LOKI Low-Latency Open-Source Kyber-Accelerator IPs

LOKI
Low-latency Open-source Kyber-accelerator IPs

Alessandra Dolmeta[1], Mattia Mirigaldi[2] Maurizio Martina[3], and Guido Masera[4]

Politecnico di Torino, Italy
name.surname@polito.it

Abstract. CRYSTALS-Kyber is a lattice-based key encapsulation mechanism (KEM) recognized as one of the finalists in NIST's post-quantum cryptography (PQC) standardization. Polynomial multiplications and hash functions, as essential operations in lattice-based PQC schemes, poses a significant time consumption challenge with respect to nowadays cryptographic protocols. This work addresses these computational efforts by incorporating LOKI, an accelerator, into a RISC-V microcontroller. By leveraging the accelerator, the performance can be enhanced, contributing to the overall efficiency of Kyber in the fundamental tasks of key generation, encryption, and decryption operations. Through empirical evaluations and benchmarking, the effectiveness and practicality of the proposed hardware architectures are demonstrated, highlighting their potential to advance the field of post-quantum cryptography.

Keywords: Hardware design, Accelerator, Security, Post-Quantum Cryptography, CRYSTALS-Kyber, Keccak, NTT, RISC-V

1 Introduction

Since its discovery in 1994, the Shor algorithm has garnered significant interest in the field of cryptography. In actual fact, it has the potential to break widely-used public-key encryption schemes. Hence, its realization has raised significant concerns about the security of modern cryptographic systems, leading to the urgent need for **post-quantum cryptography** (PQC) in today's digital landscape. PQC refers to cryptographic schemes specifically designed to withstand attacks from quantum computers. The field of PQC encompasses multiple mathematical approaches and primitives, including lattice-based cryptography, code-based cryptography, hash-based signatures, and more. The current goal of researchers and organizations worldwide is to identify and standardize quantum-resistant cryptographic schemes that can seamlessly replace the vulnerable algorithms currently in use. Hence, 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, in 2022, CRYSTALS-Kyber has been the first algorithm to be selected for standardization. [1] However, these algorithms have a non-negligible computational burden

[1] NIST: https://www.nist.gov

if compared to their pre-quantum counterparts. When running them on a microcontroller, performance is not optimal, resulting in slow and energy-consuming execution. To overcome these challenges, **accelerators** are often used. In fact, by offloading tasks to dedicated hardware, microcontrollers can handle complex operations effectively, improving overall performance and energy efficiency. Another important feature of an accelerator is versatility, namely the property of being easily integrable and testable within different microcontrollers. Being open-source, **RISC-V** has been selected as the target. In fact, the main aim of RISC-V is to provide a free and open ISA suitable for processor designs, without imposing a particular implementation technology.

2 Preliminaries

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 encrypts it. Compared to traditional protocols, KEM adds a shared secret to the process, encapsulating and decapsulating the key using the public and the secret keys . In principle, it is a triplet of algorithms: a **key generation algorithm**, an **encapsulation algorithm**, and a **decapsulation algorithm**. Kyber includes **three parameter sets**, corresponding to three security levels of NIST: Kyber512 (I-level), Kyber768 (III-level), and Kyber1024 (V-level). Independently of the levels of security, Kyber always uses the same polynomial arithmetic for all variants. In particular, it works on rings of integer polynomials modulo prime q, denoted as $Z_q[X]$. Polynomials modulo both q and $X^n + 1$ compose the ring $R_q = Z_q[X]/(X^n + 1)$. The only difference is given by the number of polynomial vectors required. A more detailed description of the algorithm can be found in the official documentation [1]. In fact, the aim of this study is to speed up the most onerous part of the algorithm. Indeed, we are focusing our attention on Numeric Theoretic Transform (NTT) and hash functions.

Numeric Theoretic Transform. NTT is a peculiar case of DFT which is conducted in the finite field $Z_q[X]$. It is generally exploited to speed up polynomial multiplication and reduce the complexity of multiplying two n-terms polynomials. NTT is commonly executed through a butterfly unit in Gentleman-Sande (GS) or Cooley-Tukey (CT) configuration. In our case, there is only one unified butterfly. Twiddle Factors are precomputed and stored in tables. Moreover, butterfly computations are executed modulo q employing Barrett and Montgomery reductions [8].

Keccak. Kyber algorithm involves several cryptographic primitives from Secure Hash Algorithm (**SHA 3**) standard. It lists four specific instances of SHA-3 and two extendable-output functions. All instances are based on a 1600-bit state, which is the parallelism of the most important part of these primitives: Keccak permutation function. In each of its 24 rounds, it calls the *f-1600* function, which is characterized by the five consecutive steps: θ, ρ, π, χ, and ι [2].

3 Implementation

In this section, LOKI is presented. The proposed hardware architecture and its main blocks are explained, starting from the implementation and ending with their integration into the PULPissimo microcontroller (RISC-V microcontroller from the open-source PULP[2]). In this work, we concentrate more on the design of the blocks conceived for NTT, INTT, and PWM accelerators. Therefore, for an exhaustive description of the Keccak blocks, refer to [3]. For the implementation, the official c-code reference model provided by NIST was used, to ensure the execution of the algorithm to be compliant with the standards. [3].

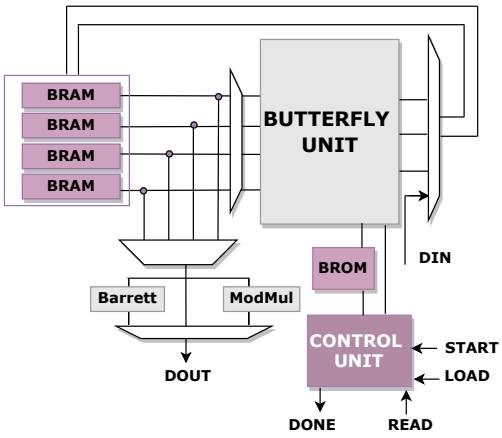

Fig. 1: Architecture's block diagram

Figure 1 represents the block diagram of the architecture. A unified butterfly unit (BU) is used, designed to work both for NTT and INTT. Although the mathematical calculations involved in both algorithms are comparable, their primary distinctions lie in how they store and retrieve data and in the execution of the butterfly operation. The butterfly operation encompasses multiplying by a specific constant, called Twiddle Factor, followed by addition and subtraction within $Z_q[X]$. Twiddle Factors are powers of $\omega_n \in Z_q[X]$, which is the n-th root of unity. These values are precomputed and stored in the BROM memory shown in Figure 1. Then, there are four dual-port BRAMs: the first two are used to store the first input polynomial and output polynomial, while the other two are used to store the second input polynomial [12]. Before any operation, input polynomials are stored in the BRAMs using input multiplexers. Subsequently, the control unit initiates its operation based on start signals, and the resulting polynomial is retrieved using output multiplexers after the operation. As previously stated, LOKI can perform NTT, INTT, and PWM. Both NTT and INTT can be divided into 7 stages, each composed of 128 butterfly operations. NTT and INTT make use of different computational structures: NTT required CT butterfly, while INTT makes use of GS structure. The first follows the decimation-in-time approach, i.e., $a + b \cdot \omega_n (\mathrm{mod}\ q)$ and $a - b \cdot \omega_n (\mathrm{mod}\ q)$, while the latter follows the decimation-in-frequency approach, i.e., $a + b (\mathrm{mod}\ q)$ and $(a - b) \cdot \omega_n (\mathrm{mod}\ q)$. With respect to the reference code mentioned above, the accelerator is able to execute the functions `poly_ntt` and `poly_invntt_tomont`

[2] https://github.com/pulp-platform
[3] Reference code is taken from PQClean: https://github.com/PQClean/PQClean.git

respectively. The architecture of the BU is shown in the left part of Figure 2. There are one integer multiplier, one modular adder, one modular subtractor, and two reduction units (Montgomery and Barrett, outlined in the right part of Figure 2).

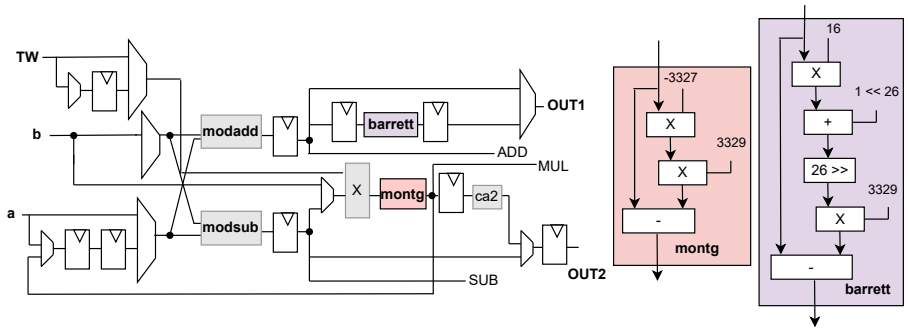

Fig. 2: Block diagram of the butterfly unit

Our BU can also be adapted to carry out PWM operations. This operation corresponds to `basemul` function in the reference model, where 5 multiplication and 2 addition operations are executed. The single butterfly unit takes 906, 906, and 649 clock cycles to execute NTT, INTT, and PMW operations.

Integration. The accelerator is driven in a memory-mapped fashion style and integrated through a robust hardware/software interface. Its versatility is achieved thanks to a *generic register interface*, which is simple, faster, and equipped with protocol adapters from APB, AXI-Lite, and AXI. This makes it easy to attach to most microcontrollers. We use the PULP RISC-V Toolchain[4] with the optimization flag 'O3' to compile the code and increase the stack's memory size. The `randombytes` file is modified as done by [5], generating a pseudo-random sequence of bytes. Considering the latency due to the interchange of data on the bus, the overall architecture takes now 2856, 2856, and 3373 clock cycles to execute NTT, INTT, and PMW operations.

4 Results and Comparison

In this section, we report our implementation results and their comparison with other prior work in the state-of-arts.

Speed-up Factors			
Vers.	**KeyGen**	**Encaps.**	**Decaps.**
I	4,49	5,08	4,15
III	4,35	4,63	4,09
V	4.39	4,81	4,21

First, we evaluate the efficiency of our design, we conduct several tests running Kyber with and without the accelerator. Speed-up factors obtained are reported in the Table aside.

Then, cycle counts and area results of our and related works are reported in

[4] https://github.com/pulp-platform/pulp-riscv-gnu-toolchain

Table 1. LOKI outperforms most of the prior works reported in Table 1. It shows up to 2×/3× better performance for all the three levels of security examined with respect to all the software optimizations [7], [4], [11]. Both in [5] and [8], they proposed a dedicated Post-Quantum Arithmetic Logic Unit in the pipeline of the RISC-V processor. With respect to [5], ours are 0.6/0.68/0.59×, 0.67/0.66/0.62×, and 0.55/0.58/0.55× slower for Kyber512, Kyber768, and Kyber1024 respectively. However, LOKI was not included in the pipeline. Therefore, in contrast to [5] and [8], the higher overhead due to data interchange between core and co-processor must be taken into account. This leads to worse performance than in [5], but better accelerator flexibility. Compared to [8] instead, performance is still better, although the results shown in Table 1 refer to encryption and decryption, and not encapsulation and decapsulation.

Ref.	**Ver.**	**Device**	**LUT/REG/DSP/BRAM**	**KeyGen**	**Encaps.**	**Decaps.**
[5]	I	RISC-V (Pulpino)	2908/179/9/-	150,106	193,076	204,843
	III			273,370	325,888	340,418
	V			349,673	405,477	424,682
[6]	I	RISC-V (VexRiscv)	1842/1634/5/34*	710,00	971,000	870,00
	III			-	-	-
	V			2,203,000	2,619,000	2,429,000
[7]	I	ARM Cortex-M4	-/-/-/-	455,191	586,334	543,500
	III			864,008	1,032,540	969,867
	V			1,404,695	1,605,707	1,525,805
[4]	I	ARM Cortex-M4	-/-/-/-	514,291	652,69	621,245
	III			976,757	1,146,556	1,094,849
	V			1,575,052	1,779,848	1,709,348
[8]	I	CVA6	178/0/5/0*	419,597	438,280	100,796
	III			694,504	731,597	130,348
	V			1,090,458	1,116,462	159,639
[11]	I	ARM Cortex-M4	-/-/-/-	499,000	634,000	597,000
	III			947,000	1,113,000	1,059,000
	V			1,525,000	1,732,000	1,653,000
OUR	I	**RISC-V (Pulpissimo)**	**938/539/10/2.5***	**245,130**	**282,272**	**344,802**
	III			**407,204**	**492,221**	**552,033**
	V			**630,158**	**703,284**	**796,591**
[10]	I	Artix-7	25674/3137/64/6	9,400	19,000	43,000
	III			14,200	26,2000	59,100
	V			18,500	33,700	77,500
[9]	I	Artix-7	7412/4644/2/3	3,800	5,100	6,700
	III			6,300	7,900	10,000
	V			9,400	11,300	13,900

Table 1: Implementation results and comparison to prior works. [*] indicates occupation results referring to NTT/INTT/PWM hardware only.

[10] and [9] have been reported for the sake of completeness, despite they rely on a very different approach. In the case of [10], performances are clearly better, but the area occupied is 27 times larger. Similar reasoning can be made with [9], which achieves better performance by implementing the entire algorithm in hardware, but with significantly higher design effort.

5 Conclusion

This study establishes an ideal starting point for the optimization of the PQC algorithm in today's microcontroller. The hardware design was done by following as closely as possible the code provided by CRYSTALS in order to obtain a version fully compliant with the software implementation. In future works, we would like to study the weaknesses of the algorithm, while analyzing it on the hardware level through electromagnetic and instantaneous power consumption analyses. In this way, we will be able to understand which points are vulnerable to side-channel attacks and modify both the code and the hardware accordingly. To wind up, we highlight the importance of embracing PQC, to pave the way for secure communications and data protection in the era of quantum computing.

References

1. Soni,D., et Al: "Power, Area, Speed, and Security (PASS) Trade-Offs of NIST PQC Signature Candidates Using a C to ASIC Design Flow," 2019 IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, 2019, pp. 337-340, doi: 10.1109/ICCD46524.2019.00054.
2. Bertoni,G., et Al: "FIPS PUB 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions".(2015). https://keccak.team/hardware.html
3. Dolmeta, A., et Al: "Implementation and integration of Keccak accelerator on RISC-V for CRYSTALS Kyber". 20th ACM International Conference on Computing Frontiers, CF23-OSHW. (2023) https://doi.org/10.1145/3587135.3591432
4. Matthias, J., et Al: "pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4", Cryptology ePrint Archive, Paper 2019/844, (2019). https://ia.cr/2019/844
5. Fritzmann, T., et Al: "RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography". IACR TCHES, 2020(4), 239–280. (2020) https://doi.org/10.13154/tches.v2020.i4.239-280
6. Alkim, Erdem et Al: "ISA Extensions for Finite Field Arithmetic: Accelerating Kyber and NewHope on RISC-V". IACR TCHES. 219-242. (2020) 10.46586/tches.v2020.i3.219-242.
7. Alkim, E. et Al: "Cortex-M4 optimizations for R,M LWE schemes", in IACR TCHES, 2020(3), 336–357. (2020). https://doi.org/10.13154/tches.v2020.i3.336-357
8. P. Nannipieri, et Al: "A RISC-V Post Quantum Cryptography Instruction Set Extension for Number Theoretic Transform to Speed-Up CRYSTALS Algorithms," in IEEE Access, vol. 9, pp. 150798-150808, 2021, doi: 10.1109/ACCESS.2021.3126208.
9. Xing, Y., et Al: "A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA". IACR TCHES, 2021(2), 328–356. (2021) https://doi.org/10.46586/tches.v2021.i2.328-356
10. Y. Zhao, et Al: "A High-Performance Domain-Specific Processor With Matrix Extension of RISC-V for Module-LWE Applications," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, no. 7, pp. 2871-2884, July 2022, doi: 10.1109/TCSI.2022.3162593.
11. Leon Botros L. et Al: "Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4". Cryptology ePrint Archive, Paper 2019/489 (2019). https://eprint.iacr.org/2019/489
12. Yaman, F., et Al:"A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2021, pp. 1020-1025, doi: 10.23919/DATE51398.2021.9474139.