UNIT: A Highly Unified and Memory-efficient FPGA-based Accelerator for Torus FHE

Yuying Zhang*, Sharad Sinha†, Jiang Xu*, Wei Zhang*
* Hong Kong University of Science and Technology, {yzhangnf, jiang.xu, wei.zhang}@ust.hk
† Indian Institute of Technology (IIT) Goa, sharad@iitgoa.ac.in

***Abstract*—Fully Homomorphic Encryption (FHE) has emerged as a promising solution for the secure computation on encrypted data without leaking user privacy. Among various FHE schemes, Torus FHE (TFHE) distinguishes itself by its ability to perform exact computations on non-linear functions within the encrypted domain, satisfying the crucial requirement for privacy-preserving AI applications. However, the high computational overhead and strong data dependency in TFHE's bootstrapping process present significant challenges to its practical adoption and efficient hardware implementation. Existing TFHE accelerators on various hardware platforms still face limitations in terms of performance, flexibility, and area efficiency.**

In this work, we propose UNIT, a novel and highly unified accelerator for Programmable Bootstrapping (PBS) in TFHE, featuring carefully designed computation units. We introduce a unified architecture for negacyclic (inverse) number theoretic transform (I)NTT with fused twisting steps, which reduces computing resources by 33% and the memory utilization of pre-stored factors by nearly 66%. Another key feature of UNIT is the innovative design of the monomial number theoretic transform unit, called OF-MNTT, which leverages on-the-fly twiddle factor generation to eliminate memory traffic and overhead. This memory-efficient and highly parallelizable approach for MNTT is proposed for the first time in TFHE acceleration. Furthermore, UNIT is highly reconfigurable and scalable, supporting various parameter sets and performance-resource requirements.

Our proposed accelerator is evaluated on the Xilinx Alveo U250 FPGA platform. Experimental results demonstrate its superior performance compared to the state-of-the-art GPU and FPGA-based implementations with the improvement of 8.3× and 3.63×, respectively. In comparison with the most advanced FPGA implementation, UNIT achieves 30% enhanced area efficiency and 3.2× reduced power with much better flexibility.

I. Introduction

As the significance of safeguarding data privacy and integrity gains increasing recognition, several privacy-preserving computing techniques have been proposed for scenarios such as cloud computing [1], secure database search [2], and federated machine learning [3]. Among these techniques, Fully Homomorphic Encryption (FHE) has emerged as a promising solution thanks to its capability of secure computation on encrypted data without revealing user information to untrusted cloud servers [4]–[8]. Practical and widely adopted FHE schemes include word-wise variants such as BFV [4], BGV [5] and CKKS [6], as well as bit-wise alternatives like FHEW [8] and its successor TFHE [9]. Among these schemes, TFHE scheme stands out for its ability to perform exact computations on non-linear functions, such as sigmoid or ReLU activation functions commonly employed in machine learning models [10]. TFHE also supports deep neural network inference and training without limitations on the network's depth, making it attractive for privacy-preserving AI applications [11].

A. Motivation and Research Gap

In all FHE schemes, the bootstrapping process is necessary to manage noise accumulation. However, the computational complexity and overhead of bootstrapping can be prohibitively high [12]. TFHE provides the fastest and most efficient bootstrapping algorithm among these schemes, known as Programmable Bootstrapping (PBS) [9]. In addition to controlling noise growth, PBS can be "programmed" to apply arbitrary functions to the ciphertext after each homomorphic operation, such as non-linear activation functions in machine learning neural networks.

The bootstrapping process, although more efficient in TFHE compared to other FHE schemes, still introduces significant computational overhead and strong data dependency, posing considerable challenges to the practical adoption and efficient hardware implementation of TFHE. Moreover, the unique ciphertext structure of TFHE renders many existing FHE accelerators incompatible, necessitating the development of TFHE-specific hardware acceleration solutions. PBS operation constitutes the main cost of TFHE homomorphic calculations, which can consume up to 99.9% of the computation time in an encrypted circuit [13]. Therefore, accelerating the PBS operation becomes a primary target for achieving high-performance hardware acceleration of TFHE. The PBS operation primarily involves the iterative calculation of the external product, which is a vector-matrix multiplication where the elements are long polynomials with large integers, requiring significant computational resources and memory utilization. Furthermore, with the rapid development of FHE applications, there is a growing demand for flexibility in hardware accelerators to support different TFHE parameter sets.

B. Related Works

Several works have explored the acceleration of TFHE bootstrapping on various hardware platforms, including CPUs, GPUs, FPGAs, and ASICs. CPU [7], [9], [14]–[17] and GPU [14], [18] TFHE accelerators typically provide only modest speedups, which still suffer from a slowdown of up to 10,000× compared to computations on unencrypted data [19]. To address the speed limitations, researchers have turned their attention to dedicated hardware implementations

978-3-9826741-0-0/DATE25/© 2025 EDAA

based on ASIC [14], [20]–[22] or FPGA [13], [21], [23]–[25]. ASIC emulations in advanced technology nodes show promise for efficient TFHE acceleration. However, the fabrication process for these ASICs can take years, and they are specialized for a limited range of parameter sets. FPGA-based implementations strike a balance between development time and performance, offering flexibility in parameter sets while delivering significant speedups. Consequently, FPGAs have become a popular choice for TFHE acceleration. However, state-of-the-art FPGA-based works still face challenges in terms of performance, flexibility, memory overhead, and noise management. Work [24] reports high parallelism at the cost of massive on-chip memory utilization, while work [21] relies on one specific modulus with fixed parallelism in its NTT design. FPT [13] and ALT [25] achieve low latency thanks to the developed approximation methods, i.e. leveraging streaming negacylic FFTs with the continuous-flow pipeline in FPT and modifying the parameters at the algorithmic level in ALT. However, both optimizations introduce extra noise in bootstrapping and sacrifice the decryption failure rate of PBS for high performance. Therefore, the most challenging part of the high-performance TFHE bootstrapping hardware is the efficient memory design while keeping flexibility and accuracy.

Our work. Our proposed work aims to address these challenges by providing a balanced design with lower latency, reduced memory utilization, and reinforced reconfigurability compared to state-of-the-art works. The contributions of this work can be summarized as follows:

- We propose a novel and highly unified architecture for negacylic (I)NTT, the most computation-intensive module in PBS. The preprocessing step before NTT and the post-processing step after INTT are both fused to the main stage leveraging the inherent algebraic relationship for reduced pre-stored factors by almost 66% and decreased computing resources of (I)NTT operations by 33%.
- For the first time, we introduce an on-the-fly twiddle factor generation technique for our MNTT (NTT for monomial polynomial) design called OF-MNTT with the help of the parallel binary modular exponentiation algorithm, which eliminates memory traffic and saves memory usage for loading and storing twiddle factors. This novel OF-MNTT approach results in a memory-efficient and highly parallelizable architecture.
- The dominant modules are highly parameterized and scalable, enabled by flexible design and simplified control logic, supporting various parameter sets and performance-resource requirements.
- The evaluation results indicate that our proposed TFHE accelerator, UNIT, achieves reduced latency, better area efficiency, and enhanced reconfigurability compared to the state-of-the-art works.

II. Analysis of PBS Process in TFHE

A brief introduction to TFHE is provided in this section, with an analysis of the Programmable Bootstrapping (PBS) process. We direct the readers to [7], [9], [26] for a comprehensive overview of the TFHE scheme.

TFHE operates on elements defined over the real Torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$, which in practice are discretized as 32-bit integers modulo q. We denote $\mathbb{T}^{n+1}$ as $n+1$ elements of $\mathbb{T}$ and $\mathbb{T}[X]_N^{k+1}$ as $k+1$ polynomials each with N degree of $\mathbb{T}$. A TFHE ciphertext is denoted as $ct = (\mathbf{a}, b = \mathbf{a} \cdot \mathbf{s} + e + m) \in \mathbb{T}^{n+1}$, where m is the plaintext, $\mathbf{s}$ is the secret key, $\mathbf{a}$ is a random polynomial, e is the noise, and n is the dimension.

TFHE bootstrapping known as PBS is a necessary process that essentially decrypts the ciphertext homomorphically within the encrypted domain, i.e. homomorphically computes $b - \mathbf{a} \cdot \mathbf{s} = e + m$, which suppresses the noise and enables unbounded computation. Due to its "programmable" feature, PBS enables additional non-linear functions $testP$ on data, which means homomorphically compute:

$$testP \cdot X^{-b+\mathbf{a}\cdot\mathbf{s}} = testP \cdot X^{-b+\sum_{i=1}^{n} a_i s_i} \tag{1}$$

Initialize the vector $\mathbf{u} = testP \cdot X^{-b}$ and then it can be calculated iteratively as:

$$\mathbf{u} \leftarrow \mathbf{u} \cdot X^{a_i s_i} \tag{2}$$

Because $\mathbf{s}$ is the binary secret key, i.e. $s_i \in 0, 1$, Equation (2) can be rewritten as:

$$\mathbf{u} \leftarrow (X^{a_i} - 1) \cdot \mathbf{u} \cdot s_i + \mathbf{u} \tag{3}$$

To homomorphically compute this process without leaking the secret key, $\mathbf{s}$ is encrypted to $BSK \in \mathbb{T}[X]_N^{n\times(k+1)l\times(k+1)}$, while $\mathbf{u}$ is encrypted to $ACC \in \mathbb{T}[X]_N^{(k+1)}$. Therefore, each iteration can be expressed as:

$$ACC \leftarrow (X^{a_i} - 1) \cdot ACC \boxdot BSK_i + ACC \tag{4}$$

where $\boxdot$ is external product because both ACC and BSK_i consist more than one polynomials. This operation of Equation (4) is called control multiplexer (CMUX) and n iterations of CMUX are called Blind Rotation. The whole PBS process is shown in Algo1.

Algorithm 1: PBS Algorithm

Input: ciphertext $ct = (a_1, a_2, ..., a_n, b) \in \mathbb{T}^{n+1}$
Input: bootstrapping key
$(BSK_1, BSK_2, ..., BSK_n) \in \mathbb{T}[X]_N^{n\times(k+1)l\times(k+1)}$
Input: test polynomial $testP \in \mathbb{T}[X]_N^{(k+1)}$
Output: $c_{out} \in \mathbb{T}[X]_N^{(k+1)}$
$ACC \leftarrow X^{-b} \cdot testP$
`// BlindRotation`
for *i from* 1 *by* 1 *to* n **do**
  `// CMUX Operation`
  $ACC \leftarrow (X^{a_i} - 1) \cdot ACC \boxdot BSK_i + ACC$
end
return ACC

The most time-consuming part of PBS is the Blind Rotation, which is implemented through a large number of CMUX gates. CMUX operation is the homomorphic evaluation of the multiplexer, mainly consisting of polynomial additions and multiplications, making polynomial computations the major

workload in TFHE. (I)NTT is a widely employed method to perform fast multiplication on integer polynomials, reducing the complexity from N^2 to $NlogN$. Thus, (I)NTT operation is necessary for PBS hardware acceleration considering that the external product involves massive polynomial multiplications. Each iteration in the PBS process is comprised of several key components: the decomposition module to suppress the noise accumulation, multiple NTT operations to transfer the polynomials (ACC and $X^{a_i}-1$) to the NTT domain, and modular multiplications (ModMult) for the external product between ACC and BSK, and the INTT units to transfer the results back to the original domain.

III. Hardware Design

In this section, we first present our novel designs of the dominant modules in TFHE, along with the full-system architecture of UNIT. The employment of a technique called Bootstrapping Key Unrolling (BKU) is demonstrated later for further performance optimization.

A. negacyclic (I)NTT Design

(I)NTT modules consume a significant portion of the computing resources in the PBS process. It maps N polynomial coefficients $(x_0, x_1, ..., x_{N-1})$ in $\mathbb{Z}_q$ into NTT domain by the form $y(m) = \sum_{n=0}^{N-1} x_n \omega^{mn} \ mod\ q$, where twiddle factor ω is the primitive root modulo N in $\mathbb{Z}_q$. Similarly, INTT is given as $z(m) = N^{-1} \sum_{n=0}^{N-1} y_n \omega^{-mn} \ mod\ q$. The original multiplication result of two polynomials of degree N has the size of $2N$, which requires a reduction operation in further steps of PBS. To remove this reduction operation, Negative Wrapped Convolution is utilized, which can retrieve the resultant polynomial of degree N directly. (I)NTT supporting NWC called negacyclic (I)NTT, is achieved by using a twisting step before NTT's and after INTT's main stage, respectively. Assume $\mathbf{x}$ is the input polynomial of NTT and $\mathbf{y}$ is the output of INTT, the twisting is implemented by multiplying each element of the polynomials with the powers of the 2N-th roots of unity φ:

$$\begin{aligned} \widetilde{\mathbf{x}} &= (x[0], \varphi \cdot x[1], ..., \varphi^{N-1} \cdot x[N-1]) \\ \widetilde{\mathbf{y}} &= (y[0], \varphi^{-1} \cdot y[1], ..., \varphi^{-(N-1)} \cdot y[N-1]) \end{aligned} \tag{5}$$

Equation (5) indicates that extra N twisting factors, i.e., N powers of φ, are needed for each NTT/INTT operation, in addition to $N/2$ twiddle factors required in the main stage. This leads to a large demand for on-chip memory usage. Additionally, the extra multiplications increase latency, hindering performance. To address these limitations caused by NWC, we propose a highly unified reconfigurable (I)NTT module with fused twisting steps, as shown in Algo.2. In our approach, the preprocessing required for NTT and postprocessing for INTT are simplified based on our exploration of their inherent algebraic relationship with the main stage. This simplification reduces the extra multiplications to a pipelining operation in the main process, with only one modular multiplication increase in each processing element (PE). Furthermore, the required twisting factors are decreased from N to $logN$ in this design with the whole reduction of pre-stored parameters by 66%, thanks to the fusion of the twisting steps and the main stage of (I)NTT.

Algorithm 2: Highly Unified Reconfigurable (I)NTT Supporting Negative Wrapped Convolution

Input: $\mathbf{x}(x) \in \mathbb{Z}[x]/(x^N+1)$
Input: Transform Length N, Modulus q, PE number α
Input: ω: N-th roots of unity in $\mathbb{Z}_q$
Input: φ: 2N-th roots of unity in $\mathbb{Z}_q$
Output: $negacyclic\ (\mathbf{I})\mathbf{NTT}(\mathbf{x}) \in \mathbb{Z}[x]/(x^N+1)$
$k = log_2 N$;
$\beta = (2^{k-1})/\alpha$;
for l *from* k *by* -1 *to* 1 **do**
  for i *from* 0 *by* 1 *to* $(\alpha-1)$ **do**
    for j *from* 0 *by* 1 *to* $(\beta-1)$ **do**
      $\delta = i \cdot \beta + j$
      $\gamma = \delta + 2^{k-1}$
      `// configured as NTT operation`
      if $sel = 1$ **then**
      $\hat{\omega} = \omega^{\lfloor \frac{2\delta+1}{2^l} \rfloor 2^{l-1}}$
      `// Butterfly Operation //`
      $\mathbf{v}[\delta] = \mathbf{x}[2\delta] + \mathbf{x}[2\delta+1] \cdot \hat{\omega} \cdot \varphi^{2^{l-1}}\ mod\ q$
      $\mathbf{v}[\gamma] = \mathbf{x}[2\delta] - \mathbf{x}[2\delta+1] \cdot \hat{\omega} \cdot \varphi^{2^{l-1}}\ mod\ q$;
      `// configured as INTT operation`
      else
      $\hat{\omega} = \omega^{-\lfloor \frac{2\delta+1}{2^l} \rfloor 2^{l-1}}$
      `// Butterfly Operation //`
      $\mathbf{v}[\delta] = (\mathbf{x}[2\delta] + \mathbf{x}[2\delta+1] \cdot \hat{\omega}) \cdot 2^{-1}\ mod\ q$
      $\mathbf{v}[\gamma] = (\mathbf{x}[2\delta] - \mathbf{x}[2\delta+1] \cdot \hat{\omega}) \cdot 2^{-1} \cdot \varphi^{2^{k-l}}\ mod\ q$;
  $\mathbf{x} = \mathbf{v}$
return $\mathbf{x}$

The presented (I)NTT architecture is also scalable with a variable number of PEs. Each PE is reconfigurable and can be set to perform butterfly operations of NTT or INTT. The hardware architecture of the reconfigurable PE is demonstrated in Fig.1. Each PE requires two ModMult units, modular addition, and subtraction, several shift registers for shifting, and RightShift units for simplifying modular multiplication with 2^{-1}. Since the NTT and INTT operations do not overlap in the PBS process, only 4 (I)NTT modules are required to accomplish 4 NTT operations and 2 INTT operations in each PBS iteration. This approach helps reduce the resource utilization of (I)NTT operations by 33%. By carefully designing the PE architecture and leveraging the inherent algebraic relationship, our proposed (I)NTT module achieves improved performance, reduced memory and computing resources, and enhanced scalability compared to traditional implementations.

B. OF-MNTT Design

On top of (I)NTT operations for standard polynomials in the PBS process, the NTT transform for the monomial polynomial $\mathbf{m} = X^{a_i} - 1$ (MNTT) also requires careful

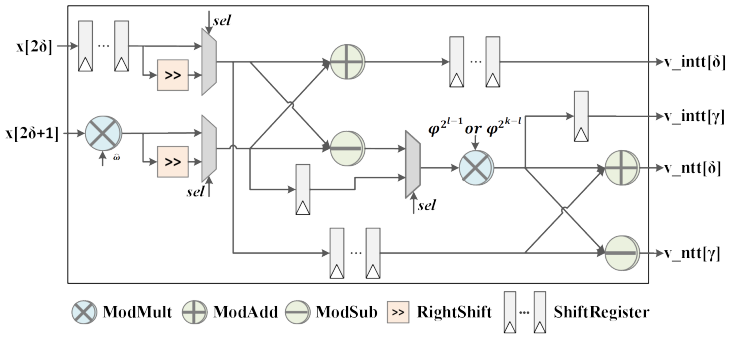

Fig. 1: Hardware Architecture of Reconfigurable Processing Element in (I)NTT module

design. Previous works have proposed two methods: Work [24] employs the typical NTT module to implement MNTT, which causes a waste of computing and memory resources considering the monomial is a special polynomial with only one non-zero element. Alternatively, ALT [25] introduces a lookup table (LUT)-like MNTT algorithm to achieve low computing cost. However, this approach requires substantial on-chip memory resources for storing pre-computed twiddle factors, especially when employing multiple MNTT units for high parallelism. We propose a novel MNTT design called OF-MNTT, which leverages on-the-fly twiddle factor generation to enable efficient computation of the monomial's NTT form while conserving memory resources, even at high parallelism. Let's first look at the algebraic expression of $\mathbf{m}$'s negacyclic NTT form with the knowledge that $\omega = \varphi^2 \ mod \ q$:

$$\begin{aligned} negacylicNTT(\mathbf{m}) &= ((\sum_{j=0}^{N-1} \omega^{a_i j} x^j) \cdot \varphi^{a_i}) - 1 \ mod \ q \\ &= ((\sum_{j=0}^{N-1} \varphi^{2a_i j} x^j) \cdot \varphi^{a_i}) - 1 \ mod \ q \\ &= (\sum_{j=0}^{N-1} \varphi^{(2j+1)a_i} x^j) - 1 \ mod \ q \end{aligned} \tag{6}$$

Equation (6) indicates that $negacylicNTT(\mathbf{m})$ can be obtained by calculating multiple specific powers of φ. If LUT-like MNTT [25] is used, $2N$ powers of φ should be stored on the chip for each MNTT operation when the parallelism degree is just 1. In contrast, our approach generates the first batch of twiddle factors on the fly by employing the parallel binary modular exponentiation algorithm, and then the remaining factors are obtained by multiplying the common ratios of the geometric progression inside as shown in Algo.3, which eliminates memory traffic and saves memory usage for loading and storing twiddle factors. This novel OF-MNTT approach results in a memory-efficient and highly parallelizable architecture while reducing the complexity from $NlogN$ to N.

C. Overall Architecture

The overall architecture with the data flow of UNIT is shown in Fig.2. The main functional modules are (I)NTT module, MNTT module, Decomposition (Decomp) module,

Algorithm 3: Proposed OF-MNTT Design

Input: Monomial polynomial $\mathbf{m} = \mathbb{X}^{a_i} - 1$
Input: Modulus q, PE number β
Input: φ: 2N-th roots of unity in $\mathbb{Z}_q$
Input: Pre-computed $\Psi_i = \varphi^{2^i} \ mod \ q$, $i \in [0, logN]$
Output: $\widetilde{\mathbf{m}} = negacyclic \ \mathbf{NTT}(\mathbf{m}) \in \mathbb{Z}[x]/(x^N + 1)$

```
// BinaryModularExponentiation Function
```

Function BME (a, Ψ, q)
  Binary representation of $(a \ mod \ 2N)$: $aBits$;
  $result = 1$;
  for i *from* $(len(aBits) - 1)$ *by* -1 *to* 0 **do**
    if $aBits[i] = 1$ **then** $result = result \cdot \Psi_i \ mod \ q$;
    else $result = result$;
  return $result$;

```
// Call BME function
```

$B_\beta \leftarrow$ BME $(2\beta a_i, \Psi, q)$;
for i *from* 0 *by* 1 *to* $(\beta - 1)$ **do**
  for j *from* 0 *by* 1 *to* $(N/\beta - 1)$ **do**
    if $j = 0$ **then** $\widetilde{\mathbf{m}}_i \leftarrow$ BME $((2i+1)a_i, \Psi, q)$;
    else $\widetilde{\mathbf{m}}_{j \cdot \beta + i} \leftarrow \widetilde{\mathbf{m}}_{(j-1) \cdot \beta + i} \cdot B_\beta mod \ q$;
return $\widetilde{\mathbf{m}}$

ModMult modules, and ModAdd module, each of which has its own computation units that can independently achieve the corresponding function. These units are called NTTU, MNTTU, DU, MMU, and MAU, respectively. Multiple processing elements (PEs) with varying numbers and local data memory are included in each NTTU or MNTTU, allowing for parallel processing. The ACC Memory provides the input data for each iteration, which is updated after obtaining the results of the last iteration. The Twiddle Factor Memory stores all the twiddle factors required for the (I)NTT module and can be accessed by all the NTTUs, while the BSK Memory stores the involved BSK data of one iteration. All the memory modules are implemented using BRAMs for fast and efficient data access.

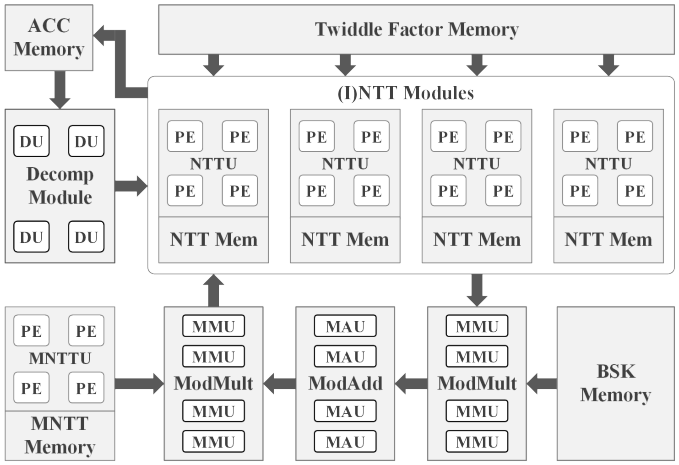

Fig. 2: Overall Architecture and Data Flow of UNIT

UNIT is designed to be scalable with varying degrees of parallelism at multiple levels, i.e. the number of computation units in each functional module and the number of PEs in each computation unit can be adjusted to accommodate different performance-resource requirements. Furthermore, UNIT is highly reconfigurable and can support different parameter sets for various application scenarios. This reconfigurability is achieved through our flexible functional module design including the (I)NTT, MNTT, and ModMult modules, which do not rely on specific parameters like special moduli or bit widths for optimization. This approach ensures that UNIT can be easily adapted to meet the diverse requirements of various TFHE applications.

D. Bootstrapping Key Unrolling

Bootstrapping Key Unrolling (BKU) is a technique proposed in [27] that increases the parallelism of the bootstrapping process by computing the values of multiple iterations in a single iteration. The main principle of BKU is to compute m bits of $\mathbf{s}$ concurrently, where m is the unrolling factor. As a result, BKU decreases the number of iterations from n to n/m and unrolls every m iterations. Take $m = 2$ as an instance, the CMUX operation can be rewritten as:

$$\begin{aligned} ACC \leftarrow & (X^{a_{2i-1}+a_{2i}} - 1) \cdot ACC \boxdot BSK_{i,0} \\ & + (X^{a_{2i-1}} - 1) \cdot ACC \boxdot BSK_{i,1} \\ & + (X^{a_{2i}} - 1) \cdot ACC \boxdot BSK_{i,2} + ACC \end{aligned} \tag{7}$$

where $i \in [1, n/2]$, i.e. only $n/2$ iterations are required in Blind Rotation. However, this comes at the cost of requiring more bootstrapping keys (BSK) and an increased number of MNTT operations per iteration, although the total number of (I)NTT operations remains unchanged.

Our design is memory-efficient, which naturally leads to the effective integration of BKU technology. The number of MNTT units and BSK memory size are configurable to support different unrolling factors m. However, the ModMult unit number is not adjusted along with m, considering the computing resource limitation. Therefore, the data flow should be modified accordingly to repeatedly utilize the original ModMult units $m^2 - 1$ times. This combination of BKU and modified data flow significantly reduces the latency without adding much hardware overhead. The implementation results across various rolling factors in Section IV-B showcase the benefits of BKU.

IV. Evaluation and Comparison

Our hardware design, UNIT, for TFHE is written in Verilog RTL, which is then synthesized, placed, and routed running at 200MHz using Xilinx Vivado 2022.2 on the Xilinx Alveo U250 FPGA platform. The FPGA has 1728K LUTs, 3456K FFs, 2688 BRAMs, and 12288 DSPs. Since the 110-bit security level is commonly recognized and employed in prior research, UNIT is implemented under this parameter set for fair comparison.

A. Comparison with CPU/GPU baselines

The CPU baselines are derived from the Concrete [15] and TFHE-rs library [16] while GPU-based designs are based on the cuFHE [18] and nuFHE [28] library for the implementation of TFHE bootstrapping. Prior works have built these libraries on various advanced platforms [13], [14], [16], [24], all of which are included in this work for a thorough comparison. As shown in Table I, our implementation UNIT with $m = 2$ achieves the performance improvement of more than 8.4× and 8.3× compared with the CPU-based and GPU-based baselines. This enhancement is mainly attributed to the superior parallel processing capacity offered by FPGA, which is leveraged by UNIT to exploit TFHE's intrinsic parallelism.

TABLE I: Comparison of Bootstrapping Latency with CPU/ GPU-based Implementations

Work	Platform	Latency (ms)	Speedup
Concrete [24]	AMD Threadripper 3990X CPU	62	**54.9**×
Concrete [13]	Intel Xeon Silver 4208 CPU	32	**28.3**×
Concrete [14]	Intel Xeon Platinum CPU	14	**12.4**×
TFHE-rs [16]	Intel Xeon Silver 4208 CPU	9.45	**8.4**×
nuFHE [14]	NVIDIA Titan RTX GPU	37	**32.8**×
cuFHE [24]	NVIDIA RTX 3090 GPU	9.34	**8.3**×
UNIT	Xilinx Alveo U250 FPGA	1.129	**-**

B. Comparison with FPGA-based Works

We compare our work with the state-of-the-art FPGA-based implementations in Table II regarding bootstrapping latency, resource utilization, power, and features. Configurability refers to the architecture's ability to support various parameter sets (security level, modulus, N, m), while scalability denotes whether the design is scalable with parallelism degree. ●, ◐, ○ here respectively represent that the accelerator provides sufficient, limited, and no configurability or scalability. We also introduce the area-time product (ATP) i.e. the product of area (LUT+FF consumption) and latency as a metric to quantify area efficiency, representing resource utilization efficiency.

As exhibited in Table II, UNIT presents superior configurability and scalability while enjoying lower latency, power, and higher area efficiency. Specifically, UNIT provides a performance enhancement of up to 3.63× and an ATP improvement of 8.66× compared with YKP [24] when the unrolling factor is the same. Even when work YKP is configured as higher unrolling factors $m = 3, 4$, UNIT at $m = 2$ can exhibit better performance and area efficiency with much lower memory resources and power. Work Kong [21] has limitations in supporting various moduli and parallelism degrees compared with UNIT. Meanwhile, UNIT outperforms work [21] up to 1.89× and 2.40× when $m = 2$ in terms of performance and area efficiency, respectively. The DSP consumption in work Kong [21] is extremely low due to its careful selection of a single modulus to eliminate multiplication operations in the modular reduction unit, which will hinder its applicability in real-world scenarios. Both YKP [24] and Kong [21] rely on large on-chip memory and logic cells provided by Xilinx Virtex VU13P FPGA for a

TABLE II: Comparison of Bootstrapping Latency and Resources with FPGA-based Implementations

Work	FPGA Platform	Con†	Scalability	Unrolling Factor	Frequency (MHz)	Latency (ms)	Resources (LUT, FF, DSP, SRAM)	ATP‡	Power (W)
Security level $= 110bit, N = 1024, n = 500, l = 2, B = 2^8$									
YKP [24]	Xilinx Virtex VU13P	●	●	1	180	7.53	925K 729K 6240 319Mb	12455K	50
YKP [24]	Xilinx Virtex VU13P	●	●	2	180	3.76	842K 662K 7202 338Mb	5655K	50
YKP [24]	Xilinx Virtex VU13P	●	●	3	180	2.51	569K 448K 6640 383Mb	2553K	50
YKP [24]	Xilinx Virtex VU13P	●	●	4	180	1.88	442K 342K 6910 409Mb	1474K	50
Kong [21]	Xilinx Virtex VU13P	◐	○	2	200	2.13	414K 625K 1281 40Mb	2213K	-
Kong [21]	Xilinx Virtex VU13P	◐	○	3	200	1.43	510K 750K 1281 56Mb	1802K	-
FPT [13]	Xilinx Alveo U280	◐	○	-	200	0.74	595K 1024K 5980 14.5Mb	1198K	99
UNIT	Xilinx Alveo U250	●	●	1	200	2.077	377K 316K 6060 18.3Mb	1439K	27.5
UNIT	Xilinx Alveo U250	●	●	2	200	1.129	450K 368K 6916 31.3Mb	923K	30.6

† configurability means whether the architecture can support various parameter settings.
‡ATP: Area (LUT+FF usage) Time (latency) Product (lower ATP means better area efficiency)

high degree of parallelism, while UNIT can achieve high parallelism with small data memory and resources on Xilinx Alveo U250 FPGA thanks to our parallelism-friendly design. FPT [13] provides lower latency than our proposed UNIT due to its approximation method, but UNIT enjoys higher flexibility with significantly lower FF utilization and power, achieving a 30% improvement in resource efficiency and over 3.2× reduction in energy. It should be noted that FPT utilizes lower SRAM consumption by employing high bandwidth memory (HBM) provided by Xilinx Alveo U280 for high-speed on-chip and off-chip communication. Moreover, FPT introduces additional noise growth due to its fixed fraction bits for FFT design and does not provide the experimental noise value or decryption failure probability. Work ALT [25] modifies the parameters of the PBS algorithm and thus its results are not shown in Table II. Although utilizing these adjusted parameters can decrease latency and resource consumption evidently, we claim that this modification is an approximation method that sacrifices the accuracy and introduces extra noise. By contrast, UNIT introduces no approximations that lead to DFR growth. In summary, UNIT achieves a balance among performance, computational and memory resources, flexibility, and accuracy compared to state-of-the-art works.

To ensure fairness, comparisons against ASIC-based designs are not presented. Unlike our focus on the flexible and practical implementation of TFHE bootstrapping, these ASIC designs [14], [20]–[22] remain in the simulation stage and are difficult to manufacture. They also significantly rely on considerably high external bandwidths of up to 300 GB/s and heavy computational resources.

C. Scalability

In this section, we present the design-space exploration in terms of parallelism degree (i.e. PE number) in (I)NTT and MNTT modules to demonstrate the scalability of our work. The implementation results of (I)NTT and MNTT modules, including latency and resource utilization with various PE numbers, are presented in Fig.3 and Fig.4 under the parameter set $N = 1024$, respectively. The number of PEs reflects the degree of parallelism in one operation. As the PE number increases, the latency declines while resource consumption rises correspondingly. Thus, scalability is heavily associated with the performance and area efficiency of the modules, which can be a trade-off for performance-resource balance.

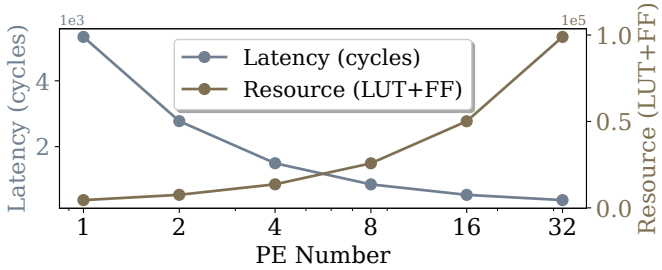

Fig. 3: Latency and Resource Utilization of (I)NTT Module with Varied PE Numbers

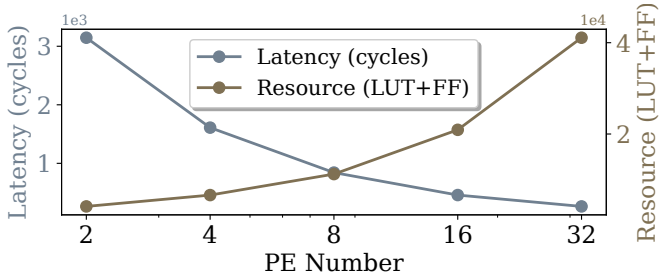

Fig. 4: Latency and Resource Utilization of MNTT Module with Varied PE Numbers

V. Conclusion

In this work, we present a novel FPGA-based accelerator called UNIT for TFHE Bootstrapping. Our main contributions lie in the highly unified reconfigurable architecture with fused twisting stages for negacyclic (I)NTT, as well as the memory-efficient and highly parallelizable OF-MNTT module that generates twiddle factors on the fly. The comparison with state-of-the-art works demonstrates that our proposed accelerator achieves lower latency, reduced memory and computing resources, and enhanced reconfigurability.

VI. Acknowledgement

This work was supported by AI Chip Center for Emerging Smart Systems, and Collaborative Research Fund C5032-23G.

REFERENCES

[1] M. Hamdaqa and L. Tahvildari, "Cloud computing uncovered: A research landscape." *Advances in Computers*, vol. 86, pp. 41–85, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/ac/ac86.html

[2] F. Han, J. Qin, and J. Hu, "Secure searches in the cloud," *Future Gener. Comput. Syst.*, vol. 62, no. C, p. 66–75, sep 2016. [Online]. Available: https://doi.org/10.1016/j.future.2016.01.007

[3] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, jan 2019. [Online]. Available: https://doi.org/10.1145/3298981

[4] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012, https://eprint.iacr.org/2012/144. [Online]. Available: https://eprint.iacr.org/2012/144

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 309–325. [Online]. Available: https://doi.org/10.1145/2090236.2090262

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3164123

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[8] L. Ducas and D. Micciancio, "Fhew: bootstrapping homomorphic encryption in less than a second," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 617–640.

[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*. Springer, 2016, pp. 3–33.

[10] J. Zhang, X. Cheng, L. Yang, J. Hu, X. Liu, and K. Chen, "Sok: Fully homomorphic encryption accelerators," 2023.

[11] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," in *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*. Springer, 2021, pp. 1–19.

[12] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[13] M. Van Beirendonck, J.-P. D'Anvers, F. Turan, and I. Verbauwhede, "Fpt: A fixed-point accelerator for torus fully homomorphic encryption," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 741–755.

[14] A. Putra, Prasetiyo, Y. Chen, J. Kim, and J.-Y. Kim, "Strix: An end-to-end streaming architecture with two-level ciphertext batching for fully homomorphic encryption with programmable bootstrapping," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1319–1331.

[15] I. Chillotti, M. Joye, D. Ligier, J.-B. Orfila, and S. Tap, "Concrete: Concrete operates on ciphertexts rapidly by extending tfhe," in *WAHC 2020-8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2020.

[16] Zama, "TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data," 2022, https://github.com/zama-ai/tfhe-rs.

[17] W. Dai and B. Sunar, "cuhe: A homomorphic encryption accelerator library," in *Cryptography and Information Security in the Balkans: Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers 2*. Springer, 2016, pp. 169–186.

[18] C. Gizem, D. Wei, O. Bogdan, and Kitsu, "Cuda-accelerated fully homomorphic encryption library," 2019, https://github.com/vernamlab/cuFHE.

[19] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," in *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*. Springer, 2021, pp. 1–19.

[20] L. Jiang, Q. Lou, and N. Joshi, "Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 235–240.

[21] T. Kong and S. Li, "Hardware acceleration and implementation of fully homomorphic encryption over the torus," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[22] A. Putra, J.-Y. Kim *et al.*, "Morphling: A throughput-maximized tfhe-based accelerator using transform-domain reuse," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 249–262.

[23] S. Gener, P. Newton, D. Tan, S. Richelson, G. Lemieux, and P. Brisk, "An fpga-based programmable vector engine for fast fully homomorphic encryption over the torus," in *SPSL: Secure and Private Systems for Machine Learning (ISCA Workshop)*, 2021.

[24] T. Ye, R. Kannan, and V. K. Prasanna, "Fpga acceleration of fully homomorphic encryption over the torus," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–7.

[25] X. Hu, Z. Li, Z. Wang, and X. Lu, "Alt: Area-efficient and low-latency fpga design for torus fully homomorphic encryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024.

[26] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 377–408.

[27] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38*. Springer, 2018, pp. 483–512.

[28] O. Bogdan, P. Derek, Tux, and F. Max, "A gpu implementation of fully homomorphic encryption on torus," 2020, https://github.com/nucypher/nufhe.