Scalable and Conflict-free NTT Hardware Accelerator Design: Methodology, Proof and Implementation

Jianan Mu, Yi Ren, Wen Wang, Yizhong Hu, Shuai Chen, Chip-Hong Chang, *Fellow, IEEE*, Junfeng Fan, Jing Ye, Yuan Cao *Member, IEEE*, Huawei Li, *Senior Member, IEEE*, Xiaowei Li, *Senior Member, IEEE*

***Abstract*—Number Theoretic Transform (NTT) is useful for the acceleration of polynomial multiplication, which is the main performance bottleneck in the next-generation cryptographic schemes. Different NTT based cryptographic algorithms have different security settings. The diverse application scenarios introduce different cost-performance trade-offs and hardware constraints. Motivated by the emerging demand for more versatile NTT hardware accelerators, we propose a new design methodology that can generate area-efficient and high-performance NTT accelerators for any length and modulus of NTT polynomials and single processing element (PE) or PE array with a varying number of layers. The proposed NTT accelerator architecture pivots on a conflict-free memory access pattern for adaptation to different combinations of security and PE array configuration parameters. The proposed memory access pattern is formally proved to be conflict-free for any parametric configurations. The criterion for read-after-write conflict without pipeline stall is also established. Our proposed design methodology can produce NTT accelerators with single PE or multi-layer PE array for different polynomial size and modulus, with hardware area and computational efficiency comparable to accelerators customized for a fixed set of parameters. Our proposed methodology produces parameterized accelerator with higher scalability than the existing parameterized accelerator design. On average, the accelerators generated by our proposed method are 71.4% more area-time efficient. Up to 30.7% area-time reduction over the most area-time efficient state-of-the-art scalable NTT accelerator can be achieved for the same security parameters.**

***Index Terms*—Number Theoretic Transform, Scalable hardware design, Memory access pattern, Post-quantum cryptography.**

I. Introduction

With the rapid development and standardization efforts of Post-Quantum Cryptography (PQC) [1] and Fully Homomorphic Encryption (FHE) [2], specific algebraic structures of ideal lattice center around number theory and polynomials have been explored for the implementation of quantum-safe and privacy-preserving cryptosystems. Despite the promise of various lattice-based constructions, very few results have been reported on the constraints and scalability issues of their efficient hardware translation. Polynomial multiplication over rings is the most computationally intensive operation of lattice-based cryptosystems. Number Theoretic Transform (NTT) has been widely used to accelerate polynomial multiplication of PQC and FHE by reducing its computational time complexity from $O(N^2)$ to $O(N \log_2 N)$, where N is the length of the polynomial. While polynomial multiplications can be accelerated by NTT, the large memory requirement and complex memory access pattern of NTT execution are the main bottlenecks that limit further performance improvement of hardware acceleration of PQC and FHE. Dedicated parallel and pipeline NTT architectures have been proposed to boost the computational efficiency for specific NTT security parameters. However, these architectural optimization strategies have difficulty in adapting to change in security level and hardware parallelism requirements.

Three scalability issues need to be overcome in the design of a unified NTT hardware accelerator. The first challenge is the hardware adaptation to different security parameters, i.e., the polynomial length N and the polynomial modulus q, of diverse cryptographic algorithms arising from the evolving standardization process of PQC and different threat models. The second challenge is the configurability of a unified hardware design methodology to support different number of processing elements (PEs) in multi-processor and reconfigurable hardware platforms without introducing significant performance and hardware resource penalties. The last challenge pertains to the configuration of PE array. A single dimension PE array can be configured into different two-dimensional pipelined array structures of different depths, d_{PE}, to reduce the hardware overhead without increasing the throughput and latency for the same set of security parameters. It is experimentally demonstrated in [3], [4] that for the same N, q and number of PEs, n_{PE}, a two-layer PE array is more area-efficient than a single-layer PE array.

Most of the existing works focus on accelerating NTT computations for a specific cryptographic algorithm with a fixed security parameter N and q, e.g., [5], [6]. The resulting design methodology can produce optimized NTT hardware accelerators for only a limited set of security parameters and

Jianan Mu, Jing Ye, Huawei Li and Xiaowei Li are with the State Key Laboratory of Processors (SKLP), Institute of Computing Technology, Chinese Academy of Sciences, and the University of Chinese Academy of Sciences, Beijing, China 100190, and also with CASTEST, Beijing, China 100190.

Yi Ren is with the School of Software & Microelectronics, Peking University, Beijing, China 100871.

Wen Wang is with the Computer Architecture and Security LAB, Yale University, New Haven, CT, United States 06511.

Yizhong Hu and Junfeng Fan are with Open Security Research, Shenzhen, China 518063.

Shuai Chen is with the Rock-solid Security Lab., Binary Semiconductor Co. Ltd., Suzhou, China 215000.

Chip-Hong Chang is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798.

Yuan Cao is with the College of Internet of Things Engineering, Hohai University, Changzhou, China 213022.

a fixed number of PEs. Most recent works [7], [8] reported experimental results of hardware accelerator architectures that are configurable for N, q and n_{PE} at design time. However, the reported results are based on a single PE or fixed configuration of multi-PE array implementation. With greater design flexibility, contention can happen when coefficients stored in the same memory block are required by different PEs at the same time. Conflict-free memory pattern for general parameter set has not been formally proved for these solutions.

The greatest challenge in universal design methodology for scalable NTT accelerator architecture is the efficient dataflow with complex memory access pattern. Typically, the polynomial coefficients are stored in RAMs. All the NTT computations are instantiated with data movements centered around these memory blocks. Memory access conflicts and read-after-write (RAW) conflicts can occur in the movement of data between PEs and RAMs. While it is possible to avoid these conflicts by adding pipeline stalls as exemplified by [7], [8], the performance of the cryptosystem will inevitably be compromised. Hence, the major challenge of designing a multi-functional NTT accelerator is the provision of a universal conflict-free data flow scheduling scheme without pipeline stall regardless of security parameter and PE array size and configuration. Insofar, no successful attempt has been reported on fully parameterizable universal architecture that approaches the ideal clock cycle of $\frac{N \cdot log_2 N}{2n_{PE}}$ for diverse combinations of N, q, n_{PE} and d_{PE}. As it is not possible to exhaustively evaluate conflicts for all possible combinations of parameter set, it is critically important to establish the formal proof for the conflict-free memory access pattern of a fully parameterizable accelerator design.

In this paper, we propose a novel parameterizable NTT hardware accelerator generator that can produce area-time efficient designs for any combinations of polynomial length N, polynomial modulus q, number of PEs, n_{PE} and number of layers d_{PE} of the PE array. Specifically, our contributions can be summarized as follows:

- A memory partition and coefficient assignment scheme is proposed for conflict-free memory access without pipeline stalls regardless of N, q, n_{PE} and d_{PE}.
- The proposed NTT memory access pattern is formally proved to be unconditionally free from memory conflict.
- A closed-form expression of the criterion for RAW conflict avoidance with our proposed access pattern is derived. The criterion can be easily met by many different parameter combinations to achieve the ideal NTT clock cycles of $\frac{N \cdot log_2 N}{2n_{PE}}$ upon filling up the pipeline.
- The conflict-free memory pattern and theoretically achievable clock cycles are validated by experimental results of the accelerators generated by our design method with different combinations of N, q, n_{PE} and d_{PE}. Compared with existing works of the same parameter sets, the NTT accelerators synthesized by our design methodology can save up to 33.2% clock cycles. We also evaluate the area-time trade-offs of our solutions on FPGA implementation and compare them against existing fixed-parameter and limited scalability parameterized designs to demonstrate the ascendency of our proposed NTT accelerator design methodology.

The rest of this paper is organized as follows. NTT based polynomial multiplication algorithm, scalability issues of NTT accelerators, memory and RAW conflicts, and conflict-free schedule for single and multi-PE architectures are introduced in Section II. In Section III, the proposed design methodology for fully scalable and conflict-free NTT accelerators is presented. In Section IV, the proposed memory access pattern is proved to be free from memory conflicts for any possible N, n_{PE} and d_{PE} and the criterion to avoid RAW conflict with the proposed access pattern is established. In Section V, a unified NTT hardware accelerator generator is implemented to automatically produce synthesizable HDL codes for conflict-free NTT accelerators with any combinations of N, q, n_{PE} and d_{PE}. The areas, latencies and area-time products of the synthesized designs are analyzed and compared against existing works for the same parameter set. The paper is concluded in Section VI.

II. Preliminaries

A. NTT-Based Polynomial Multiplication

Polynomial multiplication is a computational bottleneck of lattice-based cryptography such as Dilithium [9] and Falcon [10] schemes from the PQC family. In these algorithms, polynomials are defined over the ring $R_q[x] = \mathbb{Z}_q[x]/(x^N+1)$, where $\mathbb{Z}_q$ is a quotient ring with prime q, N is an integer and q satisfies $q \equiv 1 \bmod 2N$. The irreducible polynomial $x^N + 1$ of degree N enables the use of $x^N \equiv -1$ to simplify polynomial multiplication. The polynomial multiplication, $c = a \cdot b$, can be performed by representing the operands, a and b, and the product c as polynomials in the ring R_q. The computational complexity of the schoolbook polynomial multiplication is $O(N^2)$. Fast Fourier Transform (FFT) can reduce this complexity to $O(N \log_2 N)$ but roundoff errors can occur in complex number operations. NTT can be seen as a Discrete Fourier Transform (DFT) in the finite field. It enables polynomial arithmetic to be carried out in the integer domain with no roundoff error. To compute a modular product in NTT, the length of operand polynomials needs to be doubled. Furthermore, modulo $(x^N + 1)$ reduction is also required. To eliminate polynomial doubling and reduction operations, negative wrapped convolution is commonly used, with a small overhead on pre-processing the input polynomials and post-processing the output polynomial. Let ω and φ be the N-th and $2N$-th primitive roots of unity in Z_q, respectively. A typical NTT-based polynomial multiplication is described as follows:

$$\text{Preprocessing}: \ a'_i = a_i \cdot \varphi^i, \ b'_i = b_i \cdot \varphi^i.$$
$$\text{Multiplication}: \ c' = \mathbf{INTT}(\mathbf{NTT}(a') \odot \mathbf{NTT}(b')).$$
$$\text{Postprocessing}: \ c_i = c'_i \cdot \varphi^{-i}.$$

where INTT is the inverse NTT.

The iterative NTT algorithm for transforming an operand into NTT domain and the algorithm for performing polynomial multiplication are described by the pseudo codes in Algorithm 1 and Algorithm 2, respectively. The symbol $\odot$ in Algorithm 2 denotes a point-wise multiplication operator.

Algorithm 1 Iterative NTT

Require: $a'(x) \in \mathbb{Z}_q/(x^N+1), \omega, N, n_{PE}$
Ensure: $A(x) = \mathbf{NTT}(a'(x)) \in \mathbb{Z}_q[x]/(x^N+1)$
1: $A(x) \leftarrow$scramble(a')
2: **for** s incremented by 1 from 0 to $log_2 N - 1$ **do**
3: $m = 2^s$
4: **for** k incremented by 2m from 0 to $N-1$ **do**
5: **for** j incremented by 1 from 0 to $m-1$ **do**
6: $OP0 \leftarrow A[k+j]$
7: $OP1 \leftarrow A[k+j+m]$
8: $A[k+j] \leftarrow OP0 + \omega^{(j\frac{N}{2m})} \cdot OP1 (\mathrm{mod} q)$
9: $A[k+j+m] \leftarrow OP0 - \omega^{(j\frac{N}{2m})} \cdot OP1 (\mathrm{mod} q)$
10: **end for**
11: **end for**
12: **end for**
13: return $A(x)$

Algorithm 2 NTT-Based Polynoimal Multiplication

Require: $a(x), b(x) \in \mathbb{Z}_q/(x^N+1)$, 2N-th primitive root of unity $\varphi \in \mathbb{Z}_q$
Ensure: $c(x) = a(x) \cdot b(x), c(x) \in \mathbb{Z}_q[x]/(x^N+1)$
1: $A(x) \leftarrow \mathbf{NTT}(a(x) \odot (\varphi^0, \varphi^1, ..., \varphi^{N-1}))$
2: $B(x) \leftarrow \mathbf{NTT}(b(x) \odot (\varphi^0, \varphi^1, ..., \varphi^{N-1}))$
3: $C(x) \leftarrow A(x) \odot B(x)$
4: $c(x) \leftarrow (\mathbf{INTT}C(x)) \odot (\varphi^0, \varphi^{-1}, ..., \varphi^{-(N-1)})$
5: return $c(x)$

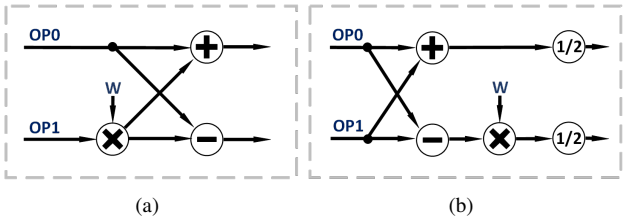
In Algorithm 1, Steps 6 and 7 load the polynomial coefficients. Steps 8 and 9 perform the NTT arithmetic depicted by the butterfly operation in Fig. 1(a) and update the coefficients. The polynomials and pre-computed twiddle factors are usually stored in separate blocks of memory. Dedicated Processing Elements (PEs) are designed to perform the butterfly operations. The data flow between the PEs and memory is managed by a finite state machine.

The pre- and post-processing operations of Algorithm 2 can be merged into the NTT and INTT operations as (1) [5]. Cooley-Tukey (CT) and Gentleman and Sande (GS) butterflies are used for the merged NTT and INTT, as shown in Figs. 1(a) and (b), respectively. W in Fig. 1 represents the twiddle factor.

$$\begin{aligned} \mathbf{NTT}_N(a) &= \sum_{i=0}^{N-1} \left(\sum_{j=0}^{N-1} a_j \varphi^j \omega^{ij} \right) x^i. \\ \mathbf{INTT}_N(C) &= \sum_{i=0}^{N-1} (N^{-1} \varphi^{-i} \sum_{j=0}^{N-1} C_j \omega^{-ij}) x^i. \end{aligned} \tag{1}$$

B. Scalability of NTT Accelerators

Apart from adaptation to different security parameters N and q, an NTT design should be able to accommodate different numbers of PEs, n_{PE} and layers, d_{PE} of PE array. As shown in Algorithm 1, for a size-N polynomial, $log_2 N$ iterations (stages) are required for one NTT operation. In each stage, with only one PE, the PE has to perform $N/2$ rounds of butterfly operation. One way of accelerating the NTT operation is to process n_{PE} pairs of coefficients parallelly in each stage with n_{PE} PEs on one layer. Every PE will have to perform only $\frac{N}{(2 \cdot n_{PE})}$ rounds of butterfly operation in each stage. This straightforward approach is adopted by most existing parameterizable NTT hardware accelerators. However, this approach may lead to throughput penalty or memory overhead if memory access conflicts are to be avoided. Memory access conflicts and their resolution will be elaborated further in Section II-C and Section II-D.



(a) (b)

Fig. 1: (a) Cooley-Tukey butterfly for NTT and (b) Gentleman and Sande butterfly for INTT.

Another approach of accelerating NTT computations with multiple PEs is to place the n_{PE} PEs onto d_{PE} layers. The PE array is then configured into a two-dimensional wave-pipelined array. The two-dimensional PE array executes adjacent butterfly operations concurrently across several NTT stages. This approach is widely known as layer-merging and has been used in resource-constrained microcontroller platforms to reduce the frequency of memory accesses [11][12][13]. Some hardware designs such as [14], [3], [4] also adopt this approach. They provide experimental results on some parameters of N and n_{PE} with $d_{PE} = 2$, but do not provide formal proof of the parameters that are not experimented.

C. Memory Conflicts and RAW Conflicts

Polynomial coefficients are stored in memory blocks and accessible by the PE array. Memory conflict arises when the same memory banks are accessed at the same time by the same or different PEs to perform butterfly operations. Memory collision has thus become a major throughput bottleneck of multi-PE NTT accelerators. In the innermost loop (see Steps 6 to 9) of Algorithm 1, a pair of coefficients, $A[k+j]$ and $A[k+j+m]$, are first read out before a butterfly operation is performed. The outputs of the butterfly, $A[k+j]$ and $A[k+j+m]$, are then written back to the memory. This completes one round of NTT computation. It is clear that if $A[k+j]$ and $A[k+j+m]$ are to be kept in the same memory locations to conserve memory, two clock cycles are required for the memory read and write operations in one round of computation. This will result in an idle cycle in an NTT design with one PE, or a memory collision will occur if the throughput is to be preserved without spare swap memory. This memory access conflict issue can complicate the operation-to-memory and the PE-to-operation scheduling as the number of PEs available for NTT computation increases.

RAW conflict is a throughput bottleneck of multi-PE NTT accelerators. When the read operation for coefficient $A[i]$ in

Stage $s+1$ happens before the write operation for the result of $A[i]$ in Stage s, RAW conflicts will occur without timing buffers. To avoid RAW conflicts, idle cycles are added after each stage [7], [8].

D. Memory-conflict-free Schedules in Single- and Multi-PE NTT Accelerators

To address the memory conflict problem in single-PE NTT hardware acceleration, several memory schedules have been proposed. Johnson [15] proposed a conflict-free memory address scheme for calculating N-point FFT with only one PE. Multiple banks of simple dual-port RAM are used. A unique bank and address are assigned to each coefficient. A formal proof of conflict-free accessing was also provided in [15] to show that the coefficients in one butterfly are always stored in different banks. This memory addressing scheme has been adopted in many existing NTT accelerators [16], [17], [5]. As each coefficient is stored in the same bank and address throughout, the intermediate results are read from and written to the same memory addresses. For ease of exposition, we henceforth refer to such addressing scheme as "in-place". In [18], a different memory addressing scheme from [15] was proposed for single-PE NTT computations. Two coefficients of a butterfly are stored in one memory bank and address so that they can be read and written in one cycle. As the two coefficients of a butterfly change in different stages, the addresses in odd and even stages are swapped. For instance, in Stage 0, $A[k]$ and $A[k+1]$ are stored in one bank and address, and are addressed in one butterfly, and $A[k+2]$ and $A[k+3]$ are stored in one bank and address, and are addressed in another butterfly; in Stage 1, $A[k]$ and $A[k+2]$ are addressed in one butterfly, and $A[k+1]$ and $A[k+3]$ are addressed in another butterfly. To avoid memory conflict, the addresses of $A[k+1]$ and $A[k+3]$ are swapped after Stage 0. This way, the two coefficients of a butterfly in each stage will be kept at one memory address. This approach is later adopted in many NTT accelerators [19], [20], [21]. As the intermediate results are not written to the same address from which they are read, we henceforth refer to such memory accessing scheme as "out-of-place". Some experimental results were provided in [18] to show that the memory schedule is conflict-free for only certain parameters of N but the general conflict-free proof was not provided. However, with only one PE, the throughput is limited. In addition, conflict-free single-PE addressing schemes may not be trivially extended to multi-PE accelerators.

A conflict-free memory access for calculating N-point FFT with multiple PEs on one layer was proposed in [22]. It is proved that the scheme is free from memory conflicts for any configuration of N and n_{PE}. This memory access scheme is not in-place. The intermediate results after each stage also need to be stored. Hardware implementation results and overheads were not provided in [22] except the proof of conflict-free access and its numerical analyses. In [23], a generalized conflict-free check for FFT memory access schemes was proposed. Some memory access examples with fixed NTT parameters that passed the check were provided. However, this conflict-free memory access scheme may not be scalable with n_{PE} and NTT parameters in general. Unfortunately, hardware implementation results were also not given in [23]. Recently, a memory-based FFT hardware implementation with multiple PEs was proposed in [24] with conflict-free access inspired by the in-place scheme of [15]. However, the scalability of this conflict-free scheme with varying n_{PE} for different NTT parameters was not formally proved.

Since the 1990s, a variety of NTT/FFT hardware accelerators have been developed with different memory access schemes to tackle the memory conflict and parametric scalability problems. To increase the throughput, multiple PEs are used in NTT accelerators. The introduction of a new dimension of n_{PE} for design scalability to different NTT parameters makes conflict-free addressing significantly more challenging. Until very recently, Mert et al. [7] proposed a scalable NTT hardware implementation. Both security parameters, N and q, as well as n_{PE}, are configurable at design time. The design was demonstrated to be free from memory addressing conflict by experiments for some configurations of N, q, and n_{PE} but no formal proof was given to ensure that other configurations will not be affected by memory conflicts. By configuring the PEs in a two-dimensional array without pipelined registers, the design can be made more efficient but conflict-free memory access becomes even more challenging if the design must also be scalable to N, n_{PE} and d_{PE} of the PE array.

III. Proposed Design Methodology for Fully Scalable and Conflict-free NTT Accelerator

A. Overall Architecture

In our scalable NTT framework, the length of the polynomial N is a power-of-two integer. The number of PEs n_{PE} is a multiple of $d_{PE} \times 2^{d_{PE}-1}$ where d_{PE} is a factor of $log_2 N$. The modulus of the polynomial coefficient q is any valid integer. An overview of the architecture is shown in Fig. 2. It consists of five parts: control unit, PE array, coefficient RAM, twiddle factor memory and interfacing glue logic. The circuit performs three functions: NTT, INTT and point-wise multiplication. The control unit controls the mode, i.e., NTT, INTT or point-wise multiplication, of the PE array. This work focuses on the NTT and INTT modes. In either NTT or INTT mode, the index generator of the control unit selects the indices of coefficients for computation, and the address generator calculates the corresponding address and bank of each of these coefficients. Depending on the "mode" signal, a PE performs either a butterfly operation or the point-wise multiplication. The pipelined PE array has n_{PE} PEs in total. With d_{PE} layers, there are $w_{PE} = \frac{n_{PE}}{d_{PE}}$ PEs on each layer, where d_{PE} and w_{PE} are referred to as the depth and width, respectively of the PE array. Only the input coefficients of layer 0 are read from the memory. Other layers are directly hard wired to the previous layers. In Fig. 2, PE (l, u) is the u-th PE on layer l. The coefficient RAM is used to store the inputs, intermediate results, and final results of all NTT/INTT calculations. The Mul block in the coefficient RAM module is deserved for the storage of polynomial coefficients for point-wise multiplication. The twiddle factors for NTT and INTT

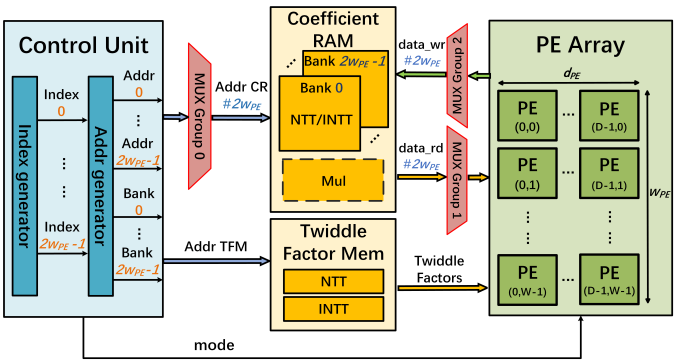


Fig. 2: Overall architecture.

are stored in separate memory blocks. The twiddle factors used in each cycle are determined by the index generator of the control unit that selects the indices of coefficients for computation. Glue logic circuits are used for the connections and communications between the control unit, memory blocks and PE array. As only $2w_{PE}$ coefficients are read by w_{PE} PEs on layer 0 in each round, a dual-port RAM of size $N \times \lceil log_2(q) \rceil$ is configured into $2w_{PE}$ banks each of depth $\frac{N}{2w_{PE}}$. To ensure that the $2w_{PE}$ coefficients needed in each round can be accessed in one cycle, each of them is stored in a different bank.

To read the $2w_{PE}$ coefficients from the corresponding bank and write the calculated results, three MUX groups within the glue logic circuits are instantiated to send $2w_{PE}$ addresses from the control unit to the coefficient RAM, and transfer $2w_{PE}$ coefficients between the RAM and the PE array. As $2w_{PE} = \frac{2n_{PE}}{d_{PE}}$, when d_{PE} increases, w_{PE} decreases. With a larger d_{PE}, the number of coefficients used in each round is smaller, which reduces the hardware consumed by the MUX groups, and hence the NTT circuit overhead. The performance of NTT is constrained by two conflicts, i.e., memory conflicts and RAW conflicts, that may arise when reading/writing coefficients from/to the coefficient RAM.

B. Conflict-free Address Assignment for Multi-layer PE Array

To avoid memory conflicts, the $2w_{PE}$ coefficients processed by the PE array in each round should be read from different memory banks. The minimum number of banks is $b = 2 \times w_{PE} = 2 \times \frac{n_{PE}}{d_{PE}}$. As shown in Fig. 2, the address generator takes the coefficient indices generated by the index generator to determine the banks and addresses of the $2w_{PE}$ coefficients in each round. Each index refers to one coefficient of the polynomial. Hence, the integer variable $idx \in [0, N-1]$ can be expressed by a string of $L = \lceil log_2 N / log_2 b \rceil$ digits in base b representation. The bank and address of each coefficient index is determined by (2), which is an extension from the address assignment equation for one PE in [15].

$$\begin{aligned} Bank &= \left(\sum_{i=0}^{L-1} d_i \right) \bmod b, \\ Addr &= \left\lfloor \frac{idx}{b} \right\rfloor . \end{aligned} \tag{2}$$

where d_i is the i-th digit of the coefficient index expressed in base b. For example, if $w_{PE} = 2$ and $N = 16$, $b = 4$ and $L = 2$. If $idx = 15_{10}$ (decimal) = 33_4 (quaternary), then $Bank = 3 + 3 \bmod 4 = 2$ and $Addr = \lfloor 15/4 \rfloor = 3$.

In Algorithm 1, $idx = j$ refers to the j-th coefficient of the polynomial. However, Algorithm 1 is not scalable for n_{PE} and d_{PE}. Even when $d_{PE} = 1$, the coefficients in each round of butterfly operations for Algorithm 1 may map to the same memory banks according to (2) if $n_{PE} > 1$ and $N > 2 \times n_{PE}$. Take $N = 16$, $n_{PE} = 2$, $d_{PE} = 1$ and $w_{PE} = 2$ as an example. The coefficients $A[0,1,2,3]$, $A[4,5,6,7]$, $A[8,9,10,11]$, and $A[12,13,14,15]$ are stored into bank $[0,1,2,3]$, $[1,2,3,0]$, $[2,3,0,1]$, and $[3,0,1,2]$ according to (2). In the first round of Stage 2, the four coefficients $A[0,4,1,5]$ are accessed by the 2 PEs simultaneously. $A[0]$ and $A[5]$ are in bank 0 and bank 2, respectively, but $A[1]$ and $A[4]$ are both stored in bank 1. The latter two coefficients cannot be accessed in one cycle.

To resolve this problem, we propose a conflict-free index generation algorithm, Algorithm 3, for multi-layer PE array. Algorithm 3 reorders the coefficient pairs such that the $2w_{PE}$ coefficients addressed in each round are from different banks.

C. Conflict-free Coefficient Index Generation for Multi-layer PE Array

In Fig. 2, $2w_{PE}$ coefficient indices to the PE array in each round are generated by the index generator based on Algorithm 3. In Algorithm 3, the "s" loop in Step 1 refers to NTT Stage "s". As the PE array of depth d_{PE} executes adjacent butterfly operations concurrently across d_{PE} NTT stages. To generate the indices for every d_{PE} stages, "s" is incremented by d_{PE} in Step 1 of Algorithm 3. As the PE array reads $2w_{PE}$ coefficients concurrently, there are $\frac{N}{2w_{PE}}$ rounds (the "r" loop in Step 8 or 17) in each Stage s. There are w_{PE} "u" loops in each round r of Stage s. Each "u" loop in Steps 10 to 13 or Steps 19 to 22 generates a pair of upper and lower coefficient indices, $idx_h(s,r,0,u)$ and $idx_l(s,r,0,u)$, for the butterfly operation of PE $(0,u)$ on layer 0. The index generation is divided into two phases. Phase 1 generates the indices for Stage 0 to Stage $log_2(\frac{N}{2w_{PE}})$, and Phase 2 generates the indices for the last $log_2(w_{PE})$ stages, i.e., Stage $log_2(\frac{N}{w_{PE}})$ to Stage $(log_2 N - 1)$. In each round of Phase 1 or Phase 2, the w_{PE} pairs of indices form an arithmetic progression of $2w_{PE}$ terms, beginning with $idx_l(s,r,0,0)$.

Fig. 3 shows the NTT calculation process and the index generation sequence for $N = 32$, $n_{PE} = 4$, $d_{PE} = 1$ and $w_{PE} = 4$. A 32-point NTT has $log_2(32) = 5$ stages. When $d_{PE} = 1$, coefficients are read from the RAM in every stage. Each stage consists of $\frac{N}{2w_{PE}} = 4$ rounds of coefficient assignment, and $w_{PE} = 4$ simultaneous butterfly operations are performed in each round. Fig. 3 uses 4 different colors to represent the four different rounds of simultaneous butterfly operations in each stage. Fig. 4 shows the RAM structure and the access patterns generated by Algorithm 3 to fulfill (2). As the 4 PEs use 8 coefficients in each round, the RAM is configured into 8 banks. In Fig. 4, the number in each block is the index of the coefficient.

As shown in Fig. 3, in Phase 1, the $2w_{PE} = 8$ indices generated in each round of Steps 8 to 14 form an arithmetic

Algorithm 3 Generation of coefficient Indices for n_{PE} PEs on d_{PE} layers

Require: $N, n_{PE}, d_{PE}, w_{PE} = \frac{n_{PE}}{d_{PE}}$
Ensure: $2w_{PE}$ indices in each round
1: **for** s from 0 by d_{PE} to $log_2(N) - 1$ **do**
2: $phase = 1$
3: $m = 2^s$
4: **if** $s > log_2(N/2w_{PE})$ **then**
5: $phase = 2$
6: **end if**
7: **if** $phase == 1$ **then**
8: **for** r from 0 by 1 to $\frac{N}{2w_{PE}} - 1$ **do**
9: $idx_l(s,r,0,0) \leftarrow \left\lfloor r/\frac{N}{2w_{PE}\cdot m} \right\rfloor + (r \bmod \frac{N}{2w_{PE}\cdot m}) \cdot (2w_{PE} \cdot m)$
10: **for** u from 0 by 1 to $w_{PE} - 1$ **do**
11: $idx_l(s,r,0,u) \leftarrow idx_l(s,r,0,0) + u \cdot 2m$
12: $idx_h(s,r,0,u) \leftarrow idx_l(s,r,0,0) + u \cdot 2m + m$
13: **end for**
14: **end for**
15: **end if**
16: **if** $phase == 2$ **then**
17: **for** r from 0 by 1 to $\frac{N}{2w_{PE}} - 1$ **do**
18: $idx_l(s,r,0,0) \leftarrow r$
19: **for** u from 0 by 1 to $w_{PE} - 1$ **do**
20: $idx_l(s,r,0,u) \leftarrow idx_l(s,r,0,0) + \lfloor u/\frac{N}{2m} \rfloor \cdot \frac{N}{2w_{PE}} + (u \bmod \frac{N}{2m}) \cdot 2m$
21: $idx_h(s,r,0,u) \leftarrow idx_l(s,r,0,0) + \lfloor u/\frac{N}{2m} \rfloor \cdot \frac{N}{2w_{PE}} + (u \bmod \frac{N}{2m}) \cdot 2m + m$
22: **end for**
23: **end for**
24: **end if**
25: **end for**

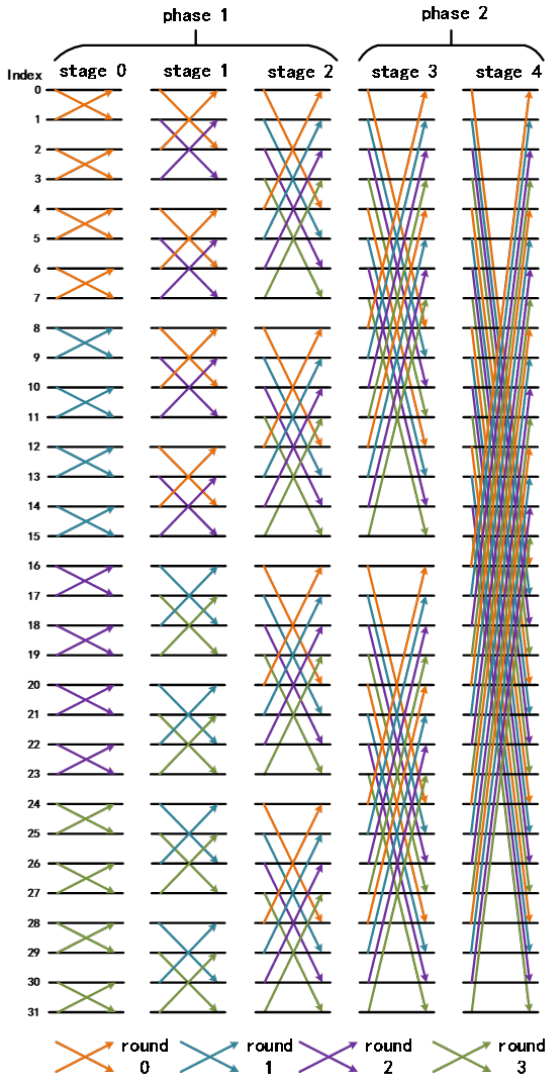


Fig. 3: NTT calculation for N = 32, n_{PE} = 4, d_{PE} = 1 and w_{PE} = 4. In each stage, the 4 butterfly operations in the same color are calculated by the 4 PEs concurrently.

progression with a common difference of $m = 2^s$, starting from $idx_l(s,r,0,0)$. In round 0 of Stage 1, $m = 2^s = 2$, $idx_l(1,0,0,0) = 0$, $idx_h(1,0,0,0) = 2$; $idx_l(1,0,0,1) = 4$, $idx_h(1,0,0,1) = 6$; $idx_l(1,0,0,2) = 8$, $idx_h(1,0,0,2) = 10$; $idx_l(1,0,0,3) = 12$, $idx_h(1,0,0,3) = 14$. Coefficients $A[0]$ and $A[2]$ are sent to PE $(0,0)$, $A[4]$ and $A[6]$ are sent to PE $(0,1)$, $A[8]$ and $A[10]$ are sent to PE $(0,2)$, $A[12]$ and $A[14]$ are sent to PE $(0,3)$. As shown in Fig. 4, the 8 coefficients $A[0,2,4,6,8,10,12,14]$ are stored in *Bank* $[0,2,4,6,1,3,5,7]$. In round 1 of Stage 1, $idx_l(1,1,0,0) = 16$, the generated indices are $[16,18,20,22,24,26,28,30]$. The 8 coefficients are stored in *Bank* $[2,4,6,0,3,5,7,1]$. In Stage 2, $m = 2^s = 4$. In round 0, $idx_l(2,0,0,0) = 0$, the generated indices are $[0,4,8,12,16,20,24,28]$. In round 1, $idx_l(2,1,0,0) = 1$, the generated indices are $[1,5,9,13,17,21,25,29]$. As shown in Fig. 4, the 8 coefficients needed in each round are stored in 8 different banks. Phase 1 ends at stage number $log_2(\frac{N}{2w_{PE}}) = log_2(\frac{32}{8}) = 2$.

Stages 3 and 4 fall in Phase 2. The $2w_{PE}$ indices generated by Steps 17 to 23 in round r of Stages 3 and 4 form an arithmetic progression with a common difference of $\frac{N}{2w_{PE}} = 4$. The 8 generated indices are the same indices in round r of Stage 2, which is the last stage of Phase 1, but they are sent to different PE inputs. In Stage 3, $m = 2^s = 8$. In round 0, $idx_l(3,0,0,0) = r = 0$, $idx_h(3,0,0,0) = 8$; $idx_l(3,0,0,1) = 16$, $idx_h(3,0,0,1) = 24$; $idx_l(3,0,0,2) = 4$, $idx_h(3,0,0,2) = 12$; $idx_l(3,0,0,3) = 20$, $idx_h(3,0,0,3) = 28$. The 8 generated indices, $[0,8,16,24,4,12,20,28]$, are the same 8 indices as in round 0 of Stage 2, which are $[0,4,8,12,16,20,24,28]$. The differences are coefficients $A[0]$ and $A[8]$ are sent to PE $(0,0)$, $A[16]$ and $A[24]$ are sent to PE $(0,1)$, $A[4]$ and $A[12]$ are sent to PE $(0,2)$, $A[20]$ and $A[28]$ are sent to PE $(0,3)$. In round 0 of Stage 4, the 8 generated indices are $[0,16,4,20,8,24,12,28]$, which are the same 8 indices of round 0 in Stage 2. As shown in Fig. 4, the 8 coefficients are stored in 8 different banks.

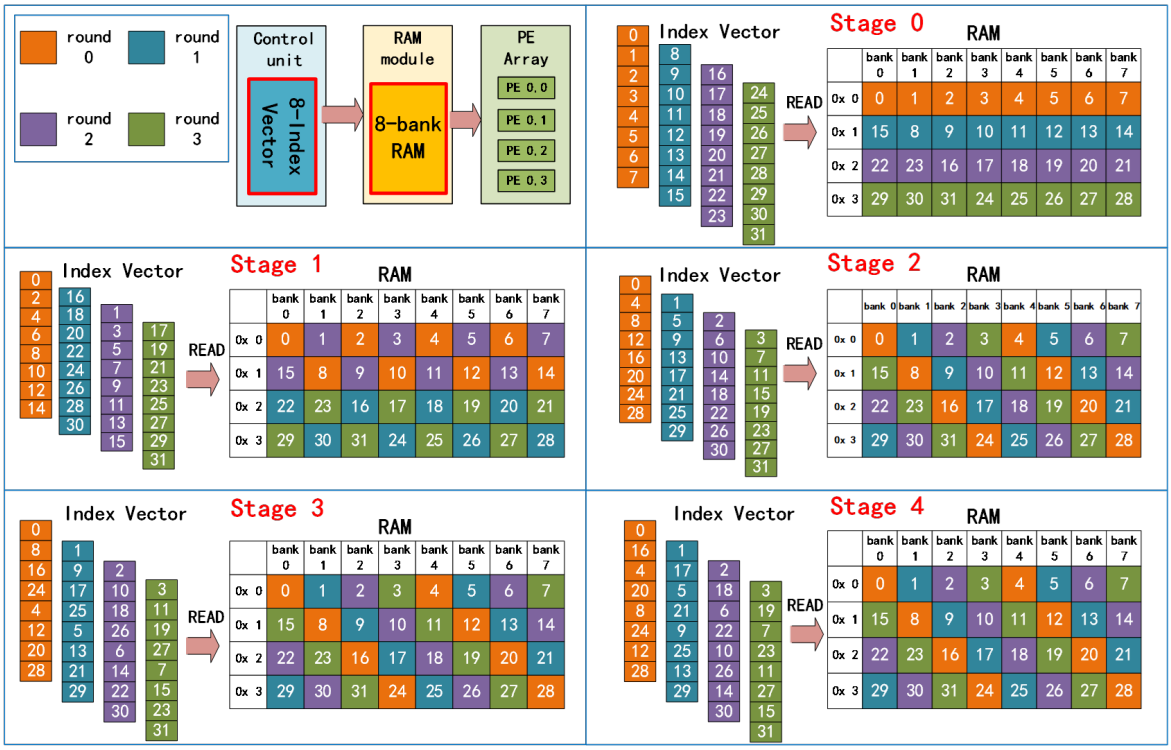


Fig. 4: RAM structure and index vectors for $N = 32$, $n_{PE} = 4$, $d_{PE} = 1$ and $w_{PE} = 4$.

D. Dataflow Connections of Multi-Layer PE Array

Within the PE array, the $2w_{PE}$ coefficients are transmitted layer by layer to perform the $w_{PE} \times d_{PE}$ butterfly operations across d_{PE} stages. PEs on layer 0 calculate the butterfly operations of Stage s, and PEs on layer l calculate the butterfly operations of Stage $s + l$. The indices of the two input coefficients for PE (l, u) have a difference of 2^{s+l}. Since the indices for layer 0 form an arithmetic progression, the data are scheduled such that the pair of coefficients assigned to each PE (l, u) on layer l have a difference of 2^{s+l} between their indices. This can be achieved by:

$$\begin{aligned}
&\text{For even } u\text{:}\\
&idx_l(s+l, r, l, u) = idx_l(s, r, 0, u'),\\
&idx_h(s+l, r, l, u) = idx_l(s, r, 0, u' + 2^{l-1}),\\
&\text{For odd } u\text{:}\\
&idx_l(s+l, r, l, u) = idx_h(s, r, 0, u'),\\
&idx_h(s+l, r, l, u) = idx_h(s, r, 0, u' + 2^{l-1}).
\end{aligned} \tag{3}$$

where $u' = \left\lfloor \frac{u}{2^l} \right\rfloor \times 2^l + \left\lfloor \frac{u \bmod 2^l}{2} \right\rfloor$.

According to (3), coefficients of indices $idx_l(s + l, r, l, u)$ and $idx_h(s + l, r, l, u)$ are assigned to PE (l, u) in round r of Stage $s + l$. According to Steps 8 to 14 and Steps 17 to 23 of Algorithm 3, the indices of these coefficients always have a difference of $2 \times 2^{l-1} \times 2^s = 2^{s+l}$ in Phases 1 and 2 when $s = [0, d_{PE}, 2d_{PE}, \cdots, (\frac{log_2 N}{d_{PE}} - 1) \cdot d_{PE}]$. The data flow depicted by (3) is fixed for a given round r and a given Stage s. Hence it can be realized at no logic cost by routing the appropriate outputs of the PEs on layer $l - 1$ to the inputs of the PEs on layer l according to (3).

Fig. 5 shows the connections of PEs in each layer of the PE array for $w_{PE} = 8$, $d_{PE} = 2, 3$ and 4, where the two input coefficients to each PE $(0, u)$ on layer 0, $idx_l(s, r, 0, u)$ and $idx_h(s, r, 0, u)$, are indicated by the orange numbers $2u$ and $2u+1$. These orange numbers are used to trace the connections from the PEs in layer 0 to the PEs in subsequent layers. For example, the output of PE (1,3) in layer 1 that has a corresponding orange input number '5' is connected to the input of PE (2,1) in layer 2 with the same orange number '5'.

Fig. 6 shows the data flow and butterfly operations of the PE array for $N = 16$, $n_{PE} = 4$, $d_{PE} = 2$ and $w_{PE} = 2$. The 2×2 PEs are connected according to (3). For $N = 16$, there are $log_2 16 = 4$ stages of butterfly operations. With $d_{PE} = 2$ and $2w_{PE} = 4$, the index generator generates 4 coefficient indices in each round of Stage 0 and Stage 2. The corresponding butterfly operations are colored orange. $A[0, 1, 2, 3]$ are read from memory in round 0 of Stage 0, with $A[0, 1]$ sent to PE $(0, 0)$ and $A[2, 3]$ to PE $(0, 1)$. The outputs of the two butterfly operations, $A[0, 2]$ and $A[1, 3]$, are sent to PE $(1, 0)$ and PE $(1, 1)$, respectively of Layer 1. In rounds 1, 2 and 3, coefficients $A[4, 5, 6, 7]$, $A[8, 9, 10, 11]$ and $A[12, 13, 14, 15]$, respectively are read from memory. In round 0 of Stage 2, $A[0, 4, 8, 12]$ are read from the memory to the PE array, with $A[0, 4]$ sent to PE $(0, 0)$ and $A[8, 12]$ to PE $(0, 1)$. The outputs of the two butterfly operations, $A[0, 8]$ and $A[4, 12]$, are sent to PE $(1, 0)$ and PE $(1, 1)$, respectively of Layer 1. In rounds 1, 2 and 3, the coefficients $A[1, 5, 9, 13]$, $A[2, 6, 10, 14]$ and $A[3, 7, 11, 15]$, respectively are read from memory. It can be inferred from (3) that w_{PE} is a multiple of $2^{d_{PE}-1}$. As shown in Fig. 5, the PE array of depth d_{PE} and $w_{PE} = 2^{d_{PE}-1}$ are connected in a group. The $2w_{PE} = 2^{d_{PE}}$ coefficients are

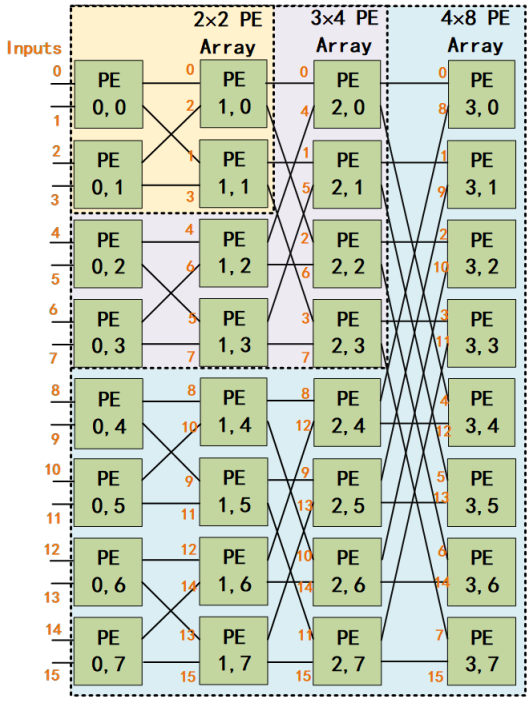


Fig. 5: Connections of 2×2 PE array (yellow shaded square box), 3×4 PE array (pink and yellow shaded square box) and 4×8 PE array (blue, pink and yellow shaded square box).

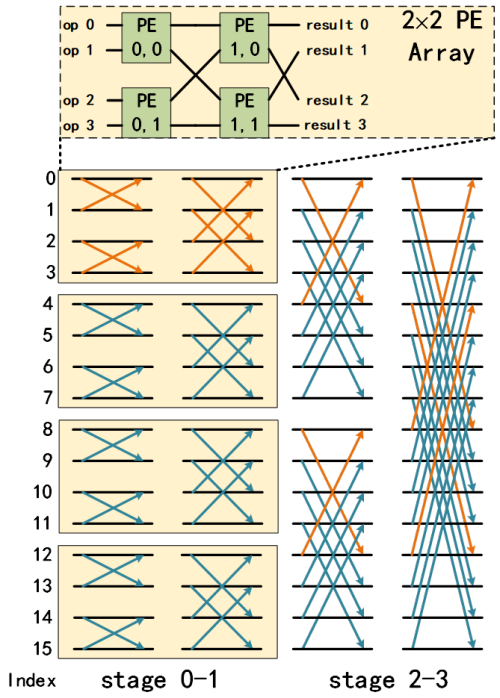


Fig. 6: NTT data flow diagram for $N = 16$, $n_{PE} = 4$ and $d_{PE} = 2$.

used within this group. This group can compute $2^{d_{PE}-1} \times d_{PE}$ butterfly operations on d_{PE} adjacent stages. If w_{PE} is not a multiple of $2^{d_{PE}-1}$, the NTT operations cannot be completed in d_{PE} stages. For example, when $d_{PE} = 2$, w_{PE} should be a multiple of $2^{2-1} = 2$. If $w_{PE} = 1$ or 3, the PE array cannot compute $w_{PE} \times 2$ butterfly operations in two stages.

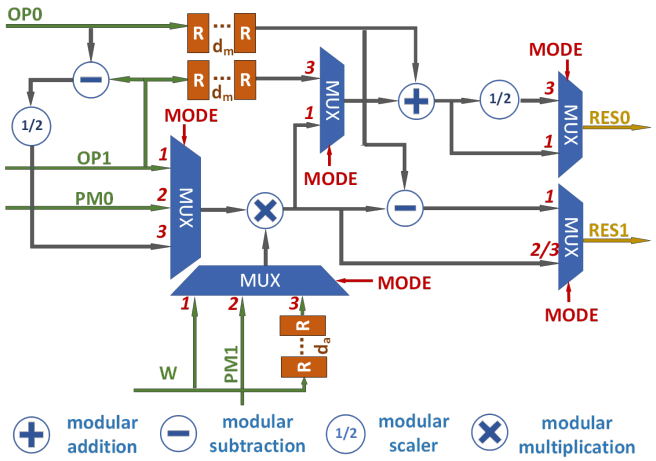


Fig. 7: Architecture of PE [5].

E. PE Structure

The basic PE architecture [5] is shown in Fig. 7. It consists of a modular adder, two modular subtractors, a modular multiplier and two modular scalers. The modular scaler for the modulo q divided-by-two operation is composed of a shifter, an adder and a multiplexer. The functional units are made scalable to q at compilation time with parameterizable bit width $\lceil log_2 q \rceil$ like [21]. Specifically, Montgomery modulo reduction [25] is adopted for parameterizable modular q multiplier design. The inputs to the PE are: $OP0$, $OP1$, $PM0$, $PM1$, W and *MODE*. The outputs are $RES0$ and $RES1$. Five multiplexers controlled by a *MODE* signal are added to schedule the data flow from the inputs to the outputs through these functional units in different modes. When $MODE = 1$, the PE performs CT-butterfly for NTT with $RES0 = (OP0 + OP1 \times W) \bmod q$ and $RES1 = (OP0 - OP1 \times W) \bmod q$; When $MODE = 2$, the PE performs point-wise multiplication with $RES1 = (PM0 \times PM1) \bmod q$; When $MODE = 3$, the PE performs GS-butterfly for INTT with $RES0 = ((OP0 + OP1)/2) \bmod q$ and $RES1 = ((OP0 - OP1)/2 \times W) \bmod q$. $OP0$ is delayed by d_m cycles before the product of $OP1$ and W is added to or subtracted from it in mode 1, where d_m denotes the number of cycles needed to complete one modular multiplication. The same delay is applied to $OP1$ to add with $OP0$ for $RES0$ in mode 3. W is delayed by d_a cycles before it is multiplied by the scaled difference of $OP1$ and $OP0$ to produce $RES1$ in mode 3, where d_a denotes the number of cycles needed to complete a modular subtraction and a modular scaling operation.

To maximize the throughput, two additional registers are added into the data path between the RAM and the PEs. Consequently, it takes $c_{PE} = (d_m + d_a + 2)$ cycles for the result of a butterfly in each PE layer to be written back to the RAM after the input data is read from the RAM to the PE. The functional units of this PE can be replaced by more optimized pipelined functional units such as the word-level Montgomery reduction in [7] but the latency of the butterfly may vary with q.

Our PE is designed for NTT/INTT with $q \equiv 1 \bmod 2N$. For CRYSTALS-Kyber [26] with $q \equiv 1 \bmod N$, the 256-

point incomplete NTT/INTT can be computed by two separate 128-point complete NTT/INTTs with the same CT and GS butterflies for $q \equiv 1 \bmod 2N$, but not the point-wise multiplication. To support point-wise multiplication for NTT/INTT with $q \equiv 1 \bmod N$, the PE structures proposed in [6], [27], for incomplete NTT/INTT can be adopted.

IV. Proofs of Memory and RAW Conflict-Free Access Patterns for Scalable NTT Pipeline

In this section, we prove that the memory access patterns proposed in Section III is free from memory and RAW conflicts for any valid N, n_{PE} and d_{PE}. Data flow for scalable conflict-free pipeline is also discussed.

A. Memory Conflict-free Proof

Memory conflict is avoided because the arithmetic progression generated in each round of Algorithm 3 are partitioned into $b = 2 \times w_{PE} = 2 \times \frac{n_{PE}}{d_{PE}}$ different banks by (2). This can be formally proved by the correctness of Lemmas 1 and 2.

Lemma 1: The binary representations of the b indices generated by Algorithm 3 in each round differ only in a segment of $l_b = log_2 b$ contiguous bits.

Proof. The b indices generated in round r of Phase 2 are the same as those generated in round r in the last stage of Phase 1 by Algorithm 3. If the indices generated in each round of Phase 1 are partitioned into $b = 2 \times w_{PE}$ different banks, the indices in each round of Phase 2 are also partitioned into b different banks. Hence, it is sufficient to prove Lemma 1 based on only the b indices generated in each round of Phase 1.

Let the b indices in round r of Stage s, $idx_l(s,r,0,0), \cdots, idx_l(s,r,0,u), \cdots, idx_h(s,r,0,n_{PE}-1)$ be denoted by $a_{s,r}[0], \cdots, a_{s,r}[t] \cdots, a_{s,r}[b-1]$, where $a_{s,r}[t], t \in [0,b)$ is the t-th index of round r in Stage s. The number series $a_{s,r}$ is an arithmetic progression with a common difference of 2^s. According to Step 9 of Algorithm 3, the first term $a_{s,r}[0]$, can be expressed as:

$$a_{s,r}[0] = i + j \cdot 2w_{PE} \cdot m = i + j \cdot 2^s \cdot b. \tag{4}$$

where $i = \left\lfloor r / \frac{N}{2w_{PE} \cdot m} \right\rfloor \in [0, 2^s)$ and $j = r \bmod \frac{N}{2w_{PE} \cdot m} \in [0, \frac{N}{2w_{PE} \cdot m}) = [0, \frac{N}{2^s \cdot b})$. Let $L' = log_2 N$ and $l_b = log_2 b$. In binary representation, the integer $j \cdot 2^s \cdot b = j \cdot 2^{s+l_b}$ is the integer j shifted left by s + l_b bits. Hence the least significant $s + l_b$ bits of the binary representation of $j \cdot 2^s \cdot b$ are all "0". Also, $i \in [0, 2^s)$ can be expressed in binary representation with only s bits. Therefore, the binary strings of the integers $j \cdot 2^s \cdot b$, i and $a_{s,r}[0]$ are given by:

$$\begin{aligned} j \cdot 2^s \cdot b &= d_{L'-1} \cdots d_{s+l_b} 0 \cdots 0 \cdots\cdots\cdots 0. \\ i &= 00 \cdots\cdots\cdots 000 \cdots 0 d_{s-1} \cdots\ d_0. \\ a_{s,r}[0] &= d_{L'-1} \cdots d_{s+l_b} 0 \cdots 0 d_{s-1} \cdots\ d_0. \end{aligned} \tag{5}$$

where $d_{L'-1}$ is the most significant bit (MSB) of $j \cdot 2^s \cdot b$ and $a_{s,r}[0]$, and d_{s-1} is the MSB of i.

From (5), it is evident that there is a segment of $l_b = log_2 b$ consecutive zero bits in $a_{s,r}[0]$, i.e., $d_{s+l_b-1} \cdots d_s$ of $a_{s,r}[0]$ are all "0". For any $t \neq 0$, $a_{s,r}[t] = a_{s,r}[0] + t \times 2^s$. Its difference from $a_{s,r}[0]$ is $t \times 2^s$, which is the integer t shifted left by s bits. $t \in [0, b-1]$ can be expressed as $t_{l_b-1} \cdots t_0$ in binary with l_b bits. Since $d_{s+l_b-1} \cdots d_s$ of $a_{s,r}[0]$ are all "0", adding a s-bit left-shifted binary string of integer t to $a_{s,r}[0]$ will not introduce any carry propagation. The binary representations of the arithmetic progression with common difference 2^s, $a_{s,r}[t]$ for $t \in [0, b-1]$, can be expressed as:

$$\begin{aligned} a_{s,r}[0] &= d_{L'-1} \cdots d_{s+l_b} 00 \cdots 00 d_{s-1} \cdots d_0. \\ a_{s,r}[1] &= a[0] + 2^s \\ &= d_{L'-1} \cdots d_{s+l_b} 00 \cdots 01 d_{s-1} \cdots d_0. \\ a_{s,r}[2] &= a[0] + 2 \cdot 2^s \\ &= d_{L'-1} \cdots d_{s+l_b} 00 \cdots 10 d_{s-1} \cdots d_0. \\ &\cdots \\ a_{s,r}[b-1] &= a[0] + (b-1) \cdot 2^s \\ &= d_{L'-1} \cdots d_{s+l_b} 11 \cdots 11 d_{s-1} \cdots d_0. \end{aligned} \tag{6}$$

From (6), it is clear that these b integer indices differ only in the middle l_b bits, i.e., $d_{s+l_b-1} \cdots d_s$. In fact, $t_k = d_{s+k}$ and each coefficient index $a_{s,r}[t]$ can be expressed as:

$$a_{s,r}[t] = d_{L'-1} \cdots d_{s+l_b} t_{l_b-1} \cdots t_k \cdots t_0 d_{s-1} \cdots d_0. \tag{7}$$

This proves that the b indices generated by Algorithm 3 differ only in a segment of l_b contiguous bits. □

Lemma 2: The b indices with bit difference occurring only in a segment of l_b contiguous bits will map to b different banks.

Proof. Equation (2) can be rewritten as

$$\begin{aligned} &\textit{Bank}(a_{s,r}[t]) \\ &= \left(\sum_{i=0}^{\lceil L'/l_b \rceil - 1} d_{i \cdot l_b + l_b - 1} \cdots d_{i \cdot l_b} \right) \bmod b. \end{aligned} \tag{8}$$

where d_i is the i-th bit of $idx = a_{s,r}[t]$ in binary form.

The bit stream of $idx = a_{s,r}[t]$ is first evenly divided into $\lceil L'/l_b \rceil$ disjoint segments each of l_b bits. The *Bank* number of $idx = a_{s,r}[t]$ is the modulo b sum of these $\lceil L'/l_b \rceil$ segments. We will prove that the *Bank* numbers so computed for the b different $idx = a_{s,r}[t]$ are b distinct integer values.

If s is divisible by l_b, t_0 to t_{l_b-1} bits of $a_{s,r}[t]$ in (7) will be one complete segment by itself, i.e., the value of the (s/l_b)-th segment in (8) is t. According to Lemma 1, the b coefficient indices differ only in these l_b bits. Except this (s/l_b)-th segment, all the other $\lceil L'/l_b \rceil - 1$ segments of the b different indices have the same value. Let the sum of all but the (s/l_b)-th segment be c. Then $\textit{Bank}(a_{s,r}[t]) = (c + t) \bmod b = (c \bmod b + t \bmod b) \bmod b$. Since c is constant for all coefficient indices $t < b$ and t is different for different indices $a_{s,r}[t]$, each of the b indices will map to a different bank.

If s does not divide b, the l_b bits $t_{l_b-1} \cdots t_0$ of $a_{s,r}[t]$ in (7) will be divided and fall into two segments, i.e., the $(\lfloor s/l_b \rfloor)$-th and $(\lfloor s/l_b \rfloor + 1)$-th segments in (8). Suppose $l_o \in [1, l_b - 1]$ bits of t, i.e., $t_{l_o-1} \cdots t_0$ fall in the $(\lfloor s/l_b \rfloor)$-th segment, the other $l_b - l_o$ bits of t, i.e., $t_{l_b-1} \cdots t_{l_o}$ fall in the $(\lfloor s/l_b \rfloor + 1)$-th segment. Let the sum of these two segments be t' and

the sum of the remaining $\lceil L'/l_b \rceil - 2$ segments be c'. Then, $Bank(a_{s,r}[t]) = (c' + t') \bmod b$. c' is the same for all the b indices. $t' \bmod b$ for any index can be expressed as:

$$\begin{aligned} t' \bmod b =& (d_{s+l_b+l_o-1} \cdots d_{s+l_b} t_{l_b-1} \cdots t_{l_o} \\ &+ t_{l_o-1} \quad \cdots \quad t_0 \; d_{s-1} \cdots d_{s-(l_b-l_o)}) \bmod b \\ =& (d_{s+l_b+l_o-1} \cdots d_{s+l_b} d_{s-1} \cdots d_{s-(l_b-l_o)} \\ &+ t_{l_o-1} \quad \cdots \quad t_0 \; t_{l_b-1} \quad \cdots \quad t_{l_o}) \bmod b \end{aligned} \tag{9}$$

In the final expression of (9), the first term $d_{s+l_b+l_o-1} \cdots d_{s+l_b} d_{s-1} \cdots d_{s-(l_b-l_o)}$ is the same for all b indices. The second term $t'' = t_{l_o-1} \cdots t_0 \; t_{l_b-1} \cdots t_{l_o}$ has the same but permuted l_b bits as t. In other words, t'' has $2^{l_b} = b$ different integer values in the range $[0, b-1]$ like t. Thus, "$t' \bmod b$" has b different integer values. Consequently, $Bank(a_{s,r}[t]) = (c' + t') \bmod b$ produces b distinct values, one for each term $t \in [0, b-1]$ of the series $a_{s,r}$.

Combining both cases of $s \mid b$ and $s \nmid b$, the b different indices will map to b different banks. □

B. RAW Conflict-free Proof

Let c_{PE} be the number of clock cycles required to complete one butterfly operation. Lemma 3 gives the RAW conflict-free criterion based on d_{PE}, $w_{PE} = \frac{n_{PE}}{d_{PE}}$ and c_{PE}.

Lemma 3: RAW conflicts will not occur if

$$d_{PE} \cdot c_{PE} \leqslant \frac{N}{2^{d_{PE}} \cdot 2w_{PE}}. \tag{10}$$

Proof. Each coefficient of a butterfly is read from the RAM to the PE array. After $d_{PE} \cdot c_{PE}$ cycles, the result of its computation is written back to the RAM. Let Δ denote the time difference in number of clock cycles between two RAM accesses for fetching the coefficients to the PE array in Stages s and $s+d_{PE}$. If a coefficient is accessed in round r of Stage s, and then accessed in round r' of Stage $s + d_{PE}$, $\Delta = [\frac{N}{2w_{PE}} - r] + r'$. To avoid RAW conflict, $d_{PE} \cdot c_{PE} \leqslant \Delta$.

Due to the different methods of generating the indices in Phases 1 and 2 of Algorithm 3, there are three possible conditions. When $s + d_{PE} \leqslant log_2(\frac{N}{2w_{PE}})$, Stages s and $s + d_{PE}$ are in Phase 1. When $s < log_2(\frac{N}{2w_{PE}}) < s + d_{PE}$, Stage s is in Phase 1 and Stage $s+d_{PE}$ is in Phase 2. When $s > log_2(\frac{N}{2w_{PE}})$, both Stages s and $s + d_{PE}$ are in Phase 2. In Phase 1, the indices generated in round r of Stage s form an arithmetic progression with a common difference of 2^s, starting from $\lfloor \frac{r \cdot T}{N} \rfloor + (r \bmod \frac{N}{T}) \cdot T$, where $T = 2w_{PE} \cdot 2^s$. In Phase 2, the indices in round r of Stage s form an arithmetic progression with a common difference of $\frac{N}{2w_{PE}}$, starting from r. Therefore, the minimum time difference, Δ_{min}, can be expressed as:

$$\Delta_{min} = \min\left(\frac{N}{2w_{PE}} - r + r'\right). \tag{11}$$

When $s + d_{PE} \leqslant log_2\left(\frac{N}{2w_{PE}}\right)$, we have

$$\begin{aligned} \left\lfloor \frac{r \cdot T}{N} \right\rfloor + \left(r \bmod \frac{N}{T}\right) \cdot T + t \cdot 2^s = \left\lfloor \frac{r' \cdot 2^{d_{PE}} \cdot T}{N} \right\rfloor \\ + \left(r' \bmod \frac{N}{2^{d_{PE}} \cdot T}\right) \cdot 2^{d_{PE}} \cdot T + t' \cdot 2^{d_{PE}+s}. \end{aligned} \tag{12}$$

When $s < log_2\left(\frac{N}{2w_{PE}}\right) < s + d_{PE}$, we have

$$\left\lfloor \frac{r \cdot T}{N} \right\rfloor + \left(r \bmod \frac{N}{T}\right) \cdot T + t \cdot 2^s = r' + t' \cdot \frac{N}{2w_{PE}}. \tag{13}$$

When $s > log_2\left(\frac{N}{2w_{PE}}\right)$, we have

$$r + t \cdot \frac{N}{2w_{PE}} = r' + t' \cdot \frac{N}{2w_{PE}}. \tag{14}$$

In (11), $r, r' \in [0, \frac{N}{2w_{PE}}), t, t' \in [0, 2w_{PE})$ and $s \in [0, log_2 N - d_{PE}]$. From (12) to (14), $\Delta_{min} = \frac{N}{2^{d_{PE}} \cdot 2w_{PE}}$. Therefore, when $d_{PE} \cdot c_{PE}$ is not more than $\frac{N}{2^{d_{PE}} \cdot 2w_{PE}}$, our proposed access pattern is free from RAW conflicts, and no pipeline stall needs to be added. When $d_{PE} \cdot c_{PE}$ exceeds $\frac{N}{2^{d_{PE}} \cdot 2w_{PE}}$, pipeline stall is needed to avoid RAW conflict. □

C. Scalable Conflict-free Pipeline

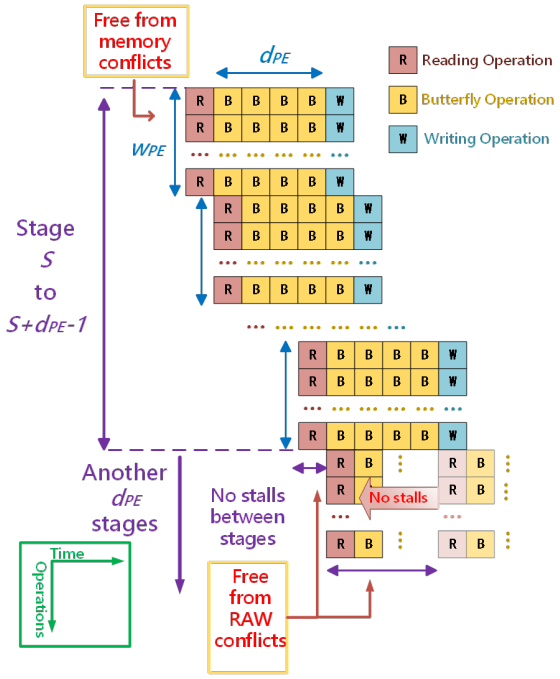


Fig. 8: The scalable NTT pipeline of our design with n_{PE}, d_{PE} and $w_{PE} = \frac{n_{PE}}{d_{PE}}$.

Fig. 8 shows the scalable pipeline of our design. In Fig. 8, the blocks marked R, B and W represent the read, butterfly and write operations, respectively. The $w_{PE} \times d_{PE}$ PE array reads w_{PE} pairs of coefficients in each round of Stage s, and performs $w_{PE} \times d_{PE}$ butterfly operations for every d_{PE} stages before the results are written back to the RAM. Because our memory access pattern is free from memory conflicts, the $2w_{PE}$ coefficients used in each round can be accessed concurrently. If $2 \cdot c_{PE} \cdot 2^{d_{PE}} \cdot n_{PE} \leqslant N$, our proposed memory access pattern can avoid RAW conflicts without adding stalls between stages. Therefore, our proposed NTT accelerator design methodology is scalable to any possible NTT and PE array configuration parameters with the minimum latency of $\frac{N \cdot log_2 N}{2n_{PE}}$ clock cycles after filling up the pipeline of the butterfly.

V. Parameterized NTT Accelerator Synthesizer and Performance Evaluation

Based on our proposed scalable NTT design methodology, an automated synthesizer for parameterized NTT accelerators is proposed. Our proposed NTT generator takes as inputs the security parameters (N, q) as well as the PE array configuration, i.e., n_{PE} and d_{PE} at design time to generate a synthesizable Verilog HDL code of an efficient fully pipelined NTT hardware accelerator. For the purpose of performance evaluation and comparison, NTT-based polynomial multipliers of different parameter settings of $N = 4096, 1024, 512, 256, 128$, $\lceil log_2 q \rceil = 13, 14, 16, 24$, $n_{PE} = 1, 2, 4, 8, 16, 32$, $d_{PE} = 1, 2$ are generated by our proposed synthesizer. Each butterfly operation can be completed in $c_{PE} = 7$ cycles in our implementation. The synthesis results of these polynomial multipliers are obtained from the Xilinx Vivado 2018.1 tool by selecting the Xilinx VIRTEX-7 FPGA (xc7vx690tffg1761-2) as the FPGA device. The coefficients and twiddle factors in Fig. 2 are stored on the block RAMs (BRAMs) of the FPGA.

A. Performance and Area Evaluation

1) Clock Cycles: We measure the number of clock cycles of our generated NTT accelerators with different parameters and compare them with the ideal cycles and the results reported by [8]. The results are presented in Table I.

TABLE I: Comparison of latencies in clock cycles

NTT of N =1024, $\lceil log_2 q \rceil = 29$					
n_{PE}, d_{PE}	$2 \cdot c_{PE} \cdot 2^{d_{PE}} \cdot n_{PE}$	Ideal	Our	[8]	(([8] - Our)/([8])) x100%
1,1	28	5120	5127	5210	1.59%
2,1[1]	56	2560	2567	2650	3.13%
4,1*	112	1280	1287	1370	6.06%
8,1	224	640	647	730	11.37%
16,1*	448	320	327	410	20.24%
32,1	896	160	167	250	33.20%
8,2**	448	640	654	–	–
16,2**	896	320	334	–	–
NTT of N =256, $\lceil log_2 q \rceil = 23$					
n_{PE}, d_{PE}	$2 \cdot c_{PE} \cdot 2^{d_{PE}} \cdot n_{PE}$	Ideal	Our	[8]	(([8] - Our)/([8])) x100%
1,1	28	1024	1031	1096	5.93%
2,1*	56	512	519	584	11.13%
4,1*	112	256	263	328	19.80%
8,1*	224	128	135	200	32.50%
4,2**	224	256	270	–	–

[*] The number of cycles for [8] are estimated based on their design description, the corresponding configuration of n_{PE} is not given in their experiments.
[**] Multi-layer PE array, i.e., $d_{PE} > 1$, is not supported by [8].

The first column of Table I lists n_{PE} and d_{PE} (delimited by comma) of the design. To validate Lemma 3, the product $2 \cdot c_{PE} \cdot 2^{d_{PE}} \cdot n_{PE}$ of our design is listed in the second column. The ideal clock cycles of NTT is $\frac{N \cdot log_2 N}{2 n_{PE}}$. As shown in Table I, the clock cycles required by our NTT accelerators with different parameter settings are identical to the ideal clock cycles, with $c_{PE} \times d_{PE} = 7$ or 14 extra cycles incurred in filling up the pipelined butterfly. The results agree with our conflict-free proof in Section IV-A. When $2 \cdot c_{PE} \cdot 2^{d_{PE}} \cdot n_{PE} \leqslant N$, our design can avoid RAW conflicts without introducing any idle cycles between NTT stages.

NTT accelerators with different N, q, n_{PE} and fixed $d_{PE} = 1$ were also proposed and evaluated in [8]. To provide a more complete comparison that includes the performance of accelerators with multiple PEs, for those configurations that were not evaluated in [8], the clock cycles are estimated based on their design descriptions. Compared with [8], the NTT hardware accelerators synthesized by our proposed method have lower cycles for all configurations in comparison. The performance improvement of our NTT hardware is more prominent as n_{PE} increases. The empirical designs of [8] cannot guarantee conflict-free memory access. To avoid potential RAW conflicts, pipelined stalls were added between stages in [8].

2) Latency and Area: The actual performance of a circuit depends on both the clock cycles and clock frequency. We also evaluate the area-time complexity of our proposed designs based on the actual time required to complete one NTT calculation and the area of the NTT accelerator.

a) Latency and Area for different n_{PE} and d_{PE}: Fig. 9 presents the results of a case study of implementing the lattice-based scheme Falcon of $N = 1024$ and $\lceil log_2 q \rceil = 14$ with different hardware accelerator configurations of $n_{PE} = 1, 2, 4, \cdots, 32$ and $d_{PE} = 1, 2$. In Fig. 9, the horizontal axis is the latency of one NTT calculation, the vertical axis is the area of the hardware in terms of the number LUTs and FFs consumed. The blue line denotes the case of $d_{PE} = 1$, the orange line represents the case of $d_{PE} = 2$.

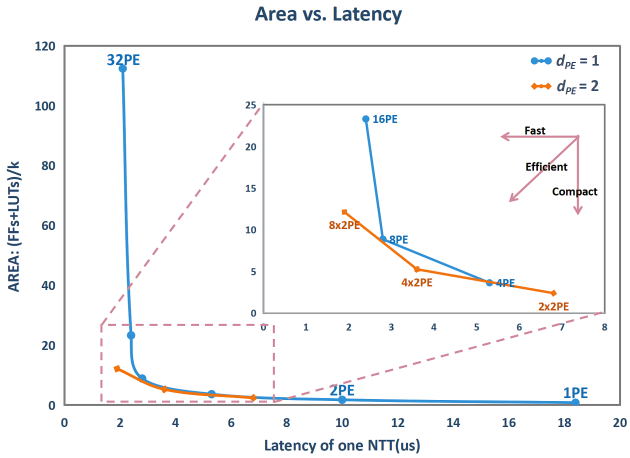


Fig. 9: Area vs. Latency for $N = 1024$ and $\lceil log_2 q \rceil = 14$ NTT implemented with different n_{PE} and d_{PE}.

From Fig. 9, with only one PE, the NTT hardware accelerator designed by our method is very compact and consumes less than 1000 LUTs and FFs. This hardware configuration takes about 18.4 µs to compute one NTT operation. The performance can be easily raised by utilizing more PEs at the cost of higher area consumption. As shown in Fig. 9, when 32 PEs are executed in parallel, the latency can be reduced by ten times down to 2 µs.

Fig. 9 also shows that the area cost of the NTT-based polynomial multiplier can be reduced by increasing d_{PE}. For example, for n_{PE} = 16, the area of the design with $d_{PE} = 2$ is reduced by 48.07% compared with the design with $d_{PE} = 1$.

The area reduction of a two-layer design over the one-layer design is accomplished with a slight increase in latency. This slight latency increase is attributed to the higher fanout in the critical paths of the address generator.

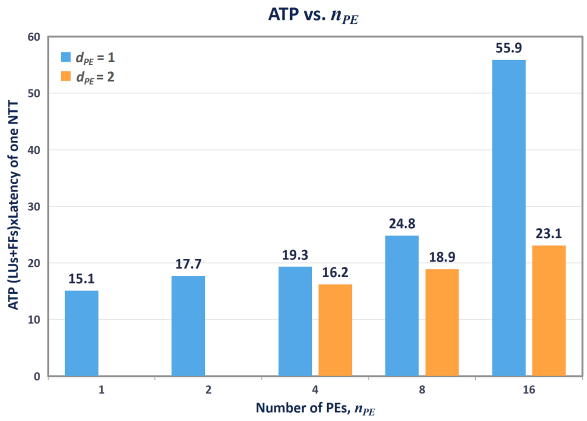


Fig. 10: ATPs for $N = 1024$, $\lceil log_2 q \rceil = 14$ with different n_{PE} and d_{PE}.

b) ATP for different n_{PE} and d_{PE}: Fig. 10 plots the area time products (ATPs) of our synthesized NTT accelerators with $n_{PE} = 1, 2, 4, 8, 16$ and $d_{PE} = 1, 2$, where the ATP of each design is calculated by the product of its Area (LUTs + FFs) and Latency (µs). The blue and orange bars corresponding to $d_{PE} = 1$ and $d_{PE} = 2$, respectively. The results indicate that the increase in ATP with n_{PE} of one layer design is not appreciably large when n_{PE} is small. However, the ATP shoots up when n_{PE} increases to 16. This is because as the size of the MUXs and the index calculators increases, the critical path of the NTT accelerator design also increases.

Another finding is the ATP can be reduced significantly by increasing d_{PE}, especially for large n_{PE}. For $n_{PE} = 16$, a two-layer design reduces the ATP by more than half from the one-layer design. This is because when d_{PE} increases, the number of used coefficients, $2w_{PE} = \frac{2n_{PE}}{d_{PE}}$ decreases, which in turn reduces the size of the MUX groups and the index calculators.

Based on the above experimental results, it is evident that the full scalability of our proposed NTT accelerator design methodology enables hardware design space exploration through n_{PE} and d_{PE} to achieve the desired cost-performance trade-offs for the target security paramters, N and q.

c) Area usage of sub-modules for different parameters: Table II shows the proportions of LUTs/FFs of different modules of NTT designs with different N and q. CT is the control circuits for the scheduled execution of NTT, point-wise multiplication and INTT. IDX_G and ADDR_G are the index generator and address generator for the index generation Algorithm 3 and address assignment (see (2)), respectively. TW_AC is the logic circuit for accessing the twiddle factors. The twiddle factors for NTT and INTT are stored in one to three BRAM tiles depending on the combination of N, q, n_{PE} and d_{PE}, and the size of the BRAM tiles allocated by the compiler. According to Algorithm 3, the maximum number of twiddle factors required in each cycle of NTT/INTT calculation for a PE array with d_{PE} layers and w_{PE} PEs on each layer is $max\,(n_{tw}) = \sum_{i=0}^{d_{PE}-1} (\frac{1}{2})^i \cdot w_{PE}$. The width of the RAM is $max\,(n_{tw}) \cdot \lceil \log_2 q \rceil = (\sum_{i=0}^{d_{PE}-1} (\frac{1}{2})^i \cdot w_{PE}) \cdot \lceil log_2 q \rceil$ to ensure that the n_{tw} twiddle factors used by the PE array can be accessed simultaneously in one cycle. The size of each single-port RAM in number of bits required to store the NTT or INTT twiddle factors of length $\lceil log_2 q \rceil$ works out to be $\left\lceil \frac{N-1}{max(n_{tw})} \right\rceil \cdot max\,(n_{tw}) \cdot \lceil log_2 q \rceil$. As an example, for $N = 1024$, $w_{PE} = 2^n$ and $d_{PE} = 1$, the BRAM size is $1024 \cdot \lceil log(q) \rceil$.

MUXs are the most costly logic besides the PE array for different N and q. The proportion of LUT usage on MUXs increases with increasing n_{PE} for the same d_{PE}. For example, with 1×8 PEs, it exceeds 50% for $N = 256$, 24-bit q and $N = 4096$, 24-bit q. The LUT proportion on MUXs reduces significantly as d_{PE} increases. For the same n_{PE}, the proportion of LUT cost on MUXs for a design with a larger d_{PE} is much smaller than that with a smaller d_{PE}. For example, the proportion of LUTs on MUXs for the design with 2×4 PEs is almost half that of the design with 1×8 PEs.

TABLE II: Proportions of resource usage (%) of sub-modules

PE	MUXs (%)	TW_AC (%)	IDX_G (%)	ADDR_G (%)	CT%	PE ARRAY (%)	Others (%)
N = 4096, q = 24 bits							
	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF
1×1	23.94; 16.38	0.0; 9.14	16.01; 10.29	4.01; 4.57	10.20; 3.24	44.02; 51.43	1.82; 4.95
2×2	20.82; 18.58	8.24; 11.58	8.19; 1.88	4.47; 5.15	11.93; 0.77	45.91; 61.47	0.43; 0.57
1×8	51.51; 15.85	7.37; 5.34	8.31; 3.19	9.52; 2.89	3.94; 1.23	18.24; 70.76	0.11; 0.74
2×4	27.19; 16.28	10.01; 5.14	6.09; 1.66	6.09; 2.95	11.91; 1.35	40.46; 72.27	0.23; 0.35
N = 1024, q = 14 bits							
	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF
1×1	28.33; 18.24	0.0; 8.51	22.58; 13.98	5.07; 6.08	15.71; 3.95	25.52; 42.55	2.80; 6.69
2×2	19.91; 22.02	10.03; 11.83	13.06; 2.56	6.12; 7.19	7.26; 2.56	42.92; 53.08	0.68; 0.76
1×8	43;06; 17.72	6.95; 5.80	11.49; 4.64	12.03; 4.11	11.67; 1.69	14;69; 64.98	0.12; 1.05
2×4	24.32; 18.74	9.01; 5.76	9.11; 2.29	5.65; 4.31	14.45; 1.06	37.17; 67.34	0.29; 0.50
N = 256, q = 24 bits							
	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF
1×1	24.20; 15.58	0.0; 10.11	12.73; 7.79	3.22; 3.37	11.09; 2.53	45.51; 56.84	3.25; 3.79
2×2	23.10; 17.33	5.12; 5.80	4.02; 1.13	3.11; 3.55	10.02; 1.34	54.20; 64.42	0.41; 0.36
1×8	59.16; 15.20	8.01; 5.30	4.31; 2.17	0.61; 2.04	8.35; 1.21	19.43; 73.56	0.12; 1.05
2×4	23.61; 15.29	10.03; 5.00	4.64; 1.01	2.57; 1.93	24.39; 0.57	34.65; 75.97	0.10; 0.50
N = 256, q = 13 bits							
	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF	LUT; FF
1*1	21.58; 17.87	0.0; 8.93	19.49; 12.71	4.87; 5.50	12.99; 4.12	38.52; 44.67	2.55; 6.19
2*2	15.19; 21.96	5.37; 12.01	6.18; 2.11	5.26; 6.33	22.41; 0.70	44.86; 56.25	0.73; 0.65
1*8	47.71; 18.01	7.51; 6.00	8.14; 4.08	10.10; 1.93	4.25; 1.56	20.82; 65.55	1.48; 0.96
2*4	25.36; 19.43	9.18; 8.30	8.35; 3.22	5.06; 7.49	11.24; 5.49	40.62; 55.47	0.20; 0.61

B. Comparisons with Related work

We compared the key results of our synthesized NTT-based polynomial multipliers with existing works in Table III. The second column of Table III shows the scalable parameters of each design. The detailed resource usages and performances are also listed in this table. The last column of the table listed their ATPs. In Table III, the designs of [5], [28], [29] have fixed q, n_{PE} and d_{PE}. Hence only results for different N are provided. The designs of [20], [18], [3], [21] have fixed n_{PE} and d_{PE}, and results for different N and q are provided. Results are provided for different N, q and n_{PE} with fixed d_{PE} for the design of [8] and [30]. Our design is more scalable and all four parameters N, q, n_{PE} and d_{PE} can be changed to optimize the accelerator's area and performance. We first compare with the state-of-the-art NTT hardware accelerator designs tailored and optimized for one specific scheme or parameter set [5], [28], [29]. As shown in Table III, all NTT accelerators synthesized by our design methodology can complete a polynomial multiplication within the shortest

TABLE III: Evaluation results of our NTT hardware and comparisons with prior works

Work	Tunable (N, q, n_{PE}, d_{PE})	Platform	N	$\lceil log_2 q \rceil$	PE Array Shape	LUT	FF	BRAM	DSP	Clock Cycles (operation)	Freq. (MHz)	Latency μs (operation)	ATP (LUT+FF)×Time
[5]	√,–,–,–	Artix-7	1024	14	2PE	847	375	6	2	2569 (NTT)	244	10.5 (NTT)	12.8
			512	14	2PE	741	330	5	2	1289 (NTT)	245	5.3 (NTT)	5.7
[28]	√,–,–,–	40nm CMOS	1024	14	1PE	–	–	–	–	6155 (NTT)	72	85.4 (NTT)	–
[20]	√,√,–,–	Zynq-7000	1024	14	1PE	980	395	2	26	10240 (NTT)	–	–	–
			512	14	1PE	980	395	2	26	4608 (NTT)	–	–	–
[18]	√,√,–,–	V6LX75T	512	14	1PE	994	994	3	1	3443 (NTT) / 4775 (INTT)	278	14.8 (AVG)	29.4
			256	13	1PE	807	851	2	1	1691 (NTT) / 2328 (INTT)	313	6.4 (AVG)	10.6
[3]	√,√,–,–	RTL Zynq-7000	1024	14	2x2PE	898	1117	10	4	2032 (NTT)	188	10.8 (NTT)	21.7
		HLS Zynq-7000	1024	14	2x2PE	1521	2695	10	4	2032 (NTT)	180	11.3 (NTT)	47.6
[29]	√,–,–,–	SPARTAN-6	1024	24	2PE	6689	–	8	4	7967 (PolyMul)	241	33.1 (PolyMul)	221.4 (PolyMul)
			512	24	2PE	3750	–	4	4	3622 (PolyMul)	254	14.3 (PolyMul)	23.52 (PolyMul)
			256	24	2PE	2829	–	4	4	1618 (PolyMul)	258	6.3 (PolyMul)	17.7 (PolyMul)
[21]	√,√,–,–	Artix-7	1024	14	1PE	944	467	3	3	11455 (PolyMul)	141	81.2 (PolyMul)	114.6 (PolyMul)
[8]	√,√,√,–	Virtex-7	1024	14	1PE	575	–	11	3	5160 (NTT)	125	41.3 (NTT)	23.7*
			1024	14	8PE	2584	–	16	24	680 (NTT)	125	5.4 (NTT)	14.0*
			1024	14	32PE	17188	–	48	96	200 (NTT)	125	1.6 (NTT)	27.5*
[30]	√,√,√,–	Virtex-7	256	13	1PE	2128	1144	3	8	1052 (NTT) 1314 (INTT)	174	6.8 (AVG)	22.2
			256	13	8PE	11k	5422	12	64	156 (NTT) 197 (INTT)	186	0.9 (AVG)	14.7
			256	13	32PE	61k	17k	48	256	95 (NTT) 112 (INTT)	167	0.6 (AVG)	46.8
Our work	√,√,√,√	Virtex-7	1024	14	1PE	515	306	3	3	5127 (NTT), 11283 (PolyMul)	278	18.4 (NTT), 40.6 (PolyMul)	9.7*
					2PE	1205	572	4	6	2567 (NTT), 5651 (PolyMul)	256	10.0 (NTT), 22.1 (PolyMul)	12.1*
					2x2PE	1467	930	4.5	13	1294 (NTT), 2849 (PolyMul)	189	6.9 (NTT), 15.1 (PolyMul)	10.1*
					4x2PE	3472	1789	7.5	25	654 (NTT), 1441 (PolyMul)	179	3.7 (NTT), 8.1 (PolyMul)	12.8*
					8x2PE	8515	3618	12	49	334 (NTT), 737 (PolyMul)	172	1.9 (NTT), 4.3 (PolyMul)	17.0*
			512	14	1PE	489	245	3	3	2311 (NTT), 5139 (PolyMul)	278	8.3 (NTT), 18.5 (PolyMul)	6.09
					2PE	1071	557	4	6	1159 (NTT), 2579 (PolyMul)	248	4.7 (NTT), 10.4 (PolyMul)	7.48
					16PE	28616	4211	24	48	151 (NTT), 339 (PolyMul)	154	1.0 (NTT), 2.2 (PolyMul)	32
			256	13	1PE	449	271	3	3	1031 (NTT), 2323 (PolyMul)	286	3.6 (NTT), 8.1 (PolyMul)	2.59
					2x2PE	1288	888	4.5	12	270 (NTT), 609 (PolyMul)	278	1.0 (NTT), 2.2 (PolyMul)	2.1
					8PE	6245	1864	12	24	135 (NTT), 307 (PolyMul)	256	0.5 (NTT), 1.2 (PolyMul)	4.27
			256	24	1PE	691	451	3	5	1031 (NTT), 2323 (PolyMul)	187	5.5 (NTT), 12.3 (PolyMul)	6.24
					2x2PE	2466	1637	4.5	20	270 (NTT), 609 (PolyMul)	175	1.5 (NTT), 3.4 (PolyMul)	6.3
			128	16	1PE	477	293	3	3	455 (NTT), 1043 (PolyMul)	250	1.8 (NTT), 4.1 (PolyMul)	1.3
					2PE	1056	574	4	6	231 (NTT), 531 (PolyMul)	228	1.0 (NTT), 2.3 (PolyMul)	1.6
			4096	24	1PE	802	525	7	4	24583 (NTT), 53267 (PolyMul)	185	132.9 (NTT), 287.9 (PolyMul)	17.6
					4x2PE	5665	3188	8.5	33	3079 (NTT), 6675 (PolyMul)	157	19.7 (NTT), 42.6 (PolyMul)	19.2
					8x2PE	14591	6468	12	80	1543 (NTT), 3347 (PolyMul)	121	12.8 (NTT), 27.7 (PolyMul)	26.9

* The ATP is computed based on LUT×Time according to [8] for fair comparison.

NTT and PolyMul in brackets next to the numerical values refer to the implementation of a single NTT operator and a complete NTT-based polynomial multiplication, respectively. PolyMul consists of one NTT (for [21] and ours) or two (for [29]) NTTs, one point-wise multiplication and one INTT

time using the same number of PEs. NTT accelerator design tailored for a fixed set of security parameter or PE number and configuration is easier to optimize than parameterizable hardware. Nevertheless, all our designs can achieve better performance for the same parameter set with better ATP than most fixed parameter designs in comparison except for the NTT accelerators of [5] and the case of $(N, \lceil log_2 q \rceil) = (256, 24)$ of [29]. The latter two works have smaller but comparable ATP as ours. This is because they are designed for fixed q, which allows customized modulus-specific optimizations. Such optimizations are not generalizable for other q, which limits their scalability. It is worth noting that BRAM usage was not taken into consideration in ATP computation. In fact, our NTT accelerators have the smallest BRAM usage among all the non-scalable designs in comparison.

Compared with [20], [18], [3], [21] that are scalable in N and q, our designs can complete an NTT operation and a complete polynomial multiplication within the shortest time using the same number of PEs and are therefore faster than them. Our NTT accelerator designs for the same N and q also have lower area cost. Hence, the ATPs of our designs are substantially lower and the average ATP reduction is 71.4%. In [8], a scalable NTT operator is proposed. Different n_{PE} are used in the experimental results of [8]. However, the accelerator architecture design based on their NTT operations is not fully pipelinable and more cycles are required to complete one NTT operation as discussed earlier in the comparison of clock cycles in Table I. In Table III, for the same number of PEs, the NTT accelerators designed by our method achieve better performance in all their evaluated parameter settings. Our accelerators are 1.6x to 2x faster than theirs. It should be noted that the design method of [8] focuses only on the scalability problem of an NTT operator whereas our method considers the design of a complete NTT-based polynomial multiplication which includes one point-wise polynomial multiplication besides one NTT and one INTT operation. Therefore, two additional RAMs, associated glue logic circuits and a function controller for each PE are needed in our design. Nevertheless, the number of LUTs and BRAMs required by our design is comparable and in many cases much smaller than those of [8] for the same parameter sets, particularly for the BRAM usage. For the same N and q, our best design option is

30.7% more area-time efficient than their best design option. Comparing with CoHA-NTT [30], a configurable hardware accelerator that supports incomplete NTT-based polynomial multiplication, our design with the same n_{PE} is still faster and consumes much less resources.

VI. Conclusion

In this paper, a unified architecture is proposed for the design of area and performance optimized NTT accelerators scalable to polynomial degree N and modulus q as well as number n_{PE} and depth d_{PE} of processing element array. The address index generator of the proposed unified architecture assures conflict-free memory access pattern for NTT computations without pipeline stall. The proposed scheme is theoretically underpinned by the proof of conflict-free memory access and criterion for conflict-free RAW. Based on the proposed memory access pattern, a parameterized NTT-based polynomial multiplier generator is modelled in Verilog HDL to facilitate physical implementation for efficient performance evaluation and comparison with existing NTT accelerator designs with different parametric configurations. Experimental results show that our generator can also produce area-time optimized NTT accelerators tailored to different application requirements. The latency and area-time performance of the solutions synthesized by our methodology are comparable to custom NTT-based polynomial multipliers optimized for fixed parameters and outshine existing scalable NTT accelerators with lower parametric scalability.

VII. Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments. The authors also thank Huajie Tan for the help in collecting experimental results. This work is supported in part by the National Key Research and Development Program of China under grant No.2020YFB1600201, National Natural Science Foundation of China (NSFC) under grant No.(U20A20202, 62090024, 61876173), and Youth Innovation Promotion Association CAS.

References

[1] National Institute of Standards and Technology (NIST), "Post-Quantum Cryptography Standardization," https://csrc.nist.gov/projects/post-quantum-cryptography, 2021, accessed: 2022-01-10.

[2] M. Albrecht et al., *Homomorphic Encryption Standard*. Cham: Springer International Publishing, 2021, pp. 31–62.

[3] D. T. Nguyen, V. B. Dang, and K. Gaj, "A high-level synthesis approach to the software/hardware codesign of NTT-based post-quantum cryptography algorithms," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 371–374.

[4] X. Chen et al., "CFNTT: Scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, p. 94–126, Nov. 2021.

[5] N. Zhang et al., "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 2, p. 49–72, Mar. 2020.

[6] Y. Xing and S. Li, "A compact hardware implementation of cca-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 2, p. 328–356, Feb. 2021.

[7] A. C. Mert et al., "A flexible and scalable NTT hardware : Applications from homomorphically encrypted deep learning to post-quantum cryptography," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 346–351.

[8] A. C. Mert et al., "An extensive study of flexible design methods for the number theoretic transform," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[9] V. Lyubashevsky et al., "CRYSTALS-DILITHIUM – Submission to Round 3 of NIST's Post-Quantum Cryptography Standardization Process," National Institute of Standards and Technology, Tech. Rep., 2020, https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/Dilithium-Round3.zip.

[10] P.-A. Fouque et al., "Falcon: Fast-fourier lattice-based compact signatures over NTRU," *Submission to the NIST's Post-Quantum Cryptography Standardization Process*, 2019.

[11] E. Alkim, Y. Alper Bilgin, M. Cenk, and F. Gérard, "Cortex-m4 optimizations for {R,M} LWE schemes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, p. 336–357, Jun. 2020.

[12] L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of kyber on cortex-m4," in *Progress in Cryptology – AFRICACRYPT 2019*, 2019, pp. 209–228.

[13] T. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe, "Software speed records for lattice-based signatures," in *Post-Quantum Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 67–82.

[14] A. Pedram, J. McCalpin, and A. Gerstlauer, "Transforming a linear algebra core to an FFT accelerator," in *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, 2013, pp. 175–184.

[15] L. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 5, pp. 312–316, 1992.

[16] H.-F. Lo, M.-D. Shieh, and C.-M. Wu, "Design of an efficient FFT processor for dab system," in *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems*, vol. 4, 2001, pp. 654–657 vol. 4.

[17] W. Wang, X. Huang, N. Emmart, and C. Weems, "VLSI design of a large-number multiplier for fully homomorphic encryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 1879–1887, 2014.

[18] S. S. Roy et al., "Compact Ring-LWE cryptoprocessor," in *Cryptographic Hardware and Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 371–391.

[19] S. S. Roy et al., "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 387–398.

[20] T. Fritzmann and J. Sepúlveda, "Efficient and flexible low-power NTT for lattice-based cryptography," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 141–150.

[21] W. Wang et al., "Parameterized hardware accelerators for lattice-based cryptography and their application to the hw/sw co-design of qTESLA," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, p. 269–306, Jun. 2020.

[22] D. Reisis and N. Vlassopoulos, "Conflict-free parallel memory accessing techniques for FFT architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3438–3447, 2008.

[23] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2290–2302, 2011.

[24] K.-F. Xia et al., "A memory-based FFT processor design with generalized efficient conflict-free address schemes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1919–1929, 2017.

[25] T. P. T. Ho and C.-H. Chang, "Towards ideal lattice-based cryptography on asic: A custom implementation of number theoretic transform," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. IEEE, 2018, pp. 1–5.

[26] V. Lyubashevsky et al., "CRYSTALS-KYBER – Submission to Round 3 of NIST's Post-Quantum Cryptography Standardization Process," National Institute of Standards and Technology, Tech. Rep., 2020, https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/Kyber-Round3.zip.

[27] F. Yarman, A. C. Mert, E. Öztürk, and E. Savaş, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC

scheme," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1020–1025.

[28] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 4, p. 17–61, Aug. 2019.

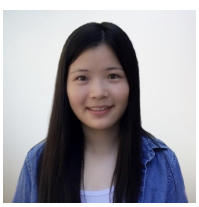
[29] D. D. Chen et al., "High-speed polynomial multiplication architecture for Ring-LWE and SHE cryptosystems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 157–166, 2015.

[30] K. Derya, A. C. Mert, E. Öztürk, and E. Savaş, "CoHA-NTT: A configurable hardware accelerator for NTT-based polynomial multiplication," *Microprocessors and Microsystems*, p. 104451, 2022.

Jianan Mu received his B.S. degree in Electronic Science and Technology from Peking University in 2019. Currently, he works as a Ph.D. candidate in State Key Laboratory of Processors (SKLP), Institute of Computing Technology, Chinese Academy of Sciences. His work research interests include applied cryptography, computer architecture, and hardware security. Email: mujianan19s@ict.ac.cn.

Yi Ren received his B.S. degree in Electronic Science and Technology from Southeast University in 2019 and M.Eng. degree in Engineering of Integrated Circuits from Peking University in 2022. His research interests include hardware security and computer architecture. Email: ren_yi@pku.edu.cn.

Yizhong Hu was graduated from University of Southampton in 2014 with master degree. His main research interests are in the implementation of cryptography. He has worked on secure and efficient implementations of different cryptography silicon IPs. Email: yizhong_hu@hotmail.com



Wen Wang received her B.S. degree from University of Science and Technology of China, M.S. degree and M.Phil. degree, and Ph.D. degree from Yale University in 2015, 2017, and 2021, respectively. Currently, she works as a research scientist in Intel Labs, United States. Her research interests include applied cryptography, computer architecture, and hardware security.

Shuai Chen received his Ph.D. degree from Southeast University in 2021. Also, he has worked in the Computer Architecture and Security Lab. of Yale University as a joint Ph.D. in the year 2017 and 2018. Currently, he works as the co-founder and director of the Rock-solid Security Lab. of Binary Semiconductor Co., Ltd.. His work focuses on research and industrialization of the frontier technology in hardware security and applied cryptography.

Chip-Hong Chang (S'92-M'98-SM'03-F'18) received the B.Eng. (Hons.) degree from the National University of Singapore in 1989, and the M. Eng. and Ph.D. degrees from Nanyang Technological University (NTU) of Singapore in 1993 and 1998, respectively. He is a Professor of the School of Electrical and Electronic Engineering (EEE) of NTU. He held joint appointments with the university as Assistant Chair of Alumni from 2008 to 2014, Deputy Director of the Center for High Performance Embedded Systems from 2000 to 2011, and Program Director of the Center for Integrated Circuits and Systems from 2003 to 2009. He was conferred the Venus International Foundation 2022 Science and Technology Award (VISTA 2022) of Excellence in Hardware Security. He has coedited 6 books, published 13 book chapters, more than 100 international journal papers (more than 80 are in IEEE) and more than 180 refereed international conference papers (mostly in IEEE), and delivered more than 50 keynotes, tutorial and invited seminars. His current research interests include hardware security, AI security, biometric security, trustworthy sensing and hardware accelerators for post-quantum cryptography and edge computational intelligence.

Dr. Chang currently serves as the Senior Area Editor of IEEE Transactions on Information Forensic and Security, and Associate Editor of the IEEE Transactions on Circuits and Systems-I and IEEE Transactions on Very Large Scale Integration (VLSI) Systems. He also served as the Associate Editor of the IEEE Transactions on Information Forensic and Security and IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems from 2016 to 2019, IEEE Access from 2013 to 2019, IEEE Transactions on Circuits and Systems-I from 2010 to 2013, Integration, the VLSI Journal from 2013 to 2015, Springer Journal of Hardware and System Security from 2016 to 2020 and Microelectronics Journal from 2014 to 2020. He guest edited around 10 special issues and served in the organizing and technical program committee of more than 70 international conferences (mostly IEEE). He is a Fellow of the IEEE, IET and AAIA, and the 2018-2019 Distinguished Lecturer of IEEE Circuits and Systems Society.

Junfeng Fan is the founder and CEO of Open Security Research (OSR). He obtained his Ph.D. in 2012 from the COSIC research group of Katholieke Universiteit Leuven (KUL), Belgium. His main research interests are in the area of implementation of cryptography. He has worked on secure and efficient implementations of both PKC and SKC, and participated in the design of EMVCo-certified smart card chips.

Jing Ye is currently an associate professor of State Key Laboratory of Processors (SKLP), Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), and the CEO of CASTEST corporation. He received the Ph.D. degree in ICT, CAS in 2014, and the B.S. degree in Electronics Engineering and Computer Science, Peking University in 2008. His current research interests include VLSI test, hardware security, cryptographic, PUF, PQC, and AI software/hardware security.

Yuan Cao (S'09-M'2014) received his B.S. degree from Nanjing University, M.E. degree from Hong Kong University of Science and Technology and Ph.D. degree from Nanyang Technological University in 2008, 2010 and 2015, respectively. He worked in Advantest from 2015 to 2017, and a Research Fellow in NTU from 2017 to 2018. Currently he is a full professor in the College of IoT of Hohai University. He was a visiting professor to Hamad Bin Khalifa University in December 2019.

Dr. Cao has edited one IET Materials, Circuits and Devices book series 66 entitled "Frontiers in Hardware Security and Trust", and has over 60 peer-reviewed conference and journal publications. Two of his papers has been selected as the finalist of the Best Paper Award for AsianHOST'2017, and AsianHOST'2019. He has served as an organizing/technical committee member in various IEEE conferences (AsianHOST, ASHES, DSP, CTC, et al.). His research interests include TRNG, PUF, PQC and analog/mixed-signal VLSI designs.

Huawei Li (M'00–SM'09) received the B.S. degree in computer science from Xiangtan University, Xiangtan, China, in 1996, and the M.S. and Ph.D. degrees from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1999 and 2001, respectively. She has been a Professor with ICT and CAS since 2008. She was a visiting Professor at the University of California at Santa Barbara, Santa Barbara, CA, USA, from 2009 to 2010. She currently serves as the Secretary General of the China Computer Federation (CCF) Technical Committee on Integrated Circuit Design, the Steering Committee Chair of IEEE Asian Test Symposium (ATS), and the Associate Editor of IEEE Transactions on Very Large Scale Integration (VLSI) Systems and IEEE Design Test. Her current research interests include testing of very large-scale integration/SoC circuits, approximate computing architecture and machine learning accelerators. She has published more than 200 technical papers and holds 34 Chinese patents in the above areas.

Xiaowei Li (SM'04) received the B.Eng. and M.Eng. degrees in computer science from the Hefei University of Technology, Hefei, China, in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1991. He was an Associate Professor with the Department of Computer Science and Technology, Peking University, Beijing, from 1991 to 2000. In 2000, he joined ICT, CAS, as a Professor, where he is currently the Deputy Director of the State Key Laboratory of Computer Architecture. He has coauthored over 280 papers in journals and international conferences, and he holds 60 patents and 30 software copyrights. His current research interests include VLSI testing, design for testability, design verification, dependable computing, and hardware security.