Received 16 October 2023, accepted 2 January 2024, date of publication 15 January 2024, date of current version 25 January 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3354275*

RESEARCH ARTICLE

# Heterogeneous Reconfigurable Accelerator for Homomorphic Evaluation on Encrypted Data

**WENQING SONG[1], SIRUI SHEN[2], CONGWEI XU[1], YILIN WANG[1], XINYU WANG[1], YUXIANG FU[3], (Member, IEEE), LI LI[1], AND ZHONGHAI LU[4], (Senior Member, IEEE)**

[1]School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China
[2]Centrum Wiskunde & Informatica (CWI Amsterdam), 1098 XG Amsterdam, The Netherlands
[3]School of Integrated Circuit, Nanjing University, Nanjing 210023, China
[4]School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Kista, 164 40 Stockholm, Sweden

Corresponding authors: Li Li (lili@nju.edu.cn) and Yuxiang Fu (yuxiangfu@nju.edu.cn)

This work was supported in part by the Joint Funds of the National Natural Science Foundation of China under Grant U21B2032, in part by the National Natural Science Foundation of China under Grant 62104098, in part by the Natural Science Foundation of Jiangsu Province for Youth under Grant BK20210178, and in part by the National Key Research and Development Program of China under Grant 2021YFB3600104.

**ABSTRACT** Homomorphic encryption (HE) enables third-party servers to perform computations on encrypted user data while preserving privacy. Although conceptually attractive, the speed of software implementations of HE is almost impractical. To address this challenge, various domain-specific architectures have been proposed to accelerate homomorphic evaluation, but efficiency remains a bottleneck. In this paper, we propose a homomorphic evaluation accelerator with heterogeneous reconfigurable modular computing units (RCUs) for the Brakerski/Fan-Vercauteren (BFV) scheme. RCUs leverage operator abstraction to efficiently perform basic sub-operations of homomorphic evaluation such as residue number system (RNS) conversion, number theoretic transform (NTT), and other modular computations. By combining these sub-operations, complex homomorphic evaluation operations like multiplication, rotation, and addition are efficiently executed. To address the high demand for data access and improve memory efficiency, we design a coordinate-based address encoding strategy that enables in-place and conflict-free data access. Furthermore, specific optimizations are performed on the core sub-operations such as NTT and automorphism. The proposed architecture is implemented on Xilinx Virtex-7 and UltraScale+ FPGA platforms and evaluated for polynomials of length 4096. Compared to state-of-the-art accelerators with the same parameter set, our accelerator achieves the following advantages: 1) 2.04× to 3.33× reduction in the area-time product (ATP) for the key sub-operation NTT, 2) 1.08× to 7.42× reduction in latency for homomorphic multiplication with higher area efficiency, and 3) support for a wider range of homomorphic evaluation operations, including rotation, compared to other BFV-based accelerators.

**INDEX TERMS** BFV, hardware acceleration, homomorphic encryption, number theoretic transform.

## I. INTRODUCTION

Homomorphic encryption (HE), first proposed in 1978 [1], has gained widespread attention for its ability to perform computation directly on ciphertexts. The first HE construction that can execute an unlimited number of homomorphic multiplication and addition operations is realized by Gentry [2] in 2009. The first-generation scheme suffers from large ciphertext sizes and high computational costs. Therefore, multiple present-generation HE schemes such as BFV [3], [4], BGV [5], CKKS [6], and TFHE [7] have been proposed with different functionalities and better performance. While HE schemes remain computationally intensive, the ability to process private data in a secure environment makes

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Masucci.

them well-suited for cloud computing. Users can upload the encrypted data to the cloud, allowing the cloud server to perform computations without gaining access to sensitive information. Homomorphic evaluation has been extensively studied and successfully applied to implement neural networks and machine learning algorithms [8], [9], [10].

Among the various HE schemes, we specifically target the Brakerski/Fan-Vercauteren (BFV) scheme due to its support for integer arithmetic, batch processing, and wide applicability. The BFV scheme has been broadly implemented in several software libraries, such as SEAL [11], FV-NFLlib [12], and OpenFHE [13]. The complexity of the BFV scheme arises from the multiplication of large-degree polynomials and the computation between large coefficients. The former challenge can be solved by the number theoretic transform (NTT), which is a variant of the fast Fourier transform (FFT) in a finite field [14]. The latter challenge can be solved by the residue number system (RNS), which utilizes sets of small bit-width integers to represent large coefficients [15]. These two methods help reduce computational complexity by avoiding the need for expensive multi-precision arithmetic in CPU, but the speed of software implementation remains unsatisfactory.

Homomorphic evaluation, which is the most complex part of HE, has been the focus of numerous works. Many previous works focus on accelerating low-level arithmetic operations such as modular multiplication [16] and NTT [17], but neglect the frequent data interaction costs with software-implemented functions. GPU implementations [18], [19] and FPGA-based accelerators [20], [21], [22], [23] show remarkable speedup compared to general-purpose CPUs. Cheetah [24] is the first ASIC accelerator to support private inference based on the BFV scheme. As a relatively new algorithm, homomorphic evaluation still requires further improvements in processing speed and efficiency to enhance the practicality of HE-based privacy-preserving applications.

In this paper, we present a heterogeneous reconfigurable accelerator to further improve the efficiency of homomorphic evaluation based on an RNS variant of BFV called Halevi-Polyakov-Shoup (HPS) [25], [26]. We implement and evaluate the design on Xilinx Virtex-7 and UltraScale+ FPGA platforms using polynomials of degree 4096. The contributions of this paper are listed as follows.

1) Based on the frequency statistics of common operators in homomorphic evaluation operations, we discover that heterogeneous computing is more suitable for accelerating the BFV scheme. A reconfigurable computing unit (RCU) that consists of three types of processing elements (PEs) is built upon this finding, aiming to balance resource utilization and computing efficiency.
2) After splitting the homomorphic evaluation into basic sub-operations such as NTT, RNS conversion, key switching, and other modular computations, we map the sub-operations efficiently on the RCU to achieve lower latency. Besides, to the best of our knowledge, our accelerator is the first BFV evaluation accelerator that can support the HE rotation operation.
3) For efficient management of internal data movement, we propose a coordinate-based address encoding scheme to address the issue of data interleaving within and between polynomials. The proposed scheme ensures in-place data storage and conflict-free data access throughout the entire process.
4) We implement the core sub-operation NTT and the overall accelerator on Xilinx Virtex-7 and UltraScale+ FPGA platform respectively, to evaluate the performance in practice. The implementation results show that our accelerator achieves the following improvements: 1) $2.04\times$ to $3.33\times$ improvement in the area-time product (ATP) compared to other NTT designs [17], [23], [27], [28], [29], 2) $1.08\times$ to $7.42\times$ reduction in latency for homomorphic multiplication with higher area efficiency compared to other BFV accelerators [21], [22], [23].

The remainder of this paper is organized as follows. Section II provides a review of HE based on the BFV scheme, as well as an overview of the basic operations involved in homomorphic evaluation. Section III analyzes the heterogeneous nature of homomorphic evaluation computations and discusses the specification of the heterogeneous RCU. The overall architecture and optimization details of the proposed accelerator are described in Section IV, and Section V presents the implementation results and comparisons. Section VI concludes the paper.

## II. PRELIMINARIES

### A. BFV SCHEME

During homomorphic evaluation, each operation introduces noise growth, and decryption will fail when the accumulated noise exceeds the tolerance. The HE scheme that supports a limited number ('depth') of operations is called leveled HE (LHE), which can be transformed into fully HE (FHE) by *bootstrapping*. Nevertheless, the *bootstrapping* mechanism requires a large parameter set consuming plenty of resources. Note that the BFV *bootstrapping* is seldom used in real-life applications [24] and is currently not supported in any open-source HE software library. Therefore, this paper only introduces the leveled BFV scheme.

For $\mathcal{A} \geq 2$ ($\mathcal{A} \in \mathbb{Z}$), the set of integers in the symmetric interval $\mathbb{Z} \cap [-\frac{\mathcal{A}}{2}, \frac{\mathcal{A}}{2})$ are denoted by $\mathbb{Z}_{\mathcal{A}}$, and the operation $x \bmod \mathcal{A}$ is denoted by $[x]_{\mathcal{A}}$. There are three core parameters $\{n, t, q\}$ in the BFV scheme. The first parameter $n$ denotes the degree of the polynomial, and all computations in the BFV scheme are performed in the polynomial ring $R = \mathbb{Z}[x]/(x^n + 1)$, where $n$ is a power of 2. The other two parameters $t, q$ denote the plaintext ($pt$) modulus and the ciphertext ($ct$) modulus respectively, i.e., the coefficients of the $pt$ and $ct$ polynomials are elements of the ring $R_t$ and $R_q$ respectively. The choice of parameter set will determine the security level and the noise budget for the operations performed on the ciphertext.

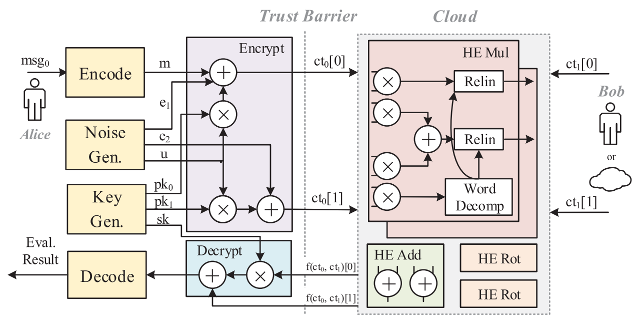


**FIGURE 1.** **BFV LHE scheme for private computing on the cloud.**

The entire BFV-based LHE flow is shown in Fig. 1, and the detailed description may follow [4]. We denote the public key and secret key by $\{pk_0, pk_1\}$ and $sk$ respectively. For *Alice*, the transmitted messages $msg_0$ are first encoded to the polynomial form $m_0 \in R_t$. After encryption, the encoded polynomials are hidden by three noise polynomials $\{u, e_1, e_2\}$ sampled from discrete Gaussian distribution and uploaded to the cloud. Combining with the ciphertexts obtained from other sources such as Bob or the cloud, homomorphic evaluation operations between ciphertexts can be performed in the cloud:

- *FV.PAdd(pt, ct)*: *pt*-*ct* addition is achieved by $\{ct[0] + \lfloor q/t \rfloor \cdot pt, ct_0[1]\}$.
- *FV.CAdd($ct_0$, $ct_1$)*: *ct*-*ct* addition is achieved directly by $\{ct_0[0] + ct_1[0], ct_0[1] + ct_1[1]\}$.
- *FV.PMul(pt, ct)*: *pt*-*ct* multiplication can directly obtain the results of $\{pt \cdot ct[0], pt \cdot ct[1]\}$:
- *FV.CMul($ct_0$, $ct_1$, rlk)*: *ct*-*ct* multiplication operation consists of two steps:
  - First, two ciphertexts are multiplied to obtain a ciphertext $ct'_\times$ containing three ring elements $\{ct_0[0]\cdot ct_1[0], ct_0[0]\cdot ct_1[1]+ct_0[1]\cdot ct_1[0], ct_0[1]\cdot ct_1[1]\}$, and scales it by multiplying $t/q$.
  - Then, $ct'_\times$ is reduced from three terms to two with the *relinearization* key $\{rlk_0, rlk_1\}$. The $sk^2$ term is decomposed, then subjected to multiply-accumulation operations with $\{rlk_0, rlk_1\}$ respectively, and finally the obtained results are added to the other two terms.
- *FV.Rot($ct_0$, rk)*: the rotation operation is provided to rotate the ciphertext based on Galois automorphism [30]. Except for the index mapping operation, the rotation key *rk* is employed to make the rotated ciphertext decryptable by the original *sk*.

Finally, the client who owns the private key *sk* can decrypt the evaluation result.

## B. IMPROVED RNS VARIANT OF BFV

To enable parallel computing, a large modulus $q$ is split into several small primes $\{q_0, \ldots, q_{k-1}\}$ which satisfies $q = \prod_{i=0}^{k-1} q_i$ using RNS. Based on the decomposition, arithmetic on $\mathbb{Z}_q$ could be represented by the same operations on $\mathbb{Z}_{q_i}$. Computations with these smaller moduli can be performed in parallel using lower bit-width arithmetic. Two RNS variants named Bajard-Eynard-Hasan-Zucca (BEHZ) [15] and HPS [25] have been applied to the BFV scheme recently. Compared to BEHZ, which uses only complicated integer operations and suffers from larger noise growth, the HPS scheme adopts a mixture of floating-point and integer operations, which has nearly the same noise growth as the original BFV scheme and performs better in practice [31]. To improve the implementation efficiency, we adopt an improved HPS scheme [26] with reduced computational complexity.

For the RNS basis $\{q_0, \ldots, q_{k-1}\}$, an integer $x \in \mathbb{Z}_q$ can be reconstructed from a group of $x_i$ ($x_i = [x]_{q_i}$) via Chinese Remainder Theorem (CRT) by (1).

$$x = (\sum_{i=0}^{k-1}[x_i \cdot \tilde{q}_i]_{q_i} \cdot q_i^*) - v \cdot q, v \in \mathbb{Z}, \tag{1}$$

where $q_i^* = \frac{q}{q_i} \in \mathbb{Z}$, and $\tilde{q}_i = (q_i^*)^{-1} \bmod q_i \in \mathbb{Z}_{q_i}$. When performing homomorphic evaluation using CRT, three operations are introduced in [25] and [26] for conversions between different RNS bases: $BaseExtend_{\times\to Q}$, $ModSwitch_{q\to p}$, and $ScaleDown_{Q\to q}$, where $p$ is a modulus that is close in value to $q$ and coprime with $q$ ($p = \prod_{j=0}^{k-1} p_j$), and $Q$ is a larger modulus with the value $qp$.

*Definition 1: $BaseExtend_{q\to Q}$* denotes the operation that convert the input $x$ from $R_q$ to $R_p$.

To extend the RNS basis, $[x]_p$ is calculated as follows.

$$[x]_{p_j} = \left[\left(\sum_{i=0}^{k-1}[x_i \cdot \tilde{q}_i]_{q_i} \cdot [q_i^*]_{p_j}\right) - v \cdot [q]_{p_j}\right]_{p_j}, \tag{2}$$

$$v = \left\lceil \sum_{i=0}^{k-1} \frac{[x_i \cdot \tilde{q}_i]_{q_i}}{q_i} \right\rfloor. \tag{3}$$

*Definition 2: $ModSwitch_{q\to p}$* denotes the process of performing modulus switching on the input $x$ from $R_q$ to $R_p$.

After adopting Def. 1, the obtained $[\hat{x}]_p = [\lceil \frac{p}{q} \cdot [x]_q \rfloor]_p$ is calculated as follows.

$$[\hat{x}]_{p_j} = \left[-\left(\sum_{i=0}^{k-1}[x_i \cdot \tilde{q}_i \cdot p]_{q_i} \cdot [q_i^{-1}]_{p_j}\right) + u\right]_{p_j}, \tag{4}$$

$$u = \left\lceil \sum_{i=0}^{k-1} \frac{[x_i \cdot \tilde{q}_i \cdot p]_{q_i}}{q_i} \right\rfloor. \tag{5}$$

*Definition 3: $ScaleDown_{Q\to q}$* denotes the process of converting $x$ from $R_Q$ back to $R_q$ and scaling it by $t/p$.

Based on two CRT components $x_i = [x]_{q_i}$ and $x'_j = [x]_{p_j}$, the scaling operation is computed as shown in (6).

$$\left[\left\lceil \frac{t}{p} \cdot x \right\rfloor\right]_{q_i} = \left[\left\lceil \sum_{j=0}^{k-1} x'_j \cdot \frac{t\tilde{Q}'_j q}{p_j} \right\rfloor + x_i \cdot [t\tilde{Q}_i q_i^*]_{q_i}\right]_{q_i}, \tag{6}$$

where $\tilde{Q}_i = [(q_i^* p)^{-1}]_{q_i}$, and $\tilde{Q}'_j = [(qp_j^*)^{-1}]_{p_j}$. For ease of the hardware implementation, the values of $\frac{t\tilde{Q}'_j q}{p_j}$ could

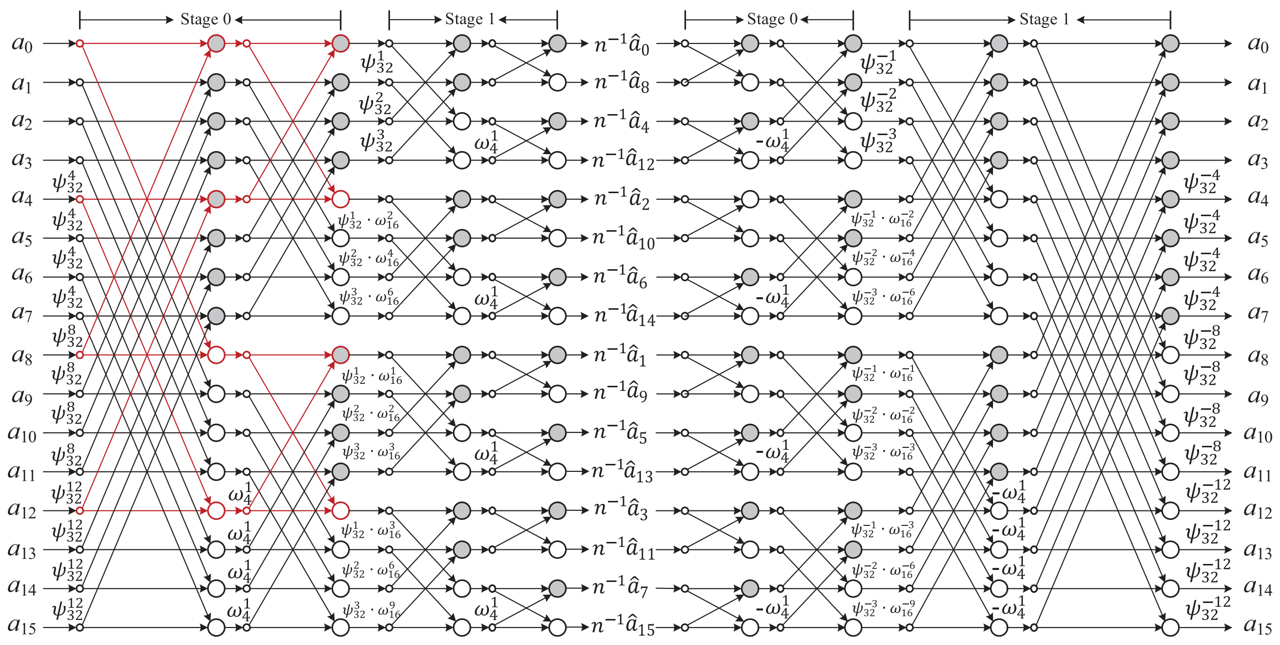


**FIGURE 2.** Signal flow graph of DIT NTT and DIF INTT algorithms, where $n = 16$.

be pre-computed and split into the integer part $\omega_j \in \mathbb{Z}_p$ and decimal part $\theta_i \in [-\frac{1}{2}, \frac{1}{2})$. For ease of hardware implementation, the decimal part $\theta_i$ can be represented by large bit-width fixed-point numbers without any impact on security [21].

Since all constant values obtained only by $q, p, Q$ can be pre-computed, the HPS scheme performs as a hardware-friendly approach without the divide-and-round operations.

## C. POLYNOMIAL MULTIPLICATION

The naive approach of polynomial multiplication between two polynomials of degree $n$ consumes a complexity of $\mathcal{O}(n^2)$. Nevertheless, polynomial multiplication over the ring can be transformed into coefficient-wise multiplication ($\circ$) by NTT to reduce the computational complexity to $\mathcal{O}(n \log_2 n)$. To avoid doubling the polynomial length with zero-padding and reducing modulo $(x^n + 1)$, the *negative wrapped convolution* (NWC) technique is exploited [32]. Let $w_n$ be a $n$-th primitive root of unity in $\mathbb{Z}_q$ that satisfies $w_n^n = 1 \bmod q$, and $q$ should satisfy $q \equiv 1 \pmod{2n}$. Before NTT, the coefficients are weighted by a $2n$-th primitive root of unity $\psi \in \mathbb{Z}_q$ ($\psi^2 = w_n$), i.e., $\{a_0, \psi \cdot a_1, \ldots, \psi^{n-1} \cdot a_{n-1}\}$. After the inverse transformation INTT, the multiplication between polynomial $\mathbf{a}(x)$ and $\mathbf{b}(x)$ with the reduction modulo $x^n + 1$ is given by (7).

$$\mathbf{a}(x) \cdot \mathbf{b}(x) \bmod (x^n + 1) = (1, \psi^{-1}, \ldots, \psi^{-(n-1)}) \circ \mathrm{INTT}(\mathrm{NTT}(\mathbf{a}(x)) \circ \mathrm{NTT}(\mathbf{b}(x))). \quad (7)$$

The polynomial $\mathbf{a}(x)$ with coefficients $\{a_0, \ldots, a_{n-1}\}$ is transformed to the NTT representation $\hat{\mathbf{a}}(x)$ with coefficients $\{\hat{a}_0, \ldots, \hat{a}_{n-1}\}$ by (8).

$$\hat{a}_i = \sum_{j=0}^{n-1} a_j w_n^{ij} \bmod q, \; i \in [0, n). \quad (8)$$

Similar to the NTT operation, the INTT operation simply replaces $w_n$ with $w_n^{-1}$ and divides the final result by $n$:

$$a_i = \frac{1}{n} \sum_{j=0}^{n-1} \hat{a}_j w_n^{-ij} \bmod q, \; i \in [0, n). \quad (9)$$

Similar to FFT, the basic operation of NTT could be achieved by two types of butterfly: the Cooley-Tukey (CT) butterfly [33] for the decimation-in-time (DIT) NTT and the Gentleman-Sande (GS) butterfly [34] for the decimation-in-frequency (DIF) NTT. NWC-based polynomial multiplication can be further enhanced by combining NTT and pre-processing together [35], INTT and post-processing together [36], and performing shift and addition operations during INTT in place of final multiplication by $1/n$ [37]. If the CT butterfly with native order to bit-reversed order is used to perform NTT and the GS butterfly with inverse order to perform INTT, the bit-reversal operation can be avoided [36]. Similar to FFT, the radix-4 NTT could achieve better throughput with the same resources compared to the radix-2 NTT [38]. The signal flow graph of the radix-4 native order to bit-reversed order DIT NTT and the inverse order DIF INTT of a polynomial $\mathbf{a}(x)$ with $n = 16$ is shown in Fig. 2, where pre-processing and post-processing are merged into NTT and INTT, respectively. It can be observed that additional bit-reversal operations on the polynomial

**TABLE 1. Basic sub-operations in homomorphic evaluation operations.**

| | RNS Conv. | (i)NTT | MA | MMUL | MMAC |
|---|---|---|---|---|---|
| Rot | | ✓ | ✓ | | ✓ |
| CMul | ✓ | ✓ | ✓ | | ✓ |
| PMul | | ✓ | | ✓ | |
| C/PAdd | | | ✓ | | |

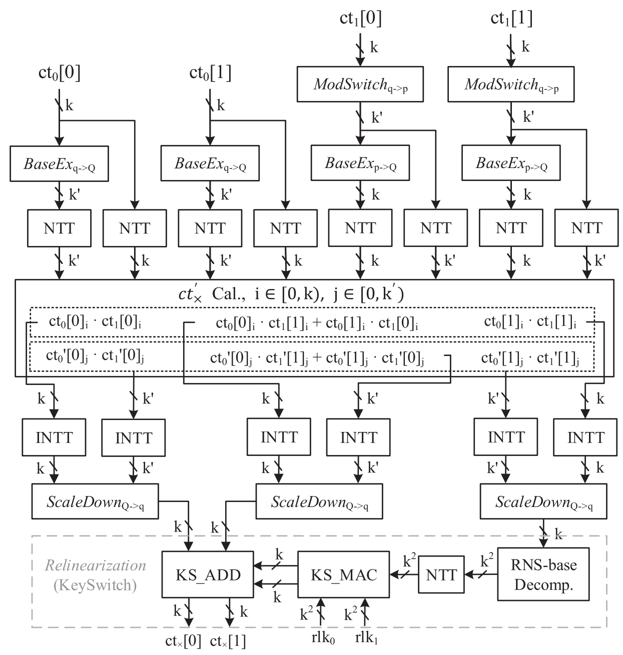


**FIGURE 3.** Flow of CMul operation.

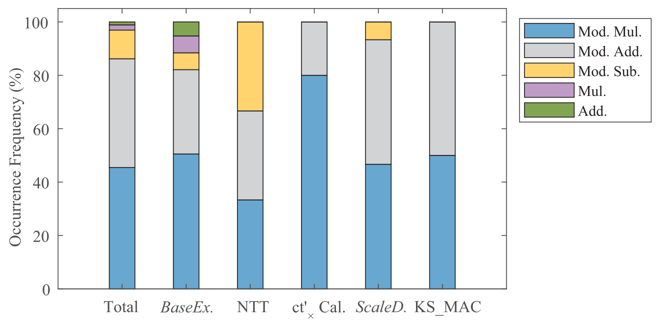


**FIGURE 4.** Occurrence frequency of each operator when $n = 4096$, $k = 6$.

coefficients can be avoided while the rearrangement of the twiddle factors between groups is required.

## III. DESIGN OF HETEROGENEOUS RECONFIGURABLE COMPUTING UNIT

### A. COMPUTATION HETEROGENEITY OF HOMOMORPHIC EVALUATION SUB-OPERATIONS

The homomorphic evaluation operations can be split into several basic sub-operations including RNS conversion, (i)NTT, modular addition (MA), modular multiplication (MMUL), and modular multiply-accumulation (MMAC), as listed in Table 1. Among these operations, the CMul operation is the most complex in homomorphic encryption. The flow of the CMul operation is shown in Fig. 3 where the numbers next to the arrows indicate the number of input or output polynomials, with each polynomial having a degree of $n$. By adopting the improved HPS scheme [26], the number of extended RNS bases $k'$ reduces from $k + 1$ to $k$, thus $k'$ is only used to denote that the polynomials are in $R_p$. Furthermore, the rotation operation consists of two steps: polynomial permutation and key switching (KS). The KS step comprises NTT, INTT, MMAC, and MA sub-operations, displaying a similar computing flow to the *Relinearization*. In the following, we will take these sub-operations as the foundation to perform operator abstraction and design the accelerator architecture.

The existing homomorphic evaluation accelerators [21], [22], [39], [40] tend to implement the sub-operations directly but neglect the commonalities among sub-operations. ReMCA [23] shows that the reconfigurable architecture that implements different sub-operations on the same resource can achieve better efficiency. Nevertheless, the adopted homogeneous architecture increases the mapping complexity of the non-modular computations involved in RNS conversion. Inspired by the design idea of domain-specific coarse-grained reconfigurable architecture (CGRA) [41], we chose to extract the basic operators from these sub-operations and form heterogeneous reconfigurable PEs according to the occurrence frequency of each operator. A typical example with $n = 4096, k = 6$ is given in Fig. 4, which presents the imbalance between different operators. Let $f_\varrho$ denote the occurrence frequency of each operator $\varrho$ during the whole homomorphic evaluation procedure. When the total number of PEs equals $M$, the number of PEs that contain the operator $\varrho$ should not be less than $f_\varrho \cdot M$.

Before determining the specific number of each operator to be implemented, we need first to determine the PE architectures inside the RCU. To reduce the difficulty of mapping sub-operations on the RCU, operator fusion is employed. By fusing operators, frequently occurring operators such as multiply-accumulation and butterfly operators can be executed within a single PE. Taking into account the moduli required by each modular computing unit (MU), the proposed design template for the heterogeneous computing unit is illustrated in Fig. 5, comprising three types of PE. Type-I PEs include an equal number of modular adders, modular subtractors, and modular multipliers, which are responsible for most modular computations and can be reconfigured as different modes. Several local buffers are implemented in Type-I PE for the butterfly unit. Type-II PEs are also composed of basic MUs, but each MU within the PE can be configured to support multiple moduli. Type-III PEs only consist of arithmetic multipliers and adders, which are utilized for implementing sub-operations related to RNS conversions. The quantity of each type of PE will be specified in the subsequent subsection.

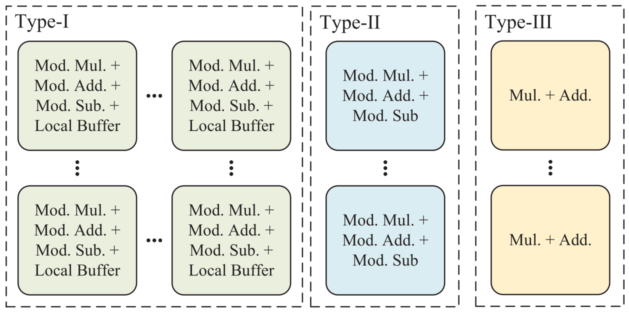

Type-I
Mod. Mul. + Mod. Add. + Mod. Sub. + Local Buffer
Type-II
Mod. Mul. + Mod. Add. + Mod. Sub
Type-III
Mul. + Add.

**FIGURE 5.** Heterogeneous computing unit template.

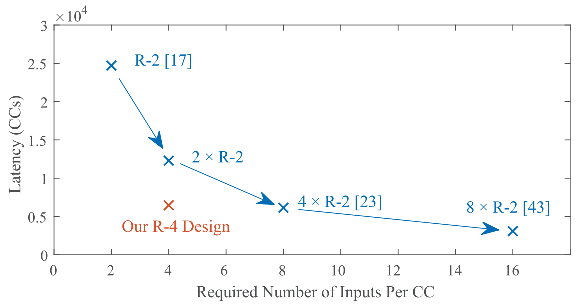


**FIGURE 6.** Impact of butterfly unit.

## B. RECONFIGURABLE PE SPECIFICATION

From a macro perspective, homomorphic evaluation exhibits parallelism in two dimensions: within a polynomial ($n$-way parallel) and between polynomials($k$-way parallel) [21]. Nevertheless, when delving into the sub-operations, certain sub-operations exhibit data interleave. For instance, there is an interleave within polynomials within NTT, while RNS conversion involves an interleave among $k$ polynomials. Therefore, the accelerator specification needs to consider both the overall parallel architecture and the specific optimization within the sub-operations.

### 1) PARALLELISM BETWEEN PES

We first determine the number of Type-I PEs since they play a crucial role in leveraging the parallelism introduced by RNS. Previous accelerators [21], [22], [23] based on the original HPS scheme maintained $(k+1)$ or $(2k+2)$-parallel channels on-chip, resulting in underutilization of the last one or two channels. With the aid of the improved HPS scheme, we set the number of Type-I PEs to $2k$. Each of the first $k$ PEs and the last $k$ PEs can support two moduli, either $q_i$ or $p_i$ ($i \in [0, k)$). Accordingly, two RNS polynomial groups can be processed simultaneously. The Type-II and Type-III PEs are only utilized by RNS conversion. To handle the interleaved computations among different RNS polynomials, two Type-II PEs are specialized for two sets of $k$ RNS polynomials. Similarly, two Type-III PEs are employed to support integer arithmetic.

### 2) PARALLELISM WITHIN EACH PE

Apart from NTT, the remaining sub-operations only involve coefficient-wise computations. Therefore, the parallel degree within each PE primarily influences the NTT design. As the basic component of NTT, the butterfly unit significantly impacts the performance of NTT and the required data bandwidth, as illustrated in Fig. 6. Compared with other radix-2-based designs, the radix-4 butterfly unit can achieve a better trade-off between latency and data bandwidth. Consequently, a parallelism level of 4 is set within each PE, allowing the implementation of either a radix-4 butterfly unit or a 4-way modular computation.

In the following section, we will present the accelerator design based on the customized heterogeneous computing unit, which includes defining the overall architecture, designing the internal structure of PEs, determining the storage scheme, and mapping the sub-operations on the accelerator.

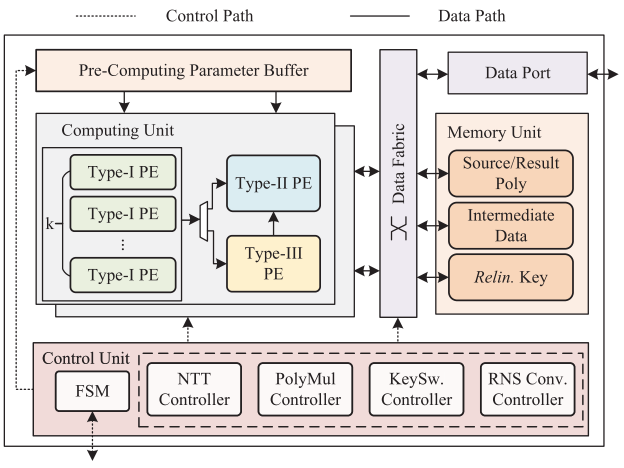


**FIGURE 7.** Overall hardware architecture.

## IV. HARDWARE ARCHITECTURE AND OPTIMIZATION

### A. OVERALL ARCHITECTURE

The overall architecture of our accelerator is shown in Fig. 7. The proposed accelerator consists of three major components: a computing unit to support basic homomorphic evaluation operations, a memory unit to store the polynomials and pre-computed parameters, and a controller unit to control the entire procedure.

#### 1) CONTROL UNIT

The control unit configures the computing unit and the storage unit through two stages. The finite-state machine (FSM) is responsible for receiving the external configuration information and scheduling the whole process. The sub-controller contains the NTT controller, the RNS conversion controller, the polynomial coefficient multiplication controller, and the *relinearization* controller. The duty of the sub-controller is to configure the computing path and data path to execute each sub-operation of homomorphic evaluation.

#### 2) COMPUTING UNIT

As introduced in Sec. III-B, the computing unit consists of $2k$ Type-I PEs, two Type-II PEs, and two Type-III PEs. Here, we will focus on introducing the internal structure

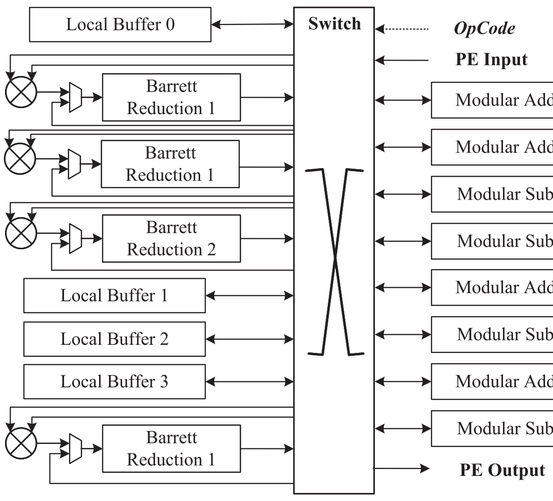


**FIGURE 8.** Architecture of Type-I PE.

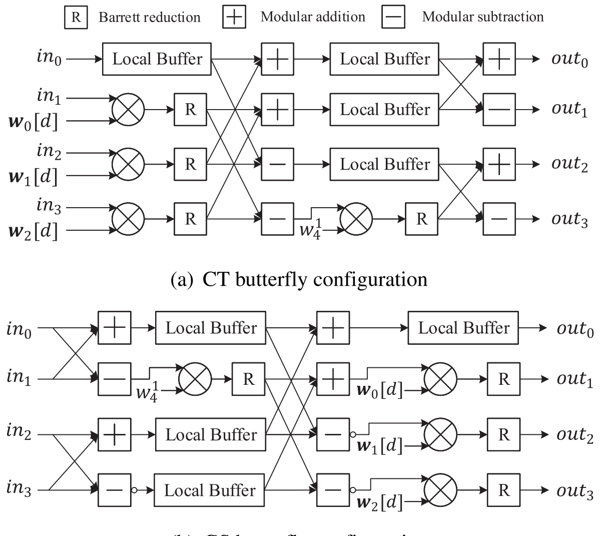


**FIGURE 9.** Configuration of Type-I PE for the radix-4 butterfly.

**TABLE 2.** Six computing modes of the reconfigurable modular computing unit.

| Function | $P$ | *OpCode* |
|---|---|---|
| CT butterfly | 1 | $3'$b000 |
| GS butterfly | 1 | $3'$b001 |
| MAC (length $= k$) | 4 | $3'$b010 |
| Polynomial coefficient multiplication | 1 | $3'$b011 |
| Modular addition | 4 | $3'$b100 |
| Modular multiplication | 4 | $3'$b101 |

of Type-I PEs, while the internal structure of Type-II and Type-III PEs will be discussed in the RNS conversion section. As shown in Fig. 8, each Type-I PE consists of four modular multipliers which include a multiplier and a Barrett reduction [42] unit, four modular adders, and four modular subtractors. The switch is used to parse the configuration information and transfer data between MUs and local buffers. Each Type-I PE can be reconfigured using a separate operation code (*OpCode*) and supports six different modes, including four parallel radix-4 NTT or INTT butterfly units, modular multiply-accumulators or other modular operators. The supporting coarse-grained operators are listed in Table 2, where $P$ denotes the parallel degree of the operation. The function polynomial coefficient multiplication is implemented as coefficient-wise computation of $\{ct_0[0] \cdot ct_1[0], ct_0[0] \cdot ct_1[1] + ct_0[1] \cdot ct_1[0], ct_0[1] \cdot ct_1[1]\}$.

### 3) MEMORY UNIT

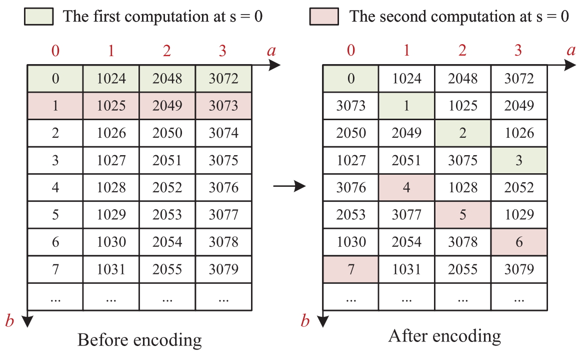
To ensure efficient and conflict-free on-chip data access, a maximum data bandwidth of $16k \cdot \lceil \log_2 q_i \rceil$ bits per clock cycle (CC) should be supported by the accelerator. Accordingly, we partition the memory units into several bank groups. One bank group is implemented for the storage of source and result polynomials, which can communicate with off-chip via the AXI interface and also participate in internal operations; another bank group is implemented for the temporary storage of intermediate results and does not communicate with peripherals. Both of them have a capacity of $4kn \cdot \lceil \log_2 q_i \rceil$ bits and are composed of $4k$ 2-port SRAMs with a bit-width of $\lceil \log_2 q_i \rceil$ bits and a depth of $n/4$. Due to the larger size of the *relinearization* key, a dedicated bank group with a capacity of $2k^2 \cdot \lceil \log_2 q_i \rceil$ bits is employed and connected to the AXI interface, which is composed of $2k$ single-port SRAMs with a bit-width of $\lceil \log_2 q_i \rceil$ bits and a depth of $kn$. The pre-computed parameters required by NTT and RNS conversion are stored in a read-only ROM.

## B. OPTIMIZATION OF KEY SUB-OPERATIONS

### 1) SINGLE-STREAM RADIX-4 NTT

To accommodate the limited resources and memory bandwidth, we employ a single-stream radix-4 butterfly structure to perform NTT of a polynomial. Each Type-I PE is responsible for an NTT/INTT sub-operation at a fixed modulus. The Type-I PE can be reconfigured into CT and GS butterflies when performing NTT and INTT respectively. By configuring *Opcode*, a Type-I PE can be reconfigured to perform the radix-4 butterfly operation, as shown in Fig. 9. The modular subtractor defaults to subtracting the lower input from the upper input. To enable the reuse of twiddle factors, the output values of the lower three modular subtractors need to be flipped, and the location of multiplication with $w_4^1$ is also adjusted. The twiddle factors are precomputed and stored in the same manner as [38].

As shown in [37], the multiplication with $n^{-1} \bmod q$ in INTT can be achieved by adding a modular multiplication by $1/2$ after every modular addition or modular subtraction. We further find that this step can be combined with modular addition and subtraction. We make slight modifications to the architecture of the modular adder and modular subtractor in [43]. By adding an additional adder and shifter to the modular adder and modular subtractor, the modular multiplication by $1/2$ can be directly implemented in the computation process without the need for an extra stage.



The first computation at s = 0
The second computation at s = 0
a
b
Before encoding
After encoding

**FIGURE 10. Address encoding scheme of NTT with $n = 4096$.**

Apart from performing butterfly computations, a more challenging issue to address is the dependent data access pattern within a polynomial. The pipelined radix-4 butterfly unit requires simultaneous reading of four source data ($in_0, in_1, in_2, in_3$) and writing back of four results ($out_0, out_1, out_2, out_3$). There are two commonly used data access patterns: in-place access strategy and out-of-place access strategy. The in-place access strategy utilizes 2-port SRAMs with a specific address encoding method to achieve conflict-free access [38]. The out-of-place access strategy separates read and write addresses using either dual-port SRAMs [22] or a double number of single-port SRAMs for *ping-pong* [44]. The in-place data access strategy is more area-efficient than the out-of-place one, but the address encoding scheme is more complex. The previous in-place strategy [38] tends to complete the address encoding before NTT computation, resulting in additional data reordering time when integrating with preceding sub-operations that adopt a sequential storage scheme, such as automorphism or $ScaleDown_{Q\to q}$.

The storage of a polynomial in the accelerator follows the principle depicted on the left side of Fig. 10, where each coefficient with index $i$ is stored at position $b$ in bank $a$. The coordinates $(a, b)$ are used to represent the storage locations, where $a = \lfloor i/(n/4) \rfloor$ and $b = [i]_{n/4}$. When performing an $n$-point radix-4 NTT, the total number of stages $stotal$ is $\log_4 n$. At stage $s$, four coefficients spaced by $4^{s_{total}-s-1}$ need to be simultaneously read (as demonstrated in Fig. 2), and the four computation results also need to be simultaneously written back. The initial data storage strategy meets the read requests of the first stage, but write conflicts occur after computation. Therefore, we perform address encoding during the writing phase of the first stage. Our proposed address encoding scheme groups every $4^v$ elements ($v > 0$) and performs rotations between groups, as shown in the right side of Fig. 10. The coordinate values when each coefficient is written back at the first stage are calculated by (10), where $\epsilon = (\log_2(n/4))/2$.

$$\begin{cases} enc\_a = \left[ a + \left[ \sum_{\delta=0}^{\epsilon} b[2\delta+1 : 2\delta] \right]_4 \right]_4, \\ enc\_b = b. \end{cases} \tag{10}$$

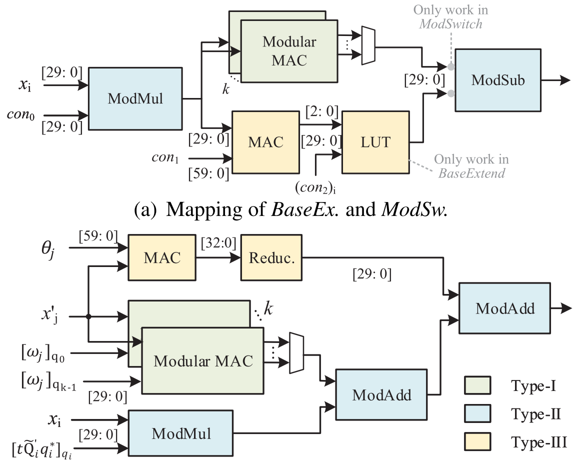


**FIGURE 11. Mapping the RNS conversion on computing unit.**

In the subsequent stages, both read and write operations are indexed based on the encoded coordinate values. During INTT execution, the data access interval within each stage is $4^s$, which is reversed compared to NTT. Our proposed encoding scheme also ensures conflict-free access. The data storage format is changed back to the one shown on the left side of Fig. 10 at the last stage when the coefficients are written back.

### 2) PIPELINED RNS CONVERSION

Due to the coefficient-wise nature of RNS conversions and the involvement of $k$ polynomials in computations, the maximum parallel degree supported by our accelerator is eight, which means that our accelerator can process two groups of RNS polynomials simultaneously in a 4-way parallelism. In the following discussion, we will focus on the implementation of a single-way configuration using 30-bit $q_i$ and $p_i$.

As shown in Fig. 3, RNS conversions in homomorphic evaluation involve four sub-operations: $ModSwitch_{q\to p}$, $BaseExtend_{p\to Q}$, $BaseExtend_{q\to Q}$, and $ScaleDown_{Q\to q}$. Among these sub-operations, *BaseEx.* and *ModSw.* share similar computational structures with minor differences in the involved precomputed parameters and whether they operate on $R_q$ or $R_p$. Besides, *ModSw.* does not involve the computation of $v \cdot con_2$ and requires swapping the positions of the two inputs during modular subtraction.

Therefore, the above three sub-operations can employ a common mapping strategy as shown in Fig. 11(a). The inputs $\{con_0, con_1, con_2\}$ represent the precomputed parameters specific to each sub-operation. The *ScaleDown* sub-operation is mapped on the computing unit following the configuration shown in Fig. 11(b). The blocks colored green are mapped on the Type-I PEs to perform the modular MAC operator. The blocks colored blue are mapped on the Type-II PE, and the blocks colored yellow are mapped on the Type-III PEs. A total of $2 \times 4$ Type-II PEs and $2 \times 4$ Type-III PEs are utilized for 4-way parallel RNS convention operation. In each

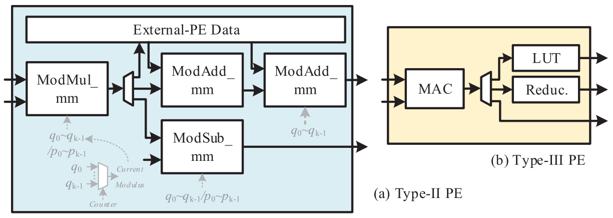


**FIGURE 12.** Architecture of Type-II & Type-III PE.

Type-II PE, the input value with subscript $i$ and the supported modulus will be alternated every clock cycle after the block is initiated, with a cycle of every six CCs. Therefore, the modular arithmetics are designed to support multi-moduli (mm). The architecture of a Type-II PE is shown in Fig. 12(a).

Each parallel path supported by the Type-III PE contains a non-modular MAC unit to perform the multiplication and accumulation between polynomial coefficients and 60-bit decimals. When executing *BaseEx.* or *ModSw.*, the 60-bit decimal part does not include the first 29-bit zeros after the decimal point of $1/q_i$. The rounded MAC result $v$ is an integer ranging from 0 to $k$. Therefore, for *BaseEx.*, the required value of $v \cdot (con_2)_i$ can be obtained using a lookup table (LUT). In the *ScaleDown* sub-operation, the fractional range is $[-1/2, 1/2)$, so the 33-bit MAC result is preserved and then reduced to 30-bit. The architecture of a Type-III PE is shown in Fig. 12(b).

The entire RNS conversion operation is executed in a fully pipelined flow. Each parallel path in the sub-operation takes turns reading data from $k$ banks and then writing back in place. The utilization of 2-port SRAM guarantees conflict-free data access.

### C. MAPPING OF CMUL OPERATION

The CMul operation is executed on the accelerator following the flow in Fig. 3. After excluding RNS conversion and NTT/INTT, the remaining parts include the $ct'_\times$ calculation and the *relinearization* step.

The $ct'_\times$ calculation is utilized to calculate the multiplication between polynomial coefficients. The calculation is achieved using the Polynomial coefficient multiplication instruction, sent by the PolyMul sub-controller to the Type-I PEs. With $2k$ Type-I PEs, all computations involving the $8k$ polynomials are carried out in a single sub-operation. Within each Type-I PE, four modular multipliers and one modular adder are utilized to complete the pipelined computation of $\{ct_0[0] \cdot ct_1[0], ct_0[0] \cdot ct_1[1] + ct_0[1] \cdot ct_1[0], ct_0[1] \cdot ct_1[1]\}$.

The *relinearization* step can be regarded as an instantiation of key switching, where the objective is to transform the $sk^2$ term to $sk$. First, the $sk^2$ term should be decomposed. Following the RNS instantiation of BV technique [25], the radix-based decomposition is changed to residue-based decomposition. To reduce computational complexity, the following polynomial multiplication computations need to be performed by NTT. To broadcast a polynomial $x$ over $R_{q_i}$ to $k$ Type-I PEs and perform NTT over $R_{q_0}$ to $R_{q_{k-1}}$, the desired

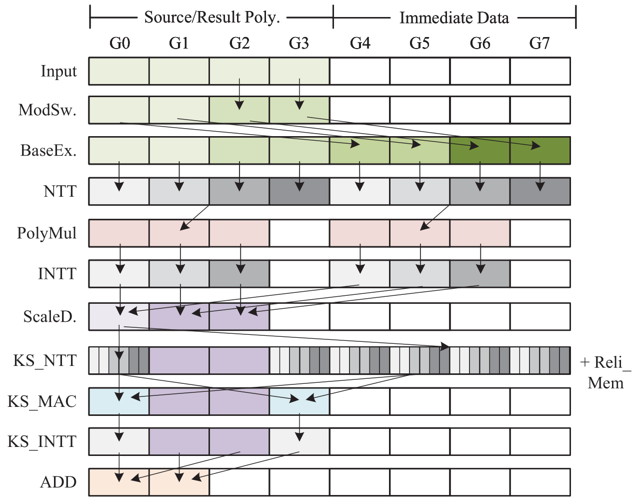


**FIGURE 13.** Memory access strategy.

decomposed polynomials of $x$ can be obtained. By employing $2k$ Type-I PEs in this manner, we can execute the process $\lfloor k/2 \rfloor$ times, obtaining $k^2$ decomposed polynomials denoted by $decomp_y$. The inner product between $decomp\_y$ and $rlk_i[0]$, $rlk_i[1]$ is performed after NTT, employing $2k$ Type-I PEs reconfigured as MAC mode. Finally, modular addition between the resulting polynomials and the remaining $sk^0$ and $sk^1$ terms is performed using $2k$ Type-I PEs reconfigured as modular addition mode.

During the homomorphic multiplication procedure, the in-place access strategy is adopted to maximize memory utilization. The access strategy is demonstrated in Fig. 13, where bank groups ($G_i$) with the same color indicate that they are accessed simultaneously. It can be observed that the bank group used for storing source and result polynomials is efficiently utilized most of the time, and another bank group implemented for storing the immediate data is accessed when the source polynomials require extension or decomposition. This efficient data access strategy maximizes the utilization of memory and contributes to the overall area efficiency improvement of the accelerator.

### D. MAPPING OF ROTATION OPERATION

The rotation operation consists of two steps: automorphism and key switching, as detailed in Algorithm 1. To reduce the number of automorphisms, we choose to perform automorphism (denoted by *auto*()) before the key switching sub-operation, rather than after word decomposition as in Gazelle [8]. The purpose of automorphism is to rearrange the positions of the slots within a polynomial. Each original slot indexed $i$ is transformed into a new slot indexed by $I$ through the transformation $I = [i \cdot 5^r]_n$ where $r$ denotes the rotation amount. The sign bit of each slot is changed according to (11). Nevertheless, the possibility of arbitrary position permutation is not favorable for conflict-free access. To achieve better throughput, we choose to perform the index transformation using row and column transformations based

**Algorithm 1** Rotation Operation

```
Input: Input polynomials $\{x_0, \dots x_{k-1}\}, i \in [0, k)$,
       polynomial length $n$, RNS basis $\{q_0, \dots q_{k-1}\}$,
       rotation amount $r$, rotation key $rk$
Output: Rotated polynomials $\{y_0, \dots y_{k-1}\}$
1  //Step 1: automorphism
2  for $i \leftarrow 0$ to $k-1$ do
3      $auto\_y[0][i] \leftarrow ntt(auto(x_i[0], r), q_i)$;
4      $auto\_y[1][i] \leftarrow auto(x_i[1], r)$;
5      $y_i[0] \leftarrow auto\_y[0][i]$;
6      $y_i[1] \leftarrow \{0, \dots, 0\}$;
7  //Step 2: key switching
8  for $i \leftarrow 0$ to $k-1$ do
9      for $j \leftarrow 0$ to $k-1$ do
10         $decomp\_y[i][j] \leftarrow ntt(auto\_y[1][i], q_j)$;
11 for $\alpha \leftarrow 0$ to $k-1$ do
12     //RNS-based decomposition
13     $y_\alpha[0] \leftarrow [y_\alpha[0] + \sum_{j=0}^{k-1} rk_0[j][\alpha] \cdot decomp\_y[j][\alpha]]_{q_i}$;
14     $y_\alpha[1] \leftarrow [y_\alpha[1] + \sum_{j=0}^{k-1} rk_1[j][\alpha] \cdot decomp\_y[j][\alpha]]_{q_i}$;
15 return $y_i$;
```

on the coefficient storage format. The transformation process is completed during data transmission.

$$sgn = \begin{cases} -1, & [i \cdot 5^r]_{2n} > n, \\ 1, & [i \cdot 5^r]_{2n} < n. \end{cases} \tag{11}$$

Based on the storage strategy proposed in this paper, each polynomial to be rotated is stored sequentially in four banks according to the left side of Fig. 10. Following the coordinate representation in Sec. IV-B1, the original index $i$ is represented by coordinates $(a, b)$, and the index $I$ after rotation is represented by coordinates $(A, B)$. The transformation from $i$ to $I$ can be computed as $I = A \cdot \frac{n}{4} + B = [(a \cdot \frac{n}{4} + b) \cdot 5^r]_n$. Let $S = \frac{n}{4}$, using the lemma in [40], the computation of the new coordinates $(A, B)$ is performed as described in (12).

$$\begin{cases} A = \left\lfloor \frac{I}{S} \right\rfloor = \left[ a + \left\lfloor \frac{b \cdot [5^r]_n}{S} \right\rfloor \right]_4, \\ B = [I]_S = [b \cdot [5^r]_n]_S. \end{cases} \tag{12}$$

The vertical coordinate $B$ is solely dependent on $b$, and the horizontal coordinate $A$ is obtained by adding a bias related to $b$ to the original $a$. Furthermore, the computation of the sign bit decision condition $[i \cdot 5^r]2n$ can be transformed into the computation of $[(a \cdot S + b) \cdot 5^r]_{2n}$. Notably, all operations, except for the computation of $b \cdot 5^r$, can be easily accomplished using shifting operations. In the pipelined data access process, where $b$ increases every CC, the result of $b \cdot 5^r$ can be obtained through accumulation. Therefore, the automorphism sub-operation can be efficiently integrated into the pipelined data access process, with only a negligible increase in pipeline setup time.

After automorphism, the key switching is performed. Following Step 2 in Algorithm 1, the key switching step involves $k/2$ NTT sub-operations, one MAC sub-operation, one INTT sub-operation, and one polynomial addition sub-operation, which is consistent with the *relinearization* step. Both the key switching step and the *relinearization* step utilize the same memory space, enabling efficient utilization of resources.

### E. MAPPING OF OTHER OPERATIONS

Except for the computationally intensive operations mentioned earlier, other sub-operations supported by our accelerator are implemented as follows.

#### 1) PMUL OPERATION

The $pt\text{-}ct$ multiplication needs to perform $\{pt \cdot ct[0], pt \cdot ct[1]\}$, i.e., two multiplications between $k$ sets of polynomials. The calculation can be seen as a variant of step 1 in the CMul operation. First, the three input polynomials are transformed into NTT form by invoking the NTT sub-operation twice. Then, the coefficient-wise multiplication is performed using the modular multiplication instruction. Finally, the result is obtained by executing the INTT sub-operation once.

#### 2) CADD/PADD OPERATION

Homomorphic addition, which includes the CAdd and the PAdd operations, requires only coefficient-wise addition between two polynomials. The way is consistent with the addition operation in the last step of key switching. Therefore, no separate control logic and storage resources are needed when configuring homomorphic addition. The FSM can directly jump to the last step of key switching to implement homomorphic addition.

## V. IMPLEMENTATION AND EVALUATION

### A. IMPLEMENTATION RESULTS

In this paper, we adopt a parameter set with specifications similar to those in [22] for the implementation of our proposed accelerator. The ciphertext modulus $q$ has a bit-width of 180-bit and is represented as the product of six 30-bit primes. The degree of the polynomial $n$ is set to 4096. The difference between our design and [22] lies in the construction of the extended modulus $Q$, which is formed by $q$ and an additional six 30-bit primes (instead of seven) whose product is denoted by $p$. The bit-width of the modular computing units and the memory banks is fixed as 30-bit. The twelve required primes are obtained from NFLlib [12] and follow the form $2^{30} - m \cdot 2^{14} + 1$ ($m \in \mathbb{Z}$). The given parameter set ensures an acceptable multiplication depth to support privacy-friendly applications as demonstrated in [22].

The components of our accelerator are verified and implemented on the Xilinx Virtex-7 and UltraScale+

**TABLE 3.** Implementation results of ntt and polynomial multiplication operations on FPGA platform.

| Design | TVLSI'19 [28] | TCASII'23 [27] | Our Work | TC'20 [17] | TCASI'22 [29] | TCASI'22 [23] | Our Work |
|---|---|---|---|---|---|---|---|
| Scheme | BFV | LBC | LHE | LBC | LBC | LHE | LHE |
| Platform | VIRTEX-7 | | | VIRTEX-7 | | | |
| $n$ | 1024 | | | 4096 | | | |
| $q$ (bit) | $32^a$ | $32^a$ | $30^a$ | $60^a$ | $32^b$ | $30^a$ | $30^a$ |
| Butterfly | Radix-32 | 8×Radix-2 | Radix-4 | Radix-2 | 8×Radix-2 | 4×Radix-2 | Radix-4 |
| LUT | 67K | 9.5K | 2.6K | 2.7K | 12.2K | 5.8K | 2.6K |
| REG | - | 4.7K | 3.0K | - | 10.3K | 4.3K | 3.0K |
| DSP | 599 | 64 | 44 | 31 | 48 | 40 | 44 |
| BRAM | 129 | 12 | 3.5 | 180 | 24 | 40 | 8.5 |
| Frequency (MHz) | 200 | 244 | 230 | 125 | 217 | 250 | 230 |
| NTT Lat. ($\mu$s / CCs) | 0.7<br>140 | 2.67<br>652 | 6.7<br>1,550 | 197.6<br>24,708 | 14.2<br>3,086 | 24.58<br>6,144 | 27.3<br>6,273 |
| Poly. Mul. Lat. ($\mu$s / CCs) | $2.1^+$<br>$420^+$ | - | 21.4<br>4,915 | - | 45.03<br>9,790 | - | 99.7<br>22,935 |
| LUT ATP NTT / Poly. Mul. | 48,026<br>144,077 | 25,974<br>- | 17,500<br>55,897 | 171,212<br>- | 239,029<br>543,129 | 146,693<br>- | 71,854<br>262,410 |
| BRAM ATP NTT / Poly. Mul. | 90.3<br>270.9 | 32.1<br>- | 23.5<br>74.9 | 35,568<br>- | 340.8<br>1080.7 | 983.2<br>- | 232.1<br>847.5 |

$^a$ The design supports multiple $q$ with general form. $^b$ The design supports multiple $q$ with specific form.
$^+$ The coefficient-wise multiplication operation is not included.

**TABLE 4.** Implementation results.

| Used by | LUT | REG | DSP | BRAM |
|---|---|---|---|---|
| Accelerator | 133,141 | 70,344 | 600 | 318 |
| Computing Unit | 57,640 | 38,471 | 600 | 0 |
| Sub-controller | 23,159 | 27,154 | 0 | 0 |
| Data Fabric | 34,719 | 2,790 | 0 | 0 |
| Param Buffer | 1,466 | 464 | 0 | 54 |

FPGA platforms for comparison with state-of-the-art works. To evaluate the performance of the accelerator, we select several key operations and homomorphic multiplication as benchmarks.

The complete implementation of our accelerator on the Virtex UltraScale+ MPSoC ZCU102 platform achieves a maximum clock frequency of 185 MHz. The resource consumption of key components is presented in Table 4. It is observed that the computing units and sub-controllers consume the majority of the resources. Additionally, the data fabric is responsible for facilitating the data distribution between the memory and the computing unit based on the instructions from the sub-controllers, therefore also consumes a relatively large amount of LUTs.

## B. PERFORMANCE OF KEY OPERATIONS

### 1) NTT ACCELERATION

NTT is not only the most frequently invoked sub-operation in homomorphic evaluation but also a core operation in other lattice-based cryptographic (LBC) schemes. Therefore, there have been many studies focusing on optimizing the hardware implementation of NTT. To better demonstrate the improvement achieved by the proposed NTT implementation approach, we implement an NTT design for a single polynomial that includes the NTT controller, one Type-I PE, and corresponding storage units on the Virtex-7 FPGA platform. The comparison of our work and the state-of-the-art works [17], [23], [27], [28], [29] in terms of area and performance is given in Table 3. Although there are other works [38], [43] that have demonstrated significant performance improvements in NTT, they mainly focus on fixed and smaller $q$, such as 12289, where the modular multiplier inside can be optimized specifically. For a fair comparison, these works have not been included in Table 3. In addition to evaluating the NTT latency, we also provide the latency of NTT-based polynomial multiplication, which involves two NTT operations, one INTT operation, and one coefficient-wise multiplication, which is provided in Table 3. The reduced implementation complexity allows the NTT design to achieve a higher maximum frequency of up to 230 MHz, surpassing the frequency of the entire accelerator.

We implement the NTT design to support two different polynomial degrees, $n = 1024, 4096$. The theoretical latencies of our design for NTT and polynomial multiplication are given by $n/4 \cdot \log_4 n$ and $3n/4 \cdot \log_4 n + n/4$, respectively. This design approach based on radix-4 butterfly units offers inherent advantages in terms of throughput and area efficiency when compared to other single-way designs [17]. To provide a clearer comparison, we introduce the concept of area-time product (ATP). Since the optimization of memory access strategy plays a crucial role in NTT optimization, we evaluate NTT designs using LUT ATP and BRAM ATP, which denotes the number of occupied LUTs and BRAM usage $\times$ the latency ($\mu$s) respectively.

For $n = 1024$, the work [28] adopts the radix-32 butterfly unit, which provides a notable advantage in latency. However, it is limited in that it can only support polynomial degrees that are powers of 32 and has a more complicated data access pattern, as reflected in the BRAM ATP. The recent in-place NTT design [27] employs an 8-way parallel radix-2 butterfly unit, which results in a higher data requirement per CC compared to our design. Therefore, the control logic of design [27] is more complex than ours, which leads to higher LUT ATP and BRAM ATP.

As $n$ increased to 4096, AC-PM [29] saves the DSP resources by optimizing the modular multiplication for certain types of moduli, but the 8-way parallelism leads to higher LUT and BRAM consumption. Since the resource of 4-way radix-2 butterfly unit is comparable to a radix-4 butterfly unit, our implementation results are close to ReMCA [23] but achieve lower LUT and BRAM consumption. The latency of ReMCA presented in [23] excludes the time for reading or storing data and pipeline establishment, giving ReMCA a slight advantage in terms of latency. Overall, our design achieves 2.04×-3.33× and 1.47×-153.2× improvements in LUT and BRAM efficiency compared to the latest designs, respectively.

### 2) AUTOMORPHISM ACCELERATION

The rotation operation plays a crucial role in homomorphic encrypted applications such as private-preserving machine learning [8]. However, the automorphism operation involved in rotation introduces irregular permutations among $n$ coefficients, which increases the implementation complexity. To the best knowledge of the authors, most existing BFV-based homomorphic evaluation accelerators [21], [22], [23] exclude the implementation of rotation. The BFV-based homomorphic convolution accelerator [45], which pipeline implements automorphism for the non-RNS-form polynomials based on the same scheme as Gazelle, can efficiently reduce rotation time. Nevertheless, the crossbar network size in the address mapper grows squarely with the increase in residues. Optimized implementations of automorphism are explored more in the CKKS-based homomorphic evaluation accelerators. Among the recent CKKS-based homomorphic evaluation accelerators, Poseidon [40] and FAB [46] designed a 4-stage automorphism unit with extra memory, while BTS [47] supported index transformation using a network-on-chip architecture between processing elements. Compared with the previous strategies, we aim to minimize implementation overhead.

We extract the automorphism unit from our accelerator and implement it using the same parameter sets as Poseidon ($q$ = 32 bit, $n$ = 65536). The resource usage comparison is listed in Table 5. The straightforward design, referred to as Auto, has the lowest resource consumption but can

**TABLE 5.** Resource usage comparison of automorphism unit.

| Design | Resource | | | | Latency |
|---|---|---|---|---|---|
| | LUT | REG | DSP | BRAM | (CCs) |
| Auto [40] | 0 | 88 | 0 | 0 | $131,073$ |
| HFAuto [40] | $25,751$ | 572 | 0 | 512 | 517 |
| Our Work | 425 | 290 | 0 | 0 | $4^{+}$ |

$^{+}$ The automorphism latency is hidden in the on-chip data transmission latency.

**TABLE 6.** Performance of each sub-operation.

| Operation | Latency (CCs) | Max. Input ({Poly}) |
|---|---|---|
| *ModSw.* | $6,188$ | $2k$ |
| *BaseEx.* | $6,188$ | $2k$ |
| NTT | $6,273$ | $2k$ |
| Coefficient Mul. | $4,116$ | $4 \cdot 2k$ |
| INTT | $6,273$ | $2k$ |
| *ScaleD.* | $6,183$ | $2 \cdot 2k$ |
| KS_MAC | $6,153$ | $2k$ |
| Add | $1,030$ | $2k$ |

only process one index in a single CC. Poseidon proposes an improved design called HFAuto, which utilizes a large first-in-first-out (FIFO) buffer to store intermediate results during coordinate transformation, resulting in significant performance improvement. In our work, we perform the coordinate transformation during data transmission, allowing the latency of automorphism to be hidden within the data transmission latency between SRAM and the computing unit. Since the loading operation is pipelined and the automorphism unit introduces only 4 levels of pipeline, the arrival of data is only delayed by 4 CCs. Furthermore, our approach leads to a significant decrease in resource consumption.

## C. PERFORMANCE OF HOMOMORPHIC EVALUATION

This experiment evaluates the performance of the full system when executing the homomorphic evaluation. The performance of basic sub-operations that constitute complex homomorphic evaluation computations is shown in Table 6, where the Max. Input denotes the maximum number of polynomials supported by a single sub-operation.

By reusing basic operations, our accelerator is able to perform complex homomorphic evaluation computations, and the resulting performance is illustrated in Fig. 14. The overall latency excludes the time required for off-chip data transmission but includes all time required for on-chip data transmission. Specifically, the homomorphic multiplication operation takes $111,240$ CCs, the homomorphic addition operation takes $1,031$ CCs, and the homomorphic rotation operation takes $32,287$ CCs. The evaluation also indicates that the NTT and INTT sub-operations are the most frequently invoked sub-operations during homomorphic evaluation.

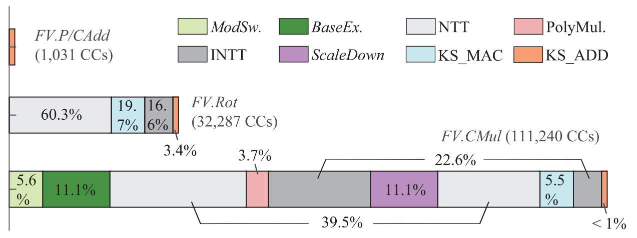


**FIGURE 14.** Performance of homomorphic evaluation operations.

**TABLE 7.** Implementation results of homomorphic multiplication operation on FPGA platform.

| Design | HPCA'19 [21] | TC'20 [22] | Our Work |
|---|---|---|---|
| Platform | UltraScale+ | | |
| Frequency (MHz) | 200 | 200 | 185 |
| LUT/ATP | 63,522<br>283,308 | 57,877<br>251,186 | 133,141<br>80,018 |
| REG/ATP | 25,622<br>114,274 | 25,648<br>111,312 | 70,344<br>42,277 |
| DSP/ATP | 208<br>927.7 | 208<br>902.7 | 600<br>360.6 |
| BRAM/ATP | 388<br>1,730.5 | 305$^{+}$<br>1,323.7 | 318<br>191.1 |
| Latency (ms) | 4.46(7.42×) | 4.34(7.22×) | 0.601 |

$^{+}$ 249 BRAMs + 56 URAMs.

In Table 7, we compare our accelerator with other existing BFV-based HE accelerators under the same parameter set when performing homomorphic multiplication. We include the implementation results of a coprocessor from the literatures [21] and [22] for reference. The comparisons with another BFV-based HE accelerator [23] are excluded in this subsection as only synthesis results were provided in [23].

The HEAWS accelerator [22] is an extended version of the work [21], utilizing the same architecture, which consists of seven residue polynomial arithmetic units, a 2-core NTT, and an RNS conversion unit in a single coprocessor. Since we adopt twelve Type-I PEs, the peak computing performance of our accelerator is nearly doubled, which is also reflected in resource consumption. Nevertheless, benefiting from the proposed efficiency reconfigurable architecture, our accelerator could achieve better area efficiency. When comparing the ATPs measured by LUT, REG, DSP, and BRAM, our accelerator achieves reductions of 3.1×, 2.6×, 2.5× and 6.9×, respectively, compared to HEAWS. Furthermore, to the best knowledge of the authors, our accelerator is the first BFV evaluation accelerator to support the HE rotation operation, offering possibilities for implementing a wider range of homomorphic applications.

### D. COMPARISON WITH THE OTHER RECONFIGURABLE ACCELERATOR

Both our accelerator and ReMCA [23] employ reconfigurable architectures to improve resource utilization. However, there are differences in the design approach. ReMCA utilizes a homogeneous 7×8 computing array, which simplifies control logic but may not be optimal for specific operations. In contrast, our accelerator designs a heterogeneous computing unit consisting of twelve Type-I PEs and other specialized arithmetic units, tailored to the characteristics of different operations. Table 8 presents the resource consumption of two accelerators. The resource consumption of RCUs in ReMCA is obtained by multiplying the synthesis results of a single PE by 56. Note that ReMCA only reports the theoretical latency, excluding the latency associated with data and instruction transmission, and the control logic is simpler than ours. To make a fair comparison, we mainly compare the resource consumption of RCUs. With similar LUT/DSP resource consumption, our accelerator achieves an 8% reduction in latency and an 18% reduction in REG consumption compared to ReMCA. Moreover, the decimal pre-computed parameters stored as constant reciprocals are kept to a precision of 89-bit after the decimal point, which is 24-bit larger than ReMCA. The area of Type-III PEs that perform the integer arithmetic could be further reduced when adopting the 65-bit parameters. Furthermore, benefiting from the efficient memory access strategy, our design consumes 57.6% fewer BRAMs than ReMCA.

**TABLE 8.** Resource usage comparison of different reconfigurable architecture.

| Design | Resource<br>LUT/REG/DSP/BRAM | Error | Latency<br>(CCs) |
|---|---|---|---|
| ReMCA [23]<br>RCU<br>Others | 64,057/49,307/560/552<br>58,408/45,416/560/-<br>5,649/3,891/560/- | $2^{-53}$ | 120,832<br>(1.08×) |
| Our Work<br>RCU<br>Type-I PE<br>Type-II PE<br>Type-III PE | 133,141/70,344/600/318<br>57,640/38,471/600/-<br>44,281/19,495/464/-<br>3,768/3,911/88/-<br>9,591/15,065/48/- | $2^{-80}$ | 111,240 |

## VI. CONCLUSION

In this paper, we present a heterogeneous reconfigurable accelerator for homomorphic evaluation on encrypted data based on the RNS variant of the BFV scheme. The flexible and reconfigurable modular computing units greatly improve the resource utilization of the accelerator. Specialized optimizations for key operations and memory access strategy reduce the processing latency. When $n = 4096$, our design achieves 1.08 to 7.42 times reduction in latency with higher area efficiency compared to other BFV accelerators when performing homomorphic multiplication. The low-latency and area-efficient features can enhance the viability of privacy computing. In the future, we will continue to explore the deployment of privacy computing applications in the cloud.

## REFERENCES

[1] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Found. Secure Comput.*, vol. 4, no. 11, pp. 169–180, 1978.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, May 2009, pp. 169–178.

[3] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. 32nd Annu. Cryptol. Conf. Adv. Cryptol.*, Aug. 2012, pp. 868–886.

[4] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, Mar. 2012.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Leveled fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, 'Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Intell. Syst. Commun., IoT Secur. (ICISCoIS)*, Feb. 2023, pp. 409–437.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[8] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Jan. 2018, pp. 1651–1669.

[9] W. Wu, J. Liu, H. Wang, J. Hao, and M. Xian, "Secure and efficient outsourced k-Means clustering using fully homomorphic encryption with ciphertext packing technique," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 10, pp. 3424–3437, Oct. 2021.

[10] S. Bian, D. E. S. Kundi, K. Hirozawa, W. Liu, and T. Sato, "APAS: Application-specific accelerators for RLWE-based homomorphic linear transformations," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4663–4678, 2021.

[11] *Microsoft SEAL(Release 4.1)*. Microsoft Research. Redmond. WA. Accessed on: Jan. 28, 2023. [Online]. Available: https://github.com/Microsoft/SEAL

[12] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, and T. Lepoint, "NFLLIB: NTT-based fast lattice library," in *Proc. Cryptographers' Track RSA Conf.*, Mar. 2016, pp. 341–356.

[13] A. Al Badawi et al., "OpenFHE: Open-source fully homomorphic encryption library," in *Proc. 10th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, Nov. 2022, pp. 53–63.

[14] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, no. 114, pp. 365–374, 1971.

[15] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full RNS variant of FV like somewhat homomorphic encryption schemes," in *Proc. Int. Conf. Sel. Areas Cryptogr.*, Aug. 2016, pp. 423–442.

[16] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "FPGA-based accelerators of fully pipelined modular multipliers for homomorphic encryption," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2019, pp. 1–8.

[17] A. C. Mert, E. Karabulut, E. Öztürk, 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.

[18] A. A. Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 2, pp. 70–95, May 2018.

[19] A. S. Özcan, C. Ayduman, E. R. Türkoglu, and E. Savas, "Homomorphic encryption on GPU," *IEEE Access*, vol. 11, pp. 84168–84186, 2023.

[20] S. Sinha Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation," *IEEE Trans. Comput.*, vol. 67, no. 11, pp. 1637–1650, Nov. 2018.

[21] S. Sinha Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 387–398.

[22] F. Turan, S. S. Roy, and I. Verbauwhede, "HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA," *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1185–1196, Aug. 2020.

[23] Y. Su, B.-L. Yang, C. Yang, and S.-Y. Zhao, "ReMCA: A reconfigurable multi-core architecture for full RNS variant of BFV homomorphic evaluation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2857–2870, Jul. 2022.

[24] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H. S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 26–39.

[25] S. Halevi, Y. Polyakov, and V. Shoup, "An improved RNS variant of the BFV homomorphic encryption scheme," in *Proc. Cryptographers' Track RSA Conf.*, 2019, pp. 83–105.

[26] A. Kim, Y. Polyakov, and V. Zucca, "Revisiting homomorphic encryption schemes for finite fields," in *Proc. Adv. Cryptol.–ASIACRYPT*, Dec. 2021, pp. 608–639.

[27] Y. Geng, X. Hu, M. Li, and Z. Wang, "Rethinking parallel memory access pattern in number theoretic transform design," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 5, pp. 1689–1693, May 2023.

[28] A. C. Mert, E. Öztürk, and E. Savas, "Design and implementation of Encryption/Decryption architectures for BFV homomorphic encryption scheme," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 353–362, Feb. 2020.

[29] 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.

[30] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Proc. Adv. Cryptol.—EUROCRYPT*, vol. 7237. Berlin, Germany: Springer, 2012, pp. 465–482.

[31] A. Al Badawi, Y. Polyakov, K. M. M. Aung, B. Veeravalli, and K. Rohloff, "Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 941–956, Apr. 2021.

[32] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. Adv. Cryptol.-EUROCRYPT*, 2010, pp. 1–23.

[33] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, 1965.

[34] W. M. Gentleman and G. Sande, "Fast Fourier transforms: For fun and profit," in *Proc. Fall Joint Comput. Conf. XX-AFIPS (Fall)*, 1966, pp. 563–578.

[35] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *Proc. 16th Int. Workshop Cryptogr. Hardw. Embedded Syst.*, 2014, pp. 371–391.

[36] T. Pöppelmann, T. Oder, and T. Güneysu, "High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers," in *Proc. Int. Conf. Cryptol. Inf. Secur. Latin Amer.*, 2015, pp. 346–365.

[37] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 2, pp. 49–72, Mar. 2020.

[38] X. Chen, B. Yang, S. Yin, S. Wei, and L. Liu, "CFNTT: Scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, no. 1, pp. 94–126, Nov. 2021.

[39] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *Proc. MICRO-54, 54th Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2021, pp. 238–252.

[40] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical homomorphic encryption accelerator," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 870–881.

[41] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, "REVAMP: A systematic framework for heterogeneous CGRA realization," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Feb. 2022, pp. 918–932.

[42] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Proc. Conf. Theory Appl. Cryptograph. Techn.*, 1986, pp. 311–323.

[43] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 4, pp. 17–61, Aug. 2019.

[44] U. Banerjee, A. Pathak, and A. P. Chandrakasan, "2.3 an energy-efficient configurable lattice cryptography processor for the quantum-secure Internet of Things," in *Proc. IEEE Int. Solid-State Circuits Conf.-(ISSCC)*, Feb. 2019, pp. 46–48.

[45] X. Hu, M. Li, J. Tian, and Z. Wang, "Efficient homomorphic convolution designs on FPGA for secure inference," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 11, pp. 1691–1704, Nov. 2022.

[46] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 882–895.

[47] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "BTS: An accelerator for bootstrappable fully homomorphic encryption," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022, pp. 711–725.

**WENQING SONG** received the B.S. degree from the School of Electronic Science and Engineering, Southeast University (SEU), Nanjing, China, in 2017. She is currently pursuing the Ph.D. degree in electronic science and engineering with Nanjing University (NJU), Nanjing. Her current research interests include polar coding algorithms, homomorphic encryption, and efficient reconfigurable architecture.

**SIRUI SHEN** received the B.S. and M.S. degrees from the School of Electronic Science and Technology, Nanjing University (NJU), Nanjing, China, in 2020 and 2023, respectively. He is currently pursuing the Ph.D. degree in computer security with Centrum Wiskunde & Informatica (CWI Amsterdam). His current research interests include hardware security, hardware implementation for cryptography, and cloud FPGA security.

**CONGWEI XU** received the B.S. and M.S. degrees from the School of Electronic Science and Engineering, Nanjing University (NJU), Nanjing, China, in 2020 and 2023, respectively. Her current research interests include homomorphic encryption, post quantum cryptography, and efficient reconfigurable architecture.

**YILIN WANG** received the B.S. degree from the School of Electronic Science and Engineering, Nanjing University (NJU), Nanjing, China, in 2021. He is currently pursuing the M.S. degree in electronic science and engineering with NJU. His current research interests include homomorphic encryption, post quantum cryptography, and efficient reconfigurable architecture.

**XINYU WANG** received the B.S. and M.S. degrees in integrated circuit engineering from the Hefei University of Technology, in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree in electronic science and technology with the School of Electronic Science and Engineering, Nanjing University. His current research interests include hardware security, reconfigurable technology, and cryptography.

**YUXIANG FU** (Member, IEEE) received the B.S. degree in microelectronics and solid state electronics and the Ph.D. degree in electronic science and technology from Nanjing University, Nanjing, China, in 2013 and 2018, respectively.

In 2018, he joined the School of Electronic Science and Engineering, Nanjing University. He is currently an Assistant Professor with the School of Integrated Circuits, Nanjing University. His current research interests include AI for chip architecture design, network-on-chip algorithms/architectures, low-power digital systems, and 3D IC design.

**LI LI** received the B.S. and Ph.D. degrees from the Hefei University of Technology, Hefei, China, in 1996 and 2002, respectively. She is currently a Professor with the VLSI Design Institute, School of Electronic Science and Engineering, Nanjing University, Nanjing, China. Her current research interests include VLSI design for digital signal processing systems, reconfigurable computing, and multiprocessor system-on-a-chip (MPSoC) architecture design methodology. She is a member of the Circuits and Systems for Communications (CASCOM) TC of the IEEE CAS Society.

**ZHONGHAI LU** (Senior Member, IEEE) received the B.Sc. degree from Beijing Normal University, in July 1989, the M.Sc. and Ph.D. degrees from the KTH Royal Institute of Technology, Sweden, in June 2002 and March 2007, respectively, and the M.B.A. degree in innovation and growth from the University of Turku, Finland, in October 2012.

He is currently a Full Professor of electronic systems design with the School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology. His research interests include system-on-chip (SoC)/network-on-chip (NoC), embedded systems, computer architecture, and artificial intelligence. He has published more than 210 articles in these areas. His research papers were nominated as the Best Paper Candidate from HPCA 2018, DATE 2003, ICCAD 2009, and NOCS 2013. He received the Best Paper Award from NOCS 2015.