Hardware architecture for CRYSTALS-Kyber post-quantum cryptographic SHA-3 primitives

Hardware architecture for CRYSTALS-Kyber post-quantum cryptographic SHA-3 primitives

Alessandra Dolmeta†∗,*Graduate Student Member, IEEE*, Maurizio Martina† and Guido Masera†

***Abstract*—Once powerful enough quantum computers become feasible, many of the regularly used cryptosystems will be completely useless. Thus, designing quantum-safe cryptosystems to replace current algorithms is more crucial than ever. This paper presents the hardware implementation of one of the fundamental building blocks of all post-quantum cryptographic (PQC) algorithms, which are PQC-primitives, having NIST-PQC-finalists CRYSTALS-Kyber algorithm as a target. This work analyzes Keccak sponge function and the four SHA-3 algorithms used in CRYSTALS-Kyber, realizing the correct processing and handling of input information and integrating the four standards into one implementation for Kyber-III level of security. The synthesis results are provided for 65-nm technology, while Artix-7 XC7A75-3 is chosen as the implementation platform. The efficiency and the performance of the proposed architecture are compared in terms of area, frequency, clock cycles, and efficiency with the state-of-the-art.**

***Index Terms*—Hardware Design, FPGA, ASIC, PQC, CRYSTALS-Kyber, Cryptography**

I. Introduction

In the modern era, cryptography is essential for online communication security. Cryptographic algorithms are based on hard mathematical and computationally infeasible problems. However, quantum computer development represents a significant threat to current cryptosystems. This is why researchers and scientists are working on building and applying quantum-resistant cryptographic algorithms, exploiting problems that are invulnerable to quantum computer attacks. Unlike quantum cryptography, which relies on quantum computing and quantum communication environments, PQC-based cryptosystems run on a classical computer, providing sufficient security. In 2012, the National Institute of Standards and Technology (NIST) launched a public evaluation process to standardize quantum-resistant public key algorithms. After three rounds of solicitations, lattice-based cryptographic algorithms result to be more interesting than others. In fact, among the four Key Encapsulation Mechanism (KEM) finalists, in 2022 CRYSTALS-Kyber has been the first algorithm to be selected for standardization [1].

Contributions: This paper deals with the hardware design of dedicated architectures implementing CRYSTALS-Kyber-768 SHA3 primitives. Two main contributions are provided: (I). we present the ASIC synthesis results derived for a 65 nm CMOS technology, and (II) we compare the FPGA implementation against available literature results in terms of complexity, latency, and efficiency. The rest of the paper is organized as follows: Sec II illustrates CRYSTALS-Kyber algorithms and introduces the theoretical basis of SHA-3 (Sec II-A). Sec III reports the hardware design process and Sec IV compares results with other implementations and Sec V gives suggestions for future improvement and work. [1]

†*Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italy*

∗ corresponding author e-mail: *alessandra.dolmeta@polito.it*

979-8-3503-0320-9/23/$31.00 ©2023 IEEE

II. Theoretical basis

CRYSTALS-Kyber is a **lattice-based cryptosystem** based on the hardness of solving the Module Learning-with-Error (Module-LWE) problem. It implements a KEM, a probabilistic algorithm that produces a random symmetric key and an encryption of that key. Traditional protocols encrypt a message using the public key of the sender, which is then decrypted by the receiver using the private key. However, they have proven to be not very resistant to quantum attacks. KEM adds a shared secret to the process, encapsulating and decapsulating it using the public and the secret keys [5].

In principle, KEM is a triplet of algorithms:

- a **key generation algorithm,** which generates public and secret keys (*pk*, *sk*);
- an **encapsulation algorithm**, that takes as input *pk* and returns an output key *w* and an encapsulated key *ck*, obtained thought the encapsulation of *pk*;
- a **decapsulation algorithm**, that takes in *ck* and *sk*, and recovers the key *w*.

Description and parameters: CRYSTALS-Kyber includes **three parameter sets**, corresponding to three security levels of NIST: Kyber512 (I-level), Kyber768 (III-level) and Kyber1024 (V-level). Independently of which level is considered, Kyber algorithm works on a polynomial ring $R_q = Z_q[X]/(X^n+1)$, fixed dimension: q is the modulo parameter (q=3329) and n is the degree of the polynomial (n=256). The difference between each security level is given by the number of polynomial vectors required: their size is specified by k parameter. [3]

Kyber C-code Analysis: To understand which are the most costly parts of the algorithm, the C-code provided for CRYSTALS has been analyzed [14]. Since key dimensions increase

[1]This work is supported by TRISTAN project, nr.101095947, which has received founding by the Key Digital Technologies Joint Undertaking and its members. This work was partially supported by project SERICS (PE00000014).

as k increases, it is surely predictable that the higher the level of security, the higher the number of calls that are performed for the different main functions.
Nevertheless, we analyze the percentages of the total execution time of the program used by the most relevant functions. From the profiling results, as Fig. 1 proves, **Keccak permutation ends up being one of the most expensive functions in terms of execution time**, despite the fact it is not the most frequently called one. For instance, Kyber makes use of an extendable output function (**XOF**), two hash functions (**H** and **G**), a pseudo-random function (**PRF**), and a key-derivation function (**KDF**).All these primitives are instantiated with functions from the Federal Information Processing Standard (**FIPS-202 standard**), as specified in Table I.

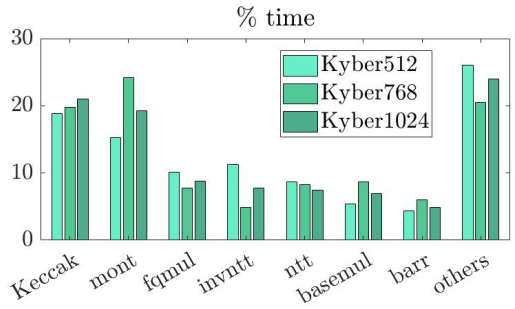

Fig. 1. CRYSTALS-Kyber profiling

Being one of the most time-consuming functions of the algorithm, together with NTT and INV-NTT, a specific and efficient implementation of PQC primitives can be crucial for a PQC algorithm. This is why a dedicated hardware of CRYSTALS-Kyber SHA3 primitives has been implemented, including a control part managing all the different functions.

A. Overview of SHA-3

The **Secure Hash Algorithms** (SHA) are a family of cryptographic hash functions published by the NIST as a U.S. FIPS. **SHA-3** has been the latest standard released by NIST in 2015, which uses the so-called **sponge function**.
A sponge function [11] is a framework for specifying functions on binary data with arbitrary output length (d). The construction employs the following three components: a function on fixed-length (b) strings (f), a parameter called the rate (r), and a padding rule ($\mathtt{pad}$), as shown in Fig. 2. The similarity to a sponge is that an arbitrary number of input bits are "*absorbed*" into the state of the function, and then an arbitrary number of output bits are "*squeezed*" out of its state. Each time r-bits are processed, in both phases, the execution is interleaved with applications of the function f [10].

SHA-3 is a family of sponge functions with members **Keccak[r,c]** [6], characterized by **bit-rate** r and **capacity** c, as shown in Table I. Their sum determines the width (b) of the Keccak-f permutation, which is 1600-bit.
For this application, the input message is padded such that the resulting message M is byte aligned, i.e. $\mathrm{len(M)} = 8\mathtt{m}$. When $\mathtt{m}$ has a lower length with respect to **r**, padding is added to

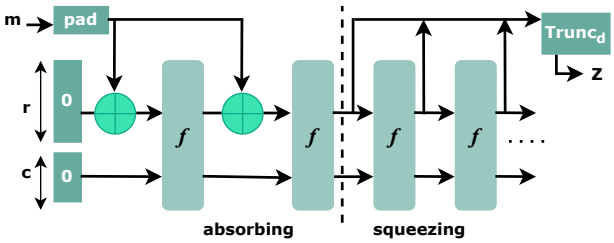

Fig. 2. Sponge function block diagram

the first and only stream present (single-block message). On the contrary, in the case of a multi-block message, all the $\mathtt{m}/r$ streams before the last one are simply padded with zeros. Independently of that, for each CRYSTALS-Kyber function in which a primitive is called, the final padded message is:

- for hash function: $\mathtt{m}$ || 0x06 || 0x00 ... || 0x80
- for XOF function: $\mathtt{m}$ || 0x1F || 0x00 ... || 0x80

The notation 0x00... indicates the string that consists of $q-2$ "*zero*" bytes, where : $q = \frac{r}{8} - \left(\mathtt{m} \textbf{ mod } \frac{r}{8}\right)$

TABLE I
FIPS 202 STANDARDS' PARAMETERS

Kyber primitives	SHA-3 primitives	r	c	Output length
H	SHA3-256	1088	512	256
G	SHA3-512	576	1024	512
XOF	SHAKE128	1344	256	unlimited
PRF, **KDF**	SHAKE256	1088	512	unlimited

The 1600-bit state of Keccak-f consists of **5x5 matrix of 64-bit words**. The permutation width determines the number of rounds, n_r, which is in this case **24**. **Each compression round consists of five different steps**. These steps are denoted as θ, ρ, π, χ and ι. [6]. Fig. 3 shows a block diagram of the hardware implementation described in Subsec. III-A.

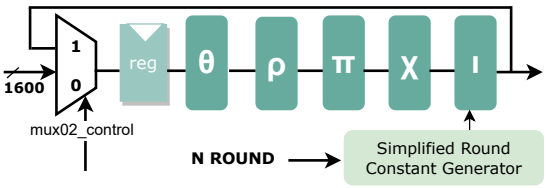

Fig. 3. Keccak Core

III. HARDWARE IMPLEMENTATION

In this section, we present the architecture and the main characteristics of the implemented SHA3 module. It is an **hardware accelerator** designed for the implementation of hash and XOF functions compliant with SHA3-algorithm and dedicated to CRYSTALS-Kyber application. Its aim is to achieve the best trade-off between throughput and complexity. It has been optimized for a single user at a time (multiple requests are not possible) and it has been designed with a **top-down approach**. The architecture was initially modeled using

VHDL language and has been designed considering parallelisms required for the III level of security of CRYSTALS-Kyber. Moreover, it has been simulated and verified for accuracy and functionality with valid input samples provided in [8] using ModelSim and Python code. Then, final testing procedures have been performed considering random inputs. Further details about the implementation can be found at [9].

A. SHA3 core

It can operate on both one-block and multi-block messages. A multi-block message is an input whose length is higher with respect to the maximum bit rate that can be processed by the specific primitive (Table I).
The main blocks of the architecture are shown in Fig 4.

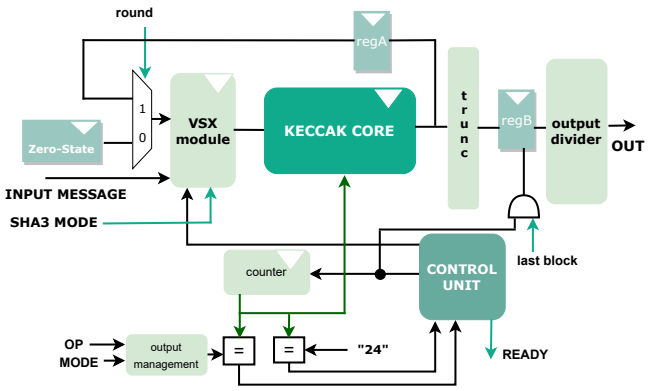

Fig. 4. SHA3 core

- **Zero-state**: it is a register full of zeros needed for the first iteration. Then, for multi-block messages, the feedback state is selected through the multiplexer instead.
- **Version Selection and XORing** (VSX): it is the module in charge of constructing the proper state per primitive. It is made up of 1344-XORs and four concatenation components, one per primitive. [2]
- **Keccak core**: this is the transformation round. [6] One register has been added before ρ round, to prevent the VSX logic to increase the critical path, as shown in Fig. 3. Keccak core iterations are handled by the counter output, which also selects the proper RC constant for ι-iteration. As in [9], the size of the round constant generator is reduced from 64-bit to one-byte size, storing only the non-zero bits in each of the round constant values. This also simplifies the computation in ι, where the number of logical XOR is reduced from 64 to 8.
- **Counter and comparators**: the counter is used both to keep track of Keccak permutation cycles and to scan the clock cycles needed for buffering the result. Therefore, its output is connected to two different comparators.
- **Truncation unit**: it is used to re-organize properly the hash value exiting from Keccak core. By analyzing the Kyber768 primitives' outputs, we found that some area can be saved sending to `regB` only 1024 out of 1600. Moreover, in order to save power, this last register is enabled only at the end of all the permutations required by the specific primitive, when the last stream has been transformed by Keccak core.
- **Output management and output divider units**: they are used to unpack the result into streams of 64 bits. A set of multiplexers is then implemented to properly choose among the different streams.

B. Control Unit

The control unit shown in Fig. 5 within the whole architecture has been developed to synchronize the flow of data in the architecture and data communication between input and output. In addition to the SHA3 core described in Subsec. III-A, there are:

- **Acquisition Unit**: it is the first unit, which aim is to properly handle the data in input. Output buffer parallelism will be 1344. It has been designed in order to work concurrently with the other units. Therefore, for multi-block messages, the latency of input processing is completely absorbed. In fact, accordingly to SHA3 permutation algorithm, we can process only r-bits at a time. When one stream has been collected, this is sent to the following unit in order to be saved or processed.
- **Stream Control Unit**: it is used in order to store the data until the core is ready to process a new message again. Therefore, it is exploited only in the case of a multi-block message. When a function requires only one permutation cycle, then the unit will simply process the data through some multiplexers. On the other hand, when a function requires more than one permutation cycle, the machine absorbs and saves incoming data streams from the acquisition unit. Each time the SHA3-core finishes a computation, the unit checks whether there are still streams to be sent or not, and in case selects and sends the correct one to the padding unit. In the case of CRYSTALS-Kyber functions, there are no SHAKE128 primitives which need more than one cycle of absorption. Therefore, we can assume the maximum parallelism of this unit to be 1088, saving some area.
- **Padding Unit**: it implements the padding operations, accordingly to Sec. II-A.

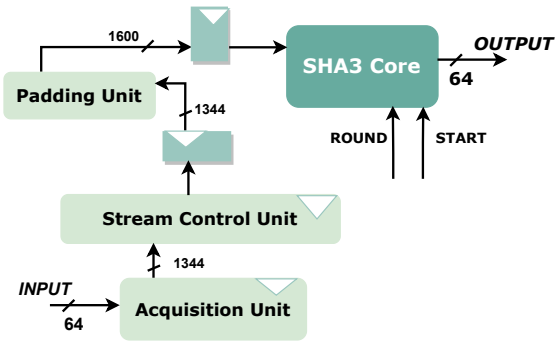

Fig. 5. Top level architecture block diagram

Each main unit adopts the classical Finite State Machine with Data path (FSMD) model. There is a **controller**, which is in charge of organizing the operations in each clock cycle, and the **data-path**, which includes functional blocks and registers. Different ROMs are present, mainly addressed by primitives control signals. Their aim is to handle data acquisition and maintenance all over the architecture, and they have been specifically designed for CRYSTALS-Kyber-768 application.

IV. Results and analysis

In this section, we report implementation results and compare them with the ones present in the literature. First, Synopsys Design Compiler has been exploited to perform logic synthesis on **65-nm technology** of both the SHA3 core and the control unit described in Sec III, obtaining a frequency of $312.5\,\mathrm{MHz}$ and an area of 340048 $\mu\mathrm{m}^2$. Then, the proposed architecture has been implemented with Xilinx Vivado 2022.1 on **Xilinx Artix XC7A75-3** device, obtaining the results reported in Tab II. The other implementations examined are all parts of larger architectures designed for the execution of the complete CRYSTALS-Kyber algorithm. Thus, for a general overview, in Tab II we have distinguished area results related to SHA3-core only from the ones related instead to the whole SHA-3 architecture (respectively in and out of the round brackets), and the total time and the area-time products (defined as the product between the number of LUTs of Keccak-core and the total time) of the shortest (256-bit) and longest (9472-bit) input messages present (respectively left and right side of the slash).

TABLE II
Comparison of Keccak-core hardware architecture in CRYSTALS-Kyber implementation

Parameter	[7]	[13]	[4]
Device	Artix-7	Virtex-6	Artix-7
Method	HW	HW	HW/SW
LUTs	4405	359	5784
FFs	1629	107	1605
Slices	1825	91	1716
Frequency (MHz)	115	311	25
Total time (µs)	0.208	5.35	0.96
Area × Time(LUTs×µs)	916	1920	5552
Parameter	[12]	[15]	**Our Work**
Device	Artix-7	Artix-7	**Artix-7**
Method	HW	SW/HW	**HW**
LUTs	4026	6322	**9651** (**6841**)
FFs	1625	6993	**8697** (**4279**)
Slices	1056	-	**3413** (**1873**)
Frequency (MHz)	159	229	**250**
Total time (µs)	-	0.48	**0.16/1.94**
Area × Time (LUTs×µs)	-	3034	**1094/13271**

Our frequency achieves a 48.9% improvement compared to the average value of the others implementations. It is lower with respect to [13], but a lower amount of clock cycles are needed in the execution. Thus, referring to the definition given in Sub. II-A, when single-block messages are required, there is a latency improvement (almost 19% less compared to the best time get in [7]). On the other hand, when multi-block messages are required, depending on the number of streams to be managed, latency results are in line with the state-of-the-art. However, as the other works do not specify the maximum input length, the comparison could be not completely fair. The same holds true for resource utilization results. Finally, as it can be observed, our best result is slightly worse than [7] in terms of area delay product (16%), but it achieves an improvement in terms of latency of the 30%. No BRAMs are used in the implementation since the Keccak state is stored using LUTs.

V. Conclusion

In this work, we present an efficient hardware architecture of cryptographic primitives required by CRYSTALS-Kyber. Our architecture has been implemented for **Kyber-768 cryptographic primitives** but it can be easily extended to the other levels of security, modifying properly memories discussed in Subsec. III-B and by changing consequently some components in the control unit. Our future work will focus on the development of the other CRYSTALS-Kyber main blocks, and their integration into a RISC-V environment, to be part of the evolution, integration, and migration of cryptography systems toward quantum-safe security protocols.

References

[1] NIST. https://www.nist.gov/h. [Online; accessed 19-July-2022].

[2] George S. Athanasiou, George-Paris Makkas, and Georgios Theodoridis. High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm. In *2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pages 538–541, 2014.

[3] Roberto Maria Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation. 2017.

[4] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols (extended version). Cryptology ePrint Archive, Paper 2019/1140, 2019. https://eprint.iacr.org/2019/1140.

[5] Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri. NIST post-quantum cryptography- a hardware evaluation study. Cryptology ePrint Archive, Report 2019/047, 2019. https://ia.cr/2019/047.

[6] G. Bertoni, J. Daemen, M. Peeters, G.V Assche, and R.Van Keer. Keccak in VHDL. https://keccak.team/hardware.html.

[7] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Instruction-set accelerated implementation of CRYSTALS-Kyber. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(11):4648–4659, 2021.

[8] NIST I.T.L. Computer Security Division. Example values - cryptographic standards and guidelines. https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/example-values. [Online; accessed 19-July-2022].

[9] Alessandra Dolmeta. Hardware architecture for CRYSTALS-Kyber cryptographic primitives. Master's thesis, Politecnico di Torino, 2022.

[10] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015.

[11] B Guido, D Joan, and P Michaël. Cryptographic sponge functions. 2011.

[12] Wenbo Guo, Shuguo Li, and Liang Kong. An efficient implementation of KYBER. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):1562–1566, 2022.

[13] Bernhard Jungk and Marc Stöttinger. Hobbit — smaller but faster than a dwarf: Revisiting lightweight SHA-3 FPGA implementations. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–7, 2016.

[14] Peter Schwabe. PQ-CRYSTALS Kyber c-code. https://pq-crystals.org/kyber/index.shtml, 2017.

[15] Deepraj Soni and Ramesh Karri. Efficient hardware implementation of PQC primitives and PQC algorithms using high-level synthesis. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 296–301, 2021.