Comparative Study of Keccak SHA-3 Implementations

cryptography

MDPI

Review

Comparative Study of Keccak SHA-3 Implementations

Alessandra Dolmeta *, Maurizio Martina and Guido Masera

Department of Electronics and Telecommunication (DET), Politecnico di Torino, 10129 Torino, Italy; maurizio.martina@polito.it (M.M.); guido.masera@polito.it (G.M.)
* Correspondence: alessandra.dolmeta@polito.it

Abstract: This paper conducts an extensive comparative study of state-of-the-art solutions for implementing the SHA-3 hash function. SHA-3, a pivotal component in modern cryptography, has spawned numerous implementations across diverse platforms and technologies. This research aims to provide valuable insights into selecting and optimizing Keccak SHA-3 implementations. Our study encompasses an in-depth analysis of hardware, software, and software–hardware (hybrid) solutions. We assess the strengths, weaknesses, and performance metrics of each approach. Critical factors, including computational efficiency, scalability, and flexibility, are evaluated across different use cases. We investigate how each implementation performs in terms of speed and resource utilization. This research aims to improve the knowledge of cryptographic systems, aiding in the informed design and deployment of efficient cryptographic solutions. By providing a comprehensive overview of SHA-3 implementations, this study offers a clear understanding of the available options and equips professionals and researchers with the necessary insights to make informed decisions in their cryptographic endeavors.

Keywords: hash function; SHA-3; Keccak; hardware design; accelerator; FPGA; ASIC; cryptography; post-quantum cryptography; HW/SW co-design

Citation: Dolmeta, A.; Martina, M.; Masera, G. Comparative Study of Keccak SHA-3 Implementations. *Cryptography* **2023**, *7*, 60. https://doi.org/10.3390/cryptography7040060

Academic Editor: Kris Gaj

Received: 16 October 2023
Revised: 7 November 2023
Accepted: 16 November 2023
Published: 20 November 2023

1. Introduction

Open contests have become a preferred method for selecting cryptographic standards in the U.S. and worldwide, beginning with the Advanced Encryption Standard (AES) contest organized by the NIST in 1997–2000. Four typical criteria taken into account in the evaluation of candidates in such contests are security, performance in software, performance in hardware, and flexibility [1]. Security, though crucial, is complex to assess quickly in contests. Hardware performance often serves as a tiebreaker when other criteria fail to declare a clear winner among cryptographic algorithms. In this survey, we focus on Secure Hash Algorithm 3 (SHA-3). It is a cryptographic hash function standard selected by the National Institute of Standards and Technology (NIST) in 2015. It emerged victorious in a rigorous competition organized by the NIST to find a successor to the aging SHA-2.

Among the contenders, Keccak, designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche, stood out for its innovative design and strong security properties, ultimately earning its place as the foundation of SHA-3. This achievement marked a significant milestone in modern cryptography, ensuring robust and efficient hash functions for various security applications. While it currently stands as the leader in resisting recent cryptanalysis attacks and excels in hardware performance, there is a continuous demand for developing an efficient implementation, be it software, e.g., Central Processing Unit (CPU), or hardware, e.g., Field-Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC). Common software implementations on a microcontroller offer high flexibility, but they may not provide the required performance for cryptographic algorithms with high computational demands. Microcontrollers are versatile and programmable, making them suitable for a wide range of applications, but they may struggle with the computational intensity of modern cryptographic algorithms. Moving up

the spectrum, an extensible processor, such as an application microprocessor, co-processors (i.e., [2]), or a Digital Signal Processor (DSP), offers more significant performance potential than fixed ones. These processors can be optimized for specific cryptographic operations, providing better throughput and efficiency than a generic microcontroller. However, they are still limited by their general-purpose architecture, which may not match the specialized requirements of specific cryptographic algorithms. A programmable datapath takes the customization a step further by allowing users to design custom hardware accelerators for cryptographic tasks. This approach offers a balance between flexibility and performance. Programmable datapaths enable the efficient execution of cryptographic algorithms through parallel processing and custom hardware instructions (i.e., [3]). Finally, the least flexible but most efficient solution is the hardwired datapath, typically implemented in ASICs (Application-Specific Integrated Circuits). ASICs are designed specifically for a particular cryptographic algorithm or set of algorithms, making them highly efficient in terms of speed and power consumption. However, their lack of flexibility means that any changes or updates to cryptographic algorithms require a new hardware design.

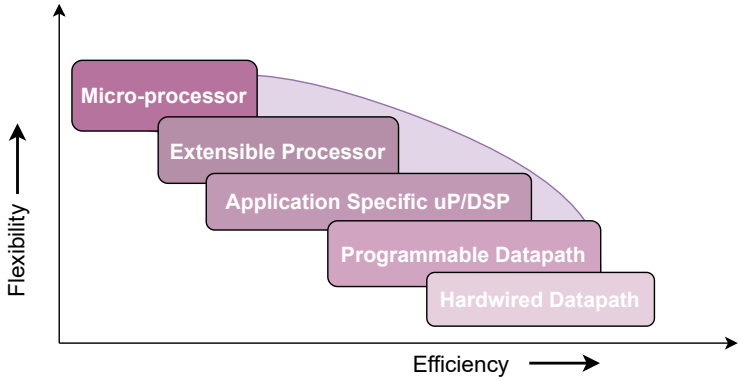

Figure 1. Space of solutions.

In summary, the choice of implementation approach for cryptographic algorithms depends on the trade-off between flexibility and efficiency, as shown in Figure 1. Selecting the most suitable implementation space depends on the specific cryptographic requirements and application constraints.

The rest of the article is organized as follows: Section 2 reports a background about SHA-3 construction. Section 3 describes Keccak structure. Section 4 is developed to report all of the solutions found in the state of the art for implementing SHA-3, compared with each other, while, in Section 6, there are comments and examinations of different hardware implementation solutions more in details; Section 6 reports implementations results, and Section 7 the conclusion of this analysis.

2. SHA-3

SHA-3 is a subset of the Keccak family standardized by the NIST. The standard lists four specific instances of SHA-3 and two extendable-output functions (SHAKE128 and SHAKE256). While the SHA-3 functions have a specified output length, the two SHAKE variants permit extraction of a variable length of output data, which makes SHA-3 a suitable candidate for pseudo-random bit generation [4]. All SHA-3 functions operate within a shared foundational framework known as the sponge construction (as shown in Figure 2a). This framework is highly adaptable and allows for the generation of hash values with variable length, making it well suited for diverse applications.

The NIST standard defines four versions of the Keccak sponge function [5] for a message M and an output length d, as shown in Table 1. The algorithm uses two parameters for the sponge construction: the bitrate with r-bits, which determines the number of bits absorbed in each step, and the capacity with c-bits, which determines the attainable security level (Figure 2a).

Table 1. SHA3 instances.

Instance	Output Size d	Rate r	Capacity c
SHA3-224	224	1152	448
SHA3-256	256	1088	512
SHA3-384	384	832	768
SHA3-512	512	576	1024

The flow of a sponge function can be understood through the following steps:

- Initialization: the sponge function is initialized depending on r and c parameters.
- Padding: The input message is padded to ensure that length is a multiple of r. Most of the architectures utilize a software scheduler for preparing the input by splitting and padding long messages into blocks of 1600 bits (multi-block messages) for truncating, if necessary, the output of the hash computation in the appropriate size of the selected mode of operation and for updating the state matrix in the case of multi-block messages. As an example, in [4,6–8] the input to the SHA-3 block is assumed to be already padded.
  In other works, i.e., [8], the hardware block is not performing only the *f-transform*, but it also has a Versioning and XOR-iring module (VSX) that is responsible for forming the appropriate state per algorithm version.
  There are some implementations in which all the steps of the sponge function are supported (padding, mapping, and truncation), but, generally, these architectures assume that the input can only be of a certain length (i.e., [9] considers input messages whose length is fixed to 64 bits), or have a precise application (i.e., [10] considers only the CRYSTALS-Kyber 768 algorithm).
- Absorbing Phase: Here, the padded message is divided into blocks of a size of r bits each, and each block is XORed with the current state of the sponge function. The resulting state is then processed through a series of bitwise operations, typically using a permutation function, to mix the input data with the current state. The function f acts on the state, with a width of $b = r + c$.
- Squeezing Phase: After all of the message blocks have been absorbed, the function produces the hash output by repeatedly extracting r bits from the state. These bits are concatenated to form the final hash value. The squeezing phase continues until the desired hash length is achieved.
- Finalization: in the end, the sponge function may perform additional operations to finalize the hash value, such as truncating it to the desired length or applying additional cryptographic transformations.

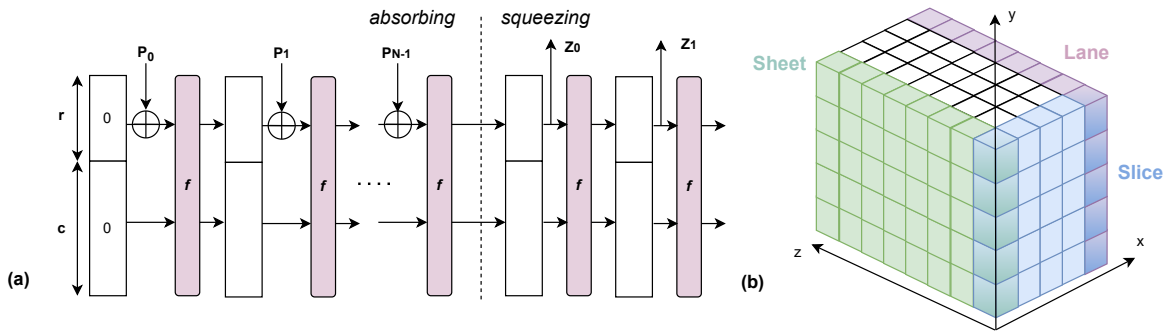

Figure 2. (**a**) Sponge Function. (**b**) Keccak State.

Central to the sponge construction is the concept of state. The state has a length of 1600 bits and consists of a three-dimensional $5 \times 5 \times 64$ table, as shown in Figure 2b. Each bit of this cube can be addressed with $A[x, y, z]$. In order to facilitate the description of the

applied functions, the following conventions are used: the part of the state that presents the word is also called a lane, a two-dimensional part of the state with a fixed z is called a slice, and all lanes with the same x-coordinate form a sheet.

3. Keccak

The most important part of the SHA-3 and SHAKE primitives is the Keccak permutation function, which calls for 24 rounds of the f-1600 function. Each round is characterized by the five consecutive steps θ, ρ, π, χ, and ι. These steps have a state array A as input and an output B, which is a processed new state array. As shown in Equation (1), θ consists of a parity computation, a rotation of one position, and a bitwise XOR.

$$\begin{aligned} \theta: \; & C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4] && 0 \le x \le 4 \\ & D[x] = C[x-1] \oplus ROT(C[x+1],1) && 0 \le x \le 4 \\ & A[x,y] = A[x,y] \oplus D[x] && 0 \le x,y \le 4 \end{aligned} \quad (1)$$

In Equation (2), ρ is a rotation by an offset that depends on the word position, and π is a permutation.

$$\rho - \pi: \quad B[y,2x+3y] = ROT(A[x,y], r[x,y]) \qquad 0 \le x,y \le 4 \quad (2)$$

In Equation (3), χ consists of bitwise XOR, NOT, and AND gates.

$$\chi: \quad A[x,y] = B[x,y] \oplus ((\overline{B[x+1,y]}) \cdot (B[x+2,y])) \quad 0 \le x,y \le 4 \quad (3)$$

Lastly, ι, in Equation (4) is a constant round addition.

$$\iota: \quad A[0,0] = A[0,0] \oplus RC \qquad 0 \le x,y \le 4 \quad (4)$$

When these five are completed, a round is completed. Table 2 reports the round constant function *RC*[i], which is a 24-value permutation that assigns 64-bit data to the Keccak function. Table 3 reports the cyclic shift offset *r*[x,y].

Table 2. Values *RC*[i] constants.

RC[0] 0x0000000000000001	*RC*[8] 0x000000000000008a	*RC*[16] 0x8000000000008002
RC[1] 0x0000000000008082	*RC*[9] 0x0000000000000088	*RC*[17] 0x8000000000000080
RC[2] 0x800000000000808a	*RC*[10] 0x0000000080008009	*RC*[18] 0x000000000000800a
RC[3] 0x8000000080008000	*RC*[11] 0x000000008000000a	*RC*[19] 0x800000008000000a
RC[4] 0x000000000000808b	*RC*[12] 0x000000008000808b	*RC*[20] 0x8000000080008081
RC[5] 0x0000000080000001	*RC*[13] 0x800000000000008b	*RC*[21] 0x8000000000008080
RC[6] 0x8000000080008081	*RC*[14] 0x8000000000008089	*RC*[22] 0x0000000080000001
RC[7] 0x8000000000008009	*RC*[15] 0x8000000000008003	*RC*[23] 0x8000000080008008

Table 3. Values r[x,y] constants.

	X = 3	X = 4	X = 0	X = 1	X = 2
Y = 2	25	39	3	10	43
Y = 1	55	20	36	44	6
Y = 0	28	27	0	1	62
Y = 4	56	14	18	2	61
Y = 5	21	8	41	45	15

More information about the Keccak algorithm can be found in [11].

4. Implementation

When developing a real implementation, a diverse array of possibilities within the design space is available. These options encompass entirely hardware-based solutions, pure software implementations, and hybrid approaches, such as Integrated Software Environments (ISE) or Application-Specific Instruction Processor (ASIP). Strictly hardware-based solutions involve dedicated IP cores, while pure software implementations rely solely on software resources. ISEs (Integrated Software Environment) or ASIP, representing a hybrid solution, enhance general-purpose processor cores with specialized hardware and instructions.

Figure 3 shows the different aspects covered in the next sections and proposes for each implementation approach a choice of proper references. Let us now delve into the intricacies of each conceivable approach.

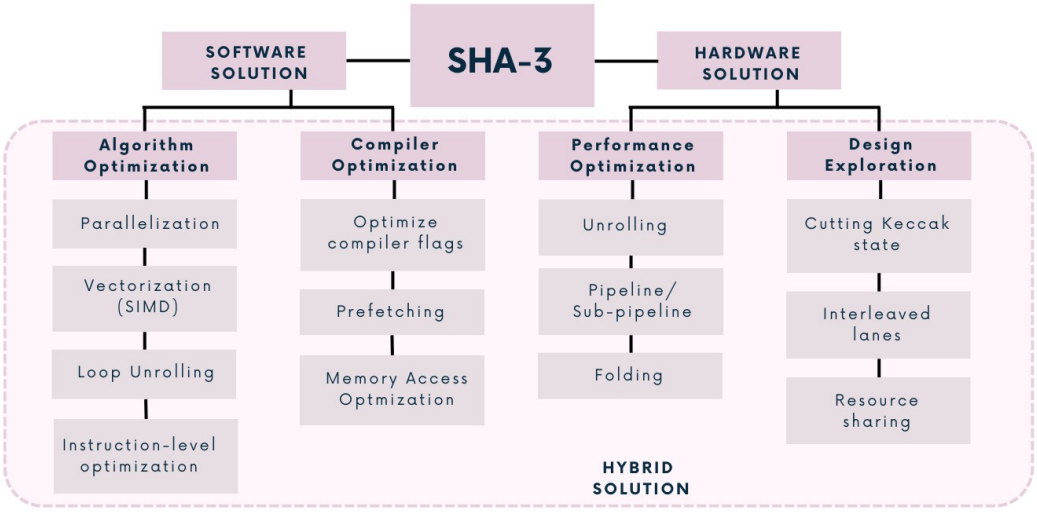

Figure 3. Implementation possibilities scheme.

4.1. Hardware Solutions

Hardware implementations of Keccak demand careful consideration of trade-offs. When implementing Keccak in hardware, the choice of design parameters and strategies heavily depends on the specific goals and constraints of the target application. These objectives typically revolve around factors such as speed, power efficiency, and area utilization. In this section, we will explore the various aspects that can be considered during the hardware implementation of Keccak, with a focus on these key parameters.

Unrolling. Unrolling is particularly efficient in improving the throughput for single-message hashing. Considering Keccak, the f-permutation block can be replicated and unrolled in the SHA-3 hash function. As an example, Refs. [6,12] implement SHA-3 considering an unrolling factor of two, while, in [7], an even higher degree of unrolling has been analyzed. Moumni et al. [9] and Nannipieri et al. [13] have made several attempts, instantiating from a single instance to twelve, going from 24 clock cycles to 2; however, this resulted in an onerous increase in area.

Pipelining. Pipelining brings the advantages of combined data throughput enhancement in multi-message hashing, where the function processes more than one message concurrently. In addition, two different types of pipelining can be distinguished:

- Classic pipelining, generally used between one round and another;
- Sub-pipelining, inserted instead between two steps of the same round.

For instance, in [8,14], the pipeline is inserted between the π and χ steps, while, in [12], it is inserted between the θ and ρ steps.

Folding. Towards a more compact SHA-3 structure, folding of the round computation can be considered. In the case of [15], each round is computed over multiple clock cycles, depending on the folding factor.

Cutting the Keccak state. The efficient management of the Keccak state is of paramount importance [16]. There are multiple alternatives, namely using slice-wise, plane-wise, and bit-interleaving techniques. Jungk et al. [17] propose a very compact slice-oriented Keccak hardware, based on the observation that all Keccak steps except ρ can be performed efficiently with slice-wise processing. However, since input messages for absorption generally arrive in a lane-oriented fashion, the plane-wise partitioning is favorable (adjacent bits in a register belong to the same lane).

Interleaved lanes. Bit-interleaving is a technique that can be used to break large 64-bit lanes of Keccak into smaller chunks [18].

Resource Sharing. Resource area sharing is a crucial optimization technique employed in hardware design, particularly in the context of FPGAs and ASICs. It aims to maximize the efficient utilization of available resources while minimizing the overall hardware footprint, which can lead to cost savings, improved performance, and reduced power consumption. An interesting example is the co-processor presented in [2], named AE$HA-3, which combines two of the NIST's standardized algorithms, i.e., Advance Encryption Standard (AES) and SHA-3. Maache et al. [3] also present a multi-purpose cryptographic system performing both AES and SHA-3, implementing it on the IntelFPGA Cyclone-V device.

To sum up, the hardware implementation of Keccak is a multifaceted task that necessitates careful consideration of various trade-offs and objectives. The specific design choices will be heavily influenced by the unique demands of the target application, whether it be a high-performance cryptographic accelerator, a low-power embedded system, or any other use case in which Keccak is employed. Each implementation will strike a balance between speed, power efficiency, area utilization, and security to meet its intended purpose effectively.

4.2. Software Solutions

Enhancing the software implementation of an algorithm holds the key to unlocking superior performance. By optimizing code, leveraging hardware-specific features, and minimizing resource overhead, software improvements can significantly boost algorithmic efficiency, resulting in faster execution and better utilization of available hardware resources. Accelerating the SHA-3 algorithm on FPGA devices, RISC-V, or ARM without dedicated hardware accelerators involves optimizing the software implementation to maximize performance. Here are some techniques to achieve this.

Parallelization. Using parallelization for implementing multi-threading or multi-processing when having multiple CPU cores can significantly improve performances. Each core can work on a separate chunk of data, improving overall throughput. Pereira et al. [19] present a technique for parallel processing on Graphics Processing Units (GPUs) of the Keccak hash algorithm. They provide the core functionality, and the evaluation is performed on a Xilinx Virtex 5 FPGA.

Vectorization (SIMD). Utilize the SIMD (Single Instruction, Multiple Data) instructions available in modern processors (e.g., ARM NEON, RISC-V RVV) to process multiple data elements in parallel. This can significantly speed up the hashing process, especially when dealing with large datasets. For example, Ref. [20] proposes a set of six custom instructions for Keccak-f, p [1600, 800, 400, 200] primitives, and, similarly to other crypto-instructions (e.g., Intel AES-NI and SHA), they exploit the wide SIMD (Single Instruction, Multiple Data) registers. Li et al. [21] explore the full potential of parallelization of Keccak-f[1600] in RISC-V-based processors through custom vector extensions on 32-bit and 64-bit architectures.

Loop Unrolling. Unroll loops in the SHA-3 algorithm code to reduce loop overhead and enable the compiler to optimize the code more effectively. This can result in faster execution, especially on CPUs with pipelined execution units. Ref. [22] reports several analyses about security versus area versus the timing of PQC decapsulation algorithms, after loop unrolling, showing how, in most cases, this brings a significant reduction in latency.

Instruction-Level Optimization. Hand-tune critical sections of the code to use processor-specific instructions and features. This may include using assembly language or

intrinsic to access specialized instructions for SHA-3 operations. Ref. [23] presents two new techniques for the fast implementation of the Keccak permutation on the A-profile of the Arm architecture: the elimination of explicit rotations in the Keccak permutation through barrel shifting, and the construction of hybrid implementations concurrently leveraging both the scalar and the Neon instruction sets of AArch64.

Optimized Compiler Flags. Use compiler optimization flags (`-O2`, `-O3`, i.e., in [24]) to instruct the compiler to apply various optimizations, including loop unrolling, inline function expansion, and instruction scheduling. Ref. [25] uses `-On` command line flags during GCC compilation to improve performance at the cost of increased compilation times.

Memory Access Optimization. Minimize memory access latency by optimizing data structures and memory access patterns. Cache-friendly data structures and efficient memory layouts can reduce the number of cache misses. Choi et al. [26] discuss optimizations, memory management strategies, and parallelization schemes, aiming to enhance the performance and throughput of SHA-3 operations on graphics processing units (GPUs).

Prefetching. Use prefetching techniques to load data into the cache before it is actually needed, reducing memory access stalls and improving data processing speed. Lee et al. [27], using NVIDIA GPU, exploit the feature for which arithmetic instructions and memory load/store instructions can be executed concurrently, as long as there is no dependency between the executing instruction and data being loaded/stored. They prefetch the input data of Keccak before XORing it into the state, so that address calculation and bitwise XOR operation can run in parallel with the memory copy operation.

The effectiveness of these techniques will depend on the specific platform, compiler, and workload, so thorough testing and profiling are essential to achieve optimal results. Continuously profiling and benchmarking the software implementation will help identify performance bottlenecks and areas for improvement. This iterative process can lead to significant performance gains.

4.3. Hybrid Solutions

Hybrid solutions, which combine both software and hardware components, represent a versatile approach to solving complex problems by harnessing the strengths of each domain. These solutions essentially encompass all of the techniques discussed in the previous section and merge them into a cohesive, integrated system.

In the realm of technology and problem-solving, software and hardware have traditionally been seen as separate entities. Software provides flexibility and adaptability, while hardware offers raw processing power and efficiency. A hybrid solution brings together the computational capabilities of hardware and the logic and adaptability of software to create a powerful and agile system. It allows optimization of the performance by distributing tasks between software and hardware according to their respective strengths. This means that computationally intensive tasks can be offloaded to dedicated hardware accelerators, while software can handle tasks that require flexibility and frequent updates. This balance ensures that the system operates efficiently without bottlenecks. Moreover, being that software is inherently adaptable, it is easier to implement changes and updates to meet evolving requirements. Fritzmann et al. [4] present RISQ-V, an enhanced RISC-V architecture that integrates a set of powerfully coupled accelerators. Here, hardware/software co-design techniques have been combined to develop complex and highly customized solutions, designing tightly and loosely coupled accelerators and Instruction Set Architecture (ISA) extensions. This is an example of how combining the hardware and software provides the flexibility to adjust algorithms, logic, or functionality in response to changing needs while maintaining the stability and speed of the hardware.

5. Micro-Architecture Details

The Keccak permutation, which consists of five main steps (θ, ρ, π, χ, and ι), is designed to be highly parallelizable, which means that the order of these steps can affect synthesis results and performance on an FPGA. The reasons are manifold.

Different FPGA architectures have distinct types and quantities of resources, and, therefore, the order of the permutation steps can influence how these resources are utilized. Additionally, the timing requirements of the FPGA may vary depending on the order of the steps. Certain steps may introduce longer or shorter critical paths, impacting the maximum achievable clock frequency. Some steps of the Keccak permutation are inherently more parallelizable than others. For example, the θ and χ steps can be parallelized more easily than the ρ step, which involves shifting bits. The order in which these steps are arranged can determine how effectively parallelism can be exploited, affecting overall performance.

Obviously, FPGA synthesis tools must be considered too, since they employ various algorithms and optimization techniques to map the design onto the target FPGA.

Considering these factors, it is useful to explore different combinations of the permutation steps during FPGA design to find the optimal arrangement for a given FPGA platform. Some syntheses have been performed on Cyclone V (5CEFA9F31C8), with a time constraint of 6 ns. In this analysis, only performance is considered. As can be observed from Table 4, the best results are in cases 2, 3, 4, 7, 8, and 9, which have in common the sequence $\chi - \iota - \theta$. Obviously, changing the rotation order disrupts the dependencies between steps, requiring additional clock cycles to ensure data integrity and correctness, thus increasing the overall processing time by at least one clock cycle (as can be seen from the fifth column of Table 4).

Table 4. Analysis of Keccak permutation steps.

Case	Rotation Order	F [MHz]	Latency [µs]	Number of Iterations	% with Respect to Case 1
1	$\theta - \rho - \pi - \chi - \iota$	168	0.143	24	
2	$\rho - \pi - \chi - \iota - \theta$	180	0.139	25	−2.78%
3	$\pi - \chi - \iota - \theta - \rho$	184	0.136	25	−4.89%
4	$\chi - \iota - \theta - \rho - \pi$	183	0.137	25	−4.37%
5	$\iota - \theta - \rho - \pi - \chi$	165	0.15	25	6.06%
6 *	$\theta - \rho - \pi - \chi - \iota$	84	0.143	12	0.00%
7 *	$\rho - \pi - \chi - \iota - \theta$	93.84	0.139	13	−3.03%
8 *	$\pi - \chi - \iota - \theta - \rho$	94.77	0.137	13	−3.98%
9 *	$\chi - \iota - \theta - \rho - \pi$	95.06	0.137	13	−4.27%
10 *	$\iota - \theta - \rho - \pi - \chi$	72.83	0.18	13	24.95%

* double instance of the round unit (unrolling factor = 2).

Moreover, there are different possible options for what concerns implementation of the round constant generator (constants reported in Table 2). There are three plausible implementations. First, a circuit constructed using Linear Feedback Shift Register (LFSR) can be employed to perform on-the-fly generation of the round constant values. Another solution is storing all of the 24 pre-calculated round constants of a 64-bit length in memory forms (such as registers or circular buffers) and transferring them to the ι module via a multiplexer [8]. Since efficient resource utilization is vital for hardware implementations, it has been proved that the length of the *RC* values can be reduced to less than a byte size (see Table 5) by storing only the non-zero bits in each of the round constant values [12,28]. This will also save 55–56 XORs in the ι step.

Table 5. Simplified round constants.

RC[0] 0x01	*RC*[6] 0xf1	*RC*[12] 0x7b	*RC*[18] 0x2a
RC[1] 0x32	*RC*[7] 0xa9	*RC*[13] 0x9b	*RC*[19] 0xca
RC[2] 0xba	*RC*[8] 0x1a	*RC*[14] 0xb9	*RC*[20] 0xf1
RC[3] 0xe0	*RC*[9] 0x18	*RC*[15] 0xa3	*RC*[21] 0x90
RC[4] 0x3b	*RC*[10] 0x69	*RC*[16] 0xa2	*RC*[22] 0xf1
RC[5] 0x41	*RC*[11] 0x4a	*RC*[17] 0x90	*RC*[23] 0e8

In addition, a combinatorial circuit using a series of logical gates can be realized to compute the round constant value upon every iteration. For example, the counter value, which was used within SHA-3 to keep track of the number of iterations, was used as input to calculate on-the-fly *RC* values.

$$\begin{aligned}
RC[0] &= A'C + CE' + BDE' + A'B'D'E' \\
RC[1] &= BD'E' + BDE + BCD' + AC'E' + AC'D + B'C'DE' + A'CD'E' + A'B'C'D'E \\
RC[2] &= 0 \\
RC[3] &= BC' + BD' + BE' + C'DE' + AC'D + A'CD'E' + B'CDE \\
RC[4] &= BD' + A'CE' + ACD' + C'D'E + A'B'DE' \\
RC[5] &= A'DE' + A'CE' + CDE + AC'E' + ACD' + A'B'C'E \\
RC[6] &= C'DE + BC'D + ADE + ACE' + B'CDE' + BCD'E' + A'B'CD'E \\
RC[7] &= AD' + AE + A'B'D + A'CD + BCE
\end{aligned} \tag{5}$$

In particular, considering A as the MSB of the counter and E as the LSB, seven logical expressions were constructed to obtain the constants shown in Table 5. This solution has been synthesized by targeting an ASIC on 65 nm technology with a time constraint of 0.73 ns. Area results are reported in Table 6: the second and third solutions provide a slight improvement in terms of area occupancy without changing the critical path of the entire circuit.

Table 6. Round constant generator analysis.

Case	Total Cell Area [µm^2]	Combinational Area [µm^2]	Noncombinational Area [µm^2]
RC-64bit	55,293.48	42,285.24	13,008.24
RC-8bit	55,061.28	42,015.96	13,045.32
RC-8bit *	54,997.20	41,954.40	13,042.80

* evaluated on the fly.

These kinds of exploration help achieve the best balance between resource utilization, performance, and power consumption, ultimately leading to a more efficient and effective hardware implementation of the Keccak permutation for a specific FPGA target or a particular technology. Additionally, it allows designers to adapt the design to different platforms without starting from scratch, saving time and effort in the development process.

6. Results

6.1. Stand-Alone Solutions

Regarding FPGA implementation, there are four main parameters to take into account:

- Maximum achievable frequency: the maximum clock frequency with which the FPGA design can operate. It indicates the speed at which the system can work. The higher it is, the faster the overall performance.
- Area: the amount of FPGA resources consumed by the design.
- Throughput: the rate at which a message can be processed. The higher it is, the higher the number of messages that can be handled in the same amount of time.
- Efficiency: the whole effectiveness of the design. The higher it is, the more the FPGA utilization resources and performance are improved.

Tables 7 and 8 both report the results gathered from the different works found in the state of the art.

Table 7. Unrolling and pipeline on FPGA.

Reference	Unrolling Factor	Pipeline Factor	Device	Frequency [MHz]	Area [Slices]	Throughput [Gbps]	Efficiency [Mbps/Slices]
[6]	1	1	Virtex-6	412	1115	9.888	8.868
[6]	2	2	Virtex-6	391	2296	18.768	8.174
[7]	3	3	Virtex-6	391	3965	28.152	7.000
[7]	4	4	Virtex-6	392	4117	37.63	9.141
[12]	2	-	Virtex-6	45	2145	2.16	1.00
[12]	2	2	Virtex-6	85	1416	4.08	2.88
[12]	2	2 (* 2)	Virtex-6	344	1406	16.51	11.47
[8]	-	- (* 1)	Virtex-6	397	1649	19.1	11.6
[15]	1	1	Virtex-5	223	1192	5.35	4.49
[15]	2	1	Virtex-5	273	1163	7.8	6.06
[9]	1	1	Virtex-6	413.77	1432	9.93	6.93
[9]	12	1	Virtex-6	57.91	15,579	16.67	1.07
[28]	2	1 (* 2)	Virtex-6	344.62	1348	16.54	12.27

* Sub-pipelining.

The throughput of a hash function is evaluated as:

$$Throughput = \frac{(\#bits) \times Frequency}{c} \tag{6}$$

In Table 7, the cases in which the unfolding and pipelining/sub-pipelining techniques described in Section 4 are used are examined. In Table 8, on the other hand, the results obtained with straightforward implementations are reported.

Generally, when multiple throughput results are given, SHA3-512 is taken as a reference. However, many of the examples do not provide the results on the different instances of SHA3 or only provide the results of one of the primitives (which may not be the reference one). For further clarification on the type of primitive performed and the length of the input messages absorbed, please refer to the proper reference paper. Obviously, efficient implementation selection does not hinge solely on architectural design; it also depends on the particulard FPGA platform in use.

Based on the results of Table 7, sub-pipelining is observed to be effective in critical path reduction, while unrolling with pipelining enables simultaneous processing in the SHA-3 hash function [12]. In general, both of the techniques bring positive effects on the throughput performance. As previously stated, Table 8 shows the results obtained with straightforward implementations. Since these are standard implementations of the algorithm, and since we have previously described all of the possible implementations that are generally used in Sections 3 and 4, more detailed descriptions of individual values are not given here. The Table is provided in order to be able to compare the different solutions.

The most relevant implementations mentioned in the previous sections are also reported in the plot of Figure 4, where the efficiency (expressed in terms of Mbit per second over the number of occupied slices) is shown together with the complexity (mentioned in terms of slices).

Table 8. Straightforward implementations results.

Reference	Device	Frequency [MHz]	Area [Slices]	Throughput [Gbps]	Efficiency [Mbps/Slices]
[29]	Virtex-4	143	2024	6.07	3.0
[30]	Virtex-5	285	2573	5.7	2.21
[31]	Virtex-5	259	1370	10.77	7.69
[32]	Virtex-5	277	1236	6.64	5.37
[33]	Virtex-5	265	448	0.05	0.11
[33]	Virtex-5	122	1330	5.2	3.9
[34]	Virtex-5	137	1647	2.39	1.45
[35]	Virtex-5	271	1414	12.3	8.68
[14]	Virtex-5	317	4793	12.68	2.71
[36]	Virtex-5	189	1117	0.085	0.42
[37]	Virtex-5	277	1217	12.56	10.32
[38]	Virtex-5	248	134	0.25	1.16
[39]	Virtex-5	282	192	81.8	4.32
[40]	Virtex-5	520	151	0.501	3.32
[41]	Virtex-5	313	1304	7.51	5.75
[42]	Virtex-6	291	1015	6.99	6.89
[43]	Virtex-6	299	106	0.136	1.28
[44]	Virtex-6	286	1207	12.98	10.75
[2]	Virtex-6	328.15	1380	8400.64	2.45
[45]	Virtex-6	195	1048	8.830	8.42
[46]	Virtex-6	197	397	1.071	2.19
[47]	Virtex-6	311	91	0.203	2.23
[48]	Virtex-6	291	1.015	6.99	6.89
[49]	Virtex-6	285	188	0.077	0.41
[48]	Virtex-7	255	1.039	6.12	5.88
[50]	Virtex-7	299	983	7.17	7.27
[8]	Virtex-7	434	1618	20.8	12.9
[28]	Virtex-7	396	1452	19.021	13.10
[51]	Virtex-7	374	1454	7.979	5.49
[52]	Virtex-7	414	1418	16.58	11.97

The utilization of distinct FPGA models—specifically Virtex 5, Virtex 6, and Virtex 7—clearly makes the whole comparison approximate. Each of these boasts slices corresponding to the same quantity of Lookup Tables (LUTs), which is set at four. However, the crux of the matter lies in the fact that, despite these FPGA models sharing an identical number of inputs, the LUTs within them possess varying numbers of outputs. Consequently, the findings presented in Figure 4 should be interpreted with caution, given the subtle yet consequential differences in LUT configurations across the three FPGA models under examination. Additional works exist that have not been incorporated into this analysis because they employed distinct FPGA architectures and evaluation metrics and were not included in this comparison table due to the complexities involved in conducting an equitable comparison with the other showcased studies.

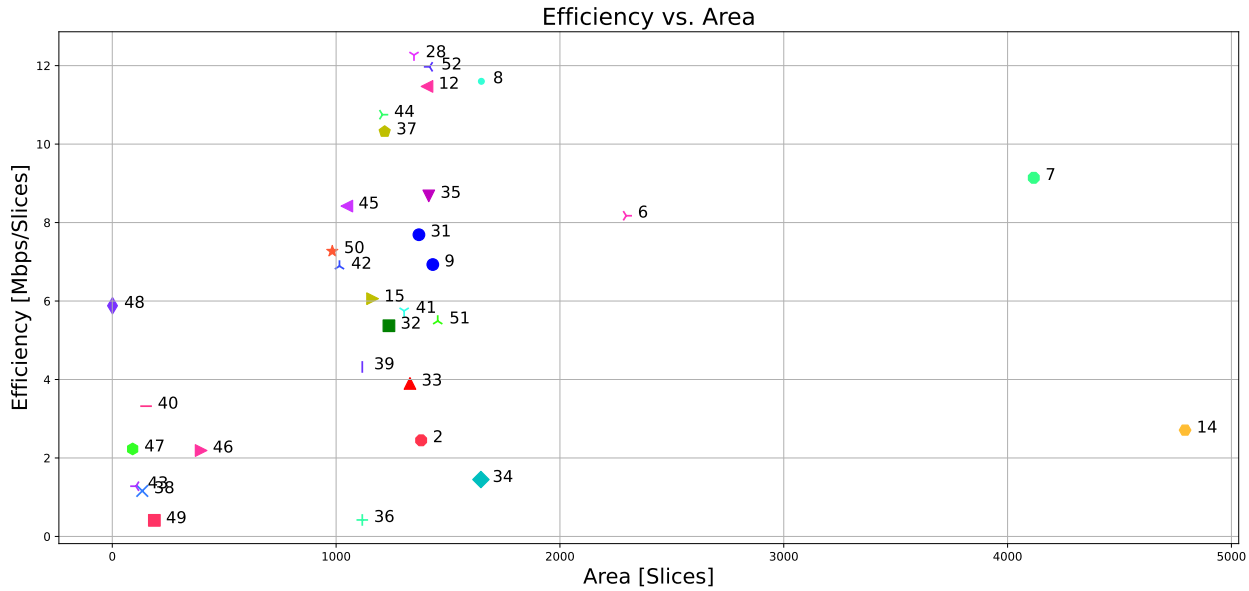

Figure 4. Efficiency vs. area scheme. The different icons and numbers must be referred to citations presented in the Bibliography.

In this application, power considerations are not typically the primary focus. Instead, the predominant emphasis lies in enhancing performance, encompassing throughput, latency, and frequency, alongside the resultant increase in area. Regrettably, relatively few studies delve into power analysis. Furthermore, comparing these sporadic power results proves challenging, as the analysis may occur post-synthesis, post-layout, or post-place and route, accounting for diverse environmental conditions and technology variations. Kundi et al. [53] propose a low-power SHA-3 design using an embedded digital signal processing slice on FPGA. Their compact design on Kintex-7 consumed 88 mW of power which is 1/7th and 1/9th of the designs in [54] and [34], respectively. Ref. [55] presents the hardware implementation of a cryptographic accelerators suite, which comprises a SHA3 unit. Considering the post-synthesis power consumption on 7 nm technology, the SHA3-unit consumes about 72 mW.

6.2. Integrated Solution

When defining integrated solutions, there are a series of slightly different metrics to be considered with respect to the stand-alone implementations. Cycle count is a measure of the complexity as well as the energy consumption of an implementation on a specific processor. It is also inversely proportional to the throughput, which describes how fast the algorithm works for a given clock frequency.

In [4], they develop an alternative solution for Keccak that presents a trade-off between performance and area. To avoid a high access rate to the main memory, the state is not split into multiple parts. However, instead of a standalone loosely coupled Keccak accelerator, they design a hardware accelerator for a single round of the Keccak permutation, using RISC-V Floating-Point Register Set (FPR) and a part of the General-Purpose Register Set (GPR). The performance is shown in Table 9.

Table 9. Clock cycles performance.

Reference	Platform	Function	Software Version	Accelerated Version	Type
[4]	PULPino	SHAKE256	31,907	308	Tightly-coupled
[24]	PULPissimo	Keccak	26,529	1348	Loosely-coupled
[56]	LEON3	Keccak-512	188,708	100,663	ASIP

The results show that their design outperforms the uniform and binomial sampling of [57], which uses a loosely coupled high-performance Keccak implementation, which is able to calculate two Keccak rounds in one clock cycle. However, the large communication overhead poses a substantial drawback to their architecture. In [24], a Keccak accelerator is proposed to speed up SHA3 computations for post-quantum cryptographic algorithms on the RISCV-based advanced microcontroller PULPissimo (more information about the microcontroller can be found in [58]). Differently from the previous work, the accelerator is driven in a memory-mapped fashion style and attached to the SoC through an AXI plug. This makes the accelerator more versatile and easier to integrate than the previous ones but at the price of a higher overhead when data need to be exchanged from the processor to the accelerator. Ref. [56] describes an FPGA-based acceleration of SHA-3 on a 32-bit processor, supporting Keccak-224, Keccak-256, Keccak-384, and Keccak-512. The LEON3 processor has been used, achieving around an 87% reduction in execution cycles.

In [59], the Keccak performance on ARMv7-M has been optimized thanks to two kinds of optimizations: the first one consists of taking advantage of the inline barrel shifter in order to remove explicit rotations in the linear layer, and the second consists of a more efficient memory access scheduling to avoid pipeline hazards. Ref. [60] presents instead a hardware implementation of an SHA-3 Application-Specific Instruction Set Processor. A custom implementation of the 32-bit MIPS ISA is developed, where a subset of the MIPS instructions are implemented with the aid of Codasip Studio, and both the C-language compiler and the HDL design are generated and tested. Afterward, two ISEs are advanced to resolve the SHA-3 bottlenecks. Ref. [61] implements algorithm-specific ISEs based on finite state machines for address generation, Lookup Table integration, and extension of computational units through microcode instructions. Rawat et al. [16] propose a set of six custom instructions to support a broad range of Keccak-based cryptographic applications in the context of an ARMv7 micro-architecture. Since the results are reported in terms of instructions/byte for various Keccak modes, they are not inserted into Table 10. It can be briefly commented that with SHA-3 (c = 1024), they obtain a speed-up of $\times$2.2.

Table 10. Clock cycles/bytes performance.

Reference	Function	Processor	Clock Cycles/Bytes	Technique
[59]	SHA3-224 SHA3-256 SHA3-384 SHA3-512	ARMv7-M4	72.4 77 97.9 138.4	ISE
[60]	SHA3	32-MIPS	137.9	Co-processor ISE
[61]	Keccak	PIC24	188	ISE

7. Conclusions

In conclusion, this paper has undertaken a thorough and insightful journey into the world of SHA-3 implementations, examining a diverse array of solutions across hardware, software, and hybrid domains. By critically evaluating their strengths, weaknesses, and performance metrics, we have contributed valuable knowledge to the field of modern cryptography. Our research has shed light on the critical factors that shape the selection and optimization of Keccak SHA-3 implementations, emphasizing computational efficiency, scalability, and flexibility in various use cases. Through a meticulous analysis of speed and resource utilization, we have provided a comprehensive view of how these implementations fare in real-world scenarios.

The outcomes of this study have far-reaching implications, enhancing the understanding of cryptographic systems and facilitating the design and deployment of efficient solutions. Professionals and researchers alike are now better equipped to make informed decisions when navigating the intricate landscape of SHA-3 implementations, ultimately contributing to the advancement of secure and resilient cryptographic practices.

As the realm of cryptography continues to evolve, our commitment to rigorous evaluation and informed decision-making remains paramount. This research stands as a testament to our dedication to strengthening the foundations of cryptographic knowledge, ensuring that we continue to safeguard the digital world with ever more robust and efficient cryptographic solutions.

Author Contributions: Conceptualization, A.D.; methodology, A.D. and G.M.; investigation, A.D.; writing—original draft preparation, A.D.; writing—review and editing, A.D., G.M. and M.M.; supervision, M.M. and G.M.; project administration, M.M. and G.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union-NextGenerationEU.

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

References

1. Homsirikamol, E.E.A. Comparing Hardware Performance of Round 3 SHA-3 Candidates using Multiple Hardware Architectures in Xilinx and Altera FPGAs. In Proceedings of the Ecrypt II Hash Workshop, Tallinn, Estonia, 19–20 May 2011.
2. Kundi, D.E.S.; Khalid, A.; Aziz, A.; Wang, C.; O'Neill, M.; Liu, W. Resource-Shared Crypto-Coprocessor of AES Enc/Dec With SHA-3. *IEEE Trans. Circuits Syst. Regul. Pap.* **2020**, *67*, 4869–4882. [CrossRef]
3. Maache, A.; Kalache, A. Design and Implementation of a flexible Multi-purpose Cryptographic System on low cost FPGA. *Int. J. Electr. Comput. Eng. Syst.* **2023**, *14*, 45–58. [CrossRef]
4. Fritzmann, T.E.A. RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *4*, 239–280. [CrossRef]
5. Dang, Q. *Recommendation for Applications Using Approved Hash Algorithms*; US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2008.
6. Ioannou, L.; Michail, H.E.; Voyiatzis, A.G. High performance pipelined FPGA implementation of the SHA-3 hash algorithm. In Proceedings of the 4th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 14–18 June 2015; pp. 68–71. [CrossRef]
7. Michail, H.E.; Ioannou, L.; Voyiatzis, A.G. Pipelined SHA-3 Implementations on FPGA: Architecture and Performance Analysis. In Proceedings of the Second Workshop on Cryptography and Security in Computing Systems (CS2 '15), Amsterdam, The Netherlands, 19–21 January 2015; pp. 13–18. [CrossRef]
8. Athanasiou, G.S.; Makkas, G.P.; Theodoridis, G. High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm. In Proceedings of the 2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP), Athens, Greece, 21–24 May 2014; pp. 538–541. [CrossRef]
9. Moumni, S.E.; Fettach, M.; Tragha, A. High Throughput Implementation of SHA3 Hash Algorithm on Field Programmable Gate Array (FPGA). *Microelectron. J.* **2019**, *93*, 104615. [CrossRef]
10. Dolmeta, A.; Martina, M.; Masera, G. Hardware architecture for CRYSTALS-Kyber post-quantum cryptographic SHA-3 primitives. In Proceedings of the 2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Valencia, Spain, 18–21 June 2023; pp. 209–212. [CrossRef]
11. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V. The keccak reference. *Submiss. Nist. Round 3* **2011**.
12. Wong, M.M.; Haj-Yahya, J.; Sau, S.; Chattopadhyay, A. A New High Throughput and Area Efficient SHA-3 Implementation. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018. [CrossRef]
13. Nannipieri, P.; Bertolucci, M.; Baldanzi, L.; Crocetti, L.; Di Matteo, S.; Falaschi, F.; Fanucci, L.; Saponara, S. SHA2 and SHA-3 accelerator design in a 7 nm technology within the European Processor Initiative. *Microprocess. Microsystems* **2021**, *87*, 103444. [CrossRef]
14. Mestiri, H.; Kahri, F.; Bedoui, M.; Bouallegue, B.; Machhout, M. High throughput pipelined hardware implementation of the KECCAK hash function. In Proceedings of the 2016 International Symposium on Signal, Image, Video and Communications (ISIVC), Tunis, Tunisia, 21–23 November 2016; pp. 282–286. [CrossRef]
15. 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]
16. Rawat, H.; Schaumont, P. Vector Instruction Set Extensions for Efficient Computation of Keccak. *IEEE Trans. Comput.* **2017**, *66*, 1778–1789. [CrossRef]
17. Jungk, B.; Apfelbeck, J. Area-Efficient FPGA Implementations of the SHA-3 Finalists. In Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 30 November–2 December 2011; pp. 235–241.
18. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V.; Keer, R.V. *KECCAK Implementation Overview*; 2012. Available online: https://keccak.team/index.html (accessed on 15 October 2023).

19. Pereira, F.; Ordonez, E.; Sakai, I.; de Souza, A. Exploiting Parallelism on Keccak: FPGA and GPU comparison. *Parallel Cloud Comput.* **2013**, *2*, 1–6.
20. Rawat, H.K.; Schaumont, P. SIMD Instruction Set Extensions for Keccak with Applications to SHA-3, Keyak and Ketje. In Proceedings of the Hardware and Architectural Support for Security and Privacy 2016 (HASP '16), Seoul, Korea, 18 June 2016; Article 4, pp. 1–8. [CrossRef]
21. Li, H.; Mentens, N.; Picek, S. Maximizing the Potential of Custom RISC-V Vector Extensions for Speeding up SHA-3 Hash Functions. In Proceedings of the 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 17–19 April 2023; pp. 1–6. [CrossRef]
22. Basu, K.; Soni, D.; Nabeel, M.; Karri, R. NIST Post-Quantum Cryptography- A Hardware Evaluation Study. *Cryptol. ePrint Arch.* **2019**. Available online: https://eprint.iacr.org/2019/047 (accessed on 15 October 2023).
23. Becker, H.; Kannwischer, M.J. Hybrid Scalar/Vector Implementations of Keccak and SPHINCS$^+$ on AArch64. In Proceedings of the International Conference on Cryptology in India, Kolkata, India, 11–14 December 2022; Isobe, T., Sarkar, S., Eds.; Springer: Cham, Switzerland, 2022; pp. 272–293.
24. Dolmeta, A.; Mirigaldi, M.; Martina, M.; Masera, G. Implementation and integration of Keccak accelerator on RISC-V for CRYSTALS-Kyber. In Proceedings of the 20th ACM International Conference on Computing Frontiers (CF '23), Bologna, Italy, 9–11 May 2023; pp. 381–382. [CrossRef]
25. Malik, A.; Aziz, A.; Kundi, D.E.S.; Akhter, M. Software implementation of Standard Hash Algorithm (SHA-3) Keccak on Intel core-i5 and Cavium Networks Octeon Plus embedded platform. In Proceedings of the 2013 2nd Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 15–20 June 2013; pp. 79–83. [CrossRef]
26. Choi, H.; Seo, S.C. Fast Implementation of SHA-3 in GPU Environment. *IEEE Access* **2021**, *9*, 144574–144586. [CrossRef]
27. Lee, W.K.; Phan, R.C.W.; Goi, B.M.; Chen, L.; Zhang, X.; Xiong, N.N. Parallel and High Speed Hashing in GPU for Telemedicine Applications. *IEEE Access* **2018**, *6*, 37991–38002. [CrossRef]
28. Sideris, A. A Novel Hardware Architecture for Enhancing the Keccak Hash Function in FPGA Devices. *Information* **2023**, *14*, 475. [CrossRef]
29. Akin, A.; Aysu, A.; Ulusel, O.C.; Savas, E. Efficient Hardware Implementations of High Throughput SHA-3 Candidates Keccak, Luffa, and Blue Midnight Wish for Single- and Multi-Message Hashing. In Proceedings of the 3rd International Conference on Security of Information and Networks, Taganrog, Russia, 7–11 September 2010; SIN'10. pp. 168–177.
30. 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]
31. Mestiri, H.; Barraj, I. High-Speed Hardware Architecture Based on Error Detection for KECCAK. *Micromachines* **2023**, *14*, 1129. [CrossRef] [PubMed]
32. Gaj, K.; Homsirikamol, E.; Rogawski, M. Comprehensive Comparison of Hardware Performance of Fourteen Round 2 SHA-3 Candidates with 512-bit Outputs Using Field Programmable Gate Arrays. In Proceedings of the 2nd SHA-3 Candidate Conference, Santa Barbara, CA, USA, 23–24 August 2010; pp. 1–14.
33. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V. Keccak Sponge Function Family Main Document. 2009.
34. Honda, T.; Guntur, H.; Satoh, A. FPGA implementation of new standard hash function Keccak. In Proceedings of the 2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE), Tokyo, Japan, 7–10 October 2014; pp. 275–279. [CrossRef]
35. Jararweh, Y.; Tawalbeh, L.; Tawalbeh, H.; Mod'd, A. Hardware Performance Evaluation of SHA-3 Candidate Algorithms. *J. Inf. Secur.* **2012**, *3*, 69–76. [CrossRef]
36. Baldwin, B.; Byrne, A.; Hamilton, M.; Hanley, N.; McEvoy, R.P.; Pan, W.; Marnane, W.P. FPGA Implementations of SHA-3 Candidates: CubeHash, Grøstl, LANE, Shabal and Spectral Hash. In Proceedings of the 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, Patras, Greece, 27–29 August 2009; pp. 783–790. [CrossRef]
37. 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]
38. Winderickx, J.; Daemen, J.; Mentens, N. Exploring the use of shift register lookup tables for Keccak implementations on Xilinx FPGAs. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–4. [CrossRef]
39. Kaps, J.P.; Yalla, P.; Surapathi, K.K.; Habib, B.; Vadlamudi, S.; Gurung, S. Lightweight Implementations of SHA-3 Finalists on FPGAs. In Proceedings of the SHA-3 Conference, Chennai, India, 11–14 December 2011.
40. San, I.; At, N. Compact Keccak Hardware Architecture for Data Integrity and Authentication on FPGAs. *Inf. Secur. J. A Glob. Perspect.* **2012**, *21*, 231–242. [CrossRef]
41. 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]
42. Latif, K.; Rao, M.M.; Mahboob, A.; Aziz, A. Novel arithmetic architecture for high performance implementation of SHA-3 finalist keccak on FPGA platforms. In Proceedings of the 8th International Conference on Reconfigurable Computing: Architectures, Tools and Applications (ARC'12), Hong Kong, China, 19–23 March 2012; pp. 372–378. [CrossRef]
43. Gholipour, A.; Mirzakuchaki, S. High-Speed Implementation of the KECCAK Hash Function on FPGA. *Int. J. Adv. Comput. Sci.* **2012**, *2*, 303–307.

44. Homsirikamol, E.; Rogawski, M.; Gaj, K. Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs. *Cryptology ePrint Archive* 2010. Available online: https://eprint.iacr.org/2010/445 (accessed on 15 October 2023).
45. Newe, T.; Rao, M.; Toal, D.; Dooly, G.; Omerdic, E.; Mathur, A. Efficient and High-Speed FPGA Bump in the Wire Implementation for Data Integrity and Confidentiality Services in the IoT. In *Sensors for Everyday Life*; Springer: Cham, Switzerland, 2017; pp. 259–285.
46. Jungk, B. Evaluation of Compact FPGA Implementations For All SHA-3 Finalists. In Proceedings of the Third SHA-3 Candidate Conference, Washington, DC, USA, 22–23 March 2012.
47. Jungk, B.; Stöttinger, M. Hobbit—Smaller but faster than a dwarf: Revisiting lightweight SHA-3 FPGA implementations. In Proceedings of the 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 30 November 2016–2 December 2016; pp. 1–7. [CrossRef]
48. Latif, K.; Rao, M.M.; Aziz, A.; Mahboob, A. Efficient Hardware Implementations and Hardware Performance Evaluation of SHA-3 Finalists. In Proceedings of the NIST Third SHA-3 Candidate Conference, Washington, DC, USA, 22–23 March 2012.
49. Kerckhof, S.; Durvaux, F.; Veyrat, N.; Regazzoni, F.; Standaert, F.X. Compact FPGA Implementations of the Five SHA-3 Finalists. In Proceedings of the CARDIS 2011, Leuven, Belgium, 14–16 September 2011; pp. 217–233.
50. Aziz, A.; Latif, K. Resource Efficient Implementation of Keccak, Skein & JH Algorithms on Reconfigurable Platform. *Cankaya Univ. J. Sci. Eng.* **2016**, *13*.
51. Hieu, D.V.; 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 November 2021; pp. 162–168. [CrossRef]
52. 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]
53. e Shahwar Kundi, D.; Aziz, A. A low-power SHA-3 designs using embedded digital signal processing slice on FPGA. *Comput. Electr. Eng.* **2016**, *55*, 138–152. [CrossRef]
54. Rao, M.; Newe, T.; Grout, I. Secure Hash Algorithm-3(SHA-3) implementation on Xilinx FPGAs, Suitable for IoT Applications. *Int. J. Smart Sens. Intell. Syst.* **2014**, *7*, 1–6. [CrossRef]
55. Nannipieri, P.; Crocetti, L.; Matteo, S.D.E.A. Hardware Design of an Advanced-Feature Cryptographic Tile within the European Processor Initiative. *IEEE Trans. Comput.* **2023**, 1–14. [CrossRef]
56. Wang, Y.; Shi, Y.; Wang, C.; Ha, Y. FPGA-based SHA-3 acceleration on a 32-bit processor via instruction set extension. In Proceedings of the 2015 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), Singapore, 1–4 June 2015; pp. 305–308. [CrossRef]
57. Fritzmann, T.; Sharif, U.; Müller-Gritschneder, D.; Reinbrecht, C.; Schlichtmann, U.; Sepulveda, J. Towards Reliable and Secure Post-Quantum Co-Processors based on RISC-V. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 1148–1153. [CrossRef]
58. Schiavone, P.D.; Rossi, D.; Pullini, A.; Mauro, A.D.; Conti, F.; Benini, L. Quentin: An Ultra-Low-Power PULPissimo SoC in 22nm FDX. In Proceedings of the 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), Burlingame, CA, USA, 15–18 October 2018; pp. 1–3. [CrossRef]
59. Adomnicai, A. An Update on Keccak Performance on ARMv7-M. Cryptology ePrint Archive, Paper 2023/773. 2023. Available online: https://eprint.iacr.org/2023/773 (accessed on 15 October 2023).
60. Elmohr, M.A.; Saleh, M.A.; Eissa, A.S.; Ahmed, K.E.; Farag, M.M. Hardware implementation of a SHA-3 application-specific instruction set processor. In Proceedings of the 2016 28th International Conference on Microelectronics (ICM), Giza, Egypt, 17–20 December 2016; pp. 109–112. [CrossRef]
61. Constantin, J.H.F.; Burg, A.P.; Gürkaynak, F.K. Instruction Set Extensions for Cryptographic Hash Functions on a Microcontroller Architecture. In Proceedings of the 2012 IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Delft, The Netherlands, 9–11 July 2012; pp. 117–124. [CrossRef]