IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 21 March 2024, accepted 4 April 2024, date of publication 10 April 2024, date of current version 18 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3386977

RESEARCH ARTICLE

A Bootstrapping-Capable Configurable NTT Architecture for Fully Homomorphic Encryption

RELLA MARETA, ARDIANTO SATRIAWAN, PHAP NGOC DUONG, AND HANHO LEE, (Senior Member, IEEE)
Department of Electrical and Computer Engineering, Inha University, Incheon 22212, South Korea

Corresponding author: Hanho Lee (hhlee@inha.ac.kr)

This work was supported in part by the Ministry of Science and Information and Communication Technology (MSIT), South Korea, through the Information Technology Research Center (ITRC) Support Program, supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2021-0-02052; in part by the National Research Foundation of Korea (NRF) grant funded by Korean Government through MSIT under Grant 2021R1A2C1011232; and in part by IITP grant funded by Korean Government through MSIT under Grant 20210007790012003.

ABSTRACT Fully homomorphic encryption (FHE) provides a solution to privacy-preserving applications because of its ability to perform computations on encrypted data without exposing raw data. However, FHE suffers from implementation bottlenecks owing to the large computations involved, particularly with bootstrapping. Bootstrapping is necessary in FHE to enable an unlimited number of multiplication. Nonetheless, implementing bootstrapping requires a significantly large polynomial length, $N = 2^{16}$ or 2^{17}, to considerably secure the system. Thus, polynomial multiplication will be challenging in terms of resources and time. This problem can be resolved by implementing the number theoretic transform (NTT) that can perform polynomial multiplication in quasi-linear complexity. However, designing an NTT architecture for FHE is challenging because it requires various parameters, particularly the high polynomial degree that will require a considerable amount of hardware resources and clock latency. This study proposes a design for FPGA implementation of the NTT architecture with flexible input lengths: 2^{16} and 2^{17} by combining radix-2 and radix-2^4. Twiddle factor generator (TFG) and efficient configurable modular multiplication (MM) unit are also utilized to achieve time and area-efficient architecture. The proposed design was synthesized on the FPGA Xilinx ALVEO U250 and demonstrated higher hardware efficiency and optimum latency that outperforms those reported in previous studies.

INDEX TERMS Homomorphic encryption, configurable architecture, lattice-based cryptography, number theoretic transform.

I. INTRODUCTION

The development of cloud computing has increased the significance of homomorphic encryption (HE) by providing enhanced data privacy and security. HE can protect sensitive information by allowing the data to be calculated in encrypted form without exposing the raw data [1]. This capability is applicable and important in many sectors that require data privacy as their main priority, such as healthcare [1], finances [2], communications [3], manufacturing [4], and many more.

The concept of HE was first proposed in the 1970s by Rivest et al. [5], [6], which envisioned a system that would allow both arithmetic addition and multiplication to be done on encrypted data without decrypting them first. However, for more than three decades following the initial proposal, it remained unclear whether the concepts of HE could be fully implemented [7].

After the proposal, many researchers proposed partially homomorphic encryption (PHE) schemes, which were only

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

capable of doing one type of operation – either addition or multiplication. Examples of well-known PHE schemes include RSA [8] and ElGamal [9] cryptosystems, which are capable of unbounded modular multiplications only; also, the Paillier [10], and Benaloh [11] cryptosystems, which are capable of unbounded modular additions only. The RSA cryptosystem is famously used for secure data transmission, and the Benaloh cryptosystem is widely used for electronic voting systems.

Gentry, in 2009, proposed a claimed "fully" homomorphic encryption using lattice-based cryptography, which is a remarkable breakthrough after three decades [12]. The scheme is based on learning with errors (LWE) problems and ring LWE (RLWE) [13]. The scheme can perform both addition and multiplication between ciphertext on the encrypted domain. However, as the ciphertext in the scheme has deliberately added a small noise, each time a multiplication is performed, the noise gets larger. Therefore, this proposed scheme is limited, as the noise of the ciphertext increases with each performed calculation. Eventually, the ciphertexts become indecipherable due to large noise levels.

This limited HE scheme is often referred to as *leveled* or *somewhat* homomorphic encryption (SHE), which means the scheme is usable up to a limited number of operations. Many leveled HE schemes were proposed during this period. Notable works include the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [14], LopezAlt-Tromer-Vaikuntanathan (LTV) scheme [15], Brakerski-Fan-Vercauteren (BFV) scheme [16], and Gentry-Sahai-Waters (GSW) scheme [17].

Many researchers then proposed to upgrade SHE to Fully homomorphic encryption (FHE) schemes, trying to eliminate the number of operation limitations. The ciphertext becomes indecipherable when the noise level is larger than desired after a certain number of operations. To overcome this problem, the ciphertext is homomorphically decrypted and then re-encrypted to reduce the noise level, thus resetting the operation limit count. This technique is called *bootstrapping*, which was also proposed by Gentry [12]. Implementing bootstrapping efficiently in various proposed schemes has been a hot topic for HE research.

Another breakthrough came in 2017 when Cheon et al. proposed a leveled FHE scheme that can perform arithmetic approximations on real and complex numbers data [18], which they named homomorphic encryption for the arithmetic of approximate numbers (HEAAN). This scheme was later known after the authors: the CKKS scheme, following the name convention of previous HE schemes.

The scheme supports the truncation of encrypted values and batch computation. One of the strong points of the CKKS scheme includes an efficient rescaling operation that scales down an encrypted message after a multiplication operation using the *key-switching* technique [19]. In other schemes, this would already require the bootstrapping technique. The key-switching implementation in the scheme itself is also a hot topic for many researchers [19], [20], [21]. A complete

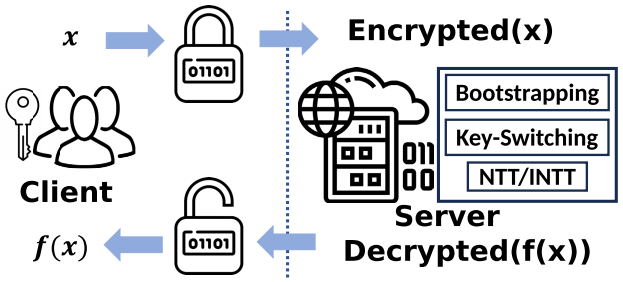

FIGURE 1. Fully homomorphic encryption scheme and the operations needed in the CKKS scheme: NTT/INTT, key-switching, and bootstrapping.

hardware implementation of a leveled version of the CKKS scheme is also one of the most researched topics [22], [23].

A challenge in classical CKKS lies in integer arithmetic utilizing an excessively large modulus Q to implement the scheme effectively. A practical workaround involves utilizing Residue Number Systems (RNS) as suggested in the next version of the CKKS scheme [24]. For the sake of simplification and differentiation, we refer to the revised scheme as RNS-CKKS. The solution involves factoring the thousands-bit modulus Q into smaller moduli $q_0, q_1, q_2, \ldots, q_L$, and making a modulus $Q := \prod_{i=0}^{L} q_i$ has $L - 1$ levels. However, implementing this workaround in hardware is also challenging because the modular multiplier and other arithmetic modules must handle diverse moduli q as inputs to achieve specific area and time performance.

Even though the CKKS scheme implements multiplication operations more efficiently. Eventually, the ciphertexts' noise also becomes greater than desired, making it indecipherable. Thus, bootstrapping is also required to reduce the noise level.

In CKKS, the implementation of bootstrapping is a trade-off between the security level and polynomial length. A lower security level can be implemented with a relatively short polynomial length; however, a higher security level ($\lambda = 128$-bit) requires a considerably long polynomial length. Moreover, since bootstrapping itself consumes L_{boot} levels, L should be significantly larger to require less frequent operations. Therefore, previous works observed and suggested that only high polynomial degrees NTT with $N = 2^{16}$ or $N = 2^{17}$ would make bootstrapping possible [25], [26].

However, the implementation of bootstrapping in CKKS still causes bottlenecks due to the expensive cost of HE arithmetic operations for decrypting and re-encrypting the ciphertext homomorphically. Therefore, many researchers [25], [26], [27], [28], [29] try to find workarounds in its implementation. Figure 1 shows the important building blocks for implementing FHE in the CKKS scheme.

The NTT [30], [31] can reduce the complexity of polynomial multiplication from quadratic $O(n^2)$ to quasi-linear $O(n \log n)$ by adapting the divide-and-conquer algorithm for the fast Fourier transform (FFT) [32] for NTT: the Cooley-Tukey (CT) [33] and Gentleman-Sande (GS) [34] butterfly algorithm for the forward and inverse transform, respectively.

In this paper, we focus on implementing the NTT part in such considerable lengths aforementioned

($N = 2^{16}$ or $N = 2^{17}$) that can support bootstrapping operation to enable indefinite computation on a CKKS ciphertext. However, implementing NTT directly in software is slow and not a straightforward task due to its complex architecture. Many researchers implement NTT in various platforms and technologies due to their advantages, such as implementing them in hardware partially or completely [35], using hardware supporting parallelism [36], or using vector-processing features [37]. Our work focuses on the FPGA implementation as a hardware accelerator.

We propose a design for FPGA implementation of the NTT architecture with flexible input lengths: 2^{16} and 2^{17}. Studies on NTT implementations that support large N up to 2^{16} are limited [38], [39], [40], [41], particularly studies supporting N up to 2^{17} in a single architecture. The contributions of this study are as follows:

1) We propose a configurable NTT architecture with an increased input length, 2^{16} and 2^{17}, that can be selected by configuring the input *sel* of the NTT to facilitate bootstrapping in the CKKS scheme, which requires increasing the polynomial order to be multiplied. Our NTT also utilized flexible modular multiplication (MM) units to accommodate any moduli q values required to support a high-security level.
2) We propose the combination of feed-forward 32-coefficient input radix-2 architecture and iterative 16-coefficient input radix-2^4 architecture to achieve optimum latency and hardware utilization. Our 2^{16} NTT architecture utilizes area optimum and conflict-free memory access calculated in four iterations. We extend this optimized architecture to support $N = 2^{17}$ without additional iterations for latency efficiency by adding one feed-forward radix-2 stage that only requires 11 CC for its operation.
3) We propose an on-the-fly twiddle-factor generator (TFG) to decrease memory utilization by approximately 96.9% compared with conventional ROM-based methods. Our NTT and TFG achieve better memory efficiency and higher throughput than most related studies. These contributions make our proposed design provide useful approaches for the development of NTT accelerator in particular or HE system in the future.

The remainder of the paper is organized as follows: Section II provides a background of the NTT and its equations. Section III explains the architecture of our proposed design and detail explanation of the system and sub-modules. Section IV provides the performance analysis and comparison with other studies. Finally, Section V concludes the paper.

II. PRELIMINARIES: THE NUMBER THEORETIC TRANSFORM (NTT)

While polynomial multiplication is a fundamental operation in cryptography, it is conventionally an expensive operation performed through schoolbook multiplication with $O(n^2)$ complexity. The NTT can be utilized to reduce the complexity to $O(n \log n)$, which is equivalent to the discrete Fourier transform (DFT) in the cyclotomic polynomial ring $\mathbb{Z}_q/\Phi(x)$.

When $\Phi(x) = x^n - 1$, the NTT requires the twiddle factor base to be the n-th root of unity of the ring $\mathbb{Z}_q$, ω, where $\omega \equiv 1 \bmod q$. This type of NTT is called *positively-wrapped convolution*. However, to multiply polynomials using this convolution type, zero-padding is needed.

Most cryptographic applications generally utilize $\Phi(x) = x^n + 1$, the n-th root of unity of the ring $\mathbb{Z}_q$, ω, where $\omega \equiv 1 \bmod q$, and ψ, the $2n$-th root of unity, as the TF base to avoid zero-padding. Where $\psi^2 \equiv \omega \bmod q$ and $\psi^n \equiv -1 \bmod q$. This type of polynomial multiplication based on the ring $\mathbb{Z}_q/(x^n + 1)$ is called *negatively-wrapped convolution*, which is the base of our design.

The forward and inverse transformation of NTT is expressed as [30] and [31]:

$$\mathrm{NTT}^{\psi}(\boldsymbol{a}) = \hat{\boldsymbol{a}}_j = \sum_{i=0}^{n-1} \omega^{ij}(a_i \psi^i) \mod q \tag{1}$$

While the inverse transformation of NTT is given by:

$$\mathrm{INTT}^{\psi^{-1}}(\hat{\boldsymbol{a}}) = \boldsymbol{a}_i = n^{-1} \sum_{j=0}^{n-1} \psi^{-j}(\omega^{-ij}\hat{a}_j) \mod q \tag{2}$$

The process of multiplying the vector by ψ^i in the forward transform is called *pre-processing* and by ψ^{-j} in the inverse transform is called *post-processing*. We can simplify the transformation by substituting the fact that $\psi^2 \equiv \omega \bmod q$ into equations (1) and (2):

$$\mathrm{NTT}^{\psi}(\boldsymbol{a}) = \hat{\boldsymbol{a}}_j = \sum_{i=0}^{n-1} \psi^{2ij+i} a_i \mod q \tag{3}$$

$$\mathrm{INTT}^{\psi^{-1}}(\hat{\boldsymbol{a}}) = \boldsymbol{a}_i = n^{-1} \sum_{j=0}^{n-1} \psi^{-(2ij+j)} \hat{a}_j \mod q \tag{4}$$

Note that these NTT-INTT pairs, when done as-is, still have a quadratic complexity $O(n^2)$. One can use the divide-and-conquer approach to reduce the complexity to quasi-linear, $O(n \log n)$. The main idea is to calculate the required values once but distribute the results to others instead of calculating the same value multiple times.

The well-known divide-and-conquer technique is the CT and GS algorithm. The CT butterfly algorithm utilizes a divide-and-conquer technique to divide the input vector recursively into two parts: odd and even. In contrast, the GS butterfly algorithm divides the input vector recursively into the upper and lower parts.

Both have a trade-off of input bit-reversal. The CT-type butterfly has normal-order input and bit-reversed-order output, whereas the GS-type butterfly has bit-reversed-order input and normal-order output. By utilizing CT for the forward transform and GS for the inverse transform, a reordering device is not required in the middle.

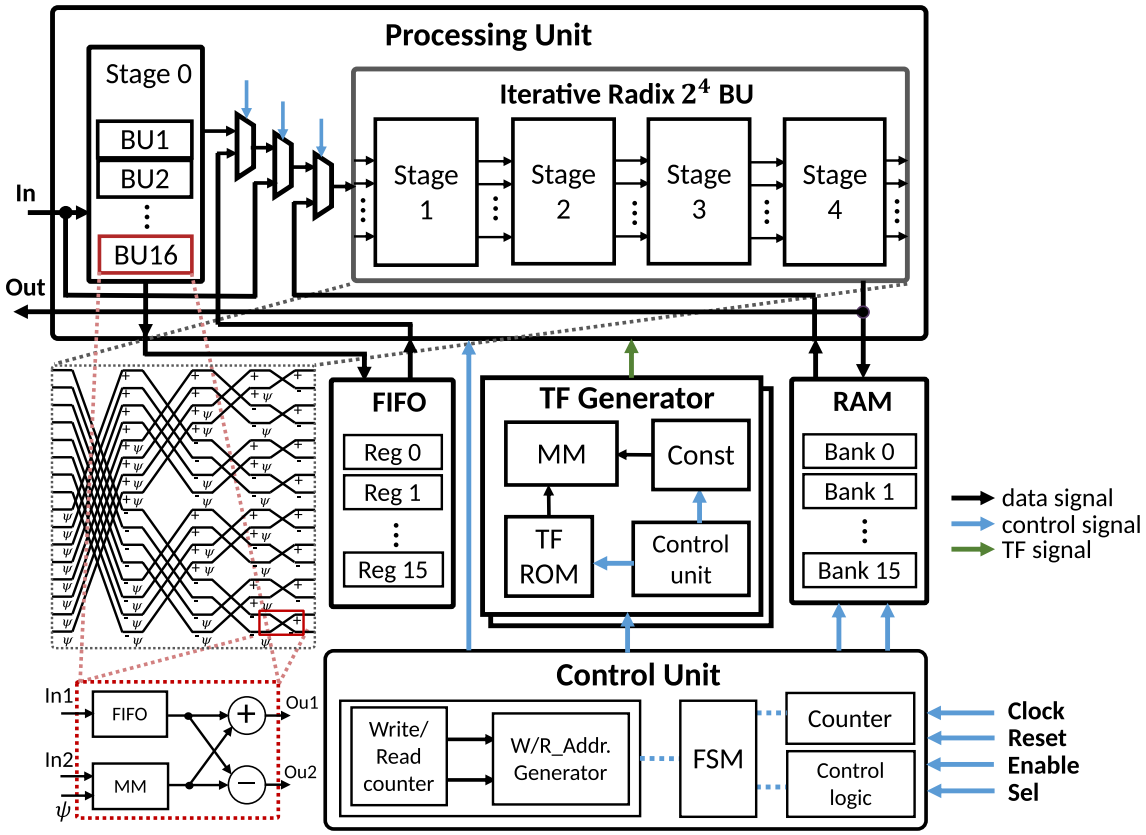

FIGURE 2. Top-level block diagram of configurable 2^{16} and 2^{17} NTT architectures.

III. PROPOSED NTT ARCHITECTURE

A. ARCHITECTURE OVERVIEW

We propose an NTT architecture with a configurable number of input coefficients, $N = 2^{16}$ or $N = 2^{17}$ by adapting the idea of configurable FFT [32]. For the input length $N = 2^{17}$, Equation (3) is expressed as:

$$\hat{\boldsymbol{a}}_j = \sum_{i=0}^{2^{17}-1} \psi^{2ij+i} a_i \bmod q \tag{5}$$

When split into lower and upper halves, the equation becomes:

$$\hat{\boldsymbol{a}}_j = \sum_{i=0}^{2^{16}-1} \psi^{2ij+i} a_i \bmod q + \sum_{i=2^{16}}^{2^{17}-1} \psi^{2ij+i} a_i \bmod q \tag{6}$$

From the equation, the radix-2^{17} NTT can be split into two radix-2^{16}. However, the iterator and exponentiation of ψ differ significantly in both summations.

This can be rewritten as a double summation:

$$\hat{\boldsymbol{a}}_j = \sum_{m_1=0}^{2^{16}-1} \sum_{m_0=0}^{2^1-1} \boldsymbol{\psi}^k \boldsymbol{a}_n \bmod q \tag{7}$$

where $n = m_1 + 65536m_0$ and $k = n(2j+1)$.

The same principle can be applied to the inside summation, dividing it into four parts:

$$\hat{\boldsymbol{a}}_j = \sum_{m_4=0}^{15} \sum_{m_3=0}^{15} \sum_{m_2=0}^{15} \sum_{m_1=0}^{15} \sum_{m_0=0}^{1} \boldsymbol{\psi}^k \boldsymbol{a}_n \bmod q \tag{8}$$

where $n = m_4 + 16m_3 + 256m_2 + 4096m_1 + 65536m_0$ and $k = n(2j+1)$. Equation (8) is fundamental to our NTT architecture.

Based on Equation (8), we propose Algorithm 1 to support the configurability of our NTT. Function BitReverse(u, v) generates the bit-reversed order of the input u in size of v bits. We separate $\log_2 N$ stages of radix-2 into stage 0 and iterative units. The iterative unit comprises four stages of radix-2. It can also be called radix-2^4 and runs in four iterations, resulting in 16 stages in total. Suppose $sel = 1$ or $N = 2^{17}$. In that case, stage 0 will be included. When $sel = 0$, stage 0 is excluded, and the iterative unit is executed directly. To minimize memory utilization in storing TF values, the TFs must be calculated on the fly. Therefore a TFG is introduced in this work. Only several TF bases and special constants W_C need to be stored. Each operation will get 15 TFs from TF bases and W_C multiplication.

The top-level architecture for the proposed NTT is shown in Figure 2. It comprises a processing unit, FIFO, TFG, RAM, and a main control unit. The processing unit which consists of

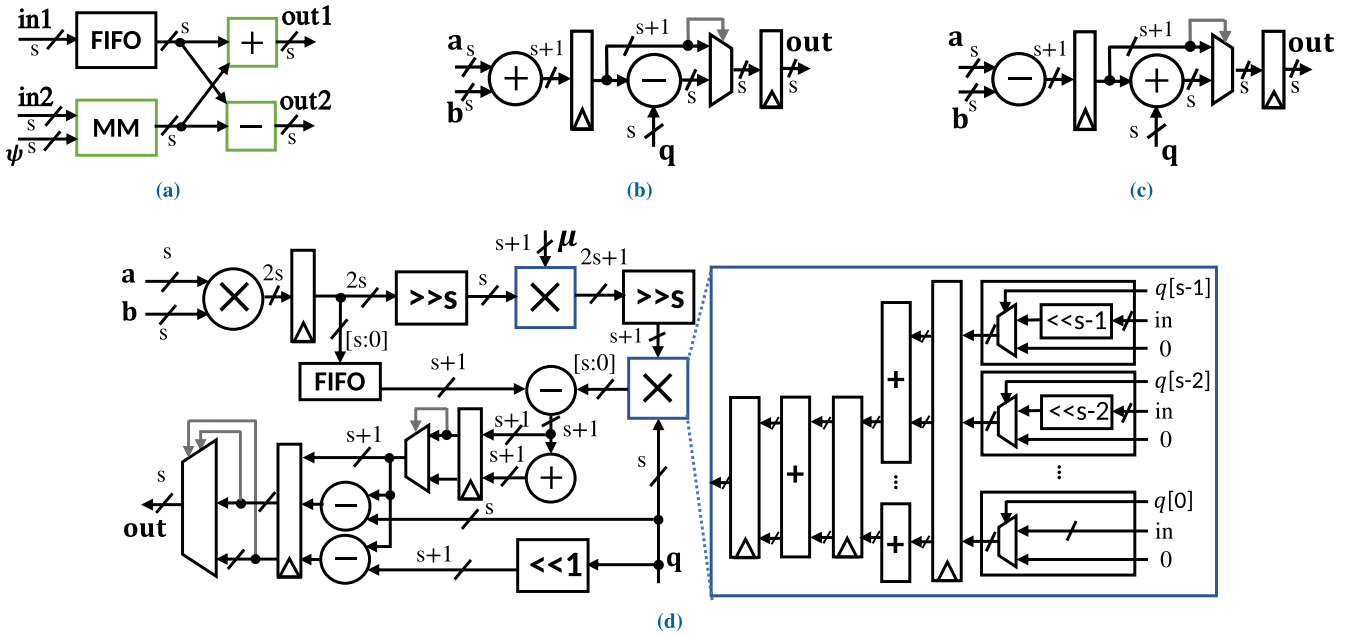

FIGURE 3. (a) Detailed CT radix-2 BU (b) MA (c) MS (d) MM unit.

butterfly units (BUs), evaluates the arithmetic computation. The part comprising stage 0 is the first summation of Equation (8), whereas the iterative radix-4 butterfly unit (BU) computes the remainder of the summation. We combined the 4-stage radix-2^4 BU (16 input coefficients) and 16 paralleled radix-2 modules (32 input coefficients) to accommodate both input lengths and optimize between the area and latency since the input data for NTT 2^{17} is extensively large, requiring a considerable amount of processing unit and latency.

We utilized the radix-2^4 BU module to calculate NTT 2^{16} in four iterations. To accommodate 2^{17} input length, an additional radix-2 stage is required before the iterative module to extend the 2^{16} architecture. We utilized 16 BU with 32 input coefficients such that the index of the output coefficient can be directly fed to the next stage (the iterative 4-stage radix-2^4 BU). The output data of the second half of this stage is stored by utilizing FIFO because the iterative stage requires 16 coefficients as input, whereas the output data of the first stage requires 32 coefficients. Thus, the output data of the FIFO is fed to the iterative stage immediately after the first half of the output data are fed to the next stage, which is after 4096 CC.

RAM is used to store intermediate data between iterations. A conflict-free memory address management was applied to achieve optimal memory utilization, which allows reading and writing data at the same time without conflict.

The control unit manages the data flow, synchronizes all signals, and provides the correct address to read and write the RAM without conflict. Moreover, it also has the important role of generating and controlling all signals of each sub-module to work correctly. The TFG provides respective ψ^k values to the input coefficient. Further, we calculated the twiddle factor (TF) on the fly to optimize memory utilization.

B. BUTTERFLY UNIT

Figure 3 shows the CT radix-2 BU that is utilized in the proposed NTT and its submodules. Using this architecture, the normal order input generates a reversed order output. Later, the reversed order output can be reordered back when it passes the INTT module in the HE system. In Figure 3(a), each BU comprises a modular adder (MA), a modular subtractor (MS), a FIFO, and an MM unit, while Figure 3(b-d) illustrate the architectures of the MA, MS, and MM unit, respectively.

The MM between the input data and TF is crucial in the entire area consumption of the NTT because 16 and 32 MM units were utilized in the first and iterative stages, respectively, as well as 15 MM units in the TFG, constituting most of the area consumed. Moreover, MM is a sophisticated operation that requires many resources. In our NTT, a total of 63 MM units are utilized as a part of the processing unit and TFG module.

Therefore, we utilized the Barrett-based modular reduction (MR) algorithm [30] and modified it to support any moduli q input with optimum speed and efficient area utilization. By applying this MM, our NTT can evaluate any moduli q input to support high-security levels with RNS in the HE system. Algorithm 2 shows MM with Barrett modular reduction that we implemented in our design.

Our proposed Barrett-based MM unit architecture is depicted in Figure 3(d). We employ adders and shifters to construct half multipliers, allowing for processing arbitrary q input with the same bit size. Recognizing the need to utilize RNS for building the complete accelerator system, which requires multiple values of q moduli, this MM unit is designed to accommodate arbitrary values for all its inputs. Additionally, FIFO modules are employed to reduce

Iter.		Stage 0					Iter	Iteration 1 (8192 clk)					Iteration 2 (8192 clk)					Iteration 3 (8192 clk)					Iteration 4 (8192 clk)				
Clk		1	2	3	...	4096	clk	1	2	3	...	8192	1	2	3	...	8192	1	2	3	...	8192	1	2	3	...	8192
Coefficient Index	0	0	1	2	...	4095	0	0	1	2	...	69631	0	1	2	...	127231	0	1	2	...	130831	0	16	32	...	131056
	1	4096	8193	8194	...	8191	1	4096	4097	4098	...	73727	256	257	258	...	127487	16	17	18	...	130847	1	17	33	...	131057
	2	8192	16385	16386	...	12287	2	8192	8193	8194	...	77823	512	513	514	...	127743	32	33	34	...	130863	2	18	34	...	131058
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	30	122880	122881	122882	...	126975	14	57344	57345	57346	...	126975	3584	3585	3586	...	130815	224	225	226	...	131055	14	30	46	...	131070
	31	126976	126977	126978	...	131071	15	61440	61441	61442	...	131071	3840	3841	3842	...	131071	240	241	242	...	131071	15	31	47	...	131071

FIGURE 4. Indexing scheme of polynomial coefficients through NTT 2^{17} iterations.

Algorithm 1 Proposed Configurable NTT Algorithm

Input: $\boldsymbol{a} = (a[0], a[0], \ldots, a[131071])$; $\boldsymbol{sel} = \{0, 1\}$;
Output: $\boldsymbol{a} = NTT_N(\boldsymbol{a})$; $N = \{2^{16}, 2^{17}\}$
Functions:
BitReverse(u, v);
NTT$_2(a, W, q)$;
NTT$_{16}(a, W, q)$;
Main:
// For $N = 2^{17}$. For $N = 2^{16}$, this if-block is skipped.
if $sel = 1$ **then**
  for $j = 0$ to $2^{16} - 1$ **do**
    $\tilde{j} =$ BitReverse$(j, 16)$
    for $i = 0$ to 1 **do**
      $temp[i] = \boldsymbol{a}[\tilde{j} \times 2 + i]$
    end for
    $\boldsymbol{a} \leftarrow$ NTT$_2(temp, \psi^{65536}, q)$
  end for
end if

// For $N = 2^{16}$
for $l = 0$ to 3 **do**
  for $k = 1$ to 15 **do**
    $\tilde{k} =$ BitReverse$((16^l) \times 2^{\lfloor \log_2 k \rfloor} + k, 17)$
    $W[k] = \psi_{\text{pow}}[\tilde{k}]$
  end for
  for $j = 0$ to $16^l - 1$ **do**
    $\tilde{j} =$ BitReverse$(j, 4l)$
    for $i = 0$ to $2^{17-4(l+1)} - 1$ **do**
      for $k = 0$ to 15 **do**
        $temp[k] = \boldsymbol{a}[\tilde{j} \times 2^{17-4l} + k \times 2^{17-4(l+1)} + i]$
      end for
      $\boldsymbol{a} \leftarrow$ NTT$_{16}(temp, W, q)$
    end for
    for $m = 1$ to 15 **do**
      $W[m] = W[m] \times W_C[17 - 4l - \lfloor \log_2 m \rfloor]$
    end for
  end for
end for

the critical path and enhance speed. Since this MM unit requires a total of 9 clock cycles, in the radix-2 BU, a FIFO is used to store the input not multiplied with a TF. This ensures that the correct data index is fed into the MA and MS.

C. CONFLICT FREE ACCESS PATTERN AND ADDRESSING

The index of coefficients through butterfly radix-2 stage 0 and all iterations are shown in Figure 4. Thirty-two input coefficients were utilized at stage 0 as 16 parallelized radix-2 units were utilized; the index numbers were multiples of 4096. Inside the first radix-2 BU of stage 0, the input coefficients were index 0 and 65536, whereas the 16$^{\text{th}}$ radix-2 input coefficients were index 61440 and 126976. This scheme guaranteed that the output of this stage could directly proceed to the next stage, which is stage 1 of iterative BU, without waiting for the entire data coefficient to pass through stage 0.

As we can see, the coefficient indices required at iteration 1 are 16 multiples of 4096 indices as half of the stage 0 input, such as $0, 4096, 8192, \ldots 61440$. Those coefficient indices are directly fed to iteration 1 input, while the other half are fed to FIFO. This direct process is beneficial to reduce significant latency, which is crucial in our big number coefficient input calculation.

The latency required for the first stage of radix-2 was only 11 clock cycles for the butterfly operation.

The next stage is the iteration stage. It utilized 4-stage radix-2^4. The module fetched 16 first input coefficients from stage 0. The other 16 input coefficients were fed to the FIFO to delay the operation until the inputs of the first half were fed in. Each iteration took 8192 CC to input data and 51 CC for butterfly operation and memory access.

The next iteration, including the reading operation, could only be conducted after all the data passed through four stages, whereas the output was directly written to the RAM after finishing the stage. The writing address and bank address to manage all intermediate data are shown in Figure 5,

Iter.		Iteration 1 (8192 clk)					Iteration 2 (8192 clk)					Iteration 3 (8192 clk)					Iteration 4 (8192 clk)				
Clk		1	2	3	...	8192	1	2	3	...	8192	1	2	3	...	8192	1	2	3	...	8192
Writing Address	0	0	0	0	...	4351	0	0	0	...	7951	0	0	0	...	8176	0	1	2	...	8191
	1	256	256	256	...	4607	16	16	16	...	7967	1	1	1	...	8177	0	1	2	...	8191
	2	512	512	512	...	4863	32	32	32	...	7983	2	2	2	...	8178	0	1	2	...	8191
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	14	3584	3584	3584	...	7935	224	224	224	...	8175	14	14	14	...	8190	0	1	2	...	8191
	15	3840	3840	3840	...	8191	240	240	240	...	8191	15	15	15	...	8191	0	1	2	...	8191
Bank Address	0	0	1	2	...	14	0	1	2	...	14	0	1	2	...	14	0	1	2	...	14
	1	1	2	3	...	15	1	2	3	...	15	1	2	3	...	15	1	2	3	...	15
	2	2	3	4	...	0	2	3	4	...	0	2	3	4	...	0	2	3	4	...	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	14	14	15	0	...	12	14	15	0	...	12	14	15	0	...	12	14	15	0	...	12
	15	15	0	1	...	13	15	0	1	...	13	15	0	1	...	13	15	0	1	...	13

FIGURE 5. Conflict free writing address scheme through NTT 2^{17} iterations.

Iter.	Iteration 1 Writing																		
Clk	1	2	...	16	17	...	32	33	...	48	49	...	4096	4097	...	4112	4113	...	8192
Writing Address 0	0	0	...	0	1	...	1	2	...	2	3	...	255	4096	...	4096	4097	...	4351
1	256	256	...	256	257	...	257	258	...	258	259	...	511	4352	...	4352	4353	...	4607
2	512	512	...	512	513	...	513	514	...	514	515	...	767	4608	...	4608	4609	...	4863
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	3584	3584	...	3584	3585	...	3585	3586	...	3586	3587	...	3839	7680	...	7680	7681	...	7935
15	3840	3840	...	3840	3841	...	3841	3842	...	3842	3843	...	3843	7936	...	7936	7937	...	8191
Bank Address 0	0	1	...	15	1	...	0	2	...	1	3	...	13	0	...	15	1	...	14
1	1	2	...	0	2	...	1	3	...	2	4	...	14	1	...	0	2	...	15
2	2	3	...	1	3	...	2	4	...	3	5	...	15	2	...	1	3	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	14	15	...	13	15	...	14	0	...	15	1	...	11	14	...	13	15	...	12
15	15	0	...	14	0	...	15	1	...	0	2	...	12	15	...	14	0	...	13

(a)

Iter.	Iteration 1 Reading for Iteration 2 Input																		
Clk	1	2	...	16	17	...	32	33	...	48	49	...	4096	4097	...	4112	4113	...	8192
Reading Address 0	0	0	...	0	1	...	1	2	...	2	3	...	3855	4096	...	4096	4097	...	7951
1	16	16	...	16	17	...	17	18	...	18	19	...	3871	4112	...	4112	4113	...	7967
2	32	32	...	32	33	...	33	34	...	34	35	...	3877	4128	...	4128	4129	...	7983
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	224	224	...	224	225	...	225	226	...	226	227	...	4079	4320	...	4320	4321	...	8175
15	240	240	...	240	241	...	241	242	...	242	243	...	4095	4336	...	4336	4337	...	8191
Bank Address 0	0	1	...	15	1	...	0	2	...	1	3	...	13	0	...	15	1	...	14
1	1	2	...	0	2	...	1	3	...	2	4	...	14	1	...	0	2	...	15
2	2	3	...	1	3	...	2	4	...	3	5	...	15	2	...	1	3	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	14	15	...	13	15	...	14	0	...	15	1	...	11	14	...	13	15	...	12
15	15	0	...	14	0	...	15	1	...	0	2	...	12	15	...	14	0	...	13

(b)

FIGURE 6. (a) Detailed conflict-free writing address and bank address scheme for iteration 1 output data writing (b) Detailed conflict-free RAM data reading address and bank address scheme for iteration 2 input data.

while detailed examples of writing and reading address management for iteration 1 are shown in Figure 6. Since the reading process of RAM was enabled after the writing process was finished, the reading address in Figure 6 (b) shows the reading address and bank address of RAM output as coefficient input of iteration 2.

One iteration will inevitably consume significant latency for writing the whole data, 8192 CC, and mathematical operation with pipelines, 51 CC, resulting in 8243 CC latency for each iteration. However, this address mapping scheme guarantees a conflict-free and efficient writing and reading operation of 16 coefficients. This process continued until the fourth iteration, whereas the order of coefficient groups changed through loop iterations based on Figure 4 and managed by the control unit.

When the input *sel* was set to 0, the main architecture operated as 2^{16} NTT. The 16 input coefficients were directly fed to the 4-stage radix-2^4 or the iterative BU for four iterations. Each iteration utilized 4096 clocks ($2^{16}/2^4$) and additional 51 CC for the butterfly operation and memory access.

D. TWIDDLE FACTOR GENERATOR

Because the input number N is large, the TFG is crucial for area efficiency. We required 65535 and 131071 TF values (ψ^k), with a width of 60 bits each, which is memory-consuming. The design of the TFG that calculated 15 TF values (TFs) on the fly thus optimizing memory utilization, is shown in Figure 7. We can see there are four groups of MMs, FIFO, and ψ_c from the left side to the right to show the stages in iterative BU. For example, we utilized 1 MM at the first stage, then 2 MM, 4 MM, and 8 MM for stages 2, 3, and 4, respectively, based on the number of TFs required. A total of 15 MM calculated the TFs by multiplying TF bases and constant values generated and controlled by the control unit. The control unit manages the counter, addresses of the generator, FSM, and several constants to be multiplied with the selected TF bases stored in the ROM.

Figure 2 shows at radix-2^4 BU, input data is fed in the order of the first, second, third, and fourth stages. Each stage is performed within 11 clock cycles. Therefore, TFs must be ready for the first clock in the first stage, 12$^{\text{rd}}$, 23$^{\text{th}}$, and 34$^{\text{th}}$ clock for the second, third, and fourth stages, respectively.

We employed FIFO to delay the MM result for stages 2, 3, and 4 to fulfill this requirement. This architecture is simpler than the architecture proposed in a previous study [38] since their TFG architecture has feedback from its TF output to be fed again as an input of the MM. That architecture required more registers to store its results and longer latency to evaluate TF calculation.

The power index of the TF bases stored in the ROM for 2^{17} and 2^{16} NTT is shown in Figure 8. It shows some samples of TF bases need to be stored for each iteration. Each

Algorithm 2 Modular Multiplication Followed by Standard Barrett Modular Reduction (BMR)

Input: $a, b, q \in \mathbb{Z}$
Output: $a \times b \bmod q$
// Pre-computation
$k = \lceil \log_2 q \rceil$ // k is the number of bits in q
$r = 2^k$
$\mu = \lfloor \frac{r^2}{q} \rfloor$
// Multiplication
$z = a \times b$
// Modular reduction
$m_1 = \lfloor \frac{z}{r} \rfloor$
$m_2 = m_1 \times \mu$
$m_3 = \lfloor \frac{m_2}{r} \rfloor$
$m_4 = (m_3 \times q)$
$z = z \bmod 2^{k+1}$
$m_4 = m_4 \bmod 2^{k+1}$
if $z < m$ **then**
  $t = 2^{k+1} + z - m_4$
else
  $t = z - m_4$
end if
if $t \geq 2q$ **then return** $t - 2q$
else if $t \geq q$ **then return** $t - q$
else return t
end if

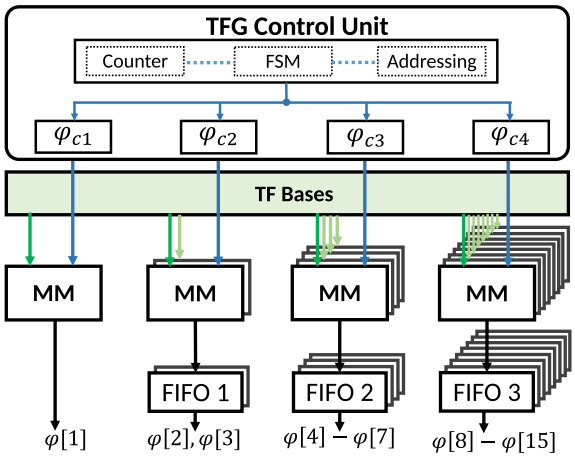

FIGURE 7. Twiddle factor generator (TFG) architecture.

column has 15 TF bases because the iterative radix-2^4 BU requires 15 TFs for each calculation. One of each 32 original TFs was saved as TF base; thus, memory utilization can be reduced up to approximately 96.88%. The total number of TFs for 2^{17} and 2^{16} NTT were originally 131071 and 65535, respectively. Because 15 TFs are utilized for each 16-input coefficient, we originally needed to store 8758×15 and 4369×15 TFs. However, by applying our proposed TFG architecture design, only 275×15 and 138×15 TFs are stored for 2^{17} and 2^{16} N input, respectively.

TABLE 1. Area breakdown of the proposed NTT architecture on the Xilinx Ultrascale+ XCU250 FPGA.

Module	LUT (%)	FF (%)	DSP (%)	BRAM (%)
Stage 0	29227 (11)	19040 (23)	160 (25)	0
Iterative BU	57488 (21)	36640 (44)	320 (51)	0
FIFO	123360 (45)	6045 (7)	0	0
RAM	0	0 (0)	0	216 (89)
TFG	24398 (9)	17475 (21)	150 (24)	25 (11)
Control Unit	39963 (14)	3917 (5)	0	0
TOTAL	274436	83117	630	241

IV. RESULTS AND DISCUSSION

We implemented the proposed NTT architecture by utilizing System Verilog HDL on the Xilinx Vivado©(v2020.1) tool and synthesized them on the Xilinx ALVEO U250 FPGA board. We selected this specific FPGA board for its large capacity to support our future research endeavors utilizing the proposed NTT, such as key switching and HE accelerator. Table 1 presents the hardware resource utilization of major sub-modules utilized in our proposed NTT module. Iterative BU and stage-0, which act as the processing unit, dominate in hardware resource consumption. Since each MM unit uses 10 DSP slices, stage-0, Iterative BU, and TFG consume 160, 320, and 150 DSP slices respectively.

Table 2 presents the hardware utilization and clock latency of a single radix-2 CT-BU in our work, comparing it with [38] and [42]. We observe that our BU and its submodules employ an optimized amount of hardware and clock latency. The BU architecture in [42] uses 32 bits of data, almost

TABLE 2. Comparison of hardware utilization and clock cycle latency for a single Radix-2 CT-BU architecture on FPGA.

Work	N	$\log_2 q$	Module	LUT	FF	DSP	CC	Device[a]
Su [42]	2^{10}	32	**BU**	**1201**	**789**	**10**	**10**	XC7V
			⌊MM	767	699	10	10	
Duong [38]	2^{16}	60	**BU**	**4354**	**2106**	**12**	**23**	ZU9EG
			⌊MM	3947	902	12	21	
This work	2^{16}/ 2^{17}	60	**BU**	**2233**	**1238**	**10**	**11**	U250
			⌊MM	1876	868	10	9	
			⌊FIFO	90	128	0		
			⌊MA	157	121	0	2	
			⌊MS	39	121	0		
			⌊Leaf	71	0	0	0	

[a] XC7V: Virtex-7 XC7VX485T; ZU9EG: Zynq Ultrascale+ XCZU9EG; U250: Xilinx Ultrascale+ Alveo U250

half the number compared to our architecture. Despite this, our architecture utilizes the same number of DSPs, and a considerable amount of LUT and FF. Our MM unit employs 9 pipelines, while [42] uses 10 pipelines, resulting in latencies of 9 and 10 CC, respectively. On the other hand, [38] which employs the same bit number size as our work, requires more hardware resources for its BU and MM, as well as a longer CC latency. This is attributed to their use of MM units based on [48] that is designed for specific moduli. However, it requires 2 half-intmult units to support one more moduli, leading to increased hardware requirements for their specific 32 moduli. In contrast, our MM unit is designed with the ability to support flexible or arbitrary moduli, resulting in efficient hardware utilization and CC latency.

We also compared the proposed NTT architecture with those from similar related studies [38], [40], [41], [43], [44], [45], [46], [47]. However, a particular metric was required to fairly compare results because each researcher utilized significantly different devices, silicon technologies platforms, and parameters for their research. One such metric that fairly compares across various platforms is the throughput per slice (TPS) [41], [49].

Throughput is defined as follows:

$$\text{Throughput (Mbps)} = \frac{\text{Total number of bits}}{\text{Latency } (\mu s)} \tag{9}$$

The throughput can be an excellent comparison among identical platforms, but it is not ideal to compare results between platforms that differ considerably. Hence, we normalized the TPS by normalizing it to the equivalent slices:

$$\text{TPS (Mbps/slice)} = \frac{\text{Throughput (Mbps)}}{\text{Equivalent slices}} \tag{10}$$

where the equivalent slices are defined as:

$$\text{Eq. slices} = \text{Number of slices} + \text{Eq. DSP} + \text{Eq. BRAM} \tag{11}$$

The number of slices is derived from (Number of LUT)/8 due to 8 LUTs in one Slice for Ultrascale FPGAs [50]. Equivalent DSP is obtained by substituting each DSP block with 51.2 slices and replacing one BRAM with 116.2 Slices to get the equivalent BRAM [49], [51].

A comparison of our design to other related studies is shown in Table 3. A previous study on configurable NTT [41]

Iter	1	2		3			4		
1	32768	2048	6144	128	2176	...	8	2056	...
2	16384	1024	3072	64	1088	...	4	1028	...
3	81920	66560	68608	65600	66624	...	65540	66564	...
4	8192	512	1536	32	544	...	2	514	...
5	73728	66048	67072	65568	66080	...	65538	66060	...
6	40960	33280	34304	32800	33312	...	32770	33282	...
7	106496	98816	99840	98336	98848	...	98306	98818	...
8	4096	256	768	16	272	...	1	257	...
9	69632	65792	66304	65552	65808	...	65537	65793	...
10	36864	33024	33536	32784	33040	...	32769	33025	...
11	102400	98560	99072	98320	98576	...	98305	98561	...
12	20480	16640	17152	16400	16656	...	16385	16641	...
13	86016	82176	82688	81936	82192	...	81921	82177	...
14	53248	49408	49920	49168	49424	...	49153	49409	...
15	118748	114944	115456	114704	114960	...	114689	114945	...
Tot	1	2		16			256		

(a)

Iter	1	2	3			4		
1	32768	2048	128	1152	...	8	1032	...
2	16384	1024	64	576	...	4	516	...
3	49152	33792	32832	33344	...	32772	33284	...
4	8192	512	32	288	...	2	258	...
5	40960	33280	32800	33056	...	32770	33026	...
6	24576	16896	16416	16672	...	16386	16642	...
7	57344	49664	49184	49440	...	49154	49410	...
8	4096	256	16	144	...	1	129	...
9	36864	33024	32784	32912	...	32769	32897	...
10	20480	16640	16400	16528	...	16385	16513	...
11	53248	49408	49168	49296	...	49153	49281	...
12	12288	8448	8208	8336	...	8193	8321	...
13	45056	41216	40976	41104	...	40961	41089	...
14	28672	24832	24592	24720	...	24577	24705	...
15	61440	57600	57360	57488	...	57345	57473	...
Tot	1	1	8			128		

(b)

FIGURE 8. Fifteen-bank memory scheme of TF bases for the TFG (a) for NTT 2^{17} (b) for NTT 2^{16}.

TABLE 3. Comparison of the proposed NTT architecture and previous works.

Work	Parameter		Freq	Latency		LUT	FF	DSP	RAM	Eq. Slices	Thro.	TPS	Device[a]
	N	$\log_2 q$	(MHz)	(CC)	μs	(K)	(K)		(KB)	(K)	(Mbps)	(Thro./ Eq.S.)	
Xin [43]	2^{12}	40	150	3072	20	176	37	1344	2,475	154.7	21,800	0.14	ZU9EG
Ye [44]	2^{12}	60	150	8284	55	17	11	286	100	38.7	4,450	0.11	VX485
Di Matteo [45]	2^{12}	32	180	24584	136.58	3.32	1.48	42	168.7	6.923	959.67	0.14	ZCU106
Su [46]	2^{14}	60	300	28672	95.6	99.4	93.2	1080	16,560	506.9	10,286	0.02	VU190
Ozturk [47]	2^{15}	32	250	12725	50.9	219	91	768	869	178.3	20,560	0.11	VX690
Roy [39]	2^{16}	30	100	47795	477.95	72.6	63.1	250	486	68.9	4,144	0.06	VLX240
Duong-Ngoc [38]	2^{16}	60	200	16776	83.8	148.5	90.9	564	616.5	63.4	40,066	0.63	ZU9EG
Li [40]	2^{16}	64	196	16656	84.98	61.2	108.94	264	1,062	91.1	49,357	0.54	5SGXEA
Kurniawan [41]	2^{16}	60	250	16628	66.51	74.5	61.4	288	3,973*	126.6	59,120	0.47	VU37P
This study	2^{16}	60	250	16552	66.2	274.4	83.1	630	1084	94.5	**59,398**	**0.63**	U250
	2^{17}	60	250	32977	131.9						**59,623**	**0.63**	U250

[a]Device: VX485 (Virtex-7 XC7VX485T); ZU9EG: Zynq Ultrascale+ XCZU9EG; VU190 (Virtex Ultrascale XCVU190);VX690 (Virtex 7 XC7VX690T); VLX240 (Virtex-6 FPGA XC6VLX240T-1FF1156); 5SGXEA (Stratix-V 5SGXEABN2F45I2); U250 (Alveo U250).
*RAM is approximated by calculating its BRAM to store TFs for all 6 sizes of N input it can support and RAM for storing intermediate data.

with high throughput did not have a TFG to calculate TFs on the fly and utilized TFs as an input to the NTT. Even though the NTT only stores TFs for one set of N, the other N values will still need ROM for storing TFs out of the NTT module. Therefore, a large ROM is required to store all TF values for the 6 sizes of N input it supports. Another work in 2^{16} NTT like [39] has a lower clock frequency and even bigger BRAM, resulting in lower throughput and TPS. The work of [38] proposed 2^{16} NTT, but the design is limited to several specified moduli q values due to the MM architecture they utilized. Reference [40] works in lower frequency and utilizes bigger equivalent of slice, resulting in lower throughput and TPS than ours. Reference [45] proposed an NTT supported by a TFG unit. However, the TFG generates all TF values before the NTT operation, causing additional latency for TF generation. This method may not be suitable for NTT with high polynomial degrees like our work. In this study, we calculate the TF on the fly to enable the generation of 15 TFs every clock cycle. Further, our proposed design achieved up to 59,623 and 0.63 in terms of throughput and TPS, outperforming the result in other studies.

Our proposed architecture is the only NTT on FPGA that can support up to $N = 2^{17}$. Moreover, the power consumption of the architecture is only 25.9 Watt, which is relatively low considering our architecture parameters. In general, under the same architecture, a 2^{17} NTT will require double RAM memory to store the intermediate data, double BRAM memory to store the TF bases, and double clock latency compared to 2^{16} NTT while maintaining the same throughput. Consequently, the equivalent slices experience a significant increase, leading to a substantial reduction in TPS. Therefore, designing an optimized 2^{17} NTT architecture poses a considerable challenge. Furthermore, our NTT architecture also supports 2^{16} NTT, requiring dedicated BRAM to store 2^{16} NTT TF bases. Therefore, we employ the optimized and highly efficient architecture described in previous sections to support both N sizes, essential for a Fully HE system to achieve efficient area utilization and relatively high-speed performance.

V. CONCLUSION AND FUTURE WORKS

A. CONCLUSION

We implemented a configurable NTT architecture with increased input lengths of 2^{16} and 2^{17}. These configurable input lengths, as well as the flexible MM units we utilized, could facilitate bootstrapping modules in HE schemes. The

proposed architecture combined parallel butterfly radix-2 and iterative butterfly radix-2^4 to achieve optimum hardware utilization and latency. Finally, we designed an on-the-fly TFG to minimize memory utilization by preventing the storage of all twiddle-factor constants in memory. By utilizing this TFG, we can reduce BRAM usage to around 96.9%. The proposed configurable NTT architecture achieves time-area efficient architecture for a bootstrappable HE system.

B. FUTURE WORKS

In future work, we will utilize the proposed NTT module to build the Fully HE system. To achieve that aim, we need to implement the essential modules such as key switching and bootstrapping. Extensive data and expensive computation are required to implement the whole system. As one fundamental module to build the system, our proposed NTT has been designed to support future aims, such as high N sizes input, as well as employing TFG to minimize ROM usage and flexible q input MM units. The use of FPGA devices and ASIC implementation will make it possible to realize the Fully HE-based accelerator.

REFERENCES

[1] K. Munjal and R. Bhatia, "A systematic review of homomorphic encryption and its contributions in healthcare industry," *Complex Intell. Syst.*, vol. 9, no. 4, pp. 3759–3786, Aug. 2023.
[2] R. K. Dhanaraj, S. Suganyadevi, V. Seethalakshmi, and M. Ouaissa, "Introduction to homomorphic encryption for financial cryptography," in *Homomorphic Encryption for Financial Cryptography: Recent Inventions and Challenges*. Cham, Switzerland: Springer, 2023, pp. 1–12.
[3] J. L. López Delgado, J. A. Álvarez Bermejo, and J. A. López Ramos, "Homomorphic asymmetric encryption applied to the analysis of IoT communications," *Sensors*, vol. 22, no. 20, p. 8022, Oct. 2022.
[4] A. Triakosia, P. Rizomiliotis, K. Tserpes, C. Tonelli, V. Senni, and F. Federici, "Homomorphic encryption in manufacturing compliance checks," in *Proc. Int. Conf. Trust Privacy Digit. Business*. Cham, Switzerland: Springer, 2022, pp. 81–95.
[5] 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.
[6] S. L. Nita and M. I. Mihailescu, "Homomorphic encryption," in *Advances to Homomorphic and Searchable Encryption*. Cham, Switzerland: Springer, 2023, pp. 27–88.
[7] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Tech. Rep., 2015.
[8] W. P. Wardlaw, "The rsa public key cryptosystem," in *Coding Theory and Cryptography: From Enigma and Geheimschreiber to Quantum Theory*. Cham, Switzerland: Springer, 2000, pp. 101–123.
[9] A. V. Meier, "The elgamal cryptosystem," in *Joint Advanced Students Seminar*. Munich, Germany: TU München, 2005.
[10] T. Sridokmai and S. Prakancharoen, "The homomorphic other property of Paillier cryptosystem," in *Proc. Int. Conf. Sci. Technol. (TICST)*, Nov. 2015, pp. 356–359.
[11] J. D. Cohen and M. J. Fischer, "A robust and verifiable cryptographically secure election scheme," Dept. Comput. Sci., Yale Univ., New Haven, CT, USA, Tech. Rep., 1985.
[12] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. forty-first Annu. ACM Symp. Theory Comput.*, May 2009, pp. 169–178.
[13] T. N. Tan and H. Lee, "High-secure low-latency ring-LWE cryptography scheme for biomedical images storing and transmitting," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–4.
[14] 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.
[15] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proc. 44th Annu. ACM Symp. Theory Comput.*, May 2012, pp. 1219–1234.
[16] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. Annu. Cryptol. Conf.* Berlin, Germany: Springer, 2012, pp. 868–886.
[17] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. 33rd Annu. Cryptol. Conf.* Santa Barbara, CA, USA: Springer, 2013, pp. 75–92.
[18] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. 23rd Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Hong Kong: Springer, 2017, pp. 409–437.
[19] P. N. Duong and H. Lee, "Pipelined key switching accelerator architecture for CKKS-based fully homomorphic encryption," *Sensors*, vol. 23, no. 10, p. 4594, May 2023.
[20] N. Neda, A. Ebel, B. Reynwar, and B. Reagen, "CiFlow: Dataflow analysis and optimization of key switching for homomorphic encryption," 2023, *arXiv:2311.01598*.
[21] R. Geelen, M. Van Beirendonck, H. V. Pereira, B. Huffman, T. McAuley, B. Selfridge, D. Wagner, G. Dimou, I. Verbauwhede, and F. Vercauteren, "BASALISC: Programmable hardware accelerator for BGV fully homomorphic encryption," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2023, no. 4, pp. 32–57, 2023.
[22] A. Al Badawi, L. Hoang, C. Fook Mun, K. Laine, and K. Mi Mi Aung, "PrivFT: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, pp. 226544–226556, 2020.
[23] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An architecture for computing on encrypted data," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2020, pp. 1295–1309.
[24] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. 25th Int. Conf.* Calgary, AB, Canada: Springer, 2019, pp. 347–368.
[25] 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.
[26] 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.
[27] A. Al Badawi and Y. Polyakov, "Demystifying bootstrapping in fully homomorphic encryption," Cryptol. ePrint Arch., ACM, New York, NY, USA, 2023.
[28] A. Kim, M. Deryabin, J. Eom, R. Choi, Y. Lee, W. Ghang, and D. Yoo, "General bootstrapping approach for RLWE-based homomorphic encryption," *IEEE Trans. Comput.*, vol. 73, no. 1, pp. 86–96, Jan. 2024.
[29] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022, pp. 173–187.
[30] A. Satriawan, I. Syafalni, R. Mareta, I. Anshori, W. Shalannanda, and A. Barra, "Conceptual review on number theoretic transform and comprehensive review on its implementations," *IEEE Access*, vol. 11, pp. 70288–70316, 2023.
[31] Z. Liang and Y. Zhao, "Number theoretic transform and its applications in lattice-based cryptosystems: A survey," 2022, *arXiv:2211.13546*.
[32] T. Adiono and R. Mareta, "Low latency parallel-pipelined configurable FFT-IFFT 128/256/512/1024/2048 for LTE," in *Proc. 4th Int. Conf. Intell. Adv. Syst. (ICIAS)*, vol. 2, Jun. 2012, pp. 768–773.
[33] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, p. 297, Apr. 1965.
[34] W. M. Gentleman and G. Sande, "Fast Fourier transforms: For fun and profit," in *Proc. Fall Joint Comput. Conf.*, 1966, pp. 563–578.
[35] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, and L. Fanucci, "A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms," *IEEE Access*, vol. 9, pp. 150798–150808, 2021.
[36] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, "VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 8, pp. 2672–2684, Aug. 2020.

[37] C.-M. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C.-J. Shih, and B.-Y. Yang, "NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2," in *IACR Transactions on Cryptographic Hardware and Embedded Systems*. Bellevue, WA, USA: IACR, 2021, pp. 159–188.

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

[39] 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.

[40] Z. Li, J. Ren, G. Du, Z. Tu, X. Wang, Y. Yin, and Y. Ouyang, "An area-efficient large integer NTT-multiplier using discrete twiddle factor approach," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 2, pp. 751–755, Feb. 2023.

[41] S. Kurniawan, P. Duong-Ngoc, and H. Lee, "Configurable memory-based NTT architecture for homomorphic encryption," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 10, pp. 3942–3946, Oct. 2023.

[42] Y. Su, B.-L. Yang, C. Yang, Z.-P. Yang, and Y.-W. Liu, "A highly unified reconfigurable multicore architecture to speed up NTT/INTT for homomorphic polynomial multiplication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 8, pp. 993–1006, Aug. 2022.

[43] G. Xin, Y. Zhao, and J. Han, "A multi-layer parallel hardware architecture for homomorphic computation in machine learning," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[44] Z. Ye, R. C. C. Cheung, and K. Huang, "PipeNTT: A pipelined number theoretic transform architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 10, pp. 4068–4072, Oct. 2022.

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

[46] Y. Su, B. Yang, J. Wang, F. Zhang, and C. Yang, "Reconfigurable multi-core array architecture and mapping method for RNS-based homomophic encryption," *AEU-Int. J. Electron. Commun.*, vol. 161, Mar. 2023, Art. no. 154562.

[47] E. Öztürk, Y. Doröz, E. Savas, and B. Sunar, "A custom accelerator for homomorphic encryption applications," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 3–16, Jan. 2017.

[48] 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.

[49] D.-e.-S. Kundi, Y. Zhang, C. Wang, A. Khalid, M. O'Neill, and W. Liu, "Ultra high-speed polynomial multiplications for lattice-based cryptography on FPGAs," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1993–2005, Oct. 2022.

[50] D. Xilinx, "Ultrascale architecture and product data sheet: Overview," Product Specification, 2020.

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

RELLA MARETA received the B.S. and M.S. degrees in electrical engineering from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 2011 and 2014, respectively, and the M.S. degree in a dual master's program from the National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan, in 2014. She is currently pursuing the Ph.D. degree with the Digital Integrated Systems Laboratory, Department of Electrical and Computer Engineering, Inha University, South Korea. Previously, she was a Lecturer with the Department of Electrical Engineering and Informatics, Vocational School, Universitas Gadjah Mada, Yogyakarta, Indonesia, from 2016 to 2018. Her research interests include algorithms and architecture for digital signal processing, hardware acceleration architecture, post-quantum cryptography, and homomorphic encryption.

ARDIANTO SATRIAWAN received the B.S. and M.S. degrees in electrical engineering from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 2013 and 2015 respectively, and the Ph.D. degree from Seoul National University of Science and Technology (SeoulTech), South Korea, in 2024. He is currently a Postdoctoral Fellow with the Artificial Intelligence System-on-Chip (AI-SoC) Research Center, Inha University, South Korea. Previously, he was with the School of Electrical Engineering and Informatics, ITB, from 2017 to 2024. His research interests include wireless networks, machine learning, algorithms and architecture for digital signal processing, hardware acceleration architecture, post-quantum cryptography, and homomorphic encryption.

PHAP NGOC DUONG received the B.S. degree in electronic and telecommunication engineering from Danang University of Technology, in 2009, the M.S. degree in electronic engineering from The University of Danang, Vietnam, in 2015, and the Ph.D. degree from Inha University, South Korea, in 2022. He was a Postdoctoral Fellow with the Artificial Intelligence System-on-Chip (AI-SoC) Research Center, Inha University. He is currently a Lecturer with The University of Danang—Vietnam-Korea University of Information and Communication Technology, Vietnam. His research interests include algorithms and architecture for digital signal processing, hardware acceleration architecture, post-quantum cryptography, and homomorphic encryption.

HANHO LEE (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Minnesota, Minneapolis, MN, USA, in 1996 and 2000, respectively. From 2000 to 2002, he was a Member of the Technical Staff with Lucent Technologies (Bell Labs Innovations), Allentown, PA, USA, where he was involved in the design of DSP multi-processor architecture. From August 2002 to August 2004, he was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Connecticut, USA. Since September 2004, he has been with the Faculty of the Department of Information and Communication Engineering, Inha University, Incheon, South Korea, where he is currently a Full Professor. He was a Visiting Researcher with the Electronics and Telecommunications Research Institute (ETRI), South Korea, in 2005. He was a Visiting Scholar with Bell Labs, Alcatel-Lucent, Murray Hill, USA, from 2010 to 2011, and a Visiting Professor with The University of Texas at Dallas, USA, from 2017 to 2018. He leads the Digital Integrated Systems Laboratory and is the Director of the Artificial Intelligence System-on-Chip (AI-SoC) Research Center, Inha University. His research interests include algorithm and architecture design for post-quantum cryptography, homomorphic encryption, forward error correction coding, artificial intelligence, and digital signal processing. He is a member of IEEE CASS and SSCS. He was the General Chair of ISICAS and the Technical Program Chair of ISCAS and APCCAS. He was the Chair of the IEEE Circuits and Systems for Communications Technical Committee (CASCOM). He was a Board of Governor (BoG) of the IEEE Circuits and Systems Society (CASS), from 2020 to 2023. He is the Vice President of Technical Activities of IEEE CASS.