information

MDPI

Article

A Pipelined Hardware Design of FNTT and INTT of CRYSTALS-Kyber PQC Algorithm

Muhammad Rashid [1,*], Omar S. Sonbul [1], Sajjad Shaukat Jamal [2], Amar Y. Jaffar [1] and Azamat Kakhorov [3]

[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 Mathematics, College of Science, King Khalid University, Abha 61413, Saudi Arabia; shussain@kku.edu.sa
[3] Department of Artificial intelligence, Tashkent State University of Economics, Tashkent 100066, Uzbekistan; a.kaxorov@tsue.uz
* Correspondence: mfelahi@uqu.edu.sa

Abstract: Lattice-based post-quantum cryptography (PQC) algorithms demand number theoretic transform (NTT)-based polynomial multiplications. NTT-based polynomials' multiplication relies on the computation of forward number theoretic transform (FNTT) and inverse number theoretic transform (INTT), respectively. Therefore, this work presents a unified NTT hardware accelerator architecture to facilitate the polynomial multiplications of the CRYSTALS-Kyber PQC algorithm. Moreover, a unified butterfly unit design of Cooley–Tukey and Gentleman–Sande configurations is proposed to implement the FNTT and INTT operations using one adder, one multiplier, and one subtractor, sharing four routing multiplexers and one Barrett-based modular reduction unit. The critical path of the proposed butterfly unit is minimized using pipelining. An efficient controller is implemented for control functionalities. The simulation results after the post-place and -route step are provided on Xilinx Virtex-6 and Virtex-7 field-programmable gate array devices. Also, the proposed design is physically implemented for validation on Virtex-7 FPGA. The number of slices utilized on Virtex-6 and Virtex-7 devices is 398 and 312, the required number of clock cycles for one set of FNTT and INTT computations is 1410 and 1540, and the maximum operating frequency is 256 and 290 MHz, respectively. The average figure of merit (FoM), where FoM is the ratio of throughput to slices, illustrates 62% better performance than the most relevant NTT design from the literature.

Keywords: Number theoretic transform (NTT); CRYSTALS-Kyber; hardware accelerator; PQC; FPGA

Academic Editor: Marco Baldi

Received: 16 October 2024
Revised: 30 November 2024
Accepted: 4 December 2024
Published: 31 December 2024

Citation: 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* **2025**, *16*, 17. https://doi.org/10.3390/info16010017

1. Introduction

The use of Shor's algorithm [1] on a quantum computer poses a severe security threat to public-key cryptography standards like RSA [2] and elliptic curve cryptography (ECC) [3,4]. Google and IBM have significantly contributed to developing quantum computers [5,6]. For instance, a 53-qubit Google Sycamore quantum computer can break the security of RSA and ECC in just 200 s—a task that would take classical computers 10,000 years [5]. IBM has advanced the field with its 100-qubit Eagle chip [7]. These quantum computing breakthroughs are driving designers and researchers to develop new quantum-resistant cryptographic schemes and algorithms.

To this end, the National Institute of Standards and Technology (NIST) launched a competition process in 2017 to establish new quantum-resistant cryptography standards.

After multiple evaluation rounds, NIST finalized four post-quantum cryptographic (PQC) algorithms in 2023: CRYSTALS-Kyber [8], CRYSTALS-Dilithium [9], SPHINCS+ [10], and Falcon [11]. These algorithms heavily rely on number theoretic transform (NTT)-based polynomial multiplications to ensure fast and efficient cryptographic operations, making them especially well-suited for hardware acceleration on field-programmable gate array (FPGA) platforms. In particular, the CRYSTALS-Dilithium and CRYSTALS-Kyber schemes, commonly used for digital signatures and key exchange, benefit significantly from NTT on FPGA devices. For instance, an FPGA-based NTT module can accelerate polynomial multiplication by utilizing parallel processing and pipelining, achieving faster execution and lower latency than software implementations. This speedup is critical for practical applications like secure messaging and real-time transaction processing, where the overhead from cryptographic operations must remain low.

Beyond PQC, NTT-based polynomial multiplications are essential for fully homomorphic encryption (FHE) schemes [12,13], which enable computation on encrypted data. Hardware-based NTT accelerators on FPGAs can improve the efficiency of FHE schemes, enabling faster-encrypted computations in cloud environments. For instance, FPGA implementations reduce the high computational cost of homomorphic multiplications, a key bottleneck in FHE, allowing real-time encrypted data processing suitable for privacy-preserving applications such as medical data analysis and financial services.

In addition, NTT-based polynomial multiplications play a crucial role in other secure applications across the Internet of Things (IoT), cloud computing, and wireless sensor networks, which require lattice-based algorithms for secure communication [14]. Lattice-based schemes depend on high-speed polynomial multiplications achievable through NTT accelerators. FPGAs can fulfill these requirements, providing energy-efficient, scalable NTT hardware solutions that can be embedded in resource-constrained devices. Consequently, these diverse applications highlight the growing need for efficient, FPGA-based NTT implementations to meet the latency, throughput, and security requirements of modern cryptographic algorithms/protocols.

NTT-based polynomial multiplication requires forward and inverse operations, which are typically executed using several methods. The traditional approach is the zero padding technique, which adds n additional zeros to n-bit input polynomials for multiplications. Zero padding requires more processing time to perform polynomial multiplication. An efficient approach to reducing the zero padding technique's processing time is negative wrapped convolution (NWC), which multiplies the input polynomials by the weights ϕ before and after the NTT computations. This introduces the pre-/post-processing complexity.

In order to simplify the complexities associated with the pre-/post-processing of the NWC method, an efficient alternative is to employ the Cooley–Tukey butterfly unit (CTBU) for forward NTT (FNTT) and the Gentleman–Sande butterfly unit (GSBU) for inverse NTT (INTT) computations [13,15,16]. The literature indicates that CTBU and GSBU naturally eliminate the need for the pre-/post-processing steps required by the NWC method [17,18]. The complete mathematical foundations/derivations that allow CTBU and GSBU to bypass these additional steps are detailed in [19]. There are various configurations for implementing Cooley–Tukey (CT) and Gentleman–Sande (GS) butterflies in the literature. In our approach, the CT butterfly processes inputs in regular order (NO) and outputs them in bit-reversed order (BO), while the GS butterfly takes bit-reversed inputs and outputs them in regular order. To examine a detailed overview of different CT and GS butterfly configurations, refer to [18].

Depending on specific application needs, the CT and GS butterfly units can be implemented on either software or hardware platforms. Software implementations, like those on microcontrollers, offer greater flexibility but are limited in throughput. In contrast,

hardware platforms such as FPGAs and application-specific integrated circuits (ASICs) can achieve much higher throughput. ASIC-based NTT implementations reduce computation time, increase operating frequency, and maximize overall throughput, though they come with high fabrication costs [20,21]. FPGA-based NTT implementations, on the other hand, provide a balanced solution with higher throughput than software, lower implementation costs, and greater market accessibility compared to ASICs [22]. Consequently, this work focuses on implementing the FNTT and INTT operations on the FPGA platform.

1.1. Related Hardware Accelerators of NTT

The existing FNTT and INTT accelerator architectures are summarized in Table 1. Column one shows the reference designs, while the architectures incorporating the memory-based and memory-free designs are shown in columns two to three. The pre-processing and post-processing features are further highlighted in columns four to five. The last column provides the optimization techniques used in the FNTT and INTT accelerators.

Table 1. Summary of the optimization techniques of existing FNTT/INTT accelerators.

Designs	Desigh Type		Pre-/Post-Processing		Other Technique(s)
	MEM-Based	MEM-Free	Needed	Avoided	
[13]	✓	–	–	✓	Pipelining
[23]	✓	–	–	✓	Parallelism
[17]	✓	–	–	✓	Naive
[16]	✓	–	✓	–	Merging NTT layers
[24]	–	✓	–	✓	Distributed memories
[25]	–	✓	✓	–	Distributed memories
[26]	✓	–	✓	–	Pipelining
[27]	✓	–	✓	–	Pipelining
[28]	✓	–	–	✓	Pipelining
[29]	✓	–	–	✓	Radix-2/-4 butterfly
[30]	✓	–	–	✓	K^2-RED
[31]	✓	–	✓	–	Pipelining
[32]	✓	–	–	✓	Pipelining
[33]	✓	–	✓	–	Parallel butterflies
[34]	✓	–		✓	Pipelining
[35]	✓	–	–	✓	Pipelining

An open-source NTT hardware architecture is presented in [13]. This open-source design is flexible and has a parameterized architecture, which supports various implementation parameters suitable for lattice-based cryptographic algorithms and computes FNTT and INTT operations on FPGA devices. A parallel NTT accelerator architecture for CRYSTALS-Kyber is described in [23]. Notably, Reference [17] differentiates between the dedicated and unified implementations of FNTT and INTT operations on FPGA and ASIC platforms through experimental results. A memory-efficient NTT implementation on an ARM Cortex-M4 microcontroller is presented in [16]. In clock cycles, the computation cost is 7725 and 9347 for FNTT and INTT operations, respectively. The design of [24] is specifically tailored to minimize the latency of the CRYSTALS-Kyber PQC algorithm on FPGA.

Three register banks are used in [25] to implement the FNTT operation. One bank is dedicated to storing twiddle factors, while the other two are iteratively reused for intermediate and final results. On an Artix-7 FPGA device, the design of [26] completes the FNTT and INTT operation in 1024 cycles and operates at a 136 MHz frequency. A ping-pong memory access scheme is presented in [27] to implement FNTT and INTT operations. This ping-pong scheme takes 490 clock cycles to compute one FNTT and INTT operation, achieving a 257 MHz frequency on an Artix-7 FPGA.

High-speed NTT designs are described in [28,29]. In [28], 4 to 16 butterfly units significantly reduce the computation time for FNTT and INTT operations. The design in [29] is tailored to CRYSTALS-Kyber and CRYSTALS-Dilithium algorithms.

A reconfigurable and high-speed NTT accelerator is detailed in [30]. This design achieves reconfigurability by incorporating a one-bit mode signal to switch between FNTT and INTT operations. With the aim of attaining higher speeds, a 2×2 butterfly core is implemented which merges two NTT layers, allowing for two butterfly operations per layer, effectively reducing the number of clock cycles. The overall number of clock cycles of the FNTT and INTT operations is reduced using four instances of the 2×2 butterfly unit in [30]. The pipelining optimizes the critical path. These strategies operate the NTT design of [30] at 222 MHz on Artix-7 FPGA, requiring only 324 clock cycles for one FNTT and INTT computation.

The design of [31] has presented an FPGA-based run-time configurable and parallel polynomial multiplication architecture that supports six different parameter sets (to be used) in FHE and PQC cryptosystems. In [32], NTT design is presented specific to the CRYSTALS-Kyber algorithm.

An attractive low-cost and high-speed NTT design is proposed in [33], featuring a Montgomery-based reduction module with a pipelined structure for modular multiplication. The design also incorporates a precomputed scheme to minimize hardware resources and a block storage technique to enhance throughput by enabling simultaneous data reading and writing operations. The work in [34] has implemented a memory-unified "CRYPHTOR" NTT accelerator for CRYSTALS-Kyber and CRYSTALS-Dilithium algorithms. The design of [35] accelerates FNTT and INTT operations supporting polynomials of different degrees. The latency is minimized by merging the CTBU and GSBU operations. This eliminates the need for bit-reversing operations in the NTT.

1.2. Limitation(s) of Existing NTT Designs

Section 1.1 show that the unified designs of the CTBU and GSBU frequently utilize a single adder, multiplier, and subtractor operator to accelerate FNTT and INTT operations [13,16,26,27,31]. While using single operators reduces hardware resource usage, it comes at the cost of performance, particularly in terms of operating frequency. The performance impact arises because the CTBU requires multiplication before addition and subtraction, whereas the GSBU requires multiplication after these operations. Implementing this with a single set of arithmetic operators necessitates routing multiplexers with feedback inputs, which can negatively impact the overall circuit frequency. To mitigate this, pipelining techniques have been extensively employed [26,33,36]. In addition to pipelining, some state-of-the-art NTT accelerators have incorporated multiple butterfly units in parallel to enhance performance in terms of clock cycles. For instance, the design in [30] uses four butterfly units, while the design in [28] employs sixteen. These architectures optimize computation time but require more hardware resources. Hence, an optimal solution that balances hardware area and performance must be explored.

1.3. Novelty and Contributions

To provide novelty in our work, we first outline the architectural distinctions between our design and existing state-of-the-art NTT accelerators below.

- The design in [25] computes only the FNTT operation without addressing the INTT. Our NTT design supports both FNTT and INTT operations, which is crucial, as the CRYSTALS suite requires multiple computations of both FNTT and INTT during implementation [8,9].

- Even though the existing NTT accelerators have used pipelining in their butterfly unit architectures, we have implemented extensive pipeline stages (up to six) within our unified butterfly unit design, distinguishing our architecture from state-of-the-art accelerators.
- The design of [28,30] uses 4 and 16 butterfly units, respectively, whereas we use only one unit.

To the best of our knowledge, an NTT accelerator incorporating extensive pipelining has not been explored before on hardware platforms such as FPGA. This highlights the novelty of our work. Therefore, our contributions are given below.

- A unified NTT hardware accelerator architecture is proposed to facilitate polynomial multiplications in the CRYSTALS-Kyber PQC algorithm. The parameters used are $n = 256$ and $q = 3329$, respectively.
- A unified butterfly unit (U-BTF) design of the CT and GS configurations is proposed to implement the FNTT and INTT operations using one adder, multiplier, and subtractor. Additionally, we shared four routing multiplexers and one Barrett-based modular reduction unit across the arithmetic operators (adder, multiplier, and subtractor) in the butterfly unit to generate the desired outputs.
- We used pipeline registers in the datapath of the proposed butterfly unit architecture to maximize the operating frequency.
- A finite-state machine (FSM) controller is implemented for control functionalities.
- A figure of merit (FoM) in the ratio of throughput to area is defined to evaluate the performance of the proposed design.

The register-transfer level (RTL) code is written in Verilog-HDL using Vivado 2018.1. The simulation results are reported for Virtex-6 and Virtex-7 FPGA devices after the post-place and -route implementation step. In addition, the proposed NTT design is physically implemented on the Xilinx Virtex-7 FPGA board for validation. Our NTT design utilizes 398 and 312 slices on Virtex-6 and Virtex-7 devices, respectively. It takes 1410 and 1540 clock cycles for one set of FNTT and INTT operation computations, respectively. The maximum operating frequency on Virtex-6 is 256 MHz, and on Virtex-7, it is 290 MHz. Overall, the proposed NTT accelerator (62%) outperforms in terms of average FoM compared to the NTT design of [13]. Therefore, the implementation results and comparisons to existing NTT designs show that the proposed NTT accelerator suits applications that demand lower hardware resources with a reasonable throughput.

The remainder of this article is structured as follows: Section 2 presents the background related to NTT. Section 3 details the proposed NTT accelerator architecture. Results and comparisons are discussed in Section 4. Finally, the article is concluded in Section 5.

2. NTT Background

In the regular algebraic domain, multiplying two polynomials $a(x)$ and $b(x)$ over the ring $R_{n,\,q}$ can be expressed as $c(x) = a(x) \cdot b(x) \bmod (x^{n+1}, q)$. The resulting polynomial $c(x)$ coefficients must lie in $[0, q]$ range, and the degree of $c(x)$ remains less than n. Various methods can perform this multiplication, including but not limited to schoolbook, Karatsuba, and Toom–Cook, but with higher computational complexity overhead (The computational complexity of schoolbook, Karatsuba, and Toom–Cook multipliers is comprehensively evaluated in [37,38].) For example, the computational complexity of the schoolbook and Karatsuba multipliers is $O(n^2)$ and $O(n^{1.59})$, respectively [15], while the complexity of the NTT-based multiplication is $O(n \log n)$.

Multiplying polynomials using the NTT method involves three steps. First, the input polynomials $a(x)$ and $b(x)$ need to transform from the regular algebraic domain into NTT-domain equivalents, $\tilde{a}(x)$ and $\tilde{b}(x)$, respectively. This process is the FNTT computation.

Then, within the NTT domain, a coefficient-wise multiplication is performed to produce the resultant polynomial $\tilde{c}(x)$. Finally, an INTT transformation is applied to convert $\tilde{c}(x)$ back to its original form in the algebraic domain, resulting in the polynomial $c(x)$. This process is the INTT. This work computes only the FNTT and INTT operations.

The FNTT of an n-degree polynomial $f = \sum_{i=0}^{n-1} f_i x^i$ is formally defined using Equation (1):

$$\text{FNTT} = \hat{f} = NTT(f) = \sum_{i=0}^{n-1} \hat{f}_i x_i \tag{1}$$

In Equation (1), $\hat{f}_i = \sum_{j=0}^{n-1} f_j \zeta^{(2i+1)j}$, where ζ is the $2n$-th primitive root of unity. The INTT can be computed using Equation (2):

$$\text{INTT} = f = NTT^{-1}(\hat{f}) = \sum_{i=0}^{n-1} f_i x_i \tag{2}$$

In Equation (2), $f_i = n^{-1} \sum_{j=0}^{n-1} \hat{f}_j \zeta^{-i(2j+1)}$, where ζ is the $2n$-th primitive root of unity.

Equations (1) and (2) can be implemented using various algorithms in the literature. However, we have used a traditional iterative NTT algorithm that is given in Algorithm 1.

Algorithm 1 Iterative NTT Algorithm [15].

Require: An n-element vector $x = [x_0, \ldots, x_{n-1}]$ *where* $x_i \in [0, q-1]$
Require: n (power of 2), modulus q ($q \equiv 1 (mod\ 2n)$)
Require: g (precomputed table of $2n$-th roots of unity, bit-reversed order)
Ensure: $x \leftarrow NTT(x)$
1: $t \leftarrow n/2$; $m \leftarrow 1$
2: **while** $(m < n)$ **do**
3: $k \leftarrow 0$
4: **for** $(i \leftarrow 0;\ i < m;\ i \leftarrow i+1)$ **do**
5: $w \leftarrow g[m+i]$
6: **for** $(j \leftarrow k;\ j < k;\ j \leftarrow j+1)$ **do**
7: $a \leftarrow x[j]$ ▷ Butterfly starts
8: $b \leftarrow x[j+t] \cdot w$
9: $x[j] \leftarrow a + b$
10: $x[j+t] \leftarrow a - b$ ▷ Butterfly ends
11: **end for**
12: $k \leftarrow k + 2t$;
13: **end for**
14: $t \leftarrow t/2$; $m \leftarrow 2m$
15: **end while**

The implementation of Algorithm 1 requires two steps. The first one is the butterfly operations used to compute FNTT and INTT operations. The second is the associated address generation for read/write intermediate results of the FNTT and INTT operations. More insights are given below.

Lines 6 to 11 in Algorithm 1 show the butterfly unit operations. Now, in the current form of Algorithm 1, we show the operations of the CTBU in lines 6 to 11. These operations are for FNTT computation and are shown in Equation (3). To compute the INTT operation, these lines must be replaced with the operations of Equation (4) in Algorithm 2. The operations in the remaining lines of Algorithm 1, apart from lines 6 to 11, primarily involve initialization or generating memory addresses for read/write operations.

$$\text{CTBU} = (u + tw)\ mod\ q\ \ \&\ \ (u - tw)\ mod\ q \tag{3}$$

$$\text{GSBU} = (u + t)\ mod\ q\ \ \&\ \ w(u - t)\ mod\ q \tag{4}$$

3. Proposed NTT Hardware Accelerator

Our NTT hardware accelerator, illustrated in Figure 1, is composed of three key components: (i) a memory unit, (ii) a unified butterfly unit (i.e., U-BTF), and (iii) an FSM-based control unit. The FSM-based control unit orchestrates the FNTT/INTT operations, generating control signals for read and write addresses for the memory unit; addr-1 and addr-2 are read/write addresses to Block-RAMs, i.e., BRAM-1 and BRAM-2 for polynomial coefficients, and adr-rom is the read address used to access the corresponding twiddle factor from the ROM unit. Similarly, din1 and din2 are the data inputs to BRAM-1 and BRAM-2. Inside the memory unit, two multiplexers, i.e., m1 and m2, select the correct polynomial coefficients u and t for the U-BTF unit. The twiddle factor ω comes from the ROM unit to U-BTF. In short, the memory unit stores data at different FNTT and INTT computation stages, including initial inputs, intermediate values, and final outputs. The U-BTF executes the arithmetic operations required for (both) the FNTT and INTT operations and produces U and T signals for the FSM unit to write back on the memory unit. More detailed descriptions of these components are provided in upcoming sections.

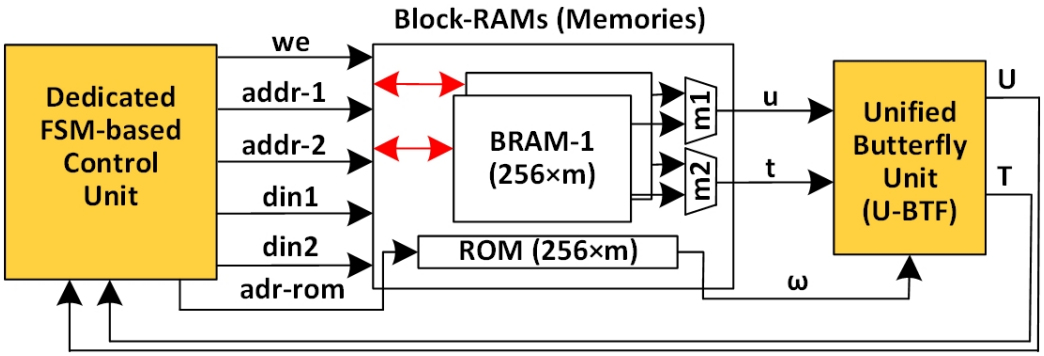

Figure 1. Proposed NTT architecture for FNTT and INTT computations. The orange-filled boxes indicate our contributed blocks in this work.

3.1. Memory Unit

As presented in Figure 1, the memory unit of our proposed NTT accelerator contains three BRAM instances to store results at different implementation stages. We remind the reader again that the parameters of the CRYSTALS-Kyber PQC algorithm are selected for FNTT and INTT implementation, which involves 12-bit 256 input coefficients. These parameters show that NTT implementation requires a memory block of 256×12, where 256 is the total number of polynomial coefficients, and 12 is the size of one coefficient. With this in mind, each instance of BRAM in Figure 1 consists of 256 addresses, each capable of storing 12 bits. Therefore, the total size of each BRAM is 256×12.

Rather than focusing on the size of the BRAMs, it is crucial to emphasize the type of memory used, specifically single or dual ports. In our NTT accelerator, depicted in Figure 1, we utilize two dual-port memories: BRAM-1 and BRAM-2. The dual-port configuration is essential because, during each clock cycle, the FNTT and INTT computations require two coefficients as inputs, as indicated in Equations (3) and (4), where u and t represent the polynomial coefficients. A third memory, a single-port ROM with a size of 256×12, is used to store the precomputed twiddle factors (i.e., w) necessary for the FNTT and INTT computations.

The BRAM-1 stores the 256 input polynomial coefficients using a control signal 'd-load'. This data loading takes 256 clock cycles. Similarly, to load the twiddle factors into the utilized ROM, we used a 't-fact' signal. This also needs 256 clock cycles. These 'd-load' and 't-fact' signals are not shown in Figure 1. In Algorithm 2, the precomputed twiddle factors are represented by g and w. Here, g holds all 256 twiddle factor values, while w stores the

corresponding 12-bit twiddle factor for each subsequent round (or each *for* loop iteration in Algorithm 1).

After initial data loading, the control unit generates the read/write control signals for FNTT and INTT computations. Note that BRAM-1 and BRAM-2 operate in a ping-pong fashion, which means reading from one memory and writing on another after the computations. To highlight this ping-pong communication, we used a red double arrow to show the bi-directional communication between the control unit and the memory block of Figure 1.

3.2. Proposed U-BTF Accelerator Design

The core building block of the proposed NTT accelerator of Figure 1 is the U-BTF, illustrated in Figure 2. The U-BTF computes the FNTT and INTT operations, implementing the arithmetic operations of Equations (3) and (4).

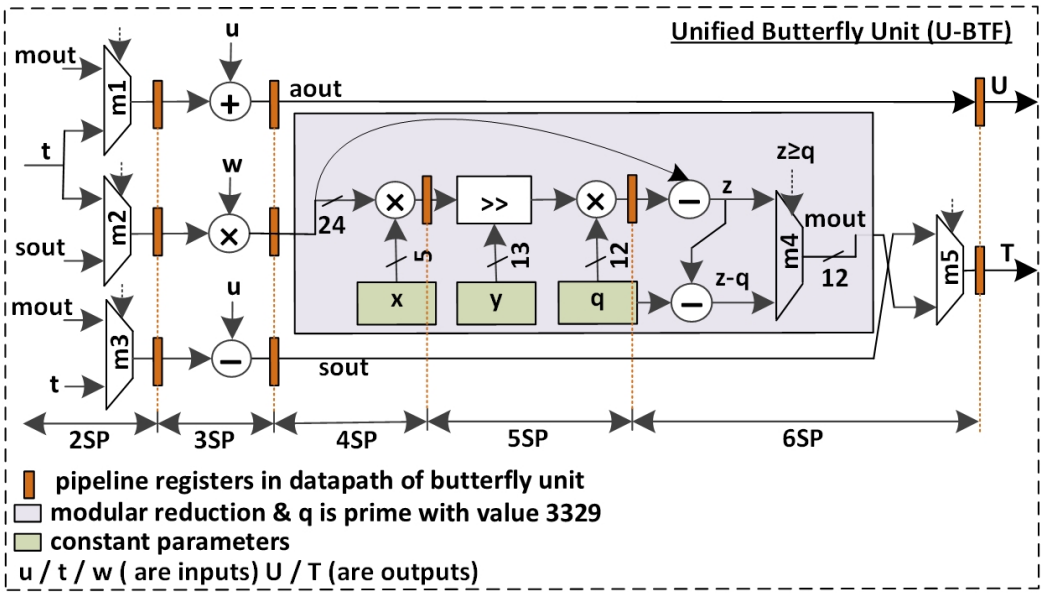

Figure 2. U-BTF: A proposed 6-stage pipelined unified architecture supporting CTBU and GSBU butterfly configurations for FNTT and INTT operation computations.

Figure 2 comprises three 12-bit inputs and two 12-bit outputs. The inputs are u, t, and w. Here, u and t are polynomial coefficients, and w is the twiddle factor. Instead of these inputs, another one-bit 'mode' input signal determines the FNTT or INTT computation based on its value (0 or 1). The 'mode' signal is not shown in Figure 2, but it is the select line of the routing multiplexers m1, m2, m3, and m5. Despite the inputs, the 12-bit outputs of the U-BTF architecture are U and T.

In addition to the inputs and outputs, three arithmetic operators are used: (i) an adder '+', (ii) a subtractor '−', and (iii) a multiplier '×'. These arithmetic operators are placed outside the purple block of Figure 2. Also, these arithmetic operators are re-utilized to compute the FNTT and INTT operations. Note that these arithmetic operators are dedicated to implementing the corresponding operations of FNTT and INTT of Equations (3) and (4), respectively. In our hardware implementation, we implement these arithmetic operators similar to the reference work of [17] using the Verilog-HDL-supported operators of '+', '−', and '×'.

Figure 2 offers the choice between FNTT and INTT implementation using the arithmetic operators discussed above and a one-bit 'mode' signal and routing multiplexers of m1, m2, m3, and m5. When the value of the 'mode' signal becomes 0, these routing multiplexers operate to compute the FNTT operation. For 'mode' signal 1, the m1, m2,

m3, and m5 multiplexers operate to implement INTT. The input and output connections between the arithmetic operators and the routing multiplexers are illustrated in Figure 2.

Up to this point, we have discussed the implementation of Equations (3) and (4), focusing primarily on the arithmetic operators and routing multiplexers without addressing the modular reduction, or *mod q* operation. From Equation (3), it is clear that two *mod q* operations are required, and similarly, Equation (4) also necessitates two *mod q* operations. This means a total of four *mod q* operations would typically be needed. However, in the U-BTF architecture of Figure 2, purple block, we optimized the design by using just one *mod q* instead of four. This approach significantly reduces the required hardware resources.

The *mod q* operation in Figure 2 ensures that computations should remain within the specified modulus range, i.e., $q = 3329$. The purple block in Figure 2 denotes the architecture of the *mod q* operation, a Barrett-based modular reduction, expressed mathematically in Algorithm 2. The reduction algorithm and corresponding architecture require two multipliers and two subtractors, along with a shift operator for the computation.

Algorithm 2 Barrett Modular Reduction [25].

Require: A
Ensure: $Z \leftarrow A \ mod \ q$
1: **Initialization** $\Rightarrow$ $K = \log_2(A)$, $X = \frac{2^K}{q}$, $q = 3329$
2: $Y \leftarrow (A \times X) \gg K$
3: $Z \leftarrow A - Y \times q$
4: **if** $Z \geq q$ **then**
5: $\quad Z \leftarrow Z - q$
6: **end if**

The brown-filled vertical bars in Figure 2 denote the pipeline registers we placed in the datapath of the U-BTF architecture to optimize the critical path and operating frequency. The pipeline stages in Figure 2 are denoted with 2SP (2-stage pipelining), 3SP (3-stage pipelining), 4SP (4-stage pipelining), 5SP (5-stage pipelining), and 6SP (6-stage pipelining). Consequently, implementing operations of Equations (3) and (4) using a 6SP architecture of U-BTF improves the circuit frequency.

3.3. Control Unit (CU)

We implemented a dedicated control unit based on a finite state machine to generate the control signals necessary for implementing Algorithm 1. The control unit is responsible for managing several key tasks: (i) it manages the loading of input data, including polynomial coefficients and twiddle factors, into the appropriate BRAM or ROM memories; (ii) it handles the execution of intermediate results for FNTT and INTT computations; and (iii) it handles the retrieval of results from the memories to the output of the proposed NTT accelerator. In addition, it provides other related control functionalities, such as handling pipelining of the U-BTF.

Implementing Algorithm 1, line 1 initializes the counters t and m with their required starting values, $n/2$ and 1, respectively. These initializations are essential for setting up the algorithm, and they require one clock cycle for implementation.

Algorithm 1 contains three conditional statements, i.e., one '*while*' and two 'for' loops. These statements, along with the corresponding assignments in lines 3, 12, and 14, are responsible for determining the memory addresses used for reading and writing data, including polynomial coefficients and twiddle factors. In the hardware implementation, the '*while*' and 'for' loops are managed by repeatedly executing the relevant FSM states. The counters t, m, j, and k, essential for performing the NTT operations, are set or reset within their corresponding FSM states.

The statements in lines 6 to 11 in Algorithm 1 are related to the U-BTF. Therefore, to implement the operations of Equations (3) and (4) for FNTT and INTT computations, the corresponding clock cycles can be calculated using Equation (5) and Equation (6), respectively. Here in Equations (5) and (6), the value of $n = 256$ and of $s = 7$, where n shows the polynomial degree or total polynomial coefficients, and s denotes the NTT stages to compute. A constant value of 2 in Equations (5) and (6) appears due to pipelining, meaning that one clock cycle is needed to enter the pipeline stage and one cycle to leave the pipelining. Hence, the total clock cycle cost of Figure 2 for FNTT and INTT computations is 898 and 1028, respectively.

$$\text{Cycles (for FNTT)} = (\frac{n}{2} \times s) + 2 \tag{5}$$

$$\text{Cycles (for INTT)} = [\{(\frac{n}{2} \times s) + 2\} + (\frac{n}{2}) + 2] \tag{6}$$

The total computational cost of Figure 1 in terms of clock cycles includes three operations: (i) initial data loading into the BRAM or ROM, (ii) FNTT and INTT computations, and (iii) final data loading from the BRAM to design output pins. The initial data loading requires 128 clock cycles to load 256 polynomial coefficients. Note that we need 128 cycles for initial polynomial coefficients loading, as our NTT design of Figure 1 incorporates two dual-port BRAMs as memories. We are utilizing two ports simultaneously to load initial coefficients into the corresponding BRAM memory. On the other hand, the ROM used as memory for twiddle factors in Figure 1 is a single port. Therefore, loading the corresponding 256 twiddle factors into the used ROM requires 256 clock cycles. The clock cycle cost for FNTT and INTT computation is already described in the manuscript, being 898 and 1028, respectively. Finally, 128 clock cycles are needed to load FNTT- and INTT-transformed polynomial coefficients as output from the corresponding BRAM memory to the NTT accelerator output pins. In total, FNTT and INTT take 1410 (128 + 256 + 898 + 128) and 1540 (128 + 256 + 1028 + 128) clock cycles for computations, respectively.

4. Results, Comparisons, and Discussions

The RTL code for the NTT design was written in Verilog HDL using a Vivado 2018.1. Moreover, for both simulations and design validations, we used the Vivado implementation tool. The simulation results were compared to the equivalent C/C++ implemented NTT design for design verification. To implement the NTT design on a real FPGA board, we used a serial bit-by-bit UART communication protocol, which provides inputs to the design and, after the computations, obtains outputs back from the targeted board. Note that the clock cycle counts and latency results in the upcoming sections are reported without the cost of the loading inputs/outputs to/from the targeted board. Therefore, we provide the implemented results in Section 4.1, while the comparison to state-of-the-art NTT accelerators is shown in Section 4.2.

4.1. Implementation Results

Before presenting the complete implementation results of our NTT accelerator architecture, we first provide an area breakdown (in slices) of the design shown in Figure 1 for Xilinx Virtex-6 (xc6vlx130t-3ff1156) and Virtex-7 (xc7vx690tffg1930-3) FPGA devices, as summarized in Table 2. The design hierarchy is detailed in the first column, while the corresponding slice usage for the Virtex-6 and Virtex-7 devices is listed in the second and third columns, respectively. Table 2 indicates that the U-BTF occupies approximately 67% of the total area of the NTT design on both Virtex-6 and Virtex-7 devices. Additionally, the FSM controller accounts for around 19% of the slices on these FPGA devices, which is

notably higher than the area utilized by the *mod q* operation. It is important to note that this area breakdown excludes the cost of the instantiated BRAMs and ROM memories.

Table 2. Slices breakdown of our NTT design for Virtex-6 and Virtex-7 FPGA devices.

Design Hierarchy	Virtex-6	Virtex-7
NTT accelerator design (of Figure 1)	398	312
⌊ U-BTF (design of Figure 2)	269 (67.59%)	207 (66.35%)
⌊ *mod q*	56 (14.08%)	44 (14.11%)
⌊ FSM controller	73 (18.35%)	61 (19.56%)

Table 3 shows the performance results for two cases: first, the case when using the clock cycles of U-BTF of Figure 2 and second, the NTT design of Figure 1. Moreover, we used the Virtex-6 and Virtex-7 FPGA devices for the implementations. Column one shows the targeted platform. The NTT operation in FNTT and INTT is shown in column two. In columns three to five, we show the FPGA area results in slices, LUTs and FFs. The memory cost of the implemented design in BRAMs and ROM utilization is shown in columns six and seven, respectively. Similarly, the timing results in clock cycles, the circuit frequency (in MHz), the computation time, and the throughput are given in columns eight to eleven. The computation time is the latency, which ensures one FNTT and INTT operation computation in µs, calculated using Equation (7). Additionally, throughput (in Kbps) is computed using Equation (8). Column twelve provides the FoM, the throughput ratio with slices, calculated using Equation (9). Finally, the last column shows the average FoM results, which are considered a metric for evaluating the overall performance of the NTT design. Let us explain the performance results for the two cases below.

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

$$\text{Throughput } (Kbps) = \frac{1}{\text{Latency } (\mu s)} \times 10^3 \tag{8}$$

$$\text{FoM} = \frac{\text{Throughput}}{\text{Slices}} \tag{9}$$

Table 3. Performance results of our NTT architecture using $n = 256$ and $q = 3329$ (after the post-place and -route) on Xilinx FPGA devices. F and I denote the NTT operations in forward (FNTT) and inverse (INTT), respectively. SLS, LUT, FF, M1, and M2 show the slices, look-up tables, flip-flops, BRAMs, and ROM, respectively. Similarly, CCs, Freq, Lat, and TP denote the clock cycles, frequency, latency, and throughput, respectively. V6 and V7 are the Virtex-6 and Virtex-7 FPGA devices, respectively. The blue check mark (✓) indicates the higher performance in terms of frequency, latency, throughput, FoM, and average FoM (Avg-FoM).

Device	Design	Hardware Resources					Timing-Related Results				FoM	Avg-FoM
		Area			Memory		CCs	Freq	Lat	TP		
		SLS	LUT	FF	M1	M2		MHz	µs	Kbps		
Performance results when using clock cycles of U-BTF of Figure 2												
V6	F	398	1592	283	2	1	898	256	3.50	285.71	717.86	672.20
	I						1028		4.01	249.37	626.55	
V7	F	312	1236	214	2	1	898	290	3.09✓	323.62✓	1037.24✓	971.31✓
	I						1028	✓	3.54✓	282.48✓	905.38✓	

Table 3. *Cont.*

Device	Design	Hardware Resources					Timing-Related Results				FoM	Avg-FoM
		Area			Memory		CCs	Freq	Lat	TP		
		SLS	LUT	FF	M1	M2		MHz	µs	Kbps		
Performance results when using clock cycles of NTT design of Figure 1												
V6	F	398	1592	283	2	1	1410	256	5.50	181.81	456.80	437.42
	I						1540		6.01	166.38	418.04	
V7	F	312	1236	214	2	1	1410	290	4.86✓	205.76✓	659.48✓	631.53✓
	I						1540	✓	5.31✓	188.32✓	603.58✓	

In both Figures 1 and 2, the hardware resource utilization remains consistent across the targeted Virtex-6 and Virtex-7 FPGA devices, as reported in Table 3. Specifically, on the Virtex-6 FPGA, the area utilization metrics for slices, LUTs, and FFs are 398, 1592, and 283, respectively. On the Virtex-7 FPGA, the corresponding area costs for slices, LUTs, and FFs are slightly lower, as the implementation platforms differ, with values of 312, 1236, and 214, respectively.

Comparatively, our Virtex-6 implementation of Figures 1 and 2 utilizes 21% ($\frac{398-312}{398} \times 100$), 22% ($\frac{1592-1236}{1592} \times 100$), and 24% ($\frac{283-214}{283} \times 100$) higher slices, LUTs, and FFs compared to Virtex-7 FPGA implementation. In other words, the Virtex-7 implementation is 21%, 22%, and 24% more efficient regarding slices, LUTs, and FFs than the Virtex-6 implementation. This area difference comes due to the use of distinct implementation devices. The Virtex-6 and Virtex-7 devices are built on 40 nm and 28 nm process technologies, respectively. Regardless of the slices, LUTs, and FFs, the area utilization for the proposed NTT architecture in BRAMs and ROM memory units is two and one, respectively, as shown in columns six and seven of Table 3.

Regarding the clock cycles, the FNTT and INTT computations need 898 and 1028 cycles when the initial loading cost to the memories is not included. Similarly, when we include the initial memory loading cost and the cost for loading outputs into the design pins as output, the required number of clock cycles for one FNTT and INTT computation is 1410 and 1540, respectively. We provide more details to compute clock cycles earlier in Section 3.3.

Concerning the circuit frequency, our U-BTF and NTT designs can operate on a maximum frequency of 256 MHz and 290 MHz on Virtex-6 and Virtex-7 devices, respectively. These higher operating frequencies are achieved due to 6-stage pipelining in the U-BTF design. From these values of circuit frequencies, it can be observed that we have a frequency difference of 34 MHz when moving from Virtex-6 to Virtex-7 devices. For the same Virtex-6 to Virtex-7 devices, a frequency difference of 31 MHz is observed in [39], where the authors utilized a non-pipelined NTT design, along with three RegBanks instead of BRAMs, for implementation. While using the RegBanks in [39], the authors reported that the frequency difference when moving from Virtex-6 to Virtex-7 could be increased if BRAMs were utilized, as BRAMs have inherently better read/write access time, which can affect the circuit frequency. In this work, we experimented with BRAMs and received an almost identical frequency difference of 34 MHz compared to [39], where the difference was 31 MHz. It is important to note that for the designs having BRAMs and implemented on Virtex-6 and Virtex-7 devices, the frequency difference when moving from Virtex-6 to Virtex-7 can be reduced by using pipelining.

For Figures 1 and 2, the lower latency or computation time and higher throughput results are achieved for Virtex-7 implementation. The reason for this is the higher operating

frequency of 290 MHz compared to that of 256 MHz for the Virtex-6 device. The blue check marks in the corresponding columns of Table 3 highlight this higher performance.

The last two columns of Table 3 present the performance of the implemented NTT design by combining throughput and area metrics. The FoM is calculated as the throughput ratio to area, and slices are considered a metric in area. A higher FoM indicates better performance. Column twelve in Table 3 shows that the Virtex-7 FPGA implementation achieves a higher FoM, indicating superior performance compared to the Virtex-6 FPGA. Since the NTT design involves both FNTT and INTT computations, we provide the average FoM for these operations to assess the overall performance of the design, which is displayed in the last column of Table 3. Specifically, the U-BTF design shown in Figure 2 on the Virtex-7 device demonstrates a 31% increase in Avg-FoM compared to the Virtex-6 implementation, calculated as $\frac{971.31-672.20}{971.31} \times 100$. Similarly, the NTT design depicted in Figure 1 on the Virtex-7 device also achieves a 31% increase in Avg-FoM over the Virtex-6 implementation, calculated as $\frac{631.53-437.42}{631.53} \times 100$.

4.2. Comparisons

The performance results are compared in Table 4 with existing NTT designs. Column one shows the reference designs, while the targeted FPGA devices are illustrated in column two. In columns three to eight, we provide the hardware cost in slices, LUTs, FFs, digital signal processors (DSPs), BRAMs, and slice-equivalent cost (SEC). Similarly, columns nine to twelve show the timing results regarding clock cycles, operating frequency (in MHz), latency (in µs), and throughput. The FoM is shown in column thirteen. Finally, the last column shows the average FoM of FNTT and INTT operations.

It is essential to note that the proposed NTT design does not use DSP blocks for implementation, while the designs from the literature frequently utilize DSPs for implementation. To give a realistic area comparison, we have calculated the SEC using a reference data sheet of [40]. Then, in the FoM and average FoM calculations using Equation (8), we used SEC as an area metric.

In comparison to [13] on a similar Virtex-7 FPGA, the proposed NTT accelerator is 56% ($\frac{2075-912}{2075}$) more efficient in terms of hardware resources, when considered SEC as comparison metric. This happens because of the utilization of three 36 Kb BRAMs in the reference work. In addition, the reference work has utilized 8 DSP blocks. Our architecture utilized only three 36 Kb BRAMs without using the DSP blocks. As shown in column nine of Table 4, the reference work requires fewer clock cycles for one FNTT and INTT computations than our NTT design. On the other hand, the pipelining in our design benefits in frequency optimization. More precisely, our work is 40% ($\frac{290-174}{290}$) faster in-circuit frequency compared to the reference NTT design. Columns eleven to thirteen in Table 4 show that the proposed NTT design is more efficient in latency, throughput and FoM compared to the NTT accelerator of [13]. One potential reason is that the proposed NTT design is specific to CRYSTALS-Kyber parameters. In contrast, the reference design is flexible and can be employed for diverse applications depending on the user's parameters. In terms of the average FoM, the proposed NTT design is 62% ($\frac{216.05-80.97}{216.05}$) more efficient than the reference work.

Recently, in [17], three designs were implemented. The first one was dedicated to the FNTT computation, the second was dedicated to INTT computation, and the third design was a unified accelerator of FNTT and INTT computations. In Table 4, we provide results for their unified design. Comparatively, our NTT accelerator utilizes lower hardware resources in terms of slices, LUTs, FFs, and DSP blocks. The reason for this is that we use three FPGA-specific BRAMs in our work, while the reference design used distributed RegFiles as memory elements. Instead of the hardware resources, Table 4 shows that our work is more efficient in terms of frequency, latency, and throughput. More precisely, in

terms of Avg-FoM, the reference work is 96% ($\frac{7.27-216.05}{216.05}$) less efficient than our proposed NTT accelerator.

Table 4. Performance comparison with state-of-the-art NTT designs. **SLS**, **LUT**, **FF**, **DSP**, and **M1** show the slices, look-up tables, flip-flops, digital signal processors, and utilized BRAMs, respectively. Similarly, **CCs**, **Freq**, **Lat**, and **TP** denote the clock cycles, frequency, latency, and throughput, respectively. **A7** and **V7** are the Artix-7 and Virtex-7 FPGA devices, respectively. A blue check mark (✓) signifies superior performance compared to existing designs, while a cross mark (✗) indicates areas where performance falls short. All of the evaluated designs use the parameters $n = 256$ and $q = 3329$ from the CRYSTALS-Kyber algorithm. The clock cycles for all of the reference designs are for only the butterfly unit to compute one FNTT and INTT without considering the cost of the read/write from/to memory units. We compare our results (**TW**) with the reference works by accommodating the cost of the read/write operations of instantiated memories.

Designs	Device	Hardware Resources						Timing-Telated Results				FoM	Avg-FoM
		SLS	LUT	FF	DSP	M1	SEC [1]	CCs	Freq MHz	Lat µs	TP Kbps		
[13] †	V7	675 *	2128	1144	8	3	2075	922	174	5.29	189.03	91.08	80.97 ✓ (62%)
[13] ‡								1184		6.80	147.05	70.86	
[17] †	V7	3698	9298	9402	0	0	1950	1410	20	70.50	14.18	7.27	7.27 ✓ (96%)
[25] †	V7	1950 *	7800	–	6	0	2550	–	72	–	–	–	–
[23] †	A7	187	360	145	3	2	887	940	115	8.17	122.39	137.98	122.87 ✓ (43%)
[23] ‡								1203		10.46	95.60	107.77	
[28] †	A7	2713 *	9508	2684	16	35	11,313	69	172	0.40	2500.00	220.98	218.28 ✗ (1.02%)
[28] ‡								71		0.41	2439.02	215.59	
TW †	V7	312	1236	214	0	3	912	1410	290 ✓	4.86	205.76	225.61	216.05
TW ‡								1540		5.31	188.32	206.49	

† for the FNTT operation. ‡ for the INTT operation. * means the approximated slices using LUTs × 0.25 + FFs × 0.125 [40]. [1] SEC denotes the slice-equivalent cost, calculated using (BRAMs × 200) + (DSPs × 100) + slices [41].

For identical parameters of the CRYSTALS-Kyber PQC algorithm, the butterfly unit implemented in the NTT design of [25] supports only the FNTT computation without focusing on the INTT. In contrast, our implemented NTT accelerator in this work supports (both) FNTT and INTT operations. Moreover, we can compare only the hardware resources and circuit frequency of [25] with our design. The other related information for clock cycles and latency parameters has not been reported. Therefore, on a similar Virtex-7 FPGA, the proposed NTT accelerator is 64% ($\frac{2550-912}{2550}$) more efficient in SEC as hardware resources. The reason for this is the use of three 256×12 sizes of RegFiles in the reference work, while we used two BRAMs and one ROM of the same size in our design. Instead of the hardware resources, our pipelined NTT design benefits by optimizing the circuit frequency compared to the design with three RegFiles as memory units. This is evident in column ten of Table 4. In other words, the proposed design is 75% ($\frac{290-72}{290}$) more efficient in operating frequency compared to the design of [25].

The Artix-7-implemented design in [23] is highly optimized for resource optimizations and is more efficient than our Virtex-7-implemented NTT accelerator (see the corresponding hardware resource utilization columns in Table 4). In addition to hardware resources, the NTT design of [23] requires fewer clock cycles than ours (see the corresponding column in Table 4). On the other hand, our 6-stage pipelined NTT design is 60% ($\frac{290-115}{290}$) more efficient in terms of operating frequency. Moreover, the corresponding latency, throughput, and FoM columns in Table 4 reveal that the proposed NTT design is more efficient than

the reference work. Indeed, the reference work is more efficient in terms of hardware resources, while our NTT design is more efficient in terms of timing-related parameters, i.e., latency, throughput, and FoM. The combined throughput and area comparison in terms of an average FoM shows that the proposed NTT design is 43% ($\frac{216.05-122.87}{216.05}$) more efficient.

In a high-speed NTT accelerator implemented on an Artix-7 FPGA, as described in [28], 16 butterfly units were employed to reduce the clock cycles required for (both) the FNTT and INTT operations. In contrast, our architecture, implemented on a Virtex-7 FPGA, utilizes a single 6-stage pipelined butterfly unit, which is iteratively used to compute (both) FNTT and INTT operations. Despite these differences in design, the reference work demonstrates superior efficiency in terms of clock cycles, latency, throughput, and FoM. However, our design excels in terms of hardware resource efficiency, using fewer slices, LUTs, FFs, and SEC. Detailed area and timing results can be found in the relevant columns of Table 4. Due to the pipelining in our design, the proposed NTT architecture achieves a 41% increase in circuit frequency, calculated as $\frac{290-172}{290}$. While the reference design outperforms in terms of latency and throughput, our design offers advantages in circuit frequency and hardware resource utilization. When considering the combined area and throughput in terms of the average FoM, our proposed design shows only a 1.02% degradation in performance compared to the reference work. In other words, the performances of the reference design and our NTT design are nearly identical.

5. Conclusions and Future Directions

This work has presented a unified NTT hardware accelerator architecture for the CRYSTALS-Kyber PQC algorithm. Moreover, a unified butterfly unit design of Cooley–Tukey and Gentleman–Sande configurations is proposed to implement the FNTT and INTT operations using one adder, one multiplier, and one subtractor, sharing four routing multiplexers and one Barrett-based modular reduction unit. The critical path of the proposed butterfly unit is minimized using pipelining. An efficient FSM controller is implemented to provide control functionalities. The post-place and -route results are provided on Xilinx Virtex-6 and Virtex-7 FPGA devices, where the number of utilized slices is 398 and 312, the required number of clock cycles for one FNTT and INTT computation is 1410 and 1540, and the maximum operating frequency is 256 and 290 MHz, respectively. The average-FoM illustrates 62% higher performance than the NTT design of [13]. This work can be extended in future using multiple instances of the U-BTF to minimize the clock cycle counts (like designs of [28,30]). Moreover, in this work, the proposed NTT design supports only the CRYSTALS-Kyber parameters. It can further be enhanced to support the parameters of other PQC algorithms, as implemented in the unified PQC accelerators of [15,34].

Author Contributions: Conceptualization, M.R. and S.S.J.; methodology, S.S.J. and A.K.; software, A.Y.J., O.S.S., and A.K.; validation, M.R., A.Y.J., and O.S.S.; formal analysis, M.R. and S.S.J.; investigation, M.R.; resources, S.S.J.; data curation, A.Y.J., O.S.S., and A.K.; writing—original draft preparation, M.R. and S.S.J.; writing—review and editing, O.S.S.; visualization, O.S.S.; supervision, M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the 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. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* **1997**, *26*, 1484–1509. [CrossRef]
2. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
3. Miller, V.S. Use of Elliptic Curves in Cryptography. *Advances in Cryptology—CRYPTO '85 Proceedings*; Williams, H.C., Ed.; Springer: Berlin/Heidelberg, Germany, 1986; pp. 417–426. Available online: https://link.springer.com/chapter/10.1007/3-540-39799-x_31 (accessed on 11 September 2024).
4. Rashid, M.; Imran, M.; Jafri, A.R.; Al-Somani, T.F. Flexible architectures for cryptographic algorithms—A systematic literature review. *J. Circuits Syst. Comput.* **2019**, *28*, 1930003. [CrossRef]
5. Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J.C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F.G.S.; Buell, D.; et al. Quantum supremacy using a programmable superconducting processor. *Nature* **2019**, *574*, 505–510. [CrossRef] [PubMed]
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. Ball, P. First 100-Qubit Quantum Computer Enters Crowded Race. *Nature* **2021**, *599*, 574. Available online: https://www.nature.com/articles/d41586-021-03476-5 (accessed on 17 September 2024). [CrossRef]
8. Schwabe, P.; Mann, J. CRYSTALS-Kyber: Cryptographic Suite for Algebraic Lattices. Available online: https://pq-crystals.org/kyber/ (accessed on 2 February 2024).
9. Schwabe, P.; Mann, J. CRYSTALS-Dilithium: Cryptographic Suite for Algebraic Lattices. Available online: https://pq-crystals.org/dilithium/ (accessed on 1 February 2024).
10. Hülsing, A.; SPHINCS Team. SPHINCS+: Stateless Hash-Based Signatures. Available online: https://sphincs.org/ (accessed on 24 December 2023).
11. 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 28 December 2023).
12. Duong-Ngoc, P.; Kwon, S.; Yoo, D.; Lee, H. Area-Efficient Number Theoretic Transform Architecture for Homomorphic Encryption. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 1270–1283. [CrossRef]
13. 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]
14. Khalid, A.; McCarthy, S.; O'Neill, M.; Liu, W. Lattice-Based Cryptography for IoT in A Quantum World: Are We Ready? In Proceedings of the 2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI), Otranto, Italy, 13–14 June 2019; pp. 194–199. [CrossRef]
15. 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]
16. Botros, L.; Kannwischer, M.J.; Schwabe, P. Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4. In *Progress in Cryptology—AFRICACRYPT 2019*; Buchmann, J., Nitaj, A., Rachidi, T., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 209–228.
17. 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**, *16*, 485–488. [CrossRef]
18. Satriawan, A.; Mareta, R.; Lee, H. A Complete Beginner Guide to the Number Theoretic Transform (NTT). Cryptology ePrint Archive, Paper 2024/585. 2024. Available online: https://eprint.iacr.org/2024/585 (accessed on 19 August 2024).
19. Zhang, N.; Yang, B.; Chen, C.; Yin, S.; Wei, S.; Liu, L. Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2020*, 49–72. [CrossRef]
20. Imran, M.; Almeida, F.; Raik, J.; Basso, A.; Roy, S.S.; Pagliarini, S. Design Space Exploration of SABER in 65nm ASIC. In Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security, ASHES '21, Seoul, Republic of Korea, 19 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 85–90. [CrossRef]
21. Imran, M.; Almeida, F.; Basso, A.; Roy, S.S.; Pagliarini, S. High-speed SABER Key Encapsulation Mechanism in 65 nm CMOS. *J. Cryptogr. Eng. (JCEN)* **2023**, *13*, 461–471. [CrossRef]
22. Kuon, I.; Rose, J. Measuring the Gap Between FPGAs and ASICs. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2007**, *26*, 203–215. [CrossRef]
23. 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]
24. 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]

25. Khan, S.; Khalid, A.; Rafferty, C.; Shah, Y.A.; O'Neill, M.; Lee, W.; Hwang, S.O. Efficient, Error-Resistant NTT Architectures for CRYSTALS-Kyber FPGA Accelerators. In Proceedings of the 31st IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2023, Dubai, United Arab Emirates, 16–18 October 2023; IEEE: New York, NY, USA, 2023; pp. 1–6.
26. Chen, Z.; Ma, Y.; Chen, T.; Lin, J.; Jing, J. Towards Efficient Kyber on FPGAs: A Processor for Vector of Polynomials. In Proceedings of the 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), Beijing, China, 13–16 January 2020; pp. 247–252. [CrossRef]
27. 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]
28. 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 (DATE), Grenoble, France, 1–5 February 2021; pp. 1020–1025. [CrossRef]
29. 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]
30. Bisheh-Niasar, M.; Azarderakhsh, R.; Mozaffari-Kermani, M. High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography. In Proceedings of the 2021 IEEE 28th Symposium on Computer Arithmetic (ARITH), Lyngby, Denmark, 14–16 June 2021; pp. 94–101. [CrossRef]
31. Mert, A.C.; Öztürk, E.; Savaş, E. FPGA implementation of a run-time configurable NTT-based polynomial multiplication hardware. *Microprocess. Microsyst.* **2020**, *78*, 103219. [CrossRef]
32. Yang, H.; Chen, R.; Wang, Q.; Wu, Z.; Peng, W. Hardware Acceleration of NTT-Based Polynomial Multiplication in CRYSTALS-Kyber. In *Information Security and Cryptology*; Ge, C., Yung, M., Eds.; Springer Nature: Singapore, 2024; pp. 111–129.
33. Xu, C.; Yu, H.; Xi, W.; Zhu, J.; Chen, C.; Jiang, X. A Polynomial Multiplication Accelerator for Faster Lattice Cipher Algorithm in Security Chip. *Electronics* **2023**, *12*, 951. [CrossRef]
34. 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]
35. Liu, S.; Kuo, C.; Mo, Y.; Su, T. An Area-Efficient, Conflict-Free, and Configurable Architecture for Accelerating NTT/INTT. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2024**, *32*, 519–529. [CrossRef]
36. Mert, A.C.; Karabulut, E.; Öztürk, E.; Savaş, E.; Aysu, A. An Extensive Study of Flexible Design Methods for the Number Theoretic Transform. *IEEE Trans. Comput.* **2022**, *71*, 2829–2843. [CrossRef]
37. Imran, M.; Rashid, M. Architectural review of polynomial bases finite field multipliers over GF(2m). In Proceedings of the 2017 International Conference on Communication, Computing and Digital Systems (C-CODE), Islamabad, Pakistan, 8–9 March 2017; pp. 331–336. [CrossRef]
38. Imran, M.; Abideen, Z.U.; Pagliarini, S. A Versatile and Flexible Multiplier Generator for Large Integer Polynomials. *J. Hardw. Syst. Secur.* **2023**, *7*, 55–71.
39. 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]
40. Xilinx. 7 Series FPGAs Data Sheet: Overview. Available online: https://docs.amd.com/v/u/en-US/ds180_7Series_Overview (accessed on 16 April 2024).
41. Liu, W.; Fan, S.; Khalid, A.; Rafferty, C.; O'Neill, M. Optimized Schoolbook Polynomial Multiplication for Compact Lattice-Based Cryptography on FPGA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 2459–2463. [CrossRef]