduction in 2012 to improve its parameters for specific applications and hardware devices. The hardware performance of SHA-3 is preferred over software due to its superior power, speed, and throughput implementation. Field-programmable gate array (FPGA) is preferred over application-specific integrated circuits (ASIC) as a hardware performance platform due to its lower price and shorter development time [10–14].

The advantage of SHA-3 in FPGA is speed over previous SHA algorithms in hardware implementations, and it is designed to perform well on various hardware platforms. Implementing SHA-3 in an FPGA allows for algorithm customization and reconfigurability flexibility. Since FPGAs can be designed to consume less power [15, 16] than traditional processors, which makes them ideal for implementing cryptographic functions like SHA-3. Finally, FPGAs can significantly increase the throughput of SHA-3 calculations [17]. As a consequence of this, several strategies have been suggested in order to implement the Keccak algorithm effectively. These approaches either concentrate on reducing the energy consumed, maximizing the area consumed, or enhancing the processing speed.

Overall, the contributions presented in this manuscript are summarised as follows:

- We present a novel architectural design optimisation strategy based on unrolling the SHA-3 algorithm. Our approach enhances and maximises the throughput and efficiency performance metrics of FPGA devices, making it an ideal solution for many applications.
- We propose a new simplified structure of the RC generator to achieve improved performance (throughput/efficiency) while effectively reducing hardware resources in the area. The new simplified structure RC generator only consists of 7 bits instead of 64, thus reducing the computation in the Iota (ι) step where the number of required XORs reduces to 7.

The remains of the paper are organised as follows: In Sect. 2, we furnish the relevant studies similar to our research. In Sect. 3, we define the overall SHA-3 architecture. Section 4 describes our new proposed hardware implementation of the SHA-3 algorithm on FPGA boards. In Sect. 5, we present the experimental outcomes of our study. In Sect. 6, we discuss the results of our method and the comparisons with other relevant research. Finally, Sect. 7 summarises our research's findings and future work.

2 Related work

The cryptographic community has conducted significant research on optimising models, architectures, and strategies for SHA-3 in FPGA devices [18–20]. All these architectures aim to increase the throughput, efficiency, and frequency while attempting to decrease the area and power consumption in the FPGA [21–27]. Nevertheless, there is a pressing need to increase throughput and efficiency with area reduction performance metrics. In this section, we present research endeavours similar to ours.

In [28], presented a method for Keccak architectures for output sizes of 256 and 512. The RC is stored in a distributed ROM of 24×64 bits. The Virtex-5 architecture for output size 256 needs 1217 slices and a 277 MHz clock and achieves 12.56 Gbps throughput, and the Virtex-7 needs 998 slices and a 300 MHz clock and reaches 13.60 Gbps throughput. The Virtex-5 architecture for output size 512 needs 1200 slices and a 270 MHz clock and achieves 6.48 Gbps throughput, and the Virtex-7 needs 983 slices, a 298.68 MHz clock and reaches 7.17 Gbps throughput. However, this architecture produced poor frequency and throughput.

Paul and Shukla [29] presented two Keccak architectures for output size 256 and an RC method with a count generator to fetch the RC with 64 bits from onboard read-only memory (ROM). The first architecture needs 4188 slices, a 390.53 MHz clock, and achieves 16.492 Gbps throughput. The second architecture needs 7139 slices, a 234.97 MHz clock, and reaches 19.99 Gbps throughput. However, these architectures produced poor frequency and increased area.

Wong et al. [30] presented a method to decrease the area required for ROM by reducing the bit length from 64 to 8 and showing five different Keccak architectures for output size 512. The first architecture needs 871 slices and a 153 MHz clock, achieving 3.68 Gbps throughput and 4.22 Mbps/Slices. The second architecture needs 1393 slices and a 335 MHz clock, reaching 8.04 Gbps throughput and 5.77 Mbps/Slices. The third architecture needs 2145 slices and a 45 MHz clock, earning 2.16 Gbps throughput and 1.00 Mbps/Slices. The fourth architecture needs 1416 slices and an 85 MHz clock and attains 4.08 Gbps throughput and 2.88 Mbps/Slices. The fifth architecture needs 1406 slices and a 344 MHz clock, gaining 16.51 Gbps throughput and 11.47 Mbps/Slices. Even if the total area occupied was not very large, more than the highest frequency attained was needed to be satisfactory.

The unrolling approach, which reduces the total number of clock cycles with an additional round operation, is implemented in Virtex-5 in [31] for output size 256 and reaches 5.38 Gbps throughput. In [32] also decreased the number of clock cycles for all output sizes by using the unrolling approach with Virtex-5 and Virtex-6. Despite this, the frequency and throughput delivered by this design could have been better.

In [33], a basic architecture of Keccak was suggested for Virtex-7 FPGA with an output size of 512 bits. A distributed ROM with a dimension of 24×64 bits was used to store the round constants (RC). The architecture operated 1454 slices and utilized a clock frequency of 374.035 MHz. This design earned a throughput of 7.979 Gbps and an efficiency rate of

5.49 Mbps/Slices. However, the area and throughput were both affected negatively by this implementation.

Assad et al. [34] suggested three Keccak implementations in Virtex-5 and Virtex-6 FPGA. The focus was on all output sizes. It is worth noting that the RC required for the Keccak implementation were stored in a ROM of 24×64 bits. The basic implementation using Virtex-5 for output size 512 required 935 slices and operated at a clock frequency of 338.409 MHz. This design achieved a throughput of 8.12 Gbps and a rate of 8.68 Mbps/Slices. The basic implementation using Virtex-6 for output size 512 required 1019 slices and operated at a clock frequency of 376.081 MHz. This configuration achieved a higher throughput of 9.02 Gbps, with a rate of 8.85 Mbps/Slices. Nevertheless, the area and efficiency were affected negatively by this design.

In [35], a basic implementation of Keccak was proposed for Virtex-5 FPGA with an output size of 512 bits. The round constants (RC) were stored in a distributed ROM of 24×64 bits. The implementation employed 1680 slices in the Virtex-5 FPGA and operated at a clock frequency of 387 MHz. This design achieved a throughput of 8.06 Gbps and an efficiency rate of 4.91 Mbps/Slices. Despite this, the area and efficiency delivered by this architecture could have been better.

Comprehensive examination and analysis of the above methodologies and their effects on the performance of the Keccak architecture indicated a demand for an improved architecture that yields high throughput combined with the low area. Hence, effectively handling RC is essential to attaining high throughput. Therefore, we propose a new RC value generation technique with a minimised structural design where the number of required XORs is reduced to 7 instead of 64. This approach has resulted in a substantial reduction in the area while simultaneously increasing a sizeable throughput. The presented Keccak architecture was tested and verified using the existing test vectors.

3 The SHA-3 architecture

In 2012, the National Institute of Standards and Technology ($NIST$) maintained a competition to establish a new standard hash function that would complement existing SHA-1 and SHA-2 standards. The objective was to choose a function that would be secure, efficient, and resistant to attacks such as collision and preimage attacks [36]. The winner of the competition was the Keccak hash function. Unlike the previous SHA standards, SHA-3 is founded on the sponge functions ($absorb/squeeze$) as presented in Fig. 1.

The sponge function is based on a state matrix of "$b = r + c$" bits, where "b" denotes the block size, "r" indicates the bit rate of the sponge function, and "c" defines the capacity. So, this state matrix starts with zero values once it is initialized for the first time. The Keccak hash algorithm ensures the

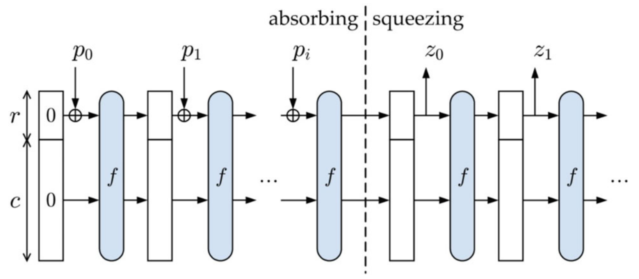

Fig. 1 The ($absorb/squeeze$) sponge structure of the SHA-3 hash function

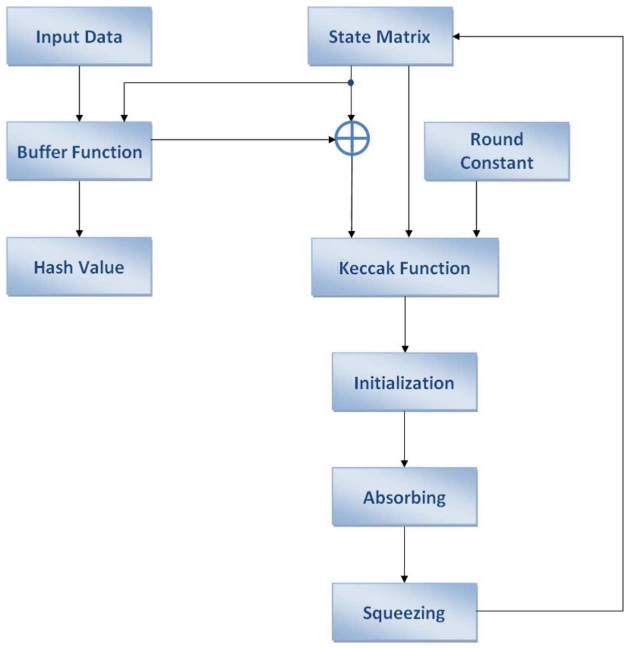

Fig. 2 The block diagram of the SHA-3 hash function

state C as a three-dimensional matrix with the dimensions $5 \times 5 \times (word - size)$.

An input message is padded, adding bits to the message so that its total size becomes a multiple of a fixed number of bits, denoted as "r". Once the message has been padded, it is separated into blocks of equal length, denoted as "Pi". In the absorbing step, "r" bits XOR with each block and permutation function "f". The "f" function is the central part of the processing of 24 rounds and consists of distinct steps, including i) "Theta ($\boldsymbol{\theta}$)", ii) "Rho ($\boldsymbol{\rho}$)", iii) "P$_i$ ($\boldsymbol{\pi}$)", iv) "Chi ($\boldsymbol{\chi}$)" and v) "Iota ($\boldsymbol{\iota}$)", each of which performs a specific operation on a 1600-bit state matrix denoted as A [37]. The block chart of the SHA-3 is illustrated in Fig. 2.

The steps "Theta ($\boldsymbol{\theta}$)", "Rho ($\boldsymbol{\rho}$)", "P$_i$ ($\boldsymbol{\pi}$)", "Chi ($\boldsymbol{\chi}$)" and "Iota ($\boldsymbol{\iota}$)" are shown in Eqs. (1)–(5). In particular, Eq. (1) refers to the computations performed in the "Theta ($\boldsymbol{\theta}$)" step. This step involves manipulating a two-dimensional array of size (5×5), where $C[i]$ and $D[i]$ are one-dimensional arrays representing the lanes, and $A[i, j]$ denotes the slices. The

Table 1 The standard round constants RC_i generator in Iota (ι) step of the SHA-3 algorithm

RC_0	0x0000000000000001	RC_{12}	0x000000008000808B
RC_1	0x0000000000008082	RC_{13}	0x800000000000008B
RC_2	0x800000000000808A	RC_{14}	0x8000000000008089
RC_3	0x8000000080008000	RC_{15}	0x8000000000008003
RC_4	0x000000000000808B	RC_{16}	0x8000000000008002
RC_5	0x0000000080000001	RC_{17}	0x8000000000000080
RC_6	0x8000000080008081	RC_{18}	0x000000000000800A
RC_7	0x8000000000008009	RC_{19}	0x800000008000000A
RC_8	0x000000000000008A	RC_{20}	0x8000000080008081
RC_9	0x0000000000000088	RC_{21}	0x8000000000008080
RC_{10}	0x0000000080008009	RC_{22}	0x0000000080000001
RC_{11}	0x000000008000000A	RC_{23}	0x8000000080008008

"Rho ($\boldsymbol{\rho}$)" and "Chi ($\boldsymbol{\chi}$)" steps compute the $B[i, j]$ array from the state matrix $A[i, j]$. During the "Chi ($\boldsymbol{\chi}$)" step, the value of $A[i, j]$ is recalculated in accordance with the Equation that is shown in (3). Finally, the "Iota ($\boldsymbol{\iota}$)" step involves adding a constant value, denoted as $RC(i)$, to the first element of the $A[0, 0]$ array.

The "Theta ($\boldsymbol{\theta}$)" step is the first step of the Keccak-f permutation. It involves i) a parity computation, ii) a rotation of one place, and iii) a bitwise XOR operation. The parity computation takes the XOR of every 5-bits in a 25-bit row, resulting in a 5-bit output. The rotation involves shifting the bits of each row by a fixed amount, which varies for each row. The bitwise XOR operation combines the output of the parity computation with the rotated row to produce a new row.

Step Theta (θ):

$$\begin{aligned}\mathrm{C[i]} &= \mathrm{A[i,0]\,XOR\,A[i,1]\,XOR\,A[i,2]}\\ &\quad \mathrm{XOR\,A[i,3]\,XOR\,A[i,4],}\\ &\quad\ \mathrm{i} \le 4\\ \mathrm{D[i]} &= \mathrm{C[i-1]\,XOR\,ROTATE(C[i+1],1),}\\ &\quad\ \mathrm{i} \le 4\\ \mathrm{A'[i,j]} &= \mathrm{A[i,j]\,XOR\,D[i],}\\ &\quad\ \mathrm{i} \le 4 \end{aligned} \tag{1}$$

The "Rho ($\boldsymbol{\rho}$)" step is a rotation step that involves rotating each bit of the state by an offset that hinges on the word assignment. The "P$_i$ ($\boldsymbol{\pi}$)" step is a permutation step that involves rearranging the words of the state. So, the state array A is also used to calculate a serviceable 5×5 array B in the following two steps. Interestingly, a bit stream consisting of w bits is referred to by the array $B[i, j]$.

Step Rho (ρ):

$$A[i, j] = \mathrm{ROTATE}\left(A'[i, j], r[i, j]\right), \quad [i, j] \le 4 \tag{2}$$

Step Pi (π):

$$\mathrm{B[j, 2i+3j] = A[i, j]}, \quad [i, j] \le 4 \tag{3}$$

The "Chi ($\boldsymbol{\chi}$)" step is a bitwise logic operation that involves performing a bitwise XOR, NOT, and AND operation on the bits of the state.

Step Chi (χ):

$$A[i, j] = B[i, j]\,\mathrm{XOR}((-B[i+1, j])AND B[i+2, j]),\quad [i, j] \le 4 \tag{4}$$

The final step, "Iota ($\boldsymbol{\iota}$)", involves adding a round constantly to a single bit of the state. The round constants are produced by the RC generator that is used in the "Iota ($\boldsymbol{\iota}$)" step. The RC_i function is present in Table 1 and comprises 24 unique permutation values that allocate 64-bit data to the SHA-3 operation [36].

Step Iota (ι):

$$A[0, 0] = A[0, 0]\,\mathrm{XOR}\,RC[i] \tag{5}$$

The NIST has determined four forms of the SHA-3 for generating hash values from a message M of any length and an output length size d, as presented in Table 2.

Several hash function applications prefer smaller output sizes, specifically, those not using them for security [38]. The larger the output length size, the stronger higher the security against assaults of the hash function. Nevertheless, the larger output length size also means a slower hash function operation, as more processing power is required to produce the hash value. Thus, this work presented a structure that permits generating all four probable output lengths.

Table 2 The SHA-3 algorithm in its four different forms

Keccak	Block size (b)	Rate (r)	Capacity (c)	Output size (d)	Level security
224(M)	1600	1152	448	224	112
256(M)	1600	1088	512	256	128
384(M)	1600	832	768	384	192
512(M)	1600	576	1024	512	256

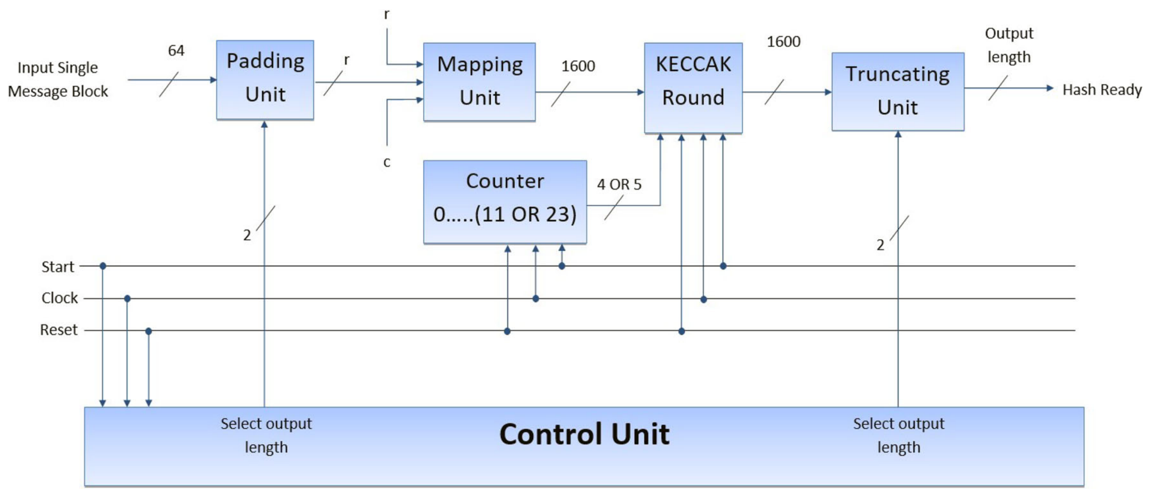

Fig. 3 Proposed optimization architectural system of the SHA-3

4 Proposed optimization architectural system

This section analyses the design components we implemented for all output lengths (576, 832, 1088, 1152) of the SHA-3 algorithm. The primary target of our work is to achieve higher throughput (Gbps) by reducing the area in our system. This target is achieved with the new simplified structure of the proposed RC generator, which eliminates the need for further hardware resources in the area and provides higher performance.

4.1 The architectural design of the SHA-3 (Keccak)

Our system architecture is presented in Fig. 3. The architecture comprises (i) padding, (ii) mapping, (iii) the Keccak round, (iv) truncate, (iv) control and (v) counter. The Keccak round is at the core of the architectural design. The responsibility for controlling, synchronizing, and communicating the flow of data inside our system lies with the control unit. The input message data is 64-bits. The values for the select output length are shown in Table 3.

4.2 Padding, mapping and truncating unit

The padding scheme of the SHA-3 (Keccak) for the input message is shown in Fig. 4. Initially, the input message of

Table 3 The four different values for the select output length of the SHA-3 algorithm

Value	00	01	10	11
Hash output	224	256	384	512

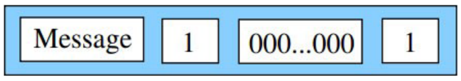

Fig. 4 The SHA-3 hashing algorithm's padding scheme

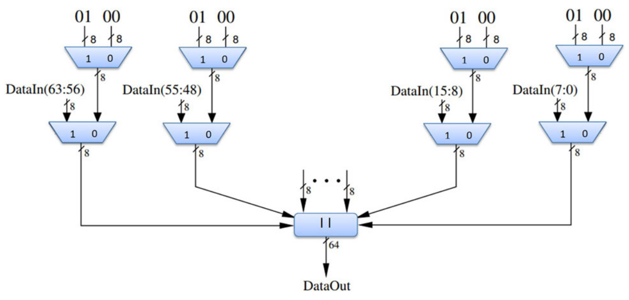

Fig. 5 Padding block diagram of the SHA-3

64-bits is appended with "1", then followed by bits of "0", then appended with "1" so that the total message size is a multiple of "r" bits (576, 832, 1088, 1152) [39].

The basic padding block diagram of the SHA-3 is illustrated in Fig. 5 and consists of multiplexers 2 to 1 for a 64-bits input message. The padding unit is shown in Fig. 6. The

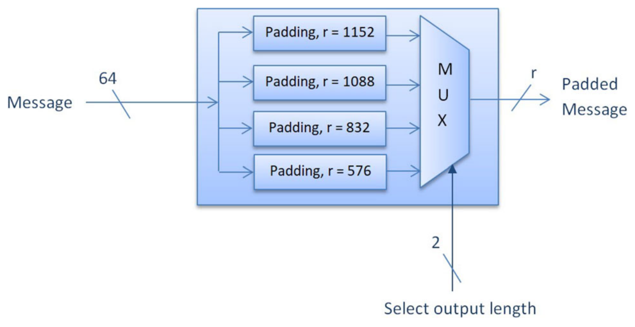

Fig. 6 Padding Unit of the SHA-3

padding unit consists of one multiplexer, 4 to 1. If 224-bits are selected as the output length, then the padding scheme for $r = 1152$ will be executed. The Padded input message (Pad) "r" bits are entered in the mapping unit and XOR with the initially of the "r" bits. After, appended the result with the initially of the "c" bits [36].

Data transformation is required, as shown in Eq. (6). The truncating unit, according to Eq. (6), cuts the digits of state depending on the output length selected (576, 832, 1088, 1152) and consists of one multiplexer 4 to 1.

$$\text{State}[x, y, z] = ((\text{Pad}r\text{XOR}r)||c) \times [64 \times (5y + x) + z] \tag{6}$$

4.3 The Keccak round architecture

In this study, one of our primary goals has been to reduce the total number of clock cycles using an unrolling strategy, ensuring a reasonably low area. The base architecture of the permutation rounds block is seen in Fig. 7, with the counter ranging from 0 to 23, indicating no attempt to minimize the total number of clock cycles. As a result, as illustrated in Fig. 7, divide the total number of the counter by half to reduce the total number of clock cycles. This part of our methodology is shown in Fig. 8, using a total of just 12 clock cycles.

The unrolling strategy refers to a technique used to optimize the implementation of an algorithm by reducing loop overhead. In the base implementation of SHA-3, as shown in Fig. 7, the computation involves a single block of transformations ($\boldsymbol{\theta} \rightarrow \boldsymbol{\rho} \rightarrow \boldsymbol{\pi} \rightarrow \boldsymbol{\chi} \rightarrow \boldsymbol{\iota}$). However, the unrolling strategy aims to further enhance the algorithm's performance by executing multiple transformation blocks.

This work applied an unrolling factor of 2, as depicted in Fig. 8. This means that an additional block of transformations was included inside the Keccak Round module. With the unrolling factor of 2, two rounds of transformations ($\boldsymbol{\theta} \rightarrow \boldsymbol{\rho} \rightarrow \boldsymbol{\pi} \rightarrow \boldsymbol{\chi} \rightarrow \boldsymbol{\iota} \rightarrow \boldsymbol{\theta} \rightarrow \boldsymbol{\rho} \rightarrow \boldsymbol{\pi} \rightarrow \boldsymbol{\chi} \rightarrow \boldsymbol{\iota}$) are performed within a single clock cycle. By incorporating this unrolling strategy, the number of clock cycles required to complete the entire SHA-3 algorithm is halved. The standard SHA-3 algorithm comprises 24 rounds, and with the unrolling factor of 2, it will now take only 12 clock cycles to accomplish these 24 rounds.

This acceleration by unrolling strategy results in computations being completed in halved clock cycles, thus reducing overall execution time and to improved performance for the SHA-3 algorithm implementation, making it well-suited for various FPGA devices.

4.4 New simplified structure of the RC generator

Our research proposes a new simplified structure for the RC generator that significantly improves the algorithm's performance while reducing hardware resources. The RC generator is a crucial component of the SHA-3 algorithm. Its primary function is to produce a sequence of pseudo-random bits used to encrypt the input data. The existing RC generator consists of 24 sets of 64-bits, which results in many computations in the "Iota ($\boldsymbol{\iota}$)" step of the SHA-3 algorithm. This step needs a large number of XOR operations to be executed, which can decrease performance and efficiency, especially in FPGA devices with limited resources.

To overcome this issue, we have designed a new simplified structure for the RC generator that only consists of 7-bits [36, 40]. By reducing the number of bits in the generator, we effectively reduce the computations required in the Iota step, resulting in improved performance and efficiency. Reducing the total number of bits also reduces the hardware resources required for the RC generator, resulting in a more compact design ideal for FPGA devices with limited resources.

The "Iota ($\boldsymbol{\iota}$)" step is to modify some of the bits of in state array A, as shown in Eq. (7).

$$A'[x, y, z] = A[x, y, z]\,\text{XOR}\; RC\,[i_r] \tag{7}$$

According to the specifications of SHA-3, the RC are given by Eq. (8),

$$\text{RC}\,[i_\text{r}]\,[0][0]\left[2^j - 1\right] = \text{rc}\,[\,j + 7i_\text{r}]\,,\ \text{for all}\quad 0 \leq j \leq \ell \tag{8}$$

and all other values of $RC[i_\text{r}][x][y][z]$ are zero. From Eq. (8), it follows that only 7 of the 64 bits can have the value 1. Table 4 presents the specific positions for the 7 bits where ℓ = 6, by the specifications of the SHA-3.

So only those 7 of the 64-bits are fundamental round constants and appear in specific places with non-zero bits while the other positions are zero. The specific bit positions that carry the value "1" are 0,1,3,7,15,31 and 63, with the rest being "0". An example of the simplified structure for RC[3] of Table 5 is shown in Table 6. Thus, seven specific bits can be set for the XOR gate in state array A.

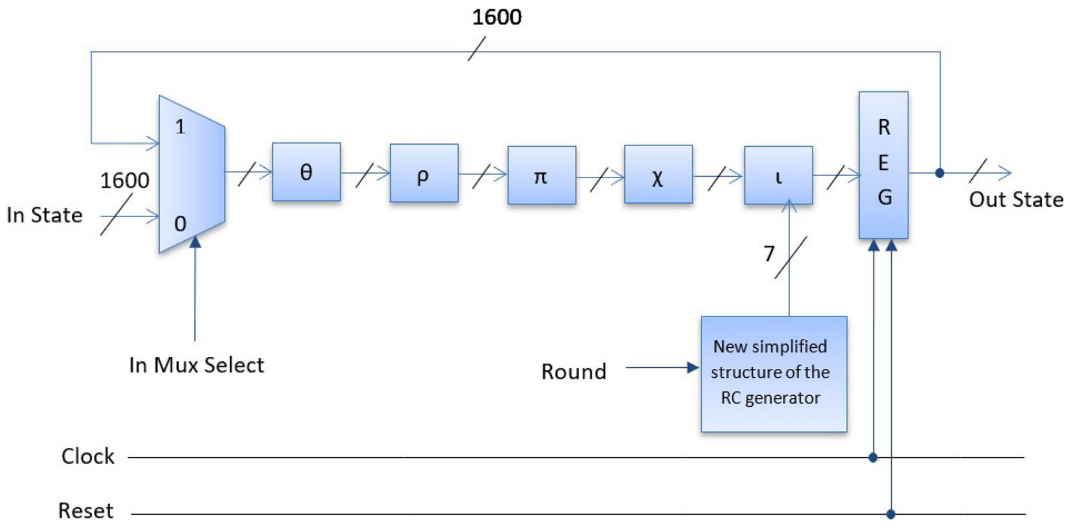

Fig. 7 Keccak round with 24 clock cycles with a new simplified structure of the RC generator

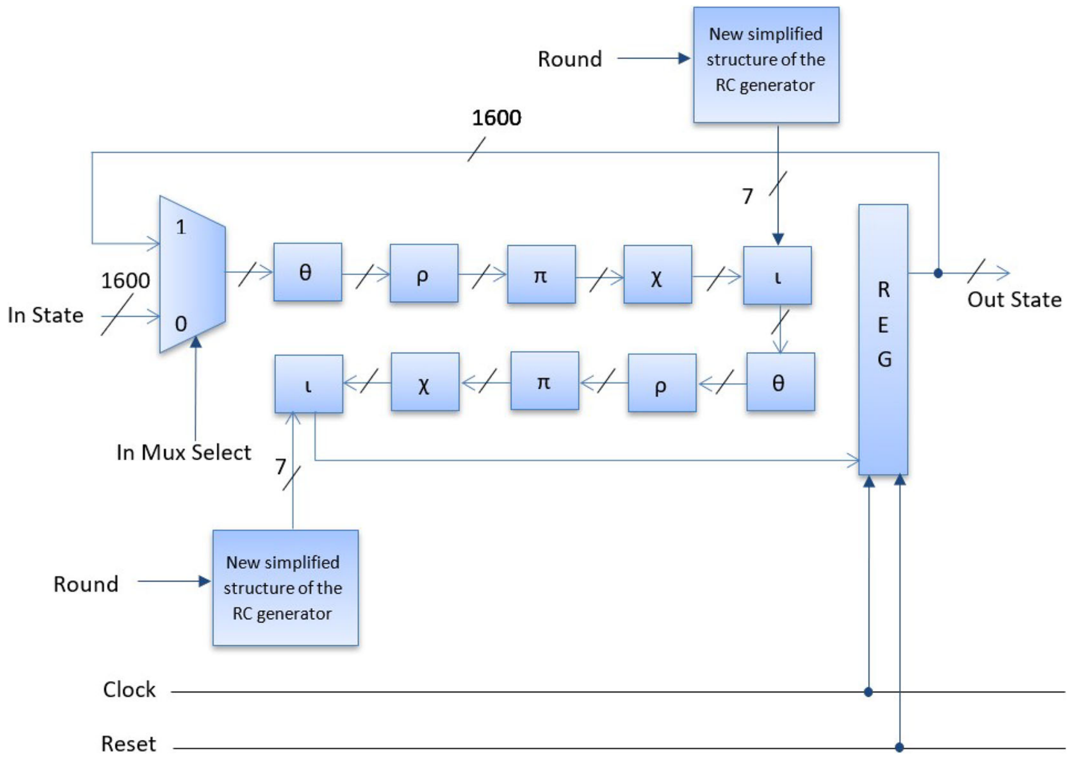
Round
New simplified structure of the RC generator
1600
7
1
0
1600
In State
θ
ρ
π
χ
ι
R E G
Out State
ι
χ
π
ρ
θ
In Mux Select
7
Round
New simplified structure of the RC generator
Clock
Reset

Fig. 8 Keccak round with 12 clock cycles with a new simplified structure of the RC generator

Table 4 Specific positions for the 7 bits with value 1

j	0	1	2	3	4	5	6
[z]	0	1	3	7	15	31	63

5 Experimental results

In our experiments, we used the Virtex-5, Virtex-6, Virtex-7, and Artix-7 FPGA boards in order to make a fair evaluation between the proposed design and the other existing works while providing a thorough, comprehensive comparison across multiple FPGA platforms for a broader assessment of the design's efficiency and throughput achievements. Xilinx ISE was used to implement the design in the Virtex-5/Virtex-6, and Virtex-7/Artix-7 was used to implement the architecture in the Xilinx Vivado. The implementation has been done with Very High Speed Integrated Circuit Hardware Description Language (VHDL). The designs are simulated

Table 5 The new simplified structure of the round constants RC_i in the Iota (ι) step of the SHA-3 algorithm

RC_0	1000000	RC_{12}	1111110
RC_1	0101100	RC_{13}	1111001
RC_2	0111101	RC_{14}	1011101
RC_3	0000111	RC_{15}	1100101
RC_4	1111100	RC_{16}	0100101
RC_5	1000010	RC_{17}	0001001
RC_6	1001111	RC_{18}	0110100
RC_7	1010101	RC_{19}	0110011
RC_8	0111000	RC_{20}	1001111
RC_9	0011000	RC_{21}	0001101
RC_{10}	1010110	RC_{22}	1000010
RC_{11}	0110010	RC_{23}	0010101

Table 6 Example of the new simplified structure of the RC[3] in Iota (ι) step

Hex	Binary				Places with value 1
8000	1000	0000	0000	0000	0th = 0
					1st = 0
					3rd = 0
					7th = 0
					15th = 1
8000	1000	0000	0000	0000	31st = 1
0000	0000	0000	0000	0000	–
8000	1000	1000	1000	1000	63rd = 1

and confirmed for the whole functionality with valid examples provided by the NIST [41].

5.1 Performance metrics

In order to ensure a fair comparison between the proposed design and other existing works, we used the established definitions of efficiency and throughput [18, 42–44] used in the literature. Maintaining consistency in performance metrics is essential for meaningful comparisons and benchmarking between different designs. Additionally, using the established definition of performance metrics as in other works enables researchers to compare our results with the existing literature and understand the performance advancements achieved by our proposed design.

Throughput symbolises the total number of bits processed per period (time) unit and is defined in Gbps or Mbps. The throughput is computed utilising Eq. (9).

$$\text{Throughput} = \frac{\text{Bmb}}{\text{Ccmb}} \times \text{Max}_f \tag{9}$$

In Eq. (9), Bmb (bits in a message block) are the bitrate size "r" (576, 832, 1088, 1152), Max_f is the maximum clock periodicity frequency, and Ccmb (clock cycles per message block) represent the number of resumption needed for the five special operations: ($\boldsymbol{\theta} \rightarrow \boldsymbol{\rho} \rightarrow \boldsymbol{\pi} \rightarrow \boldsymbol{\chi} \rightarrow \boldsymbol{\iota}$) to generate the hash value. The efficiency is calculated by using Eq. (10).

$$\text{Efficiency} = \frac{\text{Throughput}}{\text{Area}} \tag{10}$$

5.2 Results

The presented architectural design attains high throughput and assures reducing hardware resources in the area for various output lengths required to produce a hash value. The results of the implementation of this architectural design are summarized in Table 7, which shows the maximum frequency and throughput of all output lengths.

As shown in Eq. (10), decreasing the total number of clock cycles and reducing the area increases the throughput, which was our primary objective. Thus, our strategy concentrated on reducing the total number of iterations required to generate a hash value. Additionally, with the new simplified structure, the RC generator has several advantages, such as reduced resource utilization required for a design, faster design time, which reduces project complexity, and higher clock frequencies, which lead to higher performance.

According to the results, the proposed architectural design achieves a maximum throughput of 36.358 Gbps for 224 output length and 18.179 Gbps for 512 output length with an area of 1375 slices. Moreover, Table 7 provides a fair comparison of the achieved outcomes with the recent studies published in the literature.

In addition to the results in Table 8, the efficiency of the proposed design has been evaluated by taking the throughput (in Mbps) and dividing it by the consumption per area (total number of slices). The results of this evaluation are summarized in Table 8 for all output lengths.

6 Result in discussion

Throughput and area are essential data processing metrics, especially for information security. This measure represents the algorithm's efficiency and resistance to cryptanalysis attacks that focus on hardware flaws.

On the Virtex-5 board with 24 total clock cycles, design [35] occupied the highest area of 1680 slices, whereas the proposed design consumed the lowest area of around 868 slices. On the Virtex-6 board, the design [32] occupied the highest area of 1432 slices, whereas the design [30] consumed the lowest area of around 871 slices. Although [30] occupied slightly less area than the proposed design, it resulted in poor efficiency, frequency, and throughput, indicating that the design's area utilization is only one of the factors to be considered in evaluating its performance.

Table 7 The outcomes of the implementation in terms of throughput and comparison

Design	FPGA device	Number of clock cycles	Max frequency (MHz)	Throughput (r = 1152)	Throughput (r = 1088)	Throughput (r = 832)	Throughput (r = 576)
[28]	Virtex-5	24	277	–	12.56	–	6.48
	Virtex-7	24	300	–	13.60	–	7.17
[29]	Artix-7	24	390.53	–	–	–	16.492
		12	234.97	–	–	–	19.99
[30]	Virtex-6	24	153	–	–	–	3.68
		12	344	–	–	–	16.51
[31]	Virtex-5	24	111.732	–	4.67	–	–
		16	84.21	–	5.38	–	–
[32]	Virtex-5	24	326.38	15.66	14.79	11.31	7.83
		12	192.25	18.45	17.43	13.32	9.228
	Virtex-6	24	413.77	19.86	18.75	14.34	9.93
		12	232.45	22.31	21.07	16.11	11.15
[33]	Virtex-7	24	374.035	–	–	–	7.979
[34]	Virtex-5	24	338.409	15.86	15.34	11.73	8.12
	Virtex-6	24	376.081	17.63	17.05	13.04	9.02
[35]	Virtex-5	24	387	–	–	–	8.06
This work	Virtex-5	24	347.49	16.680	15.753	12.046	8.340
		12	203.28	19.515	18.431	14.094	9.757
	Virtex-6	24	438.49	21.048	19.878	15.201	10.524
		12	347.84	33.393	31.537	24.117	16.696
	Virtex-7	24	498.27	23.917	22.588	17.273	11.958
		12	378.73	36.358	34.338	26.259	18.179
	Artix-7	24	397.41	19.075	18.015	13.776	9.537
		12	254.46	24.428	23.071	17.642	12.214

Table 8 The outcomes of the implementation in terms of efficiency and comparison

Design	FPGA device	Number of clock cycles	Area (total number of slices)	Efficiency ($r = 1152$)	Efficiency ($r = 1088$)	Efficiency ($r = 832$)	Efficiency ($r = 576$)
[28]	Virtex-5	24	1217	–	10.31	–	5.4
	Virtex-7	24	998	–	13.63	–	7.27
[29]	Artix-7	24	4188	–	–	–	3.93
		12	7139	–	–	–	2.80
[30]	Virtex-6	24	871	–	–	–	4.22
		12	1406	–	–	–	11.47
[31]	Virtex-5	24	1434	–	3.32	–	–
		16	1562	–	3.44	–	–
[32]	Virtex-5	24	1365	11.47	10.83	8.28	5.73
		12	2144	8.60	8.13	6.21	4.30
	Virtex-6	24	1432	13.87	13.10	10.02	6.93
		12	3557	6.27	5.93	4.53	3.14
[33]	Virtex-7	24	1454	–	–	–	5.49
[34]	Virtex-5	24	935	16.96	16.40	12.54	8.68
	Virtex-6	24	1019	17.30	16.73	12.80	8.85
[35]	Virtex-5	24	1680	–	–	–	4.91
This work	Virtex-5	24	868	19.22	18.15	13.88	9.61
		12	1112	17.55	16.57	12.67	8.77
	Virtex-6	24	946	22.25	21.01	16.07	11.12
		12	1287	25.95	24.50	18.74	12.97
	Virtex-7	24	1094	21.86	20.65	15.79	10.93
		12	1375	26.44	24.97	19.10	13.22
	Artix-7	24	902	21.14	19.97	15.27	10.57
		12	1184	20.63	19.48	14.90	10.31

On the Virtex-7 board, the proposed design marginally occupied the highest area of 1094 slices, while the design [28] consumed the lowest area of around 998 slices. However, it is observed that [28] produced low frequency, efficiency, and throughput. Thus, it indicates that the proposed design may have slightly higher area utilization but is still more efficient in performance compared to the design [28].

On the Artix-7 board, the design [29] occupied a significantly larger area, specifically 4188 slices, whereas the proposed design consumed a much lower area of approximately 902 slices. Even though the design [29] achieved a higher throughput for 512 output length performed poorly, with an efficiency measure of 3.93 Mbps/slices. In contrast, the proposed design exhibited a higher efficiency rate of 10.57 Mbps/slices. This indicates that the proposed design is more effective at area and efficiency metrics and is more area-efficient than [29]. The proposed design with 24 clock cycles delivers the highest throughput when implemented on Virtex-5, Virtex-6, and Virtex-7 boards. In addition to its impressive throughput achievements, the proposed design also excels in low area utilization and efficiency when deployed on the Artix-7 board.

On the Virtex-5 board with 12 total clock cycles, the design mentioned in [32] occupied the highest area of 2144 slices, while the proposed design consumed the lowest area of around 1112 slices. It indicates that the proposed design is more area-efficient than the design mentioned in [32]. On the Virtex-6 board with 12 total clock cycles, the design mentioned in [32] occupied the highest area of 3557 slices, whereas the proposed design consumed the lowest area of around 1287 slices. In this case, the proposed design is also more area-efficient than the design mentioned in [32]. On the Artix-7 board, the design [29] occupied a much larger area, 7139 slices, but the proposed design consumed a substantially lower area, about 1184 slices. The design [29] performed poorly in terms of efficiency, with an efficiency value of 2.80 Mbps/slices, while the presented design demonstrated a higher efficiency rate of 10.31 Mbps/slices. The proposed design with 12 clock cycles achieves the highest throughput when utilized on Virtex-5, Virtex-6, and Virtex-7 FPGA boards. Furthermore, it exhibits exceptional efficiency and low area utilization when implemented on the Artix-7 board.

Consequently, the assessment of the presented architectural design concerning recent works in the literature demonstrates that it surpasses them in terms of throughput and area, making it an essential contribution to hash function design. So, applications that need speedy and efficient hash functions may benefit from this performance improvement.

7 Conclusions and future work

The importance of cryptography in ensuring the security and confidentiality of digital media cannot be overstated in today's interconnected world. With the transmission of sensitive information in various forms, including text, image, video, and audio, it is crucial to have robust encryption algorithms that offer high-level security and resistance to attacks.

The SHA-3 (Keccak) algorithm is one such algorithm that has gained popularity due to its strong resistance to cryptanalysis attacks and its good combination of speed, performance, and security. Its adoption by NIST as a more secure replacement for SHA-1 and SHA-2 highlights its significance in ensuring the safety and integrity of digital data.

This study focuses on optimizing the performance of the SHA-3 algorithm for all output lengths (224, 256, 384, and 512 bits) on the and Artix-7, Virtex-5, Virtex-6, and Virtex-7 FPGA boards. The research compares the proposed innovative method to similar designs. It shows that the presented architectural design has the highest performance in the standard evaluation criteria of area (slices), throughput (Gbps), and efficiency (Mbps/slices).

The study achieved an area of 1375 slices, a throughput of 36.358 Gbps, and an efficiency of 26.44 Mbps/slices with the Virtex-7 FPGA board, demonstrating the efficacy of the proposed architecture. However, future research will enhance throughput and efficiency performance metrics per round and propose more practical experiments implementing FPGAs and entire systems-on-chip.

Author Contributions Conceptualization, A.S.; Formal analysis, A.S.; Investigation, A.S. and T.S.; Methodology, A.S.; Software, A.S.; Validation, A.S. and T.S.; Project administration, A.S.; Supervision, M.D.; Writing-original draft, A.S. and T.S.; All authors reviewed the manuscript.

Funding Open access funding provided by HEAL-Link Greece.

Declarations

Conflict of interest The authors declare no conflict of interests.

References

1. Gnatyuk, S., Kinzeryavyy, V., Kyrychenko, K., Yubuzova, K., Aleksander, M., Odarchenko, R.: Secure hash function constructing for future communication systems and networks. In: Advances in Artificial Systems for Medicine and Education II 2, pp. 561–569 (2020). Springer
2. Guo, S., Zeng, D., Xiang, Y.: Chameleon hashing for secure and privacy-preserving vehicular communications. IEEE Trans. Parallel Distrib. Syst. **25**(11), 2794–2803 (2013)
3. Chi, L., Zhu, X.: Hashing techniques: a survey and taxonomy. ACM Comput. Surv. (CSUR) **50**(1), 1–36 (2017)
4. Sravani, M.M., Durai, S.A.: Attacks on cryptosystems implemented via vlsi: A review. J. Inform. Secur. Appl. **60**, 102861 (2021)
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32, pp. 313–314 (2013). Springer
6. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full sha-1. In: Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I 37, pp. 570–596 (2017). Springer
7. Mladenov, T., Nooshabadi, S.: Implementation of reconfigurable sha-2 hardware core. In: APCCAS 2008-2008 IEEE Asia Pacific Conference on Circuits and Systems, pp. 1802–1805 (2008). IEEE
8. Li, L., Lin, S., Shen, S., Wu, K., Li, X., Chen, Y.: High-throughput and area-efficient fully-pipelined hashing cores using BRAM in FPGA. Microprocess. Microsyst. **67**, 82–92 (2019)
9. Pham, H.L., Tran, T.H., Le, V.T.D., Nakashima, Y.: A high-efficiency FPGA-based multimode sha-2 accelerator. IEEE Access **10**, 11830–11845 (2022)
10. Rosero-Montalvo, P.D., István, Z., Hernandez, W.: A survey of trusted computing solutions using FPGAS. IEEE Access (2023)
11. Rothmann, M., Porrmann, M.: A survey of domain-specific architectures for reinforcement learning. IEEE Access **10**, 13753–13767 (2022)
12. Knezevic, M., Kobayashi, K., Ikegami, J., Matsuo, S., Satoh, A., Kocabas, Ü., Fan, J., Katashita, T., Sugawara, T., Sakiyama, K., et al.: Fair and consistent hardware evaluation of fourteen round two sha-3 candidates. IEEE Trans. Very Large Scale Integr. VLSI Syst. **20**(5), 827–840 (2011)
13. Siddiqui, F., Amiri, S., Minhas, U.I., Deng, T., Woods, R., Rafferty, K., Crookes, D.: FPGA-based processor acceleration for image processing applications. J. Imaging **5**(1), 16 (2019)
14. Ruiz-Rosero, J., Ramirez-Gonzalez, G., Khanna, R.: Field programmable gate array applications-a scientometric review. Computation **7**(4), 63 (2019)
15. Winderickx, J., Braeken, A., Singelée, D., Mentens, N.: In-depth energy analysis of security algorithms and protocols for the internet of things. J. Cryptogr. Eng. **12**(2), 137–149 (2022)
16. Templin, J.R., Hamlet, J.R.: A new power-aware FPGA design metric. J. Cryptogr. Eng. **5**, 1–11 (2015)
17. Rodríguez-Andina, J.J., Valdes-Pena, M.D., Moure, M.J.: Advanced features and industrial applications of FPGAS-a review. IEEE Trans. Industr. Inf. **11**(4), 853–864 (2015)
18. Al-Odat, Z.A., Ali, M., Abbas, A., Khan, S.U.: Secure hash algorithms and the corresponding FPGA optimization techniques. ACM Comput. Surv. (CSUR) **53**(5), 1–36 (2020)
19. Jungk, B., Stöttinger, M.: Serialized lightweight SHA-3 FPGA implementations. Microprocess. Microsyst. **71**, 102857 (2019)
20. Yang, Y., He, D., Kumar, N., Zeadally, S.: Compact hardware implementation of a SHA-3 core for wireless body sensor networks. IEEE Access **6**, 40128–40136 (2018)
21. Sideris, A., Sanida, T., Dasygenis, M.: High throughput implementation of the Keccak hash function using the NIOS-II processor. Technologies **8**(1), 15 (2020)
22. Sideris, A., Sanida, T., Dasygenis, M.: High throughput pipelined implementation of the sha-3 cryptoprocessor. In: 2020 32nd International Conference on Microelectronics (ICM), pp. 1–4 (2020). IEEE
23. Sundal, M., Chaves, R.: Efficient FPGA implementation of the SHA-3 hash function. In: 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 86–91 (2017). IEEE
24. Arshad, A., Aziz, A., et al.: Compact implementation of SHA3-512 on FPGA. In: 2014 Conference on Information Assurance and Cyber Security (CIACS), pp. 29–33 (2014). IEEE
25. Jungk, B., Stöttinger, M.: Among slow dwarfs and fast giants: A systematic design space exploration of keccak. In: 2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1–8 (2013). IEEE
26. Latif, K., Aziz, A., Mahboob, A.: Look-up table based implementations of SHA-3 finalists: JH, Keccak and skein. KSII Transactions on Internet and Information Systems (TIIS) **6**(9), 2388–2404 (2012)
27. Gaj, K., Homsirikamol, E., Rogawski, M.: Fair and comprehensive methodology for comparing hardware performance of fourteen round two sha-3 candidates using fpgas. In: Cryptographic Hardware and Embedded Systems, CHES 2010: 12th International Workshop, Santa Barbara, USA, August 17-20, 2010. Proceedings 12, pp. 264–278 (2010). Springer
28. Aziz, A., Latif, K., et al.: Resource efficient implementation of the Keccak, skein & JH algorithms on a reconfigurable platform. Cankaya Univ. J. Sci. Eng. **13**(1), 40–57 (2016)
29. Paul, R., Shukla, S.: Partitioned security processor architecture on FPGA platform. IET Comput. Digital Techn. **12**(5), 216–226 (2018)
30. Wong, M.M., Haj-Yahya, J., Sau, S., Chattopadhyay, A.: A new high throughput and area efficient SHA-3 implementation. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2018). IEEE
31. Gholipour, A., Mirzakuchaki, S.: High-speed implementation of the Keccak hash function on FPGA. Int. J. Adv. Comput. Sci. **2**(8), 303–307 (2012)
32. El Moumni, S., Fettach, M., Tragha, A.: High throughput implementation of sha3 hash algorithm on field programmable gate array (FPGA). Microelectron. J. **93**, 104615 (2019)
33. Van Hieu, D., Khai, L.D.: A fast Keccak hardware design for high performance hashing system. In: 2021 15th International Conference on Advanced Computing and Applications (ACOMP), pp. 162–168 (2021). IEEE
34. Assad, F., Fettach, M., El Otmani, F., Tragha, A.: High-performance FPGA implementation of the secure hash algorithm 3 for single and multi-message processing. Int. J. Electr. Comput. Eng. **12**(2), 1324–1333 (2022)
35. Mestiri, H., Barraj, I.: High-speed hardware architecture based on error detection for Keccak. Micromachines **14**(6), 1129 (2023)
36. Dworkin, M.J.: Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, National Institute of Standards and Technology (NIST) (2015)
37. Guido, B., Joan, D., Michaël, P., Gilles, V.: Cryptographic sponge functions. Citeseer (2011)
38. Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., Song, L.: Practical collision attacks against round-reduced sha-3. J. Cryptol. **33**, 228–270 (2020)

39. Baldwin, B., Byrne, A., Lu, L., Hamilton, M., Hanley, N., O'Neill, M., Marnane, W.P.: Fpga implementations of the round two sha-3 candidates. In: 2010 International Conference on Field Programmable Logic and Applications, pp. 400–407 (2010). IEEE
40. Bertoni, G., Daemen, J., Peeters, M., Assche, G.: The keccak reference. Submission to NIST (Round 3) **13**, 14–15 (2011)
41. Computer Security Division, I.T.L.: Example Values - Cryptographic Standards and Guidelines: CSRC (2016). https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/example-values Accessed 20 February 2023
42. Ferraz, O., Subramaniyan, S., Chinthala, R., Andrade, J., Cavallaro, J.R., Nandy, S.K., Silva, V., Zhang, X., Purnaprajna, M., Falcao, G.: A survey on high-throughput non-binary LDPC decoders: ASIC, FPGA, and GPU architectures. IEEE Commun. Surv. Tutor. **24**(1), 524–556 (2021)
43. Michail, H., Kakarountas, A., Milidonis, A., Goutis, C.: A top-down design methodology for ultrahigh-performance hashing cores. IEEE Trans. Dependable Secure Comput. **6**(4), 255–268 (2008)
44. Mohajerani, K., Haeussler, R., Nagpal, R., Farahmand, F., Abdulgadir, A., Kaps, J.-P., Gaj, K.: FPGA benchmarking of round 2 candidates in the nist lightweight cryptography standardization process: methodology, metrics, tools, and results. Cryptology ePrint Archive (2020)