ATHOS: A Hybrid Accelerator for PQC CRYSTALS-Algorithms exploiting new CV-X-IF Interface

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

IEEE SYSTEMS, MAN AND CYBERNETICS SOCIETY SECTION

Received 10 September 2024, accepted 28 November 2024, date of publication 4 December 2024,
date of current version 12 December 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3511340

RESEARCH ARTICLE

ATHOS: A Hybrid Accelerator for PQC CRYSTALS-Algorithms Exploiting New CV-X-IF Interface

ALESSANDRA DOLMETA, (Graduate Student Member, IEEE), MAURIZIO MARTINA, (Senior Member, IEEE), AND GUIDO MASERA, (Senior Member, IEEE)
Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Turin, Italy
Corresponding author: Alessandra Dolmeta (alessandra.dolmeta@polito.it)

This work was supported by the EU TRISTAN Project through the Chips Joint Undertaking and its members under Grant 101095947.

ABSTRACT ATHOS (**A**ccelerated **T**echnology for **H**ardware **O**ptimization with RI**S**C-V) is introduced as a robust solution for enhancing cryptographic operations, explicitly designed for RISC-V architectures. Addressing the challenges of implementing cryptographic algorithms, ATHOS leverages a combination of both tightly and loosely coupled accelerators. A key innovation of ATHOS is its exploitation of the CV32E40X core via the novel Core-V eXtension InterFace (CV-X-IF). This pioneering work is one of the first to utilize this interface in real-world applications, offering a unique foundation for extensive exploration of acceleration approaches. Utilizing the CV-X-IF simplifies the insertion of new instructions into the Instruction Set Architecture (ISA) and streamlines the integration of tightly coupled accelerators without requiring modifications to the toolchain. This work focuses on the implementation and integration of various accelerators into the RISC-V microcontroller X-HEEP, adding new instructions and external IPs for the Numeric Theoretic Transform (NTT), its Inverse (INTT), and Keccak transformation. Our complete architecture is implemented on ASIC 65µm technology, resulting in a 1.47× area overhead for the microcontroller. Additionally, it improves CRYSTALS-Kyber's and CRYSTALS-Dilithium's total clock cycle respectively by up to 7.74× and to 4.12× compared to the baseline software implementation, demonstrating the potential of this hybrid system and marking one of the first real applications of the CV-X-IF interface.

INDEX TERMS RISC-V, CV-X-IF, accelerator, applied cryptography, post-quantum cryptography, lattice-based codes, hardware design, ASIC.

I. INTRODUCTION

Advances in quantum computing notably threaten the security of conventional public-key cryptosystems. In response, Post-Quantum Cryptography (PQC) has been developed to safeguard data confidentiality in communication channels. To address this urgent need, the National Institute of Standards and Technology (NIST) has taken the lead in standardizing PQC, announcing selected algorithms for standardization and round 4 candidates in 2022 [1]. These algorithms can be implemented on classical computers and integrated into contemporary communication infrastructures. Given the memory-intensive nature and repetitive operations of PQC, extensive efforts have been made to expedite its processes through hardware and software enhancements. Recent data indicate that within the past five years alone, the number of publications on PQC has surged, comprising nearly half of the entire body of work published in this domain over the preceding 25 years [2]. Relying solely on software implementations presents notable inefficiencies, due to intricate schemes, frequent memory access, and iterative operations inherent to PQC algorithms. Recognizing these limitations, researchers

The associate editor coordinating the review of this manuscript and approving it for publication was Joao Bernardo Ferreira Sequeiros.

have increasingly focused on hardware-based solutions to accelerate PQC implementations. Recent studies have proposed various hardware accelerators explicitly designed for this purpose. These accelerators can be broadly categorized into two distinct implementation methods: i) loosely coupled accelerators, and ii) tightly coupled accelerators. The first method integrates a memory-mapped accelerator onto system buses like OBI, AXI, and AHB. For instance, [3] uses HW/SW co-design techniques to accelerate NTT transformation and hash generation. The same approach has been used for signature algorithms in [4]. Reference [5] presents a memory-mapped Keccak [6] accelerator for Kyber [7]. The second method involves embedding a tightly coupled accelerator as a functional unit within the CPU's microarchitecture. Reference [8] integrates a set of powerful tightly coupled accelerators to speed up lattice-based PQC; an optimized and masked version is then proposed in [9]. Reference [10] provides an Instruction Set Extension (ISE) that implements finite field operations with subsequent reduction of the result. Reference [11] exploits a PQC arithmetic unit, with *fqmul* and *reduce* accelerations, including logic for butterfly operations in CRYSTALS-algorithms. Reference [12] implements a flexible NTT that can be used in several lattice-based cryptography protocols. Reference [13] explores hardware acceleration through ISEs in a low-end 32-bit RISC-V core for four different parametrizations of Kyber symmetric primitives. Reference [14] proposes a unified memory arrangement and dedicated ALUs for Kyber and Dilithium, capable of accelerating several polynomial operations. Reference [15] proposes ISE to improve the efficiency of polynomial arithmetic and sampling. Reference [16] implements NTT and integrates a k^2-reduction instruction and auxiliary instructions to facilitate NTT-coefficient storage. Reference [17] integrates Keccak and NTT for Dilithium [18]. Reference [19] implements a lattice-based cryptography processor with customized Single-Instruction-Multiple-Data (SIMD) instructions. Reference [20] presents a domain-specific coprocessor based on a matrix extension of RISC-V architecture for Module-Learning-With-Error (Module-LWE). Reference [21] proposes RISC-V ISE for NIST PQC standard algorithms and round 4 candidates. Reference [22] propose a PQC coprocessor with RISC-V Instruction Set Architecture for PQC-algorithms. Reference [23] present a custom extension for polynomial arithmetic, supporting Kyber and Dilithium. Reference [24] implement custom instructions that can perform vectorized operations on variable length and data width polynomials for lattice-based algorithms. Reference [25] present a Dilithium-based hardware accelerated secure boot architecture for RISC-V.

A. OUR WORK

Our work focuses on investigating new RISC-V interfaces to expedite post-quantum cryptographic algorithms. We implement ATHOS, a combination of both loosely and tightly coupled accelerators, which communicate with the primary RISC-V CPU core via the Extendible Accelerator Interface (XAIF) and the novel Core-V eXtension InterFace (CV-X-IF) [26]. This new interface facilitates smooth integration, since it does not require any core or toolchain modification, and offers high flexibility, low implementation cost, and low latency. This study offers four key contributions. *First*, we show how the CV-X-IF interfaces enable seamless integration with the primary RISC-V CPU core in implementing cryptographic primitives. *Second*, we present tightly coupled accelerators that integrate into the core pipeline using the CV-X-IF interface. We introduce over 20 new instructions tailored to CRYSTALS-Kyber without requiring core or toolchain modifications. *Third*, we propose loosely coupled accelerators capable of performing: a) Keccak function, and b) NTT and its Inverse (INTT). *Four*, we present the ASIC implementation to evaluate ATHOS's performance and cost. Furthermore, while Kyber is the primary example, the implementation is also tested with Dilithium.

B. ORGANIZATION

Section II analyzes hardware-software acceleration methods for PQC algorithms, while subsection II-C focuses on the integration strategy used. Section III examines ATHOS architecture, focusing first on the hardware implementation of the tightly accelerators' block and secondly on ATHOS IPs. Section IV discusses results and compares them with previous approaches. Finally, Section V outlines future work and summarizes findings.

II. BACKGROUND

A. HARDWARE ACCELERATION ON RISC-V

RISC-V represents a liberating force in processor architecture, offering an open and extensible Instruction Set Architecture (ISA). An ISA serves as a bridge between hardware and software, defining fundamental operations for processors. One of the main characteristics of RISC-V is that it enables developers to create open-source hardware with dedicated accelerators. This section will introduce the types of integration for these accelerators, namely tightly and loosely coupled configurations, and the various interfaces commonly employed to facilitate this integration. Each combination offers different trade-offs between implementation cost and performance, depending on integration difficulty, flexibility, available memory, and speed.

Loosely coupled accelerators use memory-mapped CPU interfacing, requiring no core microarchitecture modifications. This simplifies integration and provides flexibility. These accelerators enhance performance and allow parallel operation with the CPU, but data transfer between the memory and the accelerator may require significant internal memories and impact latency. This issue can be mitigated by employing Direct Memory Access (DMA), which is helpful for PQC algorithms handling large data volumes, as demonstrated in [3], [4], [5], and [25].

Instead, tightly coupled accelerators are integrated within the CPU's microarchitecture, offering direct access to the

register file. This reduces area overhead and data transfer latency. However, implementing tightly coupled accelerators is more challenging than their loosely coupled counterparts. They require modifications to the CPU's microarchitecture and the toolchain, limiting compatibility and flexibility and making integration more difficult. This is evidenced in [8], [9], [10], [11], [13], [14], [15], [19], [20], [22], [23], and [24].

In these cases, the critical aspect is the interface between the accelerator and the CPU, which must maintain coherence with the core's pipeline signals unless significant changes are made. Several interfaces are commonly used in implementing RISC-V accelerators, each providing distinct advantages depending on specific requirements and design constraints. AXI (Advanced eXtensible Interface) [27] is one of the most popular interface standards, known for high-performance and high-bandwidth communication. It is widely used in various implementations, such as [5], [28], and [29]. Reference [30] exploits the NICE (Nios II Custom Instruction Extensions) interface, which allows the creation of user-defined instructions thanks to four different channels: request channel, response channel, memory request channel, and memory response channel. TileLink [31] is another standard interface designed for on-chip communication within the RISC-V open-source processor ecosystem. Used in [32], it provides a coherent, high-performance communication protocol that supports complex memory hierarchies. Reference [33] exploits a coprocessor interface, which is called Rocket Chip Coprocessor (RoCC) [34], obtaining complete customization only with the Rocket chip generator tool. All these kinds of interfaces require major modifications to the RTL (such as to the core) or necessitate bridges to connect the accelerator to the system. They also often require modifications to the toolchain or specific tools from the interface designer. These requirements can significantly complicate the integration process.

The CV-X-IF interface overcomes many integration barriers by providing a generalized framework suitable for implementing custom coprocessors and ISA extensions for existing RISC-V processors [26]. It simplifies the implementation of tightly coupled accelerators by adding custom or standardized instructions without altering the CPU's decode unit. This approach retains the efficiency benefits of tightly coupled accelerators while reducing integration complexity, offering a more accessible solution. By understanding and leveraging these interfaces, developers can optimize the design and implementation of RISC-V-based systems, enhancing both performance and flexibility. Figure 1 illustrates the main differences between the traditional method and the CV-X-IF method. In a typical RISC-V core, incorporating a custom instruction requires altering the decode and execution stages to identify and manage new custom opcodes, ensuring correct operand routing, and integrating the Coprocessor into the existing datapath. On the other hand, the CV-X-IF method equips the RISC-V core with a dispatcher that sends instructions to the interface. This approach doesn't alter the CPU itself but redirects all necessary signals to the CV-X Interface.

B. ALGORITHMS

In the context of the NIST PQC standardization process, significant strides were made in cryptographic standards in 2022. This process, aimed at identifying quantum-resistant cryptographic algorithms to secure digital information in the post-quantum era, advanced four round 3 candidates and introduced four additional candidates for round 4 evaluation. These candidates, chosen based on their robust mathematical foundations, fall into three distinct categories: lattice-based, code-based, and hash-based cryptography. Among the four standardization schemes of round 3, there are CRYSTALS-Kyber [7], [35] and CRYSTALS-Dilithium [18], [36]. Kyber is a Key Encapsulation Mechanism (KEM) based on the Module-LWE problem. It achieves IND-CCA2 security, and it is built from Kyber.CPAPKE, an IND-CPA-secure Public Key Encryption (PKE) scheme; besides, it uses a modified Fujisaki-Okamoto (FO) transform. This transformation involves utilizing a random oracle (hash function) to create a shared secret, encapsulating and decapsulating it, and deriving the symmetric key and final shared secret through a Key Derivation Function (KDF). Kyber offers three security levels (512, 768, and 1024), each defined by specific parameters including polynomial length, polynomial vector dimension, modulo, and values for Centered Binomial Distribution (CBD) sampling. Further details on Kyber can be found in [35]. Dilithium is a lattice-based Digital Signature scheme. As CRYSTALS-Kyber, it shares the same polynomial length but employs a larger modulo. The different security levels in Dilithium are indicated as 1, 3, and 5. Further information on Dilithium can be found in [36]. This study focuses on optimizing the Kyber algorithm, which serves as the primary test case for evaluating the ATHOS structure. All 20 new instructions are tailored for Kyber. Dilithium has also been tested, using only the Keccak IP, but with a narrower focus.

C. INTEGRATION STRATEGY

When implementing a specific cryptographic function on a RISC-V core, three strategies can be followed: software optimization, ISA extension with custom instructions, and loosely coupled accelerator design. Emphasis should be first placed on software optimization due to its inherent flexibility and cost-effectiveness. Then, performance analysis of the software model allows us to pinpoint remaining computational bottlenecks. Subsequently, different options must be evaluated regarding area, latency, memory usage, and energy consumption. Custom processor instructions are explored as a secondary option, leveraging their adaptability with minimal hardware overhead. Only when these methods fall short we do entertain the notion of a loosely coupled accelerator. However, a comprehensive design and synthesis of both architectures would be necessary to ascertain whether a tightly or loosely coupled approach is more suitable. The

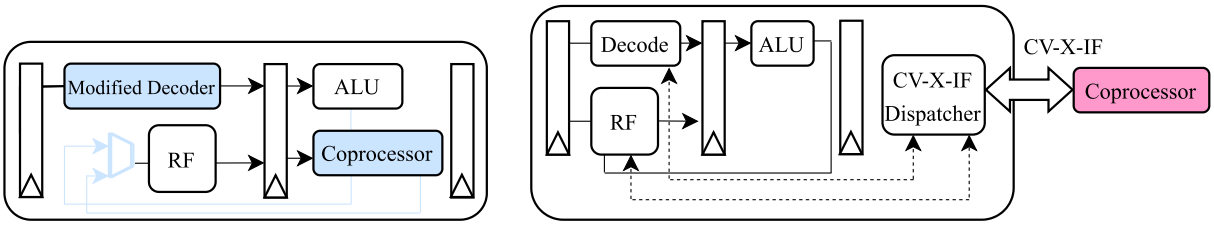

FIGURE 1. Common tightly-coupled approach vs CV-X-IF.

early evaluation of the potential of each solution is helpful to avoid committing to a specific accelerator design prematurely and to save time. To achieve our objectives, we employed a straightforward and practical design flow for determining whether operations should be offloaded to loosely coupled or tightly coupled accelerators.

1) PROFILING

The initial step in identifying functions suitable for acceleration is to profile the code. After pinpointing bottlenecks, we must identify the functions that are the best candidates for acceleration.

TABLE 1. Number of calls (Calls), operands (OP1, OP2) and result (RES) lengths of some the main functions in the CRYSTALS-Kyber-512 algorithm.

Group	Function	Calls	OP1	OP2	RES
Reduction	montg	29,056	32	-	16
	barrett	13,440	16	-	16
Hash	Keccak	79	1600	-	1600
Mul	NTT	10	4096	2048	4096
	INTT	7	4096	2048	4096
Sampling	rej_uniform	12	4048	-	4048
	cbd	6	4096	-	4096

2) ANALYSIS

Important factors in this analysis include the nature and the dimension of the data, the complexity of the operations, and the structure of the function.

- For operations where a small number of inputs result in a single output, tightly coupled accelerators are always preferable, as they can efficiently handle such tasks, potentially within a single clock cycle or over multiple cycles depending on the operation's complexity and system constraints. In this way, inputs and output are read/written directly from/to the register file of the core.
- For operations where input and output data are vectors or large dimensions, a more detailed analysis is required.
  - If the operations on vector elements are independent of each other, a tightly coupled accelerator is usually beneficial. In this case, the best solution is to find a sub-function that is called and executed many times, capable of working with a low number of inputs, and implement it with tightly coupled accelerators.
  - If the operations involve interdependent data with significant memory handling and access requirements, the choice becomes less clear-cut. In this circumstance, it would be necessary to pre-load all the data necessary for the correct execution of the function within the accelerator. In a tightly coupled accelerator, it is required to initially load each data into the core register file, and then send them to the accelerator, which can only start the computation after completing the load instructions. There are therefore two loads for each data. Using a loosely coupled accelerator with a DMA unit would instead allow the loading of all the necessary data directly from memory to the accelerator. In this case, a loosely coupled approach would be preferable.

Table 1 shows Kyber-512 analysis' results, reporting some of the main functions for illustration purposes. The first two functions, belonging to the Reduction group, are excellent candidates for tightly acceleration, as discussed in subsection III-A. They are called numerous times within the algorithm and handle minimal input and output data. The analysis of the other functions is more complex. For these functions, the input and output data are too large to be suitable for a tightly integrated accelerator. However, two distinct cases can be identified. For functions like Keccak, NTT, and INTT, which are well-defined transformations and arithmetic operations with specific structures involving multiple steps and loops, it is beneficial to implement loosely-coupled accelerators that handle the entire execution of the function (subsubsection III-B1 and subsubsectionIII-B2). Also, this approach is preferable when the function that must be accelerated needs to handle the entire vector or state simultaneously. For instance, in the case of Keccak, all 1600 bits are required concurrently to initiate the permutation. This is advantageous when the acceleration gain outweighs the overhead from data transfer. In contrast, sampling functions (discussed in more detail later in subsection III-A) also have multiple steps and loops, but they lack a well-defined structure that can be easily implemented in a single loosely coupled block. Therefore, it is more convenient to accelerate sub-functions within these cases, looking for smaller tasks called multiple times, which work with small amounts of data. For example, the inner loops of rej_uniform and cbd functions are called respectively 1890 and 1728 times, with input-output lengths of 16-bit, making them the perfect functions for this kind of acceleration. Additionally to sampling functions, further analyses were conducted on other portions of the Kyber algorithm, identifying ideal candidates for tight

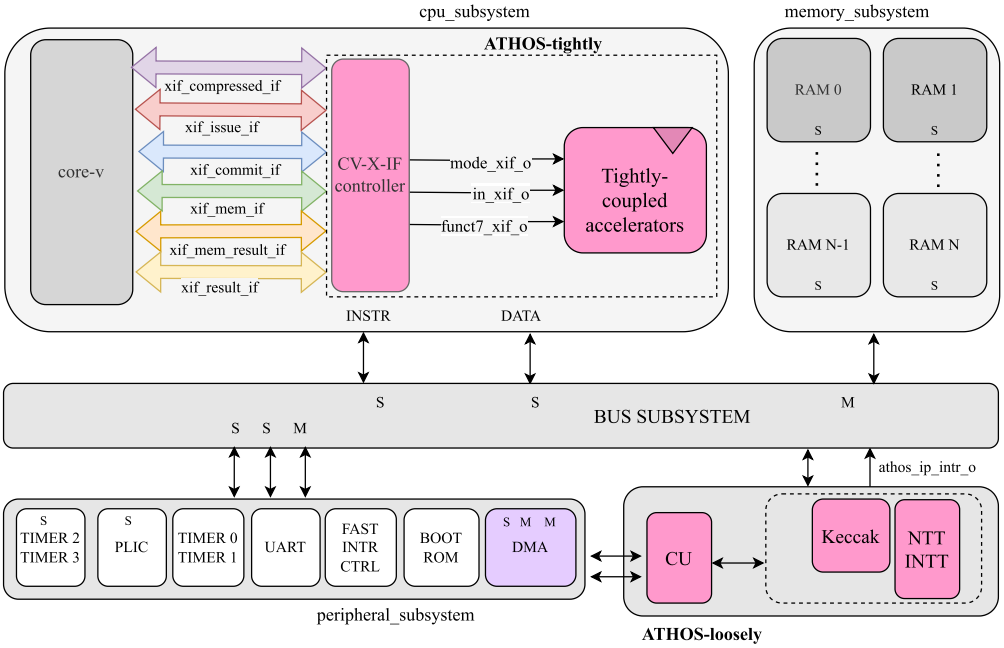

FIGURE 2. ATHOS complete architecture - the grey components belong to X-HEEP.

acceleration in polynomial arithmetic functions such as: poly-compress, poly-decompress, poly-to-bytes, poly-from-bytes, poly-from-msg, and poly-to-msg [37].

III. ARCHITECTURE

In the case of ATHOS, the authors opted to leverage loosely and tightly coupled accelerators. This decision was driven by the pursuit of the best trade-off between performance and area and by the objective of creating a highly flexible system of accelerators. By adopting this approach, ATHOS provides a versatile framework with a *skeleton* for external IPs and internal accelerators that can be easily extracted from the system in which it was tested and deployed onto another microcontroller with specific requirements, ensuring adaptability and scalability across various RISC-V environments. The proposed hardware architecture is shown in Figure 2. It includes two main modules: **ATHOS-tightly**, which contains all the accelerators used with the new Instruction Set Extensions and the controller for the CV-X-IF interface (top-middle part of Figure 2), and **ATHOS-loosely**, which is the collection of the two external IPs (bottom-right part of Figure 2). These accelerators have been integrated into X-HEEP (eXtendable Heterogeneous Energy-Efficient Platform), an open-source RISC-V microcontroller described in SystemVerilog. Grey components in Figure 2 are part of X-HEEP system, while the colored ones have been implemented in this work. The processing system executes the software application and is connected to other system elements needed for the simulation (i.e., timer, DMA, memories, UART, ...). The X-HEEP DMA has been modified. For further details, refer to [38]. *Firstly*, the XAIF framework facilitates the effortless integration of loosely coupled accelerators into the system without necessitating any alterations to the Microcontroller Unit (cpu_subsystem). This interface has slave and master ports, seamlessly interfacing with the internal bus via the OBI bus protocol [39]. It also incorporates interrupt ports, enabling efficient communication with the host ATHOS-loosely upon completion of accelerator operations (athos_ip_intr_o). Each line is connected to the X-HEEP PLIC interrupt controller, which can be controlled via software. *Secondly*, the CV-X-IF interface enables the incorporation of ATHOS-tightly into the system architecture. The CV-X-IF empowers the expansion of the CPU's functionality by adding custom instructions that do not require modification to the RISC-V toolchain or the core decoding unit. Indeed, extensions are implemented as distinct modules external to the CPU and are seamlessly integrated into the pipeline of the system. Inline assembly is used to leverage ATHOS's specialized capabilities, allowing direct embedding of assembly instructions for precise hardware control.

A. ATHOS-TIGHTLY ARCHITECTURE

The proposed hardware architecture for the ATHOS-tightly module is shown in the right part of Figure 3. The interface of this module is designed to facilitate smooth communication between the core and the accelerator through structured packaged signals (in *italic*). Initially, the CV-X-IF controller scrutinizes the instruction opcode, taken from the instruction field of the issue request package ('*issue req*' in Figure 3), coming from the decoding stage of the core. Then, it activates the 'issue valid' signal if it recognizes a match with one

of the custom instructions' opcode. This signal triggers the '*issue_resp*', indicating the accelerator's readiness to receive data. The validity of input source registers is also defined by the issue request package (rs_valid signals, from the '*issue req*' package). If validated, the accelerator samples its inputs in the same clock cycle, ensuring synchronous and efficient data transfer. Once the instruction has been executed, if the core is ready to receive the result and the result is valid, it is written back in the register file through the result interface. This exchange protocol and all the signals involved are shown on the left side of Figure 3. Notably, instructions offloaded by the RISC-V core during its ID stage are speculative, requiring rigorous verification before the accelerator can write back results through commit and result interfaces. Further information about the protocol of this interface can be found in the original repository.[1] The CV-X-IF controller we have implemented evaluates whether the instruction aligns with its predefined set whenever the core signals an 'issue valid' event. If the instruction is unrecognized, ATHOS triggers an exception. However, if the instruction matches its predefined set, ATHOS activates the appropriate block from the range of accelerators integrated within its structure. Then, a multiplexer chooses the output based on the specific instruction's requirements. The standout feature of this accelerator lies in its versatility, facilitated by a meticulously crafted structure that allows for the integration of various accelerator designs. In the context of this paper, ATHOS is designed specifically for integrating various accelerator designs, with just minor modifications to the multiplexer and the CV-X-IF-controller components. Although our focus here is on Kyber for the tightly coupled accelerators, the true versatility of ATHOS lies in its ability to incorporate any accelerator. Once the RTL of the accelerator is completed, it can be inserted into the structure, connected, and its corresponding codification added to the CV-X-IF decoder. This adaptability allows for straightforward integration of different accelerator designs. While there is an initial design effort, once the structure is constructed, as in our case, the benefits of this approach are substantial. To leverage ATHOS's specialized capabilities, we use inline assembly in C, allowing direct embedding of assembly instructions for precise hardware control. Custom instructions are defined with the `.insn` directive, creating processor-specific commands that interface with the accelerators. By specifying the opcode, function codes, and operands, these instructions perform specialized operations on ATHOS. Inline assembly ensures correct mapping of input and output operands to C variables, enabling efficient execution of complex operations and full utilization of the accelerators while maintaining C's high-level flexibility.

1) TIGHTLY-COUPLED ACCELERATORS

This study focuses on accelerating the Kyber algorithm. After conducting a comprehensive analysis of the algorithm and identifying its performance bottlenecks, as discussed in subsection II-C, five top-level accelerators have been specifically designed. Each accelerator is tailored to perform particular instructions based on the opcode, funct3, and funct7 fields.

The five accelerators developed for this purpose are:

- **Montgomery Accelerator** (montg): specialized for efficient Montgomery reduction operations. The Montgomery method [40] implements a fast modular multiplication by converting the modulus into a special form and performing a series of multiplications and shifts.
- **Barrett Accelerator** (barrett): designed to accelerate Barrett reduction. It is another division-free method for modular reduction, first introduced in [41]. It involves precomputing a constant and performing a series of multiplication and shifts to achieve the modular reduction. As for Montgomery, timing leakages are avoided.
- **Matrix Generation Accelerator** (rej_uniform): specifically developed to accelerate one of the matrix composition operations. There is an inner for loop which is called thousands of times and performs multiple arithmetic and shifting operations on two 16-bit inputs. All these steps are implemented in two instructions.
- **Central Binomial Distribution Operations Accelerator** (cbd): targeted at accelerating some of the operations required by the Centered Binomial Distribution (CBD) sampling function (i.e., $\eta 1$ and $\eta 2$). CBD is the function from which noise is sampled [7].
- **Polynomial Arithmetic Accelerator** (poly): tailored to optimize performance for accelerating some of the polynomial arithmetic functions essential to Kyber. In these operations, various bitwise manipulations, such as XOR, shifts, OR, and AND, are performed repeatedly within loops. The polynomial accelerator consolidates these bitwise operations into single, specialized instructions, significantly reducing the number of clock cycles required per loop iteration. This optimization results in a considerable performance improvement by minimizing the overhead associated with executing multiple individual operations.

Figure 4 shows the original clock cycles, obtained running the original code, and the accelerated clock cycles obtained with the new custom instructions. More details of the implementation of the different accelerators can be found in the doc of the following repository.[2] A direct comparison with other works is challenging. In [14], the authors present an ISE for polynomial arithmetic, but there is no direct comparison with our implementation. In [13], the authors develop the *XKyber* extension, achieving a 72% improvement for CBD3 and 78% for compression, compared to our 63.21% and 61.66% gains. However, the code being accelerated in their case is not exactly the same as ours. According to

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

[2] https://github.com/vlsi-lab/ATHOS_doc

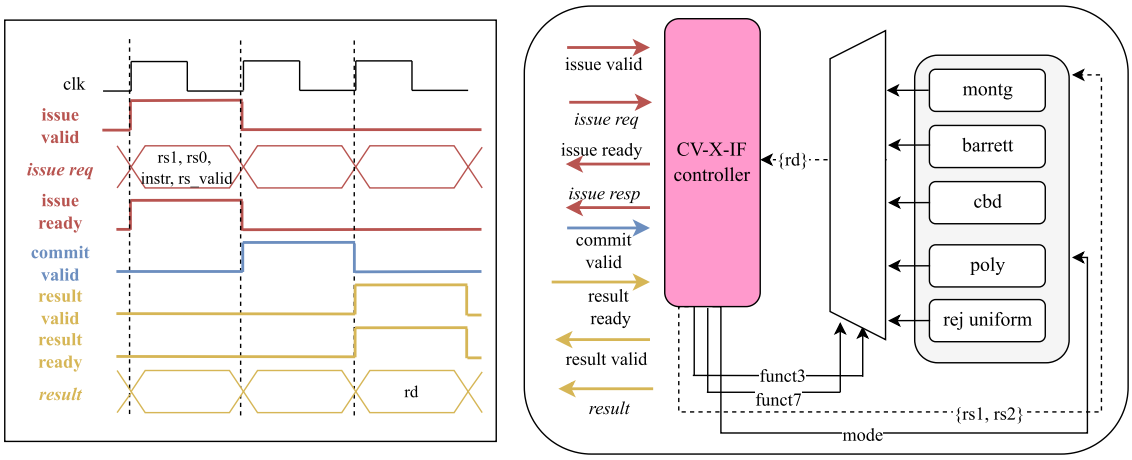

FIGURE 3. CV-X-IF protocol and ATHOS-tightly architecture.

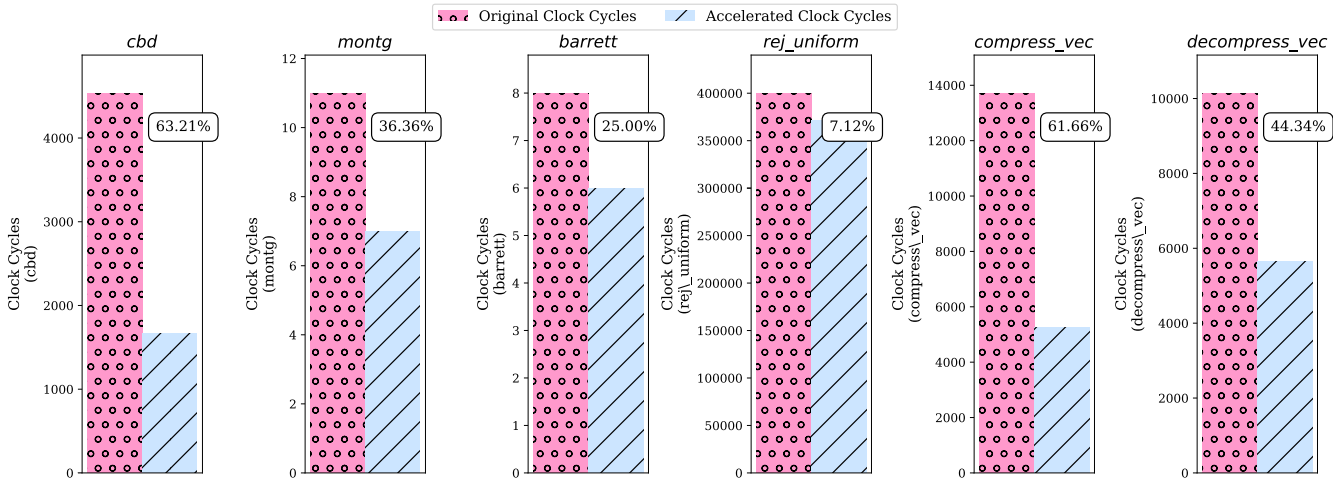

FIGURE 4. Performance of some of the ATHOS-tightly accelerators for Kyber512.

the RISC-V base opcode map [42], we have selected for new instructions *custom-0* instruction opcode: $0 \times 0\text{b}$. All the newly introduced instructions are presented in Table 2. The parameter β represents a value encoded in the funct7 field of the RISC-V instruction; it changes depending on the instruction invoked since different functions require slightly different parameters for completion.

TABLE 2. Instruction set extension for Kyber. `rd` - Destination register, `rs1` - Source register 1, `x0` - Hardwired register.

Instr. Group	.insn	Option
montg	".insn r 0x0b, 0x003, 2, rd, rs1, x0"	
barrett	".insn r 0x0b, 0x004, 0, rd, rs1, x0 "	
cbd	".insn r 0x0b, 0x005, β, rd, rs1, x0 "	cb3 - cycle i
	".insn r 0x0b, 0x005, β, rd, rs1, x0 "	cb2 - cycle i
poly	".insn r 0x0b, 0x006, β, rd, rs1, x0 "	compress
	".insn r 0x0b, 0x006, β, rd, rs1, x0 "	decompress
	".insn r 0x0b, 0x006, β, rd, rs1, x0 "	tobytes
	".insn r 0x0b, 0x006, β, rd, rs1, x0 "	frombytes
	".insn r 0x0b, 0x006, β, rd, rs1, x0 "	tomsg
	".insn r 0x0b, 0x006, β, rd, rs1, x0 "	frommsg
rej_uniform	".insn r 0x0b, 0x001, β, rd, rs1, x0 "	

The optimization results are reported in Table 6. Through profiling, it is evident that the primary bottlenecks are from Keccak and polynomial multiplication. The relative speed-up due to ISA modifications alone is significantly lower than the final achieved speed-up, but it underscores the substantial benefit derived from modifying critical sub-functions which conventional RISC-V extensions may struggle to optimize effectively.

B. ATHOS-LOOSELY ARCHITECTURE

The proposed hardware architecture for the ATHOS-loosely module is shown in Figure 5.

The accelerators communicate via a unique interface, requiring connection through a register file for standard processor control. This is enabled by a versatile *generic register interface*, compatible with APB, AXI-Lite, OBI, and AXI protocols, simplifying integration with most micro-controllers. A description of the required registers in *hjson* format is utilized, with tools generating System-Verilog

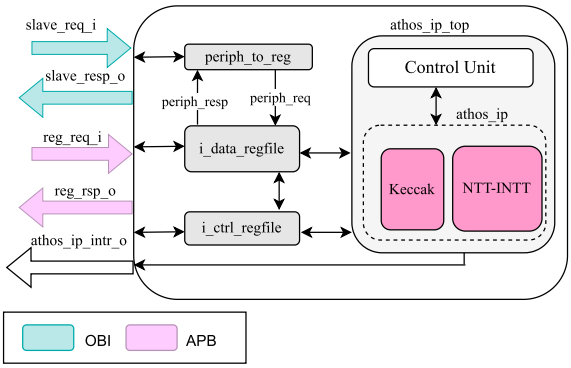

FIGURE 5. ATHOS-loosely top view.

structures (i_data_regfile and i_ctrl_regfile in Figure 5). These structures define registers and fields, along with byte address offsets, facilitating integration and communication. The control unit is designed to efficiently manage the accelerators by overseeing input and output data control, as well as executing three distinct functions. Then, the system manages these multiple accelerators via different software drivers. These drivers act as intermediaries between the processor and accelerators, overseeing their setup, operation, and synchronization. Initially, the system initializes the accelerators by configuring essential parameters like interrupt controllers and DMA, enabling interrupts for processor communication, and ensuring synchronization, which requires the reset of the relevant parameters. These operations set the module for subsequent interactions. After the initialization, dedicated drivers interact with each accelerator to support the following actions:

- **Data Transfer**: input data moves from the memory to the accelerators and vice-versa using DMA, which does not require load/store operations. This step optimizes the transfer process, directly connecting data memory banks to the memory inside the loosely coupled accelerators. This enhances the accelerator's readiness for execution by swiftly providing the required input parameters. As previously mentioned, with this approach, large datasets can be efficiently transferred between the main memory and the accelerator's memory space, by simply proving the starting address of the chunk of data and its size.
- **Operation Execution**: the driver dynamically configures specific bits within the control register file (i_ctrl_regfile in Figure 5), based on the invoked IPs. This mechanism enables ATHOS to discern which IP module has been invoked, facilitating the transmission of corresponding activation signals. Subsequently, upon receipt of these activation signals, the respective IP module initiates its designated function, leveraging pre-collected data to execute its operations.
- **Interrupt Handling**: upon initiating the operation, the driver enters a loop, waiting for an interrupt signal to indicate the completion of processing by the accelerator (athos_ip_intr_o in Figure 5). This loop incorporates interrupt handling mechanisms to efficiently await the signal while allowing the processor to perform other tasks concurrently.

Through this structured approach, software drivers facilitate seamless communication and interaction between the processor and accelerators, optimizing performance and resource utilization within the computational system. The following subsections, subsubsection III-B1 and subsubsection III-B2, describe NTT-INTT and Keccak IPs respectively.

1) NTT-INTT IP

Polynomial arithmetic within polynomial rings R_q typically involves computing the product of two polynomials using the schoolbook multiplication method. This results in a $O(n^2)$ complexity, where n represents the number of digits. To enhance efficiency, cryptographers often opt for NTT-based approaches, which reduce complexity to $O(n \log(n))$. The Cooley-Tukey (CT) [43] and Gentleman-Sande (GS) [44] algorithms are commonly used for NTT computation. CT butterflies facilitate transformation into the NTT domain and GS butterflies for the reverse transformation. Twiddle Factors, denoted as ω, play a crucial role, representing roots of unity for the polynomial and coefficient rings. The work in [45] proposes a pipelined NTT/INTT FPGA architecture with GPMA and SRU, achieving up to 4.8× latency and 4.3× ATP improvements. The authors of [46] introduce a Kyber NTT/INTT accelerator using K-RED and Brent-Kung methods, achieving 262 MHz on Artix-7 with 1405 LUTs. In [47], a Kyber polynomial multiplier with Bi-Core and Barrett reduction achieves 39% better area-time product and the fastest speed on Artix-7.

The proposed IP implements a unified butterfly unit (BU) to handle both NTT and INTT operations. Their computational structures diverge, with NTT employing Cooley-Tukey (CT) butterflies and INTT utilizing Gentleman-Sande (GS) structures. The former follows a decimation-in-time approach ($a + b \cdot \omega_n \pmod q$ and $a - b \cdot \omega_n \pmod q$), while the latter adheres to a decimation-in-frequency approach ($a + b \pmod q$ and $(a - b) \cdot \omega_n \pmod q$). Twiddle Factors are precomputed and stored in ROM memory. The structure includes input and output polynomial memories, with input multiplexers facilitating input polynomial storage before operations. The control unit initiates operations based on start signals, and output multiplexers retrieve resultant polynomials post-operation. Data in Table 3 compare various NTT and INTT implementations across different RISC-V systems. Many cited studies speed up NTT and INTT by employing tightly integrated accelerators embedded within processor architecture. This solution is widely used in the implementation of cryptographic systems because of its efficiency. However, as shown in Table 3, our approach based on loosely coupled accelerators, competes well with state-of-the-art methods, thanks to the use of DMA and external small memories for polynomial coefficient storage. This demonstrates the effectiveness of the proposed approach in accelerating complex cryptographic operations within a

loosely coupled configuration. For further details about the implementation refer to [48].

TABLE 3. Number of cycles NTT-INTT accelerators.

Ref.	Device	Method	NTT	INTT
[11]	RISC-V (CVA6)	Tightly coupled	18,488	18,488
[8]	RISC-V (PULPino)	Tightly coupled	1,935	1,930
[10]	RISC-V (VexRiscv)	Tightly coupled	6,868	6,867
[16]	RISC-V (Hummingbird)	Tightly coupled	4,189	3,481
[21]	CVA6 (CV-X-IF)	Coprocessor	13,880	-
[15]	RISC-V (Ibex)	Tightly coupled	1,454	1,726
[19]	RISC-V	Tightly coupled	1,705	1,925
OURS	**RISC-V (X-HEEP)**	**Loosely coupled**	**1,531**	**1,531**

2) KECCAK IP

Keccak is a fundamental component of the Federal Information Processing Standard (FIPS-202 standard) SHA-3. It is denoted as Keccak[r,c], characterized by a bit rate (r) and capacity (c), which sum determines the width of the state of the Keccak-f permutation. This state width is 1600 bits, organized as a 5×5 matrix of 64-bit words. The number of rounds, n_r, for this permutation is 24. The Keccak algorithm operates through a series of compression rounds, each consisting of five distinct steps: θ, ρ, π, χ, and ι [49]. These steps collectively transform input data into a secure hash. A high-speed Keccak core, developed by the Keccak teams, is often utilized for cryptographic tasks. In this implementation, modifications are made to optimize performance. For instance, the size of the round constant generator is reduced from 64 bits to one byte, simplifying computations within the ι step. Considering the whole ATHOS-loosely structure, shown in Figure 5, the i_data_regfile register file accompanying the accelerator has been specifically tailored to gather the entire state of Keccak. An alternative approach could involve storing the state within a register directly within the accelerator itself. This design choice was made to optimize resource utilization. It allows for a larger register file dedicated to data storage and expands the capacity for accommodating data requirements of other accelerators within the ATHOS-loosely structure. As a result, the trade-off between utilizing space within the accelerator versus the register file was carefully considered, ultimately resulting in an efficient overall system design. The data in Table 4 compares various Keccak implementations across different RISC-V devices.

TABLE 4. Number of cycles in RISC-V based Keccak accelerators.

Reference	Device	Method	Keccak
[21]	CVA6 (CV-X-IF)	Coprocessor	1,632
[8]	PULPino	Tightly coupled	308
[5]	RISC-V (PULPissimo)	Loosely coupled	1,348
[19]	RISC-V	Tightly coupled	404
OURS	RISC-V (X-HEEP)	Loosely coupled	**801**

The solution proposed in this work outperforms other memory-mapped solutions (i.e., [5]) and it surpasses the performance of the coprocessor-based approach implemented on CVA6 ([21]). However, it falls short compared to the two tightly coupled solutions, mainly because of the overhead of exchanging data between the accelerator and the processor. However, it is noteworthy that our approach achieves competitive performance without necessitating extensive core modifications. On the contrary, [8] incorporates Floating Point Registers and General Purpose Registers into the core pipeline structure, which increases the complexity of the core, while [19] extends the datapath from 256 bits to 320 bits and allows more read ports in the SIMD register files, enabling the processing of five 64-bit data simultaneously. Finally, the proposed accelerator is widely applicable across various Post-Quantum Cryptography (PQC) algorithms and, in general, for applications demanding swift and secure hashing, such as cryptographic protocols and secure communication systems. Its efficiency has been demonstrated not only for Kyber operations but also for other cryptographic algorithms, including CRYSTALS-Dilithium.

IV. RESULTS AND COMPARISON

This Section presents experimental results for CRYSTALS-Kyber and CRYSTALS-Dilithium regarding code size, cycles, latency, and implementation complexity. Our project is implemented at the register transfer level (RTL) using SystemVerilog. Functional RTL-level simulations allowed the evaluation of the improvements made by ATHOS. Comparisons against state-of-the-art solutions are shown in terms of code size (Table 5) clock cycles (Table 6 and Table 7) and implementation complexity (Table 9).

A. CODE SIZE

Table 5 summarizes the measured code size for both the baseline and the optimized implementation exploiting always the CV32E40X core. The code size was evaluated using the *riscv32-unknown-elf-size* command and `-O2` optimization flag. This Table and the following ones refer to Kyber security levels as K-512, K-768, and K-1024, and Dilithium levels as D-2, D-3, and D-5. Results in square brackets represent percentage differences with respect to the reference model.

TABLE 5. CRYSTALS-Kyber and -Dilithium code size [bytes].

Alg	Version	Code Size	Alg	Version	Code Size
K-512	pure-sw	23,942	D-2	pure-sw	31,596
	ATHOS	**20,898** [-12.72%]		ATHOS	**25,780** [-18.43%]
K-768	pure-sw	23,900	D-3	pure-sw	31,368
	ATHOS	**20,578** [-13.91%]		ATHOS	**26,002** [-16.95%]
K-1024	pure-sw	24,362	D-5	pure-sw	31,472
	ATHOS	**22,986** [-5.64%]		ATHOS	**26,086** [-17.07%]

ATHOS version demonstrates reductions in code size compared to the *pure-sw* version across all variants: Kyber512, Kyber768, and Kyber1024 get 12.72%, 13.91%, and 5.64% improvement respectively. For Dilithium, the

TABLE 6. Performance comparison of CRYSTALS-Kyber and -Dilithium [C.C. - Clock Cycles].

Alg.	Task	Original C.C.	Final C.C. (ISA)	Speedup (ISA)	Final C.C.	Speedup	Alg.	Task	Original C.C.	Final C.C.	Speedup
K-512	Keygen	1,052,145	895,421	1.10×	135,936	**7.74**×	D-2	Keygen	3,810,662	924,918	**4.12**×
	Enc	1,106,228	959,027	1.15×	161,966	**6.83**×		Sign	11,193,500	6,253,352	**1.79**×
	Dec	1,231,155	991,011	1.24×	222,230	**5.54**×		Verify	3,877,772	1,399,918	**2.77**×
K-768	Keygen	1,674,185	1,597,524	1.05×	238,488	**7.02**×	D-3	KeyGen	6,612,747	1,883,797	**3.51**×
	Enc	1,798,912	1,680,990	1.07×	271,739	**6.62**×		Sign	18,722,385	11,144,277	**1.68**×
	Dec	1,968,664	1,815,300	1.09×	356,642	**5.52**×		Verify	6,578,728	2,215,060	**2.97**×
K-1024	KeyGen	2,612,887	2,459,463	1.06×	399,524	**6.54**×	D-5	KeyGen	11,197,346	3,076,194	**3.64**×
	Enc	2,757,344	2,553,693	1.08×	432,864	**6.37**×		Sign	24,474,065	18,540,985	**1.32**×
	Dec	2,983,573	2,742,547	1.09×	545,443	**5.47**×		Verify	11,303,386	3,521,304	**3.21**×

transition to ATHOS-version resulted in code size reductions of approximately 18.43%, 16.95%, and 17.07%, respectively.

B. CLOCK CYCLES

Table 6 presents performance benchmarks for CRYSTALS-Kyber and CRYSTALS-Dilithium cryptographic algorithms across various security levels. The speed-up factor is evaluated as the ratio between the original and the final clock cycles of each algorithm, where the original clock cycles are the cycles needed by the microcontroller to run the algorithm without any modification to the system, while the final clock cycles are the ones obtained using ATHOS. For Kyber, the analysis includes key generation (KeyGen), encryption (Enc), and decryption (Dec) processes. It should be noted that our decapsulation results include the encapsulation operation. Similarly, Dilithium's evaluation covers key generation (KeyGen), signature (Sign), and verification (Verify) tasks. The reference implementation is executed on the unmodified X-HEEP microcontroller as a baseline, whereas the accelerated version enables the accelerators discussed previously. To make use of them, the reference C code was modified by inserting the proper drivers, or by writing the reference assembly inline code: `-O2` flags is used both for pure software and accelerated simulations. In contrast to the baseline implementations, our optimizations yielded significant speedup factors. While ATHOS shows significant performance improvements specifically tailored to Kyber, Dilithium results demonstrate its versatility and ease of integration with other cryptographic algorithms. In the case of Dilithium, which relies solely on Keccak without any ISE extensions, ATHOS still provides notable speedups across the different processes. Although the performance enhancements for Dilithium are not as pronounced as for Kyber, these results underscore ATHOS's adaptability and potential for customization to suit various cryptographic requirements. Our research outperforms the cycle count achieved by the most recent assembler-optimized ARM Cortex-M4 implementations, as documented in [50] and [51]. The results of clock cycle benchmarks are summarized in Table 7. Leveraging accelerators and ISA extensions for RISC-V architecture has proven to be highly effective in enhancing performance, as exemplified in all the works cited.

TABLE 7. CRYSTALS-Kyber performance comparison: KeyGen/Enc/Dec cycles [kCC - kilo Clock Cycles].

Ref.	Cycles [kCC] K-512	Cycles [kCC] K-768	Cycles [kCC] K-1024
[7]	123/155/188	199/235/275	307/347/397
[50]	455/586/544	864/1,032/970	1,405/1,606/1,526
[10]	710/971/870	-	2,203/2,619/2,429
[8]	150/194/205	274/326/341	350/406/425
[16]	622/785/713	988/1,236/1,133	1,543/1,851/1,719
[11]	420/439/100	695/732/130	1,091/1,127/159
[9]	116/176/186	214/299/313	266/369/393
[19]	89/89/108	164/166/197	195/198/244
OURS	136/162/222	238/272/357	400/433/545

1) CRYSTALS-KYBER

Compared to [10], our approach, utilizing VexRiscv, demonstrates significant performance improvements across all key sizes (K-512, K-768, K-1024). Specifically, for K-512, our method reduces clock cycles by approximately 80.87% (from 710 to 136), 83.34% (from 971 to 162), and 74.52% (from 870 to 222). Reference [11] employs a PQC ALU with *fqmul* and *reduce* accelerations and butterfly operations, achieving lower overall improvement compared to ATHOS. Our method shows substantial performance gains compared to [16], with reductions in clock cycles ranging from approximately 68.32% to 79.32%. Although [20] proposes a dual-core architecture using the Rocket Core (RV64IMC configuration) and Chisel, it is mentioned for completeness but not included in Table 7. RISQ-V, presented in [8], is one of the most similar works to ATHOS, featuring tightly coupled accelerators and 29 new instructions for lattice-based algorithms. This includes parallel butterfly operations, on-the-fly Twiddle factor generation, vectorized modular arithmetic, and efficient random polynomial generation, similar to the accelerators in ATHOS. Both works are implemented on the same technology, as discussed in subsection IV-C. Reference [9] is similar but uses masked HW accelerators. For comparison, non-masked and optimized clock cycles are considered. Our approach outperforms [8] in keygen and encapsulation processes for all Kyber security levels, except for decapsulation. For K-512, our KeyGen shows a 9.33% decrease in clock cycles, encryption (Enc) improves by 16.49%, but decryption (Dec) increases by 8.29%. The optimized version of [9] performs better in KeyGen and Dec, while our results excel in K-512 and K-768 Enc. Comparing clock cycles offers insights into implementation efficiency,

but the overall latency, considering clock cycles and implementation frequency, provides a more comprehensive understanding of operation time. Both implementations use UCM65 technology, and a detailed latency comparison will be presented in subsection IV-C. Lastly, [19] outperforms us in clock cycle counts using a RISC-V architecture with SIMD instructions. However, trade-offs between clock cycles, area utilization, and synthesis frequency are considered in Table 8 and Table 9.

TABLE 8. Area ASIC synthesis comparison (UMC 65 µm).

Ref	Freq [MHz]	Cell Count	Cell Area Comb. [μm^2]	Cell Area Seq. [μm^2]
PULPino	79.66	36,173	78,676	92,304
[8]	45.47	57,413 [+59%]	143,198 [+82%]	102,273 [+11%]
[9]	79.74	63,119 [+74%]	148,838 [+89%]	130,779 [+42%]
X-HEEP	100	94,127	128,120	220,692
OURS	100	**138,296** [+47%]	**218,721** [+71%]	**267,125** [+21%]
X-HEEP	400	100,424	142,513	221,247
OURS	400	**146,451** [+46%]	**235,490** [+65%]	**267,154** [+21%]

2) CRYSTALS-DILITHIUM

The enhancements for Dilithium, as shown in Table 6, are limited because the study focused on Kyber acceleration. The versatile accelerators used for Kyber were applied to Dilithium to demonstrate ATHOS's flexibility. However, Dilithium only uses one memory-mapped accelerators (Keccak) and none of the new instructions. In contrast, [4] employs a hybrid approach with a Loosely coupled NTT accelerator and a Tightly coupled Keccak accelerator, significantly boosting performance. Compared to [11], which accelerates Dilithium's polynomial multiplication, our approach performs better in hashing and randomness generation for KeyGen and Verify, though not for Sign. Reference [17] provides different results based on the amount of function accelerated. `Custom-A` shows the cycle count with custom instructions for Keccak, while `Custom` includes both Keccak and NTT-based polynomial multiplication. Comparing to `Custom-A`, our results outperform [17] for Dilithium2, 3, and 5 in KeyGen, Sign, and Verify, achieving clock cycle reductions ranging from around 46% to over 67%.

C. ASIC RESULTS

Table 8 shows ASIC synthesis results for ATHOS, obtained with the CMOS UMC 65µm technology; it also compares two similar works adopting the same technology. Design Vision by Synopsys is employed for comprehensive analysis, elaboration, and compilation of the ATHOS design. Furthermore, the appropriate memories tailored to the UMC 65µm technology are meticulously selected and integrated where necessary.

In ATHOS, integrating both tightly and loosely coupled accelerators results in a 46% increase in cell count and a 65% increase in combinatorial cell area, along with a 21% increase in sequential cell area. This is a relatively predictable result, considering that as many as three loosely coupled accelerators are used. Most of the state-of-the-art works confirm this, favoring tightly coupled accelerations precisely when specific constraints on the final area are present. However, this is only sometimes true. In Table 8, comparing ATHOS and [8] shows differences, but not as substantial as one would expect from a comparison between an acceleration done entirely within the core pipeline and one performed instead partly with memory-mapped accelerators. In fact, despite only using tightly coupled accelerators, the relative increase in cell count and cell area combinatorial of [8] is higher than ATHOS's. It has a lower increase in sequential area since it has an additional register file within the core, which reduces the need for an external register file. However, relative increments are not so far from each other. Among the differences between the two acceleration methods (fully tightly and the hybrid method proposed in this paper), the one that stands out most from Table 8 is the frequency at which the circuit could be synthesized in ASIC. In the case of [8], the maximum clock frequency was reduced from 79.66 MHz to 45.47 MHz as an effect of the modified pipeline: in particular, the Modular Arithmetic Unit inserted in the processing core has a relatively long critical path, which limits the achievable frequency. With the same technology, the ATHOS architecture can be synthesized at the same frequency of the processor, without limiting its potential. The work in [19] adopts the 28n nm High Voltage Threshold (HVT) technology and achieves a frequency of 200 MHz. With this implementation, the percentage increase in area, compared to the baseline implementation, is equal to 389%, growing from approximately 30,000 to 149,100 equivalent gates. This area overhead (about eight times larger than ATHOS) does not balance the improvement in performance (which is more or less double than the one reported for ATHOS). This shows that a careful analysis of the operations to be accelerated and a careful choice of acceleration method can lead to better trade-offs between complexity and performance, as shown in the next Section.

D. LATENCY

In Table 9, we now compare the results obtained with recent works regarding latency. Latency is evaluated by multiplying Table 7 clock cycles and the period to which the different architectures are synthesized (as shown in Table 8). Time efficiency is defined as the ratio between the proposed solution's latency and the compared implementations' latency.

Time-efficiency values less than 1 indicate that the corresponding operation in the compared design is slower than the reference (i.e., OURS). Conversely, values greater than 1 would suggest that the design's operation is faster than the reference's. Although ATHOS, in terms of clock strokes, does not always outperform the compared architectures,

TABLE 9. Latency comparison.

Ref	Freq [MHz]	Alg	Latency [m sec]	Time-Efficiency
[8]	45.47	K-512	3.30/4.26/4.51	0.1/0.10/0.12
		K-768	6.02/7.16/7.50	0.1/0.1/0.12
		K-1024	7.69/8.93/9.34	0.13/0.13/0.15
[9]	79.74	K-512	1.46/2.21/2.33	0.23/0.19/0.24
		K-768	2.68/3.75/3.92	0.22/0.18/0.23
		K-1024	3.34/4.63/4.93	0.29/0.23/0.28
[19]	200	K-512	0.44/0.44/0.54	0.77/0.93/1.01
		K-768	0.82/0.83/0.985	0.73/0.81/0.90
		K-1024	0.975/0.99/1.22	1.025/1.09/1.11
OURS	**400**	K-512	0.34/0.41/0.55	
		K-768	0.60/0.68/0.89	
		K-1024	1/1.08/1.36	

we can see that considering the alternatives in their entirety, ATHOS achieves the shortest latency. Mostly all the time-efficiency values reported in Table 9 are lower than 1. It is important to note that the comparison is made with the same technology. This shows that the guided choice of accelerator type allows a higher clock frequency to be maintained, and thus lower latencies to be achieved, even with an increase in the number of cycles.

V. DISCUSSION AND CONCLUSION

The CV-X-IF interface represents a significant advancement in RISC-V acceleration. Our work leveraged the interface within the CV32E40X core, demonstrating its potential. While interest in this interface is growing, with a recent release candidate, our future work aims to delve deeper into its capabilities. We plan to optimize performance and implement safeguards against potential attacks. Additionally, expanding the range of accelerators and supporting new algorithms, such as code-based cryptography, is on our agenda.

In this study, ATHOS introduces novel insights into cryptographic acceleration on the RISC-V platform. Our hybrid acceleration approach strategically combines tightly and loosely coupled accelerators. Integration via the CV-X-IF interface offers versatility without necessitating core pipeline modifications. We prioritize integration methods based on operation complexity. Simple operations are accelerated with tightly accelerators, while complex tasks utilize external accelerators. This approach, facilitated by CV-X-IF, DMA, and inline assembly, simplifies accelerator integration. While our focus was on Kyber and Dilithium, our methodology can extend to various PQC algorithms and beyond.

REFERENCES

[1] National Institute of Standards and Technology. (2022). *PQC Candidates to be Standardized and Round 4*. [Online]. Available: https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4

[2] D.-T. Dam, T.-H. Tran, V.-P. Hoang, C.-K. Pham, and T.-T. Hoang, "A survey of post-quantum cryptography: Start of a new race," *Cryptography*, vol. 7, no. 3, p. 40, Aug. 2023.

[3] T. Fritzmann, U. Sharif, D. Müller-Gritschneder, C. Reinbrecht, U. Schlichtmann, and J. Sepulveda, "Towards reliable and secure post-quantum co-processors based on RISC-V," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1148–1153.

[4] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl, "Post-quantum signatures on RISC-V with hardware acceleration," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 2, pp. 1–23, Mar. 2024.

[5] A. Dolmeta, M. Mirigaldi, M. Martina, and G. Masera, "Implementation and integration of keccak accelerator on RISC-V for CRYSTALS-kyber," in *Proc. 20th ACM Int. Conf. Comput. Frontiers*. New York, NY, USA: Association for Computing Machinery, May 2023, pp. 381–382.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak," in *Advances in Cryptology—EUROCRYPT*, T. Johansson and P. Q. Nguye, Eds., Berlin, Germany: Springer, 2013, pp. 313–314.

[7] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS-kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Apr. 2018, pp. 353–367.

[8] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 239–280, Aug. 2020.

[9] T. Fritzmann, M. Van Beirendonck, D. B. Roy, P. Karl, T. Schamberger, I. Verbauwhede, and G. Sigl, "Masked accelerators and instruction set extensions for post-quantum cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, pp. 414–460, Nov. 2021.

[10] E. Alkim, "ISA extensions for finite field arithmetic—Accelerating kyber and NewHope on RISC-V," Cryptol. ePrint Arch., Tech. Rep., 049, 2020. [Online]. Available: https://eprint.iacr.org/2020/049

[11] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, and L. Fanucci, "A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms," *IEEE Access*, vol. 9, pp. 150798–150808, 2021.

[12] E. Karabulut and A. Aysu, "RANTT: A RISC-V architecture extension for the number theoretic transform," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, Aug. 2020, pp. 26–32.

[13] C. Gewehr, L. Luza, and F. G. Moraes, "Hardware acceleration of crystals-kyber in low-complexity embedded systems with RISC-V instruction set extensions," *IEEE Access*, vol. 12, pp. 94477–94495, 2024.

[14] S. D. Matteo, I. Sarno, and S. Saponara, "CRYPHTOR: A memory-unified NTT-based hardware accelerator for post-quantum CRYSTALS algorithms," *IEEE Access*, vol. 12, pp. 25501–25511, 2024.

[15] T. Stelzer, F. Oberhansl, J. Schupp, and P. Karl, "Enabling lattice-based post-quantum cryptography on the OpenTitan platform," in *Proc. Workshop Attacks Solutions Hardw. Secur.* New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 51–60.

[16] L. Li, G. Qin, Y. Yu, and W. Wang, "Compact instruction set extensions for kyber," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 3, pp. 756–760, Mar. 2024.

[17] L. Li, Q. Tian, G. Qin, S. Chen, and W. Wang, "Compact instruction set extensions for dilithium," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 2, pp. 1–21, Mar. 2024.

[18] L. Ducas, T. Lepoint, and V. Lyubashevsky, "CRYSTALS-dilithium: Digital signatures from module lattices," Cryptol. ePrint Arch., Tech. Rep., 633, 2017. [Online]. Available: https://eprint.iacr.org/2017/633

[19] 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 Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2024, no. 2, pp. 130–153, Mar. 2024.

[20] Y. Zhao, R. Xie, G. Xin, and J. Han, "A high-performance domain-specific processor with matrix extension of RISC-V for module-LWE applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2871–2884, Jul. 2022.

[21] 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, Nov. 2023.

[22] J. Lee, W. Kim, S. Kim, and J.-H. Kim, "Post-quantum cryptography coprocessor for RISC-V CPU core," in *Proc. Int. Conf. Electron., Inf., Commun. (ICEIC)*, Feb. 2022, pp. 1–2.

[23] K. Miteloudi, J. Bos, O. Bronchain, B. Fay, and J. Renes, "PQ.v.ALU.e: Post-quantum RISC-v custom ALU extensions on dilithium and kyber," Cryptol. ePrint Arch., Tech. Rep., 1505, 2023.

[24] Y. Zhao, H. Kuang, Y. Sun, Z. Yang, C. Chen, J. Meng, and J. Han, "Enhancing RISC-V vector extension for efficient application of post-quantum cryptography," in *Proc. IEEE 34th Int. Conf. Application-Specific Syst., Architectures Processors (ASAP)*, Jul. 2023, pp. 10–17.

[25] N. Gupta, A. Jati, and A. Chattopadhyay, "CRYSTALS-dilithium on RISC-V processor: Lightweight secure boot using post-quantum digital signature," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2023, pp. 1–7.
[26] *CV-X-IF eXtension Interface*. Accessed: Apr. 3, 2024. [Online]. Available: https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/en/latest/x_ext.html
[27] *AMBA AXI ACE Protocol Specification*, ARM, 2011.
[28] Y. Xi, Z. Zhang, Y. Wang, W. Wang, X. Han, W. Wang, and P. Liu, "A heterogeneous RISC-V SoC for confidential computing and password recovery," in *Proc. 7th Int. Conf. Integr. Circuits Microsystems (ICICM)*, Oct. 2022, pp. 500–504.
[29] A. Kamaleldin, S. Hesham, and D. Göhringer, "Towards a modular RISC-V based many-core architecture for FPGA accelerators," *IEEE Access*, vol. 8, pp. 148812–148826, 2020.
[30] Z. Ge, P. Xiao, M. Li, and H. Wang, "Soc design of intelligent recognition based on RISC-V," in *Proc. Int. Conf. Autom., Robot. Comput. Eng. (ICARCE)*, Dec. 2022, pp. 1–4.
[31] *SiFive TileLink Specification*, SiFive, Santa Clara, CA, USA, 2017.
[32] Z. Li, Y. Li, K. Wang, K. Ma, and S. Yu, "Model checking TileLink cache coherence protocols by murphi," in *Proc. IEEE 41st Int. Conf. Comput. Design (ICCD)*, Nov. 2023, pp. 30–37.
[33] T. Gomes, P. Sousa, M. Silva, M. Ekpanyapong, and S. Pinto, "FAC-V: An FPGA-based AES coprocessor for RISC-V," *J. Low Power Electron. Appl.*, vol. 12, no. 4, p. 50, Sep. 2022.
[34] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, P. Dabbelt, J. Hauser, and A. M. Izraelevitz, "The rocket chip generator," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep., UCB/EECS-2016-17, 2016.
[35] R. Avanzi and J. Bos. (2021). *CRYSTALS-Kyber Documentation*. [Online]. Available: https://pq-crystals.org/kyber/resources.shtml
[36] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. (2021). *CRYSTALS-Dilithium Documentation*. [Online]. Available: https://pq-crystals.org/dilithium/resources.shtml
[37] (2024). *PQ-CRYSTALS*. [Online]. Available: https://github.com/pq-crystals/
[38] 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, *arXiv:2401.05548*.
[39] OpenHW Group. (Apr. 7, 2024). *Open Bus Interface Protocol*. [Online]. Available: https://github.com/openhwgroup/obi
[40] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, p. 519, Apr. 1985.
[41] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 263, A. M. Odlyzko, Ed., Berlin, Germany: Springer-Verlag, 1987, pp. 311–323.
[42] *RISC-V Instruction Set Manual Volume I: User-Level ISA, Version 2.2*. Accessed: Mar. 15, 2024. [Online]. Available: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf
[43] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.
[44] W. M. Gentleman and G. Sande, "Fast Fourier transforms: For fun and profit," in *Proc. Amer. Fed. Inf. Process. Societies Joint Comput. Conf. (AFIPS)*, Nov. 1966, pp. 563–578.
[45] Z. Wang, Y. Yang, J. Wang, J. Hou, Y. Su, and C. Yang, "A scalable and efficient NTT/INTT architecture using group-based pairwise memory access and fast interstage reordering," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, early access, Oct. 9, 2024, doi: 10.1109/TVLSI.2024.3465010.
[46] K. Javeed and D. Gregg, "Efficient number theoretic transform architecture for CRYSTALS-kyber," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, early access, Sep. 20, 2024, doi: 10.1109/TCSII.2024.3465273.
[47] M. Li, J. Tian, X. Hu, and Z. Wang, "Reconfigurable and high-efficiency polynomial multiplication accelerator for CRYSTALS-kyber," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 8, pp. 2540–2551, Aug. 2023.
[48] A. Dolmeta, E. Valpreda, M. Martina, and G. Masera, "Implementation and integration of NTT/INTT accelerator on RISC-V for CRYSTALS-kyber," in *Proc. 21st ACM Int. Conf. Comput. Frontiers: Workshops Special Sessions*, New York, NY, USA, May 2024, pp. 59–62.
[49] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, and V. K. Ronny, "FIPS PUB 202—SHA-3 standard: Permutation-based hash and extendable-output functions," Tech. Rep., 2015.
[50] E. Alkim, "Cortex-M4 optimizations for RMLWE schemes," Cryptol. ePrint Arch., Tech. Rep., 012, 2020. [Online]. Available: https://eprint.iacr.org/2020/012
[51] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stölen, "pqm4: Testing and benchmarking NIST PQC on ARM cortex-M4," Tech. Rep., 2019.

ALESSANDRA DOLMETA (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Electronics and Telecommunications Department, Politecnico di Torino, under the supervision of Prof. Guido Masera and Maurizio Martina. Her research interests include the design of hardware architectures for post-quantum cryptography integrated into RISC-V, to speed up, and optimize PQC algorithms. She is an active member of the IEEE Student Branch of the Politecnico di Torino.

MAURIZIO MARTINA (Senior Member, IEEE) received the Dr.-Ing. and Ph.D. degrees in electronic engineering and electronic and communications engineering from the Politecnico di Torino, Italy, in 2000 and 2004, respectively. He has been a Professor with the Electronics and Telecommunications Department, Politecnico di Torino, since 2014. He has published more than 100 publications and holds two patents. His research interests include computer architecture and VLSI design of digital integrated circuits for image and video coding, forward error correction, cryptography, and artificial intelligence. He served as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS and as a Guest Editor of several special issues, including BioCAS 2017 Special Issue in IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS and ISCAS 2023 Special Issue in IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS. He has been part of the organizing and technical committee of several IEEE conferences, including BioCAS 2017, AICAS 2020, and PRIME 2023.

GUIDO MASERA (Senior Member, IEEE) received the Dr.-Ing. (summa cum laude) and Ph.D. degrees in electronic engineering from the Politecnico di Torino, Italy. He has been a Professor with the Electronics and Telecommunications Department, Politecnico di Torino, since 1992. His research interests include several aspects in the design of digital integrated circuits and systems, with a special emphasis on high-performance architectures for communications, forward error correction, image and video coding, cryptography, and hardware accelerators for machine learning. He has more than 200 publications, two patents, and was a designer of several ASIC components. He is an Associate Editor of *Electronics* (MDPI) and a Former Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS, and the *IET Circuits, Devices & Systems*.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement