IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 2 June 2024, accepted 18 June 2024, date of publication 10 July 2024, date of current version 18 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3425813

RESEARCH ARTICLE

Unif-NTT: A Unified Hardware Design of Forward and Inverse NTT for PQC Algorithms

ALI YAHYA HUMMDI[1], AMER ALJAEDI[2], ZAID BASSFAR[3], SAJJAD SHAUKAT JAMAL[1], MOHAMMAD MAZYAD HAZZAZI[1], AND MUJEEB UR REHMAN[4]

[1]Department of Mathematics, College of Science, King Khalid University, Abha 61413, Saudi Arabia
[2]College of Computing and Information Technology, University of Tabuk, Tabuk 71491, Saudi Arabia
[3]Department of Information Technology, University of Tabuk, Tabuk 71491, Saudi Arabia
[4]Cyber Technology Institute, School of Computer Science and Informatics, De Montfort University, LE1 9BH Leicester, U.K.

Corresponding author: Mujeeb Ur Rehman (mujeeb.rehman@dmu.ac.uk)

This work was supported by the Deanship of Research and Graduate Studies at King Khalid University through the Large Research Project under Grant R.G.P. 2/5/44.

ABSTRACT Polynomial multiplications based on the number theoretic transform (NTT) are critical in lattice-based post-quantum cryptography algorithms. Therefore, this paper presents a platform-agnostic unified hardware accelerator design (Unif-NTT) to compute the forward and inverse operations of the NTT for the CRYSTALS-Kyber algorithm. Moreover, a unified design (Unif-BU) of the Cooley-Tukey and Gentleman-Sande butterflies is presented using two adders, multipliers, subtractors, routing multiplexers and barret-based modular reduction units. Finally, a dedicated controller is implemented for efficient control functionalities. The implementation results are realized on field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) platforms. The Unif-NTT requires 1664 and 1792 clock cycles for one forward and inverse NTT computations, respectively. It can operate up to a maximum frequency of $212MHz$ and $2.5GHz$ over Virtex-7 FPGA and 28nm ASIC platforms, respectively. The Unif-NTT is 26% more efficient in Area-Time-Product compared to the most area-optimized NTT accelerator from the state-of-the-art. The Unif-NTT design is suited for applications that demand reasonable hardware resources with processing speed.

INDEX TERMS Number theoretic transform, hardware, accelerator, post-quantum cryptography, FPGA, ASIC.

I. INTRODUCTION

The security strength of public-key cryptography algorithms, such as RSA [1] and Elliptic Curve Cryptography (ECC) [2], can be broken using Shor's algorithm [3] on a quantum computer. The leading research and development labs by Google and IBM have (already) developed quantum computers [4], [5]. Google's Sycamore quantum computer, with 53 qubits, can break the security strength of RSA and ECC in 200 seconds, which on classical computers takes 10,000 years [4]. Moreover, IBM has developed a 100-qubit Eagle chip [6]. Due to these remarkable developments in quantum technology, designers and researchers are constructing reliable and secure algorithms/protocols to protect current and future communications against attacks by quantum computers.

The National Institute of Standards and Technology (NIST) started a new contest in 2017 to define the post-quantum cryptography (PQC) standards. After several rounds, the NIST has standardized four PQC algorithms in 2023. These include CRYSTALS-Kyber [7], CRYSTALS-Dilithium [8], SPHINCS+ [9] and Falcon [10]. These new standards require Number Theoretic Transform (NTT)-based polynomial multiplications. Moreover, most of the algorithms in the ongoing (round-1) NIST contest to standardize additional digital signature algorithms [11] require NTT-based polynomial multiplications. Rather than the PQC algorithms, several other applications demand NTT-based polynomial multiplications, such as error correction codes [12], [13], digital signal processors [14], [15], and fully homomorphic

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

encryption (FHE) schemes [16], [17]. Therefore, there is a critical need for the NTT-based polynomial multiplications.

The NTT-based polynomial multiplication involves forward and inverse operations to compute. The forward NTT can be implemented using the Cooley-Tukey butterfly unit (CTBU), while the inverse NTT can be executed using the Gentleman-Sande butterfly unit (GSBU) [17], [18], [19]. Depending on the application requirement, these butterfly units can be implemented on software and hardware platforms. Software implementations, such as microcontrollers, offer higher flexibility but limited throughput. On the other hand, higher throughput or processing speed can be achieved on hardware platforms such as field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). The ASIC-accelerated NTT implementations reduce computation time, increase operating frequency, and maximize overall throughput with high fabrication cost overhead [20], [21]. Comparatively, FPGA-based NTT implementations offer advantages like higher throughput than software-based implementations, low implementation cost, and ease of access in the market compared to ASICs [22]. Therefore, we target our implementations on FPGA and ASIC platforms to achieve high operating frequency and lower computation costs.

A. EXISTING NTT-HARDWARE ACCELERATORS

We provide a summary of the contributions of the state-of-the-art hardware accelerators of the NTT in Table 1. Also, we describe the corresponding architectural details of the existing NTT accelerators below.

The reference design of [23] uses a doubled bandwidth ping-pong memory access scheme to implement the NTT operations (forward and inverse). It can compute one forward and inverse operation in 490 clock cycles. Moreover, the design can operate on a maximum circuit frequency of $257MHz$ (on Artix-7 FPGA), and it requires $1.9\mu s$ for one forward and inverse NTT execution. Another NTT accelerator on Artix-7 device is realized in [24]. This design computes one forward and inverse NTT operation in 1024 cycles, operates on a maximum frequency of $136MHz$, and requires $7.5\mu s$ for computations. A high-speed NTT accelerator is described in [25], where the authors utilize 4 and a maximum of 16 butterfly units to reduce the computation time of the forward and inverse NTT operations.

An interesting reconfigurable and high-speed NTT accelerator is reported in [26]. The reconfigurability is achieved using a one-bit *mode* signal in the reference design to choose between forward and inverse NTT operations. Moreover, a 2×2 butterfly core is implemented in the reference design to merge two NTT layers to perform two butterfly operations (in each layer). This merging layer strategy benefits in reducing the clock cycles. In addition, the authors of [26] use four instances of a 2×2 butterfly unit to reduce further the clock cycles. Also, pipelined registers are placed in the data path of the butterfly unit to optimize the critical path. Therefore, these strategies allow the circuit to operate on a $222MHz$ and require 324 clock cycles for computation on an Artix-7.

In [27], three register banks are utilized to compute (only) the forward NTT operation. One of the register banks is dedicated to twidle factors, while the remaining two are reused iteratively to keep the intermediate and the final results of the forward NTT computation. A memory-efficient high-speed implementation of NTT on an ARM Cortex-M4 microcontroller is presented in [19], where the design requires 7725 and 9347 clock cycles for forward and inverse NTT computations, respectively.

A low-complexity and high-speed NTT hardware accelerator is implemented in [29], where authors proposed a Montgomery-based reduction module with a pipeline structure to realize the modular multiplication. Moreover, a precomputed scheme based on the existing hardware resources is presented to minimize the hardware resources. To improve the throughput of the NTT architecture, the reference design has implemented a block storage technique to read and write data simultaneously.

In [32], an FPGA implementation of NTT accelerator for homomorphic SEAL-Embedded library is presented, where the dual-port FPGA memories are utilized. In contrast, we used three RegFiles as memory elements. We want to highlight that the reference design is FPGA-specific, while the NTT accelerator presented in this work is not technology-dependent. This will be discussed further in the subsequent sections of this paper.

The work described in [17] implemented a flexible and parameterized hardware architecture to compute the NTT operations (i.e., forward and inverse). Also, the reference architecture supports parameters of the various lattice-based cryptographic algorithms. Similar to [17], the reference design of [28] describes an FPGA implementation of a run-time configurable and highly parallelized NTT-based polynomial multiplication architecture that supports six different parameter sets, which are used in lattice-based homomorphic encryption and/or post-quantum cryptosystems.

Recently, in 2024, an area-efficient, conflict-free, and configurable architecture for the acceleration of forward and inverse NTT operations is presented in [31]. The reference architecture supports polynomials of different degrees. To minimize latency, the CTBU and GSBU operations are merged to eliminate the need for bit-reverse operations in the NTT algorithm. A dedicated post-quantum arithmetic logic unit is designed in [30], which is embedded directly in the pipeline of a CVA6 RISC-V processor to accelerate the performance of the CRYSTALS-Kyber and CRYSTALS-Dilithium PQC algorithms. Instead of the accelerators of [30] and [31], some more interesting NTT designs are recently published in [33], [34], [35], [36], and [37]. Specifically, through several experiments over CRYSTALS-Kyber as a case study on FPGA and ASIC platforms, the work in [37] highlighted the importance of dedicated and unified hardware architectures of CTBU and GSBU butterfly units for NTT realizations.

TABLE 1. Summary of the contributions of the state-of-the-art hardware accelerators of the NTT designs. The ✔ indicates the characteristics/features of the implemented NTT accelerators. The – shows where the reference design does not consider the corresponding feature. The ✔ indicates that the designs can be extended in future to provide support for the corresponding features. The TW is our implemented NTT accelerator.

Ref	NTT and INTT Design(s) for Computations			CTBU and GSBU Designs		Employed Memory Units		Platforms	
	Dedicated	Run-Time Configurable	Flexible	Unified	Individual	BRAMs / SRAMs	RegBanks	FPGA	ASIC
[23] †	✔	–	–	✔	–	✔	–	✔	–
[24] †	✔	–	–	✔	–	✔	–	✔	–
[25] †	✔	–	–	✔	–	✔	–	✔	–
[26] †	✔	–	–	✔	–	✔	–	✔	–
[27] †	✔	–	–	–	✔	–	✔	✔	✔
[19] †	✔	–	–	✔	–	✔	–	✔	–
[17] ‡	–	✔	✔	✔	–	✔	–	✔	–
[28] ‡	–	✔	✔	✔	–	✔	–	✔	–
[29] ‡	✔	–	–	✔	–	✔	–	✔	–
[30] †	✔	–	✔	✔	–	✔	–	✔	–
[31] †	✔	–	✔	✔	–	✔	–	✔	–
TW †	✔	–	✔	✔	–	–	✔	✔	✔

† Supports $n = 256$ & $q = 3329$ parameters of CRYSTALS-Kyber (where n & q represent the polynomial degree and prime for modular reduction).
‡ Uses different values of n and q regardless of the specific CRYSTALS-Kyber parameters.

B. LIMITATIONS OF THE EXISTING NTT DESIGNS

The NTT designs of [19], [23], [24], [25], [26], [27], [30], and [31] are specific to FPGA platform as these frequently uses the BRAM-based memory units and DSP macros for implementations. Therefore, these accelerators (in their present form) can not be realized on the ASIC platform. Hence, the platform-agnostic architectures are required to investigate the performance of the same NTT design on both FPGA and ASIC platforms.

Even if the design described in [27] is platform-agnostic, this implements only the forward NTT operation without the focus on the inverse NTT. Note that it is essential to consider (both) the forward and inverse NTT operations in a single design as the PQC algorithms, such as the CRYSTALS suite [7], [8], require several times the computation of the forward and inverse NTT operations during the implementations.

The unified designs of CTBU and GSBU (described in [17], [19], [23], [24], and [28]) have been frequently implemented using a single adder, multiplier and subtractor units to accelerate the forward and inverse NTT operations. Indeed, the use of single adder, multiplier and subtractor units reduces the hardware resources but with performance (in operating frequency) overhead. Let us look at how the performance will be affected. To implement the CTBU, multiplication must be computed before addition and subtraction operations. In the case of GSBU, the multiplication operation must be computed after the addition and subtraction operations. Therefore, the use of one adder, one subtractor and one multiplier requires routing multiplexers with some feedback inputs to generate the desired output. The feedback inputs to the routing multiplexers affect the overall circuit frequency of the design. Therefore, to improve the performance of the circuit (in terms of operating frequency), pipelining is extensively employed [24], [29], [38]. Moreover, to further improve the performance in terms of clock cycles, the parallel use of several butterfly units in a single design has been used in state-of-the-art accelerators. More precisely, the design of [26] uses 4 butterfly units, while 16 butterfly units are utilized in [25]. Note that these architectures optimize performance in computation time with larger hardware resource utilization. Therefore, the optimal solution must be explored with balanced hardware area and performance implementations.

The power consumption of the NTT design has been (mostly) overlooked in the existing NTT accelerators. Thus, it also needs to be considered during the NTT implementations.

C. NOVELTY

To discuss the novelty, we first provide the architectural differences of our work regarding the state-of-the-art NTT accelerators below.

- Table 1 shows that the existing NTT accelerators utilize the unified design of the CTBU and GSBU to compute the forward and inverse operations. We also unify the operations of the CTBU and GSBU butterfly units in a single design. Note that the designs of [19], [23], [24], [27], [28], and [29] use single arithmetic operators (such as adder, multiplier and subtractor) in their unified butterfly units. As reported in section I-B, the use of a single adder, multiplier and subtractor in a unified design of the CTBU and GSBU affects the circuit frequency. Therefore, our NTT accelerator significantly differs from the NTT designs of Table 1 as we have used two adders, two multipliers, two subtractors, two routing multiplexers and two modular reduction blocks in the unified design of the butterfly unit. This choice allows us to obtain a higher operating frequency. Moreover, it facilitates the parallel computation of the forward and inverse operations (if required in the future).
- The NTT architectures of [25] and [26] incorporate 4 and 16 unified butterfly units, respectively, while in our design, we have used only one unified butterfly unit.

- To accommodate the initial, intermediate and final results, the state-of-the-art NTT designs use FPGA-based BRAMs. We have used register banks (RegBanks) to obtain a platform-agnostic accelerator architecture. The NTT design of [27] is platform-agnostic. It has used three RegBanks. In our design, we also used three RegBanks but the architecture of [27] is dedicated to the computation of only the forward NTT operation without concerning the inverse NTT. In contrast, our NTT architecture computes forward and inverse NTT operations as the CRYSTALS suite requires forward and inverse NTT computation several times during the implementations [7], [8].

To our knowledge, an NTT accelerator using three RegBanks (as memory elements) and two adders, multipliers, subtractors, multiplexers, and modular reductions (in a unified design of the butterfly unit) has never been explored before on both FPGA and ASIC platforms. This is the novelty.

D. CONTRIBUTIONS

To address the limitations highlighted in section I-B, our contributions are as follows:

- We have presented a platform-agnostic unified NTT (Unif-NTT) hardware accelerator architecture using parameters of $n = 256$ and $q = 3329$ of the CRYSTALS-Kyber PQC algorithm to facilitate polynomial multiplications. A platform-agnostic architecture is obtained using RegBanks, which are used in the Unif-NTT design as memory elements to read/write initial, intermediate, and final results (see section III).
- To implement the forward and inverse NTT operations, we have proposed a unified butterfly unit (Unif-BU) design of the Cooley-Tukey and Gentleman-Sande butterflies using two adders, two multipliers and two subtractors. Moreover, we shared two routing multiplexers and two barret modular reduction units across the arithmetic operators to generate the desired outputs.
- A dedicated finite-state-machine (FSM) controller is implemented to provide control functionalities to implement the Unif-NTT architecture (see section III-C).
- To present the performance evaluation of the Unif-NTT design on both FPGA and ASIC platforms, we have defined a metric in Area-Time-product (ATP). Additionally, a comprehensive comparison in area and timing to existing NTT accelerators is presented (see section IV).

E. SIGNIFICANCE

The proposed Unif-NTT architecture is implemented in a Verilog Hardware Description Language (HDL) using Vivado 2018.1. We provide the implementation results on Xilinx Virtex-6 and Virtex-7 FPGA devices. Also, we have implemented the Unif-NTT design on a 28nm ASIC process technology. It utilizes 1409 and 1287 slices on the Virtex-6 and Virtex-7 FPGA devices, respectively. For one forward and inverse NTT computation, Unif-NTT requires 1664 and 1792 clock cycles, respectively. Moreover, it can operate up to a maximum frequency of $181MHz$, $212MHz$ and $2.5GHz$ over Virtex-6, Virtex-7 and 28nm ASIC platforms. The Unif-NTT accelerator is 26% more efficient in terms of ATP than the most optimized area-efficient NTT design from the literature. Therefore, the implemented results and comparison to existing state-of-the-art accelerators reveal that the Unif-NTT architecture is suitable for applications that demand reasonable hardware area with processing speed.

The rest of this article is organized as follows: Section II introduces NTT-related mathematical background. The NTT accelerator architectures are discussed in Section III. We describe the implementation results and comparison to existing NTT-related architectures in Section IV. Finally, Section V concludes the article.

II. BACKGROUND RELATED TO NTT-BASED POLYNOMIAL MULTIPLICATION

There are several multiplication methods in the literature, including but not limited to schoolbook and Karatsuba, to multiply the $(n-1)$-degree polynomials [39], [40], [41], [42], [43]. However, the computational complexity of schoolbook and Karatsuba multipliers is $O(n^2)$ and $O(n^{1.59})$ [18], respectively. The NIST-defined PQC standards, such as CRYSTALS-Kyber and CRYSTALS-Dilithium, utilize NTT-based polynomial multiplications due to their reduced time complexity of $O(n \log n)$.

The computational complexity of NTT-based polynomial multiplications depends on the computation of forward and inverse operations. The forward NTT transforms an $(n-1)$-degree polynomial (in coefficient representation) into n polynomials of degree 0 (in value representation) [18], [40], [42]. Subsequently, the polynomials in their value-representation form (NTT domain) can be multiplied coefficient-wise to obtain the multiplied values in the NTT domain. To revert to the polynomial's coefficient representation, an inverse NTT transform is necessary. The conversion to and from the NTT domain has a time complexity of $O(n \log n)$. Coefficient-wise multiplication, on the other hand, has a time complexity of $O(n)$. Consequently, the total time complexity of the process is $O(n \log n)$ [18]. The forward NTT transformation of an n-degree polynomial $f = \sum_{i=0}^{n-1} f_i x^i$ is defined using Eq. 1.

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

In Eq. 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. Similarly, the inverse NTT can be computed using Eq. 2.

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

In Eq. 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. It is essential to note that various algorithms exist in the literature to implement Eq. 1 and Eq. 2 for forward and inverse NTT transformations. The traditional

Algorithm 1 Iterative NTT Algorithm [18]

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**

iterative NTT algorithm is given in Algorithm 1 to implement these two equations.

In Algorithm 1, lines 6 to 11 are important as these lines relate to the butterfly unit. In the present form of Algorithm 1, lines 6 to 11 show the operations of the CTBU, as illustrated in Eq. 3. In other words, we can say that the operations of Eq. 3 are repeated in lines 6 to 11 of Algorithm 1 to compute the forward NTT operation. Similarly, we provide the GSBU operations in Eq. 4. Note that the operations of Eq. 4 must be replaced in lines 6 to 11 in Algorithm 1 to compute the complete inverse NTT operation. Instead of these butterfly operations in Algorithm 1, the operations in the remaining lines either denote the initialization or memory address generation for read/write operations.

$$\text{CTBU} = (a + bw)\ mod\ q\ \ \&\ \ (a - bw)\ mod\ q \quad (3)$$

$$text GSBU = (a + b)\ mod\ q\ \ \&\ \ w(a - b)\ mod\ q \quad (4)$$

III. UNIF-NTT: PROPOSED HARDWARE ACCELERATOR

We provide our proposed Unif-NTT accelerator architecture in Figure 1. It comprises (i) three RegBanks, (ii) one Unif-BU and (iii) a dedicated control unit (CU). The RegBanks store the initial, intermediate and final results for computation of the forward and inverse NTT operations. The Unif-BU implements the corresponding arithmetic operators for forward and inverse operations. The CU generates the related control signals for the Unif-BU and RegBanks. The corresponding insight details are given in sections III-A, III-B and III-C.

A. REGBANKS

The grey part in Figure 1 denotes the RegBanks, i.e., RegBank1, RegBank2 and RegBank3. These RegBanks are memory units that keep initial, intermediate and final results. The reason for the use of RegBanks is to obtain a platform-agnostic accelerator architecture. Moreover, RegBanks allow designers to obtain an (overall) higher circuit frequency. It is possible to use BRAMs/SRAMs for memory elements specific to FPGAs/ASICs, but these have a specific read/write access time when loading/storing data. This access time affects the performance of the entire NTT design. An alternate approach is to use RegBanks instead of BRAMs/SRAMs (depending on the platform used), but RegBanks are more expensive in terms of hardware resources. There is always a trade-off.

The orange, white and green blocks in Figure 1 present the RegBank1, RegBank2 and RegBank3. As we mentioned, we selected parameters of a CRYSTALS-Kyber PQC algorithm, which features 256 input polynomial coefficients, each with a size of 12 bits. Therefore, every RegBank comprises 256 registers capable of accommodating 12 bits on each address. Consequently, the overall size of each RegBank is 256×12.

The RegBank1 stores the 256 input polynomial coefficients. For data loading on RegBank1, a PC signal is used (shown in Figure 1). It takes 256 clock cycles. Similarly, RegBank3 holds the precomputed 256 twiddle factors (in bit-reversed order). A TF signal loads the twiddle factors on RegBank3, shown in Figure 1. Also, this needs 256 clock cycles to load data. The precomputed twiddle factors are g and w in Algorithm 1, where g contains all the 256 twiddle factor values and w contains the corresponding 12-bit twiddle factor for the subsequent round (or *for* loop value in Algorithm 1). It is important to mention that the twiddle factors are essential for respective forward or inverse NTT operations. Moreover, twiddle factors, representing complex exponential coefficients, play a crucial role in manipulating and converting data between the time domain and the NTT domain [17].

After loading the initial polynomial coefficients and the corresponding twiddle factors into the RegBanks, the CU generates the control signals for reading polynomial coefficients and the twiddle factors from the RegBanks for further computations. Note that the bidirectional signals, i.e., $pc1$ and $pc2$ in Figure 1, denote the reading and writing from/to RegBank1 and RegBank2 (we highlighted these RegBanks with the red colour box in Figure 1). On the other hand, RegBank3 (highlighted with the green portion in Figure 1) communicates with the control unit using a unidirectional signal of tf to provide the corresponding twiddle factor for forward and inverse NTT computations.

B. PROPOSED UNIF-BU ACCELERATOR DESIGN

The Unif-BU is the core processing component in Unif-NTT and is responsible for the implementation of arithmetic operations of Eq. 3 and Eq. 4. Again, we want to remind the reader that Eq. 3 is for forward NTT computations, while Eq. 4 is to compute the inverse NTT. Therefore, our proposed Unif-BU architecture to implement Eq. 3 and Eq. 4

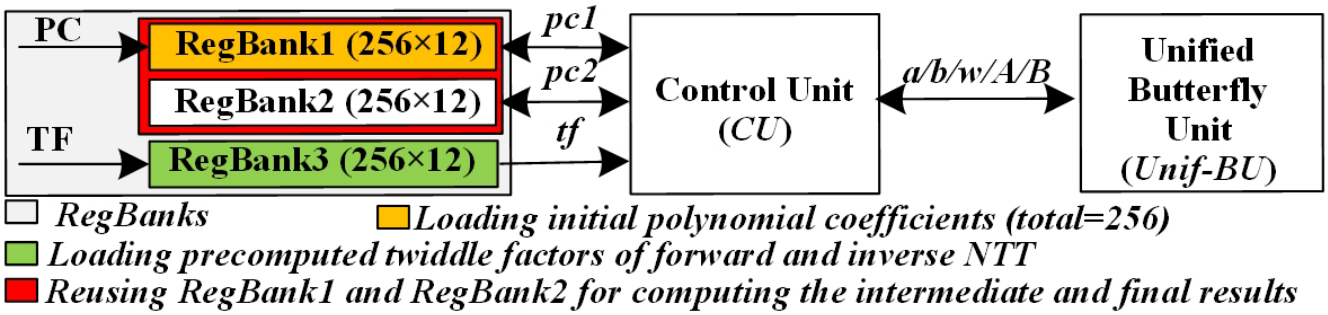


FIGURE 1. Unif-NTT: Block diagram of the proposed accelerator architecture to compute the forward and inverse NTT operations.

is shown in Figure 2. It consists of four inputs and two outputs. The inputs are a, b and w, each represented by 12 bits. Here, a and b contain the polynomial coefficients, while w corresponds to the twiddle factor. A single-bit input labelled *select* determines the forward or inverse NTT computation based on its value (0 or 1). More precisely, when the value of the *select* signal becomes 0, the forward NTT will be computed; otherwise, the inverse NTT will be computed. On the other hand, the outputs of the Unif-BU architecture are A and B, each comprising 12 bits.

We divide our Unif-BU architecture into three parts, differentiated by three colours (grey, brown and green). The grey and brown portions in Figure 2 contain components dedicated to the CTBU and GSBU. These include 12-bit arithmetic operators such as adder, subtractor and multiplier. The reason for using 12-bit operators is that our targeted CRYSTALS-Kyber PQC algorithm supports 12-bit polynomial coefficients. Moreover, we want to provide that we use the HDL-supported operators ($+$, $-$ and $\times$) to implement these arithmetic operations, i.e., adder, subtractor and multiplier. In addition, we want to highlight that we used separate adders, subtractors and multipliers to implement the CTBU and GSBU using Eq. 3 and Eq. 4. The reason for using separate arithmetic operators is that the Unif-BU corresponds to a high operating frequency with a minor hardware resource overhead. The cost of the additional resources is only the cost of one 12-bit adder, one 12-bit subtractor and one 12-bit multiplier. Using only one adder, subtractor and multiplier to process the CTBU and GSBU operations is also possible, as implemented in the unified butterfly designs of [18], [23], [25], [27]. However, this requires feedback signals, which result in longer critical path delay and influence the circuit frequency.

Despite the grey and brown colours in Figure 2, the green portion is dedicated to (both) CTBU and GSBU. It includes two routing multiplexers, i.e., $mux1$ and $mux2$. A single-bit *select* signal is standard to both these multiplexers. Whenever the *select* signal becomes 0, the $mux1$ and $mux2$ pass corresponding data related to CTBU; otherwise, these pass data related to the GSBU operations. As shown in Figure 2, the output of these multiplexers is further connected as input

Algorithm 2 Barret Reduction. Taken From [27]

Require: a
Ensure: $z \leftarrow a \bmod q$
1: **Pre-computation** $\Rightarrow$ $k = \log_2(a), x = \frac{2^k}{q}, q = 3329$
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**

to the modular reduction operations. The modular reduction operation is denoted with *mod q* in Eq. 3 and Eq. 4, and also in Figure 2. It can be observed from Eq. 3 that two *mod q* operations are needed to implement it. Similarly, Eq. 4 also requires two *mod q* operations for implementation. In total, we need four *mod q* operations. Therefore, instead of four, we employed only two *mod q* operations in the architecture of Unif-BU; see orange blocks in Figure 2. This strategy helps to decrease the hardware resources.

The *mod q* blocks in Figure 2 ensure that computations should remain within the specified modulus (i.e., $q = 3329$) range. Therefore, we have used Algorithm 2 to implement the *mod q*. It requires arithmetic and shift operations for the computation. Note that this work does not describe the architecture of the *mod q*. We refer readers to follow [27] for more insight details into the reduction architecture.

C. CONTROL UNIT (CU)

A dedicated control unit (based on FSM) is implemented in this work to generate the control signals for Unif-NTT to implement the Algorithm 1. Specifically, the CU generates the related signals for loading input data (polynomial coefficients and twiddle factors) into the corresponding RegBanks, the execution of NTT operations by the Unif-BU and retrieving results from the RegBanks to the Unif-NTT accelerator output. We provide more details below.

- The line 1 in Algorithm 1 initializes the counters t and m with their required initial values of $n/2$ and 1, respectively. These are the initializations; one clock cycle is needed to implement them.

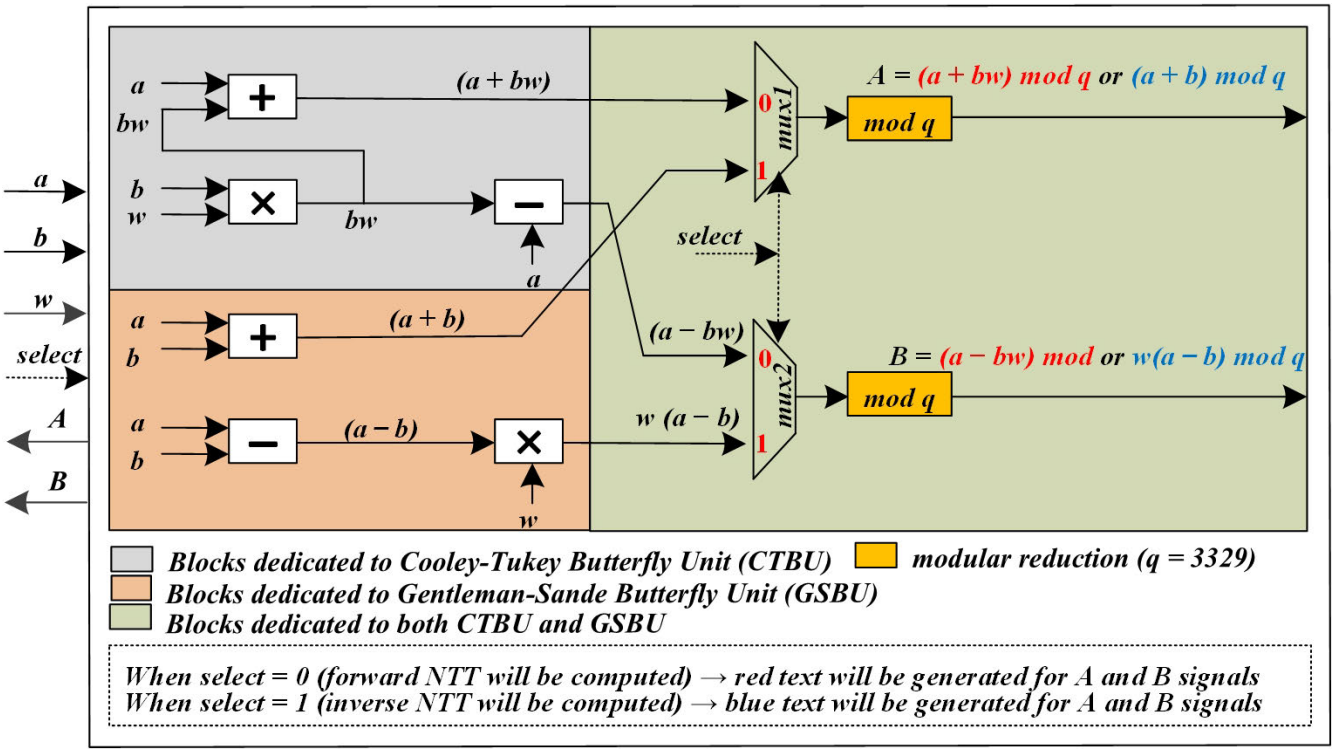


FIGURE 2. Unif-BU: Proposed unified hardware accelerator architecture of the butterfly unit for forward and inverse NTT computations.

- We have three conditional statements (one *while* and two *for*) in Algorithm 1. These conditional statements and their corresponding assignments in lines 3, 12 and 14 correspond to the memory addresses for reading/writing data (polynomial coefficients and twiddle factors). Specifically, we dealt with the *while* and *for* loops in hardware by repetitively executing the FSM states. Moreover, the t, m, j and k are the counters to implement the NTT-related operations. We set/reset these counters in their corresponding FSM states.
- The statements in lines 6 to 11 in Algorithm 1 are related to the Unif-BU. This needs 896 and 1024 clock cycles to compute the related forward and inverse NTT operations, respectively.

Regarding the clock cycles for computation, the (forward and inverse) NTT operations take 256 clock cycles to load 256 input polynomial coefficients into the RegBank1. Similarly, the corresponding precomputed values for the twiddle factors must be loaded into RegBank3, which requires 256 clock cycles. The Unif-BU requires 896 clock cycles to compute the forward NTT. Similarly, the same Unif-BU requires 1024 clock cycles to compute the inverse NTT operation, requiring additional post-processing for the CRYSTALS-Kyber PQC algorithm. Finally, 256 clock cycles are needed to load output from the corresponding RegBank to the Unif-NTT output pins. In summary, forward NTT needs 1664 (256+256+898+256) clock cycles for computations. Similarly, the inverse NTT requires 1792 (256+256+1024+256) clock cycles for computation.

TABLE 2. Area (in slices) breakdown of our Unif-NTT architecture for Xilinx Virtex-6 and Virtex-7 FPGA devices.

Design Hierarchy	Virtex-6	Virtex-7
Unif-NTT	1409	1287
⌊ Unif-BU	283	249
⌊ RegBanks + Control logic	1126	1038

IV. RESULTS, COMPARISONS, AND DISCUSSION

The implementation results are reported in section IV-A. The comparison to state-of-the-art NTT accelerators is shown in section IV-B.

A. IMPLEMENTATION RESULTS

We provide the area (in slices) breakdown of our Unif-NTT architecture for Xilinx Virtex-6 and Virtex-7 FPGA devices in Table 2. Column one shows the design hierarchy. The corresponding slice values for Virtex-6 and Virtex-7 devices are shown in columns two to three. Table 2 reveals that the Unif-BU utilizes 20% (area) of the Unif-NTT architecture for both Virtex-6 and Virtex-7 devices. Similarly, the RegBanks and control logic incorporate around 80% (slices) for the same FPGA devices.

Table 3 provides the complete implementation results of our proposed Unif-NTT architecture on FPGA and ASIC platforms. The targeted devices we used for FPGA implementations are Xilinx Virtex-6 and Virtex-7. We used TSMC 28nm process technology for ASIC synthesis. The first column of Table 3 shows the implementation device. Column two provides the NTT operation (i.e., forward and

TABLE 3. Implementation results of our Unif-NTT architecture using $n = 256$ and $q = 3329$ (after the post-place-and-route) on Xilinx FPGA and ASIC platforms. FNTT shows the forward NTT computation. Similarly, INTT determines inverse NTT computations. The ✔ compares the FPGA implementations and indicates the higher performance regarding circuit frequency, latency, throughput, consumed power and average area-time-product (Avg-ATP).

Platform	Design	Hardware Resource Utilizations				Timing Related Information				Total Power (mW)	Avg-ATP [1,2]
		FPGA			ASIC (mm^2)	Clock Cycles	Frequency (MHz)	Latency (μs)	Throughput ($Kbps$)		
		Slices	LUTs	FFs							
Virtex-6	FNTT	1409	5761	3196	–	1664	181	9.19	108.81	393 ✔	13.44×10^{-3}
	INTT					1792		9.90	101.01	401 ✔	
Virtex-7	FNTT	1287	5109	3184	–	1664	212 ✔	7.84 ✔	127.55 ✔	463	10.48×10^{-3} ✔
	INTT					1792		8.45 ✔	118.34 ✔	469	
28nm	FNTT	–	–	–	0.042	1664	2500	0.665	1503.75	67.161	0.029001×10^{-3} ✔
	INTT					1792		0.716	1396.64	69.358	

[1] ATP for FPGA ⇒ Slices × Latency.
[2] ATP for ASIC ⇒ $(mm^2)\times$ Latency.

inverse). We show the FPGA area metrics in columns three to five, including slices, look-up tables (LUTs) and flip-flops (FFs). The cost for ASIC is shown in column six in mm^2. Similarly, columns seven to ten present timing details, such as clock cycles, circuit frequency (in MHz), computation time and throughput. The computation time for a single forward and inverse NTT operation is represented as latency (in μs), calculated using Eq. 5. Additionally, throughput (in $Kbps$) is computed using Eq. 6. The total consumed power is shown in column eleven, which is the sum of static and dynamic powers computed using the Value Change Dump (VCD) files. Finally, the last column shows the average Area-Time-Product (Avg-ATP), considered a metric to evaluate the overall performance of the Unif-NTT design.

$$\text{Latency } (\mu s) = \frac{Clock\ Cycles}{Frequency\ (MHz)} \tag{5}$$

$$\text{Throughput } (Kbps) = \frac{1}{Latency\ (\mu s)} \times 10^3 \tag{6}$$

As shown in Table 3, our Unif-NTT design on Xilinx Virtex-6 FPGA utilizes 1409 slices, 5761 LUTs and 3196 FFs. This area cost on Virtex-7 is 1287 slices, 5109 LUTs and 3184 FFs. Comparatively, our Virtex-6 implementation utilizes 9% ($\frac{1409-1287}{1409} \times 100$), 11% ($\frac{5761-5109}{5761} \times 100$) and 0.37% ($\frac{3196-3184}{3196} \times 100$) higher slices, LUTs and FFs compared to Virtex-7 FPGA implementation. In other words, the Virtex-7 implementation is 9%, 11% and 0.37% more efficient regarding slices, LUTs and FFs compared to the Virtex-6 implementation of Unif-NTT. This area difference comes due to the use of distinct implementation devices. Note that the Virtex-6 and Virtex-7 devices are built on 28nm and 40nm process technologies, respectively. Therefore, FPGA devices built on modern technologies utilize lower hardware resources, like Virtex-7, compared to Virtex-6 in our experiments. Instead of the hardware cost on FPGA devices, our Unif-NTT utilizes $0.042mm^2$ on a commercial 28nm ASIC technology. If we normalize the area cost of ASIC implementation in terms of gate equivalent (total area of the implementation/area of 1 NAND gate), the cost of our Unif-NTT architecture is 29,166 ($\frac{0.042mm^2}{1.44\mu m^2}$).

The clock cycle cost of Unif-NTT to compute one forward and inverse NTT operation is 1664 and 1792, respectively. More specific details to calculate the cycle counts are discussed in section III-C. Regarding the circuit frequency, our Unif-NTT can operate on a maximum frequency of $181MHz$ and $212MHz$ on Virtex-6 and Virtex-7 devices, respectively. On similar FPGA devices, our unified butterfly unit design of Unif-BU, shown in Figure 2, can operate on a maximum frequency of $197MHz$ and $226MHz$, respectively. From these values of circuit frequencies, it can be observed that we have a frequency difference of $29MHz$ and $31MHz$ for the Unif-NTT design compared to Unif-BU on Virtex-6 and Virtex-7 devices. This frequency difference may be increased if BRAMs are utilized, as BRAMs have their read/write access time, which affects the circuit frequency. On 28nm ASIC, our Unif-NTT architecture can operate at a maximum frequency of $2500MHz$. Note that the authors of [20] reported that the access time of the inferred memories can significantly impact the overall performance, particularly the operating frequency of the design. We achieved higher circuit frequency using RegBanks instead of the BRAMs (for FPGA) and SRAMs (for ASIC).

Columns nine and ten in Table 3 show that the Unif-NTT outperforms in latency and throughput on Virtex-7 FPGA compared to Virtex-6 implementation. In other words, when we move from Virtex-6 to Virtex-7 FPGA, the average decrease in latency is 18%, and the average increase in throughput is 15%. The ASIC implementation of Unif-NTT is more efficient in latency and throughput than our efficient Virtex-7 FPGA implementation. This happens due to the different operating frequencies for Virtex-6, Virtex-7 and 28nm ASIC. On the other hand, the average power consumption of Unif-NTT is $397mW$ on Virtex-6. Similarly, the average power consumption over Virtex-7 is $466mW$. Comparatively, the Virtex-6 implementation is 15% more power-efficient than our Virtex-7 FPGA implementation. The reason is that the Virtex-6 implementation operates on a lower circuit frequency of $181MHz$ instead of $212MHz$ (which is for Virtex-7). There is always a trade-off.

The combined area and latency parameters are considered for comparisons in terms of an average ATP in the last column of Table 3. The lower the value of the Avg-ATP, the higher the performance of the Unif-NTT. Comparatively, the Unif-NTT on Virtex-7 device achieves 22% ($\frac{13.44-10.48}{13.44} \times 100$) increase in the Avg-ATP compared to the Virtex-6 implementation.

TABLE 4. Comparison to state-of-the-art NTT accelerators. FNTT shows the forward NTT computation. Similarly, INTT determines inverse NTT computations. The ✔ shows where we have higher perfromance than state-of-the-art NTT accelerators. The ✗ shows where we lose perfromance compared to state-of-the-art NTT accelerators. All these designs utilize $n = 256$ and $q = 3329$ parameters from the CRYSTALS-Kyber algorithm.

Ref.	Device	Hardware Resource Utilizations					Timing-related Results			SEC[1]	Avg-ATP[2]
		Slices	LUTs	FFs	DSPs	BRAMs	Clock Cycles	Frequency (MHz)	Latency (μs)		
[17]†	Virtex-7	675×	2128	1144	8	3	922	174	5.29	2075 (✔38%)	12.54×10^{-3} (✔54%)
[17]‡							1184		6.80		
[27]†	Virtex-7	1950×	7800	–	6	0	–	72	–	2550 (✔49%)	–
[45]†	Artix-7	187	360	145	3	2	940	115	8.17	887 (✗)	8.26×10^{-3} (✔26%)
[45]‡							1203		10.46		
[25]†	Artix-7	2713×	9508	2684	16	35	69	172	0.40	11313 (✔89%)	4.58×10^{-3} (✗)
[25]‡							71		0.41		
TW†	Virtex-7	1287	5109	3184	0	0	896	212 ✔	4.22	1287	5.82×10^{-3}
TW‡							1024		4.83		
TW†	Artix-7	1193	4731	3169	0	0	896	187 ✔	4.79	1193	6.12×10^{-3}
TW‡							1024		5.47		

† designs for the computation of forward NTT operation.
‡ designs for the computation of inverse NTT operation.
× shows the approximated slices using LUTs × 0.25 + FFs × 0.125 [44].
[1] SEC denotes the Slice-Equivalent-Cost, calculated using (BRAMs × 200) + (DSPs × 100) + Slices [46].
[2] Average ATP (Avg-ATP) ⇒ SEC × Latency (in μs).

A realistic comparison of Avg-ATP between FPGA and ASIC implementations is impossible as the area metrics are distinct.

B. COMPARISONS TO STATE-OF-THE-ART

Table 4 provides the comparison of the Unif-NTT accelerator to the state-of-the-art NTT designs. The reference designs are shown in the first column, while we show the implemented FPGA device in column two. We provide the hardware cost in Slices, LUTs, FFs, digital signal processors (DSPs) and BRAMs in columns three to seven. Similarly, in columns eight to ten, we show the timing results in terms of clock cycles, operating frequency (in MHz) and latency (or computation time in μs). The power consumption (in mW) of the designs is shown in column eleven. Finally, the last column defines the metric in terms of average ATP for realistic comparisons.

Note that the proposed Unif-NTT design does not use the DSP blocks and BRAMs for implementations. On the other hand, the reference designs from the literature frequently utilize DSPs and BRAMs for NTT accelerator architectures. Therefore, to provide a realistic area comparison, we have calculated the Slice Equivalent Cost (SEC) using a reference data sheet of [44]. Moreover, for area comparisons, we have used the SEC values. We want to highlight that the architectures compared in Table 4 provide clock cycles for only the butterfly unit to compute one forward and inverse NTT computation without considering the cost of the read/write from/to memory units. We also used the same approach to provide a realistic comparison to the state-of-the-art NTT architectures.

On identical Virtex-7 FPGA, our Unif-NTT accelerator utilizes 38% ($\frac{2075-1287}{2075}$) lower hardware resources in terms of SEC compared to [17]. This is due to the use of three 36Kb BRAMs in the reference design. Also, the reference design has used 8 DSP blocks. In our architecture, we are not using the DSPs and BRAMs as our Unif-NTT design is platform-agnostic. Also, the proposed Unif-NTT accelerator is more efficient in terms of timing-related parameters. Specifically, our accelerator operates on a maximum frequency of $212MHz$, while the NTT architecture of [17] can operate at a maximum frequency of $174MHz$. Comparatively, our Unif-NTT is 1.21× (ratio of 212 with 174) more efficient in-circuit frequency. Instead of the operating frequency, the Unif-NTT for forward and inverse NTT computations utilizes fewer clock cycles and requires lower computation time (or latency), as shown in columns eight and ten of Table 4. Overall, the Unif-NTT is 54% efficient in ATP.

Like our Unif-NTT accelerator, the reference design of [27] contains three 256×12 size of RegFiles as memory elements. Note that the NTT design of [27] supports only the computation of the forward NTT operation. The proposed Unif-NTT design supports both forward and inverse NTT operations. On the same Virtex-7 FPGA device, the Unif-NTT accelerator utilizes 49% lower area (in terms of SEC) than the reference accelerator of [27]. In addition, our Unif-NTT operates on a higher circuit frequency of $212MHz$, while the reference design can operate up to a maximum frequency of $72MHz$. The comparison of clock cycles, latency, power, and ATP is infeasible as the information related to these parameters is not available in the reference architecture.

On the Artix-7 FPGA, the NTT accelerator of [45] is more efficient in hardware resources compared to the proposed Unif-NTT design, as shown in Table 4. The reason is that the reference design has incorporated 2 BRAMs as we used three RegBanks. Despite the hardware resources, the reported clock cycles, circuit frequency and latency values in Table 4 reveal the superior performance of Unif-NTT compared to the reference design. Specifically, let us consider the computation time (or latency) for exact comparison. Our Unif-NTT architecture requires 1.70× and 1.91× lower computation time than the reference design for the execution of the forward and inverse NTT operations on the same Artix-7 FPGA device. This is due to the lower clock cycle

requirement and higher circuit frequency of the Unif-NTT design. Moreover, the proposed Unif-NTT design is 26% more efficient in terms of the ATP.

An interesting NTT accelerator is implemented in [25] over Artix-7 FPGA, where the computation time is optimized using 16 butterfly units and 35 BRAMs. If we look at the calculated SEC value, the reference design achieves 11313, which is comparatively 89% higher than the Unif-NTT accelerator. In other words, the proposed Unif-NTT design is 89% area-efficient. On the other hand, the reference design requires lower clock cycles and computation time compared to the Unif-NTT architecture, as seen in columns eight and ten of Table 4. The reason is that the reference design uses 16 butterfly units in the data path of the NTT architecture, and our accelerator follows an iterative approach with one butterfly unit for computations. Regarding the circuit frequency for comparison, our architecture is 1.08× (ratio of 187 with 172) faster. Apart from the individual comparison of area and timing results, the reference design is 25% efficient in terms of the ATP.

C. DISCUSSIONS

The blue-checkmark in Table 3 reveals that the proposed Unif-NTT architecture outperforms on Virtex-7 FPGA than Virtex-6. This is expected due to the different implementation technologies. The comparison to the state-of-the-art in Table 4 indicates that the Unif-NTT architecture operates on higher circuit frequencies due to the use of three RegBanks.

When we look at the individual area and timing results from Table 4, we can not say that the Unif-NTT design is the best one in area-efficient or latency-optimized amongst state-of-the-art designs because compared to NTT designs of [17], [25], and [27], we outperform in the hardware area. In contrast, in the case of [17] and [45], we are winning in computation time (or latency). Therefore, our Unif-NTT accelerator is suitable for applications that demand reasonable hardware resources with processing speed.

As shown in Table 4, the most optimized NTT accelerator in terms of hardware resources over Artix-7 device is described in [45]. Overall, the proposed Unif-NTT shows 54% ($\frac{12.54-5.82}{12.54} \times 100$) and 26% ($\frac{8.26-6.12}{8.26} \times 100$) speed-up in terms of Avg-ATP compared to the NTT accelerators of [17] and [45], respectively.

V. CONCLUSION AND FUTURE DIRECTION

This paper has presented a Unif-NTT accelerator to compute the forward and inverse NTT operations for PQC algorithms. As a case study, the parameters of $n = 256$ and $q = 3329$ from the CRYSTALS-Kyber algorithm are utilized for realization on Xilinx FPGA and ASIC platforms. The Unif-NTT comprises three RegBanks of each 256×12, one unified butterfly unit of CT and GS operations and a dedicated controller. The core component is the butterfly unit, which is implemented in our accelerator using two adders, multipliers and subtractors. Additionally, two routing multiplexers and two modular reduction units are shared across the arithmetic operators to generate the desired outputs.

The use of RegBanks instead of the BRAMs (for FPGA) and SRAMs (for ASIC) results in an optimized circuit frequency with area overhead. Our Unif-NTT can operate up to a maximum frequency of $212MHz$ and $2.5GHz$ over Virtex-7 FPGA and 28nm ASIC platforms, respectively. Also, sharing two routing multiplexers and two modular reduction units across the arithmetic operators in the butterfly unit allows us to obtain the 26% ATP-efficient implementation compared to the most optimized area-efficient NTT design from the state-of-the-art. The detailed evaluation of results and comparison to existing NTT accelerators indicate that the Unif-NTT design is appropriate for applications that demand reasonable hardware resources with processing speed.

The proposed Unif-NTT accelerator can further be optimized in future using pipelining to maximize the circuit frequency. Also, parallelism can be achieved using several instances of the butterfly unit in the Unif-NTT to reduce the overall cycle counts. The proposed Unif-NTT design is specific to CRYSTALS-Kyber parameters of $n = 256$ and $q = 3329$. However, to increase the versatility of Unif-NTT design in future, it can be optimized by considering the techniques of [47] and [48]. Moreover, the Unif-NTT design computes only the forward and inverse NTT operations. However, it can further be optimized to include support for the matrix-vector polynomial multiplications of the CRYSTALS-Kyber algorithm, like the design(s) of [49]. To make the Unif-NTT design configurable, interesting approaches from [17], [28], and [31] can be considered.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, doi: 10.1145/359340.359342.
[2] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology—CRYPTO '85*. Cham, Switzerland: Springer, pp. 417–426.
[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, Jan. 1999.
[4] F. Arute, T. S. Humble, D. Liakh, and J. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
[5] M. Gong et al., "Quantum walks on a programmable two-dimensional 62-qubit superconducting processor," *Science*, vol. 372, no. 6545, pp. 948–952, May 2021.
[6] P. Ball, "First 100-qubit quantum computer enters crowded race," *Nature*, vol. 599, p. 574, Jun. 2021.
[7] P. Schwabe. (2024). *Crystals-Kyber: Cryptographic Suite for Algebraic Lattices*. Accessed: Feb. 02, 2024. [Online]. Available: https://pq-crystals.org/kyber/
[8] (2024). *Crystals-Dilithium: Cryptographic Suite for Algebraic Lattices*. Accessed: Feb. 1, 2024. [Online]. Available: https://pq-crystals.org/dilithium/
[9] A. Hulsing. (2023). *Sphincs+: Stateless Hash-Based Signatures*. Accessed: Dec. 24, 2023. [Online]. Available: https://sphincs.org/
[10] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. (2023). *Falcon: Fast-Fourier Lattice-Based Compact Signatures Over NTRU Specifications V1.1*. Accessed: Dec. 28, 2023. [Online]. Available: https://falcon-sign.info

[11] NIST. (2024). *Post-Quantum Cryptography: Digital Signature Schemes: Round 1 Additional Signatures*. Accessed: Mar. 15, 2024. [Online]. Available: https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures

[12] M. Sudan, "Ideal error-correcting codes: Unifying algebraic and number-theoretic algorithms," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*. Cham, Switzerland: Springer, 2001, pp. 36–45.

[13] E. Brier, J.-S. Coron, R. Geraud, D. Maimut, and D. Naccache, "A number-theoretic error-correcting code," in *Proc. 8th Int. Conf.*, 2015, pp. 25–35.

[14] A. Pedrouzo-Ulloa, J. R. Troncoso-Pastoriza, and F. Pérez-González, "Number theoretic transforms for secure signal processing," *IEEE Trans. Inf. Forensics Security*, vol. 12, pp. 1125–1140, 2017.

[15] G. A. Jullien, "Number theoretic techniques in digital signal processing," in *Advances in Electronics and Electron Physics*. New York, NY, USA: Academic, 1991, pp. 69–163.

[16] P. Duong-Ngoc, S. Kwon, D. Yoo, and H. Lee, "Area-efficient number theoretic transform architecture for homomorphic encryption," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 3, pp. 1270–1283, Mar. 2023.

[17] K. Derya, A. C. Mert, E. Özturk, and E. Savas, "CoHA-NTT: A configurable hardware accelerator for NTT-based polynomial multiplication," *Microprocessors Microsystems*, vol. 89, Mar. 2022, Art. no. 104451.

[18] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "KaLi: A crystal for post-quantum security using Kyber and Dilithium," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 2, pp. 747–758, Feb. 2023.

[19] L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of kyber on Cortex-M4," in *Progress in Cryptology—AFRICACRYPT*. Cham, Switzerland: Springer, 2019, pp. 209–228.

[20] M. Imran, F. Almeida, J. Raik, A. Basso, S. S. Roy, and S. Pagliarini, "Design space exploration of SABER in 65nm ASIC," in *Proc. 5th Workshop Attacks Solutions Hardw. Secur.*, Nov. 2021, pp. 85–90.

[21] M. Imran, F. Almeida, A. Basso, S. Sinha Roy, and S. Pagliarini, "High-speed SABER key encapsulation mechanism in 65nm CMOS," *J. Cryptograph. Eng.*, vol. 13, no. 4, pp. 461–471, Nov. 2023.

[22] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[23] C. Zhang, D. Liu, X. Liu, X. Zou, G. Niu, B. Liu, and Q. Jiang, "Towards efficient hardware implementation of NTT for kyber on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[24] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "Towards efficient kyber on FPGAs: A processor for vector of polynomials," in *Proc. 25th Asia South Pacific Design Autom. Conf.*, Jan. 2020, pp. 247–252.

[25] F. Yaman, A. C. Mert, E. Ozturk, and E. Savas, "A hardware accelerator for polynomial multiplication operation of crystals-Kyber PQC scheme," in *Proc. Design, Autom. Test Eur. Conf. Exhibition*, 2021, pp. 1020–1025.

[26] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography," in *Proc. IEEE 28th Symp. Comput. Arithmetic*, Jun. 2021, pp. 94–101.

[27] S. Khan, A. Khalid, C. Rafferty, Y. A. Shah, M. O'Neill, W.-K. Lee, and S. O. Hwang, "Efficient, error-resistant NTT architectures for crystals-Kyber FPGA accelerators," in *Proc. IFIP/IEEE 31st Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2023, pp. 1–6, doi: 10.1109/vlsi-soc57769.2023.10321885.

[28] A. C. Mert, E. Özturk, and E. Savas, "FPGA implementation of a run-time configurable NTT-based polynomial multiplication hardware," *Microprocessors Microsystems*, vol. 78, Oct. 2020, Art. no. 103219.

[29] C. Xu, H. Yu, W. Xi, J. Zhu, C. Chen, and X. Jiang, "A polynomial multiplication accelerator for faster lattice cipher algorithm in security chip," *Electronics*, vol. 12, no. 4, p. 951, Feb. 2023.

[30] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, and L. Fanucci, "A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms," *IEEE Access*, vol. 9, pp. 150798–150808, 2021.

[31] S.-H. Liu, C.-Y. Kuo, Y.-N. Mo, and T. Su, "An area-efficient, conflict-free, and configurable architecture for accelerating NTT/INTT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 3, pp. 519–529, Mar. 2024.

[32] S. Di Matteo, M. L. Gerfo, and S. Saponara, "VLSI design and FPGA implementation of an NTT hardware accelerator for homomorphic SEAL-embedded library," *IEEE Access*, vol. 11, pp. 72498–72508, 2023.

[33] M. Saoudi, A. Kermiche, O. H. Benhaddad, N. Guetmi, and B. Allailou, "Low latency FPGA implementation of NTT for kyber," *Microprocessors Microsystems*, vol. 107, Jun. 2024, Art. no. 105059.

[34] S. D. Matteo, I. Sarno, and S. Saponara, "CRYPHTOR: A memory-unified NTT-based hardware accelerator for post-quantum crystals algorithms," *IEEE Access*, vol. 12, pp. 25501–25511, 2024.

[35] H. Yang, R. Chen, Q. Wang, Z. Wu, and W. Peng, "Hardware acceleration of NTT-based polynomial multiplication in crystals-Kyber," in *Information Security and Cryptology*. Cham, Switzerland: Springer, 2024, pp. 111–129.

[36] T.-H. Nguyen, B. Kieu-Do-Nguyen, C.-K. Pham, and T.-T. Hoang, "High-speed NTT accelerator for crystal-Kyber and crystal-Dilithium," *IEEE Access*, vol. 12, pp. 34918–34930, 2024.

[37] M. Imran, S. Khan, A. Khalid, C. Rafferty, Y. Shah, S. Pagliarini, M. Rashid, and M. O'Neill, "Evaluating NTT/INTT implementation styles for post-quantum cryptography," *IEEE Embedded Syst. Lett.*, vol. 1, no. 1, pp. 1–6, Jun. 2024.

[38] A. C. Mert, E. Karabulut, E. Özturk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, vol. 71, no. 11, pp. 2829–2843, Nov. 2022.

[39] M. Imran and M. Rashid, "Architectural review of polynomial bases finite field multipliers over GF(2m)," in *Proc. Int. Conf. Commun., Comput. Digit. Syst.*, Mar. 2017, pp. 331–336.

[40] P. He, Y. Tu, T. Bao, L. Sousa, and J. Xie, "COPMA: Compact and optimized polynomial multiplier accelerator for high-performance implementation of LWR-based PQC," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 4, pp. 596–600, Apr. 2023.

[41] M. Imran, Z. U. Abideen, and S. Pagliarini, "An open-source library of large integer polynomial multipliers," in *Proc. 24th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2021, pp. 145–150.

[42] X. Hu, J. Tian, M. Li, and Z. Wang, "AC-PM: An area-efficient and configurable polynomial multiplier for lattice based cryptography," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 2, pp. 719–732, Feb. 2023.

[43] M. Imran, Z. U. Abideen, and S. Pagliarini, "An experimental study of building blocks of lattice-based NIST post-quantum cryptographic algorithms," *Electronics*, vol. 9, no. 11, p. 1953, Nov. 2020.

[44] (2024). *7 Series FPGAs Data Sheet: Overview*. [Online]. Available: https://docs.amd.com/v/u/en-U.S./ds180_7Series_Overview.

[45] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-set accelerated implementation of crystals-Kyber," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4648–4659, Nov. 2021.

[46] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized school-book polynomial multiplication for compact lattice-based cryptography on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2459–2463, Oct. 2019.

[47] B. Li, Y. Yan, Y. Wei, and H. Han, "Scalable and parallel optimization of the number theoretic transform based on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 2, pp. 291–304, Feb. 2024.

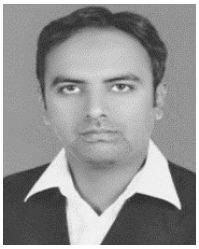
[48] J. Mu, Y. Ren, W. Wang, Y. Hu, S. Chen, C.-H. Chang, J. Fan, J. Ye, Y. Cao, H. Li, and X. Li, "Scalable and conflict-free NTT hardware accelerator design: Methodology, proof, and implementation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1504–1517, May 2023.

[49] W. Tan, Y. Lao, and K. K. Parhi, "KyberMat: Efficient accelerator for matrix-vector polynomial multiplication in crystals-Kyber scheme via NTT and polyphase decomposition," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2023, pp. 1–9.

ALI YAHYA HUMMDI received the Ph.D. degree from The University of Sheffield, U.K., in 2022. He is currently an Assistant Professor with the Department of Mathematics, King Khalid University, Abha, Saudi Arabia. His research interests include non-commutative Noetherian rings and their modules (especially rings of differential operators and D-modules), dimension (the Krull dimension and the Gelfand-Kirillov dimension), graded and filtered algebras, and cryptography.

AMER ALJAEDI received the B.Sc. degree from King Saud University, Saudi Arabia, in 2007, the M.Sc. degree in information systems security from Concordia University of Edmonton, Canada, in 2011, and the Ph.D. degree in security engineering from the Department of Computer Science, Colorado University, Colorado Springs, CO, USA, in 2018. He is currently an Associate Professor with the College of Computing and Information Technology, University of Tabuk. Before that, he was a Senior Research Member with the Cybersecurity Laboratory, Colorado University. His research interests include software-defined networking, artificial intelligence, cloud computing, the IoT, and cybersecurity. He received multiple research awards from UCCS, UT, and SACM for his outstanding research articles.

ZAID BASSFAR received the B.S. degree in computer science, the M.S. degree in information technology and communication, and the Ph.D. degree in web applications, in 2007, 2010, and 2014, respectively. He has been an Associate Professor with the College of Computer and Information Technology, University of Tabuk, since 2020. His research interests include web applications, digital transformation, augmented reality, virtual reality, cloud computing, virtual learning environments, e-learning, m-learning, and human–computer interaction.



SAJJAD SHAUKAT JAMAL received the Ph.D. degree in mathematics from Quaid-i-Azam University, Islamabad, Pakistan. He is currently an Assistant Professor with the Department of Mathematics, King Khalid University, Abha, Saudi Arabia. He has several quality research articles in well-reputed journals on the application of mathematics in multimedia security. His research interests include mathematics, number theory, cryptography, digital watermarking, and steganography.

MOHAMMAD MAZYAD HAZZAZI received the Ph.D. degree in mathematics from the University of Sussex, Brighton, U.K. He is currently an Associate Professor with the Department of Mathematics, King Khalid University, Abha, Saudi Arabia. He has more than 30 research publications in various international research journals. His research interests include cryptography and coding theory.

MUJEEB UR REHMAN received the Ph.D. degree (Hons.) in AI and cybersecurity, in 2022. He is currently a Lecturer with the School of Computer Science and Informatics, Cyber Technology Institute, De Montfort University, U.K. He is also a Professional Engineer. He has numerous high-impact factor publications, including contributions to prestigious transactions within his area of expertise. His research interests include artificial intelligence, cybersecurity, non-invasive health care, the Internet of Things (IoT), and multimedia encryption.