

Article

A Novel Hardware Architecture for Enhancing the Keccak Hash Function in FPGA Devices

Argyrios Sideris , Theodora Sanida  and Minas Dasygenis 

Department of Electrical and Computer Engineering, University of Western Macedonia, 50131 Kozani, Greece; thsanida@uowm.gr (T.S.); mdasyg@ieee.org (M.D.)

* Correspondence: asideris@uowm.gr; Tel.: +30-24610-56534

Abstract: Hash functions are an essential mechanism in today's world of information security. It is common practice to utilize them for storing and verifying passwords, developing pseudo-random sequences, and deriving keys for various applications, including military, online commerce, banking, healthcare management, and the Internet of Things (IoT). Among the cryptographic hash algorithms, the Keccak hash function (also known as SHA-3) stands out for its excellent hardware performance and resistance to current cryptanalysis approaches compared to algorithms such as SHA-1 and SHA-2. However, there is always a need for hardware enhancements to increase the throughput rate and decrease area consumption. This study specifically focuses on enhancing the throughput rate of the Keccak hash algorithm by presenting a novel architecture that supplies efficient outcomes. This novel architecture achieved impressive throughput rates on Field-Programmable Gate Array (FPGA) devices with the Virtex-5, Virtex-6, and Virtex-7 models. The highest throughput rates obtained were 26.151 Gbps, 33.084 Gbps, and 38.043 Gbps, respectively. Additionally, the research paper includes a comparative analysis of the proposed approach with recently published methods and shows a throughput rate above 11.37% Gbps in Virtex-5, 10.49% Gbps in Virtex-6 and 11.47% Gbps in Virtex-7. This comparison allows for a comprehensive evaluation of the novel architecture's performance and effectiveness in relation to existing methodologies.

Keywords: Keccak hash function; hardware acceleration; cryptography circuits; Field-Programmable Gate Array (FPGA); Secured Hash Algorithm-3 (SHA-3)



Citation: Sideris, A.; Sanida, T.; Dasygenis, M. A Novel Hardware Architecture for Enhancing the Keccak Hash Function in FPGA Devices. *Information* **2023**, *14*, 475. <https://doi.org/10.3390/10.3390/info14090475>

Academic Editors: Nelly Leligou, Theodore Zahariadis, Panagiotis Trakadas and Panagiotis A. Karkazis

Received: 3 July 2023

Revised: 17 August 2023

Accepted: 24 August 2023

Published: 28 August 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Hash functions play a vital role in the domain of information security, serving as fundamental tools in various applications. One of their primary purposes is to ensure information security by providing services like authentication and integrity. In the context of password storage and verification, hash functions are extensively utilized to transform passwords into irreversible hashes, protecting the original passwords from unauthorized access [1,2]. Additionally, these functions find application in generating pseudo-random sequences, which are essential in cryptography and crucial derivation for various purposes such as military, online commerce, healthcare management, banking, and the Internet of Things (IoT) [3–5].

However, it is worth noting that several commonly used hash algorithms, including SHA-1, Snefru, MD4, MD5, RIPEMD, and HAVAL, have been discovered to be vulnerable to collision attacks. Collision attacks occur when two distinct inputs produce the same hash output, compromising the hash function's security [6–8]. The National Institute of Standards and Technology (NIST) took proactive measures to address this issue and enhance information security. They organized a three-round contest to replace the SHA-2 hashing standard, which was considered relatively secure at the time. The contest sought to identify a new hashing standard that could withstand potential attacks and provide robust security [9–11].

The NIST contest eventually concluded with selecting and adopting the Keccak function as the new hashing standard. Keccak, also known as SHA-3, is designed to resist collision attacks and other cryptographic vulnerabilities. Its selection as the new standard signifies the importance of continuously advancing hash functions to meet evolving security requirements and protect sensitive information in today's interconnected digital landscape [12]. By adopting more secure hash algorithms like Keccak, organizations and individuals can enhance the overall security of their systems and data, reducing the risk of unauthorized access and malicious activities [13]. The Keccak algorithm is a widely utilized cryptographic hash function that plays a crucial role in ensuring data privacy and maintaining the integrity of data exchange in various daily systems. However, there is an ongoing challenge in enhancing the performance of Keccak's circuit implementation, particularly in the context of embedded systems [14,15].

One advantage of implementing Keccak in Field-Programmable Gate Arrays (FPGAs) is its superior speed compared to previous SHA algorithms when implemented in hardware. Keccak is designed to deliver excellent performance across various hardware platforms. Utilizing FPGAs for Keccak implementation offers the benefits of algorithm customization and reconfigurability flexibility. FPGAs can be tailored to consume less power than traditional processors, making them ideal for implementing cryptographic functions like Keccak. Moreover, FPGAs can significantly enhance the throughput of Keccak calculations [16,17]. As a result of these advantages, several strategies have been proposed to implement the Keccak algorithm effectively. These approaches focus on reducing energy consumption, maximizing the utilization of area resources, or enhancing processing speed [18]. Researchers and developers are actively exploring innovative techniques to optimize the implementation of Keccak in embedded systems, leveraging the capabilities of FPGAs to achieve improved efficiency, security, and performance [19]. By addressing the challenges associated with Keccak's circuit implementation, particularly in the realm of embedded systems, it becomes possible to unlock the full potential of this algorithm in ensuring data privacy and maintaining the integrity of data exchange across various applications. The continuous advancements in FPGA technology and the ongoing research efforts in algorithm optimization contribute to the evolution and broader adoption of Keccak in securing sensitive information in daily systems [20,21].

The following is a brief summary of the various contributions that were made in this article:

- We propose a novel technique optimization strategy for enhancing the efficiency of the Keccak algorithm. Our approach is based on the concepts of unrolling and pipelining, which are well-established methods in FPGA devices. The primary objective of this work is to improve the performance of Keccak in terms of throughput, frequency, and area utilization (Section 4.3.1).
- We propose a new format for the Round Constant (RC) generator, aiming to improve its performance in terms of throughput and efficiency while also decreasing the hardware resources required. The new format introduces a more straightforward structure for the RC generator, in which the number of necessary XOR operations is reduced to only seven. The decreased computation leads to faster execution times, thereby improving the overall throughput of the algorithm (Section 4.3.2).
- To ensure the accuracy and reliability of our proposed method, we conducted a thorough validation process using established examples provided by the NIST. This validation step is crucial in ensuring that our proposed strategy maintains the necessary cryptographic properties and adheres to the specifications outlined by NIST for the Keccak algorithm (Section 5.1).
- Finally, we conducted an extensive evaluation and analysis of our proposed architecture, comparing it to other similar methods described in the published literature. The evaluation focused on key performance metrics, including area utilization (in terms of slices), throughput (in Gbps), frequency (in MHz), and efficiency (in Mbps/slice). By leveraging these evaluations and comparisons, we can confidently

assert the superiority of our design in terms of performance, efficiency, and area utilization (Section 5.3).

The remainder of the study is organized as follows: In the next Section 2, we present the related studies in the literature. In Section 3, we shortly present the Keccak outline. Section 4 describes our new suggested hardware optimization strategy of the Keccak algorithm on FPGA. In Section 5, we demonstrate the experimental outcomes of our work and the comparisons with other suitable works. In Section 6, we discuss our optimization strategy. Finally, Section 7 summarizes our study's findings and future work.

2. Related Work

The cryptography community has conducted extensive investigations to optimize architectures and approaches for implementing the Keccak algorithm on FPGA devices. The primary goal of these architectures is to enhance the FPGA's throughput while reducing its area requirements [19,22]. However, despite these efforts, a critical need remains to improve further performance measurements associated with throughput and area decrease [23–25]. This section will examine other similar works and discuss their findings in detail.

The study [26] proposed a new technique for implementing the Keccak design with an output size of 512 bits. The authors performed an evaluation of the suggested design of the Virtex-5. In this implementation, for an output size of 512 bits, the Virtex-5 FPGA required 1680 slices and operated at a clock frequency of 387 MHz. This specific design achieved a throughput of 8.06 Gbps. Furthermore, the efficiency of this design was measured and found to be 4.91 Mbps/Slice. In [27], the authors suggested a new approach for implementing the Keccak method with an output size of 512 bits. The authors evaluated the performance of the presented design on the Virtex-7 FPGA device. In this implementation, for an output size of 512 bits, the Virtex-7 FPGA needed 1454 slices and operated at a clock frequency of 374.035 MHz. This configuration achieved a throughput of 7.979 Gbps and an efficiency of 5.49 Mbps/Slice.

Rao et al. [28] proposed a new method for Keccak implementation in Virtex-5 and Virtex-6 FPGA. Their emphasis was primarily on output sizes of 256 and 512 bits. For the Virtex-5 architecture, when implementing Keccak with an output size of 256 bits, the design utilized 1291 slices and operated at a clock frequency of 377.86 MHz. This configuration achieved a throughput of 17.132 Gbps. On the other hand, when targeting an output size of 512 bits, the Virtex-5 architecture employed 1409 slices and attained a throughput of 10.19 Gbps. In the case of the Virtex-6 architecture, the proposed Keccak implementation with an output size of 256 bits required 1028 slices and operated at a clock frequency of 424.44 MHz. This resulted in a higher throughput of 19.241 Gbps than the Virtex-5 implementation. The Virtex-6 implementation with an output size of 512 bits utilized 1227 slices and achieved a throughput of 8.22 Gbps.

In [29], the authors suggested a new design for implementing Keccak architecture with an output size of 512. This design offers a trade-off between the maximum frequency and the area implementation, allowing flexibility in optimizing performance and resource utilization. The authors evaluated the performance of the proposed design on different FPGA devices, specifically focusing on Virtex-5, Virtex-6, and Virtex-7. For an output size of 512, the Virtex-5 architecture required 1388 slices and operated at a clock frequency of 287.39 MHz. This configuration achieved a throughput of 11.50 Gbps. Moving to the Virtex-6 architecture, the proposed design required 1167 slices and operated at a higher clock frequency of 394.01 MHz. This increased clock frequency improved throughput of 15.76 Gbps. Finally, when considering the Virtex-7 architecture, the design utilized 1418 slices and operated at a clock frequency of 414.54 MHz. This configuration achieved the highest throughput among the evaluated architecture, reaching 16.58 Gbps.

The authors of [30] introduced a design approach for implementing the Keccak design with an output size of 512 bits. The authors performed an evaluation of the suggested design of the Virtex-5 and Virtex-6 FPGA. In this design, the Virtex-6 FPGA required 2296 slices and operated at a clock frequency of 391 MHz. This specific design achieved a throughput

of 9.38 Gbps. Furthermore, the efficiency of this design was measured and found to be 8.17 Mbps/Slice. In [31], the authors suggested a new approach for implementing the Keccak method with an output size of 512 bits. The authors evaluated the performance of the presented design on the Virtex-5 FPGA device. In this implementation, for an output size of 512 bits, the Virtex-5 FPGA needed 2326 slices and operated at a clock frequency of 306 MHz. This design earned a throughput score of 5.56 Gbps and an efficiency score of 2.40 Mbps/Slice. The study [32] presented a new design for implementing the Keccak design with an output size of 512 bits. In this implementation, the Virtex-5 FPGA required 1163 slices and operated at a clock frequency of 273 MHz. This design earned a throughput score of 7.80 Gbps. Also, the efficiency score of this design reached 6.06 Mbps/Slice.

Table 1 includes the Keccak algorithm with recently published methods. Most previous works on the Keccak algorithm have primarily focused on utilizing the classic 64-bit RC generator. However, this work aims to improve upon these existing approaches by introducing an optimized RC generator that significantly reduces its size. The primary objective of this study is to compare the performance metrics, specifically the efficiency and the throughput, with the enhanced RC generator integrated into the Keccak algorithm. By reducing the size of the RC generator, the proposed optimization technique aims to achieve superior performance results compared to previous investigations. The findings of this study demonstrate that the proposed optimization technique surpasses the performance measures achieved by previous approaches. The results of this study suggest that the optimized RC generator can serve as a promising strategy for FPGA boards.

Table 1. Outline in the recent publications for the Keccak algorithm.

Study	Output Size	RC Generator	FPGA
[26]	512	64	Virtex-5
[27]	512	64	Virtex-7
[28]	512	64	Virtex-5 and Virtex-6
[29]	512	64	Virtex-5, Virtex-6, and Virtex-7
[30]	512	64	Virtex-5, and Virtex-6
[31]	512	64	Virtex-5
[32]	512	64	Virtex-5

3. Keccak Outline

In the context of the NIST hash function competition [33], the SHA-3 family of cryptographic hash functions was developed. It was developed as an alternative to the SHA-2 family, which is commonly used, to provide enhanced security together with enhanced performance features. The Keccak family has a total of four different hash functions, which are referred to as Keccak-224, Keccak-256, Keccak-384, and Keccak-512, respectively. These functions are all supported by the same fundamental framework, which is referred to as the sponge construction [34]. The sponge construction is a flexible framework that enables the production of hash values of varying lengths. It is ideal for a wide variety of applications because of these features. The key concept behind the sponge construction is the use of a state represented as a two-dimensional array of bits. The state is divided into bitrate (r) and capacity (c). The input message is processed in blocks of size r , and the internal state is updated accordingly. The capacity serves as a buffer to introduce non-linearity and enhance the security properties of the algorithm.

As shown in Figure 1, the Keccak hash functions operate in two main phases: the absorbing and squeezing phases. The input message is absorbed into the state using the sponge construction in the absorbing phase. This phase prepares the input data by applying the f function, which incorporates bitwise operations, modular addition, and rotation operations to introduce diffusion and confusion. In the squeezing phase, the desired hash output is obtained by repeatedly squeezing blocks of data from the state.

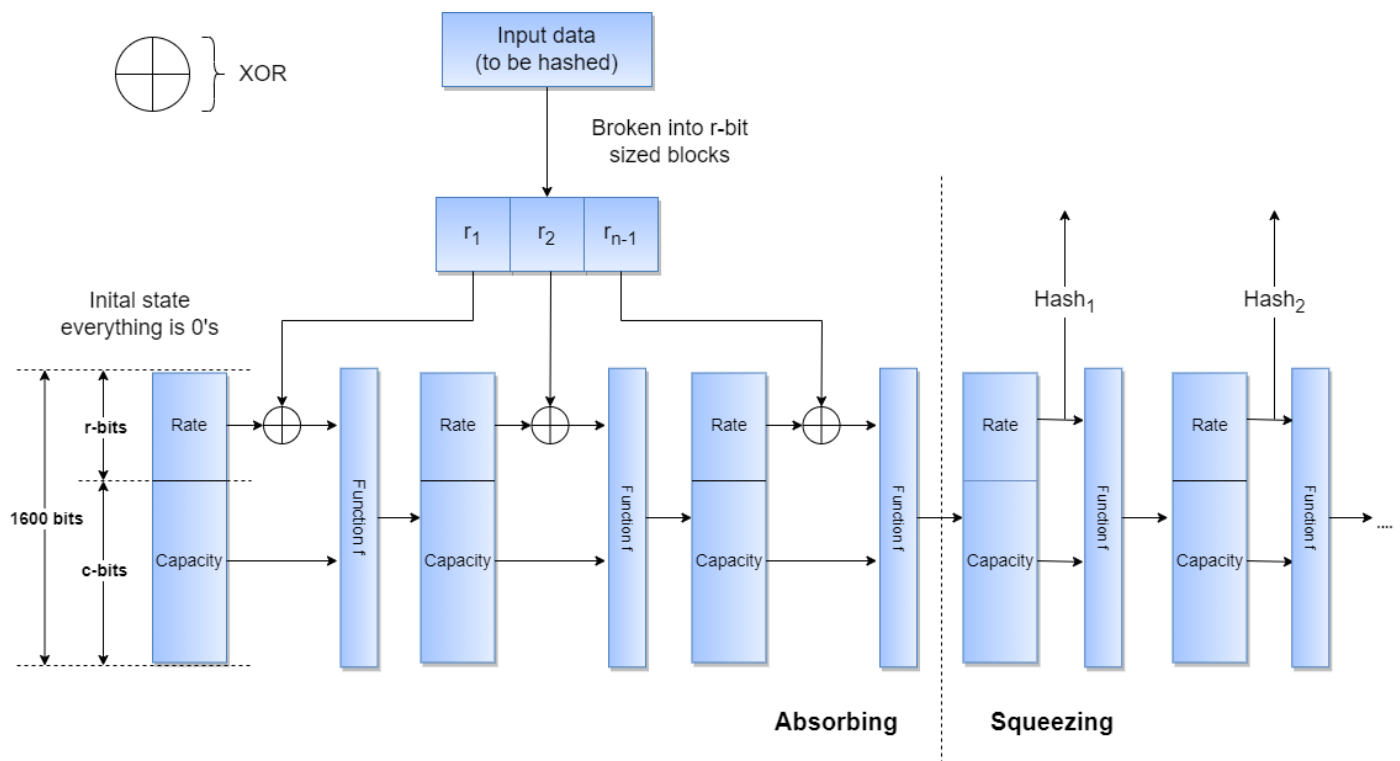


Figure 1. Sponge construction of the Keccak algorithm.

The permutation function f consists of a round function applied iteratively to the state array. Each round applies a series of bitwise operations, such as bitwise rotations, XOR, AND, and NOT, to modify the state array nonlinearly and highly complexly. The round function incorporates a set of constant values called RC, which introduce additional entropy into the computation.

During each round of the permutation function, the first of the steps the state array goes through is called “theta” using Equation (1). The theta step aims to introduce diffusion and increase the mixing of the bits within the state array. In the theta step, each bit of the state array is XORed with a linear combination of its neighbouring bits. This XOR operation helps to distribute information and propagate changes across the state array. By incorporating this linear combination, the theta step ensures that small changes in one part of the state array can significantly impact other parts, increasing diffusion and complexity. The specific linear combination used in the theta step may vary depending on the cryptographic algorithm or permutation function. It is designed to achieve a balanced distribution of bits and prevent the concentration of information in specific regions of the state array. Applying the theta step iteratively in each round of the permutation function increases the diffusion and mixing of bits throughout the state array. This helps to ensure that the output of the permutation function exhibits a high degree of randomness and complexity, making it difficult for an attacker to extract meaningful information or identify any patterns within the encrypted data.

Theta (θ) step:

$$\begin{aligned}
 C[x] &= A[x,0] \text{ XOR } A[x,1] \text{ XOR } A[x,2] \text{ XOR } A[x,3] \text{ XOR } A[x,4], \\
 x &= 0 \dots 4 \\
 D[x] &= C[x-1] \text{ XOR } \text{ROTATE}(C[x+1], 1), \\
 x &= 0 \dots 4 \\
 A[x,y] &= A[x,y] \text{ XOR } D[x], \\
 x &= 0 \dots 4
 \end{aligned} \tag{1}$$

After the theta step in the permutation function, the state array undergoes another step called “rho” using Equation (2). The rho step involves performing a series of predefined circular bit rotations on specific positions within the state array. This step enhances the diffusion of bits and ensures that each bit interacts with a wide range of other bits. In the rho step, specific positions in the state array are selected, and a fixed number of positions rotates the bits within those positions. The circular rotation means that the bits rotated out of one end of the position reappear at the other end, creating the circular shifting effect. The rho step introduces further complexity and randomness into the state array by applying these circular bit rotations. Additionally, it helps to spread the influence of each bit across different positions and ensures that each bit interacts with a larger number of other bits in subsequent rounds.

Rho (ρ) step:

$$A[x, y] = \text{ROTATE} (A'[x, y], r[x, y]), \quad [x, y] \leq 4 \quad (2)$$

After the rho step in the permutation function, the state array proceeds to undergo the “pi” step using Equation (3). The pi step is responsible for rearranging the bits within each lane of the state array, introducing a form of permutation that contributes to a high degree of confusion and diffusion. In the pi step, the positions of the bits within each lane are rearranged according to a predefined permutation pattern. This permutation pattern determines the new positions of the bits within their respective lanes. By shuffling the bits this way, the pi step ensures that each bit interacts with a different set of neighbouring bits in subsequent rounds. The rearrangement of bits within each lane provides significant confusion as it disrupts any potential patterns or relationships between the bits.

Pi (π) step:

$$B[y, 2x + 3y] = A[x, y], \quad [x, y] \leq 4 \quad (3)$$

In each round of the permutation function, the “chi” operation follows the pi step using Equation (4). The chi step introduces nonlinearity and plays a crucial role in amplifying the effects of small changes in the input, resulting in significant changes throughout the state array. Each bit in the state array is combined with two other bits during the chi step to produce a new bit value. The specific combination is performed using bitwise logical operations, XOR (exclusive OR), AND (logical AND), and NOT (logical NOT). The purpose of the chi step is to introduce nonlinearity into the permutation function. Combining each bit with two others ensures that even small changes in one bit will have a cascading effect on multiple bits in subsequent rounds. This amplification of changes enhances the diffusion and spreading of information within the state array.

Chi (χ) step:

$$A[x, y] = B[x, y] \text{ XOR } (\text{NOT } B[x + 1, y]) \text{ AND } (B[x + 2, y]), \quad [x, y] = 0..4 \quad (4)$$

Finally, the “iota” step using Equation (5) is a simple XOR operation that introduces additional randomness into the state array. It involves XORing a specific bit position in each lane of the state array with a constant value derived from the round index. The constant values used in the iota step are known as RC and are unique to each round of the Keccak hash function. By incorporating the iota step in the permutation function, Keccak ensures that each round of the hash function introduces additional complexity and randomness, enhancing the algorithm’s overall security and cryptographic strength.

Iota (ι) step:

$$A[0, 0] = A[0, 0] \text{ XOR } \text{RC}[i] \quad (5)$$

4. New Hardware Optimization Strategy

Our proposed architecture combines the benefits of unrolling and pipelining to achieve significant performance improvements in terms of throughput and efficiency. By leveraging the capabilities of unrolling and the enhanced frequency achieved through pipelining, we

can optimize the Keccak algorithm for increased throughput, higher operating frequency, and efficient area utilization. Figure 2 illustrates the system architecture of the proposed optimization strategy.

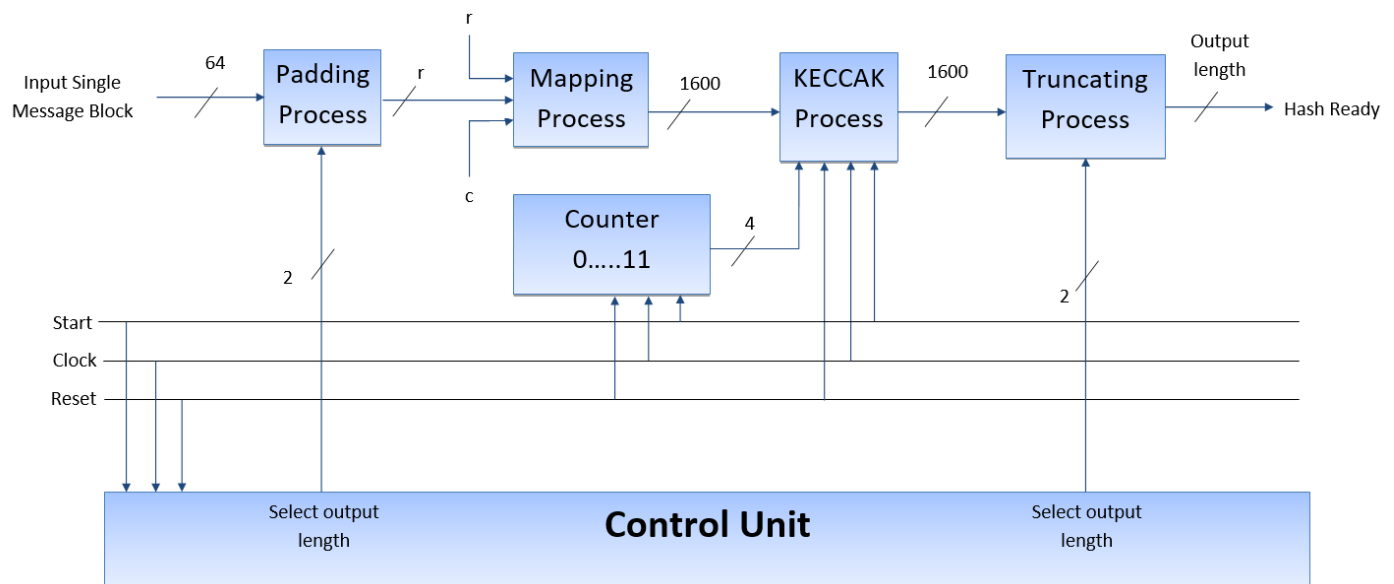


Figure 2. The proposed architecture overview approach.

4.1. Padding Process

The cryptographic hash function is designed to process messages of arbitrary size. However, the internal permutation function within the hash function requires a fixed size, denoted as “ r ”, of data to be processed at a time. This misalignment between the variable-sized input message and the fixed-size internal permutation necessitates the use of padding. The padding technique is applied to the initial message to generate a padded message that aligns with the required block size. A padded message of size $w \times r$, where w is an integer, is created by concatenating a set of bits to the initial message. The specific bits added during padding depend on the chosen padding scheme. The block size “ r ” choice depends on the desired size of the resulting digest. Different variants of the Keccak hash function, such as Keccak-224, Keccak-256, Keccak-384, and Keccak-512, have different block sizes and output sizes. Table 2 supplies an overview of the values of “ r ” and “ c ” for these Keccak variants, where “ c ” represents the capacity. In addition to providing compatibility between the input message and the internal permutation, the padding phase also strengthens the security of Keccak against length extension-based attacks. Length extension attacks exploit vulnerabilities in hash functions that allow an attacker to append further data to an existing hash value without knowing the authentic input.

Table 2. The output lengths of the Keccak and the parameters (r, c).

Desired Output	Block Size “ r ”	Capacity “ c ”
Keccak – 224	1152	448
Keccak – 256	1088	512
Keccak – 384	832	768
Keccak – 512	576	1024

4.2. Mapping Process

The mapping process in the preprocessing phase of the Keccak algorithm aims to generate input data in three dimensions. This is achieved by utilizing the following Equation (6):

$$\text{State}[x, y, z] = [(((\text{Padded data } r \text{ XOR } r) || c))^*[(z + 64 * (5 * y + x))]] \quad (6)$$

The equation maps the coordinates (x, y, z) to a linear index within the state to generate the input data in three dimensions. The state has a size of $5 \times 5 \times 64$, with x and y ranging from 0 to 4 and z ranging from 0 to 63. By substituting the values of x , y , and z into the equation, the resulting State (x, y, z) value can be calculated. This process allows for generating the desired input data for the subsequent stages of the Keccak algorithm.

4.3. Keccak Process

4.3.1. Optimization Strategy

In our proposed optimization, as shown in Figure 3, we have implemented a two-stage sub-pipelining approach within the f-permutation block of the Keccak hash function. Additionally, we have unrolled the overall hash function by a factor of 2 and inserted two pipelines between the rounds. The two-stages sub-pipelining specifically divides the computation between the “theta” step and the remaining four steps (“rho”, “pi”, “chi”, and “iota”) of the f-permutation block. This division allows for more efficient processing and reduces the critical path, ultimately aiming to achieve a higher clock frequency. In the first half of the computation, which includes the “theta” step, the longest delay comprises five XOR operations. On the other hand, the second half, covering the “pi” to “iota” steps, incurs the most extended delay of two XOR operations, one AND operation, and one additional XOR operation. By implementing this sub-pipelining approach and optimizing the critical path, we can significantly reduce the overall delay and improve the clock frequency at which the hash function can operate. This enhancement leads to a more efficient and high-performance implementation of the Keccak hash function in our proposed architecture.

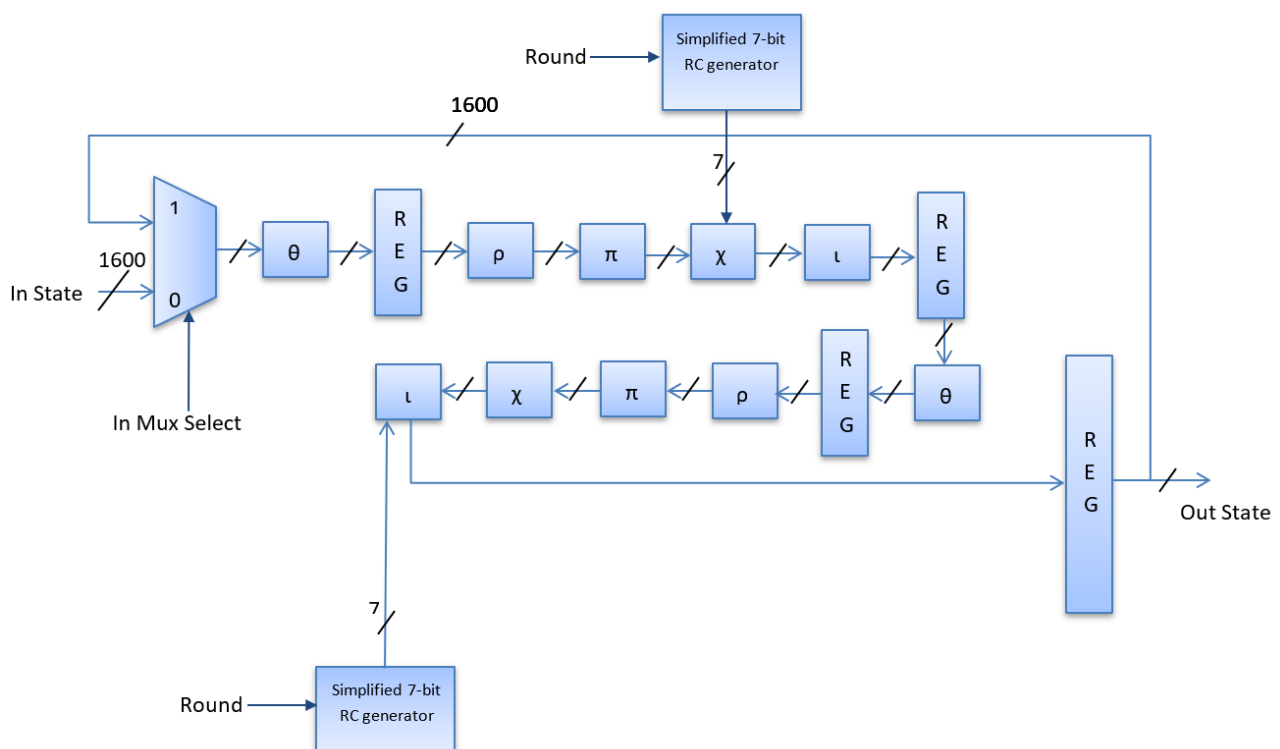


Figure 3. The proposed optimization with unrolling and pipelining techniques.

4.3.2. 7-Bit RC Generator

In this work, we have improved the RC generator in the Keccak algorithm by significantly reducing its size. Previously, the RC generator stored 24 pre-calculated constants, each with a certain length, as shown in Table 3. However, we reduce the size, which is much smaller.

Table 3. The standard 64-bit RC values.

RC_0	0000000000000001	RC_1	0000000000008082	RC_2	800000000000808A
RC_3	8000000080008000	RC_4	000000000000808B	RC_5	0000000080000001
RC_6	8000000080008081	RC_7	8000000000008009	RC_8	000000000000008A
RC_9	0000000000000088	RC_{10}	0000000080008009	RC_{11}	000000008000000A
RC_{12}	000000008000808B	RC_{13}	800000000000008B	RC_{14}	8000000000008089
RC_{15}	8000000000008003	RC_{16}	8000000000008002	RC_{17}	8000000000000080
RC_{18}	000000000000800A	RC_{19}	800000008000000A	RC_{20}	8000000080008081
RC_{21}	8000000000008080	RC_{22}	0000000080000001	RC_{23}	8000000080008008

The size reduction is achieved by storing only the non-zero bits in each RC value, as shown in Equation (7). According to the Keccak specification, as shown in Equation (8), each RC value has a maximum of seven non-zero bits.

$$A'[x, y, z] = A[x, y, z] \text{ XOR } RC[i_g] \quad (7)$$

$$RC[i_g][0][0][2^h - 1] = gc[h + 7i_g] \text{ for all } 0 \leq g \leq n \quad (8)$$

These Equations (7) and (8) highlight how the RC generator size is decreased through the retention of non-zero bits within the RC values. So, we have taken advantage of this observation and simplified the round constant values accordingly. The simplified seven-bit RC values are presented in Table 4.

Table 4. The simplified seven-bit RC values.

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

This reduction in the size of the RC simplifies the computation in the “iota” step of the Keccak algorithm. Previously, the “iota” step required 64 logical XOR operations. However, with the simplified RC, the number of XOR operations needed in the “iota” step is reduced to only seven. Specifically, the bitwise XOR operation is now performed on bit positions 0, 1, 3, 7, 15, 31, and 63 of the state array $A[0, 0]$ as shown in Table 5, according to Equation (8). These positions correspond to the non-zero bits in the simplified RC values.

Table 6 illustrates an instance of the simplified values used for RC [7]. By optimizing the RC generator and simplifying the “iota” step, we achieve a more efficient computation process in the Keccak algorithm. This improvement contributes to reducing computational overhead and enhancing the overall performance of the hash function.

Table 5. The positions with non-zero bits.

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

Table 6. Instance of the simplified values used for RC [7].

Hexadecimal	Binary	Positions with Value 1
8009	1000000000001001	15th = 1, 7th = 0, 3rd = 1, 1st = 0, 0th = 1
0000	0000000000000000	31st = 0
0000	0000000000000000	-
8000	1000000010001000	63th = 1

4.4. Truncating Process

The truncating process in the Keccak algorithm serves as the inverse operation to the mapping phase. It aims to generate a 1600-bit binary word (string) from a state represented as a three-dimensional array with dimensions $5 \times 5 \times 64$ bits. Once the 1600-bit binary word is generated, it undergoes a segmentation process to produce a digest output of the desired size. The digest output size can vary depending on the specific requirements or security level for applying the Keccak algorithm. The truncation process involves converting the three-dimensional state array into a linear sequence of bits. The $5 \times 5 \times 64$ bits are concatenated together, resulting in a 1600-bit binary word. Subsequently, the generated 1600-bit binary word is segmented or divided into smaller portions to produce the desired digest output. The segmentation process typically involves extracting a contiguous sequence of bits from the binary word, which matches the desired output size. By truncating the 1600-bit binary word and extracting the appropriate segment, the Keccak algorithm produces the final digest output, which results from applying the cryptographic hash function on the input data. The digest output can be of varying sizes, such as 224 bits, 256 bits, 384 bits, or 512 bits, depending on the specific variant of Keccak being used and the desired level of security.

5. Experimental Outcomes

In our study, we employed the Virtex-5, Virtex-6, and Virtex-7 FPGA boards to comprehensively compare the suggested strategy with other existing studies, ensuring a fair assessment. To implement the methods, we utilized the Xilinx ISE tool for the Virtex-5 and Virtex-6 designs, while the Virtex-7 design was implemented using Xilinx Vivado. The information provided in Tables 7 and 8 corresponds to the results obtained after the post-implementation stage in the FPGA design process. We want to emphasize that the post-implementation stage is critical, as it considers the complete design and provides the most accurate representation of the resources used by the design on the FPGA chip.

5.1. Verification Tests

We performed simulations and verification tests to validate our techniques' functionality. In particular, we utilized valid examples provided by the NIST [35] to verify the full functioning of our implementation. This validation process ensures that the implemented techniques are functioning as intended by correctly producing the desired outcomes, and it verifies whether the outcomes match the expected results, ensuring that the methods produce reliable and precise outputs.

5.2. Performance Metrics and Outcomes of Our Architecture

The FPGA implementation results were extensively examined to evaluate various standard performance metrics to guarantee a fair and meaningful comparison employed in the existing literature [19], including achievable frequency (maximum), area utilization,

throughput, and efficiency. Throughput [36] is a crucial measure in message hashing, as it determines the rate at which messages can be processed. Higher throughput indicates the ability to handle a greater number of messages within a given time frame, which is desirable for applications that require fast and efficient hashing.

$$Throughput_{Fpga} = \frac{\text{Bitrate size "r"}}{\text{Total clock cycles}} \times \text{Frequency maximum clock} \quad (9)$$

The achievable frequency [37] represents the maximum clock frequency the FPGA design can operate reliably. It indicates the speed at which the system can process incoming data and execute the hashing operations. A higher achievable frequency signifies improved processing capabilities and faster overall performance.

In the context of FPGA implementation, efficiency [38] assesses the ratio of useful work performed to the amount of resources utilized. It provides insights into the overall effectiveness of the design and its ability to achieve the desired objectives with minimal wastage or redundancy. Higher efficiency values signify optimized utilization of FPGA resources and improved performance.

$$Efficiency_{Fpga} = \frac{Throughput_{Fpga}}{Area_{Fpga}} \quad (10)$$

Area [39,40] utilization refers to the amount of FPGA resources consumed by the design. Lower area utilization implies more efficient utilization of FPGA resources and potentially lower manufacturing costs. These metrics are presented in Table 7, allowing for a clear comparison and analysis of the results.

Table 7. The FPGA implementation results.

FPGA	Block Size "r"	Virtex-5	Virtex-6	Virtex-7
Frequency (MHz)		272.41	344.62	396.28
Area (slices)		1186	1348	1452
Throughput (Gbps)	1152	26.151	33.084	38.043
	1088	24.699	31.246	35.929
	832	18.887	23.894	27.475
	576	13.076	16.542	19.021
Efficiency (Mbps/slices)	1152	22.05	24.54	26.20
	1088	20.83	23.18	24.74
	832	15.93	17.73	18.92
	576	11.03	12.27	13.10

As shown in Table 7, the Virtex-7 FPGA board exhibits the highest area utilization among the three boards, with 1452 slices. The Virtex-6 board follows it with 1348 slices, and the Virtex-5 board with 1186 slices. Secondly, regarding frequency, the Virtex-7 FPGA board achieves the highest value of 396.28 MHz, indicating its ability to operate at a faster clock speed. The Virtex-6 board follows closely behind 344.62 MHz, while the Virtex-5 board has the lowest frequency at 272.41 MHz. Lastly, the Virtex-7 board consistently exhibits the highest throughput and efficiency across all 'r' values, followed by Virtex-6 and Virtex-5.

5.3. Comparative Analysis with Other Equivalent Models

Table 8 displays the comparison with other equivalent models for a 512-bit output length, focusing on the frequency (MHz), area (Slices), throughput (Gbps), and efficiency (Mbps/slice) for the Keccak algorithm. All the reported outcomes are based on single-block

messages. The proposed design utilizing Virtex-5 FPGA achieves a slice count of 1186, which is lower than the slice counts of the Virtex-5 designs presented in the works [26,28–31]. Although the operating frequency of the proposed Virtex-5 design is 272.41 MHz, which is lower than the highest frequency mentioned in [26] (387 MHz), it manages to achieve a higher throughput of 13.076 Gbps compared to the other Virtex-5 designs. Additionally, the proposed Virtex-5 design demonstrates higher efficiency with a rate of 11.03 Mbps/slice, outperforming the efficiencies of the other Virtex-5 designs.

Table 8. Outcomes and comparisons for the SHA-3 algorithm of 512 output length.

Design	FPGA	Area (Slices)	Frequency (MHz)	Throughput (Gbps) “r” = 576	Efficiency (Mbps/Slices) “r” = 576
[26]	Virtex-5	1680	387	8.06	4.91
[27]	Virtex-7	1454	374.035	7.979	5.49
[28]	Virtex-5	1409	377.86	8.22	5.83
	Virtex-6	1227	424.44	10.19	8.30
[29]	Virtex-5	1388	287.39	11.50	8.48
	Virtex-6	1167	394.01	15.76	13.83
	Virtex-7	1418	414.54	16.58	11.97
[30]	Virtex-5	2652	352	8.44	6.37
	Virtex-6	2296	391	9.38	8.17
[31]	Virtex-5	2326	306	5.56	2.40
[32]	Virtex-5	1163	273	7.80	6.06
Proposed	Virtex-5	1186	272.41	13.076	11.03
	Virtex-6	1348	344.62	16.542	12.27
	Virtex-7	1452	396.28	19.021	13.10

The proposed design using Virtex-6 FPGA exhibits a higher slice count of 1348 compared to the Virtex-6 designs [28–30]. Although the proposed Virtex-6 design operates at a frequency of 344.62 MHz, which is lower than the highest frequency reported in [28] (424.44 MHz), it achieves a higher throughput of 16.542 Gbps when compared to the other Virtex-6 designs. Similarly, the proposed Virtex-6 design showcases improved efficiency with a value of 12.27 Mbps/slice, surpassing the efficiencies of the other Virtex-6 designs.

Furthermore, the proposed design utilizing Virtex-7 FPGA presents a slice count of 1452, comparable to that of the Virtex-7 designs mentioned in the works [27,29]. The operating frequency of the proposed Virtex-7 design is 396.28 MHz, surpassing the frequencies reported in the other works, which range from 374.035 MHz to 414.54 MHz. Moreover, the proposed Virtex-7 design achieves a higher throughput of 19.021 Gbps than the other Virtex-7 designs. Additionally, the efficiency of the proposed Virtex-7 design stands at 13.10 Mbps/slice, which is higher than the efficiencies of the other Virtex-7 designs. The above analysis makes it abundantly clear that the design that has been proposed demonstrates superior performance in terms of both throughput and efficiency when compared to the other designs that have been discussed in the relevant publications.

6. Discussion of Our Optimization Strategy

Cryptographic algorithms and hardware implementations have witnessed significant advancements in recent years. However, despite these advancements, certain challenges and gaps remain. One prominent issue is the ever-increasing demand for enhanced efficiency in cryptographic systems. Traditional cryptographic algorithms need help keeping up with modern adversaries’ growing computational power. Additionally, the need for

faster and more resource-efficient implementations in constrained environments, such as IoT devices and embedded systems, presents a unique set of challenges. Existing hash functions, while effective, often struggle to strike the right balance between performance and resource utilization. This creates a research gap where there is room for innovative solutions that address these challenges comprehensively.

The motivation behind our research stems from the abovementioned gap and the need for novel approaches that can bridge the divide between resource utilization and efficiency. The Keccak hash function holds promise due to its strong security properties for efficient hardware implementation. Keccak's sponge construction offers the flexibility to adapt the hash function to varying security requirements without compromising performance. FPGA technology presents an opportunity to leverage hardware acceleration to achieve efficient and customizable cryptographic implementations. However, resource constraints can make optimizing cryptographic algorithms for these devices challenging.

Our proposed approach is centred on harnessing the strengths of the Keccak algorithm and FPGA technology to address the challenges posed by the research gap. By exploring the potential of FPGA acceleration for Keccak-based cryptographic operations, we aim to provide a solution that enhances efficient cryptographic implementations. Our work contributes to the body of knowledge by demonstrating the feasibility and advantages of FPGA-based Keccak implementations. We emphasize the potential of FPGA technology in achieving a harmonious synergy between cryptographic strength and computational efficiency. Our findings open avenues for further research into optimizing and refining FPGA-based cryptographic systems with extensions to other cryptographic algorithms and applications.

7. Conclusions and Future Work

Hash functions play an essential part in the field of information security, serving various purposes in today's digital world. The significance of hash functions extends to various domains, including military, online commerce, banking, healthcare management, and the Internet of Things (IoT). Among the various hash algorithms available, the Keccak algorithm stands out for its significantly higher level of security. The Keccak algorithm provides a suitable combination of performance, acceleration, and safety, making it a preferred choice for many cryptographic applications in today's information security landscape.

The emphasis of this article is on studying the optimal performance of throughput and efficiency criteria for the Keccak algorithm across various output lengths (224, 256, 384, and 512 bits) on the Virtex-5, Virtex-6, and Virtex-7 FPGA boards. By conducting a comprehensive analysis, we compare our approach to similar plans and demonstrate that our proposed strategy achieves the highest performance in terms of the standard evaluation measures of throughput and efficiency. The highest throughput rates for a 512-bit output length were above 11.37% Gbps in Virtex-5, 10.49% Gbps in Virtex-6 and 11.47% Gbps in Virtex-7 compared with other recently equivalent models. In future work, we intend to reduce the crucial path further and enhance overall performance and performance measures per lap.

Author Contributions: Methodology, A.S.; investigation, A.S.; conceptualization, A.S.; resources, A.S.; software, A.S.; formal analysis, A.S.; project administration, A.S.; visualization, A.S. and T.S.; validation, A.S. and T.S.; writing—original draft preparation, A.S. and T.S.; writing—review and editing, A.S. supervision, M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
FPGAs	Field-Programmable Gate Arrays
NIST	National Institute of Standards and Technology
RC	Round Constant
SHA	Secure Hash Algorithm
VHDL	Very High Speed Integrated Circuit HDL

References

- Reddy, N.M.; Ramesh, G.; Kasturi, S.B.; Sharmila, D.; Gopichand, G.; Robinson, L.T. Secure data storage and retrieval system using hybridization of orthogonal knowledge swarm optimization and oblique cryptography algorithm in cloud. *Appl. Nanosci.* **2023**, *13*, 2449–2461. [\[CrossRef\]](#)
- Adeniyi, E.A.; Falola, P.B.; Maashi, M.S.; Aljebreen, M.; Bharany, S. Secure sensitive data sharing using RSA and ElGamal cryptographic algorithms with hash functions. *Information* **2022**, *13*, 442. [\[CrossRef\]](#)
- Almalki, J.; Al Shehri, W.; Mehmood, R.; Alsaif, K.; Alshahrani, S.M.; Jannah, N.; Khan, N.A. Enabling Blockchain with IoMT Devices for Healthcare. *Information* **2022**, *13*, 448. [\[CrossRef\]](#)
- Kore, A.; Patil, S. Cross layered cryptography based secure routing for IoT-enabled smart healthcare system. *Wirel. Netw.* **2022**, *28*, 287–301. [\[CrossRef\]](#)
- Khari, M.; Garg, A.K.; Gandomi, A.H.; Gupta, R.; Patan, R.; Balusamy, B. Securing data in Internet of Things (IoT) using cryptography and steganography techniques. *IEEE Trans. Syst. Man. Cybern. Syst.* **2019**, *50*, 73–80. [\[CrossRef\]](#)
- Sadeghi-Nasab, A.; Rafe, V. A comprehensive review of the security flaws of hashing algorithms. *J. Comput. Virol. Hacking Tech.* **2022**, *19*, 287–302. [\[CrossRef\]](#)
- Mishra, N.; Islam, S.H.; Zeadally, S. A comprehensive review on collision-resistant hash functions on lattices. *J. Inf. Secur. Appl.* **2021**, *58*, 102782. [\[CrossRef\]](#)
- Sravani, M.M.; Durai, S.A. Attacks on cryptosystems implemented via VLSI: A review. *J. Inf. Secur. Appl.* **2021**, *60*, 102861. [\[CrossRef\]](#)
- Nita, S.L.; Mihailescu, M.I. Hash Functions. In *Cryptography and Cryptanalysis in Java: Creating and Programming Advanced Algorithms with Java SE 17 LTS and Jakarta EE 10*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 101–112. [\[CrossRef\]](#)
- Sideris, A.; Sanida, T.; Dasygenis, M. Hardware acceleration of SHA-256 algorithm using NIOS-II processor. In Proceedings of the 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCASST), Thessaloniki, Greece, 13–15 May 2019; pp. 1–4. [\[CrossRef\]](#)
- Nakamura, K.; Hori, K.; Hirose, S. Algebraic Fault Analysis of SHA-256 Compression Function and Its Application. *Information* **2021**, *12*, 433. [\[CrossRef\]](#)
- Alagic, G.; Apon, D.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Miller, C.; Moody, D.; Peralta, R.; et al. *Status Report on the Third Round of the Nist Post-Quantum Cryptography Standardization Process*; US Department of Commerce, NIST: Gaithersburg, MD, USA, 2022. [\[CrossRef\]](#)
- Kim, Y.B.; Youn, T.Y.; Seo, S.C. Chaining optimization methodology: A new sha-3 implementation on low-end microcontrollers. *Sustainability* **2021**, *13*, 4324. [\[CrossRef\]](#)
- Braeken, A. Highly efficient symmetric key based authentication and key agreement protocol using Keccak. *Sensors* **2020**, *20*, 2160. [\[CrossRef\]](#)
- Vandervelden, T.; De Smet, R.; Steenhaut, K.; Braeken, A. SHA 3 and Keccak variants computation speeds on constrained devices. *Future Gener. Comput. Syst.* **2022**, *128*, 28–35. [\[CrossRef\]](#)
- Sideris, A.; Sanida, T.; Dasygenis, M. High throughput implementation of the keccak hash function using the nios-ii processor. *Technologies* **2020**, *8*, 15. [\[CrossRef\]](#)
- Sideris, A.; Sanida, T.; Chatzisavvas, A.; Dossis, M.; Dasygenis, M. High Throughput of Image Processing with Keccak Algorithm using Microprocessor on FPGA. In Proceedings of the 2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Ioannina, Greece, 23–25 September 2022; pp. 1–4. [\[CrossRef\]](#)
- Caba, J.; Díaz, M.; Barba, J.; Guerra, R.; Escolar, S.; López, S. Low-power hyperspectral anomaly detector implementation in cost-optimized FPGA devices. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2022**, *15*, 2379–2393. [\[CrossRef\]](#)
- Al-Odat, Z.A.; Ali, M.; Abbas, A.; Khan, S.U. Secure hash algorithms and the corresponding FPGA optimization techniques. *ACM Comput. Surv. (CSUR)* **2020**, *53*, 1–36. [\[CrossRef\]](#)
- Sideris, A.; Sanida, T.; Dasygenis, M. High throughput pipelined implementation of the SHA-3 cryptoprocessor. In Proceedings of the 2020 32nd International Conference on Microelectronics (ICM), Aqaba, Jordan, 14–17 December 2020; pp. 1–4. [\[CrossRef\]](#)
- Assad, F.; Elotmani, F.; Fettach, M.; Tragha, A. An optimal hardware implementation of the KECCAK hash function on virtex-5 FPGA. In Proceedings of the 2019 International Conference on Systems of Collaboration Big Data, Internet of Things & Security (SysCoBioTS), Casablanca, Morocco, 12–13 December 2019; pp. 1–5. [\[CrossRef\]](#)

22. Bensalem, H.; Blaqui re, Y.; Savaria, Y. An efficient OpenCL-Based implementation of a SHA-3 co-processor on an FPGA-centric platform. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *70*, 1144–1148. [[CrossRef](#)]
23. Ruiz-Rosero, J.; Ramirez-Gonzalez, G.; Khanna, R. Field programmable gate array applications—A scientometric review. *Computation* **2019**, *7*, 63. [[CrossRef](#)]
24. Mitra, J.; Nayak, T.K. An FPGA-based phase measurement system. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *26*, 133–142. [[CrossRef](#)]
25. 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.* **2021**, *24*, 524–556. [[CrossRef](#)]
26. Mestiri, H.; Barra, I. High-Speed Hardware Architecture Based on Error Detection for KECCAK. *Micromachines* **2023**, *14*, 1129. [[CrossRef](#)] [[PubMed](#)]
27. Van Hieu, D.; Khai, L.D. A Fast Keccak Hardware Design for High Performance Hashing System. In Proceedings of the 2021 15th International Conference on Advanced Computing and Applications (ACOMP), Ho Chi Minh City, Vietnam, 24–26 September 2021; pp. 162–168. [[CrossRef](#)]
28. Rao, M.; Newe, T.; Grout, I.; Mathur, A. High speed implementation of a SHA-3 core on Virtex-5 and Virtex-6 FPGAs. *J. Circuits Syst. Comput.* **2016**, *25*, 1650069. [[CrossRef](#)]
29. Kahri, F.; Mestiri, H.; Bouallegue, B.; Machhout, M. High speed FPGA implementation of cryptographic KECCAK hash function crypto-processor. *J. Circuits Syst. Comput.* **2016**, *25*, 1650026. [[CrossRef](#)]
30. Ioannou, L.; Michail, H.E.; Voyiatzis, A.G. High performance pipelined FPGA implementation of the SHA-3 hash algorithm. In Proceedings of the 2015 4th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 14–18 June 2015; pp. 68–71. [[CrossRef](#)]
31. Provelengios, G.; Kitsos, P.; Sklavos, N.; Koulamas, C. FPGA-based design approaches of keccak hash function. In Proceedings of the 2012 15th Euromicro Conference on Digital System Design, Cesme, Turkey, 5–8 September 2012; pp. 648–653. [[CrossRef](#)]
32. Sundal, M.; Chaves, R. Efficient FPGA implementation of the SHA-3 hash function. In Proceedings of the 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, 3–5 July 2017; pp. 86–91. [[CrossRef](#)]
33. Bertoni, G.; Daemen, J.; Peeters, M.; Van Assche, G. Keccak. In Proceedings of the Advances in Cryptology—EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, 26–30 May 2013; pp. 313–314. [[CrossRef](#)]
34. Homsirikamol, E.; Morawiecki, P.; Rogawski, M.; Srebrny, M. Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. In Proceedings of the Computer Information Systems and Industrial Management: 11th IFIP TC 8 International Conference, CISIM 2012, Venice, Italy, 26–28 September 2012; pp. 56–67. [[CrossRef](#)]
35. Computer Security Division, I.T.L. Example Values—Cryptographic Standards and Guidelines: CSRC. 2016. Available online: <https://nist.gov/itl/csd> (accessed on 4 April 2023).
36. Della Sala, R.; Bellizia, D.; Scotti, G. High-Throughput FPGA-Compatible TRNG Architecture Exploiting Multistimuli Metastable Cells. *IEEE Trans. Circuits Syst. Regul. I Pap.* **2022**, *69*, 4886–4897. [[CrossRef](#)]
37. Wang, J.; Zhang, T.; Zhang, B.; Jeremy-Gillbanks; Zhao, X. An Innovative FPGA Implementations of the Secure frequency hopping communication system based on the improved ZUC algorithm. *IEEE Access* **2022**, *10*, 54634–54648. [[CrossRef](#)]
38. Pham, H.L.; Tran, T.H.; Le, V.T.D.; Nakashima, Y. A high-efficiency fpga-based multimode sha-2 accelerator. *IEEE Access* **2022**, *10*, 11830–11845. [[CrossRef](#)]
39. Aljaedi, A.; Jamal, S.S.; Rashid, M.; Alharbi, A.R.; Alotaibi, M.; Alanazi, D.J. Area-Efficient Realization of Binary Elliptic Curve Point Multiplication Processor for Cryptographic Applications. *Appl. Sci.* **2023**, *13*, 7018. [[CrossRef](#)]
40. Kieu-Do-Nguyen, B.; Pham-Quoc, C.; Tran, N.T.; Pham, C.K.; Hoang, T.T. Low-Cost Area-Efficient FPGA-Based Multi-Functional ECDSA/EdDSA. *Cryptography* **2022**, *6*, 25. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.