Hardware Acceleration for Post-Quantum Cryptography in Resource Constrained Embedded Systems with RISC-V ISEs

Carlos Gewehr[1] **and Fernando Moraes**[1]

[1]Pontificia Universidade Catolica do Rio Grande do Sul (PUC-RS)

carlos.gewehr@edu.pucrs.br, fernando.moraes@pucrs.br

Abstract. *The imminent rise of practical quantum computing threatens well-established cryptography algorithms for secret key exchange in use today, such as Diffie-Hellman and Elliptic Curve (ECC) based schemes. These algorithms are currently being replaced by quantum-safe Crystals-Kyber, also known as ML-KEM. This work aims to explore hardware acceleration through RISC-V Instruction Set Extensions (ISEs) in a low-end 32-bit core in a comprehensive evaluation comprising performance, energy consumption, memory footprint and die area costs, enabling an efficient implementation of a cryptosystem that can withstand attacks from the emergence of quantum computers and is compliant to modern cryptographic standards and algorithm suites. Three different parametrizations of Kyber symmetric primitives are evaluated: the well-known SHA-3 and AES/SHA-2 based versions, as well as a novel parametrization using Ascon. This work also explores ISE-enhanced implementations of algorithms for authenticated encryption (AEAD) and hash functions at the 128 and 256 bit security levels, evaluating improvements due to the use of specialized instructions in each algorithm.*

1. Introduction

The advancement of quantum computing has raised concerns about the long-term security of current key exchange mechanisms, such as Diffie-Hellman and Elliptic Curve Diffie-Hellman (ECDH), widely used in current cryptographic systems. These traditional methods are vulnerable to quantum-based attacks, which could compromise the confidentiality and integrity of sensitive information in the future. Consequently, researching and deploying new key exchange algorithms based on mathematical foundations that are naturally resistant to such attacks is crucial, introducing a new field of research called Post-Quantum Cryptography (PQC). Lattice-based problems such as Learning with Errors (LWE) and its variants, particularly Module Learning with Errors (MLWE), emerged as potential solutions, showing resistance to classical and quantum-based attacks. Several algorithms have been proposed using the MLWE problem as the basis for its security guarantees.

To address this need, the National Institute of Standards and Technology (NIST) initiated a competition to standardize new quantum-resistant key exchange algorithms. After several submissions for Key Encapsulation Mechanisms (KEM), the winning algorithm was announced in July 2022 to be the MLWE-based scheme Crystals-Kyber. As a result of this competition, Kyber is now standardized under the name ML-KEM [NIST 2024] since August 2024.

As with most lattice-based schemes, Kyber introduces performance and memory usage overheads. These overheads become even more pertinent in embedded systems and Internet of Things (IoT) devices, which feature low-complexity processors and limited on-chip memory. It should be noted that such devices typically serve as the first point of data collection from sensor interfaces, which are then forwarded to upstream complex nodes. Upstream nodes possess greater computational power and are better equipped to secure data effectively. However, the same cannot be said for downstream, less complex nodes. The efficient provision of post-quantum secure communication in low-complexity network endpoints is paramount for the further advancement of the IoT.

Parallel to the post-quantum competition, NIST has also promoted the lightweight cryptography (LWC) competition, aiming to standardize algorithms tailored to devices with constrained resources. The LWC competition accepted the submission of algorithms that provide symmetric encryption and optional hashing capabilities. Such algorithms should provide better performance and memory footprint than traditionally used algorithms at the 128-bit security level. In February 2023, Ascon [Dobraunig et al. 2021] was selected as the winner of the LWC competition. The intersection between Lightweight Cryptography and Post-Quantum Cryptography is still an open research problem.

Hardware acceleration can significantly enhance performance, memory, and energy efficiency in crucial operations of an algorithm. One way to achieve hardware acceleration is by implementing specialized instructions in a general-purpose processor, a practice known as Instruction Set Extensions (ISEs). Compared to memory-mapped accelerators, extending a base instruction set with specialized instructions for a given algorithm provides benefits relevant to low-power embedded systems, including: (i) resource sharing among generic and specialized components, such as the register file and RAM interfaces; (ii) trivial data transfer among generic and specialized components via the register file, avoiding costly memory accesses; (iii) no added system complexity to, e.g. the bus and interrupt controllers.

The primary motivation for this work is the scarcity of research in the literature concerning Kyber implementations and hardware acceleration via ISEs in low-resource embedded systems, despite the clear need for post-quantum security solutions tailored to the IoT context. Most previous efforts in Kyber implementations focus on software-only implementations that do not explore hardware acceleration or highly complex, performance-driven implementations that do not prioritize crucial metrics for IoT, such as energy consumption and memory footprint. Given the increasing deployment of IoT devices, addressing this issue is becoming more critical [García-Morchón et al. 2019].

Since the NIST PQC competition, recommended algorithm sets encompassing all cryptographic services usually needed for practical security are being updated to include quantum-safe algorithms. Of such algorithm sets, the most relevant is the Commercial National Security Algorithm Suite (CNSA) 2.0, maintained by the National Security Agency (NSA), listing publicly-known algorithms recommended for use in US government systems [NSA 2023]. Even though the 128-bit security level is widely regarded as sufficient for practical security, CNSA 2.0 demands the use of algorithms at the 256-bit security level for symmetric cryptography and hash functions. The challenges in providing post-quantum security are further aggravated by the need to support 256-bit level primitives if CNSA 2.0 compliance is to be pursued.

2. Related Works

In this section, previous relevant works are presented and discussed in the context of this work. For brevity, only Kyber-specific hardware acceleration is presented in this document. For related works concerning hardware acceleration of symmetric primitives used within Kyber and as standalone Hash and AEAD functions, please refer to the full dissertation text.

[Fritzmann et al. 2019] propose two accelerators for hash (Keccak) and NTT operations coupled to a main processor via an AHB bus. Subsequent enhancements are discussed in [Fritzmann et al. 2020], where the Authors integrate the aforementioned co-processors inside a general-purpose CV32E40P processor. Their approach requires the presence of a floating point register file for the integration of the Keccak core, leading to significant area overheads in a higher complexity processor than the Ibex processor. Furthermore, the vectorized polynomial arithmetic component is implemented independently of the existing integer arithmetic ALU in the base processor, not exploring resource sharing.

[Nannipieri et al. 2021] show a superscalar 6-stage 64-bit CVA6 RISC-V processor extended with custom instructions for the Kyber and Dilhitium algorithms. These instructions are implemented based on a separate functional block, named PQ-ALU. The PQ-ALU implements the usual modular arithmetic operations, as well as a one-cycle CT butterfly. This approach is not usable in a low-resource context due to a lack of resource sharing, both within the PQ-ALU and among existing arithmetic elements in the base processor. There are 5 16-bit multipliers in a combinational path within the PQ-ALU, which, surprisingly, are reported to not be in the critical path of the extended processor. This approach is reasonable considering the context of the CVA6 64-bit superscalar processor, but not for the low-complexity Ibex core.

[Lee et al. 2022] present a co-processor based approach, with the usual operations needed for Kyber and other MLWE-based algorithms, such as Keccak, sampling, and polynomial multiplication, as well as the standardized RISC-V scalar cryptography extensions, accelerating AES and SHA-2. No quantitative results are presented, but we can conclude that this approach is not optimal, considering that the arithmetic modules are separated from the processor integer ALU, and intermediary results are committed to a separate register file in the co-processor. Again, resource sharing is not explored to the extent necessary for IoT applications.

[Alkim et al. 2020], hardware-software co-design is considered in the context of a 32-bit 5-stage RISC-V processor. New instructions for optimizing the NTT computation are implemented such that twiddle factors can be generated locally instead of being fetched from a pre-computed table stored in memory, as usually done in software implementations, reducing code size and the number of memory accesses. In the base processor (VexRiscv), regular integer multiplication computations are distributed between pipeline stages of the processor, and the same structure is followed for modular multiplication and reductions. The Authors implement the modular multiplication and reduction completely independently of the integer multiplication units in the base processor, such that the processor can be instanced without the integer multiplier while the optimizations proposed by the authors still apply. This is not an optimal use of resource sharing between regular integer arithmetic elements and modular arithmetic elements.

[Karabulut and Aysu 2020] present a different approach to optimizing the NTT. Instead of implementing new instructions as in previous works, the Authors propose modifications to the base processor control signals, such that memory dependence prediction and out-of-order execution are accomplished when purely software butterflies are detected during code execution. Again, this proposal is made in the context of a complex processor and does not apply to resource-constrained devices.

[Miteloudi et al. 2023] present a custom extension for polynomial arithmetic, supporting the Kyber and Dilithium algorithms. The supported operations are modular addition, subtraction, and multiplication, as well as CT and GS butterflies, implemented in a separate ALU decoupled from the regular integer arithmetic elements. As in previous proposals, the Authors only consider the acceleration of modular arithmetic. The single-cycle butterfly computations consider a register file with two write ports, which is unusual and not expected to be present in most low-complexity RISC-V implementations.

In summary, none of the works reviewed adequately addresses the issue of PQC and the Kyber algorithm in resource-constrained environments considering ISEs. Previously proposed solutions are either naturally high complexity or integrated inside complex processors. An implementation that explores extensive resource sharing with existing integer arithmetic elements in a naturally low-complexity processor is still missing in the literature.

Previous works often overlook the evaluation of energy consumption and memory footprint for their proposed solutions. Our work addresses this gap by presenting comprehensive analyses of both factors, offering a deeper evaluation of the implications associated with hardware acceleration through ISEs. Additionally, reported area overhead values are based on open-source synthesis tools or FPGA implementations, which may not accurately reflect the metrics in practical ASIC implementations. In contrast, this work employs a commercial synthesis tool with a manufacturable cell library, ensuring high-quality results.

Table 1 summarizes characteristics of the reviewed works concerning their hardware acceleration of symmetric primitives in Kyber, polynomial arithmetic, lightweight SIMD, CBD sampling, and coefficient compression features.

Table 1. Related works comparison

Work	Symmetric Primitives	Modular Arithmetic	SIMD	CBD Sampling	Compression
[Albrecht et al. 2018]	X	X			
[Park et al. 2022]		X			
[Banerjee et al. 2019]	X	X			
[Xin et al. 2020]		X		X	
[Fritzmann et al. 2020]	X	X		X	
[Nannipieri et al. 2021]		X			
[Lee et al. 2022]		X		X	
[Alkim et al. 2020]		X			
[Karabulut and Aysu 2020]		X			
[Miteloudi et al. 2023]		X			
This work	X	X	X	X	X

3. Hardware Acceleration of Crystals-Kyber

Kyber performance relies on symmetric primitives for its internal sampling and hashing operations. The evaluation of these symmetric primitives is presented only in the context of Kyber, please refer to the full dissertation text for their evaluation in their standalone use as hash functions and AEAD algorithms, as well as their hardware implementation within the Ibex processor. The choice of Kyber symmetric primitives for each parametrization and the RISC-V extension used to accelerate them is shown in Table 2.

Table 2. Symmetric primitives for each Kyber parametrization.

Kyber Parameterization	Extension	XOF	H	G	PRF	KDF
Kyber-Keccak	Zbkb	SHAKE128	SHA3-256	SHA3-512	SHAKE256	SHAKE256
Kyber-90s	Zkne/Zknh	AES-256 CTR	SHA-256	SHA-512	AES-256 CTR	SHA-256
Kyber-Ascon	Xascon	Ascon-XOF	Ascon-XOF	Ascon-XOF	Ascon-XOF	Ascon-XOF

In addition to symmetric primitive acceleration, the *XKyber* extension is proposed, containing 6 new instructions designed to optimize Kyber implementations in low cost RISC-V 32 bit processors. Four of these instructions rely on the fact that Kyber polynomial coefficients are 12 bit integers, stored in memory as arrays of 16 bit variables, meaning that 2 coefficients can fit in the native 32 bit word size. Processing can be performed in parallel for 2 coefficients at a time using regular integer arithmetic elements in the base processor, without the need for costly vector co-processors or extensive additional logic. Single Instruction Multiple Data (SIMD) processing also reduces the amount of memory accesses, seeing as 2 coefficients are loaded/stored from/to memory in a single word-wide access. The instructions proposed in *XKyber* are presented in Table 3.

Table 3. XKyber instructions

Instruction	Functionality
kybercbd2 RD, RS1	$RD[11{:}0] \leftarrow CBD_2(RS1[3{:}0]); RD[27{:}16] \leftarrow CBD_2(RS1[7{:}4]);$ (Algorithm 5.1)
kybercbd3 RD, RS1	$RD[11{:}0] \leftarrow CBD_3(RS1[5{:}0]); RD[27{:}16] \leftarrow CBD_3(RS1[11{:}6]);$ (Algorithm 5.1)
kyberadd RD, RS1, RS2	$RD[11{:}0] \leftarrow RS1[11{:}0] + RS2[11{:}0] \ mod\ q; RD[27{:}16] \leftarrow RS1[27{:}16] + RS2[27{:}16] \ mod\ q;$
kybersub RD, RS1, RS2	$RD[11{:}0] \leftarrow RS1[11{:}0] - RS2[11{:}0] \ mod\ q; RD[27{:}16] \leftarrow RS1[27{:}16] - RS2[27{:}16] \ mod\ q;$
kybermul RD, RS1, RS2	$RD \leftarrow RS1[11{:}0] \times RS2[11{:}0] \ mod\ q;$
kybercompress RD, RS1, RS2	$RD \leftarrow Compress_{RS2}(RS1[11{:}0]);$ (Algorithm 5.2)

The *XKyber* implementation and integration into Ibex is illustrated in Figure 1, along with other previously discussed ISEs, shown with dotted lines. Existing elements are shown in green, added elements in blue and control signals in orange. ISEs *Zknh* (SHA-2 sigma functions), *Xascon* (Ascon sigma functions), *Zkne* (AES) are implemented as functional units, while *Zbkb* (bit-manipulation) is implemented in the ALU.

As Figure 1 shows, we employ fine-grained, tightly coupled acceleration. Fine-grained acceleration executes only the most critical parts of the algorithms in dedicated hardware. Tightly coupled accelerators are closely integrated into the processor, sharing resources with its main architectural blocks. This contrasts with the co-processor acceleration approach, which implements the acceleration in a dedicated hardware block. This approach reduces memory accesses and processing power, particularly advantageous for resource-constrained embedded environments.

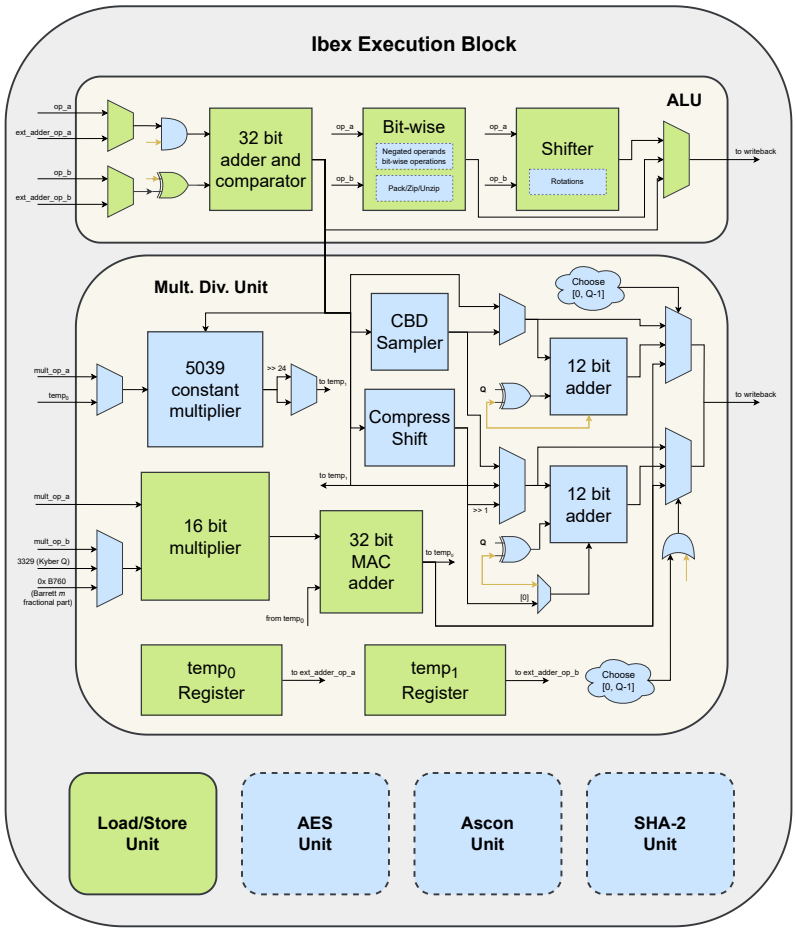

Figure 1. Ibex execution block with extensions

4. Experimental Evaluation

A comprehensive comparative evaluation of the Kyber-512, Kyber-768, and Kyber-1024 security levels was performed, considering the symmetric primitives parametrizations Kyber-Keccak, Kyber-90s, and the novel Kyber-Ascon for each of the `KeyGen`, `Enc`, `Dec` steps. In this document we present only the comparative evaluation of Kyber-512. The evaluation is performed both with and without hardware acceleration for the symmetric primitives and Kyber-specific operations via the *XKyber* extension, comprising a performance comparison as well as energy consumption, memory footprint and die area cost analyses.

Metric	AES-256 CTR		SHAKE128		SHAKE256		Ascon-XOF	
Clock Cycles	1,490	311	30,197	22,355	30,197	22,355	1,558	738
Instructions Retired	1,234	240	22,715	13,713	22,715	13,713	1,501	671
Cycles per byte	128	**26**	180	133	222	164	195	92
Instructions per byte	108	**20**	135	82	167	101	188	84

Table 4. Core operation profiling for each XOF and PRF choice (smaller is better) (white = software, blue = hardware accelerated)

Figure 2 shows the performance of each of the `KeyGen`, `Enc`, `Dec` steps, considering the symmetric primitive choices of Kyber-Keccak, Kyber-90s and the novel Kyber-

Ascon. Software and ISE-accelerated implementations are hatched with circles and diagonal stripes, respectively. The observed performances follow the analysis of the core operation throughputs shown in Table 4, with Kyber-90s being the fastest Kyber symmetric parametrization. Notably, software only Kyber-90s is faster than hardware accelerated Kyber-Keccak and Kyber-Ascon for all 3 Kyber steps. Kyber-Keccak and Kyber-Ascon show very similar performance, with a slight advantage to Kyber-Ascon comparing software only implementations (<1% difference) and a greater advantage to Kyber-Ascon comparing hardware accelerated implementations (17% difference). Kyber-Ascon shows the most significative performance benefits from the use of ISEs (28% difference).

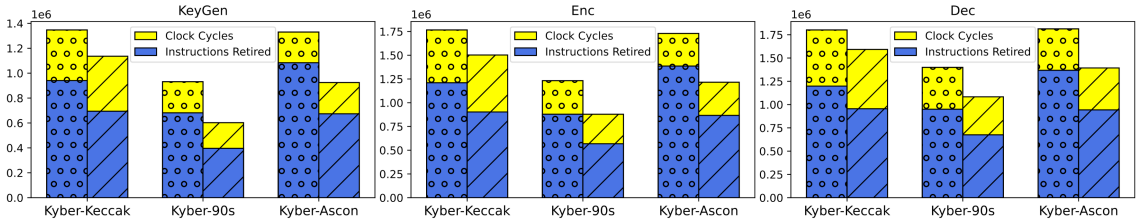

Figure 2. Kyber performance with hardware acceleration of symmetric primitives

Figure 3 shows the performance of the 3 Kyber steps using the *XKyber* extension. A baseline software implementation of Kyber-90s is compared to an implementation with only the symmetric primitives accelerated with *Zkne* and *Zknh* and an implementation with both symmetric acceleration and Kyber-specific acceleration. The impact of accelerating Kyber operations via *XKyber* is roughly the same as accelerating the symmetric primitives via *Zkne* and *Zknh*. From Figure 2, Kyber-90s is the best case for hardware acceleration of symmetric primitives. If the Kyber variant in question were Kyber-Keccak or Kyber-Ascon, hardware acceleration via *XKyber* would outperform hardware acceleration of symmetric primitives via *Zbkb* or *XAscon*.

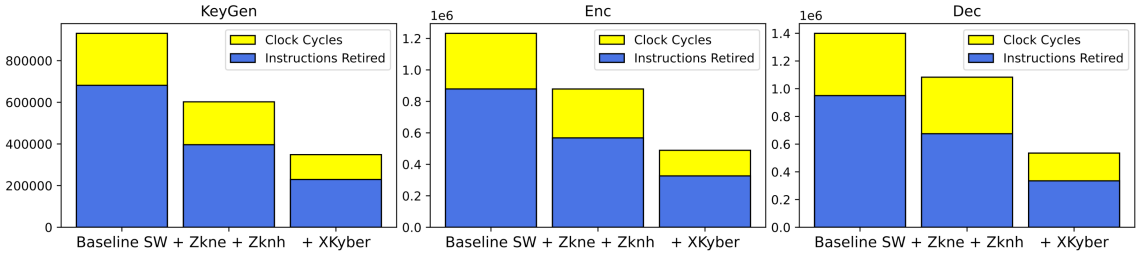

Figure 3. Kyber-90s performance with XKyber extension

Internal Kyber operations show better code size, as depicted in Figure 4. The baseline software implementations are hatched with circles, while *XKyber*-accelerated implementations are hatched with diagonal stripes. The baseline implementation of Kyber-512 shows a total code size of 5090 bytes (not including symmetric primitives), while the *XKyber*-accelerated implementation shows a code size of 4314 bytes, a 15% improvement.

A more comprehensive memory footprint analysis is performed considering the entire cryptosystem, not only the Kyber implementation but the symmetric primitives used in Kyber as well. Table 5[1] compares 3 different practical cryptosystems considering

[1] Adapted from subproduct [Gewehr et al. 2024]

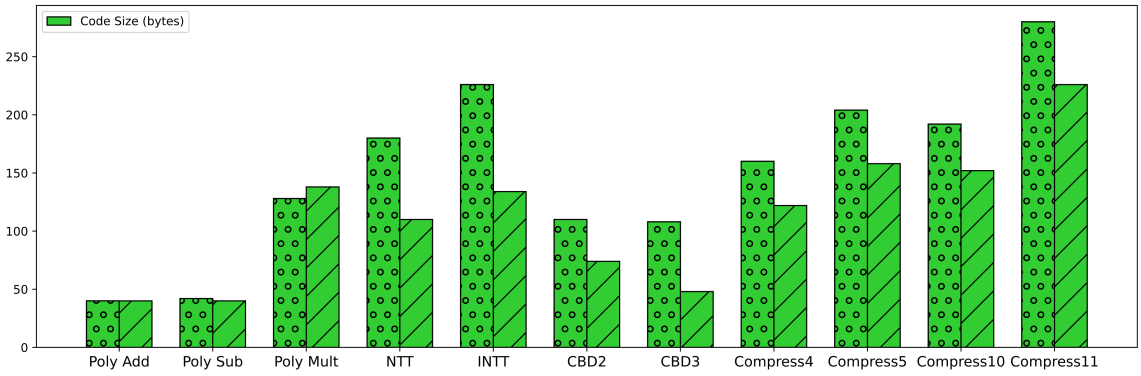

Figure 4. Memory footprint of Kyber operations accelerated by XKyber.

authenticated encryption (AES in CCM mode or Ascon-128), hash functions (SHA-2, SHA-3 or Ascon-Hash) and KEM (Kyber-512).

Cryptosystem	AES CCM + SHA-3 + Kyber-Keccak		AES CCM + SHA-2 + Kyber-90s			Ascon-128 + Ascon-Hash + Kyber-Ascon	
Code size	10,238	10,076	11,842	10,490	9,788	10,080	9,842
Static data	1,480	1,480	2,248	1,223	1,223	256	256
Total	14,854	14,692	17,226	14,849	14,147	13,472	13,234

Table 5. Kyber-512 cryptosystem total memory footprint in bytes (white = software, blue = [Zbkb, Zkne + Zknh, Xascon], green = Zkne + Zknh + XKyber)

The energy consumption of each Kyber step for each Kyber symmetric parametrization is shown in Figure 5. Energy spent due to data and instructions memory operations are shown, respectively, in yellow and green. Energy spent due to data processing inside the Ibex processor is shown in blue. The baseline software-only implementations are shown with bars hatched with circles, while implementations with hardware-accelerated symmetric primitives are shown in bars hatched with diagonal stripes.

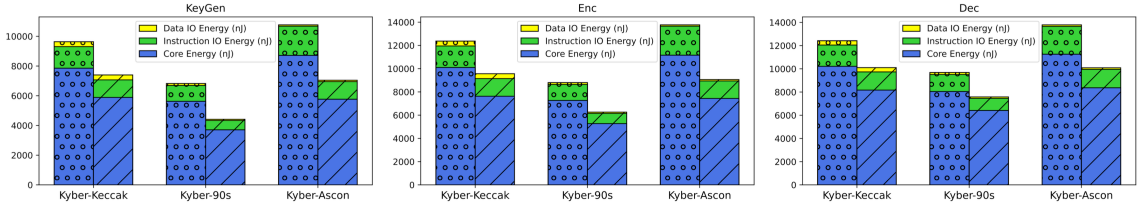

Figure 5. Energy consumption of Kyber variants

The core energy dominates the total energy consumption due to the internal processing of data rather than moving data to/from memory. Despite both having similar performance, Kyber-Ascon shows higher energy consumption than Kyber-Keccak, as Kyber-Ascon executes more instructions per time unit, increasing average power, and therefore, total energy. Following the performance trend shown in Figure 2, software-only Kyber-90s shows slightly better energy consumption than hardware-accelerated Kyber-Keccak, which further improves with hardware acceleration.

Figure 6 shows the energy consumption of each Kyber step, considering a baseline software-only implementation of Kyber-90s, an implementation with hardware accelerated symmetric primitives only, and an implementation with hardware-accelerated

symmetric primitives and Kyber internal operations via *XKyber*. The total energy consumption follows the trend for performance, with energy consumption gains obtained from the *Zkne* and *Zknh* extensions being equivalent to gains obtained from the *XKyber* extension. Note the scale difference for each plot.

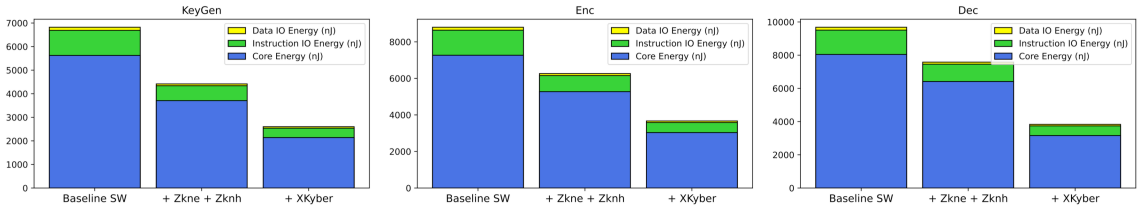

Figure 6. Energy consumption of XKyber extension in Kyber-90s

Table 6 shows synthesis results for the Ibex core and each ISE being explored. The *Zkne* and *Zknh* combination showing a 11% increase and *Xascon* showing a 9% increase. *Zbkb* shows a smaller 1% cell area increase, but shows the greatest increase in cell count at 13%. *XKyber* shows a similar increase of 4 kGE as the *Zkne* and *Zknh* case.

Synthesis results	**Ibex baseline**	**Ibex + Zknh + Zkne**	**Ibex + Zbkb**	**Ibex + XAscon**	**Ibex + Zknh + Zkne + XKyber**
Cell Area (μm^2)	11,238	12,447	11,307	12,210	13,734
Net Area (μm^2)	6,992	7,278	5,142	8,181	7,568
Total Area (μm^2)	18,230	19,726	16,449	20,391	21,302
Cell Count (# instances)	10,289	11,687	11,769	11,010	13,131
Equivalent NAND2 gates	34,433	38,132	34,642	37,408	42,076
Slack @ 500 MHz	0	0	0	0	0

Table 6. Area comparison of ISEs for Kyber hardware acceleration.

5. Conclusion

The choice of Kyber symmetric primitives significantly impacts the performance and energy consumption of Kyber in resource-constrained embedded systems. Compared to the standard Kyber-Keccak, Kyber-90s offers 28% better performance while consuming 27% less energy in a software-only implementation of Kyber-512. Kyber-Ascon shows similar performance to Kyber-Keccak, but may be an attractive option for severely memory-constrained systems at the 128-bit security level.

Hardware acceleration of Kyber symmetric primitives also shows significant gains. In Kyber-512, 32% gains for both performance and energy consumption for the Kyber-90s parametrization are observed. Additionally, adding the *XKyber* extension for accelerating internal Kyber operations can enhance performance and energy consumption even further, providing gains of 46% and 44%, respectively, considering Kyber-512 with hardware acceleration via *Zkne* and *Zknh*. As an extra benefit, *XKyber* reduces Kyber code size by 15%. Hardware acceleration comes at an area cost of 10% of a baseline Ibex core, both for symmetric primitives and *XKyber*.

References

Albrecht, M. R., Hanser, C., Hoeller, A., Pöppelmann, T., Virdia, F., and Wallner, A. (2018). Implementing RLWE-based Schemes Using an RSA Co-Processor. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

Alkim, E., Evkan, H., Lahr, N., Niederhagen, R., and Petri, R. (2020). ISA Extensions for Finite Field Arithmetic - Accelerating Kyber and NewHope on RISC-V. Cryptology ePrint Archive, Paper 2020/049.

Banerjee, U., Ukyab, T. S., and Chandrakasan, A. P. (2019). Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. (2021). Ascon v1.2: Lightweight Authenticated Encryption and Hashing. *Journal of Cryptology*.

Fritzmann, C., Sharif, U., Mueller-Gritschneder, D., Reinbrecht, C. R. W., Schlichtmann, U., and Sepúlveda, M. J. (2019). Towards Reliable and Secure Post-Quantum Co-Processors based on RISC-V. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

Fritzmann, T., Sigl, G., and Sepúlveda, M. J. (2020). RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.

García-Morchón, Ó., Kumar, S. S., and Sethi, M. (2019). Internet of Things (IoT) Security: State of the Art and Challenges. *RFC*, 8576:1–50.

Gewehr, C., Luza, L., and Moraes, F. G. (2024). Hardware Acceleration of Crystals-Kyber in Low-Complexity Embedded Systems With RISC-V Instruction Set Extensions. *IEEE Access*.

Karabulut, E. and Aysu, A. (2020). RANTT: A RISC-V Architecture Extension for the Number Theoretic Transform. *International Conference on Field-Programmable Logic and Applications (FPL)*.

Lee, J., Kim, W., Kim, S., and Kim, J.-H. (2022). Post-Quantum Cryptography Coprocessor for RISC-V CPU Core. *International Conference on Electronics, Information, and Communication (ICEIC)*.

Miteloudi, K., Bos, J., Bronchain, O., Fay, B., and Renes, J. (2023). PQ.V.ALU.E: Post-Quantum RISC-V Custom ALU Extensions on Dilithium and Kyber. Cryptology ePrint Archive, Paper 2023/1505.

Nannipieri, P., Matteo, S. D., Zulberti, L., Albicocchi, F., Saponara, S., and Fanucci, L. (2021). A RISC-V Post Quantum Cryptography Instruction Set Extension for Number Theoretic Transform to Speed-Up CRYSTALS Algorithms. *IEEE Access*.

NIST (2024). Module-lattice-based key-encapsulation mechanism standard.

NSA (2023). Commercial National Security Algorithm Suite 2.0.

Park, J.-Y., Moon, Y.-H., Lee, W., Kim, S.-H., and Sakurai, K. (2022). A Survey of Polynomial Multiplication With RSA-ECC Coprocessors and Implementations of NIST PQC Round3 KEM Algorithms in Exynos2100. *IEEE Access*.

Xin, G., Han, J., Yin, T., Zhou, Y., Yang, J., Cheng, X., and Zeng, X. (2020). VPQC: A Domain-Specific Vector Processor for Post-Quantum Cryptography Based on RISC-V Architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*.