IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–26. DOI:XXXXXXXX

A Crypto-Agile Mixed-Radix NTT Accelerator for Negacyclic and Trinomial Rings

JaeSeok Lee, DongHyun Shin, YoungBeom Kim and Seog Chung Seo*

Kookmin University, Seoul, Republic of Korea
{dlwotjr11101,dkldkl13,darania,scseo}@kookmin.ac.kr

Abstract. Hardware accelerators for lattice-based post-quantum cryptography (PQC) have largely converged on the radix-2 number-theoretic transform (NTT). The spectrum of standards that crypto-agility must accommodate, however, extends beyond NIST. In response to advances in cryptanalysis and to reduce the attack surface, national authorities are standardizing or recommending lattice-based algorithms with alternative algebraic structures, as well as schemes built on entirely different primitives. For instance, NTRU+, selected in the Korean PQC competition and currently undergoing standardization, is defined over a trinomial ring whose degree is divisible by three and thus requires a mixed-radix NTT comprising both radix-2 and radix-3 layers, whereas SMAUG-T employs an NTT-unfriendly modulus. Support for heterogeneous algebraic structures is therefore no longer optional but a prerequisite for crypto-agile design; existing radix-2-centric accelerators, however, fail to meet this requirement. In this paper, we present a crypto-agile mixed-radix NTT accelerator that performs transforms over both negacyclic and trinomial rings on a single datapath. Our contributions are threefold. First, we design a unified butterfly unit that integrates two parallel radix-2 butterflies, one radix-3 butterfly, and the trinomial transform on shared arithmetic resources, in which a generic 17-bit Montgomery path and an ML-DSA-specific 23-bit reduction path share the same two DSPs. Second, we propose a conflict-free memory mapping that guarantees in-place parallel access across all mixed-radix layers with a single 1-bit bank-selection function, namely address parity for radix-2 and a fixed address bit for radix-3, and requires neither mapping tables, nor coefficient reordering, nor additional BRAMs. Third, we introduce a generalized modular constant division that combines the multiple alignment of Montgomery reduction with the quotient estimation of Barrett reduction to perform the 3^{-1} scaling of the radix-3 inverse NTT without any DSP. Implemented on an Artix-7 FPGA, the proposed core supports all security levels of ML-KEM, ML-DSA, HAETAE, SMAUG-T, and NTRU+ using 5,279 LUTs, 4 DSPs, and 9 BRAMs at 141.5 MHz. For NTRU+, it achieves a 4.1–13.0× improvement in area-time product (ATP) over approaches that resort to radix-2-only cores via degree lifting, and new moduli of up to 17 bits can be integrated at the cost of fewer than 20 LUTs.

Keywords: Post-quantum cryptography · NTT · Crypto-agility · Lattice-based Cryptography · Radix-3 · Conflict-free memory mapping · FPGA

1 Introduction

To counter the threat of quantum computing, the transition to post-quantum cryptography (PQC) is well underway. In 2024, NIST standardized the lattice-based ML-KEM [Nat24b]

*Corresponding author

Licensed under Creative Commons License CC-BY 4.0.

and ML-DSA [Nat24a] together with the hash-based SLH-DSA [Nat24c], while standardization of the lattice-based Falcon [FHK+20] and the code-based HQC [GAA+26] is still in progress. NIST is furthermore running the Additional Digital Signature competition to mitigate the bias toward lattice-based schemes. Some submissions not selected by NIST have been adopted by other government bodies, and several countries are independently developing new PQC algorithms. For example, the German BSI recommends FrodoKEM and Classic McEliece, and Korea has selected HAETAE [CCD+24] and AIMer [CHK+26] for digital signatures and SMAUG-T [CCC+26] and NTRU+ [PK26] for key encapsulation mechanisms (KEMs). As of July 2026, China is also conducting its own competition. Compared with RSA and elliptic-curve cryptography, PQC algorithms remain relatively immature in terms of both security analysis and implementation. Multiple standards therefore coexist as a hedge against potential vulnerabilities uncovered by advancing cryptanalysis and to ensure long-term security, which has naturally raised the demand for crypto-agility: deployed systems must be able to switch between, and operate alongside, multiple algorithms. Hardware research has likewise expanded from optimizing the performance of a single algorithm toward flexibly sharing the common operations of multiple schemes on limited resources.

Related Work. Within the lattice-based setting, accelerator designs supporting ML-KEM and ML-DSA dominate the literature; representative examples include the Kali system [AMI+23] and the implementation of Wang et al. [WZZ+24]. More recently, several works have attempted to share resources across different primitives. Examples include HORCRUX [DPM+25] and PHOENIX [RLC+26]; in particular, PHOENIX accelerates the lattice-based ML-KEM and the code-based HQC on a single coprocessor by merging the butterfly structures common to the NTT and the Frobenius additive FFT (FAFFT) into a single SuperButterfly circuit. These agility-oriented designs, however, still build on radix-2 arithmetic optimized for power-of-two degrees, and therefore cannot accommodate schemes with fundamentally different algebraic structures. Meanwhile, cryptanalytic research has introduced new attack models against lattice-based cryptography and exposed the limits of existing security assumptions, prompting continued efforts to reduce the attack surface through structural modifications. In this context, NTRU [CDH+20], NTRU+ [PK26], and NCC-Sign [SKA22] adopt algebraic structures that depart from the existing standards, yet hardware research targeting these schemes remains scarce. NTRU is defined over an NTT-unfriendly non-cyclotomic ring, while NTRU+ and NCC-Sign, although defined over NTT-friendly rings, have polynomial degrees that are not powers of two and thus require a multi-stage decomposition mixing radix-2 and radix-3 layers after an initial polynomial split. A few hardware attempts have been reported, either introducing heterogeneous datapaths or reusing existing radix-2 structures through multi-moduli-based homomorphisms [Kee21], but clear limitations remain in integrating radix-3 arithmetic and in reusing hardware across security levels.

Motivation. As discussed above, agility-oriented designs are becoming increasingly common, yet existing sharing strategies presuppose radix-2^k arithmetic as the common denominator, as exemplified by PHOENIX applying the FAFFT to HQC. Even within the lattice-based family, however, the polynomial ring structures and primes adopted by individual algorithms differ fundamentally, and so do the required polynomial multiplication techniques [CDH+20, CCC+26, PK26]. In software, optimizations that flexibly accommodate such algebraic diversity have been actively developed [CHK+21, ACC+22]; as a result, even NTRU+, which operates over a trinomial ring, has been reported to achieve performance competitive with ML-KEM. In hardware, by contrast, to the best of our knowledge, no attempt has been made to accommodate this algebraic diversity within a single design, nor has a flexible design methodology that accounts for algebraic

structure been established. In particular, radix-3-based schemes over trinomial rings are structurally incompatible with existing radix-2^k-centric accelerators for two reasons. First, unifying a multi-stage decomposition that mixes radix-2 and radix-3 into a single butterfly structure demands heterogeneous datapaths, degrading pipelining and parallelization efficiency while increasing area and latency. Second, the polynomial degree and layer composition vary across security levels, and the coefficient modulus varies across schemes; supporting each case with dedicated circuitry severely limits hardware reuse across security levels and schemes. Consequently, despite their competitiveness in software, the absence of a hardware-friendly mapping has remained a practical bottleneck. Efficient hardware crypto-agility ultimately hinges on the ability to flexibly share the common operations of different algebraic structures on limited resources. This paper addresses this gap by proposing a radix-2/3 unified NTT architecture for crypto-agility.

Contribution. Our main contributions are as follows.

- **Efficient unified radix-2/3 NTT architecture for crypto-agility.**
  We propose an NTT architecture that unifies radix-2, radix-3, and trinomial transforms on a single datapath for crypto-agility. Specifically, the core butterfly operates either as two parallel radix-2 butterflies or as one radix-3 butterfly, additionally handling the trinomial transform on the same path, and the Cooley–Tukey (CT) and Gentleman–Sande (GS) dataflows share the same arithmetic units. Moreover, the proposed loosely coupled NTT accelerator refines its control signals down to the granularity of individual NTT layers, so that the radix type and execution of each layer can be configured selectively. As a result, the architecture supports the NTTs of the radix-2-based ML-KEM, ML-DSA, and HAETAE, of SMAUG-T via modulus lifting [CHK+21], and of NTRU+, which requires radix-3 and trinomial transforms. For NTRU+, it achieves ATP improvements of 4.1× over a degree-lifting-based radix-2 OpenNTT [KHMR24] (cyclic) configuration and of 7.3×/13.0× over the MDC/SDF configurations of Proteus [HMR24], respectively. Our source code will be released as open source upon publication.

- **Area-efficient and extensible unified modular multiplication unit.**
  For area efficiency, we propose `unifiedMUL`, a single arithmetic unit that handles modular multiplications for different moduli, together with a generalized modular constant division. Specifically, a generic 17-bit Montgomery path and an ML-DSA-specific 23-bit reduction path are merged to share the same two DSPs, and the per-modulus constant multiplications required for reduction are realized as shift-add trees, eliminating scheme-specific multipliers. In addition, for the 3^{-1} scaling of the radix-3 inverse NTT (INTT), we propose a DSP-free generalized modular constant division that combines the multiple alignment of Montgomery reduction with the quotient estimation of Barrett reduction. As a result, any new modulus of up to 17 bits can be supported merely by adding its shift positions; as a case study, we add the Falcon prime ($q = 12{,}289$) without any new arithmetic circuitry, at the cost of only 16 LUTs and no additional FFs or DSPs, demonstrating the extensibility of the design.

- **Conflict-free memory mapping for unified radix-2/3 NTT.**
  In a mixed-radix NTT, the parallel memory access pattern changes from layer to layer, making it difficult for a single mapping to avoid bank conflicts. To resolve this, we propose a conflict-free memory mapping that supports both radix-2 and radix-3 butterflies. Specifically, a 1-bit bank-selection function, using the parity of the full address for radix-2 and a fixed address bit for radix-3, handles 4-read/4-write parallel accesses in place on a ping-pong BRAM structure without mapping tables, permutation memories, or additional BRAMs. Moreover, at boundaries where the

radix changes between layers, computation proceeds without explicit coefficient reordering by merely switching the write mapping, and all three security levels are supported with identical bank-selection wiring. Since the bank-selection logic consists only of simple bitwise operations, namely parity and fixed-bit selection (3 LUTs for an 11-bit address width), and is not tied to specific parameters, the proposed mapping can serve as the memory structure of a broad range of mixed-radix NTT accelerators beyond this design.

2 Background

2.1 Number-Theoretic Transform (NTT)

Polynomial multiplication is one of the main computational bottlenecks in lattice-based cryptography. If the coefficient modulus q does not admit roots of unity of the required order, the product can be computed using modulus-independent algorithms such as Karatsuba or Toom–Cook [AMR23]. When such roots exist, multiplication based on the number-theoretic transform (NTT) has complexity $O(n \log n)$ [LN16]. Algebraically, the NTT is a linear map induced by a Chinese remainder theorem (CRT) decomposition of the quotient ring. Given a primitive N-th root of unity ω, the NTT maps a polynomial a to $\widehat{a}$, where $\widehat{a}_i = \sum_{j=0}^{N-1} a_j \omega^{ij} \bmod q$. The inverse NTT (INTT) uses ω^{-1} and scales the result by $N^{-1} \bmod q$. Throughout this paper, n denotes the polynomial degree and N the transform length.

Radix-2 Butterfly. When N is a power of 2, the CRT decomposition proceeds through a sequence of radix-2 steps. At each step, a factor $x^{2m} - W^2$ is split as $(x^m - W)(x^m + W)$, where the twiddle factor W is a power of the root of unity associated with that factor. Given inputs (a, b), the radix-2 NTT and INTT butterflies used in this work compute $(a, b) \mapsto (a+bW,\, a-bW)$ and $(a, b) \mapsto (a+b,\, (a-b)W)$, respectively [LN16]. If $2n \nmid (q-1)$, the defining polynomial does not split completely into linear factors over $\mathbb{Z}_q$. An incomplete NTT instead stops the CRT decomposition at low-degree factors, with the remaining products computed by base multiplication in the corresponding quotient rings. ML-KEM, for example, uses $q = 3{,}329$ and performs 7 radix-2 layers followed by base multiplication in 128 quadratic quotient rings [Nat24b, Sei18].

Radix-3 Butterfly. When $3 \mid N$, each radix-3 step splits a factor $x^{3m} - W^3$ as $x^{3m} - W^3 = \prod_{k=0}^{2}(x^m - \omega_3^k W)$. This decomposition requires a primitive cube root of unity ω_3. For prime q, such a root exists if and only if $3 \mid (q-1)$. For inputs (a, b, c), let $P_0 = Wb$, $P_1 = W^2 c$, and $T = \omega_3(P_0 - P_1)$. Using the identity $\omega_3^2 = -1 - \omega_3$, the radix-3 NTT butterfly can be written as $(a, b, c) \mapsto (a + P_0 + P_1,\, a - P_1 + T,\, a - P_0 - T)$.

Trinomial Factorization. The defining trinomial of NTRU+, $f(x) = x^n - x^{n/2} + 1$, satisfies $f(x) = \Phi_6(x^{n/2})$, where $\Phi_6(y) = y^2 - y + 1$ is the cyclotomic polynomial. Let q be a prime such that $6 \mid (q-1)$. Then $\mathbb{Z}_q$ contains a primitive 6th root of unity z, and $\Phi_6(y)$ factors as $(y - z)(y - z^{-1})$. Since z satisfies the identity $z^2 - z + 1 = 0$, we have $z(1 - z) = 1$ and hence $z^{-1} = 1 - z$. The corresponding CRT isomorphism is

$$\mathbb{Z}_q[x]/(x^n - x^{n/2} + 1) \;\cong\; \mathbb{Z}_q[x]/(x^{n/2} - z) \;\times\; \mathbb{Z}_q[x]/(x^{n/2} - z^{-1}).$$

Under this isomorphism, $a = a_0 + a_1 x^{n/2}$ is mapped to $a_0 + z a_1$ and $a_0 + z^{-1} a_1 = a_0 + a_1 - z a_1$. Both components thus share the product $z a_1$, allowing each coefficient pair to be processed with 1 multiplication by the constant z, together with additions and subtractions. NTRU+ uses $q = 3{,}457$, for which $q - 1 = 2^7 \cdot 3^3$. Hence, both ω_3 and z exist. Following the initial

Table 1: Target schemes and polynomial-multiplication methods.

	ML-KEM	**ML-DSA**	**HAETAE**	**SMAUG-T**	**NTRU+**
Reduction polynomial	$x^n + 1$	$x^n + 1$	$x^n + 1$	$x^n + 1$	$x^n - x^{n/2} + 1$
Degree n	256	256	256	256	$\{768, 864, 1{,}152\}$
Modulus	3,329	8,380,417	64,513	$\{1{,}024, 2{,}048\}$	3,457
NTT-friendly	✓	✓	✓	✗	✓
Reference method (This work)	NTT (radix-2)	NTT (radix-2)	NTT (radix-2)	Toom–Cook and Karatsuba Lifted NTT (radix-2)	NTT (radix-2/3)

trinomial split, the transform performs the radix-3 layers before the radix-2 layers, in the order specified by NTRU+ [PK26].

2.2 Modulus Lifting for NTT-Unfriendly Rings

Schemes such as SMAUG-T and Saber use a coefficient modulus q that is a power of 2, for which $\mathbb{Z}_q$ does not contain roots of unity of the order required by the NTT. Polynomial multiplication therefore cannot be computed using an NTT directly over the native ring, and early implementations relied on modulus-independent algorithms such as Karatsuba or Toom–Cook. To use NTT-based multiplication in this setting, prior work lifts the computation to one or more larger NTT-friendly moduli and subsequently reduces the product modulo the native modulus q [CHK+21, DMG23, AMJ+23].

Let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. The coefficients of $a(x)b(x)$ have a known bound, so a lifting modulus q' larger than this bound recovers the exact integer product, giving $c(x) \equiv \big(a(x)b(x) \bmod q'\big) \bmod q$. The lifting modulus can be either one NTT-friendly prime larger than the bound or several smaller primes whose product exceeds it. In the second approach, the products are reconstructed using the CRT [ACC+22]. A single modulus requires fewer transforms but a wider datapath, whereas multiple moduli reduce the datapath width for each prime at the cost of additional transforms and CRT reconstruction.

2.3 Target Schemes

NIST standardized the KEM ML-KEM [Nat24b] and the signature scheme ML-DSA [Nat24a]. In 2025, the Korean Post-Quantum Cryptography Competition (KpqC) selected the KEMs SMAUG-T and NTRU+ and the signature schemes HAETAE and AIMer [CCC+26, PK26, CCD+24, CHK+26]. Among these schemes, this work targets ML-KEM, ML-DSA, HAETAE, SMAUG-T, and NTRU+, whose polynomial multiplication can be accelerated with the NTT datapath considered here.

The polynomial multiplication methods used by the target schemes can be classified into 3 categories. ML-KEM, ML-DSA, and HAETAE use radix-2 NTTs over negacyclic rings. Since the native modulus of SMAUG-T is a power of 2, its coefficient ring does not admit a direct NTT. We instead use a radix-2 NTT under a lifting modulus. NTRU+ is defined over cyclotomic trinomial rings with degrees divisible by 3 and therefore requires an initial trinomial split followed by radix-3 and radix-2 layers. Table 1 summarizes these differences.

ML-KEM, ML-DSA, and HAETAE. All 3 schemes operate over $R_q = \mathbb{Z}_q[x]/(x^{256} + 1)$. Their coefficient moduli are fixed across security levels: $q = 3{,}329$ for ML-KEM, $q = 8{,}380{,}417$ for ML-DSA, and $q = 64{,}513$ for HAETAE. ML-DSA and HAETAE use complete

8-layer radix-2 NTTs, whereas ML-KEM uses an incomplete 7-layer radix-2 NTT followed by quadratic base multiplication.

NTRU+. NTRU+ uses the trinomial ring $R_q = \mathbb{Z}_q[x]/(x^n - x^{n/2} + 1)$, where $n \in \{768, 864, 1{,}152\}$ depends on the security level and $q = 3{,}457$ is used for all parameter sets. The final specification fixes the order of the NTT layers. The transform first splits the trinomial ring into 2 quotient rings, applies the radix-3 layers, and then applies the radix-2 layers. Applying the radix-3 layers before the radix-2 layers avoids an additional precomputation table for base multiplication [PK26]. Each schedule begins with 1 trinomial layer. For NTRU+768/864/1,152, the numbers of radix-3 layers are 1/2/2, the numbers of radix-2 layers are 5/4/4, and the final quotient ring degrees are 4/3/4.

SMAUG-T. SMAUG-T is based on the Module Learning With Errors (Module-LWE) and Module Learning With Rounding (Module-LWR) problems. SMAUG-T1 uses $q = 1{,}024$, while SMAUG-T3 and SMAUG-T5 use $q = 2{,}048$. Since these moduli do not admit roots of unity of the required order, the reference implementation uses Toom–Cook and Karatsuba multiplication [CCC+26]. We apply the modulus lifting from Section 2.2 to reuse the 23-bit arithmetic path provided for ML-DSA.

3 Unified Mixed-Radix NTT Architecture

3.1 NTT Strategies for Unified Hardware

The 5 schemes in Table 1 differ in whether they admit a direct NTT, the required radix schedule, and the coefficient modulus width. To support these differences, the modular multiplication datapath shares 2 multipliers before branching into a generic 17-bit Montgomery reduction path and a 23-bit reduction path specialized for the ML-DSA modulus. The architecture also provides 3 butterfly modes for radix-2, radix-3, and trinomial operations, all using the same coefficient memory.

Shared 17-Bit Path for ML-KEM, HAETAE, and NTRU+. ML-KEM uses $q = 3{,}329$, HAETAE uses $q = 64{,}513$, and NTRU+ uses $q = 3{,}457$. All 3 moduli fit within 17 bits. HAETAE software typically uses 32-bit arithmetic because its modulus is close to the 16-bit limit and 16-bit arithmetic requires an immediate reduction after each operation [CCD+24, KvVBS26]. On an FPGA, the datapath can be set to the required width, allowing all 3 schemes to share a 17-bit path. The shared path uses Montgomery reduction with $R = 2^{17}$. Selecting q and $q^{-1} \bmod R$ for the active scheme allows the same arithmetic units to process all 3 moduli. Barrett and Plantard arithmetic are possible alternatives, but they require wider multiplication operands and datapaths in this configuration [Bar87, HZZ+22, HZZ+24]. We therefore use Montgomery reduction to keep the shared path within 17 bits and reuse 2 DSPs.

Shared 23-Bit Path for ML-DSA and SMAUG-T. Single modulus lifting for SMAUG-T requires a prime that exceeds 22 bits to cover the coefficient bound [KSK+25]. Multiple lifting moduli would require CRT reconstruction and twice as many NTT executions. We therefore use the ML-DSA modulus $q = 8{,}380{,}417$ as the lifting modulus and reuse its 23-bit datapath, twiddle factor storage, and reduction logic. The reduction logic follows the method for the ML-DSA prime in [MCL+23].

Direct Mixed-Radix Support for NTRU+. Mapping NTRU+ to hardware limited to radix-2 requires embedding the polynomial in a power-of-two transform and using a

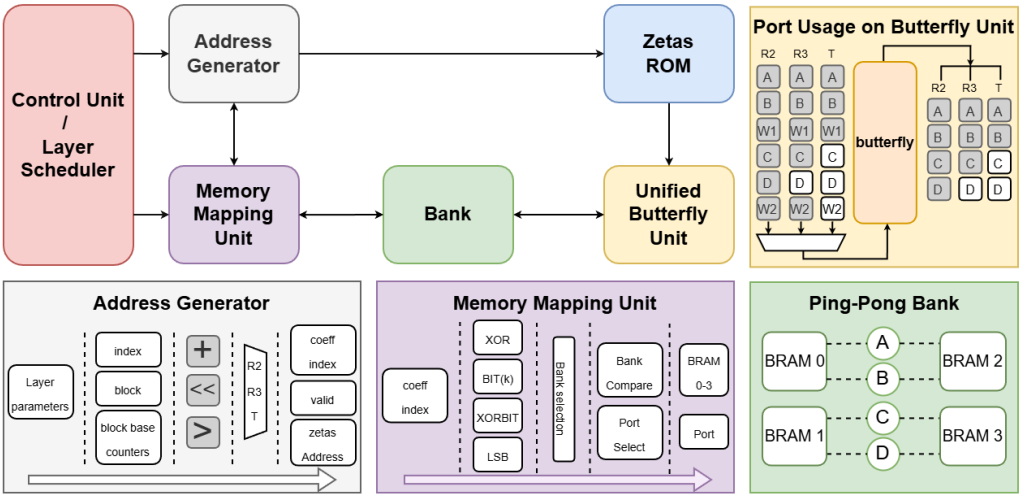

Figure 1: Unified Mixed-Radix NTT Architecture

sufficiently large lifting modulus [Kee21]. For the largest parameter set, this approach requires transform length $N = 4{,}096$ and a 33-bit lifting modulus, increasing the number of layers, twiddle factor storage, and multiplier width. NTRU+ also represents its public key in the NTT domain, so the hardware must reproduce the mixed-radix representation defined by the specification [PK26]. We therefore support the initial trinomial split and radix-3 layers directly, while reusing the 17-bit Montgomery path for their modular multiplications.

3.2 Architecture Overview

Figure 1 shows the top-level architecture. It uses a shared coefficient memory and butterfly datapath for radix-2, radix-3, and trinomial NTT/INTT layers. For each layer, a layer descriptor configures the address generation, memory mapping, twiddle factor order, and butterfly mode. The coefficient memory consists of 2 ping-pong sets, each containing 2 logical dual-port banks.

3.2.1 Memory-Based Iterative Architecture

NTT accelerators generally use either a memory-based iterative architecture or a streaming architecture. A memory-based architecture repeatedly reads and writes coefficients in memory, whereas a streaming architecture passes them through delay lines. Memory-based designs [CYY+22, DMÖS22, AMI+23, MR24a, BAA23] can be configured with different numbers of butterfly units and memory banks and support different address generation and operation modes for each layer. Streaming designs provide high throughput for a fixed transform but require delay buffers and commutators tailored to its polynomial degree and radix schedule [KMSH24, MR24b].

Streaming pipelines are inefficient for the target transforms because the layer type changes from trinomial to radix-3 and radix-2 within a transform, while the polynomial degree and layer schedule also vary across parameter sets. Supporting all schedules would require either a pipeline configured for the largest case with unused stages and delay buffers or a separate pipeline for each schedule, increasing area and control complexity. We therefore use a memory-based iterative architecture that reconfigures the address generation, memory mapping, twiddle factor selection, and butterfly mode for each layer while reusing the shared datapath and coefficient memory. The coefficient memory is organized as 2 ping-pong sets. During each layer, one set serves all reads and the other

serves all writes, and their roles are exchanged at the layer boundary. This separation prevents read-after-write conflicts and allows the butterfly pipeline to accept a new input set every cycle. It also passes the results to the next layer without coefficient reordering.

3.2.2 Butterfly Datapath Organization

Prior unified NTT designs share modular multipliers and butterfly datapaths across schemes with similar radix-2 dataflow, such as Kyber and Dilithium [AMJ+23, AMI+23, BAA23, MR24a], but remain limited to radix-2 schemes. To the best of our knowledge, our unified butterfly unit is the first to support radix-2, radix-3, and trinomial transforms on a single datapath. A radix-3 butterfly requires 2 twiddle factor multiplications per cycle, so the unit contains 2 modular multipliers. Radix-2 mode uses both multipliers for 2 parallel butterflies, whereas trinomial mode uses only 1. The unit accepts coefficient inputs (A, B, C, D) and twiddle factor inputs (W_1, W_2) and produces up to 4 outputs. Section 4.2 describes the operator mapping for each mode and the 15 cycle pipeline.

3.2.3 Address Generation and Bank Mapping

The address generator produces the coefficient index and zeta address for each butterfly from the radix type, layer parameters, and coefficient stride. It uses only counters, adders, shifters, comparators, and multiplexers and performs no division or modulo operation in the combinational path, including for the radix-3 $\times 3$ stride. This keeps the critical path short across mixed-radix layer transitions. The memory mapping unit computes the physical bank index from each coefficient index generated by the address generator. The 4 true dual-port BRAMs form 2 ping-pong sets. The mapping guarantees that each BRAM receives at most 2 accesses in the same direction per cycle, allowing its 2 ports to serve them without bank conflicts.

Prior work [XHY+20, RMD+15] uses memory permutations or conflict-free schedules to avoid bank conflicts. In-place schemes [CYY+22, ZYC+20] reduce coefficient reordering overhead. However, these methods either support only radix-2 and radix-4 or handle radix-3 separately without supporting an in-place transition from radix-3 to radix-2. For NTRU+864, the outputs of the stride-48 radix-3 layer must be written according to the bank mapping required by the following stride-24 radix-2 layer. Our scheme switches the write mapping at this boundary, allowing the 2 layers to execute consecutively without coefficient reordering or modulo operations. Section 5 presents the mapping in detail.

3.2.4 Layer Scheduler

For each layer, the control unit issues a descriptor containing the butterfly mode, block size, stride, block count, zeta base and direction, read and write memory sets, and memory mapping mode. The scheduler processes the descriptor in 5 cycles for register update, address generation, bank/port mapping, coefficient and zeta reads, and BRAM output selection and write-back control. All descriptor fields are derived solely from public scheme and layer parameters, including the radix, stride, and block size. The scheduler therefore follows the same fixed 5 cycle sequence for every scheme and layer, including mapping-mode changes and mode transitions. The 5 cycle operation is hidden by the butterfly pipeline, adding no delay between consecutive layers.

When the same schedule is repeated across multiple blocks, the scheduler keeps the butterfly position fixed and iterates over `block_num`. The address generation and bank mapping pattern computed for the first block is reused for the remaining blocks by changing only the block offset. For example, each radix-2 layer of NTRU+768 processes 6 blocks consecutively using the same pattern [PK26]. This reuse avoids restarting address generation or adding pipeline latency at block boundaries, allowing the entire layer to

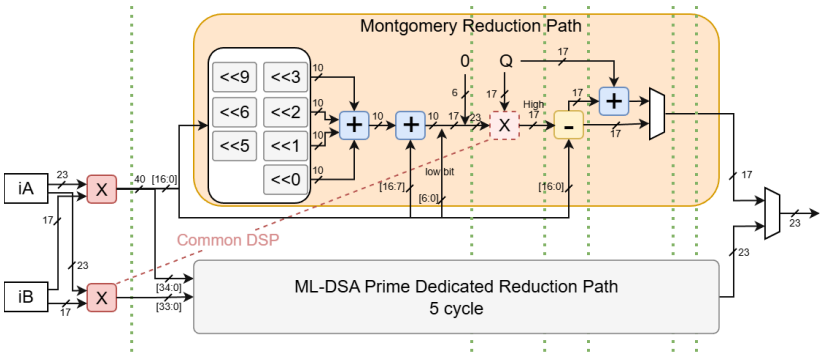

Figure 2: Unified Modular Multiplication (`unifiedMUL`) structure

execute without stalls. A new scheme can be added through descriptors and constants if its modulus width, reduction type, radix, and memory mapping mode are supported.

4 Unified Arithmetic Datapath

4.1 Unified Modular Multiplication (`unifiedMUL`)

Following the arithmetic mapping in Section 3.1, `unifiedMUL` shares 2 DSPs and branches into a 23-bit reduction path for the ML-DSA modulus and a 17-bit Montgomery path for ML-KEM, HAETAE, and NTRU+. Figure 2 shows the 6-stage datapath. The scheme mode selects the reduction path, and the module contributes 6 cycles to the `unifiedBUT` pipeline.

The 23-bit path implements modular multiplication for the ML-DSA modulus $q = 8{,}380{,}417$ using the reduction method in [MCL+23]. A DSP48E1 supports multiplication up to 25×18 bits, so a 23×23-bit product requires 2 DSPs. Operand b is split into a 12-bit low part b_l and an 11-bit high part b_h, which are multiplied by a in parallel. The 2 aligned partial products are added and reduced using the relation $q = 2^{23} - 2^{13} + 1$.

The 17-bit path uses Montgomery multiplication with $R = 2^{17}$. One DSP computes $t = ab$. The other computes the modulus multiple uq. We define $q' = q^{-1} \bmod R$, and the reduction computes

$$u = (tq') \bmod R, \qquad r = \frac{t - uq}{R}.$$

If r is negative, the reduction adds q to bring the result into $[0, q)$. The precomputed inverse q' is 62,209 for ML-KEM, 12,929 for NTRU+, and 66,561 for HAETAE. Because only the low 17 bits of tq' are required, the mode selects the shifts of t for the set bits of q', and a shared adder combines them to form u. This shift-add structure supports all 3 moduli without a separate constant multiplier or an additional DSP. Twiddle factors are stored in BRAM in the Montgomery domain as $WR \bmod q$, so $a(WR)R^{-1} = aW$ remains in the standard domain without a separate domain conversion for the coefficients.

Because each transform uses only one modulus, the 2 reduction paths share the same 2 DSPs and a 6 cycle interface. The 17-bit path can support a new modulus by adding the shift positions required by its q' and extending the scheme mode. For Falcon [FHK+20] ($q = 12{,}289$), the required shift positions are $\{0, 12, 14, 15, 16\}$, all of which are already present. Adding Falcon therefore requires only a 1-bit extension to the scheme mode and no new arithmetic unit. Synthesis adds 16 LUTs, with no additional FFs or DSPs.

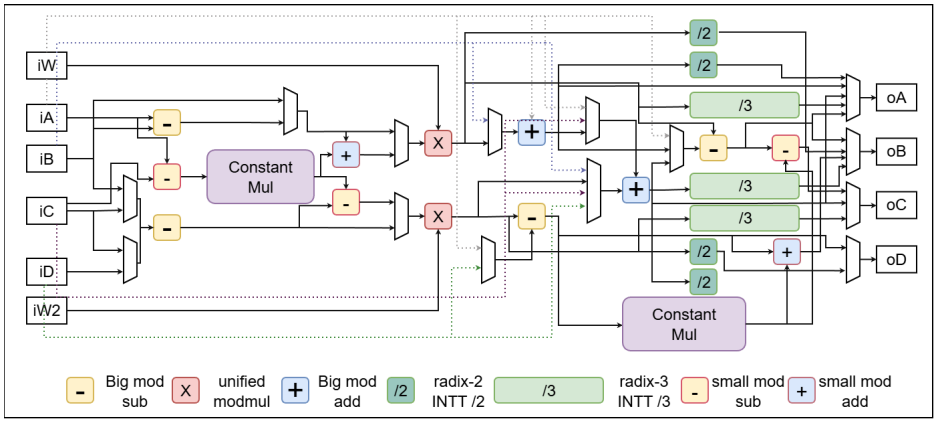

Figure 3: Unified Butterfly Unit (`unifiedBUT`) structure

4.2 Unified Butterfly Unit (`unifiedBUT`)

Prior unified NTT designs share multipliers and datapaths among schemes with similar radix-2 structures, such as Kyber and Dilithium [AMJ+23, AMI+23, BAA23, MR24a]. The mixed-radix schedule of NTRU+ adds radix-3 NTT/INTT and trinomial NTT/INTT, giving 6 modes in total. Implementing each mode as a dedicated unit would duplicate multipliers and modular add/sub units and negate the area benefit of integration.

Because `unifiedMUL` dominates the area cost of the butterfly datapath, `unifiedBUT` minimizes the number of `unifiedMUL` instances and shares them across all 6 modes. A radix-3 butterfly requires 2 twiddle factor multiplications per cycle, making 2 instances the minimum for an initiation interval of 1 cycle. In radix-2 mode, the same 2 instances process the butterflies (A, B, W_1) and (C, D, W_2) in parallel, whereas trinomial mode uses only 1 instance because it requires 1 multiplication. Thus, no mode uses more than 2 instances, and the interface consists of 4 coefficient inputs (A, B, C, D) and 2 twiddle factor inputs (W_1, W_2).

The non-multiplier operators follow 2 resource-sharing principles. First, operations on different pipeline paths are mapped to the same physical operator when the mode signal ensures that they are never active simultaneously. Second, operations at different pipeline positions across modes are advanced or delayed to align with the shared operator. The supported schemes determine the datapath width. NTRU+ alone would require only 12-bit arithmetic, but support for ML-DSA (q = 8,380,417) requires 23-bit arithmetic on the shared radix-2 path. Because radix-3 and trinomial modes are used only for NTRU+, the additional add/sub units required by these modes, the DSP-free `CONSTMUL` for $K = \omega_3 \bmod q = 2{,}734$ (Section 2), and the scaling use 12-bit arithmetic. In the following, m_0, m_1 denote the 2 multiplier inputs, and $P_0 = m_0 W_1 \bmod q$ and $P_1 = m_1 W_2 \bmod q$ denote their outputs. The notation $x /_q d$ denotes $x \cdot d^{-1} \bmod q$, and every add/sub is a modular operation corrected into $[0, q)$.

Operator Mapping. As shown in Figure 3, we divide the datapath at `unifiedMUL` into a pre-stage and a post-stage. NTT modes multiply the coefficients directly, so their modular add/sub operations are placed in the post-stage, whereas INTT modes first form the coefficient differences for the multiplier inputs, so these operations are placed in the pre-stage. Table 2 lists the operands assigned to each physical operator across the 6 modes. Reading down a column shows the dataflow of one mode, while reading across a row shows how one physical operator is shared across modes. For example, ADD_1 computes $A + P_0$ in the NTT modes and $A + B$ in the INTT modes. Because both additions occur at the

Table 2: Shared-operator mapping of `unifiedBUT` across the 6 operation modes. Numbers in parentheses denote the number of simultaneous operations.

			Radix-2 (2)		**Radix-3** (1)		**Trinomial** (1)	
Stage	**Operator**	**Width**	NTT	INTT	NTT	INTT	NTT	INTT
Pre	SUB_1	23-bit	—	$B-A$	—	$B-A$	—	$B-A$
	SUB_2	23-bit	—	$D-C$	—	$C-B$	—	—
	`CONSTMUL`	12-bit	—	—	—	$T_1{=}K(B-A)$	—	—
	Others	12-bit	—	—	—	$C-A$ $(C-A)+T_1$ $(C-B)-T_1$	—	—
Mul	MUL_0 (m_0)	23-bit	B	$B-A$	B	$(C-A)+T_1$	B	$B-A$
	MUL_1 (m_1)	23-bit	D	$D-C$	C	$(C-B)-T_1$	—	—
Post	ADD_1	23-bit	$A{+}P_0$	$A+B$	$A{+}P_0$	$A+B$	$A{+}P_0$	$A+B$
	ADD_2	23-bit	$C{+}P_1$	$C+D$	$(A{+}P_0){+}P_1$	$(A+B)+C$	$A+B$	—
	SUB_3	23-bit	$C{-}P_1$	—	$A{-}P_1$	—	—	—
	SUB_4	23-bit	$A{-}P_0$	—	$A{-}P_0$	—	$(A+B){-}P_0$	$(A+B){-}P_0$
	`CONSTMUL`	12-bit	—	—	$T_2{=}K(P_0{-}P_1)$	—	—	—
	Others	12-bit	—	—	P_0-P_1 $(A-P_1)+T_2$ $(A-P_0)-T_2$	—	—	—
	Scaling	23-bit	—	$/_q 2$ (4)	—	—	—	—
		12-bit	—	—	—	$/_q 3$ (3)	—	$/_q 2$ (1)

same pipeline position, they share a single physical adder.

Radix-2 Mode. Radix-2 mode uses the shared datapath to process 2 butterflies, (A, B, W_1) and (C, D, W_2), in parallel. The outputs are

$$(A_o, B_o, C_o, D_o) = \begin{cases} \big(A + P_0,\ A - P_0,\ C + P_1,\ C - P_1\big), & \text{NTT}, \\ \big((A + B)\ /_q\ 2,\ P_0\ /_q\ 2,\ (C + D)\ /_q\ 2,\ P_1\ /_q\ 2\big), & \text{INTT}. \end{cases}$$

In NTT mode, B and D are multiplied directly, whereas in INTT mode, the pre-stage forms $B - A$ and $D - C$ before multiplication. As shown in Table 2, radix-2 uses 2 subtractors before `unifiedMUL` ($\mathsf{SUB}_1, \mathsf{SUB}_2$) and 2 adders ($\mathsf{ADD}_1, \mathsf{ADD}_2$) and 2 subtractors ($\mathsf{SUB}_3, \mathsf{SUB}_4$) after it. All 6 operators use 23-bit arithmetic to support ML-DSA. Together, they form the shared add/sub datapath. The INTT outputs also pass through $x\ /_q\ 2$ scaling (Section 4.3).

Radix-3 Mode. The difference between NTT and INTT operator placement is largest in radix-3 mode. As shown in Table 2, radix-3 uses 4 subtractors and 1 adder before `unifiedMUL`, and 3 adders and 4 subtractors after it. This longer path sets the 15 cycle pipeline latency and makes resource sharing more difficult than in radix-2 mode. The position of the multiplication by K also differs between NTT and INTT. INTT computes $T_1{=}K(B{-}A)$ before `unifiedMUL`, whereas NTT computes $T_2{=}K(P_0{-}P_1)$ after it. The $\mathsf{SUB}_{1,2}$, $\mathsf{ADD}_{1,2}$, and $\mathsf{SUB}_{3,4}$ units reuse the 23-bit operators of radix-2, and T_1 reuses the $B{-}A$ output of SUB_1. A single `CONSTMUL` is shared across NTT and INTT modes, so radix-3 adds only two 12-bit adders and four 12-bit subtractors for $C{-}A$, T_1, and T_2. The `CONSTMUL` computes multiplication by $K = 2{,}734$ by summing 3 lookup table outputs for the 4-bit digits, digit $\times\ K \cdot 16^i \bmod q$, followed by at most 2 conditional subtractions of q. This produces a modular product in 2 cycles without a DSP. Section 4.3 describes the

$x \,/_q\, 3$ scaling of the INTT outputs. The radix-3 outputs are

$$(A_o, B_o, C_o) = \begin{cases} \big(A + P_0 + P_1,\ (A - P_1) + T_2,\ (A - P_0) - T_2\big), & \text{NTT}, \\ \big((A + B + C) \,/_q\, 3,\ P_0 \,/_q\, 3,\ P_1 \,/_q\, 3\big), & \text{INTT}. \end{cases}$$

We considered lazy reduction to defer modular reductions, but did not adopt it because the wider inputs required by `CONSTMUL` and the partial product path would cost more area than they save.

Trinomial Mode. We use only MUL_0 for the single trinomial layer and leave MUL_1 idle because processing 2 butterflies in parallel would complicate coefficient memory access. The outputs are

$$(A_o, B_o) = \begin{cases} \big(A + P_0,\ (A + B) - P_0\big), & \text{NTT}, \\ \Big(\big((A + B) - P_0\big) \,/_q\, 2,\ P_0\Big), & \text{INTT}. \end{cases}$$

As shown in Section 2, the 2 trinomial outputs share the product za_1, so the NTT requires only 1 multiplication by z. The NTT path therefore reuses $\mathsf{ADD}_{1,2}$, SUB_4, and MUL_0. For the INTT, storing $-\rho$ as W_1, where $\rho = (z_1 - z_2)^{-1}$, converts $(A - B)\rho$ to $(B - A)(-\rho)$ and reuses the existing $B - A$ pre-stage path. Only the first INTT output requires $x \,/_q\, 2$ scaling.

Pipeline Organization. All 6 modes use a fixed 15 cycle latency consisting of a 3 cycle pre-stage, a 6 cycle `unifiedMUL`, and a 6 cycle post-stage. Radix-3 determines this latency because it has the longest path in both the pre-stage and post-stage. The fixed latency eliminates mode-specific completion control in the higher-level scheduler. All additional paths and scaling units are pipelined to sustain a throughput of 1 input set per cycle while the mode remains fixed. Radix-2, radix-3, and trinomial modes consume 4, 3, and 2 coefficients per cycle. Section 5 describes the corresponding memory access.

4.3 Layer Scaling Unit

An INTT requires an overall scaling by $N^{-1} \bmod q$, which can be applied to every coefficient after the transform using a constant multiplier. However, the supported schemes have different moduli, radix schedules, and layer counts, so this approach requires a separate scaling constant and modular arithmetic for each configuration.

Scaling by 2^{-1}. For radix-2, modular halving satisfies $a \cdot 2^{-1} \bmod q = \frac{a + (a \bmod 2)\, q}{2}$, allowing the scaling to be implemented using only a parity-dependent addition of q and a right shift [MCL+23, ZYC+20]. The design integrates this operation into the scaling path after the radix-2 INTT butterfly with low area overhead.

Generalized Modular Constant Division. Radix-3 scaling uses the divisor $d = 3$, which is not a power of 2 and cannot be handled by a shift. Adding a general modular arithmetic unit to `unifiedBUT` would require additional DSPs and logic. We therefore implement scaling by a non-power-of-two constant without a general modular multiplier. The unit uses precomputed constants for a fixed coprime pair (q, d) rather than computing a modular inverse at runtime.

Montgomery reduction aligns an intermediate value to a multiple of $R = 2^n$ by adding a multiple of q to the input, replacing division by R with a shift. We apply this alignment idea to an arbitrary non-power-of-two scaling constant d: computing $k \leftarrow (-a\, q^{-1}) \bmod d$

Algorithm 1 Generalized modular constant division

Require: $a \in [0, q)$, $d \geq 2$, $\gcd(q, d) = 1$
Require: Precomputed: $q^{-1} \bmod d$, $\ell = \min\{n \in \mathbb{N} : 2^n > q(d-1)\}$, $\mu = \lceil 2^\ell / d \rceil$
Ensure: $\text{Res} = a\, d^{-1} \bmod q$
1: $k \leftarrow (-a\, q^{-1}) \bmod d$
2: $v \leftarrow a + kq$
3: $\text{Res} \leftarrow \lfloor (v \cdot \mu)/2^\ell \rfloor$
4: **return** Res

and adding the correction kq makes the intermediate value v a multiple of d, so the modular division reduces to the exact integer division v/d, whose remainder is known to be zero. This integer division is replaced, following the reciprocal-multiplication technique of invariant integer division [GM94], by a multiplication with the precomputed constant $\mu = \lceil 2^\ell / d \rceil$ and an ℓ-bit shift. Unlike Barrett reduction, which uses a floor reciprocal and may need a correction subtraction, this estimate uses the ceiling reciprocal μ and is exact without correction for exact division. The algorithm combines this with Montgomery-style alignment to extend it to a modular division for a modulus q coprime to d. Setting $d = 2$ recovers the radix-2 case: for odd q, $q^{-1} \equiv 1 \pmod 2$, so $k = a \bmod 2$ and $\mu = \lceil 2^\ell / 2 \rceil = 2^{\ell-1}$, making the 2^{-1} scaling rule of radix-2 a special case of Algorithm 1. Lemma 1 shows that, under the stated condition on ℓ, the estimate is exact without correction and equals $a\, d^{-1} \bmod q$.

Lemma 1. *Let $q, d \in \mathbb{N}$ satisfy $q > 1$, $d \geq 2$, and $\gcd(q, d) = 1$. Define ℓ as the smallest natural number satisfying $2^\ell > q(d-1)$, and let $\mu = \lceil 2^\ell / d \rceil$, as in Algorithm 1. Then, for every $a \in [0, q)$, Algorithm 1 returns $\text{Res} = a\, d^{-1} \bmod q$.*

Proof. First, $k \equiv -aq^{-1} \pmod d$ gives $v = a + kq \equiv a - a \equiv 0 \pmod d$. Thus, $w = v/d$ is an integer. Moreover, $a \leq q - 1$ and $k \leq d - 1$ imply $v = a + kq < dq$, and therefore $0 \leq w < q$.

Next, we show that $\lfloor v\mu / 2^\ell \rfloor = w$. By the definition of μ, $0 \leq d\mu - 2^\ell \leq d - 1$, and

$$\frac{v\mu}{2^\ell} = \frac{dw\mu}{2^\ell} = w + \frac{w(d\mu - 2^\ell)}{2^\ell}.$$

The error term satisfies $0 \leq w(d\mu - 2^\ell)/2^\ell \leq (q-1)(d-1)/2^\ell < q(d-1)/2^\ell < 1$, where the final inequality follows from $2^\ell > q(d-1)$. Since w is an integer, it follows that $\lfloor v\mu / 2^\ell \rfloor = w$.

Finally, $dw = v = a + kq \equiv a \pmod q$. Since $\gcd(d, q) = 1$, we have $w \equiv a\, d^{-1} \pmod q$. Since $w \in [0, q)$, $\text{Res} = w = a\, d^{-1} \bmod q$. □

NTRU+ 3^{-1} Scaling. Algorithm 2 specializes Algorithm 1 to NTRU+ with $(q, d) = (3{,}457, 3)$. Since d and q are fixed, all multiplications are eliminated or replaced by shift-add operations. Since $q = 3{,}457 \equiv 1 \pmod 3$, we have $q^{-1} \equiv 1 \pmod 3$. Thus, setting $r = a \bmod 3$ gives $k = (-r) \bmod 3$ without multiplication. The mapping $(r, k) \in \{(0,0), (1,2), (2,1)\}$ selects v from $\{a,\ a{+}2q,\ a{+}q\}$. The smallest ℓ satisfying $2^\ell > q(d{-}1) = 6{,}914$ is 13, which gives $\mu = \lceil 2^{13}/3 \rceil = 2{,}731$. Therefore, $a \cdot 3^{-1} \bmod 3{,}457 = \lfloor 2{,}731 v / 2^{13} \rfloor$.

Algorithm 2 uses only small adders, multiplexers, and fixed shifts, and requires no DSP blocks or dividers. Since $4 \equiv 1 \pmod 3$, line 1 computes $a \bmod 3$ from the sum of the 2-bit chunks of the 12-bit input. Lemma 1 guarantees that the result lies in $[0, q)$, eliminating a final conditional subtraction. The unit therefore completes the scaling within the layer without stalling the NTT pipeline.

Algorithm 2 NTRU+ modular division by three (specialization of Algorithm 1 to $(q, d) = (3{,}457, 3)$)

Require: $a \in [0, q)$, $(q, d) = (3{,}457, 3)$
Ensure: $\text{Res} = a \cdot 3^{-1} \bmod q$
1: $r \leftarrow a \bmod 3$
2: **if** $r = 0$ **then**
3: $\quad v \leftarrow a$
4: **else if** $r = 1$ **then**
5: $\quad v \leftarrow a + 2q$
6: **else**
7: $\quad v \leftarrow a + q$
8: **end if**
9: $t_1 \leftarrow v + (v \ll 2)$
10: $t_2 \leftarrow t_1 + (t_1 \ll 4)$
11: $t_3 \leftarrow t_2 + (t_1 \ll 8)$
12: $p \leftarrow v + (t_3 \ll 1)$
13: $\text{Res} \leftarrow p \gg 13$
14: **return** Res

Extensibility to Other Non-Power-of-Two Divisors. Algorithm 1 also applies to other non-power-of-two divisors d satisfying $\gcd(q, d) = 1$, with $q^{-1} \bmod d$, ℓ, and $\mu = \lceil 2^\ell / d \rceil$ selected for each d. In this extension, the multiplication by μ uses shift-add operations, and its cost depends on the hamming weight of μ. From the definition of ℓ, $2^{\ell-1} \le q(d-1) < 2^\ell$,

$$q\left(1 - \frac{1}{d}\right) < \mu \le \left\lceil 2q\left(1 - \frac{1}{d}\right)\right\rceil \le 2q.$$

Thus, μ is bounded by $2q$ for any supported d. For $q = 3{,}457$, $d = 3$, 5, and 7 give $\mu = 2{,}731$, 3,277, and 4,682, with hamming weights of 7, 7, and 5, respectively. Since $\mu \le 2q = 6{,}914 < 2^{13}$, μ requires at most 13-bits for any supported d.

5 Conflict-Free Memory Mapping for Mixed-Radix NTTs

This section proposes a conflict-free memory mapping for an NTT using the unified butterfly unit and 4 dual-port BRAMs. As described in Section 4.2, the unit processes either 2 radix-2 butterflies or 1 radix-3 butterfly. Radix-2 mode requires 4 reads and 4 writes for 4 coefficients, while radix-3 mode requires 3 reads and 3 writes for 3 coefficients. Supporting both modes therefore requires the 8 ports provided by the 4 BRAMs. The proposed mapping distributes these accesses across the BRAMs without conflicts and adds little logic or latency.

To avoid memory conflicts, the proposed mapping computes a 1-bit signal σ from each coefficient index to select its memory bank. The same index always produces the same σ, so each coefficient is read from and written to the same bank and can be updated in place. We define σ according to the radix of the current layer as follows. Here, w denotes the address width and $x_{[i]}$ denotes the i-th bit of x.

$$\sigma(x) = \begin{cases} \bigoplus_{j=0}^{w-1} (x_{[j]}), & \text{radix-2 layer} \\ x_{[d_N]}, & \text{radix-3 layer} \end{cases}$$

For a radix-2 layer, σ is the XOR reduction of all bits of x, or equivalently the parity of

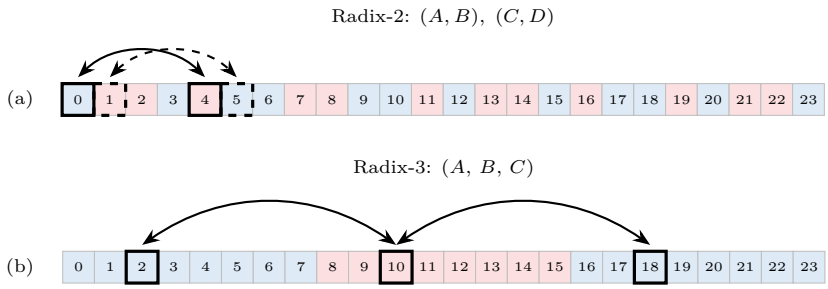

Figure 4: Conflict-free memory mapping for an example of degree $24 = 2^3 \cdot 3$. Colors indicate the bank index σ. (a) For a radix-2 layer with stride $s = 4$, the parity mapping maps the 4 coefficients $(A, B, C, D) = (0, 4, 1, 5)$ accessed in 1 cycle to the 2 banks with a $2 + 2$ distribution. (b) For a radix-3 layer with stride $s = 8$ and $d_N = 3$, the fixed-bit mapping maps the 3 coefficients $(2, 10, 18)$ to the 2 banks with a $2+1$ distribution, following the $(\alpha, \overline{\alpha}, \alpha)$ pattern.

x. For a radix-3 layer, σ is the fixed bit $x_{[d_N]}$. Figure 4 illustrates the resulting conflict-free bank mapping for a reduced example of degree 24.

5.1 Conflict-Free Mapping for Radix-2 Layers

In radix-2 mode, the unified butterfly unit processes 2 radix-2 butterflies in parallel, so the 4 coefficients must be read and written without bank conflicts. Let the input pairs of the 2 butterflies be (A, B) and (C, D). The coefficients within each pair are separated by a stride $2^n 3^m$. For radix-2-only schemes such as ML-KEM and ML-DSA and schemes with complete radix-3 NTTs such as NTRU+768 and NTRU+1,152, $m = 0$. For schemes with incomplete radix-3 NTTs such as NTRU+864, $m > 0$. XOR address mapping is an established technique for conflict-free parallel access in radix-2 FFTs. With 2 banks, the bank index is the parity of all address bits [TJS03]. Our design applies this mapping to the 2 parallel radix-2 butterflies and maps the 4 coefficients accessed in each cycle to the 2 dual-port BRAMs with a $2 + 2$ distribution.

$$\sigma(A) = \bigoplus_{j=0}^{w-1} A_{[j]} = \alpha, \qquad \sigma(B) = \bigoplus_{j=0}^{w-1} (A + 2^n 3^m)_{[j]} = \beta,$$

$$\sigma(C) = \bigoplus_{j=0}^{w-1} (A + 1)_{[j]} = \overline{\alpha}, \qquad \sigma(D) = \bigoplus_{j=0}^{w-1} (A + 2^n 3^m + 1)_{[j]} = \overline{\beta}.$$

The 2 butterflies select consecutive coefficient pairs within the same block of the NTT layer, so $C = A + 1$ and $D = B + 1$. Since A is the sum of an even block start address and an even offset, A is always even. When the stride is even ($n \geq 1$), B is also even. Adding 1 to an even index flips only the least significant bit and therefore flips the parity, giving $\sigma(C) = \overline{\alpha}$ and $\sigma(D) = \overline{\beta}$. In the last layer of a complete NTT, where the stride is 1, 2 consecutive blocks are paired to process $(A, A + 1, A + 2, A + 3)$ with A even, so the same parity flip holds within each pair.

Since the radix-2 layer stride of the target schemes is always even or 1, the 4 coefficients produce exactly 2 σ values of 0 and 2 of 1 in every layer, avoiding bank conflicts. Since the same σ is always generated for each coefficient index, each coefficient is read from and written to the same bank and updated in place.

This $2 + 2$ distribution also provides conflict-free access at the port level. In pipelined operation, the current butterfly inputs are read in the same cycle in which earlier outputs

are written after the pipeline latency. Therefore, 4 reads and 4 writes must be served simultaneously. We group the 4 dual-port BRAMs into 2 pairs. In each layer, 1 pair serves reads and the other serves writes, and their roles are exchanged in ping-pong fashion at each layer boundary. Therefore, reads and writes never access the same physical BRAM in a cycle. Within each active pair, σ gives the bank index. The 4 read addresses and the 4 delayed write addresses each produce 2 σ values of 0 and 2 of 1. Thus, each physical BRAM receives at most 2 accesses in the same direction per cycle, which are served by its 2 independent ports.

This XOR-based mapping has low hardware overhead. For an address width w, σ is computed by an XOR reduction of the w address bits, which the FPGA synthesizer maps to a shallow LUT tree. On an FPGA with 6-input LUTs and $w = 11$, the parity computation requires 3 LUTs and 2 logic levels. This logic is independent of n and m in the stride 2^n3^m, so no reconfiguration is required between layers. Bank index generation requires no mapping table, permutation memory, DSP, or additional BRAM.

5.2 Conflict-Free Mapping for Radix-3 Layers

In radix-3 mode, the unified butterfly unit performs 1 radix-3 butterfly, so the reads and writes of 3 coefficients must be served without bank conflicts.

NTRU+ uses a trinomial ring of degree $N \in \{768, 864, 1{,}152\}$, so the first NTT layer splits the trinomial ring into 2 subrings using a radix-2 butterfly. The resulting subring degrees are $2^7 \cdot 3 = 384$ for $N = 768$, $2^4 \cdot 3^3 = 432$ for $N = 864$, and $2^6 \cdot 3^2 = 576$ for $N = 1{,}152$. Each subsequent radix-3 layer splits a block into 3 subblocks.

In a radix-3 layer over a block of degree 2^n3^m, the 3 butterfly inputs are equally spaced with stride $s = 2^n3^{m-1}$ and are written as $(A, B, C) = (A, A + s, A + 2s)$. We write $s = t \cdot 2^n$, where $t = 3^{m-1}$. Since t is a power of 3, it is always odd regardless of m. The lower n bits of $t \cdot 2^n$ are all 0. Thus, adding s to A produces no carry into the n-th bit, and the n-th bit changes by exactly t mod 2. Therefore, we set the selection bit to $\sigma(x) = x_{[n]}$ and let $\alpha = \sigma(A)$. For any address A, we have

$$\begin{aligned}
\sigma(A) &= A_{[n]} = \alpha, \\
\sigma(B) &= (A + t2^n)_{[n]} = (\alpha + t) \bmod 2 = \overline{\alpha}, \\
\sigma(C) &= (A + 2t2^n)_{[n]} = (\alpha + 2t) \bmod 2 = \alpha.
\end{aligned}$$

Since t is odd and $2t$ is even, the 3 inputs follow the pattern $(\alpha, \overline{\alpha}, \alpha)$. Thus, A and C map to the same bank, while B maps to the opposite bank, and no bank conflict occurs.

As the radix-3 layers proceed, only the exponent m in the block degree decreases, so the selection bit position n remains fixed throughout the radix-3 phase. However, n depends on the security level. To support all 3 security levels with the same bank selection wiring, we set the selection bit parameter d_N according to n. We set $d_N = 7$ for NTRU+768, $d_N = 4$ for NTRU+864, and $d_N = 6$ for NTRU+1,152.

Port-level bank conflicts are also avoided by the same argument as for radix-2, and each coefficient is updated in place. The fixed-bit mapping is simpler than the radix-2 parity mapping. Computing σ requires no dedicated combinational logic and uses only wiring to the d_N-th address bit. It requires no mod-3 operation or additional bank and shares the same 2-bank structure with radix-2. Therefore, a single memory structure supports both radix-2 and radix-3 layers without a mapping table or additional BRAM.

For NTRU+, which computes radix-2 layers after the radix-3 layers, the last radix-3 layer uses the radix-3 mapping for reads and the radix-2 mapping for writes. This allows the 2 phases to execute consecutively without additional latency or memory reordering.

An exception occurs in the stride-48 radix-3 layer of NTRU+864. Writing the 3 coefficients A, $A+48$, and $A+96$ directly with the parity mapping of the next radix-2 layer would map all 3 coefficients to the same bank and cause a write conflict. For this layer,

NTRU+864 therefore uses $\sigma(x) = x_{[0]} \oplus x_{[4]}$ as the write mapping. This mapping maps the 3 radix-3 outputs to the 2 banks with a $2 + 1$ distribution and provides conflict-free reads in the following stride-24 radix-2 layer. Subsequent layers use the radix-2 mapping.

Leakage Considerations. The memory-mapping logic uses only coefficient indices and public layer parameters, including the radix, stride, and block size. This also applies to the NTRU+864 write mapping. As described in Section 3.2.4, each layer descriptor is processed in fixed 5 cycles regardless of scheme or layer, including mode transitions. Since the descriptor inputs are public, the memory-mapping control introduces no secret-dependent timing or power leakage.

6 Implementation Results and Evaluation

6.1 Evaluation Methodology

Prior evaluations of NTT/INTT accelerators have focused on the area and latency of the computation core itself, or have targeted configurations tightly coupled to a processor. In a real embedded SoC, however, a loosely coupled configuration that keeps the existing software structure and invokes the programmable logic (PL) accelerator is easier to adopt, and there the end-to-end performance depends not only on the core execution time but also on the data movement between the processing system (PS) and PL, the control register accesses, and the completion wait. We therefore evaluate the design from two perspectives. First, we compare the actual execution time from software call to completion in a loosely coupled environment on a Zynq-7000 SoC (Section 6.2). Second, we compare the area and latency of the NTT/INTT core alone, with the DMA and software overhead removed, on an Artix-7 FPGA (Section 6.3). We distinguish three latency measures. The core latency is the cycle count of the NTT/INTT computation itself; the full latency is the total PL cycle count, adding to the core latency the cycles for coefficient transfer over AXI-Stream; and the HW/SW performance is the actual execution time measured from the function call on the PS to completion, including DMA setup, bus transfer, control register accesses, and the completion wait. The proposed hardware–software co-design is evaluated on the Zynq-7000 SoC (XC7Z020CLG484-1) platform, realized on a ZedBoard. The hardware accelerators are synthesized and implemented using Vivado 2020.2, while the software components are integrated via Vitis 2020.2 and executed on the ARM Cortex-A9 processor. For the standalone NTT/INTT core comparison, the accelerator is additionally synthesized on an Artix-7 FPGA (XC7A200TFBG484-2).

6.2 HW/SW Co-Design Performance

Our loosely coupled environment is implemented as an HW/SW co-design in which software running on the Zynq PS invokes the NTT/INTT module in the PL. Control registers are accessed through AXI4-Lite, and data is transferred through the MM2S and S2MM channels of the AXI DMA over a 64-bit AXI4-Stream interface. Table 3 compares the execution time of our software reference NTT/INTT code and HW/SW co-design with Zynq-based baselines using the latency measures defined in the previous subsection. We limit the baselines to loosely coupled HW/SW co-designs in which a Zynq PS invokes a PL accelerator through standard AXI interfaces and the reported execution time includes data transfer. All baselines use the same Cortex-A9 PS at 666 MHz. We use the software execution time reported in [MCL+23], which is close to our software baseline. Since [DCLR22] does not report a software baseline, we use the execution time of the reference implementation measured on the same PS in this work.

Table 3: Execution time comparison of Zynq-family HW/SW co-designs.

Design	Scheme	Operation	SW/HW (MHz)	HW cycle	SW (μs)	SW–HW (μs)	Speedup
This work	HAETAE	NTT	666/133	817	177.1	12.2	14.5×
	NTRU+768		666/133	2,141	644.6	25.5	25.3×
	NTRU+864		666/133	2,533	931.6	28.9	32.2×
	NTRU+1,152		666/133	3,229	1,240.0	35.3	35.1×
	ML-KEM		666/133	733	165.1	11.5	14.4×
	ML-DSA		666/133	945	175.5	14.6	12.0×
[DCLR22]	Kyber		666/100	2,072	165.1	37.2	4.4×
[MCL+23]	Dilithium		666/150	1,405	177.6	15.6	11.4×
This work	HAETAE	INTT	666/133	817	231.6	12.2	19.0×
	NTRU+768		666/133	2,141	784.5	25.5	30.8×
	NTRU+864		666/133	2,533	969.6	28.9	33.6×
	NTRU+1,152		666/133	3,229	1,297.3	35.3	36.8×
	ML-KEM		666/133	733	274.2	11.4	24.1×
	ML-DSA		666/133	945	227.1	14.6	15.6×
[DCLR22]	Kyber		666/100	2,075	274.2	37.2	7.4×
[MCL+23]	Dilithium		666/150	1,405	227.4	15.6	14.6×
This work	SMAUG-T	polynomial mul	666/133	–	1,061.4	160.6	6.6×

SW/HW (MHz): PS/PL clock frequencies; HW cycle: full PL latency.
Speedup: SW(μs) / SW–HW(μs).

The SMAUG-T result is not a comparison of individual NTT/INTT kernels. The SW result is the reference Toom–Cook multiplication, whereas the SW–HW result is an NTT-based multiplication that reuses the ML-DSA datapath. The pointwise multiplication between the HW NTT and INTT uses the ML-DSA Montgomery multiplier. The measured execution time also includes the final multiplication by the Montgomery constant and the reduction. SMAUG-T achieves a 6.6× speedup. For the NTT/INTT operations alone, the degree-256 transforms of ML-KEM, ML-DSA, and HAETAE achieve speedups of 12–24×, while NTRU+ achieves 25.3–36.8×. As the polynomial degree increases, DMA and software invocation account for a smaller fraction of the total execution time. Supporting NTRU+ also increases the butterfly latency. This additional latency is required for NTRU+ but becomes an overhead for the other radix-2 operations.

[DCLR22] is a Kyber implementation with a PS–AXI-MM–PL path on MicroZed/Zynq-7000, and in the ML-KEM/Kyber comparison the difference in the transfer path directly accounts for the resulting difference in latency. It uses a 32-bit AXI-MM transfer that packs two 16-bit coefficients into one word and reads four 16-bit words at once internally through a dual-port BRAM. Our design, in contrast, packs four ML-KEM coefficients into a single 64-bit AXI4-Stream transfer, reducing the number of stream transfers for the same degree-256 NTT/INTT. In addition, the PL clock rises from 100 MHz to 133 MHz, so our design shows 11.5/11.4 μs, shorter than the 2,072/2,075 cycle NTT/INTT and 37.2/37.2 μs HW/SW performance of [DCLR22].

[MCL+23] is a Dilithium implementation using a Zynq-7000 Cortex-A9 with AXI-DMA. In the ML-DSA/Dilithium comparison, the performance difference comes from butterfly parallelism rather than the bus width. It also uses 64-bit AXI-DMA transfers, so packing 2 Dilithium coefficients into 1 transfer is similar to our ML-DSA path. However, its NTT/INTT module uses 1 butterfly unit and requires 1,405 cycles for both NTT and INTT. Our design uses a 2-butterfly unified path, reducing the ML-DSA core latency to 672 cycles. The full PL latency, including reads and writes, is 945 cycles. The PL clock in [MCL+23] is 150 MHz, whereas our co-design operates at 133 MHz. The synthesizable PL frequency on Zynq is limited, while the standalone hardware design reaches approximately 140 MHz.

Table 4: Comparison of ML-KEM/Kyber and ML-DSA/Dilithium NTT/INTT hardware on the Artix-7 FPGA.

Design	Target	LUT	FF	BRAM	DSP	Cycles	Freq. (MHz)	Time (μs)	SEC (10^3)	ATP (10^3)	Ratio
Crypto-agile (ML-KEM, ML-DSA, HAETAE, NTRU+, SMAUG-T)											
This work	**ML-KEM**	**5,279**	**3,601**	**9**	**4**	**588**	**141.5**	**4.16**	**4.0**	**16.5**	**1**
This work	**ML-DSA**	**5,279**	**3,601**	**9**	**4**	**672**	**141.5**	**4.75**	**4.0**	**18.8**	**1**
Crypto-agile (Kyber, Dilithium)											
[MR24a]-U1	Kyber	1,315	1,280	4.5	2	448	263	1.70	1.6	2.7	÷6.1
[MR24a]-U2	Dilithium	3,105	2,389	4.5	4	512	200	2.56	2.4	6.1	÷3.1
Single-scheme dedicated											
[AB24]	Kyber	590	671	7.5	0	458	342	1.34	1.7	2.3	÷7.1
[GL24]	Kyber	784	441	3	2	313	200	1.57	1.1	1.6	÷10.0
[LSG22]	Dilithium	524	759	1	17	536	311	1.72	2.1	3.7	÷5.1
[GJCJ23]	Dilithium	2,759	2,037	7	4	614	163	3.77	2.7	10.3	÷1.8

Therefore, the final HW/SW execution-time difference is smaller than the cycle-count difference. Our design requires 14.6 μs, compared with the 15.6 μs reported in [MCL+23].

For NTRU+ there is no direct baseline under the same Zynq PS–PL conditions, but achieving up to a 36.8× speedup over software while executing radix-3 and trinomial layers on a single core demonstrates the hardware acceleration benefit of supporting radix-3 and trinomial transforms.

6.3 NTT/INTT Core Comparison

6.3.1 ML-KEM and ML-DSA NTT/INTT Core Comparison on Artix-7

In this comparison, our goal is not to achieve the best ATP for a fixed ML-KEM or ML-DSA NTT, but to quantify the area and time cost of supporting different transform types on a single small core. We restrict the comparison to Artix-7 implementations that report both resource usage and operating frequency and have the same 2-butterfly parallelism as our design. Table 4 separates the compared designs according to their supported schemes into single-scheme dedicated and crypto-agile implementations.

Prior Artix-7 studies use different slice-equivalent equations to compare FPGA resource usage. [AB24] uses $\text{Slices} + 120 \cdot \text{DSP} + 196 \cdot \text{BRAM}$, while other works approximate one DSP and one 36-Kb BRAM as 102.4 and 196.2 slices, and use rounded weights of 100 and 200 [LFK+19, LDL+26, RLC+26]. A 7-series slice contains 4 LUTs and 8 FFs [AMD25]. Based on these resource ratios, we calculate SEC for all entries in Table 4 as

$$\text{SEC} = \left\lfloor \frac{\text{LUT}}{4} \right\rfloor + \left\lfloor \frac{\text{FF}}{8} \right\rfloor + 200 \cdot \text{BRAM} + 100 \cdot \text{DSP}, \qquad \text{ATP} = \text{SEC} \times \text{Time}\ (\mu s).$$

Crypto-agility requires coefficient memory for the maximum N, datapath width for the largest supported modulus, arithmetic paths for the supported radices and modes, zeta memory, and control logic. For example, Sapphire [BUC19] uses 39 KB of polynomial and NTT-constant memory, which decreases to 8.53 KB when configured only for NewHope-512.

As Table 4 shows, hardware agility requires more resources and a higher ATP than a single-scheme dedicated NTT accelerator. Compared with [MR24a], which supports only ML-KEM and ML-DSA, our ATP overhead is 6.1× for ML-KEM and 3.1× for ML-DSA. Compared with the low-resource Kyber-only cores in [AB24, GL24], the overhead increases to 7.1–10.0×. Compared with Dilithium-only 2-butterfly cores, it is 1.8–5.1×, depending on the implementation.

Table 5: Comparison of NTRU+ mixed-radix and radix-2 NTT implementations on Artix-7.

Design	N	BUT	LUT	FF	BRAM	DSP	Cycles	Freq. (MHz)	Time (μs)	SEC (10^3)	ATP (10^3)	Ratio
This Work	**1,152**	**2**	**5,279**	**3,601**	**9**	**4**	**2,636**	**141.5**	**18.63**	**4.0**	**73.9**	**1**
This Work	**864**	**2**	**5,279**	**3,601**	**9**	**4**	**2,084**	**141.5**	**14.73**	**4.0**	**58.5**	–
This Work	**768**	**2**	**5,279**	**3,601**	**9**	**4**	**1,740**	**141.5**	**12.30**	**4.0**	**48.8**	–
cyclic*	4,096	4	4,933	7,456	8	56	6,253	193.4	32.33	9.4	302.8	4.10×
negacyclic*	4,096	4	5,377	7,432	8	56	6,253	189.2	33.05	9.5	313.1	4.23×
SDF†	4,096	12	8,950	5,256	8	84	8,298	111.2	74.62	12.9	962.2	13.0×
MDC†	4,096	12	12,283	7,348	8	84	4,286	110.7	38.72	14.0	541.6	7.32×

*: [KHMR24], †: [HMR24].
BUT : the number of butterfly units.
Ratio: ATP relative to This Work for NTRU+1,152.

A large part of this overhead comes from the storage structure shared by all supported schemes. Of the 9 BRAMs, 6 form the ping-pong memory that accommodates coefficients up to NTRU+1,152. This logical 4-bank memory is synthesized into 6 BRAMs because of its required depth. The remaining 3 BRAMs store the zetas for the supported schemes. These 9 BRAMs account for 45.4% of the total SEC. The reported area therefore includes the shared storage and the latency and area costs of the radix-3 path, in addition to the ML-KEM or ML-DSA computation datapath. The unified butterfly also retains the arithmetic paths, data selection, and control logic for radix-3 and trinomial operations when executing ML-KEM or ML-DSA.

[MR24a] widens the Dilithium path to 24 bits and places two 12-bit Kyber coefficients on the same path, achieving high efficiency for the ML-KEM/ML-DSA combination. In contrast, our design maintains an independent Montgomery-based computation lane instead of placing 2 coefficients on a shared path, allowing it to support different moduli of up to 17 bits. This choice is less area-efficient than a design that unifies only ML-KEM and ML-DSA, but it allows a single core to process HAETAE, SMAUG-T, NTRU+768/864/1,152, and radix-3 and trinomial layers. A new scheme can also be added using only transform settings and constants if its modulus does not exceed 17 bits and its transform structure is supported. This requires no additional butterfly or modular multiplier and does not change the latency of the supported operation modes.

The cycle-count difference is relatively small when comparing designs with the same 2-butterfly parallelism. Our ML-KEM core requires 588 cycles, which is about 28–31% more than the 458 cycles in [AB24] and the 448 cycles of [MR24a]-U1. Our ML-DSA core requires 672 cycles, which is 25% more than the 536 cycles in [LSG22] and 9% more than the 614 cycles in [GJCJ23]. The 313 cycle result in [GL24] is specialized to a Kyber-only split-radix operation order. The unified datapath executes two radix-2 butterflies in parallel for ML-KEM and ML-DSA. The additional cycles result from the longer pipeline latency of the unified radix-2, radix-3, and trinomial butterfly and the iterative architecture used for mixed-radix processing. This mixed-radix capability is evaluated for NTRU+ in the following subsection, where the proposed design achieves 4.1–13.0× lower ATP than the radix-2-only implementations.

6.3.2 Mixed-Radix and Radix-2 Implementations for NTRU+

Processing all 3 NTRU+ parameter sets, 768, 864, and 1,152, on a single radix-2 core requires a power-of-two transform of length $N = 4{,}096$. The lifting prime q' must satisfy the coefficient bound $q' > Nq^2/2$ ($N = 1{,}152, q = 3{,}457$). We therefore select the 33-bit prime $q' = 6{,}883{,}762{,}177$, which is close to the bound 6,883,689,024. This transform can

use either a cyclic or negacyclic factor polynomial. Because Proteus does not support the cyclic mode, we synthesize only its negacyclic configuration. For OpenNTT, we synthesize both configurations. They require the same cycle count and the same numbers of DSPs and BRAMs. Their LUT and FF counts differ only slightly because of differences in address generation, twiddle scheduling and control, and supporting logic. Table 5 compares our mixed-radix design with the OpenNTT configurations and the single-path delay feedback (SDF) and multi-path delay commutator (MDC) configurations of Proteus.

The 12 radix-2 layers for $N = 4{,}096$ require 24,576 butterfly operations. The cyclic and negacyclic OpenNTT configurations execute 4 butterfly units in parallel and require 6,253 cycles. The Proteus SDF configuration connects 12 streaming stages but accepts only 1 coefficient per cycle, resulting in 8,298 cycles. Its MDC configuration uses 2 coefficient lanes and requires 4,286 cycles. In comparison, our design completes the mixed-radix schedule for NTRU+1,152 in 2,636 cycles, or 18.63 μs at 141.5 MHz. It is 1.74× faster than the cyclic OpenNTT configuration, 4.01× faster than Proteus SDF, and 2.08× faster than Proteus MDC.

The radix-2-only transform with $N = 4{,}096$ requires additional hardware resources regardless of the implementation. Its twiddle ROM is approximately 3.6× larger than that for $N = 1{,}152$. The 33-bit lifting prime q' also requires a wider modular multiplier than the 12-bit NTRU+ modulus used in our design. This difference is particularly clear in the DSP count. The cyclic and negacyclic OpenNTT configurations use 56 DSPs for 4 butterflies and 4 twiddle-generation paths, each of which uses 7 DSPs. The Proteus SDF and MDC configurations use 84 DSPs because each of their 12 radix-2 stages contains a 7-DSP butterfly.

The BRAM count is determined by the coefficient and twiddle storage required for each transform length. OpenNTT uses 8 BRAMs for its 2-bank ping-pong memory and twiddle ROMs assigned to the butterfly units. Proteus also uses 8 BRAMs for the FIFOs and twiddle ROMs in its SDF and MDC stages. The results in Table 5 show the hardware cost of increasing both N and q' for a radix-2-only transform. In terms of SEC, our design is 2.36× smaller than cyclic OpenNTT, 3.25× smaller than Proteus SDF, and 3.52× smaller than Proteus MDC. It also has the lowest ATP of 73,938 SEC·μs. This ATP is 4.10× lower than that of cyclic OpenNTT, 7.32× lower than that of Proteus MDC, and 13.0× lower than that of Proteus SDF. The radix-2-only approaches also require post-processing to map the result back to the original ring and reduce the coefficients modulo the original prime q. Including this post-processing would further increase their area and execution time. These results show that a radix-2-only architecture is inefficient for the NTRU+ family and motivate unified hardware support for mixed-radix transforms, particularly radix-3.

Acknowledgments

We are deeply grateful to the editor and the anonymous reviewers for their helpful comments during the review process. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1C1C1013368).

References

[AB24] Abdullah Alhassani and Mohammed Benaissa. High-speed polynomials multiplication HW accelerator for CRYSTALS-Kyber. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(12):6105–6113, 2024.

[ACC+22] Amin Abdulrahman, Jiun-Peng Chen, Yu-Jia Chen, Vincent Hwang, Matthias J. Kannwischer, and Bo-Yin Yang. Multi-moduli NTTs for Saber on Cortex-M3

and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):127–151, 2022.

[AMD25] AMD. *7 Series FPGAs Configurable Logic Block User Guide (UG474)*. Advanced Micro Devices, Inc., April 2025. Version 1.9.

[AMI+23] Aikata Aikata, Ahmet Can Mert, Malik Imran, Samuel Pagliarini, and Sujoy Sinha Roy. KaLi: A crystal for post-quantum security using Kyber and Dilithium. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(2):747–758, 2023.

[AMJ+23] Aikata Aikata, Ahmet Can Mert, David Jacquemin, Amitabh Das, Donald Matthews, Santosh Ghosh, and Sujoy Sinha Roy. A unified cryptoprocessor for lattice-based signature and key-exchange. *IEEE Transactions on Computers*, 72(6):1568–1580, 2023.

[AMR23] Harish Prasad Allam, Suraj Mandal, and Debapriya Basu Roy. A comparative analysis between Karatsuba, Toom-Cook and NTT multiplier for polynomial multiplication in NTRU on FPGA. In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST 2023)*, pages 1–6. IEEE, 2023.

[BAA23] Luke Beckwith, Abubakr Abdulgadir, and Reza Azarderakhsh. A flexible shared hardware accelerator for NIST-recommended algorithms CRYSTALS-Kyber and CRYSTALS-Dilithium with SCA protection. In *Topics in Cryptology – CT-RSA 2023*, volume 13871 of *Lecture Notes in Computer Science*, pages 469–490. Springer, 2023.

[Bar87] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1987.

[BUC19] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(4):17–61, 2019.

[CCC+26] Jung Hee Cheon, Hyeongmin Choe, Joongeun Choi, Dongyeon Hong, Jeongdae Hong, Chi-Gon Jung, Honggoo Kang, Minhyeok Kang, Taekyung Kim, Jeongbeen Ko, Janghyun Lee, Seonghyuck Lim, Aesun Park, Seunghwan Park, Jungjoo Seo, Hyoeun Seong, Junbum Shin, and MinJune Yi. SMAUG-T: The key exchange algorithm based on Module-LWE and Module-LWR. Algorithm specification, version 260521, KpqC competition. https://kpqc.cryptolab.co.kr/smaug-t, May 2026.

[CCD+24] Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Güneysu, Dongyeon Hong, Markus Krausz, Georg Land, Marc Möller, Damien Stehlé, and MinJune Yi. HAETAE: Shorter lattice-based Fiat–Shamir signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):25–75, 2024.

[CDH+20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Tsunekazu Saito, Peter Schwabe, William Whyte, Keita Xagawa, Takashi Yamakawa, and Zhenfei Zhang. NTRU: Algorithm specifications and supporting documentation. Submission to the NIST Post-Quantum Cryptography Standardization Project, round 3. https://ntru.org/, September 2020.

[CHK+21] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):159–188, 2021.

[CHK+26] Jihoon Cho, Jincheol Ha, Seongkwang Kim, Jihoon Kwon, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Sangyub Lee, Dukjae Moon, Mincheol Son, and Hyojin Yoon. The AIMer signature scheme. Algorithm specification, version 26.Jan.30, KpqC competition. https://aimer-signature.org/docs/AIMer_Specification_v260130.pdf, January 2026.

[CYY+22] Xiangren Chen, Bohan Yang, Shouyi Yin, Shaojun Wei, and Leibo Liu. CFNTT: Scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):94–126, 2022.

[DCLR22] Vinícius L. R. Da Costa, Ândrei Camponogara, Julio López, and Moisés V. Ribeiro. The feasibility of the CRYSTALS-Kyber scheme for smart metering systems. *IEEE Access*, 10:131303–131317, 2022.

[DMG23] Viet Ba Dang, Kamyar Mohajerani, and Kris Gaj. High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber. *IEEE Transactions on Computers*, 72(2):306–320, 2023.

[DMÖS22] Kemal Derya, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. CoHA-NTT: A configurable hardware accelerator for NTT-based polynomial multiplication. *Microprocessors and Microsystems*, 89:104451, 2022.

[DPM+25] Alessandra Dolmeta, Valeria Piscopo, Guido Masera, Maurizio Martina, and Michael Hutter. HORCRUX – a lightweight PQC-RISC-V eXtension architecture. Cryptology ePrint Archive, Paper 2025/1934, 2025.

[FHK+20] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. Submission to the NIST Post-Quantum Cryptography Standardization Project, version 1.2. https://falcon-sign.info/, 2020.

[GAA+26] Philippe Gaborit, Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jérôme Lacan, Jean-Marc Robert, Pascal Véron, Paulo L. Barreto, Santosh Ghosh, Shay Gueron, Tim Güneysu, Rafael Misoczki, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, and Valentin Vasseur. Hamming quasi-cyclic (HQC). Algorithm specification, version of October 12, 2026. https://pqc-hqc.org/, October 2026.

[GJCJ23] Naina Gupta, Arpan Jati, Anupam Chattopadhyay, and Gautam Jha. Lightweight hardware accelerator for post-quantum digital signature CRYSTALS-Dilithium. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(8):3234–3243, 2023.

[GL24] Wenbo Guo and Shuguo Li. Split-radix based compact hardware architecture for CRYSTALS-Kyber. *IEEE Transactions on Computers*, 73(1):97–108, 2024.

[GM94] Torbjörn Granlund and Peter L. Montgomery. Division by invariant integers using multiplication. In *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation (PLDI)*, pages 61–72. ACM, 1994.

[HMR24] Florian Hirner, Ahmet Can Mert, and Sujoy Sinha Roy. Proteus: A pipelined NTT architecture generator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 32(7):1228–1238, 2024.

[HZZ+22] Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray C. C. Cheung, Çetin Kaya Koç, and Donglong Chen. Improved Plantard arithmetic for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):614–636, 2022.

[HZZ+24] Junhao Huang, Haosong Zhao, Jipeng Zhang, Wangchen Dai, Lu Zhou, Ray C. C. Cheung, Çetin Kaya Koç, and Donglong Chen. Yet another improvement of Plantard arithmetic for faster Kyber on low-end 32-bit IoT devices. *IEEE Transactions on Information Forensics and Security*, 19:3800–3813, 2024.

[Kee21] Kris Keersmaekers. A compact and flexible hardware accelerator for NTRU polynomial multiplication using NTT. Master's thesis, KU Leuven, 2021. https://cosicdatabase.esat.kuleuven.be/backend/publications/files/these/412.

[KHMR24] Florian Krieger, Florian Hirner, Ahmet Can Mert, and Sujoy Sinha Roy. OpenNTT – an automated toolchain for compiling high-performance NTT accelerators in FHE. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2024)*, pages 1–9. ACM, 2024.

[KMSH24] Dur-e-Shahwar Kundi, Jose M. Bermudo Mera, Pierre-Yves Strub, and Michael Hutter. High-performance NTT hardware accelerator to support ML-KEM and ML-DSA. In *Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security (ASHES 2024)*, pages 100–105. ACM, 2024.

[KSK+25] MinGi Kim, DongHyun Shin, WooHyung Ko, YoungBeom Kim, and Seog Chung Seo. Optimized implementation of SMAUG-T on resource-constrained 16-bit MSP430 MCU. *ICT Express*, 11(5):851–857, 2025.

[KvVBS26] YoungBeom Kim, Christine van Vredendaal, Gustavo Banegas, and Seog Chung Seo. Low-stack HAETAE for memory-constrained microcontrollers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(4):293–315, 2026.

[LDL+26] Qiuzhu Liao, Gaoming Du, Zhenmin Li, Xiaolei Wang, Yukun Song, and Duoli Zhang. A configurable and resource-efficient polynomial multiplier for CRYSTALS-Kyber. *IEICE Transactions on Electronics*, E109-C(1):1–11, 2026.

[LFK+19] Weiqiang Liu, Sailong Fan, Ayesha Khalid, Ciara Rafferty, and Máire O'Neill. Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(10):2459–2463, 2019.

[LN16] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *Cryptology and Network Security – CANS 2016*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2016.

[LSG22] Georg Land, Pascal Sasdrich, and Tim Güneysu. A hard crystal – implementing Dilithium on reconfigurable hardware. In *Smart Card Research and Advanced Applications – CARDIS 2021*, volume 13173 of *Lecture Notes in Computer Science*, pages 210–230. Springer, 2022.

[MCL+23] Gaoyu Mao, Donglong Chen, Guangyan Li, Wangchen Dai, Abdurrashid Ibrahim Sanka, Çetin Kaya Koç, and Ray C. C. Cheung. High-performance and configurable SW/HW co-design of post-quantum signature CRYSTALS-Dilithium. *ACM Transactions on Reconfigurable Technology and Systems*, 16(3):44:1–44:28, 2023.

[MR24a] Suraj Mandal and Debapriya Basu Roy. KiD: A hardware design framework targeting unified NTT multiplication for CRYSTALS-Kyber and CRYSTALS-Dilithium on FPGA. In *37th International Conference on VLSI Design and 23rd International Conference on Embedded Systems (VLSID 2024)*, pages 455–460. IEEE, 2024.

[MR24b] Suraj Mandal and Debapriya Basu Roy. Winograd for NTT: A case study on higher-radix and low-latency implementation of NTT for post quantum cryptography on FPGA. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(12):6396–6409, 2024.

[Nat24a] National Institute of Standards and Technology. Module-lattice-based digital signature standard. Federal Information Processing Standards Publication FIPS 204, National Institute of Standards and Technology, August 2024.

[Nat24b] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Federal Information Processing Standards Publication FIPS 203, National Institute of Standards and Technology, August 2024.

[Nat24c] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Federal Information Processing Standards Publication FIPS 205, National Institute of Standards and Technology, August 2024.

[PK26] Jong Hwan Park and Jonghyun Kim. NTRU+: Compact construction of NTRU using simple encoding method. Algorithm specification, final version of the KpqC competition. https://data.ntruplus.org/NTRU%2B_20260710.pdf, July 2026.

[RLC+26] Antonio Ras, Antoine Loiseau, Mikaël Carmona, Simon Pontié, Guénaël Renault, Benjamin Smith, and Emanuele Valea. PHOENIX: Crypto-agile hardware sharing for ML-KEM and HQC. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(3):1228–1255, 2026.

[RMD+15] Stephen Richardson, Dejan Marković, Andrew Danowitz, John Brunhaver, and Mark Horowitz. Building conflict-free FFT schedules. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(4):1146–1155, 2015.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. Cryptology ePrint Archive, Paper 2018/039, 2018.

[SKA22] Kyung-Ah Shim, Jeongsu Kim, and Youngjoo An. NCC-Sign: A new lattice-based signature scheme using non-cyclotomic polynomials. Submission to the Korean Post-Quantum Cryptography Competition (KpqC), round 1. https://kpqc.or.kr/images/pdf/NCC-Sign.pdf, 2022.

[TJS03] Jarmo H. Takala, Tuomas S. Järvinen, and Harri T. Sorokin. Conflict-free parallel memory access scheme for FFT processors. In *Proceedings of the 2003 IEEE International Symposium on Circuits and Systems (ISCAS 2003)*, volume 4, pages IV–524–IV–527. IEEE, 2003.

[WZZ+24] Tengfei Wang, Chi Zhang, Xiaolin Zhang, Dawu Gu, and Pei Cao. Optimized hardware-software co-design for Kyber and Dilithium on RISC-V SoC FPGA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):99–135, 2024.

[XHY+20] Guozhu Xin, Jun Han, Tianyu Yin, Yuchao Zhou, Jianwei Yang, Xu Cheng, and Xiaoyang Zeng. VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(8):2672–2684, 2020.

[ZYC+20] Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu. Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):49–72, 2020.