

Analysis, Design and Development of Advanced Verification Systems for Post-quantum Cryptography Implementations on FPGA

René Rodríguez Paredes

*Institute for Applied Microelectronics
(IUMA)*

*Universidad de Las Palmas de Gran
Canaria*

Agoney Jesús Moreno Rodriguez

*Arquimea Research Center
Arquimea*

Pedro P. Carballo, Ph.D.

*Institute for Applied Microelectronics
(IUMA)*

*Universidad de Las Palmas de Gran
Canaria*

Abstract—The transition to post-quantum cryptography requires hardware implementations whose functional correctness and interface compliance can be established efficiently. This paper presents a unified verification pipeline for FPGA cryptographic accelerators, using a Number Theoretic Transform (NTT) accelerator for ML-KEM (FIPS 203) as a case study. The strategy assigns four complementary verification approaches according to their respective strengths and design hierarchy: formal verification with Sby for arithmetic modules, functional verification with Cocotb, and shared Python reference models for general-purpose modules and integration, VUnit with OSVVM for verifying the AXI4-Stream and AXI4-Lite interfaces and hardware-in-the-loop (HIL) verification with the accelerator implemented on an Arty Z7-20 development platform to test the design in an environment close to production. A `Makefile` and a centralized YAML configuration (`project_config.yml`) provide simplified execution flows and light/full profiles intended for CI/CD integration. Six formal tasks, 13 Cocotb test files and 2 VUnit testbenches exposed an input range limitation in the modular reduction, an ignored validation signal, and an AXI4-Stream TLAST protocol violation. Aggregated Cocotb/VUnit code coverage reached 92.75% per design unit and 80.71% per instance. Formal, Cocotb, VUnit+OSVVM, and HIL stages took 22:14, 5:29, 2:50 and 8:06 (min:s), respectively, distributing Cocotb across open-source and commercial simulators reduced its time to 2:46. The results show a unified verification pipeline that reduces duplicated verification effort while easing the verification tasks and technique orchestration complexities.

Index Terms—post-quantum cryptography, ML-KEM, NTT, FPGA, functional verification, formal verification, Cocotb, hardware-in-the-loop

I. INTRODUCTION

The growing demand for advanced computing in data centers, embedded systems, next-generation communications and artificial intelligence is producing increasingly complex digital systems [1]. Under supply-chain uncertainty and constrained ASIC manufacturing capacity [2], [3], FPGAs provide design flexibility, rapid prototyping and lower development cost. Their increasing complexity, however, expands the behaviors that must be checked before production and the risk of non-trivial defects escaping verification [1]. The Intel Pentium FDIV bug, whose recall cost was estimated at \$475 million [4], [5], illustrates the potential impact of an undetected hardware error reaching production.

No single verification technique covers all types of designs and potential risks. Simulation offers control and visibility but generally cannot provide exhaustive guarantees, emulation and prototyping exchange internal observability for execution speed, and formal verification explores state spaces exhaustively with a solver engine, but it is not apt for modules with high latency or complex arithmetic [6].

Verification is especially relevant in critical systems, where potential failures can negatively impact large swaths of society. The internet, instant messaging, financial transactions, and all kinds of information exchange in general are secured by cryptographic algorithms.

For a cryptographic algorithm to provide the guarantees a given application requires (confidentiality, integrity, authentication, and non-repudiation), a correct implementation is not sufficient [7]. The algorithm itself must be appropriate, that is, not vulnerable to the threat model under consideration. Modern cryptography rests on the notion of semantic security, under which confidentiality holds because extracting information about a message from its ciphertext is computationally infeasible [8], [9]. Because that infeasibility is defined relative to the adversary's computational power, the threat model must widen once adversaries with new computational models are considered, such as quantum computing.

Quantum algorithms such as Shor's [10] threaten current public-key schemes and motivate the transition toward quantum-resistant primitives [11]. The urgency is heightened by *Harvest Now, Decrypt Later* (HNDL) attacks, in which encrypted data is stored until sufficiently capable quantum computers become available [12]. Post-quantum cryptography is the response to the “quantum threat”, which consists of devising cryptographic algorithms that are resistant to attacks from both classical and quantum computers. As part of the global effort to transition to post-quantum cryptography, the National Institute of Standards and Technology (NIST) initiated a process [13] to develop appropriate standards. The process has resulted in the standardization of 3 algorithms: ML-KEM (FIPS 203), ML-DSA (FIPS 204), and SLH-DSA (FIPS 205). With the standards in place the challenge of correctly developing implementations remains.

The two main contributions of this work are:

- A reproducible pipeline that shares reference models, centralizes configuration and supports fast and complete execution profiles.
- An evaluation on the ML-KEM NTT accelerator that was provided, reporting defects, integration, observations, coverage, and execution time.

The work addresses functional correctness and integration of the accelerator, it does not claim coverage of physical attacks such as side-channel analysis which could be part of a broader verification effort. The remainder of this paper presents how each verification approach was selected, the design that was provided, the pipeline and results and conclusions.

II. VERIFICATION FRAMEWORK AND TECHNIQUE SELECTION

FPGA-oriented verification can be organized along six axes: approach, verification domain, granularity level, stimulus generation, development stage, and type of evidence. This view makes explicit that each technique contributes complementary evidence rather than acting as interchangeable alternatives.

UVM [14], UVVM [15], OSVVM [16], VUnit [17], and Cocotb [18] were evaluated according to the following criteria: learning curve, capabilities, reuse, productivity, flexibility, and applicability for cryptographic designs. With uniform weighting on a 1-to-5 scale, Cocotb obtained 25/30, VUnit 23/30, UVVM, OSVVM 20/30, and UVM 17/30. The purpose of these scores is to guide the selection of a framework in this project rather than being an absolute ranking of each one. Cocotb was chosen for general functional verification, because Python simplifies the integration and reuse of complex cryptographic reference models. VUnit was selected to provide a low-friction approach for integrating verification into existing designs, while OSVVM was chosen for standard interface verification because it provides pre-verified Verification Components (VCs). Sby from the YosysHQ OSS CAD Suite [19], [20] was selected for formal verification of compact arithmetic blocks. Finally, Hardware-In-The-Loop (HIL) verification complements these techniques by exercising the implemented system on its target FPGA platform.

III. CASE STUDY: AN NTT ACCELERATOR FOR ML-KEM

ML-KEM [21], [22] is a key encapsulation mechanism (KEM) based on the Module-LWE problem over the ring $Z_q[X]/(X^{256} + 1)$, with $q = 3329$. Its key generation, encapsulation, and decapsulation operations concentrate their cost in SHA3-based randomness generation, sampling and compression, and polynomial arithmetic. Direct polynomial multiplication costs $O(n^2)$, whereas the NTT reduces it to $O(n \log n)$ by moving polynomials to a representation in which multiplication is coefficient-wise. Because Z_q contains a primitive 256th root of unity but not a 512th root, the transform is evaluated in seven stages of 128 butterfly operations [23]. The NTT is therefore a central optimization target in ML-KEM hardware [24], [25], with subtle error classes such as modular-reduction corner cases, coefficient reordering, and

incorrect twiddle-factor indexing. For these reasons, the ML-KEM NTT accelerator was selected as the case study.

The design under verification is based on the lightweight accelerator by Yaman *et al.* [26]. Its Verilog core contains modular arithmetic, a unified butterfly, an address generator, four dual-port BRAMs, a twiddle-factor ROM and control logic. The butterfly supports Cooley-Tukey forward NTT and Gentleman-Sande inverse NTT configurations while sharing one DSP block. Inverse scaling is distributed across the stages through modular divisions by two [27]. A VHDL wrapper supplied by Arquimea Research Center exposes NTT and inverse NTT operations through AXI4-Stream data interfaces and an AXI4-Lite control interface. This mixed-language hierarchy, Fig. 1, combines arithmetic, control, memory, and standardized interfaces, making it representative of the verification challenges considered in this work.

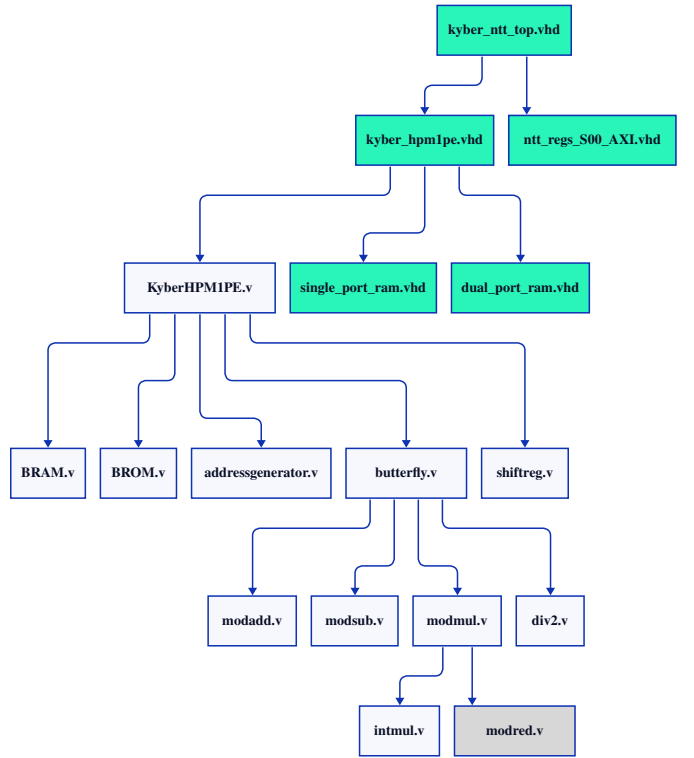


Fig. 1. Design hierarchy. Modules highlighted in green were developed by Arquimea Research Center. The module highlighted in gray has other submodules.

IV. VERIFICATION PIPELINE

The verification flow shown in Fig. 2 starts with an analysis of the design hierarchy and the nature of each design area, then assigns a technique to each area, and finally orchestrates the execution of the tests and generates the output artifacts (coverage, PASS/FAIL messages and traces).

A. Formal Verification of Modular Arithmetic

Six Sby verification tasks were used to check the integer multiplier and the modular operations (addition, subtraction, multiplication, reduction, and division-by-two) against reference models expressed with SystemVerilog Assertions. Inputs

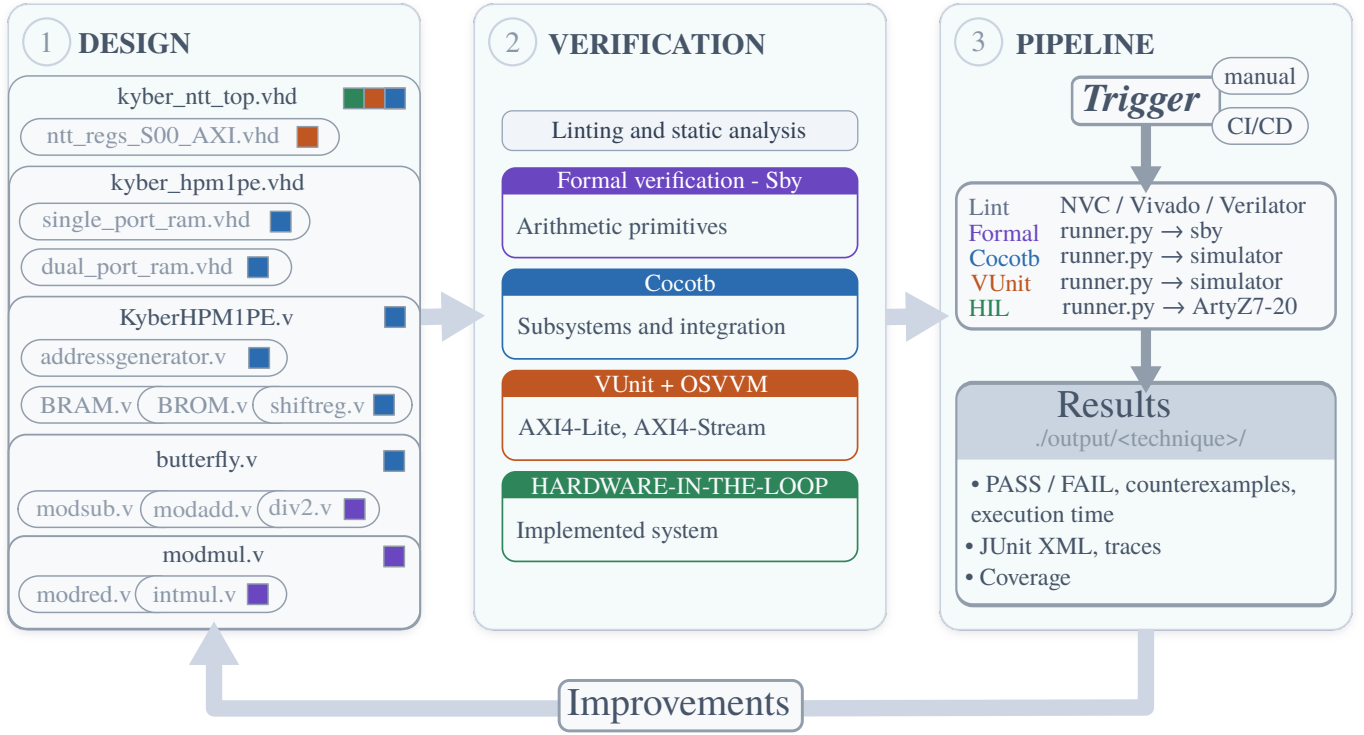


Fig. 2. Design-oriented verification flow and unified pipeline.

remain symbolic, as the solver explores their domains exhaustively. For example, all 2^{24} combinations of the 12-bit operands of `intmul.v` are covered. This provides stronger evidence than a simulation sweep for arithmetic blocks whose inputs may not be fully constrained when reused.

B. Functional Verification with Cocotb

Thirteen Cocotb test files cover blocks with complex references, subsystems, and integration (top-level module). The three levels reuse Python reference models, in particular the `ntt_model.py` reproduces forward and inverse transforms, including the distributed inverse scaling implemented by the accelerator.

C. AXI Verification with VUnit and OSVVM

Two VUnit testbenches verify the AXI4-Lite register bank and the AXI4-Stream input/output path using OSVVM VCs. The testbenches check register access, reset behavior and backpressure, including a directed stall on the last output sample. The testbenches deliberately do not check the correctness of the NTT/INTT operation, this represents the different roles that VUnit with OSVVM and Cocotb have.

D. Hardware-in-the-Loop Verification

The HIL flow targets an Arty Z7-20 development platform and comprises three components. A Vivado block design connects the processing system and DDR memory to the accelerator through an AXI DMA, a bare-metal application configures the AXI4-Lite register bank and performs interrupt-driven DMA transfers, and a Python host script opens the serial port for communication. The application handles cache coherency and transfer timeouts, while the host checks the

operation identifier, result length and every returned coefficient against the shared `ntt_model.py` reference.

A single `make hil` command recreates the Vitis project from the Vivado-exported `.xsa`, builds and loads the bare-metal application, programs the bitstream, and starts the host regression. This stage exercises the implemented accelerator together with the processing system, DMA, memory hierarchy, software, and physical interfaces under the tested operating conditions.

E. Orchestration

The `project_config.yml` file, a Makefile and the dedicated Python runners unify the four techniques and simplify the execution the different tasks. The YAML file is the single source of truth for parameters, HDL sources, tests, simulators, and execution profiles (`ci-light` and `ci-full`). The `ci-light` profile selects a fast, license-conscious subset intended for each commit, whereas `ci-full` executes the entire suite, including HIL, for less frequent runs. The pipeline is designed for CI/CD integration [28]. Deployment on a Git hosting and runner infrastructure remains outside the scope of this work. Generated traces, reports, and coverage databases are collected under `output/`.

V. RESULTS

All six formal tasks completed successfully within their stated assumptions, and the simulation stages produced the expected outputs. The full HIL regression applied 1,000 deterministic pseudorandom vectors of 256 coefficients to both the NTT and inverse NTT modes. All 2,000 hardware executions (NTT then INTT) returned the expected operation, result

length and matched `ntt_model.py` coefficient by coefficient. More importantly, the layered flow exposed behaviors that had not been documented despite prior testing by the core developers and the collaborating company:

- **Bounded modular reduction.** Although `modred.v` accepts a 24-bit input, it does not produce correct results over the entire input range. The first ascending counterexample occurs at 15,056,938, and a subsequent Cocotb sweep identified approximately 500 incorrect reductions. Under its intended operating conditions, accelerator functionality is unaffected because the input is the product of two canonical coefficients and is bounded by $(3329 - 1)^2 = 11075584$. Nevertheless, this limitation restricts the safe reuse of the module and should be documented or enforced through an explicit input constraint.
- **Ignored validation signal.** There is an input signal, `din_valid`, that based on its name and other commented code appears to have been intended to generate internal load signals depending on the operation. As the core is currently implemented it serves no functional purpose: a complete polynomial is loaded and transformed even when this signal remains low. This is one of the more relevant examples of “dead code” in the design.
- **AXI4-Stream output violation.** Under backpressure on the final output coefficient, `m_axis_tlast` is generated combinationaly using `TREADY`. It consequently fails to remain stable while `TVALID` is asserted and the receiver is not ready, contrary to the AXI4-Stream requirement [29].
- **Fixed-length input framing.** HIL confirmed that the omitted input `TLAST` signal, which is optional in AXI4-Stream, does not prevent correct operation when the bare-metal application treats exactly 256 transfers as one polynomial frame. This is an integration constraint rather than a protocol violation.

TABLE I
MEASURED EXECUTION TIME OF THE FULL VERIFICATION STAGES.

Stage	Tests	Time (min:s)
Formal (Sby)	6 tasks	22:14
Cocotb, Questa	13 files	5:29
Cocotb, mixed simulators	13 files	2:46
VUnit/OSVVM	2 benches	2:50
HIL, Arty Z7-20	1,000 vectors	8:06

Regarding the verification pipeline execution time Table I shows that formal verification is the most time-consuming stage. The 8:06 HIL measurement includes Vitis workspace recreation, application build, FPGA programming and all 2,000 UART-driven hardware cases, Fig. 3. Assigning compatible Cocotb tests to NVC [30] and Verilator [31] while retaining Questa Sim [32] for mixed-language cases reduces the Cocotb stage from 5:29 to 2:46, nearly halving its duration and reducing demand for commercial licenses. The UCDB databases generated by Questa Sim for the Cocotb and VUnit tests were merged to obtain aggregate code coverage: 92.75%

per design unit and 80.71% per instance. The lower per-instance result is due to the distributed nature of the test suite.

```

RUN NTT
KYBER REGO = 0x00000000
DMA transfers started
TX DMA done
RX DMA done
RESULT NTT 238 961 525 2698 3146 1001 849 1843 2238...
DONE

READY format: NTT|INTT c0 c1 ... c255
PASS NTT vector=999
case 2000/2000: INTT pattern=random seed=1 vector=999

RUN INTT
KYBER REGO = 0x00000001
DMA transfers started
TX DMA done
RX DMA done
RESULT INTT 2494 916 2616 2144 1778 2014 1842 912...
DONE

READY format: NTT|INTT c0 c1 ... c255
PASS INTT vector=999
PASS 2000/2000 cases

```

Fig. 3. Results of the HIL verification.

VI. CONCLUSIONS AND FUTURE WORK

This work presented a unified verification pipeline for FPGA-based cryptographic accelerators that combines complementary verification techniques into a reproducible and adaptable workflow. The ML-KEM NTT accelerator case study demonstrates how assigning each verification approach according to its strengths improves verification effectiveness while reducing duplicated verification effort.

Formal verification supplied exhaustive arithmetic evidence and identified the modular-reduction reuse limit. Cocotb made complex reference models practical and reusable across hierarchy levels and HIL. VUnit with OSVVM exposed the AXI4-Stream violation under directed backpressure, and HIL closed the integration loop on the implemented platform with 1,000/1,000 passing vectors while establishing the fixed-length input framing constraint. Together, the techniques revealed arithmetic assumptions, unused control behavior and interface non-compliance that a single end-to-end test strategy had not exposed.

A centralized configuration file and the implementation of regular interfaces for all the verification approaches turn a collection of tools into a reproducible pipeline while reducing duplicated verification effort via common reference models and correctly assigning verification approaches. The pipeline supports separating frequent, license-dependent verification tasks from complete regressions, although a deployed CI/CD service remains future work. Likewise, the reported functional and code-coverage evidence applies to the tested properties and conditions.

Future work will generalize the directory and runner interfaces, add simulators, linters, toolchains, and deploy the pipeline on an on-premises GitLab service using containers and/or virtual machines. Other priorities are requirements-

to-test traceability, functional coverage, broader support for cryptographic designs and deeper formal analysis, including mutation-based assessment with MCY and proofs spanning more cycles.

REFERENCES

- [1] Harry D. Foster, “2024 Wilson Research Group FPGA Functional Verification Trend Report,” 2024, [Online]. Available: <https://verificationacademy.com/topics/planning-measurement-and-analysis/wrg-industry-data-and-trends/2024-siemens-eda-and-wilson-research-group-asic-functional-verification-trend-report/>
- [2] Christian de Looper, “TSMC Has Hit Maximum Capacity.” Accessed: Apr. 12, 2026. [Online]. Available: <https://fcrtech.com/semiconductor-news/tsmc-maximum-capacity/>
- [3] W.-Y. Lee, “Broadcom Flags Supply Constraints, Says TSMC Capacity a Bottleneck,” *Reuters*, Mar. 2026, Accessed: Apr. 12, 2026. [Online]. Available: <https://www.reuters.com/world/asia-pacific/broadcom-flags-supply-constraints-says-tsmc-capacity-bottleneck-2026-03-24/>
- [4] A. Edelman, “The Mathematics of the Pentium Division Bug,” *SIAM Review*, vol. 39, no. 1, pp. 54–67, Jan. 1997, doi: 10.1137/S0036144595293959.
- [5] “Intel Historical Annual Reports.” Accessed: Jul. 01, 2026. [Online]. Available: <https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2025-05/history-1994-annual-report.pdf>
- [6] T. Grimm, D. Lettner, and M. Hübner, “A Survey on Formal Verification Techniques for Safety-Critical Systems-on-Chip,” *Electronics*, vol. 7, no. 6, p. 81, Jun. 2018, doi: 10.3390/electronics7060081.
- [7] E. B. Barker, “Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms,” NIST SP 800–175B, Aug. 2016. doi: 10.6028/NIST.SP.800-175B.
- [8] D. Boneh and V. Shoup, “2.2 Computational Ciphers and Semantic Security,” in *A Graduate Course in Applied Cryptography*, 2023. Accessed: Apr. 19, 2026. [Online]. Available: <https://toc.cryptobook.us/>
- [9] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, Third edition. in Chapman & Hall/CRC Cryptography and Network Security Series. Boca Raton London New York: CRC Press, Taylor & Francis Group, 2021.
- [10] P. W. Shor, “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997, doi: 10.1137/S0097539795293172.
- [11] Bas Westerbaan, “Cloudflare Targets 2029 for Full Post-Quantum Security.” Accessed: Apr. 12, 2026. [Online]. Available: <https://blog.cloudflare.com/post-quantum-roadmap/>
- [12] J. Mascelli and M. Rodden, “Harvest Now Decrypt Later”: Examining Post-Quantum Cryptography and the Data Privacy Risks for Distributed Ledger Networks,” *Finance and Economics Discussion Series*, no. 2025–93, Sep. 2025, doi: 10.17016/FEDS.2025.093.
- [13] I. T. L. Computer Security Division, “Post-Quantum Cryptography | CSRC | CSRC.” Accessed: Apr. 12, 2026. [Online]. Available: <https://csrc.nist.gov/projects/post-quantum-cryptography>
- [14] “IEEE Standard for Universal Verification Methodology Language Reference Manual,” *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017)*, pp. 1–458, Sep. 2020, doi: 10.1109/IEEESTD.2020.9195920.
- [15] “Introduction — UVVM Documentation.” Accessed: Jun. 20, 2026. [Online]. Available: https://uvvm.github.io/uvvm_intro.html
- [16] “Open Source VHDL Verification Methodology – VHDL Made Better!” Accessed: Jun. 20, 2026. [Online]. Available: <https://osvvm.org/>
- [17] “VUnit Documentation.” Accessed: Jun. 24, 2026. [Online]. Available: <https://vunit.github.io/>
- [18] “Cocotb.” Accessed: Jun. 11, 2026. [Online]. Available: <https://www.cocotb.org/>
- [19] “YosysHQ/Oss-Cad-Suite-Build.” Accessed: Jun. 26, 2026. [Online]. Available: <https://github.com/YosysHQ/oss-cad-suite-build>
- [20] “YosysHQ Tool Documentation - YosysHQ Docs Documentation.” Accessed: Jun. 26, 2026. [Online]. Available: <https://yosyshq.readthedocs.io/en/latest/tools.html>
- [21] National Institute of Standards and Technology, “Module-Lattice-Based Key-Encapsulation Mechanism Standard,” Gaithersburg, MD, Federal Information Processing Standards Publication NIST FIPS 203, Aug. 2024. doi: 10.6028/NIST.FIPS.203.
- [22] R. Avanzi *et al.*, “Algorithm Specifications And Supporting Documentation.”
- [23] V. Lyubashevsky and G. Seiler, “NTTRU: Truly Fast NTRU Using NTT.” Accessed: Jul. 08, 2026. [Online]. Available: <https://eprint.iacr.org/2019/040>
- [24] T.-H. Nguyen *et al.*, “An Area-Time Efficient Hardware Architecture for ML-KEM Post-Quantum Cryptography Standard,” *IEEE Access*, vol. 13, pp. 103834–103847, 2025, doi: 10.1109/ACCESS.2025.3579379.
- [25] T.-H. Nguyen, D.-T. Dam, P.-P. Duong, B. Kieu-Do-Nguyen, C.-K. Pham, and T.-T. Hoang, “Efficient Hardware Implementation of the Lightweight CRYSTALS-Kyber,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 2, pp. 610–622, Feb. 2025, doi: 10.1109/TCSI.2024.3443238.
- [26] F. Yaman, A. C. Mert, E. Öztürk, and E. Savaş, “A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQ Scheme.” Accessed: Jul. 08, 2026. [Online]. Available: <https://eprint.iacr.org/2021/485>
- [27] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, “Highly Efficient Architecture of NewHope-NIST on FPGA Using Low-Complexity NTT/INTT,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 49–72, Mar. 2020, doi: 10.13154/tches.v2020.i2.49-72.
- [28] “Git.” Accessed: Jul. 09, 2026. [Online]. Available: <https://git-scm.com/>
- [29] “AMBA AXI-Stream Protocol Specification.” Accessed: Jul. 16, 2026. [Online]. Available: <https://developer.arm.com/documentation/ih0051/b/>
- [30] N. Gasson, “Nickg/Nvc.” Accessed: Jul. 10, 2026. [Online]. Available: <https://github.com/nickg/nvc>
- [31] “Veripool.” Accessed: Jul. 10, 2026. [Online]. Available: <https://www.veripool.org/verilator/>
- [32] “Questa One Sim.” Accessed: Jul. 10, 2026. [Online]. Available: <https://www.siemens.com/en-us/products/ic/questa-one/simulation/questa-one-sim/>