electronics

MDPI

Article

Efficient Twiddle Factor Generators for NTT

Nari Im [1], Heehun Yang [1], Yujin Eom [1], Seong-Cheon Park [2] and Hoyoung Yoo [1,*]

[1] Department of Electronics, Chungnam National University, Daejeon 34134, Republic of Korea; narisharon@o.cnu.ac.kr (N.I.); hhyang@o.cnu.ac.kr (H.Y.); aremda2320@o.cnu.ac.kr (Y.E.)
[2] Electronics and Telecommunications Research Institute, 3F Research Bldg A, Global R&D Center, 712-22 Daewangpangyo-ro, Bundang-gu, Seongnam-si 13488, Republic of Korea; scpark@etri.re.kr
* Correspondence: hyyoo@cnu.ac.kr

Abstract: Fully Homomorphic Encryption (FHE) allows computations on encrypted data without decryption, providing strong security for sensitive information. However, computational and memory demands for FHE are significant challenges, particularly in the Number Theoretic Transform (NTT) phase. This paper presents three efficient Twiddle Factor Generators (TFGs) to address these challenges: the Half-Memory TFG, the On-the-fly Serial TFG, and the On-the-fly Parallel TFG. The Half-Memory TFG reduces memory usage by storing only half of the twiddle factors and calculating the rest as needed. The On-the-fly Serial TFG eliminates memory requirements by computing twiddle factors, while the On-the-fly Parallel TFG enhances computational speed through parallel processing. Implemented on the FPGA KCU105 board, these TFGs demonstrated significant improvements in hardware resource utilization and computational efficiency. The Half-Memory TFG effectively reduces memory footprint, the On-the-fly Serial TFG eliminates memory usage with acceptable computational overhead, and the On-the-fly Parallel TFG offers superior performance for high-throughput applications. These innovations make FHE more practical for real-world applications, contributing to the broader goal of enabling secure, privacy-preserving computations on encrypted data.

Keywords: twiddle factor; twiddle factor generator; polynomial multiplication; number theoretic transform; fully-parallel architecture; fully homomorphic encryption

check for updates

Citation: Im, N.; Yang, H.; Eom, Y.; Park, S.-C.; Yoo, H. Efficient Twiddle Factor Generators for NTT. *Electronics* **2024**, *13*, 3128. https://doi.org/10.3390/electronics13163128

Received: 17 June 2024
Revised: 24 July 2024
Accepted: 4 August 2024
Published: 7 August 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Recent advancements in cloud-based Artificial Intelligence (AI) have significantly increased the importance of data privacy and security. As organizations and individuals increasingly rely on cloud services for data storage and processing, protecting sensitive information from unauthorized access has become paramount. Effective encryption algorithms are crucial in ensuring that large volumes of information can be securely stored and processed without compromising data integrity or confidentiality. Among various encryption algorithms, Fully Homomorphic Encryption (FHE) has gained considerable attention for its unique ability to perform computations on encrypted data without needing decryption. This property enables secure data processing in untrusted environments, such as cloud computing platforms, where data privacy concerns are paramount [1,2]. Unlike traditional encryption methods, which require decryption before any operation can be performed on the data, FHE maintains the confidentiality of the data throughout the computation process, thus significantly enhancing security. However, FHE inherently requires significantly greater hardware complexity compared to classical encryption algorithms. This is because FHE schemes involve performing complex mathematical operations directly on encrypted data, which imposes substantial computational overhead. Consequently, the hardware complexity of FHE is recognized as one of the key challenges that must be addressed for its successful application [3,4].

The concept of FHE was first introduced by Craig Gentry in 2009. Gentry's groundbreaking work provided the first construction of an FHE scheme based on lattice-based

problems, specifically the Learning With Errors (LWE) problem and the Ring Learning With Errors (RLWE) problem [5]. Gentry's scheme not only demonstrated the feasibility of FHE but also introduced a blueprint for constructing general FHE schemes capable of performing arbitrary computations on encrypted data. Since Gentry's pioneering work, numerous FHE schemes have been proposed, each aiming to improve the efficiency and practicality of FHE. Notable examples include the Brakerski–Gentry–Vaikuntanathan (BGV) scheme [6], the Brakerski–Fan–Vercauteren (BFV) scheme [7], and the Cheon–Kim–Kim–Song (CKKS) scheme [8]. These schemes have made significant strides in reducing the computational overhead and improving the performance of FHE, making it more viable for practical applications.

A critical aspect of most FHE schemes is the reliance on polynomial operations over finite fields. Polynomial multiplication, in particular, is one of the most computationally intensive and time-consuming operations in FHE [9]. As such, it often becomes a performance bottleneck [10]. To mitigate this issue, the Number Theoretic Transform (NTT) is commonly employed to optimize polynomial multiplication [11–15]. The NTT reduces the time complexity of polynomial multiplication from $\mathcal{O}(N^2)$ to $\mathcal{O}(NlogN)$ where N is the degree of the polynomial. The NTT is a specialized form of the Fast Fourier Transform (FFT) adapted for finite fields [16]. While the FFT operates on complex numbers, the NTT operates on integer numbers within finite fields, making it particularly suitable for lattice-based cryptographic schemes like FHE [9,17]. The NTT transforms polynomial coefficients into the frequency domain, enabling efficient pointwise multiplication. After the multiplication, an inverse NTT (INTT) is used to convert the results back to the time domain. Despite its advantages, implementing the NTT requires substantial hardware resources, particularly for storing the Twiddle Factors (TFs) used during the transformation. This memory requirement poses a significant challenge, especially in resource-constrained environments. Therefore, efficient Twiddle Factor Generators (TFGs) are essential to reduce memory usage and enhance the overall performance of the NTT in FHE applications [18,19].

In this paper, we propose three innovative Twiddle Factor Generators (TFGs) aimed at improving hardware efficiency: Half-Memory-based TFG, On-the-fly Serial TFG, and On-the-fly Parallel TFG. The main contributions of this paper are summarized as follows:

- We propose a Half-Memory TFG that reduces the memory size required for TF storage by half, while doubling the throughput.
- We introduce two on-the-fly TFGs that do not utilize memory for storing TFs, offering significant enhancements in memory utilization for FHE schemes requiring various sets of TFs for NTT implementation.
- We evaluate the proposed TFGs using the Figure of Merit (FoM) to assess their performance in terms of equivalence.

The rest of this paper is organized as follows: Section 2 provides the theoretical background of the Number Theoretic Transform and Twiddle Factor calculation. Section 3 discusses the proposed TFGs. Section 4 presents the experimental results and analysis. Finally, Section 5 concludes the paper.

2. Background

This section delves into the core operations of Fully Homomorphic Encryption (FHE), focusing on the Number Theoretic Transform (NTT) and the Twiddle Factor Generators (TFGs). These components are pivotal in optimizing the performance of FHE schemes, which enable efficient polynomial multiplication computations on encrypted data while preserving data privacy and security.

2.1. Number Theoretic Transform

Most Fully Homomorphic Encryption (FHE) schemes operate on polynomial rings over finite fields, which are typically represented in the form $\mathcal{R}_q[x] = \mathbb{Z}_q[x]/f(x)$ [20]. In this notation, $\mathcal{R}_q[x]$ denotes the ring of polynomials modulo $f(x)$, where $f(x)$ is usually chosen as $x^N + 1$. Here, $\mathbb{Z}_q$ signifies the finite field with elements represented as integers

modulo q [9,19]. This setup ensures that polynomial coefficients are constrained within a finite set of integers, which is critical for maintaining arithmetic operations within a manageable range and ensuring security in cryptographic applications.

A general polynomial $a(x)$ within this ring can be expressed as

$$a(x) = a_0 + a_1x^1 + a_2x^2 + \ldots + a_{N-1}x^{N-1} = \sum_{i=0}^{N-1} a_i x^i \in \mathcal{R}_q. \quad (1)$$

In this expression, a_i represents the coefficients of the polynomial, and x is the indeterminate. The polynomial ring $\mathcal{R}_q[x]$ essentially denotes that we are working within the constraints of modular arithmetic defined by the polynomial $f(x)$ and the finite field $\mathbb{Z}_q$. Polynomial multiplication is a fundamental operation in cryptographic computations, particularly in FHE schemes. However, this operation is computationally intensive, with a time complexity of $\mathcal{O}(N^2)$. The multiplication of two polynomials, $a(x) = \sum_{i=0}^{N-1} a_i x^i$ and $b(x) = \sum_{i=0}^{N-1} b_i x^i$, involves computing the product polynomial $c(x) = \sum_{i=0}^{N-1} \sum_{i=0}^{N-1} a_i b_j x^{i+j}$. This double summation results in a complexity that grows quadratically with the number of terms, making polynomial multiplication a significant bottleneck in cryptographic processes.

To overcome this inefficiency, the Number Theoretic Transform (NTT) is employed, which significantly accelerates polynomial multiplication. NTT reduces the time complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(NlogN)$ by transforming polynomial coefficients into the frequency domain [21]. This transformation involves multiplying the coefficients by constant factors known as Twiddle Factors (TFs), represented as W^{ij}, where W is a primitive Nth root of unity modulo q. In the frequency domain, polynomial multiplication is reduced to pointwise multiplication, which is computationally less demanding.

After the pointwise multiplication is completed in the frequency domain, an inverse NTT (INTT) operation is applied to convert the result back into the time domain. This reconstruction is necessary to obtain the final result of the polynomial multiplication in the original polynomial form. The NTT and INTT operations can be expressed mathematically as

$$A_i = \sum_{j=0}^{N-1} a_j W^{ij} \; mod \; q, \quad (2)$$

where A_i represents the coefficients in the frequency domain, W^{ij} are the Twiddle Factors, and q is the modulus used to keep the coefficients within the finite field.

$$a_i = N^{-1} \sum_{j=0}^{N-1} A_j W^{-ij} \; mod \; q. \quad (3)$$

where N^{-1} denotes the modular inverse of N, ensuring that the result is scaled correctly to match the original polynomial coefficients. This process effectively reconstructs the polynomial from its frequency domain representation back into the time domain, completing the polynomial multiplication operation. These mathematical transformations are fundamental for optimizing polynomial operations in FHE schemes and are essential for the efficient implementation of cryptographic protocols that rely on polynomial arithmetic.

2.2. Twiddle Factor Calculation

Twiddle Factors (TFs) are a crucial component in the NTT used for efficient polynomial multiplication in Fully Homomorphic Encryption (FHE) schemes. Twiddle Factors denoted as W^i are elements of a finite field used in the context of the NTT. In the case of the NTT, these factors are defined as powers of a primitive root of unity. A primitive root of unity W in a finite field $\mathbb{Z}_q$ is an element that generates all the non-zero elements of the field when raised to various powers, modulo q. This means that W satisfies the equation $W^n = 1 \bmod q$, where N is the size of the polynomial or the length of the transform.

Figure 1 illustrates an example of primitive roots of unity when n is 16 and q is 17. The primitive 16-th roots of unity modulo 17, denoted as $W = 3$, implies that every integer in mod 17 can be represented as a power of 3. As shown in Figure 1, every non-zero number from 1 to 16 can be expressed as a power of 3 modulo 17.

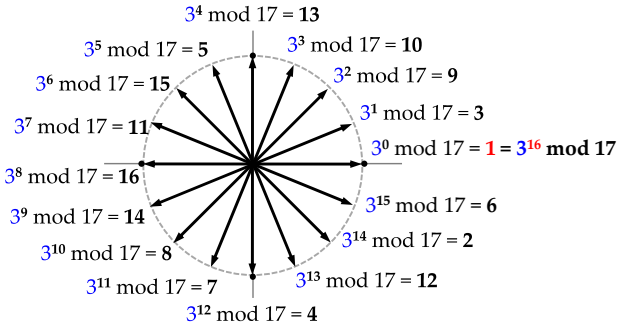


Figure 1. Primitive roots of unity for $N = 16$.

The primary purpose of Twiddle Factors is to enable efficient polynomial multiplication by transforming polynomial coefficients into a frequency domain where pointwise multiplication is more straightforward, according to (2). In this transformed domain, polynomial multiplication becomes a simple element-wise multiplication of these transformed coefficients. After performing the polynomial multiplication in the frequency domain, it is necessary to revert to the time domain to obtain the result in its original polynomial form. This is achieved through the INTT, which involves the inverse of the Twiddle Factors, according to (3). Overall, Twiddle Factors are indispensable in the realm of FHE and polynomial arithmetic, facilitating the efficient and secure handling of complex polynomial operations within cryptographic systems.

2.3. Implementation of NTT

When implementing NTT, the Cooley–Tukey (CT) algorithm [22] is commonly used, due to its efficiency and flexibility, similar to FFT. The CT algorithm optimizes performance through a recursive divide-and-conquer approach, breaking down the problem into smaller, more manageable components. The CT algorithm is summarized in Algorithm 1. In practice, the CT algorithm works by processing two polynomial coefficients at a time. Initially, one coefficient is multiplied by a Twiddle Factor (TF), and modular reduction is applied. The results are then combined to produce two new coefficients. This process is repeated at each stage of the algorithm, progressively simplifying the polynomial multiplication. Figure 2 illustrates the block diagram of CT algorithm, which serves as a processing element (PE) in the NTT architecture. As represented in Algorithm 1, the PE operates on two polynomial coefficients, denoted as a_i and a_{i+t}. Initially, a_{i+t} is multiplied by the TF, and then modular reduction is performed. Subsequently, the output A_i is computed by adding a_i to the operated a_{i+t} and then performing modular reduction. Similarly, the output A_{i+t} is obtained by subtracting a_i from the operated a_{i+t} and performing modular reduction. According to Algorithm 1 and Figure 2, this method provides a systematic approach to reduce the time complexity of polynomial multiplication from $\mathcal{O}(N^2)$ to $\mathcal{O}(NlogN)$.

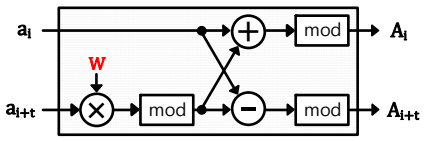


Figure 2. Block diagram of Processing Element (PE).

The entire process of NTT, which can be implemented in various architectures based on PE, is described in Algorithm 1 [23]. However, in this paper, NTT is implemented in a single-path delay feedback architecture, as shown in Figure 3. This architecture sequentially receives N polynomial coefficients $(\text{IN} : a_0, a_1, a_2, \ldots, a_{N-1})$ and operates along a single path over $log_2 N$ stages. Since one coefficient is received at a time, specific input coefficients need to be stored in the feedback register to be processed by the PE. These stored coefficients are operated when they come out of the feedback register. Moreover, the feedback register

stores the result of the PE, indicated as A_{i+t}, as a single datum is transferred to the next stage. Consequently, this architecture utilizes log_2N multipliers, log_2N adders, log_2N subtractors, and $3log_2N$ modular reductions for PE. Additionally, $(N-1)$ feedback registers are used for implementing the serial data flow.

Algorithm 1: NTT Based on the Cooley–Tukey (CT) Algorithm

Input:

- Vectors of polynomial coefficients $a = (a_0, a_1, a_2, \ldots, a_{N-1}) \in \mathbb{Z}_q^N$
- Twiddle Factors $W = (W^0, W^1, \ldots, W^{N-1}) \in \mathbb{Z}_q^N$

Output:

- $A \Leftarrow \mathrm{NTT}(a)$

```
1.   t = N;
2.   for (m = 1; m < N; m = 2m) do     // NTT
3.           t = t/2;
4.       for (i = 0; i < m; i++) do
5.           j1 = 2 · i · t;
6.           j2 = j1 + t − 1;
7.           S = W[m + i];
8.               for ( j = j1; j < j2; j++) do          // Processing Element
9.                   U = a[j];
10.                  V = a[j + t] · S;
11.                  A[j] = U + V mod q;
12.                  A[j + t] = U − V mod q;
13.  return A;
```

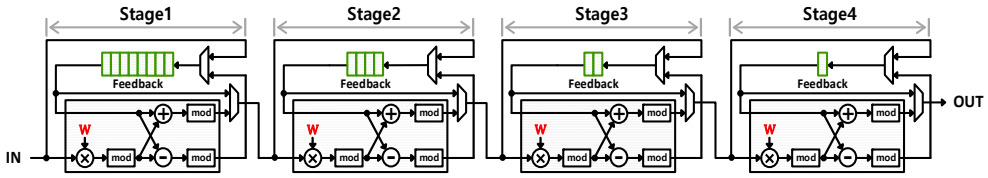


Figure 3. Diagram of NTT for $N = 16$.

2.4. Conventional Memory-Based TFG

In the traditional approach to implementing TFGs for NTT, a conventional memory-based TFG is employed. This approach relies on pre-calculating and storing the TFs in a fixed memory structure, such as Read-Only Memory (ROM). The main advantage of this method is the rapid access to precomputed TF values, which can significantly speed up the computation process during NTT. The TFs are essential for transforming polynomial coefficients into the frequency domain, where pointwise multiplication is performed. The TFs are derived from the primitive n-th roots of unity, and their computation typically involves raising these roots to specific powers and taking the result modulo as a prime number q.

In a memory-based TFG, the TFs are computed beforehand and stored in ROM. The size of the ROM required depends on the number of TFs and their bit width. For example, if the polynomial degree is N and the *bit_width* of the modular constant q is 64 bits, the total memory size required for storing TFs can be significant. Specifically, it scales with $N \times (bit_width)$ bits. To illustrate this with an example, suppose N is 2^{16} and the *bit_width* of q is 64 bits. The total memory required for TF storage would be $2^{16} \times 64$ bits, which amounts to 524 KB of memory. This storage requirement can be substantial, especially for larger polynomial degrees or when dealing with multiple sets of TFs for different modular constants.

Figure 4 illustrates the conventional memory-based TFG integrated with the SDF structure from Figure 3. This setup operates according to Equations (2) and (3) to perform

both NTT and INTT operations. Notably, the TFG structure is designed to provide pre-stored TFs as requested, at each stage of the process. While the conventional memory-based approach offers the advantage of quick and straightforward access to TFs, it has a significant drawback: the need for a substantial amount of ROM to store the TFs and the time required to initialize this memory.

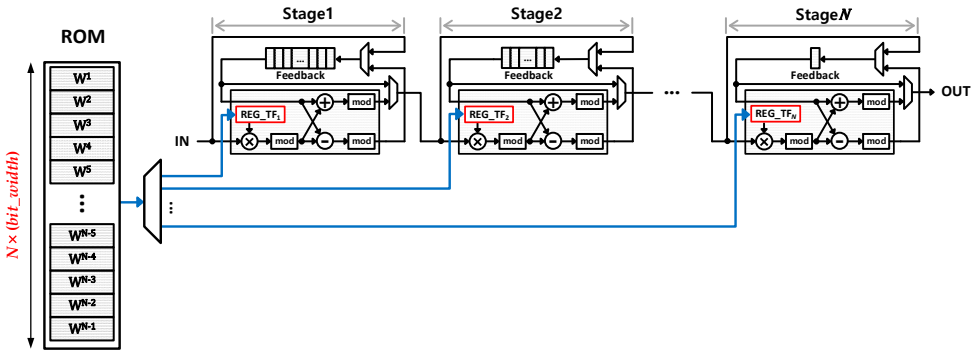


Figure 4. Block diagram of conventional Memory-based TFG.

3. Proposed Twiddle Factor Generators

In this section, we present three novel Twiddle Factor Generators (TFGs) designed to address the limitations of conventional memory-based TFGs. Each proposed TFG aims to optimize memory usage and enhance computational efficiency in different ways. The proposed generators include the Half-Memory TFG, the On-the-Fly Serial TFG, and the On-the-Fly Parallel TFG. These TFGs are introduced to reduce memory footprint and improve performance while maintaining the required functionality for Fully Homomorphic Encryption (FHE) operations.

3.1. Proposed Half-Memory TFG

The Half-Memory TFG leverages the inherent symmetry of TF to reduce memory usage. Twiddle Factors exhibit symmetrical properties due to their origins as primitive n-th roots of unity. For example, if W is a primitive root of unity,

$$W^n = \left(W^{n/2}\right)^2 = 1 \ mod \ q, \tag{4}$$

which implies

$$W^{n/2} = -1 \ mod \ q. \tag{5}$$

Since a root of unity W is considered primitive if it satisfies $W^n = 1 \ mod \ q$ and does not satisfy $W^m = 1$ for any integer m smaller than n, due to this characteristic, the following equations show that $W^{n/2}$ can be derived as -1, as it has to be an integer that is not equal to 1.

In addition, modular arithmetic has a multiplication property, which states that $(a \times b) \ mod \ q = [(a \ mod \ q) \times (b \ mod \ q)]$ mod q. This property ensures that the result of multiplying two numbers of modulo q is equivalent to the product of their respective remainders when each is taken as modulo q. Therefore, the symmetry property of TFs can be demonstrated as follows:

$$\begin{aligned} W^{n/2+i} \ mod \ q &= (W^{n/2} \ mod \ q) \times (W^i \ mod \ q) \\ &= -W^i \ mod \ q \\ &= q - W^i. \end{aligned} \tag{6}$$

This symmetry means that we only need to compute half of the Twiddle Factors, as the remaining factors can be derived from these precomputed values. This symmetry implies that half of the TFs can be computed directly, using the other half. Figure 5 shows the

computation using the symmetry property. For instance, when $n/2$ is 8, 3^{8+1} *mod* 17 can be computed as $17 - (3^1 \ mod \ 17) = 17 - 3 = 14$. The Half-Memory TFG is designed to exploit this symmetry, effectively halving the memory required for storing TFs compared to traditional methods. This generator operates by calculating half of the TFs directly and deriving the remaining TFs using subtraction operations. As shown in Figure 6, this approach minimizes memory usage to $N/2 \times (bit_width)$ bits, thus improving efficiency without sacrificing performance.

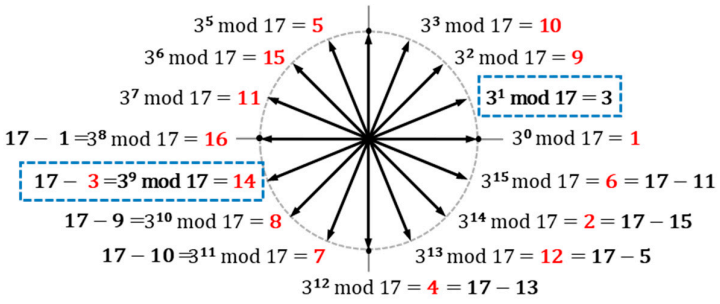


Figure 5. Symmetry properties of primitive roots of unity for $N = 16$.

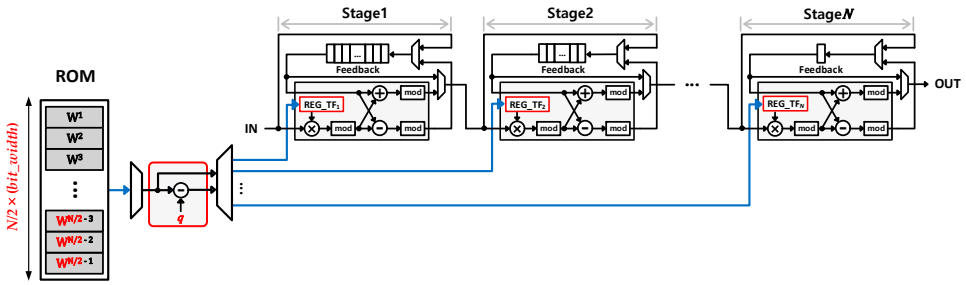


Figure 6. Block diagram of proposed Half-Memory-based TFG.

3.2. Proposed On-the-Fly Serial TFG

The proposed On-the-Fly Serial TFG operates without the need to pre-store Twiddle Factors. Instead, it generates TFs dynamically during runtime. This TFG calculates TFs on-the-fly using the following recursive relationship:

$$W^i = W^{i-1} \times W \ mod \ q. \tag{7}$$

As illustrated in Figure 7, the On-the-Fly Serial TFG uses this relationship to compute TFs sequentially. The generator replaces the conventional memory storage with a multiplexer, a flip-flop, and a modular reduction unit. This setup significantly reduces memory requirements by eliminating the need for storing precomputed TFs. This approach is particularly beneficial for FHE algorithms requiring various sets of TFs, as it avoids the overhead associated with memory storage and initialization. By generating TFs on-the-fly, the TFG optimizes memory utilization and ensures that only the necessary TFs are computed as needed.

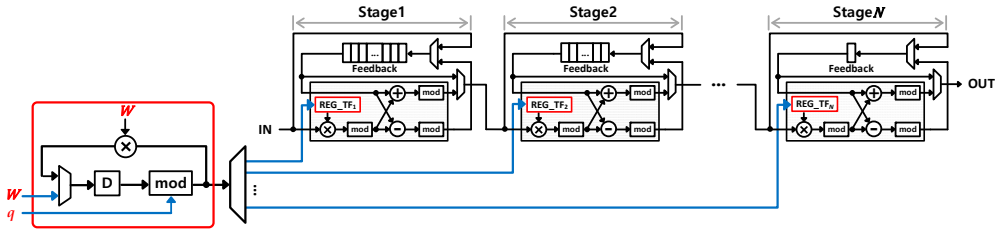


Figure 7. Block diagram of proposed On-the-fly Serial (O-Serial) TFG.

3.3. Proposed On-the-Fly Parallel TFG

The proposed On-the-Fly Parallel TFG extends the concept of the On-the-Fly Serial TFG by employing a parallel processing approach to further enhance performance. This generator calculates multiple TFs simultaneously, thus increasing throughput and reducing the overall computation time. The proposed O-Parallel TFG can be expressed by the following equation:

$$W^{pi+j} = W^{p(i-1)+j} \times W^{p} \ mod \ q, \quad for \ 0 \leq j \leq \boldsymbol{p} - 1 \tag{8}$$

where $\boldsymbol{p}$ represents the number of parallel processing units. For example, 4-parallel of O-Parallel TFG generates four TFs simultaneously as follows: (W^1, W^2, W^3, W^4), (W^5, W^6, W^7, W^8), ..., and $(W^{N-4}, W^{N-3}, W^{N-2}, \ldots, W^{N-1})$.

Figure 8 demonstrates the parallel architecture of this TFG, which generates TFs in parallel streams, thereby improving computational efficiency. By implementing a parallel structure, the On-the-Fly Parallel TFG enhances throughput by a factor of $\boldsymbol{p}$ and increases area utilization proportionally. This design is ideal when performance is a primary concern, as it offers significant improvements in speed and efficiency compared to the On-the-Fly Serial TFG. The choice of $\boldsymbol{p}$ allows for balancing between performance gains and hardware resource usage.

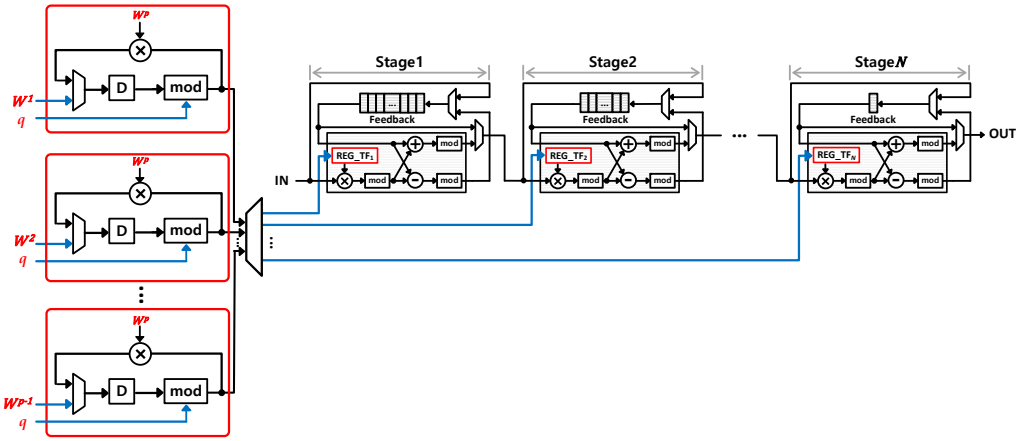


Figure 8. Block diagram of proposed On-the-fly Parallel (O-Parallel) TFG.

4. Experimental Results and Discussion

This section presents the experimental results for the proposed Twiddle Factor Generators (TFGs), showcasing their performance compared to conventional memory-based approaches. The goal is to evaluate the efficiency improvements and practical benefits of each TFG in real-world applications of Fully Homomorphic Encryption (FHE). To assess the effectiveness of the proposed TFGs, we conducted a series of experiments focusing on memory usage, computation time, and overall efficiency. We compared the proposed Half-Memory TFG, the On-the-Fly Serial TFG, and the On-the-Fly Parallel TFG against a straightforward memory-based TFG.

Firstly, Table 1 presents a qualitative comparison of various TTFGs in the context of N-polynomial NTT. The conventional Memory-based TFG, as depicted in Figure 4, is the most straightforward and intuitive approach. It computes N TFs in advance and stores them in a memory with N elements for subsequent use. While this method is simple and easy to understand, it suffers from significant hardware complexity due to its large memory size requirements. In contrast, the proposed Half-Memory TFG in Figure 6 reduces the memory usage required by the conventional approach. By leveraging the symmetric properties of TFs, the Half-Memory TFG reduces the memory requirement from N to $N/2$. It generates the remaining TFs using a subtractor, and since it can create two TFs simultaneously, it achieves twice the throughput compared to the conventional

Memory-based TFG. Additionally, the on-the-fly generation methods, namely the On-the-Fly Serial TFG and On-the-Fly Parallel TFG, offer alternative approaches. These methods eliminate the need for memory by generating TFs dynamically using D Flip-Flops (D-FFs), multipliers, and modular reduction circuits. According to Figure 7, the O-Serial TFG generates one TF at a time, requiring a single set of D flip-flops, multipliers, and modular reduction circuits. This design has the lowest hardware complexity among the proposed structures. To further enhance throughput, the On-the-Fly Parallel TFG has been proposed. As shown in Figure 8, this structure can achieve p times the throughput of the O-Serial TFG by generating multiple TFs simultaneously. However, this comes at the cost of increased hardware complexity, which is $\boldsymbol{p}$ times greater than that of the O-Serial TFG.

Table 1. Quantitative comparison of TFGs.

	Conventional Memory	**Proposed 1 Half-Memory**	**Proposed 2 O-Serial**	**Proposed 3 O-Parallel (p)**
Memory Size	N	$N/2$	-	-
D flip-flops	-	-	1	p
Subtractors	-	1	-	-
Multiplier	-	-	1	p
Modular Reductions	-	-	1	p
Throughput	1	2	1	p

Next, Table 2 provides a quantitative comparison of various TFGs. For the experiment, we implemented TFGs using Verilog HDL on the Vivado Design Suite 2021.2. All TFGs were synthesized on the FPGA KCU105 board with an operating frequency of 150 MHz. To introduce real FHE, we implemented the HEAAN library, which is prominent in the field of FHE [8]. The HEAAN library implements the Homomorphic Encryption for Arithmetic of Approximate Numbers (HEAANs) scheme, designed for efficient arithmetic operations on approximate numbers. This makes it particularly suitable for applications requiring processing encrypted data, such as secure data analysis and privacy-preserving machine learning. The parameters used are $N = 2^{15}$, $p = 64'\text{d}57646075230827315$ and $W = 64'\text{d}327717960149216710$.

Table 2. Synthesis result of TFGs.

	Conventional Memory	**Proposed 1 Half-Memory**	**Proposed 2 O-Serial**	**Proposed 3 O-Parallel (4)**
#LUT	72 K	36 K	13 K	53 K
#Flip-Flop	-	-	209	830
#DSP	-	24	138	545
Total Area [#Slice]	18 K	9.2 K	4.6 K	18.8 K
Throughput [Gbps]	9.6	19.2	9.6	38.4
FoM [Mbps/#Slice]	0.53	2.08	2.06	2.04
Normalied FoM	1	3.89	3.86	3.83

9.638.4 As shown in Table 1, creating TFGs requires Memory, D FF, Subtractor, Multiplier, and Modulo reduction circuits. When implemented on an FPGA chip, these components are allocated as Look Up Tables (LUTs), Flip Flops (FFs), and Digital Signal Processing (DSPs), with each TFG requiring a different number of these components, as detailed in Table 2. Hardware complexity is expressed by the number of LUTs, FFs, and DSPs. To provide a comprehensive comparison, we converted the necessary components for hardware implementation into FPGA internal SLICEs, labeled as Total Area. Note that each DSP is equivalent to 10 SLICEs in terms of device resources on the KCU105 board. In terms of hardware complexity, the O-serial structure is the most efficient, followed by Half Memory, O-parallel, and Conventional structures. Hardware performance is measured by throughput. Based on an operating frequency of 150 MHz, the throughput for generating

$N = 2^{15}$ TFs with the bit width of 64 is described in Gbps. The Conventional and O-serial structures generate one TF per clock cycle, the Half Memory structure generates two TFs per clock cycle, and the O-parallel structure, adopting a parallelism of 4, generates four TFs simultaneously.

Finally, considering both hardware complexity and performance, the Figure of Merit (FoM) was obtained by dividing the throughput by the area, demonstrating the performance improvement of the proposed TFGs. According to the normalized FoM, the proposed Half-Memory TFG achieves a performance improvement of 3.89 times by reducing memory size by nearly half and doubling the throughput. Next, the proposed O-serial structure significantly reduces hardware complexity by completely eliminating memory, resulting in the lowest area hardware configuration among the proposed structures. The O-serial structure shows a significant improvement in FoM. The proposed O-serial structure shows a significant improvement in FoM, achieving a 3.86-times enhancement compared to the Conventional structure. Lastly, the O-parallel structure, the parallel version of the O-serial structure, shows higher hardware complexity than the proposed Half-Memory TFG and O-serial structure but achieves p-fold increased throughput. Consequently, the O-parallel structure also demonstrates a notable improvement in FoM, amounting to 3.83.

In summary, the three proposed structures have almost similar FoM levels. Depending on the design requirements, it is advisable to choose the most suitable TFG for implementation. The O-serial structure is most suitable for low-area implementations, the O-parallel structure is ideal for high throughput, and the Half-Memory structure is best for achieving the highest FoM. Consequently, this paper offers a solution allowing the selection of the most suitable TFG design based on implementation requirements.

5. Conclusions

In this paper, we propose three efficient TFGs including Half-Memory-based TFG, On-the-fly Serial TFG, and On-the-fly Parallel TFG to enhanced the conventional TFGs. Each of these generators was designed with specific goals in mind. The Half-Memory TFG reduces the memory footprint by storing only half of the twiddle factors and calculating the remaining factors as needed. The On-the-fly Serial TFG computes twiddle factors in real-time during the NTT computation, further minimizing memory requirements. The On-the-fly Parallel TFG leverages parallelism to compute multiple twiddle factors simultaneously, aiming to balance memory usage and computational speed. We implemented these TFGs on the FPGA KCU105 board and conducted comprehensive evaluations to assess their performance. The results demonstrated significant improvements in hardware resource utilization and computational efficiency, compared to traditional approaches. Specifically, the Half-Memory TFG reduced memory usage without compromising performance, the On-the-fly Serial TFG eliminated memory footprint with acceptable computational overhead, and the On-the-fly Parallel TFG offered the best performance in terms of speed, making it suitable for high-throughput applications. These advancements in TFG design have the potential to make FHE more practical for real-world applications, particularly in scenarios where resource constraints are critical. By reducing memory requirements and improving computational efficiency, our proposed TFGs contribute to the broader goal of making FHE a viable option for secure data processing and privacy-preserving computations.

Author Contributions: Conceptualization, H.Y. (Hoyoung Yoo); methodology, N.I. and H.Y. (Hoyoung Yoo); software, N.I., H.Y. (Heehun Yang) and Y.E.; validation, N.I. and H.Y. (Heehun Yang); formal analysis, N.I., H.Y. (Heehun Yang) and H.Y. (Hoyoung Yoo); investigation, N.I., H.Y. (Heehun Yang) and Y.E.; resources, N.I., H.Y. (Heehun Yang) and Y.E.; data curation, N.I., H.Y. (Heehun Yang) and Y.E.; writing—original draft preparation, N.I.; writing—review and editing, S.-C.P. and H.Y. (Hoyoung Yoo); visualization, N.I.; supervision, H.Y. (Hoyoung Yoo); project administration, H.Y. (Hoyoung Yoo); funding acquisition, S.-C.P. and H.Y. (Hoyoung Yoo) All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A5A8026986), and Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. RS-2021-II210779, Development of high-speed encryption data processing technology that guarantees privacy based hardware).

Data Availability Statement: The data presented in this study are available upon request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Marcolla, C.; Sucasas, V.; Manzano, M.; Bassoli, R.; Fitzek, F.; Aaraj, N. Survey on fully homomorphic encryption, theory, and applications. *Proc. IEEE* **2022**, *110*, 1572–1609. [CrossRef]
2. Mareta, R.; Satriawan, A.; Duong, P.N.; Lee, H. A Bootstrapping-Capable Configurable NTT Architecture for Fully Homomorphic Encryption. *IEEE Access* **2024**, *12*, 52911–52921. [CrossRef]
3. Arnold, D.; Saniie, J.; Heifetz, A. *Homomorphic Encryption for Machine Learning and Artificial Intelligence Applications*; Argonne National Lab.: Argonne, IL, USA, 2022. [CrossRef]
4. Rahman, M.S.; Khalil, I.; Atiquzzaman, M.; Yi, X. Towards privacy preserving AI based composition framework in edge networks using fully homomorphic encryption. *Eng. Appl. Artif. Intell.* **2020**, *94*, 103737. [CrossRef]
5. Tan, T.-N.; Lee, H. High-secure low-latency ring-lwe cryptography scheme for biomedical images storing and transmitting. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–4.
6. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory* **2014**, *6*, 1–36. [CrossRef]
7. Brakerski, Z. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology—CRYPTO 2012*, Proceedings of the 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012; Proceedings; Springer: Berlin/Heidelberg, Germany, 2012; pp. 868–886.
8. Cheon, J.-H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology—ASIACRYPT 2017, Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017*; Part I; Springer: Cham, Switzerland, 2017; pp. 409–437.
9. Liang, Z.; Zhao, Y. Number Theoretic Transform and Its Applications in Lattice-based Cryptosystems: A Survey. *arXiv* **2022**, arXiv:2211.13546.
10. Mert, A.C.; Karabulut, E.; Öztürk, E.; Savaş, E.; Aysu, A. An Extensive Study of Flexible Design Methods for the Number Theoretic Transform. *IEEE Trans. Comput.* **2022**, *71*, 2829–2843. [CrossRef]
11. Duong, P.N.; Kwon, S.; Yoo, D.; Lee, H. Area-Efficient Number Theoretic Transform Architecture for Homomorphic Encryption. *IEEE Trans. Circuits Syst. 1 Regul. Pap.* **2022**, *70*, 1270–1283. [CrossRef]
12. Yao, K.; Wang, C.; O'Neill, M.; Liu, W. Towards CRYSTALS-Kyber: A M-LWE Cryptoprocessor with Area-Time Trade-Off. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021; pp. 1–5.
13. Acar, A.; Aksu, H.; Uluagac, A.S.; Conti, M. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.* **2018**, *51*, 1–35. [CrossRef]
14. Pedrouzo-Ulloa, A.; Troncoso-Pastoriza, J.R.; Pérez-González, F. Number Theoretic Transforms for Secure Signal Processing. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 1125–1140. [CrossRef]
15. Ye, T.; Yang, Y.; Kuppannagari, S.R.; Kannan, R.; Prasanna, V.K. Fpga acceleration of number theoretic transform. In *High Performance Computing, Proceedings of the 36th International Conference, ISC High Performance 2021, Virtual Event, 24 June–2 July 2021*; Springer: Cham, Switzerland, 2021; pp. 98–117.
16. Scott, M. A note on the implementation of the number theoretic transform. In *Cryptography and Coding, Proceedings of the 16th IMA International Conference, IMACC 2017, Oxford, UK, 12–14 December 2017*; Springer: Cham, Switzerland, 2017; pp. 247–258.
17. Tan, T.N.; Nguyen, T.T.B.; Lee, H. High Efficiency Ring-LWE Cryptoprocessor Using Shared Arithmetic Components. *Electronics* **2020**, *9*, 1075. [CrossRef]
18. Lee, J.H.; Duong, P.N.; Lee, H. Configurable Encryption and Decryption Architectures for CKKS-Based Homomorphic Encryption. *Sensors* **2023**, *23*, 7389. [CrossRef] [PubMed]
19. Duong, P.N.; Lee, H. Configurable Mixed-Radix Number Theoretic Transform Architecture for Lattice-Based Cryptography. *IEEE Access* **2022**, *10*, 12732–12741. [CrossRef]
20. Longa, P.; Naehrig, M. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *Cryptology and Network Security, Proceedings of the 15th International Conference, CANS 2016, Milan, Italy, 14–16 November 2016*; Proceedings; Springer: Cham, Switzerland, 2016; pp. 124–139.
21. Fritzmann, T.; Sepúlveda, J. Efficient and Flexible Low-Power NTT for Lattice-Based Cryptography. In Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 5–10 May 2019; pp. 141–150.

22. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex fourier series. *Math. Comput.* **1965**, *19*, 297–301. [CrossRef]
23. Choi, S.Y.; Shin, Y.R.; Lim, K.H.; Yoo, H.Y. Efficient Partially-parallel NTT Processor for Lattice_based post-quantum Cryptography. *J. Semicond. Technol. Sci.* **2022**, *22*, 459–474. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.