Karu64: An Open RVA23 Processor with Vector Keccak Support

Markku-Juhani O. Saarinen
Tampere University
Tampere, Finland
markku-juhani.saarinen@tuni.fi

Thomas Szymkowiak
Tampere University
Tampere, Finland
thomas.szymkowiak@tuni.fi

***Abstract*—Can a whole-permutation Keccak instruction provide useful post-quantum acceleration inside a general-purpose vector application processor, despite its 1600-bit state and limited register-file bandwidth? We present, to our knowledge, the first hardware implementation of the standards-track RISC-V Zvknhk Vector Keccak extension. We integrate it alongside other vector cryptography support into Karu64, a new open-source application processor supporting the RVA23S64 profile. The complete system boots Debian Linux on a VCU118 FPGA at 75 MHz, allowing realistic OpenSSL benchmarking. Architecturally, the vkeccak.vi instruction gathers the state once, performs the permutation, and returns it through existing vector registers. This gives it excellent performance and allows usage from unmodified Linux userspace. In ideal-memory RTL simulation, resident-state SHAKE128/256 absorption and squeezing are 79–83× and 105–116× faster, respectively, than Clang-generated vector-enabled C with Zbb and Zvbb extensions. FPGA measurements under Linux show end-to-end ML-KEM and ML-DSA operations gain 1.48× and 1.65×, respectively, over hand-vectorized implementations, due to the central role of Keccak in these PQC standards. A NanGate45 synthesis comparison shows a 2.29 % core-area increment over the implemented ratified-crypto baseline. These results establish practical feasibility of the fixed-group instruction, connecting standards-track semantics to integrated hardware cost and measured application benefit in a working Linux system.**

***Index Terms*—RISC-V, Vector Cryptography, Keccak, SHA-3, Post-Quantum Cryptography, ML-KEM, ML-DSA**

I. Introduction

The transition to the ML-KEM [1] and ML-DSA [2] quantum-safe cryptographic standards profoundly affects the nature of cryptographic compute workloads. "Big integer modular arithmetic" is no longer dominant; the cost is now split between polynomial arithmetic and SHA-3/SHAKE, which supplies hashing, seed expansion, and sampling streams. Optimizing only the Number Theoretic Transform (NTT) and other polynomial operations in these Lattice-based algorithms therefore leaves a substantial symmetric-cryptography cost; roughly half of the cycles on many platforms.

SHA-3 and SHAKE are *sponge modes* built from the 1600-bit Keccak-p[1600,24] permutation [3]. Software vectorization significantly accelerates polynomial operations, but cross-lane movement makes Keccak relatively hard to vectorize [4]. In pure hardware, the Keccak permutation is very fast and comparatively simple to implement. The recognized architectural challenge is moving a 1600-bit state efficiently between the application and that hardware.

The ratified RVA23 application profiles [5] provide a long-term compatibility target; they reduce software porting and maintenance effort, and facilitate binary software distribution. For example, Ubuntu Linux has required RVA23S64 since release 25.10 [6], and RVA23 is also the baseline for RISC-V Android ABI [7]. Vector extensions V, Zvbb, and Zvkt are mandatory in RVA23, while the vector-cryptography suites Zvkng and Zvksg [8] are options. Our stock OpenSSL results (Section VI-C) demonstrate the benefit of ecosystem support already available for ratified vector cryptography extensions.

Researchers have previously designed custom RISC-V Keccak Instruction Set Extensions (ISEs), including round-step instructions [9], [10], a scalar extension with private permutation state [11], and tightly coupled coprocessors [12], [13]. While many of these are open source, they have not been submitted to RISC-V International for ratification consideration.

In September 2026, the RISC-V PQC Task Group released version 0.1 of its specification with the Zvknhk Vector Keccak extension [14], whose single instruction `vkeccak.vi` performs a full 24-round (or 12-round) Keccak-p[1600] permutation in place on a fixed 2048-bit vector register group. Many ratification-relevant artifacts exist: A Sail specification, Spike and QEMU reference models, and middleware integration (an OpenSSL backend). However, at the time of writing, it is still a standards-track draft, not a fully ratified extension.

We evaluate this emerging architectural contract, which departs from most previous proposals in its vector-integration approach. We take a whole-systems design approach and ask whether coarse-grained acceleration remains worthwhile after operand movement and processor integration. Linux execution on FPGA exposes these costs through real virtual-memory and cache/DRAM paths, making the evaluation more realistic.

This paper makes three main contributions:

- **A new RISC-V processor platform.** We describe Karu64, an open, size-optimized, RVA23S64 compatible vector processor that integrates the Zvknhk extension. We validate it against current references such as RISC-V ACT 4.1 (Architectural Certification Tests) (Sections III and IV).
- **Evaluating the cost and benefit of Zvknhk.** FPGA measurements under Linux show 1.48× ML-KEM and 1.65× ML-DSA gains beyond vector intrinsics. Controlled synthesis places incremental Keccak integration at 87.19 kGE, or 2.29 % over Zvk (Sections V-B and VI).

SHAKE blocks gain 79–116× over compiler-generated vector-enabled C in ideal-memory RTL simulation, and the resident permutation needs 20× fewer cycles than the best published RISC-V software [4].
- **A microarchitectural analysis.** We explain how full-permutation fusion avoids the repeated register traffic of round-step ISEs [9], [10], distinguishing engine latency, register transfers, and the software interface used in the measurements (Section VI-E).

II. Background and Related Work

A. RISC-V Vectors and Element Groups

RVV 1.0 adds 32 vector registers of VLEN bits ($128 \leq \mathsf{VLEN} \leq 65536$, an implementation constant); the vector length vl and the vtype CSR (element width SEW, group multiplier LMUL) determine how many elements an instruction processes. The Zvk extensions [8] introduced *element groups*: an AES instruction treats four 32-bit elements as one 128-bit block ($\mathsf{EGW} = 128$, $\mathsf{EGS} = 4$); SHA-512 and SM3 use $\mathsf{EGW} = 256$.

Zvkt specifies data-independent execution latency (DIEL) for designated vector instructions. Crypto instructions avoid the secret-indexed table accesses responsible for software AES cache attacks [15], [16] and are hence "constant time".

B. Keccak in Post-Quantum Cryptography

Keccak-p[1600,24] operates on a 5×5 array of 64-bit lanes $A[x, y]$. In software, the instruction mix of each of its 24 rounds consists of word shuffling, rotations, and logical XORs, ANDs, and NOT operations. Our RV64GC non-vector permutation needs 6601 instructions, or 4268 with Zbb's `RORI` and `ANDN` – roughly 275 or 178 instructions per round.

Software must issue rotations and move data between registers for permutations, but those are just static wiring in a hardware round datapath; parallel XOR and AND-NOT networks implement the remaining operations. The logic depth of a Keccak round is remarkably short, allowing a 1-cycle hardware round implementation at high clock frequencies, with full permutation taking 24 cycles.

ML-KEM and ML-DSA invoke the permutation dozens to hundreds of times per operation ($\bar{n}_f$ in Table IV); A fast Keccak instruction speeds up essentially all PQC operations.

C. Related Work

Zhang et al. [4] demonstrate the importance of optimized software baselines for Keccak and lattice cryptography on RISC-V. Arm's four SHA-3 helper instructions (`EOR3`, `RAX1`, `XAR`, `BCAX`) [17] fuse two or three primitive operations on 128-bit registers; around 30 instructions are needed per round.

RISC-V Keccak ISEs differ mainly in where the 1600-bit state lives and in how much of the permutation one instruction performs. Li et al. [9] add custom round-step vector instructions with a Keccak-oriented register layout on an Alveo U250 FPGA at 100 MHz; their best 64-bit configuration takes 75 cycles per round, and PQC integration is left as future work. Bolat et al. [10] execute one round per `shatr` instruction, prototyped on CVA6, holding the state in dedicated internal registers.

`vkeccak.vi vd, imm5`

31:26	25	24:20	19:15	14:12	11:7	6:0
101001	1	imm5	10010	010	vd	1110111
funct6	vm	rounds	selector	funct3	in–out	OP-VE

`imm5=0`: Keccak-p[1600,24] `imm5=1`: Keccak-p[1600,12]

Fig. 1. Encoding of `vkeccak.vi` in Zvknhk draft v0.1 [14]. Bit 31 is on the left; OP-VE is `0x77`. The `vd` field selects the in-place fixed register group, and `imm5` selects 24 rounds or the last 12 rounds. Other immediates and `vm=0` are reserved.

PQCUARK [11] is a scalar extension whose two-rounds-per-cycle engine keeps a private state behind the load/store unit and moves it through memory in 64-bit words; on the Sargantana RV64 core it gains 2.3× over optimized ML-KEM/ML-DSA software, with a 1028 kGE Keccak unit in GF22. Coprocessors attached through CV-X-IF [12], [13] mirror the state in a private register file and complete a permutation call in 385–553 cycles on 32-bit microcontrollers; related designs target code-based schemes [18], [19]. None of these proposals are on the RISC-V standards track – full upstream compiler, emulator, or distribution support is unlikely.

D. Application Processor and Linux Integration

Most cited proposals (except [9]) store the internal Keccak state in a manner which is not architecturally-visible; the state must be saved/restored using custom kernel code or confined to single-threaded, bare-metal execution. Zvkhnk stores the state within a fixed group of architectural vector registers, supporting the existing vector context switch, signal delivery and trap semantics used within Linux. This is what allows us to use it from unmodified Debian Linux userspace. Our distinction is hence the combination of the standards-track Zvknhk ISA contract, an RVA23-targeted vector application processor, Linux execution, and end-to-end gains after vectorizing lattice arithmetic.

III. The Zvknhk Vector Keccak Extension

The Keccak extension Zvknhk [14] depends on the RISC-V vector extension baseline with $\mathsf{VLEN} \geq 128$ and defines a single instruction, `vkeccak.vi vd, imm5`. Its design reflects the hardware/software asymmetry above.

Figure 1 shows the concrete instruction encoding: the round selector occupies the usual `vs2` field, and `10010` selects Keccak in the vector-crypto opcode space. For example, `vkeccak.vi v0, 0` permutes the state in the group starting at `v0`, using all 24 rounds.

1) Whole permutation, not rounds: Because a round is cheap and shallow in hardware, the instruction performs all 24 rounds (`imm5=0`: Keccak-f[1600] as used by SHA-3, SHAKE, ML-KEM, ML-DSA, and SLH-DSA) or the last 12 rounds with constants $RC[12..23]$ (`imm5=1`: Keccak-p[1600,12] as used by TurboSHAKE and KangarooTwelve [20]); other immediates are reserved. Vector register file traffic – the expensive part – is thus paid once per permutation, and the internal round rate is an implementation choice.

2) One fixed element group: The 1600-bit state does not fit the Zvk convention of a *vector of* element groups strip-mined by vl. Instead the instruction operates on a group of EGS $=$ 32 64-bit elements (EGW $= 2048$) at `vd`, spanning NREG $= \lceil 2048/\mathsf{VLEN} \rceil$ consecutive registers (16 at VLEN $= 128$, 8 at 256, 4 at 512, one at ≥ 2048). The group is independent of vl (including $\mathsf{vl} = 0$) and LMUL and exempt from the EGS $\leq$ VLMAX and EMUL ≤ 8 rules; software only selects SEW $=$ 64. Elements 0–24 hold $A[x, y]$ at index $x + 5y$; elements 25–31, the *state tail*, and all other register bits are preserved.

3) Reserved encodings, atomicity, DIEL: SEW $\neq 64$, a misaligned `vd`, `vm=0`, and a nonzero vstart raise an illegal-instruction exception; the group update must be atomic with respect to traps, and execution latency must be data-independent.

4) Architectural state and software integration: The instruction has no persistent architectural state outside the vector registers. State can be transferred by plain unit-stride `vle64`/`vse64`, and `vxor` can be used for block absorption.

IV. The Karu64 Core

Karu64 is a single-issue, in-order RV64GCV core written in portable Verilog-2001 and released under the BSD 3-Clause license[1]. Its main development priorities were RVA23 feature completeness, portability, and compact size – not raw performance. The main OS target is Linux; we use Debian for tests[2].

A. From Marian to Karu64

Our original intention was to implement the standards-track Keccak extension within an existing open-source core for system evaluation. However, we found that the available open-source candidates were not suitable for full system evaluation. For instance, CVA6/Ara [21] has gaps in RVV 1.0 compliance and provides no MMU support for vector operations. This, and the desire to run standard RVA23 Linux software on an FPGA, motivated us to develop the Karu64 vector core using only a small number of pre-existing hardware components:

Marian [22] integrated ratified Zvk cryptography into a CVA6/Ara [21] vector subsystem using operand collection, cryptographic execution, and writeback stages. Karu64 reuses its AES, SHA-2, SM3, and SM4 logic, with several adaptations. A new GHASH engine replaces Marian's combinational implementation. We adopt the Keccak round RTL from the Sloth SLH-DSA accelerator [23].

Controlled use of Large Language Models (see Acknowledgment section) to write RTL, together with the availability of the RISC-V Sail specification, ACT 4.1, and other open specification/certification infrastructure (Section IV-F) greatly accelerated development.

Architecturally, Marian demonstrated the complications of retrofitting the 128- or 256-bit element groups required by standard Vector Cryptography into an existing lane-oriented vector architecture. Our relatively simple (but feature-complete) vector unit was designed from the start to facilitate ratified Zvk extensions and the proposed 1600-bit Keccak vector groups.

[1]Karu64 RTL source, flows, HW tests: https://github.com/karucore/karu64

[2]Karu's Debian drivers, benchmarks: https://github.com/karucore/karudeb

B. Scalar Core and Privilege Architecture

A prefetcher with an optional 4 KiB instruction cache feeds the decoder and instruction data is subsequently registered for execution; a `busy` signal from the multi-cycle execution units prevents further instructions from being issued before the currently executing instruction is retired. The units include the integer ALU, Zba/Zbb/Zbs, configurable multiply/divide, IEEE 754 single/double FPUs with fused multiply-add, atomic load/store, and vector execution. The evaluated configuration implements the mandatory RVA23S64 feature set: the user baseline including full Zvbb, privileged-1.13 CSRs, Sv39/Svade, Svnapot, Svpbmt, Svinval, Sstc, Sscofpmf, and H/Sha with two-stage translation and state enables [5], [8], [24]. Furthermore, CLINT/PLIC supports OpenSBI and Linux; optional Smcntrpmf enables privilege-filtered fixed counters (for benchmarks). Scalar multiply/divide use 4/64 cycles and vector multiply/divide 16/64 cycles, with two vector pipeline stages.

C. Vector Unit

The vector register file (VRF) is a dual-port memory, accessed in 128-bit *granules* with exact byte enables, realizing RVV "undisturbed" policies without whole-register read-modify-write.

At VLEN $= 256$, each vector register contains four 64-bit elements; the measured configuration processes granules through two 64-bit SIMD lanes, both with floating-point support. A unified sequencer implements reductions, permutations (`vrgather`, `vcompress`, slides), and fixed-point arithmetic. The vector load/store unit supports all RVV addressing modes with precise page faults through the shared Sv39 MMU.

D. Zvk Crypto Unit

Because a 64-bit lane cannot present a 128- or 256-bit element group atomically, Zvk instructions execute outside the lanes: the sequencer gathers the operand groups from the VRF, pulses a request to one isolated crypto unit containing adapted AES, SHA-2, SM3, SM4, and GHASH datapaths behind a uniform request/busy/done handshake, and writes the result group back. AES is combinational with a registered output; SHA-2 compression and message instruction schedules take two and four cycles; SM4 iterates its four rounds one per cycle; GHASH uses a radix-2^8 iterative multiplier (16 cycles) in place of Marian's combinational 128-stage carryless multiplier; Zvkb is ordinary lane logic. All datapaths have data-independent latency, and the core advertises Zvkt.

E. The `vkeccak.vi` Datapath

The Keccak instruction reuses this pattern with a larger operand (Figure 2). At issue, reserved encodings and invalid configuration trap without side effects. Otherwise, a sub-FSM *loads* the eight registers into a 2048-bit buffer, *runs* the permutation on bits 0–1599 for 24 or 12 rounds, and *stores* the registers back, preserving tail bits 1600–2047. Reload uses both BRAM ports to fetch a whole 256-bit register per fill. The round datapath implements the FIPS 202 mappings with LFSR-generated ι constants; a 12-round run starts at $RC[12]$.

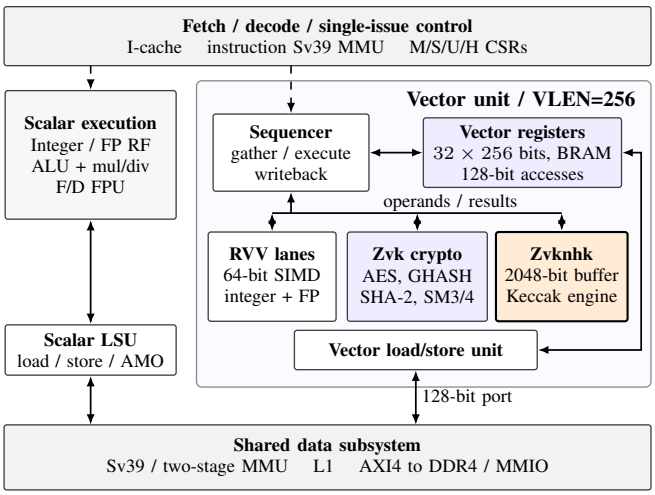


Fig. 2. Karu64 architectural overview. Dashed arrows denote issue; solid arrows denote operand/result or memory transfers. Zvk and Keccak are shared engines outside the RVV lanes. The highlighted Zvknhk path combines a sequencer-held 2048-bit buffer with one 1600-bit round engine. Instruction-memory, page-table-walk, and detailed control wiring are omitted.

One engine serves all lanes. No register is written before permutation completion, and no trap interrupts writeback, making the update atomic. The buffer is reloaded on each instruction invocation; there is no persistent Keccak shadow. For 24 rounds, the core engine has a 25-cycle request-to-completion latency; this excludes VRF movement and sequencing.

On FPGA, a permutation loop benchmark measures 88 cycles for a full 24-round permutation, including those costs, loop control, and amortized counter overhead (Section VI-A).

F. Functional Verification

We tested architectural compliance using ACT 4.1 (RISC-V Architectural Certification Tests) against Sail, supplemented by RTL assertions and directed microarchitectural tests. Directed tests cover two-stage translation, timers, counter filtering, vector context, access faults, and AXI backpressure. Zvk checks include multi-group AES/SM4 scalar-source broadcast, and 215 operand-class DIEL checks pass with the FPGA pipeline settings. Keccak tests cover 12 and 24 rounds, dependencies, fixed groups, tails, and illegal cases. Separate simulated SHAKE tests check independent `hashlib` known answers across multiple blocks, byte alignments 0–15, page boundaries, and injected stalls. We also ran AI-powered security, correctness, and standards compliance reviews of the source code.

On the measured FPGA image, Linux memory, vector ABI, crypto, and KVM API acceptance pass; the supplied stock-OpenSSL capture records 92/92 known-answer comparisons.

V. Implementation Results

A. FPGA System and Linux

The evaluation system is a Xilinx VCU118 (Virtex UltraScale+ XCVU9P) with 2 GiB DDR4, LiteEth gigabit Ethernet, UART, and a boot ROM. OpenSBI 1.8.1 and U-Boot 2025.01 boot Linux 7.2.4 with a Debian "trixie" NFS root. All reported FPGA benchmarks use the same RVA23S64 image.

TABLE I
YOSYS/NANGATE45 AREA-ONLY COMPARISON. Δ IS THE COMPLETE-CORE CELL-AREA CHANGE FROM RV64GCV+ZVBB. THE FINAL ROW INCLUDES THE PROFILE, I-CACHE, AND FPGA PIPELINE/COUNTER OPTIONS.

Configuration	Total (kGE)	Δ (kGE)	Keccak bucket (kGE)
RV64GCV+Zvbb	3667.90	–	–
+ Zvk	3806.98	+139.08	–
+ Zvk + Zvknhk	3894.17	+226.27	31.20
FPGA CPU configuration	4390.60	–	31.20

Vivado 2026.1 reports 366 098 CLB LUTs, 86 857 registers, 317.5 BRAM tiles, and 29 DSPs for the complete SoC. At 75 MHz, the CPU and whole-design timing closes cleanly. Linux programs exercise vector and crypto instructions through real virtual memory, caches, and DDR.

B. Area

Table I uses controlled Yosys 0.69+24/NanGate45 mapping, fast ABC, and disabled sharing; NAND2 area is 0.798 μm^2, and 200 MHz target. Here memories map into cells, without memory compilers or physical design.

The CPU configuration used for FPGA tests maps to 4.391 MGE, including profile and pipeline options; its difference from the feature baseline is not a Keccak-only cost. In the Yosys flow we observe that Zvknhk adds 87.19 kGE (2.29 %) over the Zvk feature baseline. The nested round-engine hierarchy is 31.20 kGE: the complete-core delta also captures buffering, control, and mapping changes.

VI. Performance Evaluation

The FPGA dataset measures resident SHAKE, ML-KEM/ML-DSA, and stock OpenSSL on one 75 MHz bitstream and Linux installation. The whole-process perf totals are not per-operation timings. ML-KEM rows average 100 operations and ML-DSA rows 10; mean speedups are arithmetic means of nine per-operation ratios per algorithm.

SHAKE reports the best repeated loop average; OpenSSL measures wall-clock throughput. Here "resident state" indicates an integration where the Keccak state stays in the vector registers between permutation iterations when absorbing or squeezing, rather than being fully transferred to/from memory for each permutation – as in "wrappers". (A Keccak instruction wrapper is a simple drop-in integration technique for software that uses Keccak APIs that pass permutation input/output in memory. This results in some memory-vector access overhead.)

A. Resident-State SHAKE Throughput

Both environments in Table II use VLEN $= 256$. State remains in `v0`–`v7`; absorption loads a rate block, XORs, and permutes, while squeezing stores a rate block and permutes. Setup/spill, padding, and checking are outside timing; pointer and counter overhead remain. A finite N-block squeeze needs $N - 1$ permutations; these steady-state loops retain the final permutation. The FPGA harness uses an assembly loop and warms input before measurement. Its 16- and 256-block results retain their different working sets and repetition counts, rather

TABLE II
CYCLES PER SHAKE RATE BLOCK, INCLUDING PERMUTATION. RATES 168/136 B CORRESPOND TO SHAKE128/256. UPPER ROWS: IDEAL-MEMORY VERILATOR, 16-BLOCK AVERAGES; CLANG 22.1.8 `-O3` C. LOWER ROWS: FPGA LOOP AVERAGES, BEST OF 21 RUNS FOR 16 BLOCKS AND 7 RUNS FOR 256 BLOCKS. GAINS COMPARE SIMULATION ROWS ONLY.

	Absorb		Squeeze	
Implementation	168 B	136 B	168 B	136 B
RV64GC C	25390	25414	24653	24970
+ Zbb	16572	16163	16059	16348
+ Zbb + V + Zvbb	17497	16385	16459	16804
RTL resident `vkeccak.vi`	221.75	197.75	156.88	144.56
RTL gain vs. vector C	78.9×	82.9×	104.9×	116.2×
FPGA, 16 blocks	242.50	224.12	286.31	257.56
FPGA, 256 blocks	306.64	274.94	292.44	260.15

TABLE III
WALL-CLOCK THROUGHPUT OF THE UNMODIFIED DEBIAN OPENSSL 3.5.7 `SPEED` BENCHMARK (16 384-BYTE BLOCKS, 10 S RUNS) ON THE 75 MHZ BOARD. OPENSSL CAPABILITY STRING IS USED TO SELECT EXTENSIONS.

Algorithm	Extensions used	Scalar (kB/s)	Zvk (kB/s)	Speedup
AES-128-ECB	Zvkned	196.2	7,523.8	38.3×
AES-128-CTR	Zvkb+Zvkned	174.8	6,267.7	35.9×
AES-128-GCM	Zvkb+Zvkg+Zvkned	127.4	4,655.8	36.5×
AES-128-XTS	Zvbb+Zvkg+Zvkned	167.8	5,610.3	33.4×
GHASH	Zvkb+Zvkg	382.2	16,551.0	43.3×
SHA-256	Zvkb+Zvknhb	213.8	3,001.8	14.0×
SHA-512	Zvkb+Zvknhb	368.4	2,815.2	7.6×
SM3	Zvkb+Zvksh	194.0	1,712.0	8.8×
SM4-ECB	Zvkb+Zvksed	219.6	2,202.6	10.0×
ChaCha20	V+Zbb+Zvkb	395.6	1,515.0	3.8×

than being pooled. The resident permutation costs 87.62/87.98 cycles and a load–permute–store loop costs 394.75/395.70 cycles for 16/256 blocks. These include loop overhead and differ from the instrumented PQC wrapper below.

The ideal simulation benchmark has data L1 enabled and instruction caching disabled. Data cache, VRF, and AXI protocol costs remain; vector pipelines match the FPGA geometry, while scalar multiply/divide retain one-cycle simulation defaults. Hardware unrolls 16 steps. The C baselines inline the portable permutation into a rolled 16-block loop, permitting register residency. The 79–83× absorption and 105–116× squeezing gains apply to compiler-generated vector-enabled C.

In the controlled simulation comparison, transfer optimizations reduce absorption cycles by 39.5/44.9 % and squeezing by 46.3/47.2 % (168/136 B). The resident permutation drops from 135.06 to 79.06 cycles through operand-movement changes around an unchanged round engine. The FPGA rows report their integrated behavior with the updated profile and real memory system.

The fastest published RISC-V software permutation we are aware of, Zhang et al.'s hand-scheduled RV64IB assembly on the dual-issue XuanTie C908 [4], takes 1770 cycles and 3405 instructions; their RVV variant is slower than their scalar code. In cycle count, the 88-cycle resident permutation is 20× below that best published RISC-V software permutation and 55× below their 4827-cycle C908 RVV implementation, at very different clock rates.

B. Wrapper Performance on Karu64

The PQC wrapper interface loads and stores 25 state words at $\mathsf{LMUL} = 8$, with `vkeccak.vi`, vector configuration, a VLEN check, and a call counter. The ML-KEM microbenchmark averages 500 cycles and 18 retired instructions for this wrapper, versus 24 316 cycles for software Keccak in the intrinsics/Zbb build (48.6×). This instrumented memory-to-memory interface is distinct from the resident-state loop and the simpler 395.70-cycle load–permute–store loop. Its transfer latencies are part of the end-to-end benchmarks.

The large cycle-count gains reflect both permutation fusion and the relatively low throughput of general-purpose instructions on our core. Our Zbb build retires 4268 instructions per permutation, matching the 4296 that Zhang et al. [4] report for compiled C on RV64IB and about 25 % more than their best hand-written RV64IB assembly (at CPI 0.52 on the dual-issue XuanTie C908), whereas the RV64GC rows of Table II run at roughly four cycles per instruction in our much simpler processor.

C. Symmetric Cryptography with Stock OpenSSL

Table III selects scalar and accelerated paths through runtime capability overrides in a Debian stock OpenSSL binary. AES-GCM gains 36.54×, GHASH 43.30×, SHA-256 14.04×, and SHA-512 7.64× over package scalar paths. The tested Zvbb+Zvkg+Zvkned path also raises AES-XTS to 33.44×. This demonstrates unmodified distribution-library acceleration; draft Zvknhk support in a separate patched OpenSSL backend is not part of these results.

D. Post-Quantum Cryptography with `vkeccak.vi`

Table IV evaluates ML-KEM/ML-DSA derived from the reference Kyber/Dilithium C [25], [26], compiled with Clang 23 (git `c07f4eef`), `-O3`, and the specification instruction encoding. Build switches select scalar or hardware Keccak and RVV intrinsics for `vrgather`-based NTT butterflies and `vcompress`-based rejection sampling. The implementations have NIST ACVP known-answer validation. Software-Keccak vector builds target GCV+Zbb; hardware-Keccak builds target GCV with the wrapper, not an otherwise identical flag toggle. For the ML-KEM 256-point NTT, autovectorization is slower than scalar GC (136 485 versus 121 508 cycles), while our intrinsics require 103 979 cycles (1.17× over GC). ML-DSA shows 164 367, 119 219, and 96 379 cycles, respectively (1.24× for intrinsics over GC). These results reflect this granule-sequenced implementation.

Table IV shows mean speedups of 1.12×/1.15× from Zbb alone for ML-KEM/ML-DSA. Default autovectorization slows both relative to scalar Zbb in our core; intrinsics improve on autovectorized software by 1.28×/1.15×. Using `vkeccak.vi` yields 1.36×/1.53× on autovectorized code and 1.48×/1.65× on intrinsics code. The latter ranges from 1.27× for ML-DSA-65 signing to 1.97× for ML-DSA-87 key generation.

TABLE IV

ML-KEM/ML-DSA FPGA MEASUREMENTS UNDER LINUX: THOUSANDS OF USER CYCLES (MEANS OF 100 ML-KEM OR 10 ML-DSA OPERATIONS; $\bar{n}_f$ = MEAN KECCAK-p[1600,24] CALLS). BUILDS: RV64GC; +ZBB; LLVM-AUTOVECTORIZED RV64GCV ("V AUTO"); WITH `VKECCAK.VI` ("+K"); HAND-WRITTEN VECTOR INTRINSICS FOR NTT AND SAMPLING ("V INTR."); AND INTRINSICS WITH `VKECCAK.VI`. SOFTWARE-KECCAK VECTOR BUILDS TARGET GCV+ZBB; +K BUILDS TARGET GCV WITH THE WRAPPER. THE FINAL COLUMN IS THE INCREMENTAL SPEEDUP FROM V INTR. TO V INTR.+K.

Parameter	Op.	$\bar{n}_f$	RV64GC	+Zbb	V auto	V auto+K	V intr.	V intr.+K	K gain
ML-KEM-512	KeyGen	27	2275.9	1974.7	2073.8	1386.4	1715.9	1059.2	**1.62**×
ML-KEM-512	Encaps	26	2645.5	2370.5	2579.2	1933.4	2034.7	1403.8	**1.45**×
ML-KEM-512	Decaps	26	3373.6	3102.2	3379.2	2726.8	2635.4	2014.5	**1.31**×
ML-KEM-768	KeyGen	43	3780.7	3315.1	3554.7	2483.0	2825.9	1773.4	**1.59**×
ML-KEM-768	Encaps	44	4399.0	3913.5	4247.8	3143.2	3263.3	2198.8	**1.48**×
ML-KEM-768	Decaps	44	5366.1	4897.5	5336.5	4218.6	4059.5	3015.5	**1.35**×
ML-KEM-1024	KeyGen	69	5971.8	5227.6	5575.0	3850.7	4368.6	2680.2	**1.63**×
ML-KEM-1024	Encaps	70	6580.9	5811.9	6399.1	4654.4	4867.2	3182.7	**1.53**×
ML-KEM-1024	Decaps	70	7813.0	7051.4	7777.7	6008.5	5892.1	4216.5	**1.40**×
ML-KEM mean speedup vs. GC			**1.00**×	**1.12**×	**1.04**×	**1.41**×	**1.33**×	**1.97**×	**1.48**×
ML-DSA-44	KeyGen	103	7226.7	6071.1	6353.0	3818.6	5781.1	3158.1	**1.83**×
ML-DSA-44	Sign	158	19914.0	18135.9	20706.7	16831.0	16662.4	12860.2	**1.30**×
ML-DSA-44	Verify	99	7746.9	6729.6	7287.3	4750.4	6251.2	3795.7	**1.65**×
ML-DSA-65	KeyGen	188	12299.1	10381.1	10690.5	6110.7	9786.3	5095.8	**1.92**×
ML-DSA-65	Sign	287	39532.1	36336.0	40928.9	33957.2	33331.1	26321.2	**1.27**×
ML-DSA-65	Verify	174	12752.4	10893.5	11606.4	7254.9	10199.6	5870.1	**1.74**×
ML-DSA-87	KeyGen	324	20992.7	17378.3	17797.8	9742.9	16494.5	8392.1	**1.97**×
ML-DSA-87	Sign	408	46978.9	42377.0	47166.1	37147.3	39290.0	29327.1	**1.34**×
ML-DSA-87	Verify	311	21428.9	18229.3	19021.5	11348.7	17170.1	9334.2	**1.84**×
ML-DSA mean speedup vs. GC			**1.00**×	**1.15**×	**1.08**×	**1.66**×	**1.23**×	**2.04**×	**1.65**×

Over scalar GC, the combined mean gains are $1.97\times$/$2.04\times$, reaching $2.50\times$ for ML-DSA-87 key generation. The shorter ML-DSA sample and rejection-dependent signing workload limits conclusions about its timing distribution.

For ML-KEM-768 key generation, approximately 43 permutations at 24.3 k cycles account for about 37 % of the 2.826 M-cycle intrinsics baseline. An Amdahl estimate with this share and $48.6\times$ permutation acceleration predicts $1.57\times$, close to the measured $1.59\times$.

E. Performance Comparison with Prior Keccak ISEs

Reported single-state permutation costs are 1800 cycles for 24 rounds using round-step instructions of Li et al. [9] (75 cycles per round at 64-bit LMUL $= 8$), 385 and 553 cycles per permutation call for the CV-X-IF coprocessors [12], [13], and 12 cycles for the PQCUARK engine [11] before its 64-bit-word state traffic through memory, versus our 87.98-cycle FPGA resident-permutation loop including VRF transfers and loop overhead. These are cross-design cycle counts at different clocks, word widths, and measurement boundaries, not matched throughput speedups, and batched multi-state throughput [9] is only partially relevant to PQC usage. The controlled same-core comparison is the simulation rows of Table II.

VII. LESSONS AND FUTURE WORK

Operand movement matters: The fixed-group definition lets Karu64 gather the 1600-bit state through existing granule paths, without a shuffle network. Full-permutation fusion pays VRF traffic cost once rather than for every round-step instruction [9], and architectural registers avoid the per-word state transfers and operating system task-switching complexity of private-state designs [11], [13].

Prototype notes and limitations: The current prototype supplies only preliminary feasibility evidence for Zvknhk ratification. ASIC tape-out is planned future work. We don't claim that our processor has comparable performance or verification maturity to commercial and high-end application processors. These will need further architectural considerations for Zvknhk.

VIII. CONCLUSION

The standards-track Zvknhk `vkeccak.vi` instruction is practical in an open RVA23S64 vector application processor. On the 75 MHz Linux/FPGA system, a resident permutation takes approximately 88 cycles, and the instruction adds $1.48\times$/$1.65\times$ ML-KEM/ML-DSA gains beyond general vector intrinsics optimizations. For SHA-3 and SHAKE throughput, ideal-memory comparisons show 79–$83\times$ absorption and 105–$116\times$ squeezing gains over vector-enabled C code. Controlled synthesis places Keccak integration at 87.19 kGE, or 2.29 % over Zvk. Full-permutation fusion removes repeated round-step register traffic, while vector-register residency avoids per-block state spills. The instruction semantics, scoped hardware cost, and working Linux applications establish a concrete feasibility case for the standards-track Keccak instruction proposal.

ACKNOWLEDGMENT

We thank the SoC Hub team at Tampere University and the RISC-V International Post-Quantum Cryptography Task Group for their support and discussions. Endrit Isufi provided original Marian Zvk testing and benchmarks. Our work was supported by the Horizon Europe EQUIP project. Anthropic (Fable 5.1, Opus 5) and OpenAI (GPT-6 Astra, GPT 5.6-Sol) AI models were used under the close supervision of the authors for experiment design, drafting, and table/figure preparation.

REFERENCES

[1] NIST, "Module-lattice-based key-encapsulation mechanism standard," NIST, Federal Information Processing Standards Publication FIPS 203, August 2024.

[2] ——, "Module-lattice-based digital signature standard," NIST, Federal Information Processing Standards Publication FIPS 204, August 2024.

[3] ——, "SHA-3 standard: Permutation-based hash and extendable-output functions," NIST, Federal Information Processing Standards Publication FIPS 202, August 2015.

[4] J. Zhang, Y. Yan, J. Huang, and Ç. K. Koç, "Optimized software implementation of keccak, kyber, and dilithium on rv{32,64}im{B}{V}," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 1, pp. 632–655, 2025. [Online]. Available: https://eprint.iacr.org/2024/1515

[5] RISC-V International, "RVA23 profiles," RISC-V International, Tech. Rep., October 2024, version 1.0, ratified October 17, 2024. [Online]. Available: https://docs.riscv.org/reference/rva23/_attachments/rva23-profile.pdf

[6] Canonical, "Canonical and Ubuntu RISC-V: a 2025 retro and looking forward to 2026," Press release, February 2026. [Online]. Available: https://canonical.com/blog/canonical-and-ubuntu-risc-v-a-2025-retro-and-looking-forward-to-2026

[7] RISC-V International, "RISC-V announces ratification of the RVA23 profile standard," Press release, October 2024. [Online]. Available: https://riscv.org/blog/risc-v-announces-ratification-of-the-rva23-profile-standard/

[8] RISC-V, "The RISC-V instruction set manual volume I: Unprivileged architecture," RISC-V International, Official Release 20260120, January 2026. [Online]. Available: https://docs.riscv.org/reference/isa/_attachments/riscv-unprivileged.pdf

[9] H. Li, N. Mentens, and S. Picek, "Maximizing the potential of custom RISC-V vector extensions for speeding up SHA-3 hash functions," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*. IEEE, 2023, pp. 1–6.

[10] A. Bolat, S. Sezer, K. McLaughlin, and H. Hui, "Microarchitecture design and benchmarking of custom SHA-3 instruction for RISC-V," in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2025, Kalamata, Greece, July 6-9, 2025*. IEEE, 2025, pp. 1–6. [Online]. Available: https://doi.org/10.48550/arXiv.2508.20653

[11] X. Carril, A. M. Pasoot, E. Parisi, O. Farràs, C. A. Lara-Nino, and M. Moretó, "PQCUARK: A scalar RISC-V ISA extension for ML-KEM and ML-DSA," in *Design, Automation & Test in Europe Conference, DATE 2026, Verona, Italy, April 20-22, 2026*. IEEE, 2026, pp. 1–7. [Online]. Available: https://eprint.iacr.org/2025/2178

[12] A. Dolmeta, V. Piscopo, M. Mirigaldi, M. Martina, and G. Masera, "RISC-V based Keccak co-processor for NIST post-quantum cryptography standards," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2025, pp. 1–5.

[13] A. Dolmeta, V. Piscopo, M. Hutter, M. Martina, and G. Masera, "HORCRUX: A complete PQC RISC-V extension architecture," *CoRR*, vol. abs/2607.13939, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2607.13939

[14] RISC-V International, "RISC-V post-quantum cryptography extension," RISC-V Post-Quantum Cryptography Task Group, Tech. Rep., September 2026, development draft v0.1, September 11, 2026. [Online]. Available: https://github.com/riscv/riscv-pqc/releases/download/v0.1/riscv-pqc-v0.1-20260911.pdf

[15] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, ser. Lecture Notes in Computer Science, L. Goubin and M. Matsui, Eds., vol. 4249. Springer, 2006, pp. 201–215.

[16] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, ser. Lecture Notes in Computer Science, D. Pointcheval, Ed., vol. 3860. Springer, 2006, pp. 1–20.

[17] ARM, "Arm A-profile A64 instruction set architecture," ARM, Guide DDI 0602 (ID092424), June 2026. [Online]. Available: https://support.arm.com/documentation/ddi0602/2026-06

[18] A. Dolmeta, S. Di Matteo, E. Valea, M. Carmona, A. Loiseau, M. Martina, and G. Masera, "TYRCA: A RISC-V tightly-coupled accelerator for code-based cryptography," in *Design, Automation and Test in Europe Conference (DATE)*, 2025, pp. 1–7.

[19] A. Dolmeta, V. Piscopo, M. Martina, and G. Masera, "CIRCE: CROSS integrated RISC-V cryptographic extension," in *Design, Automation and Test in Europe Conference (DATE)*, 2026, pp. 1–3.

[20] B. Viguier, D. Wong, G. Van Assche, Q. Dang, and J. Daemen, "KangarooTwelve and TurboSHAKE," IETF, RFC 9861, October 2025.

[21] M. Perotti, M. A. Cavalcante, R. Andri, L. Cavigelli, and L. Benini, "Ara2: Exploring single- and multi-core vector processing with an efficient RVV 1.0 compliant open-source processor," *IEEE Trans. Computers*, vol. 73, no. 7, pp. 1822–1836, 2024.

[22] T. Szymkowiak, E. Isufi, and M.-J. Saarinen, "Poster: Marian: An open source RISC-V processor with Zvk vector cryptography extensions," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, October 14–18, 2024, Salt Lake City, UT, USA*, 2024.

[23] M.-J. O. Saarinen, "Accelerating SLH-DSA by two orders of magnitude with a single hash unit," in *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part I*, ser. Lecture Notes in Computer Science, L. Reyzin and D. Stebila, Eds., vol. 14920. Springer, 2024, pp. 276–304.

[24] RISC-V, "The RISC-V instruction set manual volume II: Privileged architecture," RISC-V International, Official Release 20260120, January 2026. [Online]. Available: https://docs.riscv.org/reference/isa/_attachments/riscv-privileged.pdf

[25] Kyber Team, "Kyber – official reference implementation," 2024, matches FIPS 203. Author used commit 10b478f. [Online]. Available: https://github.com/pq-crystals/kyber

[26] Dilithium Team, "Dilithium – official reference implementation," 2024, matches FIPS 204. Author used commit cbcd875. [Online]. Available: https://github.com/pq-crystals/dilithium/