High-Performance NTT Hardware Accelerator to Support ML-KEM and ML-DSA

Dur E Shahwar Kundi
PQShield
Oxford, United Kingdom
dur-e-shahwar.kundi@pqshield.com

Jose M. Bermudo Mera
PQShield
Leuven, Belgium
jose.mera@pqshield.com

Pierre-Yves Strub
PQShield
Paris, France
pierre-yves.strub@pqshield.com

Michael Hutter
PQShield
Vienna, Austria
michutte@gmail.com

Abstract

Large polynomial multiplications are crucial for Post-Quantum Cryptography standards like Module-Lattice-based Key Encapsulation Mechanism (ML-KEM) and Module-Lattice-based Digital Signature (ML-DSA). These multiplications, being complex, are often accelerated using the Number Theoretic Transform (NTT). This work presents a novel architecture of a high-performance NTT accelerator capable of performing both NTT and inverse NTT operations using a single set of hardware resources. The design makes use of a single butterfly configuration unit to reduce resource requirements and improve critical path. The Multi-path Delay Commutator (MDC) strategy is employed to enable fully pipelined and parallel processing of multiple coefficients, supporting both ML-KEM and ML-DSA computations. Practical results show that our proposed NTT engine requires 3,821 LUTs, 2970 FFs, 20 DSPs, and 5 BRAMs on an AMD Zynq UltraScale+ FPGA, and can run up to 322 MHz. Our design provides the best Area-Time Product (ATP) among current NTT architectures.

CCS Concepts

• **Security and privacy** → **Digital signatures**; **Public key encryption**; • **Hardware** → **Hardware accelerators**.

Keywords

Polynomial Multiplication, NTT, Multi-path Delay Commutator (MDC), ML-KEM, ML-DSA, CRYSTALS-Kyber, CRYSTALS-Dilithium

ACM Reference Format:
Dur E Shahwar Kundi, Jose M. Bermudo Mera, Pierre-Yves Strub, and Michael Hutter. 2024. High-Performance NTT Hardware Accelerator to Support ML-KEM and ML-DSA. In *Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security (ASHES '24), October 14–18, 2024, Salt Lake City, UT, USA.*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3689939.3695785

1 Introduction

The latest advances in the field of quantum computation have posed an imminent threat to the currently deployed Public-Key Infrastructure, underpinning digital security. The Shor's quantum algorithm [18] will be able to solve the mathematical complexity of current Public Key Encryption (PKE) algorithms via quantum computers including factorization of large integers and solving discrete logarithms. In response, various government agencies have acknowledged this threat. National Institute of Standards and Technology (NIST), in particular, started a standardization process in 2016 to develop a new set of quantum-resistant cryptographic algorithms, known as *Post-Quantum Cryptography (PQC)* [16].

In 2022, NIST finalized PQC standards for Key-Encapsulation Mechanism (KEM) and Digital Signature (DSA) schemes [17]. *Lattice-based Cryptography (LBC)* has stood out because of its strong foundation on hard mathematical problems and enhanced performance, leading to Module-Lattice-based KEM (ML-KEM) [7] standard based on CRYSTALS-Kyber [3] and Module-Lattice-based DSA (ML-DSA) [8] standard based on CRYSTALS-Dilithium [4]. LBC constructions also support advanced security features like Identity-based Encryption (IBE) [1], Fully Homomorphic Encryption (FHE)) [11], and Zero-Knowledge Proofs (ZKP) for the post-quantum era.

The *polynomial multiplication*, a key operation in all these LBC schemes, involves convolution and *modular arithmetic*, making it the most *computationally intensive* task in these cryptosystems. Hardware acceleration of this operation is crucial for high performance, with the Number Theoretic Transform (NTT) being the preferred method due to its quasi-linear complexity of $O(n\log(n))$. The NTT is, in fact, the part of ML-KEM and ML-DSA standards [7, 8], supporting lattice dimensions of $n = 256$ with moduli-q of 12-bit and 23-bit, respectively. Table 1 summarizes the frequency of forward and inverse NTT (iNTT) computations required for each parameter set in the respective standards. In contrast, FHE and ZKP require much larger parameters, with n ranging from 2^9 to 2^{17} and modulus-q from 28-bit to 64-bit [13].

A high-speed NTT accelerator is essential for optimizing cryptosystem's performance, as the design choice of it alone will significantly impacts overall efficiency. This work introduces a high-performance, unified NTT/iNTT accelerator that processes two coefficients in parallel and pipelined fashion via Multi-path Delay Commutator (MDC) [19] strategy. The accelerator incorporates

Table 1: Frequency of NTT/iNTT operations.

Standard	Modulus q	n	NTT	iNTT
ML-KEM-512			10×	8×
ML-KEM-768	3329	256	15×	11×
ML-KEM-1024			20×	14×
ML-DSA-44			32×	24×
ML-DSA-65	8380417	256	44×	36×
ML-DSA-87			60×	48×

unique features that minimize hardware resource usage while enhancing performance. ML-KEM and ML-DSA parameter sets are considered to benchmark the area and performance metrics.

The **contributions** of our work are summarized as follows:

- We present a unified NTT hardware architecture that supports both ML-KEM and ML-DSA moduli to be used for all NIST recommended security levels. The architecture makes use of a single datapath for both forward and inverse NTT operations without duplicated control logic.
- Our design applies a fully pipelined, 8-stages deep, Multi-path Delay Commutator (MDC) approach, processing 2 coefficients per cycle (radix-2). This allows the core to continuously consume data input without causing data dependencies in memory accesses and eliminates the need for additional control effort.
- We propose a novel design that uses a single butterfly configuration in each pipeline stage. By using a Cooley-Tukey (CT) butterfly-only configuration, our design eliminates the need to switch configurations when changing the NTT direction (forward or inverse), thereby reducing area overhead. This approach also shortens the critical path by eliminating the need for multiplexers, resulting in a higher operating frequency.
- We present practical implementation results on an AMD Zynq UltraScale+ FPGA, demonstrating high throughput along with best Area-Time Product (ATP) compared to related work as shown in Table 3. Compared to [2], our design uses nearly the same number of LUTs but increases throughput by more than fourfold. Therefore, our design is an excellent choice for applications where LUT resources are the limiting factor.

The rest of the paper is organized as follows. Section 2 provides preliminary details about our used notation and NTT in general. Section 3 provides the analysis of state-of-the-art of NTT architectures and their limitations. The design of our proposed Radix-2 MDC (R2MDC) NTT/iNTT engine is explained in Section 4 and implementation results are presented and discussed in Section 5. Finally, in Section 6, conclusions are drawn.

2 Preliminaries

2.1 Notation

We denote the ring of integers modulo q as Z_q. In this paper, we assume q is a prime number, making Z_q a prime field. We denote as $R_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ the ring of polynomials reduced by $x^n + 1$ with coefficients in $\mathbb{Z}_q$. We assume that n is a power of two. In such ring, any element can by uniquely represented by a polynomial over $\mathbb{Z}_q$ of degree smaller than n. In the algorithms and equations, lowercase letters (e.g., a) are used to represent polynomials, while vectors of polynomials are represented with bold case and matrices with the upper case. The i−th coefficient of a polynomial a is written a_i. The hat tilde is used to represent elements in the NTT domain (e.g., $\hat{a}$).

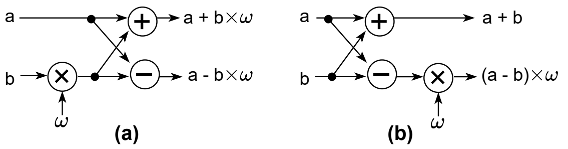

Figure 1: (a) Cooley-Tukey (CT), (b) Gentleman-Sande (GS).

2.2 Number Theoretic Transform

NTT is a generalisation of classical Discrete Fourier Transform (DFT) $\mathbb{Z}_q$ with q a prime satisfying $q \equiv 1 \mod 2n$. We fix ω as a n-th primitive root of unity (also referred to as *twiddle factor*) – i.e. ω is an element of $\mathbb{Z}_q$ s.t. $\omega^n = 1$ and $\omega^i \neq 1$ for any integer i in $[1, n-1]$. We also fix ϕ as a $2n$-th primitive-root of unity s.t. $\phi^2 = \omega$.

The (Negative-Wrapped) NTT is defined as the function from $\mathcal{R}_q$ to $\mathcal{R}_q$ that maps a polynomial a to $\hat{a}$ where, for $i \in [0, n-1]$:

$$\widehat{a_i} = \sum_{j<n} a_j \phi^j \omega^{ij} \tag{1}$$

$$= \sum_{j<n} a_j \phi^{2ij+j} \tag{2}$$

The inverse NTT (written iNTT) transforms an NTT-output back into its original representation:

$$a_i = n^{-1} \sum_{j=0}^{n-1} \widehat{a_j} \phi^{-j} \omega^{-ij} \tag{3}$$

$$= n^{-1} \sum_{j=0}^{n-1} \widehat{a_j} \phi^{-(2ij+j)} \tag{4}$$

The NTT as the following remarkable property: for any element $a, b \in \mathcal{R}_q$, we have that $\widehat{ab} = \hat{a} \circ \hat{b}$, where $\hat{a} \circ \hat{b}$ stands for the polynomial coefficient-wise multiplication. This leads to a direct algorithm for polynomial multiplication whose complexity reduces to the ones of the NTT/iNTT (the convolution having intrinsically a linear complexity).

It is well known that DFT optimization techniques are applicable when implementing the NTT. Among them, the Fast-Fourier Transform (FFT), a divide-and-conquer algorithm proposed independently by Cooley-Tukey [6] and Gentleman-Sande [10], achieves a $O(n \log(n))$ complexity. The radix-2 decimation-in-time (DIT) with bit-reversal is one of the most common variant of the FFT. It takes a polynomial in normal order $(a_0, a_1, \ldots, a_{n-1})$ and generates an output polynomial in bit-reverse order $(\hat{a}_{\mathrm{br}(0)}, \hat{a}_{\mathrm{br}(1)}, \ldots, \hat{a}_{\mathrm{br}(n-1)})$. It generally relies on the Cooley-Tukey (CT) butterfly for the forward NTT, and on the Gentleman-Sande (GS) butterfly unit for the inverse iNTT. The difference in CT and GS butterflies is the order of the operations, the CT butterfly relying on a multiplication preceded an addition/subtraction while the GS butterfly relies on

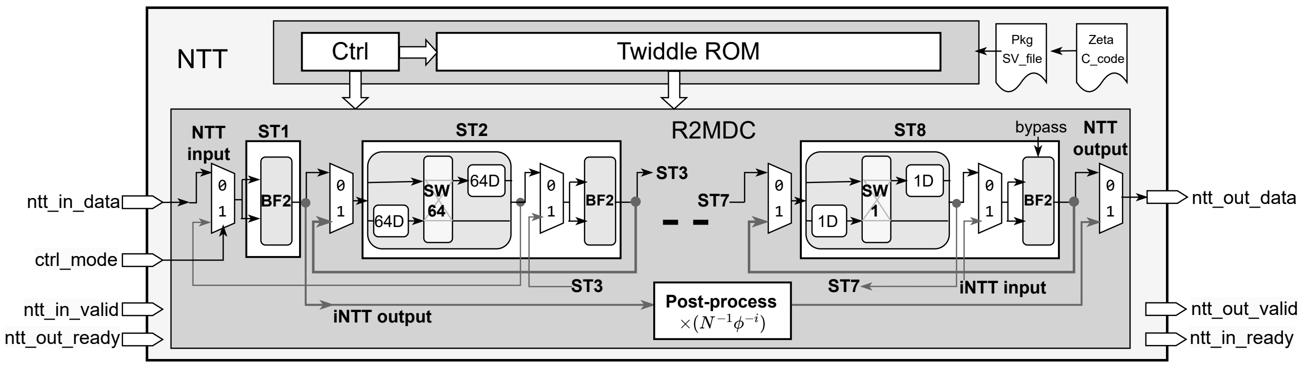

Figure 2: Our Proposed R2MDC NTT/iNTT engine.

an addition/subtraction preceded by a multiplication as shown in Figure 1.

Note that the multiplication by powers of ϕ (resp. inverse power of ϕ) in the NTT (Eq. (1)) (resp. in the iNTT (Eq. (3))) can be seen as pre-processing and post-processing steps over the polynomials before the NTT/iNTT computation is actually done. However, it is possible to combine these within the NTT/iNTT computations by integrating them in the CT/GS butterflies (Eqs. (2) and (4)).

3 Analysis of NTT variants

Since NTT is one of the crucial blocks in LBC, its implementation requires careful design considerations by keeping in mind the performance, independent and continuous execution, and resource requirements. A common strategy to implement NTT is an *iterative* approach that can be done either using a single butterfly unit [9, 22] or multiple butterfly units [2, 5, 12, 14, 21, 23] that perform NTT computations stage by stage. The ML-KEM and ML-DSA require 7 and 8 stages of NTT computations based on n and the modulus q, respectively. The disadvantage of this approach is that separate read and write memories are always needed to store computations for each stage along with complex control logic to manage read/write memory accesses/conflicts. Furthermore, the latency is higher as such a core will wait to finish the NTT computations per stage per vector (if multiple vectors need to be processed) and can not be parallelized with other operations in the ML-KEM and ML-DSA system.

In contrast to the previously mentioned approach, a *pipelined* NTT implementation offers significant advantages by delivering high performance while avoiding complex memory accesses. The Multi-path Delay Commutator (MDC) [19] is a classical approach for processing NTT computations in a pipelined manner, with the level of parallelism determined by the choice of radix. Radix-2 (R2) enables processing 2 coefficients per clock cycle whereas Radix-4 (R4) enables processing of 4 coefficients per clock cycle. Prior work on high-performance MDC based NTT designs have one or more limitations [15, 24]. First, they are either targeted for ML-KEM or ML-DSA, but not for both standards. Note that the NTT engine being proposed for ML-KEM [15] is unable to support ML-DSA because of its in-complete NTT that supports one less necessary pipeline stage. Second, the R2MDC NTT design described in [15] employs two sets of hardware units dedicated to NTT and iNTT operations. It processes even and odd data sets separately rather than utilizing continuous data processing. On the other hand, the NTT design in [24] has a partially pipelined architecture with two stages folded together as well as processes a single coefficient per cycle instead of two. Hence, both available designs come with data-memory dependencies to store and load the intermediate computations during the NTT/iNTT flow and have a higher latency requirement in terms of clock cycles.

All of the state-of-the-art NTT architectures [2, 15, 22, 24] utilize a CT butterfly configuration for the NTT while a GS butterfly configuration is used for the iNTT flow. The NTT hardware designs therefore implement a $unified$ butterfly unit that supports both configurations in a single hardware module but with the overhead of duplicated hardware. In particular, the butterfly units utilize two sets of modular adders and modular subtractors along with a shared modular multiplier. It also comprises of multiplexer circuitry for selection between the NTT and iNTT datapath flow. This architectures therefore result in an unnecessarily increased area requirement and critical path.

4 Architecture of our NTT/iNTT engine

To cater above limitations, this work proposes a high-performance fully pipelined and parallel R2MDC NTT/iNTT accelerator with the following features: (i) Single-set of NTT/iNTT hardware with unique control logic, (ii) single butterfly configuration, i.e., CT for NTT/iNTT that reduced area and improved the critical path, (iii) fully pipelined and parallel architecture to enable continuous consumption of input by NTT engine whereas consumption of output by proceeding construction, without any data memory dependency, and (iv) support both ML-KEM and ML-DSA NTT/iNTT computations, or even either of it.

4.1 Top-Level Design and Interface

The block diagram of our proposed R2MDC NTT/iNTT accelerator is shown in Figure 2. The accelerator takes inputs via the *ntt_in_data* bus. The data width was chosen to be 46 bits comprising of 2 coefficients (2×23 bits) of ML-DSA. ML-KEM uses only the least significant 12 bits of each reserved 23-bit field, the remaining most significant bits are padded with '0'. The output is driven at the *ntt_out_data* bus. For both busses, we implemented a signal

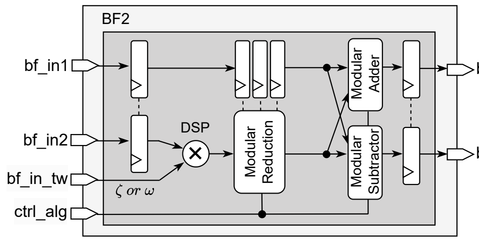

Figure 3: Radix-2 Cooley-Tukey-only Butterfly (BF2) Unit.

synchronization hand-shake via *valid/ready* inputs and outputs, respectively. The *ctrl_mode* input allows switching between Dilithium and Kyber and the type of operation, i.e., NTT or iNTT.

The engine is composed of two primary components: the R2MDC engine and the twiddle ROM unit. Based on $n = 256$, the MDC architecture results in 8 stages of the NTT, which equals to $log_2(n)$. Except for the first stage (ST1) that only consists of a single butterfly unit (BF2), all other stages are comprised of the same BF2 unit and a *commutator* unit. The commutator unit is made up of a Switch (SW) and one or more Delay (D) buffers that synchronize the arrival of data towards the BF2 of the same stage in a required sequence. Based on the NTT flow, the delay buffer has a variable depth starting from 64D for ST1, 32D for ST2,... , to 1D for ST8. Whereas the iNTT flow requires the delay buffers in a sequence of 1D for ST1, 2D for ST2,... , to 64D for ST8. Contrary to [15], we utilize the same NTT flow commutator units for the iNTT by devising a control logic through multiplexers. The logic '0' via *ctrl_mode* signal selects an NTT operation where the data starts flowing from ST1 to ST8, whereas a logic '1' selects the reverse flow starting from ST8 to ST1 to perform an iNTT operation, thereby fully utilizing the hardware by 100 %.

4.2 Design of the NTT Stages

Each of the eight stages of our R2MDC engine requires a separate and identical butterfly unit (BF2). Instead of using a $unified$ butterfly unit [2, 15, 24], a single BF2 configuration, i.e., Cooley-Tukey (CT) as shown in Figure 3 is utilized. This single BF2 unit is then used for both NTT and iNTT operations. The bf_in1 and bf_in2 inputs expect the two 23-bit coefficients, and bf_out1 and bf_out2 drive the two output coefficients accordingly. The $ctrl\_alg$ signal selects the appropriate modulus q and type of modular reduction used after the multiplication operation. Modulo reduction after addition and subtraction was implemented by either adding or subtracting the modulus q, but without opening a timing side channel, i.e., the BF2 unit runs in constant time. The bf_in_tw input provides the twiddle factors from the twiddle ROM. Note that we added input and output registers and also pipelined the modulo reduction unit in order to reduce critical path of our design. The total latency of the BF2 unit is 5 clock cycles.

Due to the single CT BF2 configuration, the pre-process step during the NTT can be merged with the butterfly computations. However, the post-process step during the iNTT cannot be merged and is implemented separately along with the multiplication by n^{-1}. It costs one extra multiplier and a ROM to store the pre-computed values for ML-KEM and ML-DSA. Hence, $\mathrm{NTT}^{CT,\zeta}_{no\rightarrow bo}$ and $\mathrm{iNTT}^{CT,\omega^{-1}}_{bo\rightarrow no}$ configurations are taken up for *forward* and *reverse* computations, respectively that also avoids the bit-reversal requirements. Normally, ζ notation is used for merged pre/post-process steps.

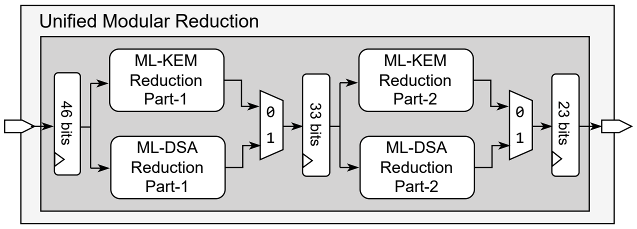

Figure 4: Unified Modular Reduction Unit.

The proposed R2MDC NTT accelerator is implemented in a fully pipelined manner, with all 8 stages unfolded and capable of processing 2 coefficients per cycle in parallel. To synchronize the NTT accelerator with the rest of the system, a signal handshake protocol was implemented that consists of $*\_valid$ and $*\_ready$ signals as shown in Figure 2. For example, the in_valid signal indicate that valid data is driven at the in_data bus. The in_ready signal indicates that the core is ready to consume new data. Data transaction only happens if both $*\_valid$ and $*\_ready$ signals are asserted. In this case, the engine consumes the input, processes it, and increments the pipeline by one cycle. Note that the engine will stall the pipeline if an output does not get consumed or if no valid input is driven at the input interface. Thus, our NTT engine is able to process the data continuously which increases the overall throughput of ML-KEM and ML-DSA.

Based on the parameter sets given in Table 1, ML-KEM results in an incomplete-NTT configuration. This basically would produce a valid output after stage ST7 but the output would not be in a correct order, requiring 1D shifting and switching. In order to avoid the incorrect order, we decided to pass the output of ST7 through stage ST8, utilizing the internal Commutator unit and bypassing the BF2 unit as shown in Figure 2. This rearranges the output into the correct sequence for continuous data processing. Similarly, during the iNTT, the first stage ST8 BF2 is bypassed accordingly. With this change, it is not necessary that the input vectors need to be split into *even* or *odd* coefficients (e.g., as reported in [15] in case of ML-KEM). In addition, it is not required to have multiple read and write operations to load/store the data from/to memory, which further increases the performance and throughput of our NTT engine.

4.3 Unified Modular Reduction Unit

The modular reduction unit reduces the coefficients from the range $[0, (q-1)^2]$ to which they belong after a multiplication to the range $[0, q-1]$ in congruence with the parameters of the scheme. For both ML-KEM and ML-DSA modular reduction units, we decided to split up the datapath into two in order to reduce the critical path. The reduction operations therefore take 2 clock cycles and are fully pipelined. Figure 4 shows the unified architecture.

The ML-KEM modular reduction is a modified version of the algorithm presented by Xing and Li [21] which in turn is a modification of the Barrett reduction. In the first clock cycle, the quotient is approximated by substituting multiplication and shift with shifts and additions. In the second clock cycle, the result is computed following the principle of Barrett reduction. In contrast to Xing and Li [21], we compute the quotient with one extra bit of precision, which ensures that the intermediate results lie in the range $[-2q_k, 2q_k)$ and allows us to simplify the output logic down to just one conditional addition and one conditional subtraction to finally correct the result.

The ML-DSA modular reduction is based on the notion that the prime modulus is of a special form, i.e., $q_d = 2^{23} - 2^{13} + 1$ (pseudo-Mersenne prime) and for any x, $x \cdot 2^{23} \equiv x(2^{13} - 1) \pmod{q_d}$. We therefore decided to apply the idea of Solinas [20] to efficiently reduce the coefficients by simple shift and addition/subtraction operations. The first part of the modular reduction reduces the original value from 46 bits to 33 bits. The second part reduces the result further from 33 bits to 23 bits. A final conditional subtraction corrects the result and brings the coefficients back to the range $[0, q-1]$.

4.4 Twiddle ROM

Each BF2 unit requires computation with a pre-calculated twiddle factor, stored in a memory (e.g., ROM or BRAM). That means that in total 8 memories are needed as each BF2 unit accesses twiddle factors during each cycle of an NTT or iNTT operation in ML-KEM and ML-DSA. We decided to implement a dedicated Twiddle ROM unit which manages the access to these 8 distinct memories as shown in Figure 2. The pre-computed values are generated using a C model and are stored in a SystemVerilog *package.sv* file. For an FPGA target, for example, the BRAMs get initialized directly using that file. This allows for an easy and flexible replacement with new or updated twiddle factor values if needed.

5 Implementation Results

The proposed NTT accelerator was implemented in SystemVerilog. We used Synopsys Spyglass linting for efficient verification and optimization of our design. To evaluate the performance, we synthesized the design on a Zynq UltraScale+ MPSoC device, i.e., 'xck26-sfvc784-2LV-c'. We decided to not allow the compiler to flatten the hierarchy in order to get exact area requirements of all implemented submodules. For maximum frequency analysis, we swept across various target frequencies starting at 50 MHz and increased the frequency until occurrence of the first timing violation after implementation, i.e., Place and Route (PnR).

Table 2 shows the area and performance figures. At a maximum frequency of 322 MHz, our R2MDC NTT/iNTT core capable of processing ML-DSA and ML-KEM utilizes 3,821 LUTs, 2,970 FFs, and 16 DSPs. The twiddle ROM consumes 4 36Kb dual-port BRAMs (or 8 18Kb single-port BRAMs) for storing the pre-computed twiddle factors. The post-process unit consumes 4 DSPs and 1 BRAM which is needed to store the values of $\phi^{-i}n^{-1}, i \in [0, n-1]$. In total, 20 DSPs and 5 BRAMs are needed.

Processing a 256-coefficient polynomial vector with our NTT accelerator takes a total of 258 clock cycles: a 130 clock-cycle latency to fill the pipeline and 128 cycles to generate the entire output. Note that once the pipeline is full, processing subsequent vectors takes only 128 ($n/2$) clock cycles each.

Table 2: Area and performance results.

Modules	Area LUTs/FFs/DSPs/BRAMs	Latency [Cycles]	Frequency [MHz]
NTT engine	3,821/2,970/16/–		
Twiddle ROM	–/–/–/4	258	322
Post-process	–/–/4/1		

5.1 Comparison With Related Work

We now compare our NTT design with related work in Table 3. We list works that offer Kyber-only [23], Dilithium-only [24], or unified NTT computations [2, 14]. We differentiate between *iterative* [2, 14, 23] and *pipelined* [24] NTT architectures. For each proposed solution, we indicate the number of internal BF2 butterfly units to ensure a fair comparison. The table also lists the number of cycles required to process one polynomial vector as well as up to 8 vectors (the maximum number for ML-DSA) to highlight the advantage of pipelined architectures. The 'Performance' column converts the latency for processing 8 vectors into microseconds (μs) and provides the design's throughput in Mbps. Additionally, we compute the Area-Time-Product (ATP) for each design, considering the LUT count as the most limiting resource on an FPGA and multiplying it by the latency to process 8 polynomial vectors.

The results indicate that our proposed pipelined NTT accelerator not only delivers superior performance—processing 8 vectors in 3.60 μs with a throughput of 569 Mpbs—but also has the lowest ATP, the same applies to ML-KEM. All the listed designs, both *iterative* and *pipelined*, require either FIFOs [15] or additional data memories to manage input/output coefficients in a correct sequence [2, 14, 23, 24]. It is important to note that these memory requirements have not been included in the reported figures (as marked with footnote *b*).

Compared to our flexible NTT accelerator, most available NTT architectures are designed for fixed parameter sets [23, 24]. Only a few, such as those in [2, 14], can handle both ML-KEM and ML-DSA by integrating two ML-KEM butterfly operations into the single butterfly unit required for ML-DSA. Aikata et al. [2] utilized 2 BF2 units for ML-DSA, while Mandal and Roy [14] offer two versions: one with 2 BF2 units and another with 4 BF2 units. In comparison, our design achieves better ATP values while using hardware resources comparable to their iterative NTT architectures.

There are few pipelined NTT architectures available in the open literature [15, 24]. The architecture proposed in [24] designed a folded R2MDC NTT for ML-DSA, utilizing half the hardware resources of our design but resulting in higher latency. Their design processed 8 vectors in 13.61 μs, whereas our NTT accelerator processes them in 3.6 μs, providing a 74% improvement in performance. Additionally, our fully pipelined architecture, which requires more FFs, achieves a higher operating frequency. Despite the different platforms used (Artix-7 vs. Zynq Ultrascale+), our design reaches a maximum frequency of 322 MHz compared to the 97 MHz reported in [24].

The closely related R2MDC NTT design, S-NTT [15] specifically targets ML-KEM and exhibits the same latency for processing 8 vectors. However, the specific hardware resource usage for NTT is not separately provided, and thus is not listed in Table 3. Their design incorporates seven stages based on ML-KEM, with the ST2

Table 3: Performance comparison of *iterative* and *pipelined* (MDC-based) NTT methods.

Design	FPGA	Algos	BF2	Area	Freq.	Latency (CC)		Performance		Area-Time
			units	LUTs/FFs/DSPs/BRAMs	MHz	1-vec	8-vec	μ s	Mbps	Prod. [ATP]
[23]^a	Artix-7	ML-KEM	4	2,543/792/4/4.5	182	232	1,856	10.20	201	25,938
[24]^b	Artix-7	ML-DSA	4	1,919/1,301/8/2	97	424	1,320	13.61	150	26,136
[2]^{a,b}	Zynq UltraScale+	ML-DSA	2	3,487/1,918/4/1	270	512	4,096	15.17	135	52,898
		ML-KEM	4			224	1,792	6.64	309	23,154
[14]^a	Zynq UltraScale+	ML-DSA	2	2,893/2,356/4/4.5	342	512	4,096	11.98	171	34,658
		ML-KEM	4			224	1,792	5.24	391	15,159
		ML-DSA	4	5,909/3,376/8/5.5	294	256	2048	6.97	294	41,185
		ML-KEM	8			112	896	3.05	672	18,022
Our Work	**Zynq UltraScale+**	**ML-DSA**	**8**	**3,821/2,970/20/5**	**322**	**258**	**1,158**	**3.60**	**569**	**13,755**
		ML-KEM	**8**			**254**	**1,154**	**3.58**	**571**	**13,679**

^{a}Iterative NTT.
^{b}Extra data RAM memory (BRAM) is needed to store intermediate coefficients, which is not included in this table.

commutator starting with a 32D delay instead of 64D, processing a 256-coefficient vector as two 128-coefficient vectors. This makes the design inflexible for other parameters. While flexibility often entails increased hardware resources or performance degradation, our proposed R2MDC NTT accelerator strikes a balance by optimizing both hardware utilization and performance.

6 Conclusion

In this work, we present a high-performance NTT architecture to support ML-DSA and ML-KEM. As a novel contribution, we propose to use a *Cooley-Tukey-only* butterfly configuration in the NTT design to reduce area requirements and critical path. We integrate the new construction in a parallel NTT architecture based on a Multi-path Delay Commutator (MDC) pipeline to obtain best results. We evaluated our design on a Zynq Ultrascale+ MPSoC device. Our results show that it outperforms existing work in terms of data throughput and Area-Time-Product (ATP) while requiring less resources compared to similar proposals. Our results also show that our MDC-based NTT design has similar LUT requirements than work based on iterative NTTs, but is significantly faster, i.e., up to a factor of 4 in case of Dilithium NTT calculation performance, due to the pipelined processing of data. Our solution is therefore best suitable for high-performance applications of PQC.

References

[1] Shweta Agrawal, Dan Boneh, and Xavier Boyen. 2010. Efficient Lattice (H)IBE in the Standard Model. In *Advances in Cryptology – EUROCRYPT 2010*, Henri Gilbert (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 553–572.

[2] Aikata Aikata, Ahmet Can Mert, Malik Imran, Samuel Pagliarini, and Sujoy Sinha Roy. 2022. KaLi: A Crystal for Post-Quantum Security using Kyber and Dilithium. Cryptology ePrint Archive, Paper 2022/1086.

[3] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2021. CRYSTALS-Kyber algorithm specifications and supporting documentation.

[4] Shi Bai, Lèo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlè. 2021. CRYSTALS-Dilithium algorithm specifications and supporting documentation (Version 3.1).

[5] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. High-Speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography. Cryptology ePrint Archive, Paper 2021/563.

[6] James Cooley and John Tukey. 1965. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.* 19, 90 (1965), 297–301.

[7] FIPS-203 (Draft). 2023. Module-Lattice-based Key-Encapsulation Mechanism Standard. National Institute of Standards and Technology (NIST).

[8] FIPS-204 (Draft). 2023. Module-Lattice-based Digital Signature Standard. National Institute of Standards and Technology (NIST).

[9] Tim Fritzmann and Johanna Sepúlveda. 2019. Efficient and Flexible Low-Power NTT for Lattice-Based Cryptography. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, McLean, VA, USA, 141–150.

[10] W. Morven Gentleman and G. Sande. 1966. Fast Fourier Transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference* (San Francisco, California) *(AFIPS '66 (Fall))*. Association for Computing Machinery, New York, NY, USA, 563–578.

[11] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing* (Bethesda, MD, USA). Association for Computing Machinery, New York, NY, USA, 169–178.

[12] Naina Gupta, Arpan Jati, Anupam Chattopadhyay, and Gautam Jha. 2022. Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium. Cryptology ePrint Archive, Paper 2022/496.

[13] Florian Hirner, Ahmet Can Mert, and Sujoy Sinha Roy. 2024. Proteus: A Pipelined NTT Architecture Generator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* Early Access, 1 (2024), 1–11.

[14] Suraj Mandal and Debapriya Basu Roy. 2024. KiD: A Hardware Design Framework Targeting Unified NTT Multiplication for CRYSTALS-Kyber and CRYSTALS-Dilithium on FPGA. In *37th International Conference on VLSI Design, VLSID 2024, January 6-10, 2024*. IEEE, Kolkata, India, 455–460.

[15] Ziying Ni, Ayesha Khalid, Dur e Shahwar Kundi, Maire ONeill, and Weiqiang Liu. 2023. HPKA: A High-Performance CRYSTALS-Kyber Accelerator Exploring Efficient Pipelining. *IEEE Trans. Comput.* 72, 12 (dec 2023), 3340–3353.

[16] NIST. 2016. Announcing Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms. https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms

[17] NIST. 2022. NIST IR 8413-upd1: Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/publications/detail/nistir/8413/final

[18] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1484–1509. http://dx.doi.org/10.1137/S0097539795293172

[19] Shousheng He and M. Torkelson. 1998. Designing pipeline FFT processor for OFDM (de)modulation. In *Proc. URSI International Symposium on Signals, Systems, and Electronics*. IEEE, Pisa, Italy, 257–262.

[20] Jerome A. Solinas. 1999. Generalized Mersenne Numbers. Technical report, University of Waterloo, 1999. https://cacr.uwaterloo.ca/techreports/1999/corr99-39.pdf

[21] Yufei Xing and Shuguo Li. 2021. A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 2 (Feb. 2021), 328–356.

[22] Ali Yahya Hummdi, Amer Aljaedi, Zaid Bassfar, Sajjad Shaukat Jamal, Mohammad Mazyad Hazzazi, and Mujeeb Ur Rehman. 2024. Unif-NTT: A Unified Hardware Design of Forward and Inverse NTT for PQC Algorithms. *IEEE Access* 12 (2024), 94793–94804. https://doi.org/10.1109/ACCESS.2024.3425813

[23] Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. 2021. A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Grenoble, France, 1020–1025.

[24] Cankun Zhao, Neng Zhang, Hanning Wang, Bohan Yang, Wenping Zhu, Zhengdong Li, Min Zhu, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. A Compact and High-Performance Hardware Architecture for CRYSTALS-Dilithium. *IACR TCHES* 2022, 1 (Nov. 2021), 270–295.