Implementation and integration of Keccak accelerator on RISC-V for CRYSTALS-Kyber

Implementation and integration of Keccak accelerator on RISC-V for CRYSTALS-Kyber

ALESSANDRA DOLMETA, Politecnico di Torino, Italy
MATTIA MIRIGALDI, Politecnico di Torino, Italy
GUIDO MASERA and MAURIZIO MARTINA, Politecnico di Torino, Italy

One of the key metrics used for defying the security of the Internet of Things (IoT) is data integrity, which mostly relies on the use of cryptographic hash functions. In the last years, the National Institute of Standards and Technology (NIST) announced SHA-3 as the new standard for better security, and exploit it in most of the current post-quantum cryptographic (PQC) protocols. Nevertheless, the algorithm that it uses, i.e. Keccak, is computationally heavy and consequently limits its utilization on RISC processors used in System on Chip (SoC). In this work, it is proposed a Keccak accelerator to speed up SHA3 computations for the CRYSTALS-Kyber algorithm on the RISCV-based advanced microcontroller PULPissimo. Compared to the plain SW implementation on RISC-V, our results show a speedup factor of up to 2.79 at the expense of a 12.4% resources overhead. The implementation of this work is publicly available at https://github.com/aledolme/pqc_riscv.

CCS Concepts: • **RISC-V accelerator → Keccak-f[1600]**; *PQC-primitives*; NIST; • **HW design** → Modelsim simulation, FPGA synthesis.

Additional Key Words and Phrases: pulpissimo, sponge function

ACM Reference Format:
Alessandra Dolmeta, Mattia Mirigaldi, Guido Masera, and Maurizio Martina. 2018. Implementation and integration of Keccak accelerator on RISC-V for CRYSTALS-Kyber. *ACM Trans. Graph.* 37, 4, Article 111 (August 2018), 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

A cryptographic hash function is designed to satisfy certain security properties, providing a random mapping from a string of binary data to a fixed-size "message digest". In response to several successful attacks on cryptographic hash algorithms, NIST held two public workshops to develop a new standard. The competition ended in 2012, when it announced Keccak as the winning algorithm to be standardized as the new SHA-3 [HASH 2021]. From that moment, there have been several published works dealing with ASIC and FPGA implementation of Keccak and SHA-3 algorithms. In fact, due to the increasing number of users and connected devices in the IoT, it is now critical to ensure the confidentiality of data and, at the same time, the portability and efficiency that typically characterize such devices. In this context, microcontroller units (MCUs) are heavily used, thanks to their flexibility and easiness of updating compared to fixed hardware, but in terms of security, it has been proved that implementing cryptographic functions in hardware is more secure. In this work, the architecture of a Keccak accelerator for RISC-V is proposed, targeting the design for the PULPissimo[1] microcontroller. By doing so, we aim to obtain an efficient device able to faster perform Keccak permutation while maintaining the versatility guaranteed by RISC-V MCU. The accelerator is connected to PULPissimo thanks to a hardware/software interface and tested with CRYSTALS-Kyber, one of the first standardized PQC algorithms by NIST. [NIST 2022].

Authors' addresses: Alessandra Dolmeta, alessandra.dolmeta@polito.com, Politecnico di Torino, Torino, Italy; Mattia Mirigaldi, mattiamirigaldi.98017@gmail.com, Politecnico di Torino, Torino, Italy; Guido Masera; Maurizio Martina, Politecnico di Torino, Torino, Italy.

0730-0301/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

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 Key Encapsulation Mechanism (**KEM**). Making a profiling of all the functions executed by the algorithm, 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 [Dolmeta 2022]. 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**). These primitives are all from the Federal Information Processing Standard (**FIPS-202 standard**) SHA-3 [Guido et al. 2011]. This is a family of sponge functions with members **Keccak[r,c]**, characterized by bit-rate r and capacity c. Their sum determines the width of the Keccak-f permutation, which is 1600-bit. 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**, denoted as: θ, ρ, π, χ and ι. [Guido et al. 2015].

3 HARDWARE AND PULPISSIMO INTEGRATION

The RISC-V PULPissimo microcontroller (Figure 1 - *I*) from the open-source PULP[2] platform project is used and configured to work with the 4-stage pipeline core **RI5CY** [Gautschi et al. 2017]. Then, Keccak and CRYSTALS-Kyber[3] algorithms are compiled using PULP toolchain[4], setting the optimization flag `-O3` and increasing the stack's memory size. The `randombytes` file is modified as done by [Fritzmann et al. 2020], generating a pseudo-random sequence of bytes, while `fips202`'s standard library functions are substituted by the one provided by PULP-team. The accelerator is driven in a memory-mapped fashion style and attached to the SoC through an

[1] https://github.com/pulp-platform/pulpissimo
[2] https://github.com/pulp-platform
[3] Reference code is taken from PQClean: https://github.com/PQClean/PQClean.git
[4] https://github.com/pulp-platform/pulp-riscv-gnu-toolchain.git

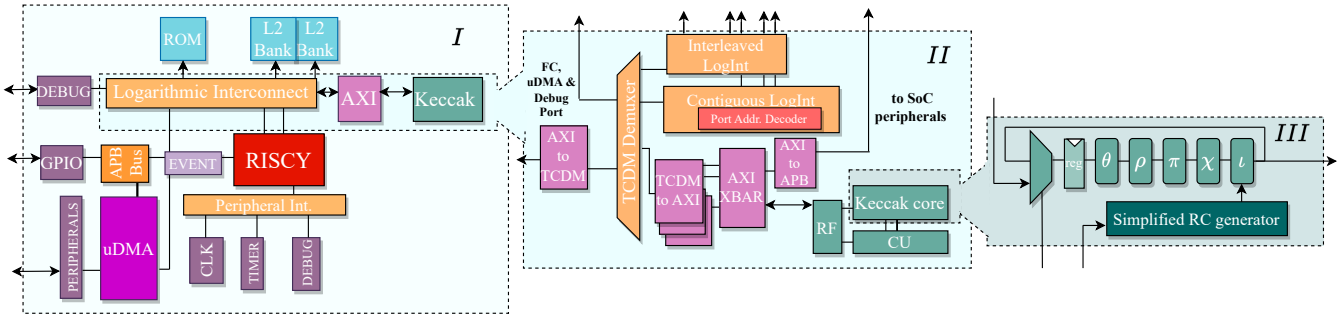

Fig. 1. Simplified architecture overview

AXI plug, as shown in Figure 1 - *II*. Its dependency to PULPissimo is managed by **Bender** [5], while the Keccak register interface is generated using *reggen* tool [6]. The high-speed **Keccak core** [Guido et al. 2015] developed by Keccak teams is optimized accordingly to our application (Figure 1 - *III*). It implements transformation rounds discussed in section 2. As in [Dolmeta 2022], the size of the round constant generator is reduced from 64-bit to one-byte size, simplifying the computation in ι. The accelerator is simulated and tested using ModelSim, exploiting a dedicated driver.

4 RESULTS AND CONCLUSION

Table 1 shows the cycle count results for all three levels of security of CRYSTALS-Kyber on PULPissimo RISC-V platform. The reference code is executed on the original SoC, while the accelerated code is modified accordingly to enable the use of Keccak accelerator.

Table 1. Cycle counts for RISC-V PULPissimo platform

		Reference	Accelerated	Speed-up factor
Kyber512	KeyGen	1,101,598	395,495	2.79
	Encaps	1,435,915	552,827	2.60
	Decaps	1,432,310	726,049	1.97
Kyber768	KeyGen	1,772,967	663,059	2.67
	Encaps	2,280,600	856,258	2.66
	Decaps	2,258,939	1,083,818	2.08
Kyber1024	KeyGen	2,767,127	1,001,350	2.76
	Encaps	3,383,780	1,247,565	2.71
	Decaps	3,352,080	1,523,411	2.20

In [Sanal et al. 2021], an optimized implementation of Kyber encryption schemes for 64-bit ARM Cortex-A processors is presented. Here, the AES accelerator in the target board is used, obtaining the results shown in Table 2. Despite the fact that we accelerated only the permutation step and not the absorbing and squeezing phases of the hash algorithm, our results are better, since our design was targeted at SHA-3. Table 3 shows the resource cost of the presented accelerator. It reports a factor of increase of 12.4 in LUTs and 8.5 in FFs occupation. The majority of the overhead is obtained because of the high dimension of the Keccak state.

Table 2. Cycle counts comparison

	Kyber-512	Kyber-768	Kyber-1024
[Sanal et al. 2021]	1.72/1.88/2.29	1.72/1.81/2.13	1.69/1.75/2.01
Our Work	2.79/2.60/1.97	2.67/2.66/2.08	2.76/2.71/2.20

Table 3. Resource occupation on Xilinx Artix7 a200tfbg676-2

	PULPissimo	PULPssimo with acc.	Increase factor
LUTs	50,132	57,260	+12.4%
FFs	40,478	44,276	+8.5%

As of future work, the proposed accelerator can be extended to perform all the SHA3 primitives, including the absorbing and squeezing phases, so that it could be used for every PQC algorithm and application.

REFERENCES

Alessandra Dolmeta. 2022. *Hardware architecture for CRYSTALS-Kyber cryptographic primitives.* Master's thesis. Politecnico di Torino.

Tim Fritzmann, Georg Sigl, and Johanna Sepúlveda. 2020. RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 4 (Aug. 2020), 239–280. https://doi.org/10.13154/tches.v2020.i4.239-280

Michael Gautschi, Pasquale Davide Schiavone, Andreas Traber, Igor Loi, Antonio Pullini, Davide Rossi, Eric Flamand, Frank K. Gürkaynak, and Luca Benini. 2017. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2700–2713. https://doi.org/10.1109/TVLSI.2017.2654506

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

Bertoni Guido, Daemen Joan, Peeters Micheal, Assche Gilles Van, and Ronny Van Keer. 2015. FIPS PUB 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Retrieved January 21, 2023 from https://keccak.team/hardware.html

HASH 2021. *National Institute of Standards and Technology.* Retrieved January 21, 2023 from https://www.nist.gov/programs-projects/cryptographic-hash-algorithm-competition#:~:text=A%20cryptographic%20hash%20algorithm%20(alternatively,and%20achieve%20certain%20security%20properties.

NIST 2022. . Retrieved January 21, 2023 from https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4

Pakize Sanal, Emrah Karagoz, Hwajeong Seo, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. Kyber on ARM64: Compact Implementations of Kyber on 64-bit ARM Cortex-A Processors. Cryptology ePrint Archive, Paper 2021/561. https://eprint.iacr.org/2021/561 https://eprint.iacr.org/2021/561.

[5] https://github.com/pulp-platform/bender
[6] https://docs.opentitan.org/util/reggen/doc/