An HLS-Based Hardware Accelerator for ML-KEM Polynomial Operations

Henrique Gregory Gimenez[1], **Edson Toshimi Midorikawa**[1], **Felipe Valência de Almeida**[1], **and Thales B. Paiva**[1]

[1] Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica
Universidade de São Paulo (USP)

{henriquegregory,emidorik,fvalencia,thales.paiva}@usp.br

Abstract. *The rapid advance of quantum computing poses a significant threat to classical cryptographic systems, motivating the NIST standardization of CRYSTALS-Kyber (ML-KEM). Due to the high computational cost of Kyber's polynomial operations, this paper proposes a High-Level Synthesis (HLS)-based hardware accelerator for Kyber-768, deployed on a PYNQ-Z2 SoC. The proposed architecture adopts a Load-Compute-Store paradigm alongside specific optimization directives to balance resource efficiency and latency. The results are evaluated by comparing the hardware performance to a software-only execution on an ARM Cortex-A9 and to existing state-of-the-art accelerators. The design demonstrates high power efficiency, with a total on-chip power consumption of 1.60W, and low resource utilization, consuming 91 DSP blocks. The highest performance gains occurred in complex vector operations, notably achieving speedups of 4.65× for the Number Theoretic Transform (NTT) and 4.86× for the inverse NTT compared to the software execution. This design suggests that HLS is a viable methodology for accelerating complex post-quantum cryptographic primitives.*

1. Introduction

The rapid advances in quantum computing represent a significant threat to modern public-key cryptographic schemes based on classical mathematical problems. The quantum algorithms introduced by Shor [Shor 1997] for factoring integers and solving discrete logarithms could render these schemes vulnerable once large-scale quantum computers are developed. To mitigate this threat, in 2016, the National Institute of Standards and Technology (NIST) launched a standardization process for Post-Quantum Cryptography (PQC) algorithms. In 2022, the key encapsulation mechanism (KEM) CRYSTALS-Kyber [Bos et al. 2018] was selected for standardization under the name ML-KEM [Alagic et al. 2024].

Kyber is based on the Module Learning With Errors (MLWE) problem [Regev 2005] and requires intensive polynomial and polynomial vector operations, leading to high computational costs. Accelerating these primitives with dedicated hardware like Field-Programmable Gate Arrays (FPGAs) ensures high efficiency and low power consumption in embedded systems. Because traditional register-transfer level (RTL) design is time-consuming and PQC standards are still evolving, HLS tools like Vitis HLS and Vivado offer a more productive alternative [Kastner et al. 2018]. These tools translate

C/C++ code into hardware description languages (Verilog/VHDL), simplifying FPGA deployment [Advanced Micro Devices, Inc. 2026].

This paper proposes the design and implementation of a hardware accelerator for polynomial operations in the Kyber reference implementation[1], utilizing High-Level Synthesis (HLS) tools. The design flow covers everything from the baseline C code to physical deployment on a PYNQ-Z2 board. Running a Linux OS with Jupyter Notebook support, the PYNQ-Z2 facilitates FPGA programming, testing, and metric collection. It also runs the standard Kyber software, allowing for an accurate hardware-software performance comparison. Through this approach, the study evaluates resource utilization and power, demonstrating the benchmark of the HLS accelerator against software-only execution.

2. Background

2.1. Kyber

CRYSTALS-Kyber relies on the algebraic structure of modules defined over the polynomial ring[2] $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, where the ring degree is $n = 256$ and the modulus is $q = 3329$ [Alagic et al. 2024]. The underlying module is denoted as R_q^k, meaning its elements are vectors of k polynomials. The dimension k determines the security level ($k = 2, 3,$ and 4 for NIST security levels 1, 3, and 5, respectively). Specifically, for Kyber-768, this dimension is set to $k = 3$.

Kyber consists of three main steps: key generation, encapsulation, and decapsulation. Key generation produces a public key pk and a secret key sk. The encapsulation algorithm takes pk as input to establish a shared secret ss and produce a ciphertext c. Finally, the decapsulation algorithm uses sk to recover the same ss from c.

Across all these stages, the polynomial vector multiplication constitutes an important computational workload. To execute these operations efficiently, Kyber relies heavily on the Number-Theoretic Transform (NTT) to perform polynomial multiplications, requiring intensive use of butterfly operations alongside constant Montgomery and Barrett modular reductions.

2.2. Reconfigurable Hardware

Field-Programmable Gate Arrays (FPGAs) are reconfigurable integrated circuits composed of dedicated primitives [Véstias et al. 2025]. The primary building blocks are Look-Up Tables (LUTs), which are used to implement the combinational logic of a given design. Some LUTs can also be configured as small, distributed memory elements, referred to as LUTRAMs. To handle sequential logic and store states or data between clock cycles, the architecture utilizes Flip-Flops (FFs). For larger data storage, FPGAs use dedicated memory modules known as Block RAMs (BRAMs). Computationally intensive mathematical operations, particularly multiplications and accumulations, are mapped to Digital Signal Processing (DSP) blocks. The synchronization of all these components is managed by a clock distribution network, which relies on Global Clock Buffers (BUFGs) to route the clock signal across the device.

[1] `https://github.com/pq-crystals/kyber/tree/main`

[2] A module generalizes a vector space by using a ring instead of a field for scalar multiplication. In Kyber, these ring elements are polynomials.

2.3. Related Works

[Yaman et al. 2021] introduced three architectures—lightweight, balanced, and high-performance—built upon a unified butterfly structure to accelerate Kyber key generation and encryption. Targeting high operating frequency and reduced resource utilization, [Zhang et al. 2023] proposed a ring butterfly core featuring dual-port support and partitioned memory access. Similarly, [Sun and Bai 2024] presented a configurable *radix*-4 NTT architecture employing *ping-pong* buffers to mitigate memory access conflicts. More recently, [Chen et al. 2025] showed that simultaneously optimizing Point-to-Point Multiplication (PWM) and using constant geometry memory access can reduce the accelerator product area-time.

Another line of research utilizes High-Level Synthesis (HLS) to increase project productivity without compromising security or performance. [Soni and Karri 2021] demonstrated the use of HLS to implement constant-time PQC primitives, ensuring protection against time-based side-channel attacks. In the context of high-volume data systems, [Carril et al. 2024] proposed a co-design hardware/software approach focused on batch processing to mitigate communication latency between CPU and FPGA.

Unlike approaches focused solely on RTL design, this work proposes an HLS-based accelerator optimized for polynomial vector operations on SoC platforms. The architecture adopts a *Load-Compute-Store* paradigm to separate external memory accesses and applies *array* partitioning and *pipelining* directives to balance resource efficiency and latency in embedded devices.

3. Proposed Accelerator Architecture

3.1. Selected Operations for Acceleration

In this work, ten computationally intensive operations critical to the Kyber key encapsulation process were identified and selected for optimization. To identify these operations, a performance profiling was conducted using the standard Kyber reference implementation's test suite, measuring the clock cycle consumption of each function. This empirical analysis aligns with the related by [Botros et al. 2019], which demonstrates that polynomial arithmetic constitutes the most significant computational bottleneck in the algorithm, excluding symmetric hashing. The description of each targeted operation is presented below.

1. Vector Operations on Polynomials
    - `POLYVEC_NTT`: applies the NTT in all elements of a vector.
    - `POLYVEC_INVNTT_TOMONT`: executes the inverse NTT on a vector, returning the coefficients to the normal domain.
    - `POLYVEC_BASEMUL_ACC_MONTGOMERY`: performs coefficient-by-coefficient multiplication of two vectors of polynomials that are already in the NTT domain, accumulating the result into a single polynomial and applying Montgomery's reduction.
    - `POLYVEC_ADD`: performs element-by-element addition between two vectors of polynomials without modular reduction.
    - `POLYVEC_REDUCE`: applies Barrett's reduction to all coefficients of the polynomial vector.

2. Operations on Simple Polynomials
    - `POLY_TOMONT`: converts all the coefficients of a polynomial from the normal domain to the Montgomery domain.
    - `POLY_INVNTT_TOMONT`: applies the inverse NTT in a single polynomial to return to normal domain and applying the Montgomery multiplication factor.
    - `POLY_ADD`: add the coefficients of two polynomials in parallel.
    - `POLY_SUB`: subtract the coefficients of two polynomials in parallel.
    - `POLY_REDUCE`: applies Barrett's reduction individually to the coefficients of a single polynomial.

3.2. Creating the top-level entity

Every HLS kernel requires a top-level entity that is subsequently synthesized into hardware. Having selected the operations to be accelerated, we encapsulate them into a single top-level module responsible for control and execution. This module receives the necessary inputs alongside a control signal that determines which specific operation the kernel will execute.

The top-level module, defined by the `polyvec_accel` function, acts as the communication interface between the Processing System (the ARM processor) and the Programmable Logic (PL) of the FPGA [TUL Corporation 2018]. The communication interfaces were configured at two distinct levels:

1. **Control Interface** (AXI4-Lite): responsible for defining the IP control signals (start, done, idle, ready) and handling the transfer of scalar parameters (operation selector) [ARM Limited 2016]. These signals are implemented using the AXI4-Lite bus that allows low-latency transfer for simple transactions, ideal to register mapping. The `#pragma HLS INTERFACE s_axilite` directive is used for these signals.
2. **Data Interface** (AXI4-Master): responsible for communication with the external RAM, where the data resides [ARM Limited 2016]. It is performed via the AXI4-Master bus. This protocol is typically used for large volumes of data, such as the individual polynomials or the vectors of polynomials in our case. To handle the data load, burst transfer parameters were configured, allowing up to 16 pending transactions. Additionally, pointers were separated into independent channels (`bundle=gmem0, gmem1, gmem2`), enabling simultaneous read and write operations. The `#pragma HLS INTERFACE m_axi` directive is used for these signals.

Individual polynomials consist of `KYBER_N` (256) 16-bit coefficients, totaling 512 bytes. Polynomial vectors contain `KYBER_N` (256) × `KYBER_K` (3) 16-bit coefficients, requiring the transfer of 1536 bytes. To mitigate data transfer latency, we propose an architecture based on the *Load-Compute-Store* paradigm to isolate external memory accesses. The processing flow follows a three sequential steps.

In the load step, the input data allocated in external memory is copied in its entirety to local memories in the FPGA. The Vitis HLS compiler automatically infers burst transfers from this function. In the case of a vector operation on the Kyber-768, this

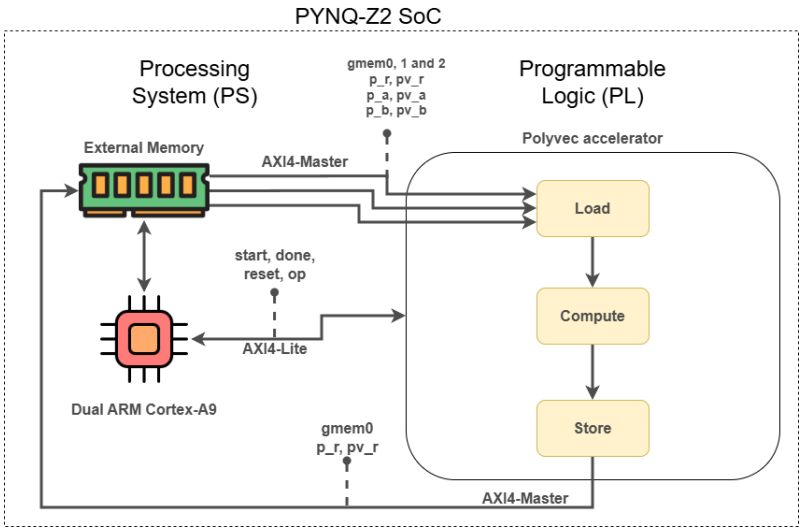

Figure 1. Architecture proposed for the accelerator

means pulling contiguous blocks of 1536 bytes directly into the Block RAMs (BRAMs) continuously and with high efficiency.

During the compute phase, the selected operation is executed on the local memories. Since the data resides in FPGA memory, it can be partitioned for parallel access, allowing the accelerator's datapath to be fed continuously without stalls caused by DDR memory latency.

Finally, after the calculations are completed, the result written to local memory is transferred back to the destination address in external memory. This step packages the results into a new write burst transaction.

Figure 1 illustrates the proposed accelerator architecture within the PYNQ-Z2 SoC environment, highlighting the physical decoupling between the Processing System (PS) and the Programmable Logic (PL). The figure highlights how the lightweight AXI4-Lite interface is used for control signals and scalar parameters, while larger data transfers are handled through AXI4-Master channels (`bundle=gmem0, gmem1, gmem2`), enabling dedicated pathways for simultaneous memory accesses.

4. HLS Optimization Strategy

This section explains the choice of optimizations used throughout the operations.

4.1. Array Partitioning

The Number-Theoretic Transform (NTT) and its inverse require simultaneous access to multiple coefficients during butterfly operations. In a standard implementation, vectors are mapped to block RAMs (BRAMs) with a limited number of read and write ports (typically two), creating an access bottleneck.

To solve this issue, we applied the `#pragma HLS ARRAY_PARTITION` directive to the input vectors of the NTT and inverse NTT functions, as well as to the higher-level functions `polyvec_ntt` and `polyvec_invntt_tomont`. The partitioning is performed cyclically with a factor of 16, meaning that vector elements are interleaved

across 16 distinct BRAM banks. This approach increases the internal memory bandwidth, allowing the datapath to read and write multiple coefficients per clock cycle, thereby supporting the high degree of parallelism required for NTT iterations.

Figure 2 illustrates how the cyclic array partitioning with a factor of 16 is applied to a vector.

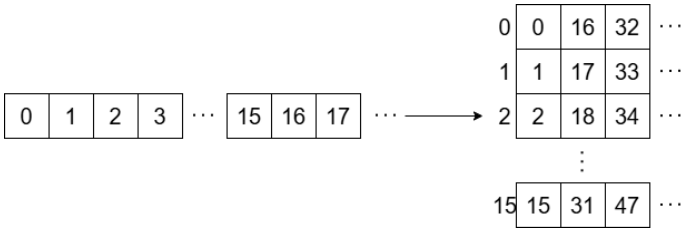

Figure 2. Cyclic Array Partitioning with a factor of 16

4.2. Pipelining

The pipelining technique was widely applied to the inner loops of the algorithms, including the NTT, inverse NTT, and vector addition and subtraction and other operations. This is implemented via the `#pragma HLS PIPELINE` directive. The *Initiation Interval* (II) is set to 1, which instructs the synthesis tool to generate a datapath in which a new loop iteration begins every clock cycle. This overlaps the execution of operations and prevents hardware from remaining idle while waiting for one operation to complete before the next begins.

4.3. Loop Unrolling

Loop unrolling was applied to `verify` and `cmov` functions. This technique replicates the physical logic of the loop body, creating multiple parallel processing paths. We can set the unroll factor that specifies to the synthesis tool how many data elements should be processed simultaneously in a single clock cycle. By unrolling the loop, the tool generates multiple instances of the combinational logic, reducing the number of sequential iterations and increasing throughput at the cost of additional hardware resources. This optimization is implemented by the directive `#pragma HLS UNROLL`.

4.4. Inlining Functions

Modularizing C code makes it easier to read, but in hardware it introduces overhead due to the creation of additional Finite State Machines (FSMs) to manage subroutine calls. To deal with this issue, we marked frequently called functions, such as modular arithmetic and basic polynomial operations, with the `#pragma HLS INLINE`. This directive instructs the compiler to ”flatten” the entire call hierarchy, inserting the logic circuit of the invoked function directly into the caller block. This approach not only saves control cycles but also allows resource sharing (such as DSP blocks) between operations.

5. Evaluation Methodology

To evaluate the proposed hardware accelerator, we established a methodology focusing on functional verification, performance benchmarking, and hardware cost analysis.

Table 1. Power Analysis from Implemented Netlist

Resource	Power (W)	Percentage
Total On-Chip Power	1.60	100%
Dynamic Power	1.46	91%
Device Static Power	0.14	9%
Dynamic Power Breakdown		
PS7	1.26	86%
Signals	0.05	4%
Clocks	0.05	4%
Logic	0.03	2%
DSP	0.03	2%
BRAM	0.03	2%

5.1. Experimental Environment

The proposed accelerator was deployed and evaluated on the PYNQ-Z2 board. This platform features a Zynq SoC that combines Programmable Logic (PL) operating at 100 MHz with a dual-core ARM Cortex-A9 processor running at 650 MHz under a Linux-based operating system. The evaluation environment was orchestrated using a Jupyter Notebook, which was used to program the FPGA, execute the reference software and collect the metrics. The original Kyber implementation in C was compiled via GCC to produce a shared library, providing a reliable software reference that could be called directly from Python.

5.2. Functional Verification

To guarantee that the HLS-based hardware accelerator faithfully reproduces the original algorithm, an automated validation test suite was implemented. For each of the supported mathematical operations, the evaluation script ran 10,000 iterations. During each iteration, random inputs were generated and processed simultaneously by the software reference (C implementation) and the hardware accelerator (FPGA). The outputs were compared using modular arithmetic (`diff % KYBER_Q`). The validation step required a zero-error rate across all 10,000 iterations to confirm functional equivalence.

5.3. Benchmarking and Metric Collection

To quantify the acceleration benefits, execution times and clock cycles were measured for both the software implementation (ARM processor) and the hardware execution across all 10,000 iterations. The total execution times were then used to calculate the performance speedup factor of the HLS-based accelerator relative to the baseline software implementation. Additionally, the obtained results were compared with other state-of-the-art approaches. The FPGA resource utilization, as well as the power dissipation, were analyzed based on the post-implementation reports generated by AMD Vivado.

6. Experimental Results

Tables 1 and 2 summarize the power dissipation per resource and the resource utilization of the accelerator.

The execution times, consumed clock cycles, and the resulting speedup factors for the implemented operations are detailed in Table 3. Reported cycles and times include the entire Load-Compute-Store flow, including data transfer.

Table 2. Hardware Resource Utilization for the Accelerator (Kyber-768)

Resource	Utilization	Available	Utilization
LUT	13984	53200	26.29%
LUTRAM	1509	17400	8.67%
FF	14274	106400	13.42%
BRAM	15.5	140	11.07%
DSP	91	220	41.36%
BUFG	1	32	3.13%

Table 3. Performance Comparison and Speedup: ARM Cortex-A9 (650 MHz) vs. HLS Accelerator (100 MHz)

Operation	SW Time (μs)	SW Cycles	HW Time (μs)	HW Cycles	Speedup
POLYVEC_NTT	1101.61	716047	236.93	23693	$4.65\times$
POLYVEC_INVNTT_TOMONT	1187.71	772008	244.35	24434	$4.86\times$
POLYVEC_BASEMUL_ACC_MONTGOMERY	427.78	278056	121.99	12199	$3.51\times$
POLYVEC_ADD	78.85	51250	121.92	12192	$0.65\times$
POLYVEC_REDUCE	192.28	124982	121.22	12122	$1.59\times$
POLY_TOMONT	91.46	59451	99.73	9972	$0.92\times$
POLY_INVNTT_TOMONT	424.18	275719	140.72	14071	$3.01\times$
POLY_ADD	59.38	38598	100.18	10017	$0.59\times$
POLY_SUB	60.34	39222	99.36	9935	$0.61\times$
POLY_REDUCE	93.90	61032	99.36	9936	$0.94\times$

Based on Table 1, the proposed design exhibits high power efficiency. The total on-chip power consumption is estimated at 1.60W, of which 91% (1.46W) corresponds to dynamic power. The Processing System accounts for the majority of the dynamic power dissipation, contributing 1.26W (86%), whereas the Programmable Logic (PL) represents a comparatively small fraction.

The resource utilization results presented in Table 2 indicate a well-balanced design with sufficient margin for future extensions. DSP blocks exhibit the highest utilization at 41.36%, which is consistent with the computational demands of polynomial multiplication and modular arithmetic. In contrast, LUTs (26.29%), Flip-Flops (13.42%), and BRAM (11.07%) show moderate utilization levels, demonstrating the effectiveness of the adopted array partitioning and pipelining strategies.

The performance evaluation in Table 3 highlights the trade-offs inherent to hardware acceleration in heterogeneous SoC architectures. Significant speedups are achieved for computationally intensive operations, reaching $4.86\times$ for POLYVEC_INVNTT_TOMONT and $4.65\times$ for POLYVEC_NTT. Similarly, operations involving Montgomery accumulation achieve a speedup of up to $3.51\times$. Despite operating at a frequency 6.5$\times$ lower, the hardware execution achieves significant speedups due to cycle-level parallelism, reducing total execution cycles.

On the other hand, lower-complexity operations such as POLY_ADD and POLY_SUB exhibit a significant speed-down in execution time (0.59$\times$ and 0.61$\times$, respectively). An analysis of the clock cycles further demonstrates that the hardware architecture itself remains highly efficient: POLY_ADD and POLY_SUB require only 10,017 and 9,935 cycles on the FPGA compared to 38,598 and 39,222 cycles on the software

Table 4. Comparison of FPGA Resources and Performance Against State-of-the-Art Accelerators

Work	Methodology	Operation	LUT	FF	DSP	Freq (MHz)	Cycles	Time (μs)
[Soni and Karri 2021]	HLS	Simple NTT	7,849	2,827	248	58	574	10.00
[Sun and Bai 2024]	RTL	Simple NTT	982	1,151	5	296	306	1.03
This Work	HLS	Vector NTT	13,984	14,274	91	100	23,693	236.93

CPU, respectively. This time penalty occurs because these operations were evaluated in isolation. In such a scenario, the latency associated with data transfer between external memory and on-chip BRAM via the AXI4 bus dictates the total cycle count. When operating at 100 MHz versus the CPU's 650 MHz, this communication bottleneck outweighs the hardware's parallel computation benefits. Therefore, these results expose a critical communication bottleneck. Accelerating these simple operations becomes highly justifiable and efficient only when they are integrated into a larger and continuous hardware execution pipeline. By keeping the intermediate data within the FPGA, redundant external memory transfers are reduced.

Table 4 compares our results with state-of-the-art solutions. It is worth noting that while [Soni and Karri 2021] and [Sun and Bai 2024] evaluate only isolated NTT core latency, our timing incorporates end-to-end AXI4-Master memory transfers to DDR. At the same time, the lack of power dissipation data in both reference works prevents a direct comparison of energy efficiency.

[Sun and Bai 2024] implements a RTL solution which achieves the lowest area consumption (982 LUTs, 1,151 FFs) due to its low-level optimization capabilities. Approaches based on High-Level Synthesis, such as this work and that of [Soni and Karri 2021], infer more logical area, but for distinct reasons. Two factors account for the higher LUT count in this work: the control logic generated for the top-level FSM, which manages ten distinct operations through a single AXI4-Lite-controlled entry point, and the combinational logic required by the 16-way cyclic array partitioning to enable parallel coefficient access. The FF count grows for a different reason: the use of II=1 pipelining across full vector-level operations. Since this work processes complete Kyber-768 vectors (768 coefficients) rather than a single 256-coefficient polynomial, as in the related works, maintaining a fully pipelined datapath requires proportionally more pipeline-stage registers to hold in-flight data. Function inlining compounds this effect, as flattening the call hierarchy into a single FSM requires the state of previously independent functions to be held simultaneously rather than sequentially reused.

For static cryptographic standards, this higher resource usage remains justified by the productivity gains of HLS, which abstracts RTL complexity and enables non-specialist designers to implement complex models and algorithmic optimizations with a favorable design-effort-to-performance ratio.

Loop unrolling explains the difference in DSP consumption and execution time relative to [Soni and Karri 2021]. This related work applies extensive loop unrolling to the NTT, replicating butterfly units to process multiple coefficients in parallel. This parallelism explains both their higher DSP allocation (248 DSPs) and their low cycle count (574 cycles for a single polynomial). In this work, loop unrolling is applied only to the `verify` and `cmov` functions, so the NTT and its associated operations remain sequen-

tial: although the II=1 pipelining allows a new iteration to start every clock cycle, at least 256 cycles are still required per polynomial.

Extrapolating [Soni and Karri 2021] cycle count to the k=3 polynomials of Kyber-768 (3 × 574 = 1,722 cycles) makes this gap clear: the 23,693 cycles measured for `POLYVEC_NTT` in this work are nearly 14× higher, a difference that cannot be explained by scope or data-transfer overhead alone, but by the sequential nature of the design. This trade-off is consistent with the resource utilization reported in Table 2: the sequential architecture requires fewer DSP blocks (91) at the cost of a substantially higher cycle count and longer execution time.

7. Conclusion

This paper presented an HLS-based hardware accelerator optimized for polynomial vector operations for the CRYSTALS-Kyber key-encapsulation mechanism. By adopting a Load-Compute-Store paradigm on a PYNQ-Z2 SoC, the architecture utilized optimization directives to balance resource efficiency and latency. All source code and supplementary materials required to reproduce the proposed architecture can be accessed at: `https://github.com/henriquegreg/hls-pqc-polynomial-guide/tree/main`.

Experimental results demonstrated the high power efficiency of the proposed design, with a total on-chip power consumption of 1.60W. The HLS approach proved highly effective for computationally intensive operations, achieving speedups of up to $4.65\times$ for the standard NTT and $4.86\times$ for the inverse NTT compared to a software implementation on an ARM Cortex-A9 processor. Furthermore, the use of optimizations resulted in an efficient allocation of hardware resources, consuming only 91 DSP blocks and outperforming other state-of-the-art HLS solutions in multiplier efficiency.

However, the performance evaluation highlighted the inherent trade-offs in heterogeneous SoC architectures. Lower-complexity mathematical tasks, such as simple polynomial additions, experienced a notable speed-down in absolute time (e.g., $0.59\times$), even though the FPGA consumed nearly four times fewer clock cycles than the CPU to complete the task. This behavior occurs because, during isolated kernel execution, AXI4 bus latency bottlenecks the 100 MHz FPGA, overshadowing its computational parallelism. This result demonstrates that to maximize the benefits of hardware acceleration for simpler tasks, they must be consolidated into a datapath within the programmable logic, mitigating off-chip communication overhead.

In summary, this work demonstrates that HLS is a viable methodology for accelerating post-quantum cryptography. By lowering the entry barrier, HLS democratizes hardware design, enabling non-specialist designers to efficiently balance engineering effort, performance, and power dissipation. Future work includes implementing full CRYSTALS-Kyber via HLS, integrating operations into a continuous pipeline to minimize data transfers, and exploring HW/SW co-design based on this unified flow.

References

Advanced Micro Devices, Inc. (2026). *Vitis High-Level Synthesis User Guide (UG1399)*. AMD, San Jose, CA. v2025.2.

Alagic, G., Dang, Q., Moody, D., Robinson, A., Silberg, H., and Smith-Tone, D. (2024). Module-lattice-based key-encapsulation mechanism standard.

ARM Limited (2016). *SoC Designer AXI4 Protocol Bundle User Guide*.

Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2018). CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE.

Botros, L., Kannwischer, M. J., and Schwabe, P. (2019). Memory-efficient high-speed implementation of Kyber on Cortex-M4. In *International Conference on Cryptology in Africa*, pages 209–228. Springer.

Carril, X., Kardaris, C., Ribes-González, J., Farràs, O., Hernandez, C., Kostalabros, V., González-Jiménez, J. U., and Moretó, M. (2024). Hardware acceleration for high-volume operations of CRYSTALS-Kyber and CRYSTALS-Dilithium. *ACM Transactions on Reconfigurable Technology and Systems*, 17(3).

Chen, H., Chen, H. W., Huang, S. H., and Chen, P. Y. (2025). A high-performance hardware design for polynomial multiplication in the CRYSTALS-Kyber algorithm. In *IEEE Symposium on Low-Power and High-Speed Chips and Systems, COOL CHIPS 2025 - Proceedings*. Institute of Electrical and Electronics Engineers Inc.

Kastner, R., Matai, J., and Neuendorffer, S. (2018). Parallel Programming for FPGAs. *ArXiv e-prints*.

Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA. Association for Computing Machinery.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509.

Soni, D. and Karri, R. (2021). Efficient hardware implementation of PQC primitives and PQC algorithms using high-level synthesis. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, volume 2021-July, pages 296–301. IEEE Computer Society.

Sun, J. and Bai, X. (2024). A high-speed hardware architecture of an NTT accelerator for CRYSTALS-Kyber. *Integrated Circuits and Systems*, 1:92–102.

TUL Corporation (2018). *PYNQ-Z2 Reference Manual*. TUL Corporation. v1.0.

Véstias, M., Flores, P., and Cláudio de Campos Neto, H. (2025). Síntese de alto nível em fpga.

Yaman, F., Mert, A. C., Öztürk, E., and Savaş, E. (2021). *A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-Kyber PQC Scheme*. IEEE.

Zhang, Z., Cui, Y., Ni, Z., Wang, C., and Liu, W. (2023). An efficient hardware accelerator of high-speed NTT for CRYSTALS-Kyber post-quantum cryptography. In *Conference Record - Asilomar Conference on Signals, Systems and Computers*, pages 1–6. IEEE Computer Society.