RISC-V Based Keccak Co-Processor for NIST Post-Quantum Cryptography Standards

RISC-V Based Keccak Co-Processor for NIST Post-Quantum Cryptography Standards

Alessandra Dolmeta *Student member, IEEE*, Valeria Piscopo, Mattia Mirigaldi *Student member, IEEE*, Maurizio Martina *Senior member, IEEE*, Guido Masera *Senior member, IEEE*
Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italia
name.surname@polito.it

***Abstract*—This paper presents the design and implementation of a RISC-V-based Keccak co-processor optimized for Post-Quantum Cryptography (PQC) algorithms. Leveraging the Core-V eXtension InterFace (CV-X-IF), the co-processor extends the Instruction Set Architecture (ISA) with three custom instructions tailored for cryptographic operations. This allows seamless integration into various PQC schemes, tested across the multiple standards proposed by the National Institute of Standards and Technology (NIST), including CRYSTALS-Kyber, CRYSTALS-Dilithium, SPHINCS+, and FALCON, which are designed to withstand quantum attacks. By employing tightly coupled hardware acceleration, the Keccak co-processor dramatically reduces the computational overhead of hash-based operations central to these algorithms. The implementation is realized on a Xilinx Artix 7 FPGA, achieving a clock cycles' improvement up to 75% and 19% resource overhead. The results presented herein demonstrate significant performance enhancement over the state of the art, underscoring its effectiveness for cryptographic applications.**

***Index Terms*—RISC-V, Core-V eXtension InterFace, Post-Quantum Cryptography, FPGA.**

I. Introduction

Post-quantum cryptography (PQC) has emerged as a critical field due to the threat posed by quantum computing to classical cryptographic systems like RSA and ECC, which are vulnerable to Shor's algorithm. To address this, the National Institute of Standards and Technology (NIST) initiated a multi-round competition to standardize quantum-resistant cryptographic algorithms, culminating in the selection of several draft standards. These drafts include FIPS 203, 204, and 205, specifying cryptographic schemes to resist future quantum attacks. FIPS 203 and FIPS 204 focus on module-lattice-based algorithms from CRYSTALS-Kyber [1] and CRYSTALS-Dilithium [2], respectively, while FIPS 205 specifies a stateless hash-based signature scheme from SPHINCS+ [3]. NIST is also developing a FIPS for the FALCON digital signature algorithm [4]. These schemes provide additional security at the cost of increased computational complexity.

Hardware accelerators offer a solution by offloading intensive tasks, like polynomial multiplication and hash transformations, from the CPU, significantly reducing execution time and energy consumption.

This work was funded by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. This work received funding from the Key Digital Technologies Joint Undertaking (KDT-JU) under grant agreement No 101095947.

The RISC-V Instruction Set Architecture (ISA) is a key advancement in this domain, offering extensive customizability for hardware accelerators. The degree of connectivity among these accelerators is crucial. Fully hardware-based solutions provide optimal performance but limited flexibility. Conversely, loosely coupled accelerators offer more adaptability since they can be shared across different applications. Tightly-coupled acceleration focuses on encapsulating simple, recurrent operations within custom instructions integrated directly into the ISA. This method places the hardware accelerator within the CPU pipeline, allowing for transparent use by application developers. However, integrating tightly coupled accelerators is more complex, as it often requires modifications to the CPU's decode unit and the compilation toolchain.

This paper introduces a novel approach to tightly coupled acceleration using the Core-V eXtension InterFace (CV-X-IF) [5], which simplifies the integration of custom instructions and hardware accelerators into the RISC-V architecture, addressing the challenges of traditional designs while enhancing performance and flexibility. This approach has been successfully applied to Keccak transformation, which plays a vital role in many PQC schemes as part of hashing constructions.

The paper is organized as follows: Section II describes the Keccak algorithm and introduces the different NIST standards in which it is exploited. Section III presents the principles of the new CV-X-IF interface. The architecture and the integration flow are presented in Section IV. Results are shown in Section V. Section VI concludes the paper.

II. NIST Standards

In response to the quantum threat, NIST launched multi-round competitions to identify post-quantum cryptographic algorithms for securing digital communications and data. Among the winners, the Keccak algorithm emerged from the SHA-3 Cryptographic Hash Algorithm Competition, leading to its specification in FIPS 202 [6]. Chosen over alternatives like AES for its superior security properties, the SHA-3 family includes functions such as SHA3-224, SHA3-256, SHA3-384, SHA3-512, and extendable-output functions (XOFs) like SHAKE128 and SHAKE256. This marks a shift towards more resilient hashing mechanisms to counter emerging threats.

Keccak. The Keccak algorithm is characterized by its innovative structure and robust security features. The width of

the state in the Keccak-f permutation is determined by the sum of the rate and the capacity (r and c), resulting in a 1600-bit state comprised of a 5×5 matrix of 64-bit words. Its strength is derived from the intricate transformations executed through 24 rounds of processing ($n_r = 24$). Each round consists of five distinct steps ($n_s = 5$): θ, ρ, π, χ, and ι, which work together to enhance the algorithm's cryptographic strength and resilience against attacks [7].

PQC Algorithms. In addition to FIPS 202, NIST has introduced three new draft standards in 2023 aimed at fortifying cryptographic security against quantum threats:

- **FIPS 203:** Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) – *CRYSTALS-Kyber*
- **FIPS 204:** Module-Lattice-Based Digital Signature Algorithm (ML-DSA) – *CRYSTALS-Dilithium*
- **FIPS 205:** Stateless Hash-Based Digital Signature Algorithm (SLH-DSA) – *SPHINCS+*

A forthcoming FIPS 206 will introduce a Fast Fourier Transform-based NTRU-Lattice Signature Algorithm (FN-DSA), derived from FALCON. While centered on lattice-based cryptography, these standards aim to strengthen cryptographic frameworks for the post-quantum era, reaffirming Keccak's role as a foundational hashing standard.

III. RISC-V and the Core-V eXtension interface

RISC-V is an open standard ISA that enables extensibility and customization, making it suitable for various computing applications, including cryptography. Its support for both standard and specialized instructions allows for significant performance and efficiency improvements, reducing memory costs and enhancing speed in implementing cryptographic algorithms. However, integrating custom instructions requires a thorough understanding of the associated costs. The SHA-3 function, for instance, is computationally intensive on lightweight embedded RISC processors due to the large size of its state. Several studies have explored the integration of custom instructions into RISC-V for cryptographic applications. For example, [8] were pioneers in proposing custom extensions for SHA-3 implementation on an FPGA utilizing a 32-bit LEON3 processor. Other research introduced various application-specific instruction set processors (ASIPs) utilising additional instructions or auxiliary registers to enhance SHA-3 performance [9], [10], [11]. Subsequent studies explored the parallelizability of the process by employing vectorized instructions or vector co-processors, incorporating multiple instruction extensions to optimize data processing in parallel [12], [13]. [14] implemented a similar accelerator but connected it to the CPU through a bus and leveraged drivers instead of relying on custom instructions. Also, recent advancements by the RISC-V Cryptography Extensions Task Group have led to the publication of the Vector Crypto Draft, although no vector extensions for Keccak are currently included in the draft [15].

Regardless of the integration method employed, several critical factors must be considered. In the case of loosely coupled accelerators, the most significant aspect is the interface, which ensures communication between the accelerator and the main processor. In contrast, tightly coupled accelerators require careful attention to the compatibility of new instructions and their seamless integration into the core's pipeline. This integration often necessitates modifications not only to the hardware components (i.e., decode and execution stages) but also to the software toolchain.

Core-V eXtension interface. The CV-X-IF [5] stands out as an optimal solution that bridges the gap between loosely and tightly coupled accelerators. As a dedicated interface, it allows for direct connectivity to the core without necessitating modifications to the core's pipeline or the software toolchain. This feature enables developers to harness the advantages of tightly coupled acceleration, exploiting dedicated channels for offloading instructions to accelerators, regardless of their type, and for writing back the resulting data using standardized signals. The CV-X-IF flexibility facilitates the integration of specialized functionalities into existing RISC-V architecture, making the processor adaptable to evolving application demands. The CV-X-IF was first introduced as a feature of the CV32E40X core [16], a standard-compliant 32-bit RISC-V processor, supporting the base RISC-V integer instruction set along with many standard extensions. Recently, the CV-X-IF was successfully integrated into the CV32E40P core, resulting in a variant labeled CV32E40PX [17]. This core features a dispatcher transmitting all required signals to the interface without modifying the CPU, merely redirecting them to the CV-X Interface. The main signals are depicted in Fig. 1.

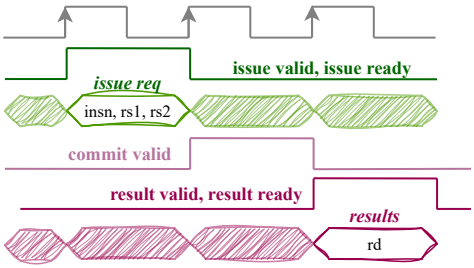

Fig. 1. Core-V eXtension interface signals.

We implement a unit responsible for decoding incoming instructions, tracking the identification of the offloaded instruction, reading the source registers, and managing the destination register address for the result. In Fig. 2 we refer to this unit as CV-X-IF decoder. This decoder examines the opcode extracted from the instruction field of the issue request signal (*issue req*). If there is a match with one of the custom instruction opcodes, it activates a valid signal of the response package (*issue valid*). This indicates the accelerator's readiness to receive data. Then, the co-processor samples its inputs from the core's register file and performs its computation. Subsequently, if the core is ready to receive the result and the result is valid (*result ready* and *result valid*), the result is written into the core's register file, at the destination address previously saved. This is done through the result interface (*result*). Further information

about the protocol of this interface can be found in the original repository [1].

IV. Hardware Implementation and Integration

Fig. 2 shows the architecture of our co-processor. It has been integrated into X-HEEP (eXtendable Heterogeneous Energy-Efficient Platform), an open-source RISC-V microcontroller described in SystemVerilog [18].

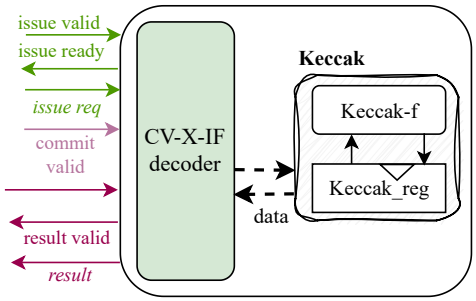

Fig. 2. Co-processor Implementation Scheme.

As mentioned in Sec.III, the SHA-3 state size exceeds the capacity of the available registers of a common RISC-V 32-bit architecture. Considering a common execution of the algorithm, this implies the swapping of portions of the state between memory and registers during computations. However, memory load and store operations are relatively costly. This is one of the main reasons why SHA-3 is so computationally intensive on lightweight embedded RISC-V. Indeed, it is crucial to minimize the frequency of load/store instruction for an efficient implementation. Therefore, we propose a dedicated co-processor register, Keccak_reg of Fig. 2, that can maintain the state, significantly reducing the number of data movements compared to relying solely on the baseline core's register file. Additionally, we implement two custom instructions to facilitate efficient loading and storing of data to this co-processor register: `load_state` and `store_state`. The `load_state` instruction is called 25-times, to upload the state matrix into Keccak_reg 64 bits at a time. The `store_state` instruction is called 50-times, to save the result back to the core's register file 32 bits at a time. The third custom instruction that has been introduced is `start_Keccak`. This instruction initiates the 24-round process of Keccak permutation. In this way, instead of computing each step one by one and using many loads and stores, we can process the entire state in just 24 clock cycles. To leverage these new instructions, inline assembly is used. In our implementation, custom instructions are defined using the `.insn` directive, facilitating the creation of new instructions that interact with the co-processor. By specifying the opcode, function codes, and operands, these instructions execute the required instruction. Only R-type instructions have been used, with two source registers and one destination register. The instruction format is the following: "$.insn\ \ r\ \ 0x4b, 0x04, 0x\beta, rd, rs1, rs2$", where β = 0, 1, 2, depending on the instruction required.

[1]https://github.com/openhwgroup/core-v-xif/tree/main

V. Results

The complete architecture is developed at the RTL using SystemVerilog and implemented on the Zynq UltraScale+ ZCU104 board (xczu7ev-ffvc1156-2-e). The synthesis and implementation are performed using Xilinx Vivado, with a global clock of 300 MHz. The system's top-level operates at a frequency of 50 MHz, enabling successful binary loading of applications via JTAG and execution with GDB. The `-O2` flag is used both for pure software and accelerated simulations. The performance of each cryptographic scheme was evaluated using performance counters. In the reference implementation, the Keccak function takes 56,556 clock cycles to execute, but with the co-processor, the number of cycles drops to 553, yielding a 102× speed-up. It is worth noting that 25 clock cycles are spent on the transformation itself, as one round of Keccak can be performed in a single clock cycle. The remaining cycles are used for loading and storing the state from the core to the co-processor and vice-versa, which involves memory transfer operations. The area utilization results are presented in Tab. I.

TABLE I
Resource utilization (Artix-7 FPGA)

	LUT	Registers	CLB
SoC top-level	36,537	30,403	8,941
◇ CV32E40PX	6,340	2,165	1,437
◇ Keccak Wrapper	6,591	3,372	1,425
○ Keccak	5,409	1,615	1,190
○ Keccak_reg	0	1,600	676
○ CV-X-IF decoder	1,181	125	878

In terms of resource usage, specifically the occupation of Configurable Logic Blocks (CLBs), adding the co-processor introduces a 19% overhead. This value is obtained by comparing the total area occupied by the entire system-on-chip (SoC) with and without the Keccak co-processor.

TABLE II
Comparisons with the state-of-the-art.

Ref.	Platform	Cycles Improvement	Area Increase	Efficiency [Cycles/Area]]
[8]	LEON3	51%	9%	5.67
[9]	MIPS	20%	9%	2.22
		38%	26%	1.46
[10]	PIC24	30%	-	-
[19]	ASIC	96%	-	-
[11]	RISC-V	99%	25%	3.96
[20]	RISC-V	97%	-	-
[14]	RISC-V	95%	14%	6.7
OUR	**RISC-V**	**99.2%**	**19%**	**5.22**

Comparing area and performance fairly with the state-of-the-art works mentioned in Sec. III is not straightforward, as we are among the first to utilize this new interface. Consequently, Tab. II presents a comparison of various implementations, highlighting the performance improvements and the area overhead in percentage terms. To enable a clearer comparison between the different works, we have introduced a more

balanced metric: the ratio between these two percentage values (performance gain and area overhead). This ratio estimates the overall efficiency of the accelerator, offering insight into how much performance improvement is achieved relative to the hardware cost, thus providing a better evaluation of its overall cost-effectiveness. In Tab. II, the work in [19] is the only one using the CV-X-IF interface but reports area results for the entire co-processor instead of focusing solely on Keccak, complicating efficiency evaluations. Despite this, we observe that their Keccak-f[1600] transformation takes 1800 clock cycles, while our implementation achieves the same transformation in just 553 clock cycles. Both approaches use a dedicated register file in the co-processor; however, while our implementation handles the entire transformation, [19] only implements a portion of it, specifically introducing custom instructions such as `XOR5`, `ROLX`, `ANDX`, and `XROL`. This challenge also applies to other works. For those where we can assess comparison metrics, our results show superior performance. While [14] achieves the best efficiency, it implements a loosely coupled accelerator and reports area overhead for the entire microcontroller, not for the core.

TABLE III
CLOCK CYCLES FOR DIFFERENT ALGORITHMS

Algorithm	Step	*Original*	*Optimized*	*Calls*
Kyber512 (K512)	**Keygen**	997,857	333,929	27
	Encaps	1,067,675	426,729	26
	Decaps	1,206,794	563,803	26
Kyber768 (K768)	**Keygen**	1,620,943	560,407	43
	Encaps	1,762,873	677,058	44
	Decaps	1,945,499	862,524	44
Kyber1024 (K1024)	**Keygen**	2,581,589	858,322	70
	Encaps	2,745,261	997,337	71
	Decaps	2,980,456	1,232,600	71
Dilithium2 (D2)	**Keygen**	3,792,362	1,222,174	105
	Sign	5,430,085	2,733,144	110
	Verify	3,901,011	1,474,879	99
Dilithium3 (D3)	**Keygen**	6,608,688	1,974,106	188
	Sign	11,352,546	5,981,893	218
	Verify	6,599,528	2,310,159	174
Dilithium5 (D5)	**Keygen**	11,330,213	3,264,290	327
	Sign	28,706,078	13,280,212	1317
	Verify	11,383,618	3,717,003	311
SPHINCS+128f-s (S128f_s)	**Keygen**	148,584,835	37,374,556	4527
	Sign	3,693,600,067	929,271,165	112,527
	Verify	201,941,915	51,039,362	6,143
SPHINCS+128f-r (S128f_r)	**Keygen**	285,782,621	70,044,757	8,774
	Sign	2,725,054,029	1,720,606,941	215,526
	Verify	411,968,696	98,856,973	12,734
Falcon512 (F512)	**Keygen**	171,175,811	160,845,231	484
	Sign	71,682,361	71,628,546	10
	Verify	614,828	417,183	9
Falcon1024 (F1024)	**Keygen**	611,283,802	543,530,620	3,136
	Sign	157,683,412	156,917,679	18
	Verify	1,261,835	887,748	17

Tab. III reports the clock cycles required by each step of the applications. For Kyber, there are Key Generation, Encapsulation, and Decapsulation, while for the different signatures, Key Generation, Signature, and Verification. Further details on the algorithms' different versions can be found in [1], [2], [3], [4]. The *Original* column refers to the pure software implementations, whereas, in the *Optimized* one, the inline assembly instructions are inserted to invoke the co-processor. *Calls* column represents the number of times the Keccak function is called during the execution. The respective speed-ups are depicted in Fig. 3, evaluated as the percentage ratio between the difference of the original and optimized execution times relative to the original time. In most cases, the speedup is higher than 60%. The lowest results are obtained for the FALCON scheme, in which the Keccak permutation plays a minor role (Tab. III).

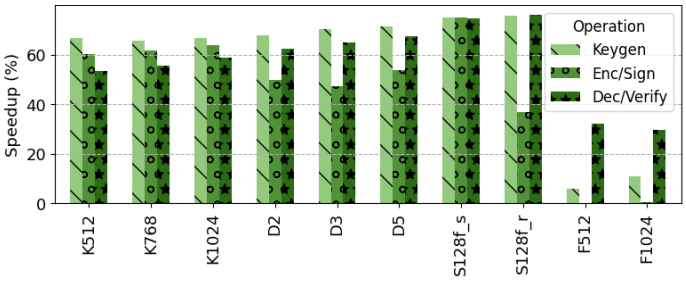

Fig. 3. Speed-up percentages for Key Generation, Encapsulation/Sign, and Decapsulation/Verify operations for each algorithm.

While many works have introduced accelerators for PQC standards fair performance comparisons remain difficult due to varying implementation strategies and specialized accelerators. For instance, [19] is one of the few works utilizing the CV-X-IF interface and among the first to introduce RISC-V instruction set extensions (ISE) for multiple PQC standards. However, despite outlining the algorithms supported by the co-processor, it does not provide comprehensive performance results for the entire set of algorithms, making direct comparisons difficult. Similarly, state-of-the-art works without the CV-X-IF interface incorporate multiple accelerators to maximize speed-up, further complicating comparisons. For example, [11] presents an architecture with accelerators for the Number Theoretic Transform (NTT), Karatsuba/Toom-Cook polynomial multiplication, and a SHA-3-based Pseudo Random Number Generator (PRNG), achieving nearly a 90% speed-up for Kyber. In [21], a co-processor with NTT and Keccak accelerators reduces the cycle count of Dilithium and FALCON by an average of three times compared to software implementations. Additionally, [22] achieves an 80% speed-up for SPHINCS+128f using a crypto accelerator equipped with AES-128/256, SHA2-256, and ECC cores. In contrast, our work focuses on designing a co-processor that supports multiple PQC schemes, with the primary goal of showcasing the flexibility and potential of the CV-X-IF interface rather than maximizing performance for individual algorithms.

VI. CONCLUSIONS

The CV-X-IF interface marks a significant advancement in RISC-V acceleration, and our research has showcased its potential within the CV32E40PX core. With growing interest in this interface, especially following a recent release candidate, we aim to further explore its features. Future work will focus on optimizing performance, enhancing security, and developing new instructions for post-quantum cryptography, while expanding support for additional accelerators and algorithms.

REFERENCES

[1] R. Avanzi, J. Bos, and et al., "CRYSTALS-Kyber Documentation," 2021. Available at: https://pq-crystals.org/kyber/resources.shtml.

[2] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium Documentation," 2021. Available at: https://pq-crystals.org/dilithium/resources.shtml.

[3] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS+ signature framework." Cryptology ePrint Archive, Paper 2019/1086, 2019.

[4] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over NTRU." Submission to the NIST post-quantum cryptography standardization process, 2018.

[5] "CV-X-IF eXtension Interface." https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/en/latest/x_ext.html. Accessed: April 3, 2024.

[6] N. I. of Standards and Technology, "FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," 2015. https://doi.org/10.6028/NIST.FIPS.202.

[7] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, "FIPS PUB 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," 2015.

[8] Y. Wang, Y. Shi, C. Wang, and Y. Ha, "FPGA-based SHA-3 acceleration on a 32-bit processor via instruction set extension," in *IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pp. 305–308, 2015.

[9] M. A. Elmohr, M. A. Saleh, A. S. Eissa, K. E. Ahmed, and M. M. Farag, "Hardware implementation of a SHA-3 application-specific instruction set processor," in *28th International Conference on Microelectronics (ICM)*, pp. 109–112, 2016.

[10] J. H.-F. Constantin, A. P. Burg, and F. K. Gürkaynak, "Instruction set extensions for cryptographic hash functions on a microcontroller architecture," in *23rd International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 117–124, 2012.

[11] 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*, vol. 2020, no. 4, pp. 239–280, 2020.

[12] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, "VPQC: A Domain-Specific Vector Processor for Post-Quantum Cryptography Based on RISC-V Architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2672–2684, 2020.

[13] J. Rao, T. Ao, S. Xu, K. Dai, and X. Zou, "Design Exploration of SHA-3 ASIP for IoT on a 32-bit RISC-V Processor," *IEICE Transactions on Information Systems*, vol. 101-D, pp. 2698–2705, 2018.

[14] A. Dolmeta, M. Mirigaldi, M. Martina, and G. Masera, "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)*, pp. 381–382, Association for Computing Machinery, New York, NY, USA, 2023.

[15] R.-V. C. E. T. Group, "riscv-crypto: Risc-v cryptography extensions." https://github.com/riscv/riscv-crypto, 2024. Accessed: 2024-10-11.

[16] O. Group, *CV32E40X User Manual*, 2024. Accessed: 2024-06-11.

[17] ESL-EPFL, "cv32e40px: Risc-v core with extensions for cryptography and machine learning." https://github.com/esl-epfl/cv32e40px, 2024. Accessed: 2024-10-11.

[18] S. Machetti, P. D. Schiavone, T. C. Müller, M. Peón-Quirós, and D. Atienza, "X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators," 2024.

[19] J. Lee, W. Kim, and J.-H. Kim, "A programmable crypto-processor for National Institute of Standards and Technology post-quantum cryptography standardization based on the RISC-V architecture," *Sensors*, vol. 23, no. 23, p. 9408, 2023.

[20] Z. Ye, R. Song, H. Zhang, D. Chen, R. C.-C. Cheung, and K. Huang, "A highly-efficient lattice-based post-quantum cryptography processor for IoT applications," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 024, no. 2, pp. 130–153, 2024.

[21] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl, "Post-quantum signatures on RISC-v with hardware acceleration." Cryptology ePrint Archive, Paper 2022/538, 2022.

[22] U. Banerjee, S. Das, and A. P. Chandrakasan, "Accelerating post-quantum cryptography using an energy-efficient tls crypto-processor," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.