Article

Deeply Pipelined NTT Accelerator with Ping-Pong Memory and LUT-Only Barrett Reduction for Post-Quantum Cryptography

Omar S. Sonbul [1], Muhammad Rashid [1,*], Muhammad I. Masud [2], Mohammed Aman [3] and Amar Y. Jaffar [1]

[1] Computer and Network Engineering Department, Umm Al-Qura University, Makkah 24382, Saudi Arabia; ossonbul@uqu.edu.sa (O.S.S.); ayjaafar@uqu.edu.sa (A.Y.J.)
[2] Department of Electrical Engineering, College of Engineering, University of Business and Technology, Jeddah 21361, Saudi Arabia; m.masud@ubt.edu.sa
[3] Department of Industrial Engineering, College of Engineering, University of Business and Technology, Jeddah 21361, Saudi Arabia; m.aman@ubt.edu.sa
* Correspondence: mfelahi@uqu.edu.sa

Abstract

Lattice-based post-quantum cryptography relies on fast polynomial multiplication. The Number-Theoretic Transform (NTT) is the key operation that enables this acceleration. To provide high throughput and low latency while keeping the area overhead small, hardware implementations of the NTT is essential. This is particularly true for resource-constrained devices. However, existing NTT accelerators either achieve high throughput at the cost of large area overhead or provide compact designs with limited pipelining and low operating frequency. Therefore, this article presents a compact, seven-stage pipelined NTT accelerator architecture for post-quantum cryptography, using the CRYSTALS–Kyber algorithm as a case study. The CRYSTALS–Kyber algorithm is selected due to its NIST standardization, strong security guarantees, and suitability for hardware acceleration. Specifically, a unified three-stage pipelined butterfly unit is designed using a single DSP48E1 block for the required integer multiplication. In contrast, the modular reduction stage is implemented using a four-stage pipelined, lookup-table (LUT)-only Barrett reduction unit. The term "LUT-only" refers strictly to the reduction logic and not to the butterfly multiplication. Furthermore, two dual-port BRAM18 blocks are used in a ping-pong manner to hold intermediate and final coefficients. In addition, a simple finite-state machine controller is implemented, which manages all forward NTT (FNTT) and inverse NTT (INTT) stages. For validation, the proposed design is realized on a Xilinx Artix-7 FPGA. It uses only 503 LUTs, 545 flip-flops, 1 DSP48E1 block, and 2 BRAM18 blocks. The complete FNTT and INTT with final rescaling require 1029 and 1285 clock cycles, respectively. At 200 MHz, these correspond to execution times of 5.14 µs for the FNTT and 6.42 µs for the INTT.

Keywords: NTT acceleration; CRYSTALS–Kyber; PQC algorithms; DSP48E1; Barrett modular reduction; Cooley–Tukey; Gentleman–Sande; FPGA

Academic Editor: Hung-Yu Chien

Received: 21 December 2025
Revised: 22 January 2026
Accepted: 23 January 2026
Published: 25 January 2026

1. Introduction

Public key cryptosystems rely on the difficulty of mathematical problems like integer factorization and discrete logarithms [1,2]. The integer factorization and discrete logarithm problems are infeasible to solve efficiently using classical computing. However, the development of quantum algorithms poses a direct threat to their long-term security [3]. Rapid progress in quantum hardware, especially quantum processors [4–6], has been remarkable.

This advancement has transformed the computational landscape. Consequently, there is an increasing need to adopt cryptographic mechanisms that remain secure even against quantum-capable adversaries.

To confront this challenge, the National Institute of Standards and Technology (NIST) began a multi-year standardization effort in 2017. Its goal was to establish secure post-quantum cryptographic (PQC) standards [7]. After several evaluation rounds, NIST finalized its selections for PQC in 2024. The chosen schemes were CRYSTALS–Kyber [8], CRYSTALS–Dilithium [9], Falcon [10], and SPHINCS+ [11]. While SPHINCS+ is a hash-based scheme, the remaining candidates are lattice-based constructions [12]. The lattice-based schemes depend on structured polynomial arithmetic. This reliance allows them to provide strong security while maintaining high performance. In addition, they are suitable for hardware acceleration.

The Number-Theoretic Transform (NTT) is the key module in lattice-based schemes [13]. It converts polynomial multiplication into pointwise operations and enables high-speed execution in both software and hardware. As a result, the design of fast and compact NTT accelerators has become a significant research focus. This interest is driven by the needs of secure embedded systems and resource-limited devices [13,14]. The NTT involves two operations: (i) forward (FNTT) and inverse (INTT). The required computations in FNTT and INTT can be performed with Cooley–Tukey (CT) and Gentleman–Sande (GS) butterfly configurations, respectively. Software and hardware are the two primary platforms used to accelerate CT and GS performance. While software implementations provide flexibility with limited computational capability, hardware implementations deliver higher security and maximized throughput.

1.1. Existing NTT Hardware Accelerators

Hardware realizations of the FNTT and INTT typically target Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). These platforms provide the parallelism, configurability, and performance required to efficiently execute the intensive arithmetic operations involved in NTT-based computations. The computations are performed using Cooley–Tukey and Gentleman–Sande butterfly operations [1,2,13–34].

Given the computational intensity of Cooley–Tukey and Gentleman–Sande butterfly operations, resource sharing has become a primary optimization strategy. This approach reduces hardware cost while maintaining high throughput [15,16]. In Ref. [15], the authors implement the NTT butterfly using custom resource-sharing techniques to reduce hardware costs. Their butterfly design contains a multiplier, a reduction unit, and an adder/subtractor. They also use pipelining to shorten the critical path and improve overall performance. A similar resource-sharing strategy is employed in Ref. [16]. The authors present a configurable hardware architecture for a unified, parameterized NTT-based polynomial multiplier. This design supports a variety of PQC alternatives, including algorithms such as CRYSTALS–Kyber.

While resource sharing focuses on reducing area and cost, another major line of work targets high-speed and parallel NTT acceleration [2,14,17–22]. A high-throughput and BRAM-efficient FNTT/INTT accelerator architecture for CRYSTALS–Kyber is implemented in Ref. [2]. Moreover, a high-speed NTT accelerator architecture is described in Ref. [17], where a novel butterfly unit and an efficient modular polynomial multiplier are introduced. Their work uses a radix-4-based configurable NTT design to compute both FNTT and INTT on a unified architecture. Another example of this trend for CRYSTALS–Kyber is presented in Ref. [18]. Their design uses one multiplier, one reduction unit, and an adder/subtractor, along with two "DIVby2" blocks that remove the need to multiply coefficients after the INTT. To further reduce latency, up to 16 butterfly units operate in parallel. Similarly, 64 butterfly units are incorporated in the design of Ref. [19] for FNTT and INTT computations.

In addition to increasing the number of butterfly units, several works pursue fully paralleled, deeply pipelined architectures to further minimize latency [14,20–22]. For example, the work in Ref. [20] achieves ultra-low FNTT/INTT latency by fully parallelizing coefficient processing. It supplies all polynomial coefficients to the NTT stages simultaneously, thereby removing RAM bandwidth bottlenecks. Moreover, the architecture exploits the constant geometric structure of the NTT to simplify interconnect routing and uses the K^2-RED modular reduction for low-cost arithmetic operations. Twiddle-factor storage is shared between the forward and inverse NTT to reduce the memory footprint. In Ref. [21], a compact modular reduction unit is proposed that requires only 52 lookup tables (LUTs) and 1 digital signal processor (DSP) block. This design also incorporates a four-way parallel NTT architecture based on radix-2 butterfly units. The work in Ref. [14] employs a dedicated butterfly unit for FNTT and INTT computations. Finally, the work in Ref. [22] incorporates several modular operators to support high-throughput NTT computation.

In addition to resource-sharing and parallel/pipelined designs, a third major optimization direction focuses on efficient memory-access schemes, particularly ping-pong addressing [13,23]. A ping-pong memory access scheme is used in Ref. [13] to implement the pipelined architecture for the CRYSTALS–Kyber NTT. The design uses two BRAMs (Block Random-Access Memories) and one BROM (Block Read-Only Memory) as its memory components. A single butterfly unit alternates between them using a ping-pong access pattern. Another ping-pong memory access scheme is used in Ref. [23] to implement the CRYSTALS–Kyber NTT. In that design, two BRAMs and one BROM serve as the main memory elements. It employs two unified butterfly units, each with a five-stage pipeline to reduce the critical path delay.

Beyond memory-access scheduling, several works also incorporate Barrett-based reduction and unified butterfly designs [24–28]. For example, the design in Ref. [24] presents an FNTT architecture for CRYSTALS–Kyber using Barrett's reduction. Using the same Barrett-based reduction, Ref. [25] analyzes three Kyber NTT variants—dedicated FNTT and INTT cores and a unified design. Using the same Barrett reduction module, the unified NTT accelerator in Ref. [26] employs a single adder, multiplier, and subtractor for both CT and GS butterflies. The configurable design in Ref. [27] similarly adopts a unified approach, integrating a radix-4 butterfly with ten BRAMs under a parallel ping-pong control scheme. In Ref. [28], a pipelined CRYSTALS–Kyber NTT is implemented using a ping-pong memory scheme and a single butterfly unit. The design also incorporates a shift-add Barrett reduction module to minimize multiplication overhead.

Several recent works extend unified butterfly and NTT architectures to support both CRYSTALS–Kyber and CRYSTALS–Dilithium [29–32]. For example, the integer multiplier unit in Ref. [29] is specifically designed for CRYSTALS–Dilithium but can also be used for CRYSTALS–Kyber. It carries out two 23-bit $\times$ 12-bit multiplications, combining the results through addition for CRYSTALS–Dilithium and through concatenation for CRYSTALS–Kyber. This enables the computation of a single coefficient in CRYSTALS–Dilithium and two coefficients in CRYSTALS–Kyber. In Ref. [30], an impressive high-speed unified NTT accelerator for CRYSTALS–Kyber and CRYSTALS–Dilithium is described. A CRYPHTOR accelerator is implemented in Ref. [31], where a unified memory addressing is employed to compute FNTT and INTT operations for CRYSTALS–Kyber and CRYSTALS–Dilithium. Recently, in Ref. [32], CRYSTALS–Kyber and CRYSTALS–Dilithium NTT are accelerated using a single Montgomery reduction-based instruction set extension on a RISC-V platform.

To address the stringent constraints of edge devices, energy-efficient and area-optimized FNTT/INTT architectures are described in [1,33,34]. In Ref. [1], an FPGA-based NTT accelerator for CRYSTALS–Dilithium is presented, which jointly optimizes area, speed, and power by combining CT and GS NTT structures with Barrett modular reduction and an iterative architecture. On an Artix-7 FPGA, the design achieves favorable area–time and power–performance trade-offs across low- and high-frequency ranges, making it well suited for both energy-efficient and high-throughput PQC applications. An energy-efficient NTT designed for FALCON key generation is presented in Ref. [33] using NTT-friendly primes and a compact Montgomery reduction scheme with reduced multiplier sizes. The architecture is optimized for ASIC implementation, resulting in lower power consumption and hardware area. Enhancements to Plantard-based arithmetic for accelerating CRYSTALS–Kyber, particularly on low-end Internet-of-Things devices, are presented in Ref. [34].

Among the NTT accelerators discussed above, our prior works appear in Ref. [1,13,14,25,28]. Reference [1] targets NTT acceleration for CRYSTALS–Dilithium, whereas the present work focuses on CRYSTALS–Kyber with different arithmetic and architectural constraints. Our earlier Kyber-oriented designs in Ref. [13,14] employed block-ROM–based twiddle storage, while this work adopts compact LUT-based ROMs optimized for low-area Artix-7 devices. In contrast to [25], which evaluates energy-efficient NTT architectures using register-file-based memories and generic Barrett reduction, the proposed design introduces a fully LUT-only pipelined Barrett reduction combined with dual-port BRAM ping-pong memory.

Although this work and our earlier design in Ref. [28] target NTT acceleration for lattice-based cryptography, the underlying architectures are fundamentally different. The accelerator in Ref. [28] employs three RegBanks to store intermediate data and twiddle factors, where the twiddle factors must be loaded before executing either the FNTT or INTT. In contrast, the proposed design replaces RegBanks with dual-port BRAMs and utilizes two dedicated LUT-based ROMs to permanently store FNTT and INTT twiddle factors, thereby eliminating twiddle reloading overhead. Moreover, unlike [28], which performs integer multiplication using the RTL "*" operator without DSP utilization, this work integrates the DSP48E1 block for multiplication. Additionally, different Barrett reduction parameters are adopted in Ref. [28] and this work to implement the modular reduction. These architectural changes clearly differentiate this work from [28]. These distinctions clearly differentiate the unique contributions of the present work, as summarized in Section 1.3.

1.2. Constraints in State-of-the-Art NTT Accelerators

Section 1.1 describes several hardware designs for NTT from the literature. Many architectures focus on minimizing latency by deploying multiple butterfly units [18,19]. Others rely on wide memory banks [12,14,25] or parallel multiply–accumulate logic [29] to further accelerate computation. These designs achieve high throughput but also introduce significant area overhead. Other works aim for compact implementations but rely on iterative modular reduction or non-pipelined multipliers. Some also adopt control-intensive architectures that ultimately limit achievable clock frequency [1,16,24]. Balancing the area–delay product continues to be difficult. The NTT engine must be small yet fast enough to meet real-time post-quantum requirements.

1.3. Proposed Work

To address the limitations discussed in Section 1.2, this work aims to develop a deeply pipelined hardware architecture for forward and inverse NTT computations. The design incorporates ping-pong memory addressing together with a LUT-only Barrett reduction scheme. The proposed design focuses exclusively on the FNTT and INTT operations, which form the core computational component in CRYSTALS–Kyber, while pointwise multiplication and higher-level Kyber procedures are considered outside the scope of this study. The salient features of the proposed design are as follows:

- **Seven-Stage Pipelined NTT/INTT Architecture:** We design a unified and deeply pipelined data path for CRYSTALS–Kyber that integrates a three-stage butterfly with a four-stage Barrett reduction unit. This pipeline sustains one butterfly result per clock cycle once it is fully filled.
- **Compact Unified Butterfly Unit of CT and GS:** We implement a unified butterfly architecture capable of executing both the Cooley–Tukey and Gentleman–Sande variants for FNTT and INTT computations. By sharing a single DSP48E1 block, the design significantly reduces hardware resources while maintaining high-speed operation.
- **Optimized LUT-only Pipelined Barrett Reduction:** We implement a four-stage pipelined Barrett reduction unit that relies exclusively on look-up tables. This approach avoids DSPs and wide combinational blocks while preserving modular arithmetic accuracy.
- **Efficient Memory Addressing:** A finite state machine (FSM) controller manages the operation of the two dual-port BRAM18 blocks in a ping-pong configuration. This coordination ensures continuous and conflict-free data flow.

1.4. Summary of Implementation Results

The proposed design is validated using only one DSP block and two BRAM18 blocks. It also requires just a small number of LUTs and pipeline registers. The proposed architecture is modeled in Verilog HDL using the Vivado IDE, and the implementation results are reported up to the post-place-and-route stage on an Artix-7 (XC7A100TCSG324-3) FPGA device. It requires 1029 clock cycles for the FNTT and 1285 clock cycles for the INTT, including the final rescaling step. Operating at 200 MHz, these correspond to execution times of 5.14 µs and 6.42 µs, respectively. Furthermore, the architecture consumes only 503 LUTs, 545 flip-flops (FFs), 1 DSP, and 2 BRAM18 blocks. These timing and area results demonstrate that the proposed NTT accelerator is well suited for lightweight post-quantum cryptographic systems and resource-constrained Internet-of-Things devices.

The article is organized as follows: Section 2 outlines the relevant mathematical foundations. Section 3 presents our NTT accelerator architecture. Section 4 reports implementation outcomes and benchmarks them against state-of-the-art solutions. Section 5 concludes the article with key insights and discusses potential avenues for future work.

2. Analysis of Mathematical Background

The FNTT and INTT operations, along with their corresponding algorithms, are described in Section 2.1. The traditional Barrett-based modular reduction method is described in Section 2.2. The butterfly configurations, along with the related arithmetic operations, are described in Section 2.3. The analysis of the mapping of mathematical concepts (described in Sections 2.1–2.3) onto hardware is illustrated in Section 2.4.

2.1. Analysis of FNTT and INTT

The NTT is the core computational primitive in CRYSTALS–Kyber to accelerate polynomial multiplication in the ring $\mathbb{Z}_q[x]/(x^{256}+1)$. In this setting, the modulus is fixed at $q = 3329$ and the transform size is $N = 256$. The FNTT maps a vector of coefficients

$a = (a_0, \ldots, a_{N-1})$ into its spectral representation under powers of a primitive 256th root of unity ζ in $\mathbb{Z}_q$. Following the Cooley–Tukey (CT) formulation, the fNTT is given by

$$FNTT = A_k = \sum_{n=0}^{N-1} a_n\, \zeta^{nk} \pmod{q}, \qquad k = 0, \ldots, N-1, \tag{1}$$

and can be computed using a sequence of radix-2 butterfly operations. The CT approach performs the NTT in a decimation-in-time (DIT) manner, where smaller subproblems are constructed by splitting the index n into even and odd parts. The CRYSTALS–Kyber reference implementation uses a stage-wise CT butterfly configuration with a fixed ordering of the precomputed twiddle factors (the so-called *zetas* array). On the other hand, the inverse transform (INTT) employs the GS butterfly configuration, which performs a decimation-in-frequency (DIF) traversal. It reconstructs the original polynomial as

$$INTT = a_n = N^{-1} \sum_{k=0}^{N-1} A_k\, \zeta^{-nk} \pmod{q}, \qquad n = 0, \ldots, N-1, \tag{2}$$

where the scaling factor N^{-1} is implemented in CRYSTALS–Kyber using multiplication by the precomputed modular inverse of 256 modulo q. Algorithm 1 describes the radix-2 CT-style forward NTT used in CRYSTALS–Kyber, while Algorithm 2 provides the complementary GS-style inverse transform.

Algorithm 1 Forward NTT with radix-2-based Cooley–Tukey butterfly.

```
Input: Coefficients $a[0..255]$
Input: Twiddle factors $\zeta[1..127]$
Output: NTT output $A[0..255]$
1:  $k \leftarrow 1$
2:  for len $\leftarrow 128, 64, \ldots, 2$ do
3:      for start $\leftarrow 0$ to 255 step $2 \cdot$ len do
4:          for $j \leftarrow$ start to start + len $- 1$ do
5:              $t \leftarrow \zeta[k] \cdot a[j + \text{len}] \bmod q$
6:              $a[j] \leftarrow a[j] + t \pmod{q}$
7:              $a[j + \text{len}] \leftarrow a[j] - t \pmod{q}$
8:              $k \leftarrow k + 1$
9:          end for
10:     end for
11: end for
12: return $a$
```

2.2. Analysis of Barrett-Based Reduction

All multiplications in the FNTT and INTT require reducing intermediate results modulo $q = 3329$. For CRYSTALS–Kyber, the operands of interest during each butterfly step are 24 bits due to the multiplication of a 12-bit coefficient with a 12-bit twiddle factor. Thus, modular reduction is essential, and this can be efficiently implemented using various approaches such as Barrett, Montgomery, and Plantard (In this work, Barrett modular multiplication is selected over the Montgomery and Plantard variants for two main reasons. First, Barrett reduction operates entirely in the standard integer domain and therefore avoids the domain-conversion steps required by Montgomery multiplication, which produces the scaled value $abR^{-1} \bmod q$ and requires explicit transformations into and out of the Montgomery domain. Second, although Plantard-style reduction is closely related to Barrett's, its efficiency depends heavily on choosing a constant with a favorable bit structure, which is not always possible for arbitrary moduli. Barrett reduction, on the other hand, works uniformly for any modulus and can always be implemented using an efficient shift–add structure. For these reasons, Barrett reduction aligns more closely with the goals of our study and forms the basis of the modular multiplication primitive used throughout this work).

Algorithm 2 Inverse NTT with radix-2-based Gentleman-Sande butterfly.

```
Input: Spectral coefficients $A[0..255]$
Input: Inverse twiddle factors $\zeta^{-1}[1..127]$
Output: Recovered polynomial $a[0..255]$
 1: $k \leftarrow 127$
 2: for len $\leftarrow 2, 4, \ldots, 128$ do
 3:     for start $\leftarrow 0$ to 255 step $2 \cdot$ len do
 4:         for $j \leftarrow$ start to start + len $- 1$ do
 5:             $u \leftarrow A[j]$
 6:             $v \leftarrow A[j + \text{len}]$
 7:             $A[j] \leftarrow u + v \pmod{q}$
 8:             $A[j + \text{len}] \leftarrow \zeta^{-1}[k] \cdot (u - v) \pmod{q}$
 9:             $k \leftarrow k - 1$
10:         end for
11:     end for
12: end for
13: Multiply each $A[j]$ by $N^{-1} \bmod q$
14: return $A$
```

The Barrett method provides an efficient, division-free procedure to reduce 24-bit resultant values in CRYSTALS–Kyber. Given a modulus q and a width parameter k, Barrett reduction introduces the constant

$$\mu = \left\lfloor \frac{2^{2k}}{q} \right\rfloor,$$

which is precomputed for the target platform. For CRYSTALS–Kyber, with $k = 12$, this evaluates to $\mu = 5039$. The traditional Barrett reduction for a signed intermediate x proceeds by approximating the quotient $t = \left\lfloor x \cdot \mu / 2^{2k} \right\rfloor$ and subtracting the scaled modulus. Algorithm 3 summarizes the classical procedure. In practice, the final result may lie in $[0, 2q)$, requiring one or two conditional subtractions to bring it into $[0, q)$.

Algorithm 3 Barrett-based reduction.

```
Input: Signed intermediate $x$
Input: Modulus $q = 3329$
Input: $\mu = \left\lfloor \frac{2^{2k}}{q} \right\rfloor = 5039$
Output: Reduced value $r = x \bmod q$
 1: $t_1 \leftarrow x \cdot \mu$
 2: $t_2 \leftarrow t_1 \gg 24$ {$24 = 2k$ for $k = 12$}
 3: $t_3 \leftarrow t_2 \cdot q$
 4: $r \leftarrow x - t_3$
 5: if $r \geq q$ then
 6:     $r \leftarrow r - q$
 7: end if
 8: if $r < 0$ then
 9:     $r \leftarrow r + q$
10: end if
11: return $r$
```

2.3. Analysis of CT and GS Butterfly Units

The butterfly operation is the fundamental building block of both the CT-based FNTT and the GS-based INTT. In FNTT, each radix-2 CT butterfly consumes a pair of coefficients (a, b) and a twiddle factor ζ, producing updated values:

$$z_0 = a + \zeta \cdot b \pmod{q}, \tag{3}$$

$$z_1 = a - \zeta \cdot b \pmod{q}. \tag{4}$$

These correspond exactly to the update steps (through lines 4 to 9) in Algorithm 1. The multiplication by ζ produces an intermediate value that is typically 24 bits wide before reduction. This product is passed through a Barrett reducer (as described in Algorithm 3) before forming the sum and difference in Equations (3) and (4). Similarly, for the inverse transform, the GS butterfly takes a pair (a, b) and applies the inverse twiddle factor ζ^{-1} in a decimation-in-frequency style. The corresponding equations are:

$$z_0 = a + b \pmod{q}, \tag{5}$$

$$z_1 = \zeta^{-1} \cdot (a - b) \pmod{q}. \tag{6}$$

These equations match the update rules (through lines 4 to 10) in Algorithm 2. The final scaling step by $N^{-1} \bmod q$ completes the INTT implementation. According to Equation (3) through Equation (6), both CT and GS butterflies share the same structural components in hardware: one modular multiplication (to apply the twiddle factor), one Barrett reduction, and two modular add/subtract operations with conditional corrections. The structural similarity between the two formulations makes it possible to implement a common butterfly unit that supports both FNTT and INTT computations.

2.4. Analysis of Hardware Complexity and DSP Mapping Considerations

Sections 2.1–2.3 present the analysis of the relevant mathematical concepts. Mapping these concepts onto hardware requires careful consideration of arithmetic complexity and resource utilization. From a hardware perspective, the dominant operation in both the CT and GS butterfly units is the modular multiplication between a coefficient and a twiddle factor. In CRYSTALS–Kyber, both operands are 12-bit wide, resulting in a 24-bit intermediate product before modular reduction. This multiplication, followed by the Barrett-based reduction, defines the critical arithmetic workload of the NTT data path. Modern Xilinx FPGAs provide dedicated DSP48E1 blocks that are well suited for such fixed-point integer multiplications. A DSP48E1 block natively supports multiplication of operands up to 25×18 bits, which comfortably accommodates the 12×12-bit multiplications required in the proposed butterfly unit. Consequently, the integer multiplication stage can be efficiently mapped to a single DSP block, while the subsequent Barrett reduction and modular addition/subtraction operations are implemented using LUT-based logic. This mapping enables a favorable balance between performance and resource utilization, allowing high clock frequencies to be achieved without excessive DSP consumption.

2.5. Summary of Symbols and Parameters

The symbols and parameters used throughout the manuscript are summarized below:

Symbol	**Description**
FNTT	Forward Number-Theoretic Transform
INTT	Inverse Number-Theoretic Transform
N	NTT size ($N = 256$)
q	Prime modulus ($q = 3329$)
a_n	Polynomial coefficient in $\mathbb{Z}_q$
ζ	Twiddle factor for FNTT
ζ^{-1}	Inverse twiddle factor for INTT
a, b	Input coefficients with 12 bits size
x	Intermediate multiplication result (up to 24 bits)
s_f	Barrett precomputed constant (20,159)
s	Barrett shift parameter ($s = 6$)
DP-BRAM-A/B	Simple Dual-port Block-RAM memory for coefficients storage
DSP48E1	FPGA block for integer multiplication

3. Design of the Proposed Accelerator

The structural overview of the proposed design is shown in Figure 1. It contains a memory unit incorporating two dual-port block RAMs (DP-BRAM-A and DP-BRAM-B) and a twiddle-factors unit, with LUT-based ζ values for both the FNTT and INTT. The architecture also includes a butterfly unit and an FSM-based controller. The memory units retain the initial data along with the intermediate values and final outputs. Based on the inputs from memory blocks along with the twiddle factors, the butterfly unit computes the required arithmetic operations for processing. The outputs generated by the butterfly unit are returned to the memory blocks for subsequent processing. The controller handles the memory unit, butterfly unit, as well as twiddle factors. It also efficiently controls the overall CT and GS computations. We further describe these blocks in the subsequent sections.

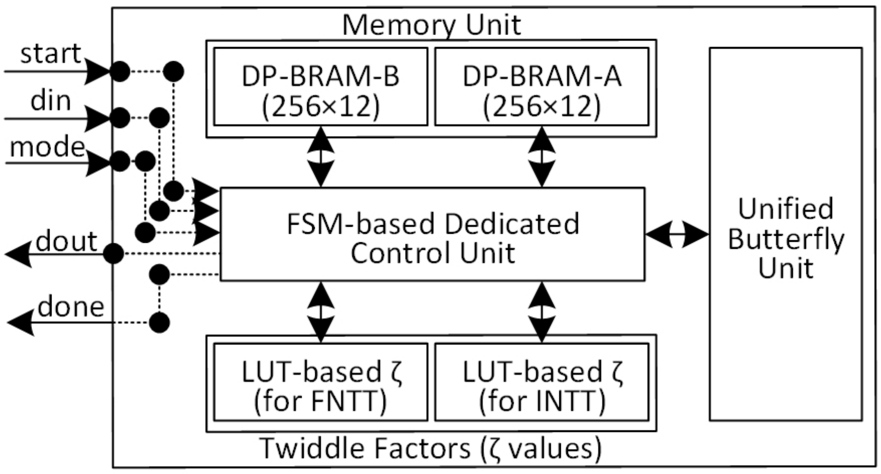

Figure 1. Block-level architecture of the proposed design incorporating register banks, butterfly unit, and control module.

3.1. Memory Unit

The proposed accelerator employs a compact and efficient memory subsystem based on two DP-BRAM (DP-BRAM-A and DP-BRAM-B), each storing 256 words of 12 bits. These blocks form a ping-pong storage structure, allowing one memory to act as the source of coefficient data while the other simultaneously receives the transformed outputs. During each stage of the radix-2 FNTT or INTT, the control unit selects the active source BRAM. It then reads two addresses per butterfly operation to implement the corresponding

Equations (3)–(6). The updated coefficients $z0$ and $z1$ produced by the unified butterfly unit are written to the destination or other BRAM at the same memory location. This alternating role of the source and destination memories eliminates the need to store intermediate results between stages. As a result, unnecessary data movement is reduced, lowering both area and power overhead. After the final stage, the `mem_sel` flag identifies which BRAM holds the completed transform. The control unit then reads the 256 coefficients sequentially from the selected BRAM and streams them through the `dout` interface of the implemented design.

3.2. Twiddle Factors Unit

The twiddle factor is a precomputed constant used in both the FNTT and INTT. As shown in Figure 1, the twiddle-factors unit includes two LUT-based ROMs that store the CRYSTALS–Kyber forward ζ table and inverse ζ^{-1} table. The `zeta_idx` counter generated by the control unit selects the appropriate entry, ensuring correct twiddle delivery for every butterfly stage. For the FNTT, the CT butterfly uses a monotonically increasing traversal of the ζ array. In contrast, the INTT's GS butterfly requires a reverse traversal of the inverse table. The unified butterfly accepts the selected twiddle factor through its ζ input. It then applies the pattern defined by the radix-2 FNTT/INTT equations, as illustrated in Equations (3)–(6). Because the twiddle factors are fixed for CRYSTALS–Kyber parameters ($N = 256$, $q = 3329$), the LUT-based ROM approach provides an area-efficient implementation. It also offers deterministic read latency with no computational overhead. This design avoids recalculation of roots of unity and enables seamless stage-by-stage progression driven solely by index arithmetic in the control logic.

3.3. Three-Stage Pipelined Unified Butterfly Unit (BU)

Figure 2 illustrates the proposed unified BU, which supports both FNTT and INTT computations. The figure uses color coding to highlight the functionality of the individual butterfly building blocks. Each of these blocks is explained in detail in the following sections.

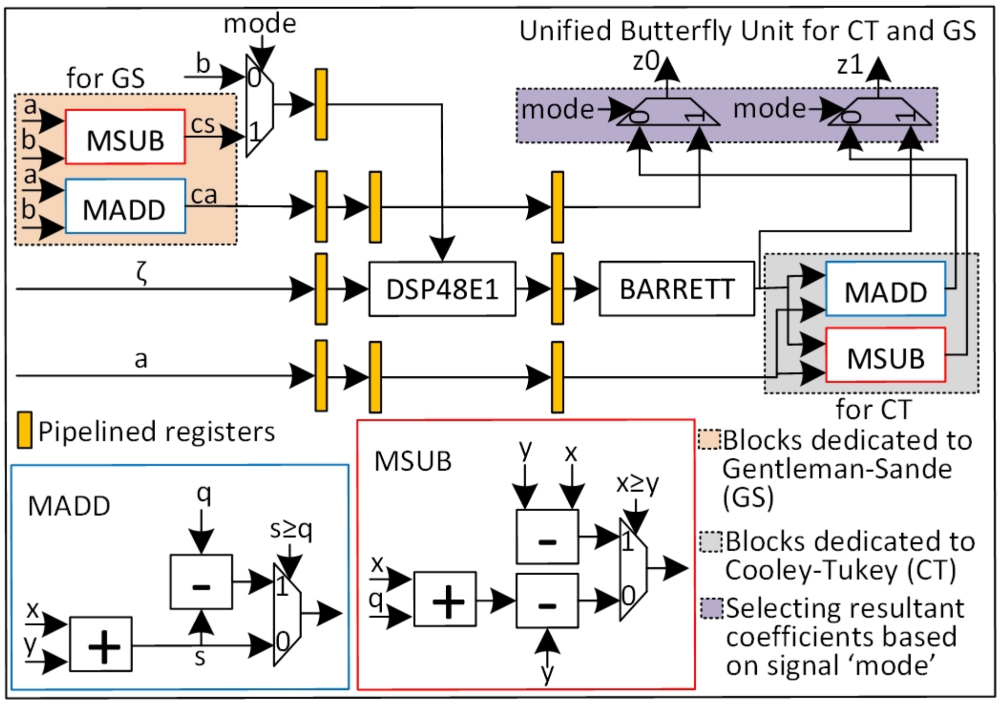
mode
Unified Butterfly Unit for CT and GS
for GS
b
0
1
a
b
MSUB
cs
a
b
MADD
ca
z0
z1
mode
mode
ζ
DSP48E1
BARRETT
MADD
MSUB
a
for CT
Pipelined registers
MADD
q
s≥q
x
y
+
-
s
1
0
MSUB
y
x
x≥y
x
q
+
-
-
y
1
0
Blocks dedicated to Gentleman-Sande (GS)
Blocks dedicated to Cooley-Tukey (CT)
Selecting resultant coefficients based on signal 'mode'

Figure 2. Three-stage pipelined architecture for butterfly unit supporting FNTT and INTT computations.

3.3.1. Dedicated Architecture for CT Configuration (Light Gray)

The right-hand portion of the figure, shaded in gray, contains the blocks dedicated to the CT butterfly. These blocks implement Equations (3) and (4). It first multiplies the ζ value by the operand b; then, the product passes to the `BARRETT` unit for modular reduction. As shown in Figure 2, the reduced multiplication result is forwarded to the gray portion of the figure when it becomes available. This value is then used to compute the modular addition (MADD) and modular subtraction (MSUB) operations. The outputs of these MADD and MSUB operations serve as inputs to the purple portion of the design. This stage handles routing and final coefficient selection, which are written back to memory according to the `mode` signal.

3.3.2. Dedicated Architecture for GS Configuration (Light Brown)

The left portion of the figure, shaded in light brown, contains the arithmetic blocks used exclusively to implement Equations (5) and (6). More precisely, this implements the $a + b \pmod{q}$ and $\zeta^{-1} \cdot (a - b) \pmod{q}$. It includes one MADD block and one MSUB block, each performing CRYSTALS–Kyber conditional reduction. The output of the MSUB is routed through a multiplexer to connect with the DSP48E1 unit for multiplication. In contrast, the output of the MADD is sent directly to the purple portion of the figure. After the multiplication, the generated product is reduced using the `BARRETT` unit. The output of the `BARRETT` unit goes to the purple portion for the final coefficient selection, depending on the `mode` signal.

3.3.3. Shared Datapath (Purple Part + Pipeline Registers + DSP Block + BARRETT Unit)

The purple and middle parts of the figure represent the hardware shared by both CT and GS modes. Specifically, the purple part in Figure 2 comprises a mode-controlled, multiplexer-based routing network. This network selects the correct coefficients after the CT and GS computations. When `mode = 0`, the coefficients emerging from the CT-dedicated region are forwarded to the memory interface for writing to the corresponding DP-BRAM unit. When `mode = 1`, the results produced in the GS region are selected instead. This separation keeps the CT and GS critical paths independent while requiring only a single selection point at the output of the butterfly. Similarly, the orange-filled blocks illustrate the pipeline registers. The objective is to optimize the butterfly critical path while ensuring correct computations. Note that no automated approach was used to insert these pipeline registers. Instead, the critical path of the butterfly data path was first identified, and registers were manually inserted at well-defined arithmetic boundaries. These boundaries occur between the coefficient preparation logic, the DSP-based multiplication, and the Barrett reduction and modular add/subtract stages.

Based on this deterministic pipeline partitioning, timing closure can be independently verified by synthesizing and implementing the design under the same clock constraint. In this work, the complete architecture was evaluated at the post-place-and-route level under a fixed 200 MHz timing constraint, ensuring that the reported operating frequency reflected realistic routing and critical-path behavior. By applying the same arithmetic-boundary-based pipelining strategy and timing constraint, readers can reproduce the timing-closure process using a standard FPGA design flow. Based on the placement of the pipeline registers, Figure 2 shows that our butterfly unit maintains a 3-stage pipelining: (i) preparing input coefficients and computing the light brown portion, (ii) computing multiplication using a single DSP48E1 slice, and (iii) passing the multiplication result to `the BARRETT` unit for modular reduction and computing the gray-colored portion. In summary, pipeline registers allow the butterfly unit to run at a high clock frequency. The BARRETT reduction unit is described in further detail in Section Four-Stage Pipelined DSP-Free BARRETT Reduction Unit.

Four-Stage Pipelined DSP-Free BARRETT Reduction Unit

Algorithm 4 summarizes the modular reduction method used throughout the NTT design. The corresponding architecture is illustrated in Figure 3. The reduction is organized into four explicitly pipelined stages (B1–B4), and the sequence matches the data-path structure shown in Figure 3. The corresponding details are described below.

Algorithm 4 Proposed DSP-free Barrett reduction for CRYSTALS–Kyber ($q = 3329$).

```
Input: Signed intermediate $x \in \mathbb{Z}$ (up to 24 bits)
Input: Modulus $q = 3329$
Input: Precomputed constant $s_f = 20159$
Input: Shift parameter $s = 6$
Output: Reduced value $y = x \bmod q$
 1: Stage B1: Input register
 2: $x_1 \leftarrow x$ {register input word}
 3: Stage B2: Scaled BARRETT constant
 4: $t_1 \leftarrow x_1 \cdot s_f$ {implemented as a shift–add network for $s_f$}
 5: Stage B3: Quotient estimate and scaled modulus
 6: $t_2 \leftarrow \lfloor t_1/2^s \rfloor$ {right shift by $s = 6$}
 7: $t_3 \leftarrow t_2 \cdot q$ {implemented as a shift–add network for $q = 3329$}
 8: Stage B4: Final subtraction and correction
 9: $t_{\text{full}} \leftarrow x - t_3$
10: if $t_{\text{full}} \geq q$ then
11:     $t_{\text{full}} \leftarrow t_{\text{full}} - q$
12: end if
13: $y \leftarrow t_{\text{full}}$ {canonical residue in $[0, q-1]$}
14: return $y$
```

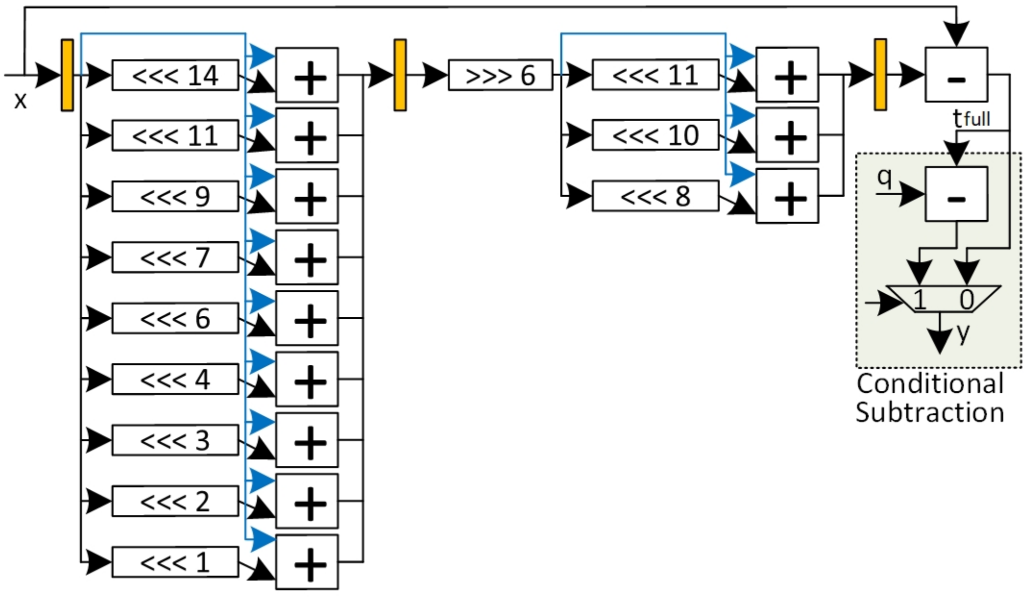

Figure 3. Proposed 4-stage pipelined DSP-free BARRETT reduction architecture. Orange-filled boxes are the pipelined registers. Blue lines differentiate only the first input from the second to the adders.

- The first stage (B1) registers the input word x, preventing any long feed-through paths.
- In stage B2, x is multiplied by the Barrett constant $s_f = 20{,}159$, The following highlights are the same. but instead of using a full multiplier, this step is implemented as a structured network of leftward binary shifts followed by additions. Figure 3 highlights the nine shifts corresponding to the decomposition of s_f and the balanced adder tree that accumulates them, producing the intermediate value t_1 .
- Stage B3 forms the quotient approximation by shifting t_1 right by six bits (i.e., division by 2^6), yielding $t_2 = \left\lfloor \frac{x \cdot s_f}{2^s} \right\rfloor$. The parameters $s_f = 20{,}159$ and $s = 6$ were selected according to the classical Barrett reduction formulation [35], where $s_f \approx \lfloor 2^s/q \rfloor$. For the CRYSTALS–Kyber modulus $q = 3329$, this choice ensures that the quotient estimation error satisfies $\left| t_2 - \left\lfloor \frac{x}{q} \right\rfloor \right| \leq 1$ for all intermediate values x generated during the FNTT and INTT computations. Consequently, the intermediate value $t_{\text{full}} = x - t_2 \cdot q$ is guaranteed to lie within the bounded interval $[0, 2q)$, which allows correct modular reduction using at most one conditional subtraction in Stage B4. The resulting value t_2 is then multiplied by $q = 3329$ using a compact shift–add network, as illustrated in Figure 3.
- Stage B4 computes the intermediate value $t_{\text{full}} = x - t_3$. Based on the bounded quotient approximation established in Stage B3, the value of t_{full} is guaranteed to lie within the interval $[0, 2q)$ for all valid intermediate inputs generated during the FNTT and INTT computations. As a result, at most one conditional subtraction of the modulus is sufficient to produce the final reduced coefficient in the canonical range $[0, q-1]$, as shown in Figure 3.

3.4. Controller Operation and Cycle Accounting

The FNTT/INTT engine is controlled by an FSM that manages memory accesses, selects data-path routing, and coordinates the timing of the butterfly and Barrett pipeline. The FSM generates the read–write addresses for both DP-BRAMs and configures the routing multiplexers shown in Figures 2 and 3. Depending on the selected mode, the controller activates the CT configuration to implement the FNTT using Algorithm 1. Alternatively, it enables the GS configuration to implement the INTT using Algorithm 2. Since only a single butterfly unit is used, the controller serializes the computation by iterating across pairs of coefficients and tracking the progression within each stage. The controller maintains counters for the stage index, the block index, and the position within each block. These counters ensure that the memory indices follow the ordering implied by the radix-2 structure of the FNTT and INTT. The ping-pong memory organization means that no intermediate transfers are required. One DP-BRAM serves as the source while the other receives the processed results, and the controller toggles their roles at every stage. In addition to the data path, the FSM determines the complete timing behavior of the proposed NTT accelerator. For an input vector of 256 coefficients, the cycle count breaks down as follows:

- **Input loading:** Both the FNTT and INTT transforms begin by streaming 256 coefficients into the DP-BRAM, one per cycle, for a total of 256 clock cycles.
- **Computational stages:** For CRYSTALS–Kyber with $N = 256$, there are 7 radix-2 stages in the NTT. Executing these stages through the pipelined butterfly requires 1029 clock cycles for the FNTT, including all pipeline latencies and address updates. The INTT additionally performs a normalization by N^{-1} (Equation (2)). This normalization in our design is implemented using the same butterfly path, requiring 256 additional cycles. As a result, the overall INTT latency amounts to 1285 cycles.
- **Output streaming:** Finally, the 256 reduced coefficients are read from the DP-BRAM and streamed to the output port, again requiring a single cycle per coefficient.

4. Experimental Results and Comparative Analysis

This section summarizes the experimental results of the proposed NTT accelerator and compares them with state-of-the-art designs. Section 4.1 presents the experimental setup and the post-place-and-route results on the target Artix-7 FPGA. Section 4.2 contrasts the obtained area–performance trade-offs against existing NTT accelerators and discusses the resulting throughput-to-area efficiency. Finally, the overall discussions are illustrated in Section 4.3.

4.1. Implementation Results

In this section, we first provide the experimental setup in Section 4.1.1, including the synthesis tools used. After that, we report the corresponding area and timing results of our proposed NTT architecture in Section 4.1.2 and Section 4.1.3, respectively. Based on the area and timing results, we highlight the practical usage of our design in Section 4.1.4.

4.1.1. Experimental Setup (Toolchain + Constraints)

The proposed accelerator was described in synthesizable Verilog HDL and implemented on an Artix-7 FPGA (XC7A100TCSG324-3) using the Vivado IDE design flow (version 14.1). All results reported in this section were obtained after the post-place-and-route stage to reflect realistic timing and resource utilization. A single clock constraint was applied to the design using an XDC timing constraint, targeting a 200 MHz operating frequency (which relates to the 5 ns clock period). Moreover, mapping on an FPGA device, our choices included coefficient memories inferred as dual-port BRAMs, the twiddle tables inferred as ROM structures, and the integer multiplication in the butterfly mapped to DSP48E1. The Barrett reduction data path was implemented using LUT-based shift–add structures, as described in Section 3.

4.1.2. Area Results

Table 1 shows the area breakdown in terms of slices, LUTs, flip-flops (FFs), DSPs, and DP-BRAMs. The implementation results summarized in the table demonstrate the compact nature of the proposed NTT architecture. The complete core occupies 184 slices, 503 LUTs, and 545 flip-flops, requiring only a single DSP48E1 slice and two 18 KB dual-port BRAMs. The memory blocks themselves contribute negligible logic (five LUTs and twelve flip-flops combined), which confirms that the design overhead is concentrated in the computational pipeline rather than the storage resources. The unified butterfly unit accounts for the largest logic share, using 322 LUTs and 258 flip-flops, while still requiring only 1 DSP slice. Within it, the DSP-free Barrett reduction unit, implemented using a structured shift-and-add network, occupies 175 LUTs and 82 flip-flops, demonstrating its lightweight footprint despite replacing a conventional modular multiplier. Finally, the LUT-based twiddle factor ROM requires only 6 LUTs and 24 flip-flops, since coefficients are indexed rather than computed. Overall, these results highlight that the design is dominated neither by multipliers nor memory, but by the highly optimized arithmetic pipeline, while maintaining extremely low DSP utilization and BRAM usage.

The area results described above are reported after place-and-route to capture the actual mapping of the butterfly pipeline, Barrett logic, and memory elements onto the Artix-7 fabric under timing constraints of 200 MHz. These may vary when timing constraints differ.

Table 1. Area breakdown on an Artix-7 (XC7A100TCSG324-3) FPGA.

Designs	**Slices**	**LUTs**	**FFs**	**DSPs**	**DP-BRAMs (18 KB)**
NTT-Core (Total Area Utilization)	184	503	545	1	2
⌊ DP-BRAM-A	0	0	0	0	1
⌊ DP-BRAM-B	5	12	0	0	1
⌊ Three-Stage Pipelined Butterfly Unit	125	322	258	1	0
⌊ Four-Stage Pipelined BARRETT Unit	63	175	82	0	0
⌊ LUT-Based ζ for FNTT and INTT	6	24	0	1	0

4.1.3. Timing Results + Figure of Merit

The timing figures reported below correspond to a post-place-and-route timing analysis under the 200 MHz clock constraint, thereby reflecting realistic critical-path and routing effects. Therefore, the timing results of the proposed NTT Core are reported in Table 2, where column one specifies the implemented NTT operation, which can be either FNTT or INTT. Columns two to four report clock cycles (CCs), operating frequency (Freq. in MHz), and computation time (Latency in µs) for a single FNTT or INTT process. Latency was computed using Equation (7). Column five presents the throughput (TP in Kbps), calculated using Equation (8), while column six defines the figure of merit (FoM) in a ratio of throughput to LUTs, as determined by Equation (9). The scaling factor of 10^3 in Equation (9) comes from the throughput (i.e., Equation (8)), which is measured in kilobits per second, resulting in the FoM being reported in a convenient numerical range. Finally, the last column shows the average FoM of the FNTT and INTT computations.

$$Latency\ (\mu s) = \frac{Clock\ Cycles}{Frequency\ (\text{MHz})} \tag{7}$$

$$Throughput\ (\text{Kbps}) = \frac{1}{Latency\ (\mu s)} = \frac{10^6}{Latency\ (s)} \tag{8}$$

$$FoM = \frac{Throughput}{Area} = \frac{Throughput}{FPGA\ LUTs} \times 10^3 \tag{9}$$

Table 2. Timing results of our NTT-Core on (XC7A100TCSG324-3) FPGA.

Op.	**Clock Cycles (CCs)**	**Freq. (MHz)**	**Latency (µs)**	**Throughput (Kbps)**	**FoM**	**Average-FoM (Avg-FoM)**
FNTT	1029	200	5.14	194.55	386.77	348.22
INTT	1285		6.42	155.76	309.66	

Table 2 reveals that the proposed NTT design works at 200 MHz (maximum) on the Artix-7 device. Moreover, the FNTT completes in 1029 cycles, corresponding to a latency of 5.14 µs, while the INTT requires 1285 clock cycles, achieving a latency of 6.42 µs. These latencies translate to throughputs of 194.55 Kbps and 155.76 Kbps for the FNTT and INTT, respectively. The FoM, defined as the throughput divided by LUT utilization, reflects the efficiency of the architecture relative to its resource footprint, yielding FoM values of 386.77 for FNTT and 309.66 for INTT, with an overall average of 348.22. These results highlight the balanced nature of the design, delivering high throughput per LUT while maintaining low area utilization.

4.1.4. Overall Evaluation Based on Implementation Results

The area and timing results presented in Sections 4.1.2 and 4.1.3 show that the proposed design achieves exceptionally low area and competitive throughput. This combination makes it attractive for hardware-security applications where resource efficiency directly impacts scalability, power consumption, and deployment cost. Achieving nearly 200 Kbps throughput using only 184 slices and a single DSP slice, the design is particularly advantageous in resource-constrained settings. This makes it well suited for FPGA-based secure IoT nodes and post-quantum cryptographic engines in embedded systems. Moreover, multiple instances of our compact butterfly architecture can be used to further minimize the clock cycles. This replication ultimately improves the throughput for high-speed post-quantum cryptographic applications. Consequently, the proposed solution demonstrates a strong balance between performance and area, making it well suited for both constrained and high-performance post-quantum cryptographic deployments.

4.2. Comparisons to Existing NTT Accelerators

Section 4.2.1 discusses the area and timing metrics of the proposed design/accelerator relative to existing implementations. The corresponding FoM analysis is reported in Section 4.2.2.

4.2.1. Area and Timing Comparisons

Table 3 compares the proposed design with existing solutions. The first column lists the reference implementations, while the second specifies whether the corresponding NTT operation is forward or inverse. The target device for each design is shown in the third column. Columns four through seven report FPGA resource utilization, including LUTs, FFs, DSPs, and BRAMs. Performance metrics are summarized in the remaining columns: the number of clock cycles is listed in column eight, operating frequency (in MHz) in column nine, computation latency (in µs) in column ten, and throughput (in Kbps) in column eleven.

The Zynq-7000 implementation in Ref. [36] is a single-butterfly Kyber NTT architecture that occupies 2908 LUTs and 9 DSPs. In contrast, our Artix-7 design uses only 503 LUTs and a single DSP, corresponding to about 83% fewer LUTs and a 9$\times$ reduction in DSP usage, while still maintaining a single-butterfly structure. Despite this much smaller footprint, our core achieves significantly lower latency: their FNTT (INTT) requires 43 µs (42.88 µs) at 45 MHz, whereas our FNTT (INTT) completes in 5.14 µs (6.42 µs) at 200 MHz. As a result, the proposed design provides roughly an order-of-magnitude improvement in latency and throughput over [36] on a low-cost FPGA device.

The flexible architecture of [18] supports configurations with 1, 4, and 16 butterfly units; in this work, we compared against their 1-butterfly design, which uses 948 LUTs, 352 FFs, and 35 BRAM18 blocks on an Artix-7 device. Our core reduces the LUT count from 948 to 503 (about 47% fewer LUTs) and lowers the BRAM usage from 35 blocks to just 2, while using a similar number of FFs and introducing no additional DSP overhead. Despite this substantial reduction in memory resources, the throughput of our design (194.55 Kbps for FNTT) remains close to that of [18] (210.52 Kbps). This demonstrates that the proposed data path and control organization achieve comparable performance with a far more memory-efficient implementation.

The two-butterfly Artix-7 design in Ref. [23] targets higher throughput by instantiating two NTT units, which results in 609 LUTs, 640 FFs, 2 DSPs, and 4 BRAMs. Our single-butterfly core instead uses 503 LUTs, 545 FFs, 1 DSP, and 2 BRAMs, thereby reducing both the LUT count and on-chip memory usage while halving the number of DSP blocks. As expected from the additional butterfly unit, [23] achieves a higher throughput (526.31 Kbps versus 194.55 Kbps), but this improvement comes at the cost of roughly 2× DSP usage and a larger area. This comparison shows that our design is better suited for resource-constrained Artix-7 devices while still providing competitive throughput for a single-butterfly engine.

The COHA architecture in Ref. [16] is another single-butterfly NTT design implemented on a Virtex-7 device, requiring 2128 LUTs, 1144 FFs, and 8 DSPs. Relative to our work, this corresponds to LUT and FF overheads of approximately 4.2× and 2.1×, respectively, while our implementation reduces the DSP count from eight to just one. In terms of performance, [16] reports FNTT and INTT latencies of 5.29 µs and 6.80 µs at 174 MHz, which are very close to our 5.14 µs and 6.42 µs latencies at 200 MHz. Thus, our design achieves essentially the same single-butterfly throughput as [16] but on a smaller Artix-7 device and with a far more compact and DSP-efficient hardware footprint.

Table 3. Comparison with state-of-the-art NTT designs. All these architectures use $n = 256$ and $q = 3329$.

Designs	NTT Type	Device	LUTs	FFs	DSPs	BRAMs (18 KB)	CCs	Freq. (MHz)	Latency (µs)	Throughput (Kbps)
[36]/2020	FNTT INTT	ZynQ-7000	2908	170	9	–	1935 1930	45	43 42.88	23.25 23.32
[18]/2021	FNTT + INTT	Artix-7	948	352	–	35	904	190	4.75	210.52
[23]/2021	FNTT + INTT	Artix-7	609	640	2	4	490	257	1.90	526.31
[16]/2022	FNTT INTT	Virtex-7	2128	1144	8	–	922 1184	174	5.29 6.80	189.03 147.05
[24]/2023	FNTT	Virtex-7	7800	–	6	0	–	72	–	–
[25]/2024	FNTT + INTT	Virtex-7	9298	9402	0	0	898	20	44.90	22.27
[14]/2024	FNTT INTT	Virtex-7	2018	1829	0	0	1154 1282	250	4.61 5.12	216.91 195.31
[19]/2024	FNTT INTT	Artix-7	18296	12134	–	–	85 104	210	0.40 0.50	2500 2000
[17]/2024	FNTT + INTT	Artix-7	1070	1071	5	10.5	306	278	1.10	909.09
[28]/2025	FNTT + INTT	Artix-7	6841	6982	0	0	898	249	3.72	268.81
This Work	FNTT INTT	Artix-7	503	545	1	2	1029 1285	200	5.14 6.42	194.55 155.76

Single-butterfly unit-based architectures:⇒ [14,16,24,25,28,36]. Two-butterfly units:⇒ [23]. One-, four- and sixteen-butterfly units:⇒ [18] (we compare results with their one-butterfly unit implementation). The architecture in Ref. [19] employs 64 modular adders, subtractors, and multipliers in the data path, resulting in a 64-butterfly unit-based accelerator.

The Virtex-7 implementation of [24] focuses only on FNTT computation and uses a large single-butterfly data path with 7800 LUTs and 6 DSPs. However, the reference work does not report complete latency or throughput figures. Even without detailed timing numbers, the area gap is clear: our Artix-7 core uses only 503 LUTs and 1 DSP, which corresponds to roughly 15.5× fewer LUTs and a 6× reduction in DSP usage. This comparison underlines that the proposed architecture is considerably more area-efficient while still supporting the same CRYSTALS–Kyber parameter set ($n = 256, q = 3329$).

The Virtex-7 CRYSTALS–Kyber NTT implementation in Ref. [25] unifies FNTT and INTT in a single-butterfly design that consumes 9298 LUTs and 9402 FFs without using DSPs or BRAMs. Instead of the BRAMs, the reference work utilizes the register files for coefficient storage. In contrast, our Artix-7 design requires only 503 LUT and 545 FFs with 1 DSP and 2 BRAMs, resulting in approximately 18.5× and 17.3× reductions in LUT and FF counts, respectively. Despite this area reduction, the latency of [25] (44.90 µs at 20 MHz) is around 8.7× larger than our FNTT latency of 5.14 µs at 200 MHz, and the throughput is correspondingly lower (22.27 Kbps versus 194.55 Kbps).

The Virtex-7 design in Ref. [14] is a single-butterfly NTT/INTT engine that occupies 2018 LUTs and 1829 FFs with no DSPs or BRAMs. Compared to this, our Artix-7 implementation reduces the LUT and FF usage by approximately 4.0× and 3.4×, respectively, at the cost of a single DSP and two small BRAM blocks. The FNTT and INTT latencies reported in Ref. [14] (4.61 µs and 5.12 µs at 250 MHz) are slightly better than our 5.14 µs and 6.42 µs, resulting in throughput that is about 10–25% higher. Overall, the comparison indicates that our design trades a modest reduction in throughput for a very substantial reduction in logic resources, which is attractive for dense PQC accelerators.

The highly parallel Artix-7 architecture in Ref. [19] instantiates 64 butterfly units, leading to a very large data path that uses 18,296 LUTs and 12,134 FFs. As expected from this level of parallelism, their FNTT (INTT) latency is only 0.40 µs (0.50 µs) and the throughput reaches 2.5 Mbps (2.0 Mbps). Our single-butterfly core targets a very different design point: it consumes about 36× fewer LUTs and 22× fewer FFs and avoids the complexity of a wide 64-way data path while still achieving 194.55 Kbps and 155.76 Kbps throughput for FNTT and INTT, respectively. This illustrates that the proposed architecture offers a compact, low-cost alternative to massively parallel NTT accelerators, suitable for resource-constrained systems where area is the primary optimization target.

The high-speed Artix-7 design in Ref. [17] integrates FNTT and INTT with aggressive parallelism, requiring 1070 LUTs, 1071 FFs, 5 DSPs, and 10.5 BRAM (of size 18KB) blocks. Running at 278 MHz, it achieves a latency of 1.10 µs and a throughput of 909.09 Kbps. Our core instead emphasizes area efficiency: we use roughly half the LUTs and FFs, 5× fewer DSPs, and more than 5× fewer BRAM blocks but accept about 4.7× lower throughput. The comparison shows that, when abundant on-chip memory and DSPs are available, [17] attains a very high speed, whereas our design is better aligned with scenarios that must minimize both logic and memory usage.

The recent Artix-7 work in Ref. [28] is a single-butterfly NTT/INTT engine that uses 6841 LUTs and 6982 FFs, with no DSPs and no BRAMs, operating at 249 MHz with a 3.72 µs latency and 268.81 Kbps throughput. Compared with this state-of-the-art single-butterfly design, our core reduces the LUT and FF counts by about 13.6× and 12.8×, respectively, while still providing a 5.14 µs FNTT latency and 194.55 Kbps throughput. Thus, although [28] achieves somewhat higher throughput, the proposed architecture is more compact and therefore better suited as a building block within larger PQC systems where multiple NTT engines must coexist on the same FPGA.

4.2.2. FoM Comparisons

Equation (9) was employed to compare the proposed design with existing architectures in terms of throughput-to-area efficiency. In this work, the FPGA LUT count was consistently used as the area metric for all evaluated designs, ensuring uniform normalization across different devices and implementations. As reported in Table 3, throughput values are expressed in kilobits per second. Consequently, the scaling factor of 10^3 in Equation (9) arises from the throughput unit and allows the FoM values to be presented within a convenient numerical range. The corresponding FoM results are illustrated in Figure 4.

Figure 4 illustrates the FoM in throughput-to-area efficiency, where higher values indicate more efficient architectures. Among the existing works, the architectures of Ref. [23] and Ref. [17] achieve the highest FoM values (864.22 and 849.61, respectively). This advantage primarily stems from their highly optimized data-path structures designed for high-throughput operation. Our implementation attains an FoM of 348.22, outperforming several related designs and ranking third overall, while using significantly fewer FPGA resources. These results show that although two designs achieve higher throughput-to-area efficiency, the proposed architecture provides a strong balance between area compactness and performance. This makes it particularly suitable for integration into lightweight and resource-constrained PQC hardware systems.

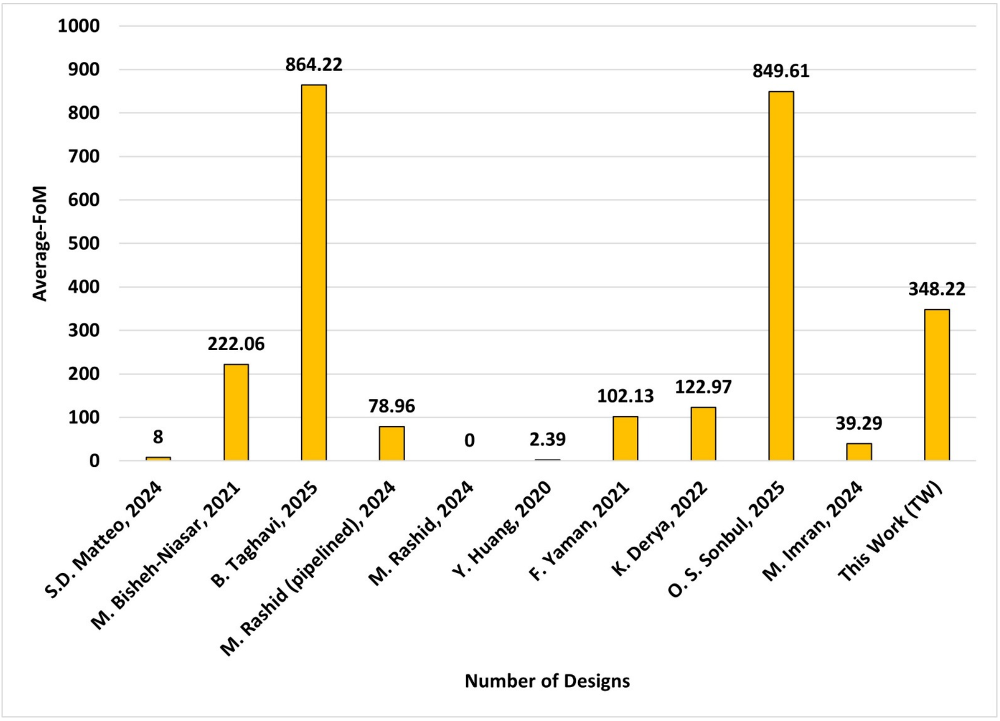

Figure 4. Average-FoM comparison with existing designs. The higher the FoM value, the higher the efficiency of the architecture. The work S.D. Matteo, 2024 is available at [31]. The work M. Bisheh-Niasar, 2021 is available at [15]. The work B. Taghavi, 2025 is available at [20]. The work M. Rashid (pipelined), 2024 is available at [13]. The work M. Rashid, 2024 is available at [14]. The work Y. Huang, 2020 is available at [22]. The work F. Yaman, 2021 is available at [18]. The work K. Derya, 2022 is available at [16]. The work O. S. Sonbul, 2025 is available at [28]. The work M. Imran, 2024 is available at [25].

4.3. Discussion

The comparisons to the state of the art in Section 4.2 show that the proposed architecture achieves an extremely low-area design point. Moreover, it delivers competitive throughput for a single-butterfly NTT engine. Designs that rely on multiple butterfly units and wide memory systems naturally achieve lower latency and higher throughput, but they incur substantial LUT, DSP, and BRAM overhead. In contrast, the proposed design deliberately minimizes its resource footprint, with particular emphasis on reducing DSP and BRAM usage. At the same time, it maintains a high operating frequency through deep pipelining. These characteristics make the accelerator well suited as a reusable building block in complete post-quantum cryptographic hardware implementations and integrated accelerator subsystems.

Although the current implementation targets an Artix-7 FPGA, the architecture can be parameterized to improve portability across different FPGA families. This flexibility also enables support for a wide range of area–throughput trade-offs. Twiddle-factor storage, for example, may be implemented as LUT-based ROM to minimize BRAM usage or as BRAM-based ROM to reduce LUT pressure and improve timing on LUT-constrained devices. Arithmetic mapping can also be adapted. Multiplication may use DSP48E1 blocks for high performance and energy efficiency. Alternatively, it may be implemented using

LUT-based structures when DSP resources are scarce or reserved for other cryptographic components. Similarly, constant multiplications in the modular reduction can be realized using shift–add networks, as demonstrated in this work, or migrated to DSP resources when higher throughput is required.

These parameter-driven mapping options allow the same RTL description to be tuned for different FPGA devices and deployment scenarios. As a result, the architecture supports the portability and performance objectives outlined in Section 1 while preserving a scalable and implementation-friendly design.

4.4. Side-Channel and Fault-Attack Considerations

Hardware implementations of lattice-based cryptographic primitives are potentially exposed to physical attacks, such as side-channel analysis and fault injection. These risks are particularly relevant in resource-constrained IoT environments. Although the primary objective of this work was to design a high-throughput and area-efficient NTT accelerator, it remains important to consider potential security implications. In particular, highlighting possible leakage sources and implementation-level vulnerabilities is essential.

In the proposed architecture, the FSM controller and the generated memory access patterns are fully deterministic and independent of secret data. The FSM-driven address generation follows a fixed radix-2 schedule for both FNTT and INTT operations, and the same sequence of memory reads and writes is executed for all inputs. This constant-time behavior significantly reduces exposure to timing-based side-channel attacks. However, data-dependent switching activity may still arise within the arithmetic data path, particularly in the DSP48E1-based multiplication and the Barrett reduction units. These components operate directly on secret-dependent coefficients, which can introduce observable variations in switching behavior. Such switching variations can lead to power or electromagnetic side-channel leakage if no countermeasures are applied. Fault injection attacks targeting the unified butterfly unit or the associated memory subsystem may also corrupt intermediate coefficients and produce incorrect transform outputs.

Although explicit side-channel or fault countermeasures are not included in the current design, the architecture offers several favorable properties for future protection. The separated pipeline stages, unified butterfly structure, and centralized control logic allow straightforward integration of additional protection techniques. Examples include coefficient masking, clock jittering, or redundancy-based fault detection. Lightweight arithmetic masking or dual-computation schemes can be added at the butterfly level without modifying the overall memory organization or control flow. Therefore, while this work does not claim inherent resistance to physical attacks, the deterministic architecture and modular data-path organization offer several structural advantages. These properties provide a strong foundation for future secure implementations targeting side-channel and fault-resilient post-quantum cryptographic hardware.

5. Conclusions

This article introduced a compact and deeply pipelined design for the FNTT and INTT operations used in the CRYSTALS–Kyber post-quantum cryptosystem. Deep pipelining in the proposed architecture ensures high operating frequency and low computation latency. At the same time, the unified three-stage butterfly engine, efficient ping-pong BRAM banking, and LUT-based modular reduction significantly reduce resource overhead without sacrificing throughput. Implemented on an Artix-7 FPGA, the design occupies only 503 LUTs, 545 flip-flops, 1 DSP48E1 block, and 2 BRAM18 blocks. It completes the FNTT and INTT in 5.14 µs and 6.42 µs, respectively, at 200 MHz. Comparative analysis against existing designs demonstrated that the presented architecture achieves an excellent throughput-to-area trade-off. This efficiency makes it well suited for lightweight post-quantum cryptographic accelerators, secure embedded systems, and resource-constrained Internet-of-Things devices. Future work will focus on extending the architecture by incorporating multiple butterfly units to further reduce clock cycles and improve overall computation time.

Author Contributions: Conceptualization, O.S.S. and M.R.; Methodology, O.S.S., M.R., M.I.M. and M.A.; Software, O.S.S., M.R. and M.I.M.; Validation, O.S.S., M.R., M.I.M. and A.Y.J.; Formal analysis, O.S.S. and M.R.; Resources, M.A.; Data curation, M.A. and A.Y.J.; Writing—original draft, O.S.S. and M.R.; Writing—review & editing, M.R. and M.I.M.; Visualization, M.R.; Supervision, M.R. and A.Y.J.; Project administration, O.S.S. and M.R.; Funding acquisition, O.S.S. and M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Umm Al-Qura University, Saudi Arabia under grant number: 25UQU4320199GSSR02.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

References

1. Imran, M.; Khalid, A.; Rafferty, C.; Rashid, M.; O'Neill, M. Evaluating Area-Time-Power Trade-offs in NTT Accelerators for Post-Quantum Cryptography. In Proceedings of the 2025 IEEE 38th International System-on-Chip Conference (SOCC), Dubai, United Arab Emirates, 29 September–1 October 2025; pp. 1–6. [CrossRef]
2. Gao, X.; Li, Y.; Li, T.; Li, X.; Wang, J. A High-Throughput, BRAM-Efficient NTT/INTT Accelerator for ML-KEM. *Electronics* **2025**, *14*, 4868. [CrossRef]
3. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* **1997**, *26*, 1484–1509. [CrossRef]
4. Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J.C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F.G.; Buell, D.A.; et al. Quantum supremacy using a programmable superconducting processor. *Nature* **2019**, *574*, 505–510. [CrossRef]
5. Imran, M.; Aikata, A.; Roy, S.S.; Pagliarini, S. High-Speed Design of Post Quantum Cryptography With Optimized Hashing and Multiplication. *IEEE Trans. Circuits Syst. II Express Briefs* **2024**, *71*, 847–851. [CrossRef]
6. Gong, M.; Wang, S.; Zha, C.; Chen, M.C.; Huang, H.L.; Wu, Y.; Zhu, Q.; Zhao, Y.; Li, S.; Guo, S.; et al. Quantum walks on a programmable two-dimensional 62-qubit superconducting processor. *Science* **2021**, *372*, 948–952. [CrossRef]
7. National Institute of Standards and Technology. NIST to Standardize Encryption Algorithms That Can Resist Attack by Quantum Computers. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography (accessed on 26 March 2025).
8. National Institute of Standards and Technology. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication. Available online: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.ipd.pdf (accessed on 26 March 2025).
9. National Institute of Standards and Technology. FIPS 204: Module-Lattice-Based Digital Signature Standard. Federal Information Processing Standards Publication. Available online: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.ipd.pdf (accessed on 26 March 2025).
10. Fouque, P.A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Prest, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU Specifications v1.1. Available online: https://falcon-sign.info (accessed on 25 March 2025).
11. National Institute of Standards and Technology. FIPS 205: Stateless Hash-Based Digital Signature Standard. Federal Information Processing Standards Publication. Available online: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.ipd.pdf (accessed on 26 March 2025).
12. Imran, M.; Abideen, Z.U.; Pagliarini, S. An Experimental Study of Building Blocks of Lattice-Based NIST Post-Quantum Cryptographic Algorithms. *Electronics* **2020**, *9*, 1953. [CrossRef]

13. Rashid, M.; Sonbul, O.S.; Jamal, S.S.; Jaffar, A.Y.; Kakhorov, A. A Pipelined Hardware Design of FNTT and INTT of CRYSTALS-Kyber PQC Algorithm. *Information* **2024**, *16*, 17. [CrossRef]
14. Rashid, M.; Khan, S.; Sonbul, O.S.; Hwang, S.O. A Flexible and Parallel Hardware Accelerator for Forward and Inverse Number Theoretic Transform. *IEEE Access* **2024**, *12*, 181351–181361. [CrossRef]
15. Bisheh-Niasar, M.; Azarderakhsh, R.; Mozaffari-Kermani, M. Instruction-Set Accelerated Implementation of CRYSTALS-Kyber. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 4648–4659. [CrossRef]
16. Derya, K.; Mert, A.C.; Öztürk, E.; Savaş, E. CoHA-NTT: A Configurable Hardware Accelerator for NTT-Based Polynomial Multiplication. *Microprocess. Microsyst.* **2022**, *89*, 104451. [CrossRef]
17. Sun, J.; Bai, X. A High-Speed Hardware Architecture of an NTT Accelerator for CRYSTALS-Kyber. *Integr. Circuits Syst.* **2024**, *1*, 92–102. [CrossRef]
18. Yaman, F.; Mert, A.C.; Öztürk, E.; Savaş, E. A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme. In *Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, 1–5 February 2021*; IEEE: New York, NY, USA, 2021; pp. 1020–1025.
19. Saoudi, M.; Kermiche, A.; Benhaddad, O.H.; Guetmi, N.; Allailou, B. Low latency FPGA implementation of NTT for Kyber. *Microprocess. Microsyst.* **2024**, *107*, 105059. [CrossRef]
20. Taghavi, B.; Azarderakhsh, R.; Mozaffari Kermani, M. ParallelNTT: Maximizing Performance of Forward and Inverse NTT on FPGA for ML-DSA and ML-KEM. In *Proceedings of the Great Lakes Symposium on VLSI 2025, GLSVLSI '25, New Orleans, LA, USA, 30 June–2 July 2025*; Association for Computing Machinery: New York, NY, USA, 2025; pp. 372–378. [CrossRef]
21. Xu, J.; Zeng, Y.; Deng, W.; Han, L.; Luo, C. An Area-Performance Balanced Hardware Accelerator of NTT for Kyber. In *Information Security and Cryptology*; Chen, R., Deng, R.H., Yung, M., Eds.; Springer Nature: Singapore, 2026; pp. 103–120.
22. Huang, Y.; Huang, M.; Lei, Z.; Wu, J. A Pure Hardware Implementation of CRYSTALS-KYBER PQC Algorithm Through Resource Reuse. *IEICE Electron. Express* **2020**, *17*, 20200234. [CrossRef]
23. Zhang, C.; Liu, D.; Liu, X.; Zou, X.; Niu, G.; Liu, B.; Jiang, Q. Towards Efficient Hardware Implementation of NTT for Kyber on FPGAs. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021; pp. 1–5. [CrossRef]
24. Khan, S.; Khalid, A.; Rafferty, C.; Shah, Y.A.; O'Neill, M.; Lee, W.K.; Hwang, S.O. Efficient, Error-Resistant NTT Architectures for CRYSTALS-Kyber FPGA Accelerators. In *Proceedings of the 2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC), Dubai, United Arab Emirates, 16–18 October 2023*; IEEE: New York, NY, USA, 2023; pp. 1–6.
25. Imran, M.; Khan, S.; Khalid, A.; Rafferty, C.; Shah, Y.A.; Pagliarini, S.; Rashid, M.; O'Neill, M. Evaluating NTT/INTT Implementation Styles for Post-Quantum Cryptography. *IEEE Embed. Syst. Lett.* **2024**, *4*, 485–488. [CrossRef]
26. Yahya Hummdi, A.; Aljaedi, A.; Bassfar, Z.; Shaukat Jamal, S.; Mazyad Hazzazi, M.; Rehman, M.U. Unif-NTT: A Unified Hardware Design of Forward and Inverse NTT for PQC Algorithms. *IEEE Access* **2024**, *12*, 94793–94804. [CrossRef]
27. Sun, J.; Bai, X.; Kang, Y. An FPGA-Based Efficient NTT Accelerator for Post-Quantum Cryptography CRYSTALS-Kyber. In *Proceedings of the 2023 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA), Hefei, China, 27–29 October 2023*; IEEE: New York, NY, USA, 2023; pp. 142–143.
28. Sonbul, O.S.; Rashid, M.; Jaffar, A.Y. Accelerating CRYSTALS-Kyber: High-Speed NTT Design with Optimized Pipelining and Modular Reduction. *Electronics* **2025**, *14*, 2122. [CrossRef]
29. Aikata, A.; Mert, A.C.; Imran, M.; Pagliarini, S.; Roy, S.S. KaLi: A Crystal for Post-Quantum Security Using Kyber and Dilithium. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 747–758. [CrossRef]
30. Nguyen, T.H.; Kieu-Do-Nguyen, B.; Pham, C.K.; Hoang, T.T. High-Speed NTT Accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium. *IEEE Access* **2024**, *12*, 34918–34930. [CrossRef]
31. Matteo, S.D.; Sarno, I.; Saponara, S. CRYPHTOR: A Memory-Unified NTT-Based Hardware Accelerator for Post-Quantum CRYSTALS Algorithms. *IEEE Access* **2024**, *12*, 25501–25511. [CrossRef]
32. Bevin, R.; Khalid, A.; Imran, M.; O'Neill, M. Accelerating CRYSTALS-Kyber and Dilithium via a Single Montgomery Reduction ISE on RISC-V. In Proceedings of the 2025 IEEE 38th International System-on-Chip Conference (SOCC), Dubai, United Arab Emirates, 29 September–1 October 2025; pp. 1–6. [CrossRef]
33. Alsuhli, G.; Saleh, H.; Al-Qutayri, M.; Mohammad, B.; Stouraitis, T. Efficient NTT/INTT processor for FALCON post-quantum cryptography. *J. Inf. Secur. Appl.* **2025**, *93*, 104177. [CrossRef]
34. Huang, J.; Zhao, H.; Zhang, J.; Dai, W.; Zhou, L.; Cheung, R.C.C.; Koç, c.K.; Chen, D. Yet Another Improvement of Plantard Arithmetic for Faster Kyber on Low-End 32-bit IoT Devices. *Trans. Info. For. Sec.* **2024**, *19*, 3800–3813. [CrossRef]

35. Barrett, P. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology—CRYPTO '86*; Springer: Berlin/Heidelberg, Germany, 1987; pp. 311–323.
36. Fritzmann, T.; Sigl, G.; Sepúlveda, J. RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2020*, 239–280. [CrossRef]