

Design and Implementation of Custom Instructions for Accelerating Post-Quantum Cryptography in RISC-V Architectures

Emma Reyes Ramos Chinae
*Institute for Applied Microelectronics
(IUMA)*

Agoney J. Moreno Rodríguez
Arquimea Research Center

Pedro P. Carballo, Ph.D.
*Institute for Applied Microelectronics
(IUMA)*

Abstract—Advances in quantum computing pose a threat to the classical cryptographic systems on which current IT security is based. To mitigate this potential vulnerability, the National Institute of Standards and Technology (NIST) has led a process to standardise new Post-Quantum Cryptography (PQC) schemes. This process involved several rounds of evaluation and continuous elimination, culminating in the standardisation of ML-KEM as a key encapsulation algorithm, and ML-DSA and SLH-DSA as digital signature algorithms. Integrating these algorithms into embedded systems with limited resources poses a challenge due to the size of the keys used in these standards and the computational power required.

In this context, the choice of computing architecture plays a key role. Open-source architectures are gaining particular prominence due to the versatility and adaptability they offer. Developments in RISC-V architectures are particularly noteworthy, with strong support led by the RISC-V International Foundation. This architecture features an open and modular design, enabling it to adapt to all types of applications.

This work presents a RISC-V instruction set extension to accelerate the ML-DSA post-quantum cryptographic algorithm. Following an evaluation of open-source cores, the NEORV32 was selected for its modularity and Custom Functions Unit (CFU). After securing key generation with the core's TRNG, a dedicated Keccak accelerator was integrated into the CFU via five custom R-type instructions to target SHAKE operations. Implementation on an Artix-7 FPGA achieved a 60x local speedup for SHAKE. However, the 1.03x overall system acceleration underscores the need for further optimizations as dictated by Amdahl's Law.

Index Terms—Post-Quantum Cryptography, RISC-V, ML-DSA, Hardware Acceleration, Keccak, NEORV32

I. INTRODUCTION

Cryptography forms the foundation of modern cybersecurity, ensuring data confidentiality and integrity [1]. While traditional public-key algorithms like RSA and ECC are currently secure, the emergence of Cryptographically Relevant Quantum Computers (CRQCs) threatens to break them [2]. Quantum algorithms, notably Shor's algorithm [3], can efficiently solve the integer factorization and discrete logarithm problems, rendering current asymmetric encryption vulnerable. Similarly, Grover's algorithm [4] reduces the effective key strength of symmetric ciphers. Furthermore, concepts like "Harvest now, decrypt later" pose an immediate threat, as malicious actors can store intercepted encrypted data today to decrypt it once quantum computers mature [5].

To address these threats, Post-Quantum Cryptography (PQC) provides algorithms resistant to both classical and quantum attacks that can be integrated into current transistor-based infrastructures [6]. PQC relies on mathematical problems, primarily lattice-based, hash-based, code-based, and multivariate approaches.

Concluding a standardization process initiated in 2016, NIST published the first Federal Information Processing Standards (FIPS) for PQC in 2024: ML-KEM (as FIPS203 and based on Kyber) [7], ML-DSA (as FIPS204 and based on Dilithium) [8], and SLH-DSA (as FIPS205 and based on SPHINCS+) [9]. Implementing these algorithms on resource-constrained embedded systems presents significant challenges due to increased key sizes, high computational demands, and larger memory footprints.

To overcome these performance bottlenecks, specialized hardware acceleration is required. Open-source Instruction Set Architectures (ISAs) like RISC-V [10] offer the flexibility to integrate custom extensions tailored to specific computational needs.

Recent works, such as the HORCRUX architecture [11] and RISQ-V [12], have demonstrated the effectiveness of tightly coupled RISC-V accelerators for PQC. The focus of this work is to research, design and develop an implementation of a custom instruction set in RISC-V architecture for a post-quantum digital signature algorithm. In particular, the aim is to design a hardware accelerator using customised instructions on a RISC-V core. The intention is to reduce the computational load on the software required to execute a cryptographic algorithm.

II. ML-DSA POST-QUANTUM DIGITAL SIGNATURE

A digital signature is the electronic equivalent of a handwritten signature. Specifically, this cryptographic mechanism is used to guarantee three information security properties:

- **Authenticity:** the alleged signer is the one who actually originated the information,
- **Non-repudiation:** the signer cannot deny having signed the information,
- **Integrity:** ensures that the data has not been modified after being signed.

ML-DSA (Module-Lattice-Based Digital Signature Algorithm), standardized as FIPS 204, is a post-quantum digital signature scheme derived from CRYSTALS-Dilithium [8]. It

provides strong security based on the hardness of module lattice problems and is designed to minimize public key and signature sizes.

The ML-DSA standard specifies three parameter sets: ML-DSA-44 (Security Level 2, comparable to AES-128), ML-DSA-65 (Level 3, AES-192), and ML-DSA-87 (Level 5, AES-256). As security levels increase, the sizes of the public key, private key, and signature also grow. For instance, ML-DSA-44 requires a 2560-byte private key and generates a 2420-byte signature, imposing significant storage and transmission overheads on embedded systems.

The ML-DSA execution comprises three main functions: key generation, signature generation, and verification. During key generation, a random seed is expanded using Extendable-Output Functions (XOFs), specifically SHAKE-128 and SHAKE-256. The signature process incorporates rejection sampling to ensure that the signature does not leak information about the private key. The extensive use of SHAKE functions in these steps, combined with the Number Theoretic Transform (NTT) for polynomial multiplication, creates a significant bottleneck in software-only implementations.

III. RISC-V ARCHITECTURE AND CORE SELECTION

RISC-V is an open, modular ISA that avoids microarchitecture-specific details, allowing a wide variety of hardware implementations. Its unprivileged ISA defines a base integer instruction set, RV32I, and standard extensions (e.g., M for integer multiplication or C for compressed instructions). There is also space set aside for future extensions and for custom extensions. The privileged ISA handles execution modes - Machine, Supervisor, User -, Control and Status Registers (CSRs), and mechanisms for exception and interrupt handling.

To select a suitable RISC-V core for this project, several open-source implementations were evaluated based on their hardware description language (VHDL/Verilog), documentation, and instruction set extensibility. Three candidate cores were synthesized and analyzed on a Nexys Video FPGA board (Artix-7): PicoRV32 [13], Ibex [14], and NEORV32 [15]. This comparative analysis was evaluated in terms of Power, Performance, and Area (PPA) metrics to determine the most suitable architecture for hardware acceleration.

- **PicoRV32:** Highly optimized for area, utilizing 1,379 LUTs and operating at a maximum frequency of 132 MHz, with a total power consumption of 180 mW.
- **Ibex:** Backed by strong industrial support, it required 3,236 LUTs and achieved a lower frequency of 53 MHz, consuming 366 mW.

- **NEORV32:** Utilized 2,074 LUTs, achieved 109 MHz, and consumed 179 mW. Additionally, it features a highly documented, modular design with a Custom Functions Unit (CFU) tailored for easily implementing custom RISC-V instructions.

The Normalized Sum of Resource Utilization (SNRU) [16] metric was employed to compare the designs comprehensively, as shown in Table I. While PicoRV32 exhibited the lowest resource utilization, the NEORV32 core was ultimately selected due to its highly modular VHDL design, excellent documentation, and the inclusion of CFU specifically intended for user-defined instruction extensions.

IV. HARDWARE/SOFTWARE CO-DESIGN

The implementation of ML-DSA was carried out using the `mldsa-native` [17] library in C, compiled for the NEORV32 architecture using the `riscv32-unknown-elf-gcc` toolchain. To ensure cryptographic security during key generation, the software's default random number generator was replaced with the NEORV32's True Random-Number Generator (TRNG). The TRNG derives physical entropy from free-running ring oscillators, ensuring true randomness independent of the FPGA platform.

A. Custom Functions Unit (CFU) Integration

To accelerate the hashing operations, a hardware module for Keccak was integrated into the CFU. A block diagram of this integration is presented in Fig. 1. The CFU allows the execution of custom instructions using the reserved `CUSTOM-0` opcode in the RISC-V RV32 instruction space. Five custom R-type instructions were defined and mapped to C intrinsic functions:

- 1) `keccak_hw_init`: Configures the Keccak core for either SHAKE128 or SHAKE256 and specifies the output size.
- 2) `keccak_hw_data`: Injects 64 bits of input data using registers `rs1` and `rs2`.
- 3) `keccak_hw_data_last`: Injects the final input data block, triggering the hash computation.
- 4) `keccak_hw_read`: Extracts the 32-bit output hash words using the destination register `rd`.
- 5) `keccak_hw_reset`: Asserts a reset signal to clear the accelerator's internal state.

TABLE I

COMPARISON OF PPA (POWER, PERFORMANCE, AND AREA) METRICS ON AN AMD ARTIX 7™ DEVICE FOR THE EVALUATED RISC-V IMPLEMENTATIONS.

Core	Resources			Performance	Power		
	LUTs	Registers	DSPs		$F_{\{\max\}}$ (MHz)	Total (mW)	RISC-V Core (mW)
PicoRV32	1382	796	4	0.015	132	180	19
Ibex	3255	1367	13	0.038	53	366	17
NEORV32	2077	1559	4	0.021	109	179	23

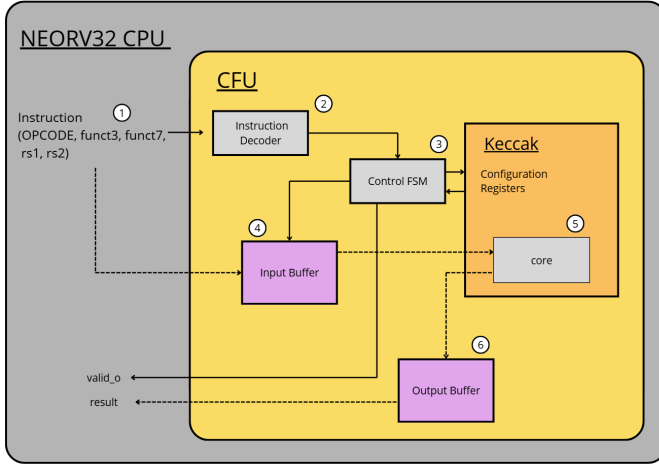


Fig. 1. Block diagram of the designed CFU.

A finite state machine (FSM) controls the CFU, managing data buffers and handling the handshake with the Keccak core. The FSM transitions through five main states: *S_IDLE*, *S_SEND*, *S_WAIT_READ*, *S_READ*, and *S_DONE*, as outlined in Fig. 2.

Initially, the hardware state is cleared using the *keccak_hw_reset* instruction. Then, the *keccak_hw_init* instruction configures the core, establishing the desired output length and the specific cryptographic task. During data injection, the CFU aggregates the 32-bit RISC-V register inputs into 64-bit blocks to match the Keccak core’s native data width. Upon receiving the *keccak_hw_data_last* instruction, this buffered data is fed into the Keccak accelerator. Once the permutation is complete, the resulting hash is stored in an output buffer, which the CPU retrieves sequentially via the *keccak_hw_read* instruction.

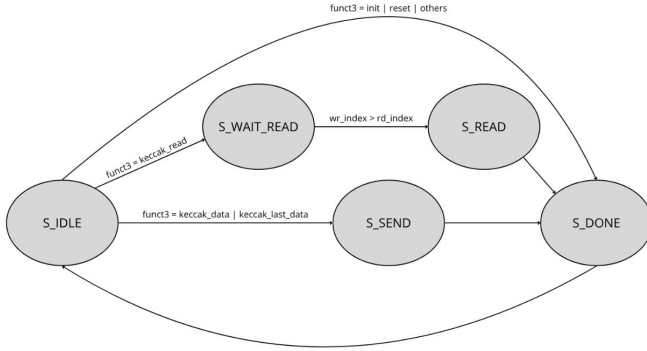


Fig. 2. CFU control FSM.

B. System Verification

To ensure the correctness of the proposed architecture, verification methodology was employed. Prior to physical implementation, the CFU’s functionality—including FSM transitions, data alignment, and the Keccak handshake—was verified cycle-by-cycle through VHDL simulation using GHDL and GTKWave. Once synthesized and deployed on the FPGA, the complete hardware/software co-design was

debugged on-chip. By exposing the NEORV32’s JTAG port, in-circuit debugging was performed using OpenOCD and GDB, allowing for precise validation of the execution flow and memory states in the real hardware environment.

V. RESULTS AND EVALUATION

The full system, including the NEORV32 CPU, TRNG, and the Keccak-enhanced CFU, was synthesized and implemented on the Xilinx Artix-7 FPGA (Nexys Video).

A. Resource Utilization and Power

The baseline NEORV32 core, including TRNG and debug wrapper, utilized 3,232 LUTs, 2,248 registers, and 34 Block RAMs (BRAMs). After integrating the Keccak CFU, resource utilization increased significantly to 37,089 LUTs and 39,769 registers. This dramatic increase was primarily due to the VHDL array structures used for input and output data buffering within the CFU, which synthesized into distributed logic rather than dedicated memory blocks. The full implementation is presented in Fig. 3.

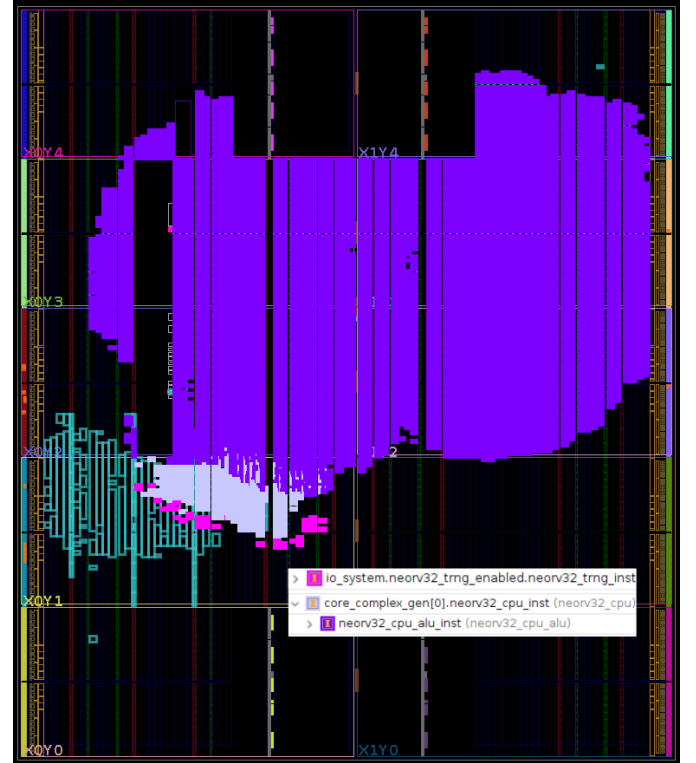


Fig. 3. Results of the implementation of CFU and TRNG in NEORV32..

Despite the larger physical footprint, the design successfully met all timing constraints for a 100 MHz clock frequency, achieving a Worst Negative Slack (WNS) of 0.214 ns. Total dynamic power consumption increased to 222 mW, with a total estimated power of 376 mW.

B. Performance Speed-up

The custom instructions reduced the execution time of the SHAKE operations. Generating a 128-byte SHAKE256 hash from a 34-byte input required 119,813 clock cycles in the pure

software implementation. With the hardware accelerator, this same operation was completed in just 2,171 cycles, yielding a local speed-up factor of approximately 55x. For larger inputs requiring multiple Keccak permutations, the local speed-up reached 61x. Overall, the hardware-accelerated SHAKE operations achieved an average local speed-up of 60.7x, as shown in Table II.

To evaluate the impact on the complete ML-DSA key generation phase, Amdahl's Law [18] was applied. The pure software key generation required 40,366,001 cycles. Since the SHAKE operations constitute only a small fraction (around 3.24%) of the total key generation time, the theoretical global speed-up was limited to approximately 1.03x. The accelerated implementation reduced the key generation time to 39,092,361 cycles.

TABLE II
EXECUTION TIMES AND LOCAL ACCELERATION FACTOR FOR SHAKE OPERATIONS.

(ML-DSA-44)	SHAKE software	SHAKE hardware	Local acceleration
1 st execution	119813	2171	x55
2 nd execution	1186782	19351	x61
TOTAL	1306595	21522	x60.7

C. Comparison with State-of-the-Art

When compared to similar RISC-V PQC accelerators, this work demonstrates highly efficient local acceleration for hashing. However, works like Karl et al. (2024) [19], which also offload polynomial arithmetic (NTT) to tightly coupled accelerators, achieve much higher global speed-ups (up to 3.33x for key generation). This highlights that while hashing is a significant component of ML-DSA, full acceleration requires addressing the mathematical operations over polynomial rings.

VI. CONCLUSION AND FUTURE WORK

This work demonstrates the integration of a custom hardware accelerator for post-quantum cryptography within a RISC-V processor. Using the NEORV32 Custom Functions Unit, the Keccak-based hashing operations essential for ML-DSA were accelerated by a factor of 60x locally.

However, as dictated by Amdahl's Law, the overall system acceleration is limited to approximately 1.03x. This marginal improvement occurs because only the SHAKE operations during keypair generation were targeted, leaving other computationally intensive bottlenecks in the ML-DSA execution unaddressed.

Future work will focus on optimizing the CFU's logic area by replacing distributed register arrays with Block RAM (BRAM). Additionally, improving the data transfer mechanisms between the CPU and CFU (e.g., using Direct Memory Access or ping-pong buffers) will reduce communication overhead. Finally, to achieve a more substantial global speed-up, future iterations must explore hardware acceleration for other critical ML-DSA functions, such as the Number Theoretic Transform (NTT).

REFERENCES

- [1] D. Boneh and V. Shoup, *A Course in Cryptography*. Online Manuscript, 2023. [Online]. Available: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_6.pdf
- [2] Ashish Kundu, Ramana Kompella, "The Quantum-Cryptographic Co-evolution." Accessed: Jul. 13, 2026. [Online]. Available: <https://arxiv.org/html/2604.02591v1>
- [3] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, Nov. 1994, pp. 124–134. doi: 10.1109/SFCS.1994.365700.
- [4] L. K. Grover, "A fast quantum mechanical algorithm for database search." Accessed: Jun. 08, 2026. [Online]. Available: <http://arxiv.org/abs/quant-ph/9605043>
- [5] Palo Alto Networks, "Harvest Now, Decrypt Later: Quantum Security Risk." Accessed: Jun. 08, 2026. [Online]. Available: <https://www.paloaltonetworks.com/cyberpedia/harvest-now-decrypt-later-hndl>
- [6] N. Bindel, M. Mosca, and B. Munson, "The Quantum Threat to Cybersecurity and Privacy," in *The Boundaries of Data*, 1st ed., London: Routledge, 2025, pp. 35–51. doi: 10.5117/9789463729192_CH03.
- [7] National Institute of Standards and Technology (US), "Module-lattice-based key-encapsulation mechanism standard," Washington, D.C., NIST FIPS 203, Aug. 2024. doi: 10.6028/NIST.FIPS.203.
- [8] National Institute of Standards and Technology (US), "Module-lattice-based digital signature standard," Washington, D.C., NIST FIPS 204, Aug. 2024. doi: 10.6028/NIST.FIPS.204.
- [9] National Institute of Standards and Technology (US), "Stateless hash-based digital signature standard," Washington, D.C., NIST FIPS 205, Aug. 2024. doi: 10.6028/NIST.FIPS.205.
- [10] RISC-V International, "Ratified Specifications." Accessed: Apr. 20, 2026. [Online]. Available: <https://riscv.org/specifications/ratified/>
- [11] A. Dolmeta, V. Piscopo, G. Masera, M. Martina, and M. Hutter, "HORCRUX - A Lightweight PQC-RISC-V eXtension Architecture." Accessed: Apr. 20, 2026. [Online]. Available: <https://eprint.iacr.org/2025/1934>
- [12] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography." [Online]. Available: <https://eprint.iacr.org/2020/446>
- [13] YosysHQ, "PicoRV32 - A Size-Optimized RISC-V CPU." Accessed: Jun. 14, 2026. [Online]. Available: <https://github.com/YosysHQ/picorv32>
- [14] "lowRISC/ibex." Accessed: Jun. 23, 2026. [Online]. Available: <https://github.com/lowRISC/ibex>
- [15] stnolting, "stnolting/neorv32." Accessed: Jun. 18, 2026. [Online]. Available: <https://github.com/stnolting/neorv32>
- [16] N. P. Schuster and G. L. Nazar, "GRASP-based High-Level Synthesis Design Space Exploration for FPGAs," in *2023 XIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2023, pp. 1–6. doi: 10.1109/SBESC60926.2023.10324241.
- [17] "pq-code-package/mldsa-native." Accessed: Jul. 19, 2026. [Online]. Available: <https://github.com/pq-code-package/mldsa-native>
- [18] Isaac, "Ley de Amdahl, regla de Pollack, ley de Koomey... ¿qué son?." Accessed: Jul. 30, 2026. [Online]. Available: <https://architecnologia.es/ley-amdahl-regla-pollack-ley-koomey-que-son>
- [19] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl, "Post-Quantum Signatures on RISC-V with Hardware Acceleration," *ACM Trans. Embed. Comput. Syst.*, vol. 23, no. 2, Mar. 2024, doi: 10.1145/3579092.