Hardware accelerator for FIPS 202 hash functions in post-quantum ready SoCs

Diamante-Simone Crescenzo, Rafael Carrera Rodriguez, Riccardo Alidori, Florent Bruguier, Emanuele Valea, Pascal Benoit, Alberto Bosio

Hardware Accelerator for FIPS 202 Hash Functions in Post-Quantum Ready SoCs

Diamante Simone Crescenzo[1], Rafael Carrera Rodriguez[1,2], Riccardo Alidori[1],
Florent Bruguier[2], Emanuele Valea[1], Pascal Benoit[2], Alberto Bosio[3]
[1]Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
[2]LIRMM, University of Montpellier, CNRS, Montpellier, France
[3]Institut des Nanotechnologies de Lyon, École Centrale de Lyon, Lyon, France

***Abstract*—In today's digital landscape, cryptography plays a vital role in ensuring communication security through encryption and authentication algorithms. While traditional cryptographic methods rely on hard mathematical problems for security, the rise of quantum computing threatens their effectiveness. Post-Quantum Cryptography (PQC) algorithms, like CRYSTALS-Kyber, aim to withstand quantum attacks. Recently standardized, CRYSTALS-Kyber is a lattice-based algorithm designed to resist quantum attacks. However, its implementation faces computational challenges, particularly with Keccak-based functions, which are crucial for security and upon which the FIPS 202 standard is based. Our paper addresses this technological challenge by designing a FIPS 202 hardware accelerator to enhance CRYSTALS-Kyber efficiency and security. We chose to implement the entire FIPS 202 standard in hardware in order to widen the applicability of the accelerator to all possible algorithms that rely on such hash functions, taking care to provide realistic assumptions on system-level integration inside a System-on-Chip (SoC). We provide results in terms of area, frequency, and clock cycles for both ASIC and FPGA targets. An area reduction of up to 22.3% is achieved with respecto to state-of-the-art solutions. In addition, we integrated the accelerator inside a 32-bit RISC-V based security-oriented SoC, where we show a strong performance gain on CRYSTALS-Kyber execution. The design presented in this paper performs better in all Kyber1024 primitives, with an improvement up to 3.21× in Kyber-KeyGen.**

I. Introduction

Hash functions play a pivotal role in modern cryptography applications. They serve as an essential tool for ensuring the security and the authenticity of data. A *Hash Function* takes as input a message and produces a fixed-size string of bytes, which is typically a unique representation of the input data. This property makes hash functions invaluable for tasks like digital signatures, password hashing, and data verification. The *Federal Information Processing Standard Publication 202* (FIPS 202) is a standard developed by the *National Institute of Standards and Technology* (NIST), that defines standardized hash functions belonging to the SHA-3 family. In addition, the FIPS 202 standard also specifies *eXtendable Output Functions* (XOFs), called SHAKE functions, that can be used to produce pseudo-random sequences of arbitrary length [1]. All SHA-3 and SHAKE functions, defined in the FIPS 202 standard, are based on the KECCAK permutation function [2].

Given the frequent utilization of FIPS 202 functions in several cryptographic primitives, they represent a computational bottleneck in software implementations [3]. For this reason, it is crucial to accelerate their computation by providing hardware support. This is essential for enhancing performance and efficiency, especially in applications like smart cards and IoT devices. Moreover, in applications where robustness against side-channel attacks is important, hardware implementations of cryptographic primitives represent an advantage with respect to their software counterpart [4], leading to more secure implementations. The importance of providing efficient implementations of FIPS 202 functions has been exacerbated in recent decades with the emergence of *Post-Quantum Cryptography* (PQC). In particular, the CRYSTALS-Kyber algorithm, that has recently become one of the NIST PQC standards [5], requires a big amount of pseudo-randomness that is generated through FIPS 202 functions [6]. For this reason, providing hardware acceleration of FIPS 202 functions also enables efficient PQC implementations.

A lot of effort has been made in the scientific community in proposing solutions for the hardware acceleration of the KECCAK function, that is at the core of all FIPS 202 primitives [3]. However, when SHA-3 and SHAKE accelerators must be implemented, an effort must be made in order to correctly adapt the KECCAK interface with the data size of the *System on Chip* (SoC) architecture in which the accelerator is integrated. This problem has been addressed by some researchers, that have proposed different solutions targeting either tightly-coupled solutions for CPU ISA extension [7], either classical accelerators for 32-bit SoCs [6], [8], [9]. However, in all the aforementioned solutions, the authors limit their scope to the set of FIPS 202 functions that are necessary to the acceleration of the CRYSTALS-Kyber PQC algorithm. As far as we know, there are no hardware acceleration solutions for the complete FIPS 202 standard.

In this paper, we propose a lightweight FIPS 202 hardware accelerator optimized for SoC integration. Differently from existing solutions, the proposed accelerator provides support for the whole FIPS 202 standard, still presenting a smaller silicon area footprint and comparable performances. Furthermore, we present the integration of the accelerator inside a 32-bit RISC-V based SoC platform and we executed

the CRYSTALS-Kyber PQC algorithm with all SHA-3 and SHAKE computations offloaded to the accelerator, achieving an increase in performances of up to 3.21× in terms of clock cycles per execution on a full key generation procedure.

The rest of the paper is organized as follows: Section II provides an overview about some background concepts and the state of the art. Section III contains a detailed description of the proposed accelerator, whose implementation results are then reported in Section IV, where they are discussed and compared with other related works. Section V contains conclusions and suggests possible directions for future research in the field.

II. Background on FIPS 202 Implementations

In this Section, we provide some background concepts on SHA-3 and SHAKE functions that are defined by the FIPS 202 standard. Then we give some details on the CRYSTALS-Kyber PQC algorithm, explaining why FIPS 202 functions are fundamental for its implementation and, finally, we discuss state-of-the-art hardware implementation solutions for FIPS 202 functions.

A. FIPS 202 and Keccak

The FIPS 202 standard defines the specifications for the implementation of SHA-3 and SHAKE functions. SHA-3 functions are *Hash Functions*, while SHAKE functions are *eXtendable Output Functions* (XOFs). A *hash function* is a one-way function that maps an input of any size, called message, to a unique output of a fixed length of bits. The SHA-3 family of the FIPS 202 standard consists of four hash functions, called SHA3-224, SHA3-256, SHA3-384, and SHA3-512. Table I shows the output size for each SHA-3 function. A XOF is a function that takes as input a message of arbitrary length and outputs a byte-string of any desired length. The SHAKE family of the FIPS 202 standard consists of two XOFs, called SHAKE128 and SHAKE256.

TABLE I
PARAMETERS OF FIPS 202 FUNCTIONS

Function	r	c	Output Size
SHA3-224	1152	448	224
SHA3-256	1088	512	256
SHA3-384	832	768	384
SHA3-512	576	1024	512
SHAKE128	1344	256	unlimited
SHAKE256	1088	512	unlimited

SHA-3 and SHAKE functions are built on top of the Keccak permutation function [2]. Keccak is a cryptographic primitive designed to *absorb* input data of arbitrary length and *squeeze* output data of arbitrary length. All SHA-3 and SHAKE functions are built using the Keccak function with different parameters. Figure 1 shows the structure of Keccak. The *state* register is 1600-bit wide and it is divided in two sections: *rate* (*r*) and *capacity* (*c*). The *r* and *c* parameters are defined by the FIPS 202 standard and their values are shown in Table I. The *permutation function* (*f*) is made, internally, of 24 iterations of the same *round* function. In

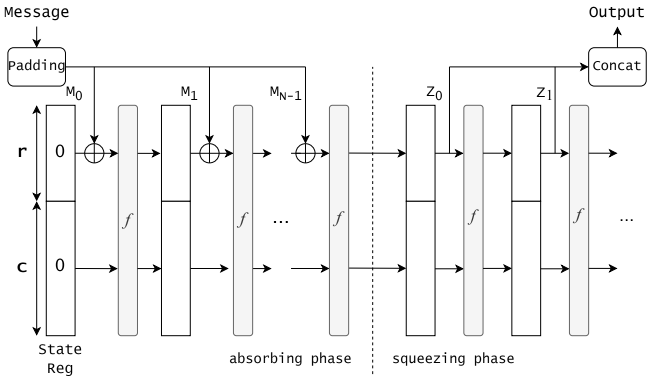

Fig. 1. Keccak Sponge Construction

the absorbing phase, the Keccak function takes an input of arbitrary length and compresses it inside the state register. Initially, the state register is set to zero. The input message is padded in order to reach a length that is multiple of *r*. This is done according to some rules that are specified by the standard, shown in Table II. Then, the message is divided in *blocks* of *r* bits. Each message block is XORed with the *rate* section of the state. After that, the whole state register is processed by the permutation function. This procedure is repeated until all message blocks are absorbed. In the squeezing phase, the output value is extracted from the *r* bits of the state register. In the case of SHA-3 functions, *r* bits are always enough to contain the whole output message. In the case of SHAKE functions, the output message could be longer than *r*, obliging to execute multiple squeeze operations, each time executing the permutation function and extracting *r* bits from the state register, as shown in Figure 1.

TABLE II
FIPS 202 MESSAGE PADDING

FIPS 202 Function	Message Padding Format
SHA-3	m || 0x06 || 0x00* || 0x80
SHAKE	m || 0x1F || 0x00* || 0x80

B. CRYSTALS-Kyber

CRYSTALS-Kyber is a lattice-based, post-quantum secure *Key Encapsulation Mechanism* (KEM) [5], built upon a public-key encryption system. The public and secret keys are generated by a *key generation* function. The negotiation of the shared key relies on *encapsulation* and *decapsulation* functions. All these functions are based on a set of basic operations which are mapped to the FIPS 202 standard as shown in table Table III. In software implementations of CRYSTALS-Kyber, FIPS 202 functions represent a computational bottleneck since they take between 64% and 81% of the execution time [7]. For this reason, having hardware-accelerated FIPS 202 primitives represent a big speed-up for CRYSTALS-Kyber.

TABLE III
FIPS 202 FUNCTIONS USED IN CRYSTALS-KYBER

Kyber Primitive	FIPS 202 Function
H	SHA3-256
G	SHA3-512
XOF	SHAKE-128
PRF, KDF	SHAKE-256

C. Hardware Implementations of FIPS 202

In the scientific literature, the vast majority of the works limit their focus on the acceleration of the KECCAK permutation function [3]. However, little effort has been made in defining how the 1600-bit state register can be efficiently interfaced with a real computer architecture, usually 32-bit or 64-bit architectures. Indeed, implementing full hardware support for FIPS 202 functions necessitates considering both the *de-serialization* and *padding* of input data, plus the *serialization* of output data.

As pointed out in subsection II-B, PQC algorithms, and in particular CRYSTALS-Kyber, present an interesting use case for SHA-3 and SHAKE function usage. For this reason, all works related to FIPS 202 accelerators design limit their focus to the functions employed in CRYSTALS-Kyber.

For instance, the authors of [8] make use of the high level-synthesis approach to design an accelerator for the KECCAK permutation function other than the SHA-3 and SHAKE functions used for accelerating CRYSTALS-Kyber. In [9], a SHA-3 and SHAKE accelerator for CRYSTALS-Kyber is proposed, which also provides padding and output management features. The authors of [6] propose a SHA-3 and SHAKE accelerator for CRYSTALS-Kyber, integrated inside a 32-bit SoC. The target platform is the PULPissimo microcontroller by means of memory mapping and through an AXI connection.

Differently from these solutions, we propose an accelerator that implements the whole FIPS 202 standard, not limiting the scope to the CRYSTALS-Kyber application. Moreover, we also provide the padding and input/output management functionalities considering, as a target, the integration inside a 32-bit SoC. Thanks to the design choices described in section III, we also obtain a smaller area footprint with respect to state-of-the-art solutions.

III. PROPOSED FIPS 202 HW IMPLEMENTATION

The proposed hardware accelerator has been designed to be compatible with 32-bit systems, which is the word size for both input and output modes. In fact, specific control signals to handle bus transactions have been included in the design together with the necessary inputs that suit the memory mapping interface, both for control and data bits. Figure 2 shows the top-level architecture of the accelerator.

Given the limited data bus width (i.e., 32 bits) with respect to the KECCAK state width (i.e., 1600 bits), the loading of the message words into the accelerator takes a certain amount of time. This led us towards the direction of including one single round of the KECCAK permutation function inside the

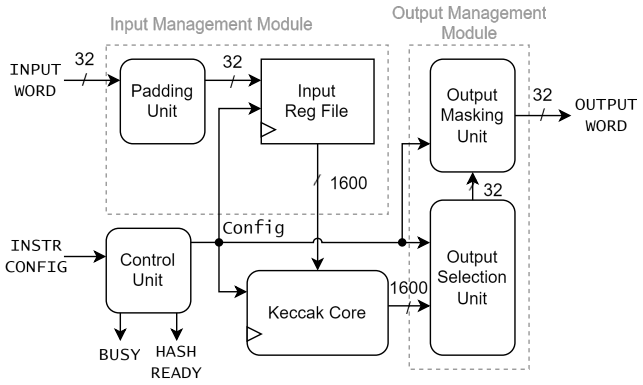

Fig. 2. Top-level of the FIPS 202 accelerator

combinational core of the accelerator (i.e., *Keccak Core* in Figure 2). Its architecture operates across the entirety of the KECCAK state at once and is able to carry out one complete round per clock cycle. Hence, 24 clock cycles are necessary to complete the permutation function. The internal architecture of the *Keccak Core* module has been taken from the reference implementation in [2], to which a wrapper has been added and some control signals have been modified to provide FIPS 202 functionalities.

In addition, we exploited the 32-bit width of the input words sent in input to simplify the padding unit. The *Input Management Module* is in charge of managing the incoming data and it contains the *Padding Unit* and the *Input Register File*. The output generation has been simplified according to the same principle, i.e., building a fully combinational *Output Management Unit* without replicating the 1600-bit state register of the *Keccak Core* on the output interface. Moreover, the *Output Management Unit* supports the scenario in which the size of the requested output message is not multiple of a 32-bit word. More details on the Input/Output Management Units will follow in the next subsections.

The *Control Unit* receives instructions through the *INSTR CONFIG* signal and provides control features over the datapath. Sending specific op-codes to the control unit, all FIPS 202 functions can be executed with the desired length for both input and output messages. With the purpose of saving area, internal pipelining has not been applied, thus the accelerator can alternatively handle receiving data blocks, permuting them, or outputting the permuted blocks.

A. Input Management Module

To improve the effectiveness of having a dedicated hardware accelerator, this design also provides functionalities to add the necessary padding to the input data. In this way, from a high-level point of view, the user only needs to feed the input data to be processed without caring about the FIPS 202 standard's implementation details. Byte-wise padding is provided to give flexibility in input data widths. Table II provides a schematic description of how padding is applied relatively to the required function (on the left). In short,

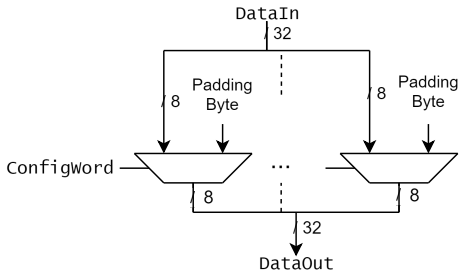

Fig. 3. Padding Unit Schematic

the message is followed by a *padding delimiter*, a series of zero bytes, to conclude with the *padding end*. As shown in Figure 3, the *Padding Unit* is composed of four *byte selection multiplexers* whose output bytes, once concatenated, form the output 32-bit word `DataOut`. Each multiplexer, based on the received selection signal, can let through the original `DataIn` byte, the filler byte (`0x00`), or the padding delimiter for SHA-3 (`0x06`) or SHAKE (`0x1F`). The last multiplexer is dedicated to the selection of the most significant byte of `DataOut` and needs three additional options, namely the normal padding end (`0x80`), or the complete one-byte padding for SHA-3 (`0x86`) or SHAKE (`0x9F`). Such options are necessary to properly configure the last word of the padding, i.e., the last word of the KECCAK rate. The `DataOut` word is then assembled based on the value of the `ConfigWord` signal, which is composed of the concatenation of all selection signal of all multiplexers (from least to most significant byte) and that is directly generated by the control unit.

An *Input Register File* is placed right after the *Padding Unit*. Since the KECCAK rate for FIPS 202 standard can be as large as 1344-bit (see Table I), and the target architecture allows only 32-bit transactions, the registers are needed to receive the input messages serially (as a sequence of 32-bit words) and output them on the entire KECCAK state 1600-bit width. In other words, they are needed as data parallelizers to satisfy the KECCAK core requirements. Due to the *serial words input* to *parallel state output* behavior, some design simplifications have been performed. The input register only needs to be addressed to write on it, no word-wise read is needed as the KECCAK core takes the output as a whole. As further optimization, the last eight words of the register have been hardwired to the all-zero pattern, since they always belong to the capacity section of the KECCAK state and this allows to reduce the total number of flip-flops needed in the design, reducing area occupation as a result. Finally, the internal register file is hidden to the user, which will only have access to a general data register that will be connected to the padding unit, and the control unit performs the addressing for the correct input data words positioning.

B. Output Management Module

At the end of the KECCAK rounds, the result of the function required to the accelerator (either SHA-3 or SHAKE) is directly available in the state, i.e., in the registers placed inside the KECCAK core. In order to be able to extract the data of interest as 32-bit words to be sent later on the bus, the most straightforward solution would be to copy the KECCAK state into a hypothetical output register file so that one can exploit its intrinsic addressability property. However, this solution is redundant because the state would be merely duplicated without applying any modification to the stored data. Moreover, it would cause an additional clock cycle delay. Hence, the *Output Selection Unit* (shown in Figure 4) abstracts the internal KECCAK state registers and makes them accessible by the control unit as if it was a classical register file. It is placed right after the KECCAK core (see Figure 2), and receives as input the state and proper configuration signals. To achieve this functionality a multiplexing structure has been designed which lets through one word at a time, with the state addressed LSB-first, and the multiplexer's control signals are driven by the address provided by the control unit. The output word is exposed on the `DataOut` signal only if `Out_Sel_en` is asserted.

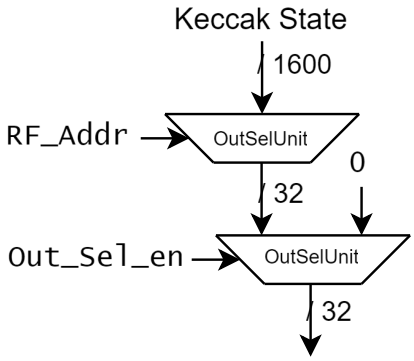

Fig. 4. Output Selection Unit Schematic

The *Output Masking Unit* performs the task of properly constructing the output word by either letting the data word coming from the *Output Selection Unit* through, or extracting a lower number of bytes from it. This allows to request byte-aligned SHAKE outputs without the restriction of being word-aligned. When this is the case, MSBs of the last output word are set to zero. In other words, our accelerator can provide an output of any number of bytes whenever the configured cryptographic function allows it (as is the case in SHAKE functions). By observing its structure in Figure 5, it can easily be noticed that it is a stripped-down version of the *Padding Unit*. Here the multiplexers are all equal and allow to select either the `DataIn` byte or set it to zero. The output word `DataOut` is composed by the concatenation of the four selected bytes. As for the case of the padding unit, the multiplexers' selection signals are concatenated in the `OutConfWord` input signal that is driven by the control unit.

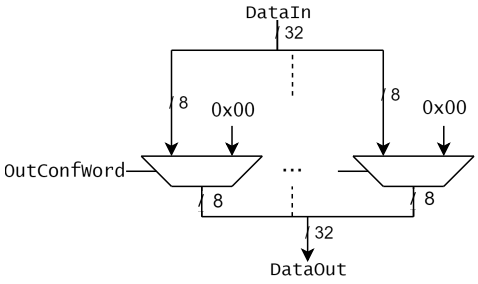

Fig. 5. Output Masking Unit Schematic

IV. Results

In the following, we report the implementation results of the proposed FIPS 202 accelerator. At first, we provide architectural performance results of the proposed architecture on all SHA-3 and SHAKE functions. After that, we show the area footprint and the maximum achievable operating frequency on both ASIC and FPGA targets. Finally, we provide integration results of the proposed accelerator inside a RISC-V based 32-bit SoC, where the proposed module is used for accelerating the execution the CRYSTALS-Kyber PQC algorithm.

The accelerator has been modeled at RTL level using the Verilog hardware description language. Logic simulations have been performed using QuestaSim 2021.4.2. Logic synthesis for ASIC target has been carried out using Synopsis Design Compiler 2019.12-SP5-4 exploiting a standard cell library on a 22nm FDSOI technology from Global Foundries. FPGA synthesis have been performed with the Vivado v2021.1.1 suite.

A. Architectural Performances

The number of clock cycles needed for all FIPS 202, can be synthetically described by means of the following Equation 1:

$$1 + \lfloor \frac{m}{r} \rfloor (\frac{r}{32} + 27) + \frac{m \bmod r}{32} + 29 + \frac{outbits}{32} \quad (1)$$

In this equation, *m* represents the total message size in bits, *r* denotes the rate of the required function (SHA-3 or SHAKE in any of their versions), and *outbits* refers to the length of the output in bits. The first two terms of Equation 1 account for the full message blocks of a multi-block message, while the third term considers the bits of a message smaller than the rate or the remaining bits of a multi-block message. The last term accounts for the latency in retrieving the output of the requested function based on its length. Each bit length parameter is divided by 32, which corresponds to the word width of the target architecture. Additionally, the equation incudes two constant terms, 27 and 29, which account for the clock cycles necessary to complete the Keccak permutation function, along with additional clock cycles necessary to generate padding and receive successive instructions. Furtermore, the equation incorporates an additional clock cycle on the left to provide the IP with instruction and configuration parameters.

B. Synthesis Results

For what concerns ASIC implementation, the presented FIPS 202 accelerator can reach an operational frequency of 875 MHz (i.e., a critical path of 1.14 ns) with an area occupation of 15,963 μm^2. The details about the area occupation of each component of the accelerator can be observed in Table IV. Here, also the Gates Equivalent (GEs) figure is reported, considering the ratio between the area of the accelerator and the smallest NAND gate provided by the technology library (i.e., 0.19968 μm^2). It is straightforward to appreciate how the *Output Selection Unit* is sensibly smaller than the *Input Register File*. From this, we deduce that this solution is roughly 8 times more area-efficient with respect to the most direct approach involving a dedicated output register file. In addition, it is worth pointing out the very small occupation of the *Padding Unit* and the *Output Masking Unit*, given their optimized design focused on serial 32-bit word data, which occupy respectively only 0.4% and 0.1% of the total area.

TABLE IV
FIPS 202 accelerator area (22 nm technology)

Unit Name	Area (μm^2)	Area (kGE)	Area Percentage
Top Level	15,962.69	79.94	100.0
Input Register File	3,640.97	18.23	22.8
Keccak Core	11,502.10	57.60	72.1
Output Masking Unit	16.64	0.08	0.1
Output Selection Unit	451.94	2.26	2.8
Padding Unit	57.11	0.29	0.4
Control Unit	292.80	1.47	1.8

Due to the lack of ASIC implementations targeting the same technology in the state-of-the-art, we decided to deploy our design on the Xilinx Virtex-7 FPGA platform. Table V shows the result of of the FPGA synthesis of the proposed FIPS 202 accelerator. A comparison with related works is also provided. The proposed implementation is smaller with respect to [8] and [9] both in terms of LUTs and FFs. Globally, the proposed architectural optimizations lead to a 22.3% reduction in slices occupation with respect to [9]. We also reported results on the execution time of a SHA3-256 function with two different input message lengths. It worth pointing out that the solution from [9] is based on a 64-bit interface, capable of transferring more input/output data in each clock cycle.

Moreover, it is necessary to consider that, as shown in Table VI, the proposed accelerator covers a broader spectrum of functionalities with respect to both [8], which focuses on SHA-3 acceleration only, and [9], which limit their scope to CRYSTALS-Kyber related functions.

C. CRYSTALS-Kyber Acceleration Results

We integrated our accelerator inside a 32-bit SoC platform that we designed. As it can be seen in Figure 6, this includes a RISC-V processor, 256kB instruction and data memories, an OBI bus, configurations registers, and a timer. The proposed FIPS 202 accelerator represents a 2.0% area overhead with respect to the overall surface of such a platform.

TABLE V
FIPS 202 ACCELERATOR FPGA RESULTS

	[8]	[9]	This Work
FPGA Platform	Artix-7	Artix-7	**Virtex-7**
Architecture	-	64-bit	**32-bit**
LUTs	13,281	9,651	**7,795**
FFs	14,498	8,697	**4,936**
Area [Slices]	5,134	3,413	**2,653**
Frequency [MHz]	207	250	**210**
SHA3-256, 256 bit message			
Latency [μs]	-	0.16	**0.21**
Area-Time Product (Slices $\times$ μs)	-	546	**557**
SHA3-256, 9472-bit message			
Latency [μs]	13.16	1.94	**2.53**
Area-Time Product (Slices $\times$ μs)	67550	6689	**6712**

TABLE VI
COMPARISON OF SUPPORTED FIPS 202 FUNCTIONS

Reference	[8]	[9]	This Work
SHA3-224	yes	no	yes
SHA3-256	yes	Kyber only	yes
SHA3-384	yes	no	yes
SHA3-512	yes	Kyber only	yes
SHAKE128	no	Kyber only	yes
SHAKE256	no	Kyber only	yes

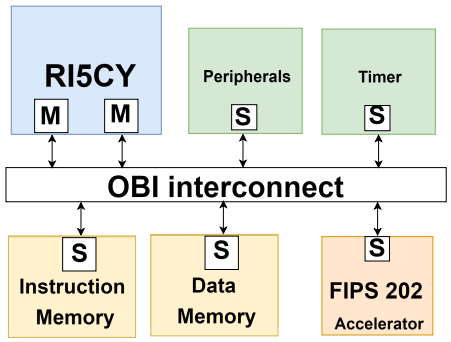

Fig. 6. FIPS 202 IP System Integration

We ran a SW implementation of CRYSTALS-KYBER on the platform, specifically the PQClean version [10]. After that, we offloaded all the FIPS 202 primitives in CRYSTALS-Kyber to the proposed FIPS 202 accelerator. Table VII shows the obtained results. The *speed-up factor* represents the ratio between the number of clock cycles required to execute the reference CRYSTALS-Kyber software implementation and the equivalent algorithm executed with the proposed FIPS 202 accelerator. We compare our work to [6] that implements the same kind of accelerator (similar to [9]) and integrates it inside the PULPissimo 32-bit SoC platform. Table VIII shows the speed-up factors of the two solutions, showing an advantage in the solution proposed by this paper.

V. CONCLUSION

In this paper, we present a FIPS 202 compliant hardware accelerator. Its purpose is to accelerate SHA-3 and SHAKE functions, providing a hardware solution optimized for the integration inside a 32-bit SoC architecture. This allowed to perform optimizations that lead to a very small area footprint, still maintaining comparable state-of-the-art performances. To show its compatibility with the acceleration of PQC algorithms we integrated it inside a RISC-V based SoC and we showed a speed-up factor on the execution of CRYSTALS-Kyber higher than state-of-the-art-solutions. Our future work will aim to make this accelerator resistant to side-channel attacks, so that its application can extend to security-oriented systems.

TABLE VII
CRYSTALS-KYBER WITH FIPS 202 ACCELERATOR

	Function	Full SW [10]	SW + HW	Speed-up
Kyber512	KeyGen	1,021,378	318,249	**3.21**
	Encaps	1,123,585	444,621	**2.53**
	Decaps	1,328,326	639,279	**2.08**
Kyber768	KeyGen	1,674,092	551,230	**3.04**
	Encaps	1,855,996	705,999	**2.63**
	Decaps	2,127,134	962,907	**2.21**
Kyber1024	KeyGen	2,638,658	836,199	**3.16**
	Encaps	2,850,912	102,1317	**2.79**
	Decaps	3,192,647	1,342,597	**2.38**

TABLE VIII
CRYSTALS-KYBER SPEED-UP (KEYGEN/ENCAPS/DECAPS)

	Kyber512	Kyber768	Kyber1024
[6]	2.79/2.60/1.97	2.67/2.66/2.08	2.76/2.71/2.20
This Work	**3.21/2.53/2.08**	**3.04/2.63/2.21**	**3.16/2.79/2.38**

ACKNOWLEDGMENT

This work was supported by the French National Research Agency (ANR) via the CARNOT LIST AxPQC funding.

REFERENCES

[1] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," July 2015.
[2] G. B. J. D. M. P. G. V. Assche, "The keccak reference," 1 2011.
[3] A. Dolmeta, M. Martina, and G. Masera, "Comparative study of keccak sha-3 implementations," *Cryptogr.*, vol. 7, p. 60, 2023.
[4] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
[5] NIST, "Module-lattice-based key-encapsulation mechanism standard," Aug. 2023.
[6] A. Dolmeta, M. Mirigaldi, M. Martina, and G. Masera, "Implementation and integration of keccak accelerator on risc-v for crystals-kyber," *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 2023.
[7] T. Fritzmann, G. Sigl, and J. Sepúlveda, "Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, p. 239–280, Aug. 2020.
[8] D. Soni and R. Karri, "Efficient hardware implementation of pqc primitives and pqc algorithms using high-level synthesis," *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 296–301, 2021.
[9] A. Dolmeta, M. Martina, and G. Masera, "Hardware architecture for crystals-kyber post-quantum cryptographic sha-3 primitives," *2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, pp. 209–212, 2023.
[10] M. J. Kannwischer, P. Schwabe, D. Stebila, and T. Wiggers, "Improving software quality in cryptography standardization projects," in *IEEE European Symposium on Security and Privacy, EuroS&P 2022 - Workshops, Genoa, Italy, June 6-10, 2022*, (Los Alamitos, CA, USA), pp. 19–30, IEEE Computer Society, 2022.