Hardware Acceleration of the Prime-Factor and Rader NTT for BGV Fully Homomorphic Encryption

David Du Pont
KU Leuven
Leuven, Belgium
david.du.pont@outlook.com

Jonas Bertels
COSIC, KU Leuven
Leuven, Belgium
jonas.bertels@esat.kuleuven.be

Furkan Turan
COSIC, KU Leuven
Leuven, Belgium
furkan.turan@esat.kuleuven.be

Michiel Van Beirendonck
COSIC, KU Leuven
Leuven, Belgium
michiel.vanbeirendonck@esat.kuleuven.be

Ingrid Verbauwhede
COSIC, KU Leuven
Leuven, Belgium
ingrid.verbauwhede@esat.kuleuven.be

***Abstract*—**
Fully Homomorphic Encryption (FHE) enables computation on encrypted data, holding immense potential for enhancing data privacy and security in various applications. Presently, FHE adoption is hindered by slow computation times, caused by data being encrypted into large polynomials. Optimized FHE libraries and hardware acceleration are emerging to tackle this performance bottleneck. Often, these libraries implement the Number Theoretic Transform (NTT) algorithm for efficient polynomial multiplication. Existing implementations mostly focus on the case where the polynomials are defined over a power-of-two cyclotomic ring, allowing to make use of the simpler Cooley-Tukey NTT. However, generalized cyclotomics have several benefits in the BGV FHE scheme, including more SIMD plaintext slots and a simpler bootstrapping algorithm.

We present a hardware architecture for the NTT targeting generalized cyclotomics within the context of the BGV FHE scheme. We explore different non-power-of-two NTT algorithms, including the Prime-Factor, Rader, and Bluestein NTTs. Our most efficient architecture targets the 21845-th cyclotomic polynomial — a practical parameter for BGV — with ideal properties for use with a combination of the Prime-Factor and Rader algorithms. The design achieves high throughput with optimized resource utilization, by leveraging parallel processing, pipelining, and reusing processing elements. Compared to Wu et al.'s VLSI architecture of the Bluestein NTT, our approach showcases 2× to 5× improved throughput and area efficiency. Simulation and implementation results on an AMD Alveo U250 FPGA demonstrate the feasibility of the proposed hardware design for FHE.

***Index Terms*—Fully Homomorphic Encryption, Brakerski-Gentry-Vaikuntanathan, Hardware Accelerator, Number Theoretic Transform, Rader's FFT, Prime-Factor FFT, Bluestein's FFT**

I. Introduction

In an era where data privacy and security have become important concerns, innovative cryptographic solutions are emerging to safeguard sensitive information while enabling efficient computations in untrusted environments. Among these advancements, Fully Homomorphic Encryption (FHE) stands out as a transformative technology that holds great promise across various sectors, including finance, web services, and beyond. FHE revolutionizes data processing by allowing computations to be performed directly on encrypted data, eliminating the risk of unauthorized access.

In 2009, Gentry introduced the first feasible construct for Fully Homomorphic Encryption, opening the door to a new realm of possibilities in secure computing [1]. Gentry's scheme allows both addition and multiplication operations, providing a foundation for constructing circuits that can perform arbitrary computations. Gentry started from a Somewhat Homomorphic Encryption scheme and modified it so it could evaluate its own decryption function and at least one more operation. A scheme like this is said to be bootstrappable. This bootstrapping operation of evaluating the decryption function homomorphically is very slow even on modern hardware [2].

FHE schemes generally encrypt plaintext data into ciphertext polynomials of a certain large degree. Smart and Vercauteren [3] observed that multiple plaintext values of a smaller ring can be packed into a single ciphertext. A single operation on this ciphertext then corresponds to performing this operation on each of the plaintext values individually in a SIMD fashion. This innovation leads to more efficient cryptosystems such as the BGV scheme [4]. The operations in BGV are defined over a cyclotomic polynomial ring, and polynomial multiplication becomes the main performance bottleneck. In practice, power-of-two cyclotomic rings have been the preferred implementation choice due to their simplicity. However, generalized non-power-of-two cyclotomics typically have more efficient packing in the plaintext slots.

Halevi and Shoup [2] proposed an improved bootstrapping method for the BGV-FHE used in the open-source software library HElib [5]. Their method relies on having a polynomial ring of degree m, where m has a special form. Practical values of m are not powers of two, but rather composed of a few large prime factors. As such, HElib is the only FHE software library to include generalized cyclotomics rings. To accelerate polynomial multiplication, HElib uses a different representation called DoubleCRT form. In DoubleCRT form, polynomial multiplication becomes a simple pointwise multiplication of vector elements. To convert between the coefficient representation of a polynomial and DoubleCRT form, a Number Theoretic Transform (NTT) is used. To compute non-power-of-two NTTs, HElib employs Bluestein's algorithm [6]. From benchmarking HElib we find that 52%-53% of the computation time is spent on the Bluestein algorithm for common operations such as ciphertext multiplication and bootstrapping.

Implementations of BGV offer a trade-off in the choice of cyclotomic rings. Power-of-two cyclotomics are simpler and more implementation-friendly, and they can make use of the standard Cooley-Tukey NTT. In contrast, non-power-of-two cyclotomics can offer more efficient plaintext packing and bootstrapping, but they are more difficult to implement in both software and hardware implementations. Especially in hardware implementations, non-power-of-two cyclotomics are underexplored in the literature. Many architectures have been explored to accelerate the NTT operation for power-of-two values of m [7]–[11], but only a single prior implementation explores the non-power-of-two Bluestein NTT [12].

In this paper, we implement a hardware architecture for the NTT targeting generalized cyclotomic rings in HElib's BGV-FHE implementation. We explore different non-power-of-two algorithm trade-offs, including the Prime-Factor, Rader, and Bluestein NTTs. Similar to the analysis of Hsu and Shieh [13], we highlight that the 21845-th cyclotomic polynomial has ideal properties for an implementation using a combination of the Prime-Factor and Rader NTT. Using the combination of these two algorithms significantly reduces the arithmetic cost of the NTT compared to a Bluestein implementation.

We are the first to implement and evaluate in hardware the Prime-Factor and Rader NTTs for this choice of cyclotomic. Different circuit optimizations are used to achieve high performance; mainly parallel processing to reduce the latency, and pipelining of the datapath to achieve a higher clock speed. An efficient memory design is used to keep up with the high throughput. Additionally, we focus on maximal reuse of functional units throughout the different computation stages to keep the area or resource utilization limited on an Alveo U250 FPGA. Finally, we compare our implementation to Wu et al.'s [12] implementation of Bluestein's FFT, showing increased throughput and area efficiency. The RTL implementation of our work is publicly available at:

https://github.com/KULeuven-COSIC/NonPowerOfTwoNTT.

II. Preliminaries

We use $\mathbb{Z}_q$ to denote the ring of integers with q elements, also known as the integers modulo q. In case q is a prime number, $\mathbb{Z}_q$ forms a finite field. Elements in a ring are denoted by lowercase letters. Vectors are expressed in bold $\mathbf{x} = \langle x_1, x_2, x_3, \ldots, x_{n-1}, x_n \rangle$. The i-th component of vector $\mathbf{x}$ is specified as x_i. We use $*$ to represent the convolution operation between two vectors, whereas $\odot$ represents element-wise multiplication.

A. The BGV Scheme

The BGV FHE scheme [4] operates on polynomials defined over the cyclotomic ring: $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$, where $\Phi_m(X)$ is the cyclotomic polynomial of order m. The plaintext space of the BGV scheme consists of vectors of integers modulo a plaintext modulus t. The ciphertext space of the BGV scheme consists of vectors of polynomials modulo a ciphertext modulus Q, which is a large integer that determines the security level and the noise budget.

An important feature of the BGV scheme is the concept of modulus switching. A freshly encrypted ciphertext starts at encryption level L, and the encryption level gradually moves down as we perform homomorphic computations. With each encryption level l, a modulus q_l is associated, and the ciphertext modulus $Q = q_L$. Modulus switching helps to reduce the noise growth because when we switch the modulus from q_l to q_{l-1} with $q_{l-1} > q_l$, the noise term of the ciphertext is reduced by the ratio $\frac{q_l}{q_{l-1}}$. When we reach the smallest modulus q_0 at encryption level 0, the noise can no longer be reduced. At this point, the ciphertext needs to either be decrypted or Gentry's bootstrapping method [1] needs to be employed to enable further computation.

In the implementation of the BGV scheme within HElib, a series of small machine-word sized prime numbers $p_0, p_1, p_2, \ldots, p_L$ are used to construct moduli as follows: [2]

$$q_l = \prod_{i=0}^{i=l} p_i \tag{1}$$

The BGV scheme involves various operations with integer polynomials, such as modular multiplications, additions, and Frobenius maps. The polynomials used in these operations are represented using the DoubleCRT format to increase efficiency. This format represents a polynomial as a collection of polynomials, each computed modulo a small prime p_l, with each individual polynomial further represented in the evaluation form. In this evaluation form, a polynomial is described as a vector containing its values at primitive N-th roots of unity within the finite field $\mathbb{Z}_{p_l}$. Within this representation, polynomial multiplication simplifies to point-wise multiplication that can be performed in linear time. The transformation between coefficient representation, where the polynomial is expressed as a list of its coefficients, and evaluation representation involves using the Number Theoretic Transform (NTT). The NTT happens to be the most time-consuming step for calculations in HElib, therefore, it is the primary focus of hardware acceleration.

B. Number Theoretic Transform

The Number Theoretic Transform (NTT) generalizes the discrete Fourier transform (DFT) over $\mathbb{Z}_p$ where p is prime.

It provides efficient algorithms for size-N polynomial multiplication in time proportional to $N\log(N)$. In this work, we are primarily interested in the case where N is not a power of two. Rather, $N = m$, with the cyclotomic parameter m a product of distinct primes.

[14] $\mathbb{Z}_p$ is a finite field, so there exists a primitive N-th root of unity if N divides $p-1$. Let ω be a primitive N-th root of unity. Then, the N-point NTT,

$$\mathbf{X} = \langle X_0, \ldots, X_{N-1}\rangle = \mathcal{N}\left\{\mathbf{x}\right\}, \tag{2}$$

is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n \omega^{nk}, \qquad k = 0, \ldots, N-1 \tag{3}$$

The inverse NTT,

$$\mathbf{x} = \langle x_0, \ldots, x_{N-1}\rangle = \mathcal{N}^{-1}\left(\mathbf{X}\right), \tag{4}$$

is defined as:

$$x_n = N^{-1}\sum_{k=0}^{N-1} X_k \omega^{-nk}, \qquad k = 0, \ldots, N-1. \tag{5}$$

Since ω is an N-th root of unity we have $\omega^{-nk} = \omega^{(N-k)n}$. This means an inverse NTT can be computed using a forward NTT by reversing the order of the elements in $\mathbf{X}$ as follows:

$$\langle X_0, X_{N-1}, X_{N-2}, \ldots, X_2, X_1\rangle \tag{6}$$

Because of the similarity between the NTT and DFT, any FFT algorithm can also be used for calculating the NTT. The only modification that needs to be made is that $e^{-\frac{j2\pi}{N}}$ is replaced by ω. [15]

When N is a power of two, the NTT is typically computed using (variations of) the radix-2 Cooley-Tukey algorithm. This algorithm re-expresses a size-N NTT as two NTTs of sizes $N/2$. When N contains large or distinct prime factors, other algorithms can be exploited for greater efficiency.

C. *Prime-Factor FFT Algorithm*

The Prime-Factor FFT algorithm (PFA) transforms a discrete Fourier transformation (DFT) with a size of $N = N_1N_2$ into a two-dimensional DFT with dimensions $N_1 \times N_2$. [16] It is important to note that the PFA requires that N_1 and N_2 are integers that are relatively prime. By applying PFA recursively, the smaller DFTs of size N_1 and N_2 can be computed. For example, $N = m = 21845$ has factorization $257 \times 17 \times 5$ and will be computed as a three-dimensional DFT with dimensions $257 \times 17 \times 5$. The factorization is similar to Cooley-Tukey but has the advantage that no multiplications with twiddle factors are required. As a trade-off, its use is limited to coprime factorizations, and a more complex re-indexing based on the Chinese Remainder Theorem (CRT) is required.

D. *Rader's FFT Algorithm*

Whereas the Prime-Factor algorithms can break up an NTT into its prime factors, Rader's algorithm permits efficiently computing the N-point NTT when N is prime. In our example above, Rader's algorithm can be used to more efficiently implement the size-257, size-17, and size-5 prime NTTs that result from PFA decomposition of the size-21845 NTT.

[17] In $\mathbb{Z}_N$, there exists a primitive root $g \in \mathbb{Z}_N$ such that for any non-zero element $n \in \mathbb{Z}_N$, there exists a unique exponent $q \in \{0, 1, 2, \ldots, N-2\}$ satisfying the equation: $n = g^q \pmod N$. Since every non-zero element in $\mathbb{Z}_N$ corresponds to a unique q this forms a bijection from q to non-zero n. In the same way $k = g^{-p} \pmod N$, with k a non-zero element in $\mathbb{Z}_N$ and $p \in \{0, 1, 2, \ldots, N-2\}$ forms a bijection from p to non-zero k. By using this re-indexing the size-N NTT can be expressed as the convolution of two sequences $\mathbf{a}$ and $\mathbf{b}$ of sizes $N-1$ as follows:

$$a_q = x_{g^q} \tag{7}$$

$$b_q = \omega^{g^{-q}} \tag{8}$$

$$X_{g^{-p}} = x_0 + \sum_{q=0}^{N-2} a_q b_{p-q} \qquad p = 0, \ldots, N-2 \tag{9}$$

In turn, the convolution of sequences $\mathbf{a}$ and $\mathbf{b}$ is computed by using the convolution theorem (Equation 10), requiring NTTs of size $N-1$:

$$\mathbf{A} = \mathcal{N}(\mathbf{a}) \tag{10}$$

$$\mathbf{B} = \mathcal{N}(\mathbf{b}) \tag{11}$$

$$\mathbf{a} * \mathbf{b} = \mathcal{N}^{-1}\left\{\mathbf{A} \odot \mathbf{B}\right\}. \tag{12}$$

E. *Bluestein's FFT Algorithm*

Bluestein's algorithm permits to efficiently compute the DFT of any size, including prime sizes. [6] We show below how this algorithm can be used to compute the NTT.

Begin with the definition of the NTT

$$X_k = \sum_{n=0}^{N-1} x_n \omega^{nk} \qquad k = 0, \ldots, N-1 \tag{13}$$

If we replace nk in the exponent by

$$nk = \frac{k^2}{2} + \frac{n^2}{2} - \frac{(k-n)^2}{2}, \tag{14}$$

we obtain:

$$X_k = \omega^{\frac{k^2}{2}} \sum_{n=0}^{N-1} \left(x_n \omega^{\frac{n^2}{2}}\right) \omega^{-\frac{(k-n)^2}{2}} \qquad k = 0, \ldots, N-1 \tag{15}$$

This summation can be expressed as the convolution of two sequences a_n and b_n as follows:

$$
\begin{aligned}
a_n &= x_n \omega^{\frac{n^2}{2}} \\
b_n &= x_n \omega^{-\frac{n^2}{2}} \\
X_k &= b_k^{-1} \left(\sum_{n=0}^{N-1} a_n b_{k-n} \right) \qquad k = 0, \ldots, N-1
\end{aligned}
\tag{16}
$$

Just like in Rader's algorithm, the convolution is computed by a pair of NTTs using the convolution theorem (10).

The advantage here is that these NTTs do not have to be of length N. The convolution can be computed after zero-padding it to a length of at least $2N-1$. By padding it to a power of two, the NTT can be computed efficiently using the Cooley-Tukey algorithm in $O(M \log_2 M)$, where M is the padded length.

III. Algorithmic Exploration

In this section, we explore different choices for the non-power-of-two NTT for practical parameter sets of BGV-FHE. We repeat the analysis of [13], highlighting that the combination of the Prime-Factor and Rader algorithms can often outperform Bluestein implementations. In their work, Hsu and Shieh [13] show that especially the 21845-th cyclotomic polynomial presents an optimal parameter choice for the non-power-of-two NTT in BGV-FHE. We also show that this parameter set features the smallest number of multiplications, and we proceed with the first hardware implementation for this parameter set in the rest of the paper.

A. Comparing PFA, Rader, and Bluestein

In this work, we are specifically interested in evaluating alternatives to a Bluestein NTT implementation. For an arbitrary N-point sequence, three different scenarios can be defined:

- When N can be factored into coprime numbers N_1 and N_2, the problem of computing an N-point NTT can be reformulated using PFA into computing N_2 instances of an N_1-point NTT and N_1 instances of an N_2-point NTT.
- When N is not a prime number but a prime power, represented as p^k, the Cooley-Tukey FFT algorithm can be used.
- When N is a prime number, the problem of computing the N-point NTT can be transformed using Rader's algorithm into computing two $(N-1)$-point NTTs.

Any of these algorithms will reduce the sizes of the NTTs to be performed. An efficient approach to performing NTTs of any size can be found by recursively selecting the appropriate algorithm for each new size encountered. [18]

Figure 1 shows how this method compares to both the direct application of Bluestein's algorithm and the application of PFA followed by Bluestein for different bootstrappable parameters of the BGV-FHE scheme from Table I. To simplify the analysis, for this comparison, it is assumed that each power-of-two length NTT requires exactly $N \log N$ multiplications. Recursive application of PFA and Rader

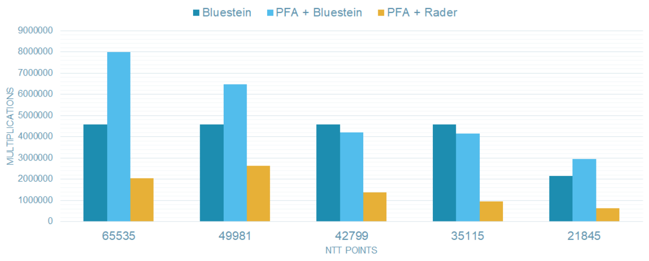

Fig. 1. Number of multiplications required to perform different NTT algorithms.

requires significantly fewer multiplications than using Bluestein, both directly or after PFA.

In the context of precomputing twiddle factors and precomputing the NTT of the sequence b_n in Bluestein's algorithm or b_q in Rader's algorithm, another important factor to consider is the size of the required lookup table. If Bluestein's algorithm is applied directly, the length of the NTT is at least doubled, and the number of twiddle factors and B_n values is also doubled. A single application of PFA reduces the number of twiddle factors and B_q values from N to $N_1 + N_2$.

Another thing to consider is the restriction on the moduli that can be used in computing the NTT. A primitive N-th root of unity exists in the ring $\mathbb{Z}_p$ when p is of the form $p = kN + 1$. Since different NTT lengths are used in the algorithm, a prime p is required that satisfies this property for each length. Table II shows the possible moduli for different bootstrappable parameters for both the PFA-Rader algorithm and Bluestein's algorithm. The table also includes the minimum number of bits needed to represent the first twenty possible values for p. Bluestein's algorithm generally requires longer word sizes.

Because of these advantages, the hardware architecture will be based on the PFA-Rader algorithm rather than Bluestein. The choice $m = 21845$ stands out as a good parameter for this algorithm, both in the number of slots and implementation metrics. Therefore, the hardware design and the remainder of the paper focuses on this parameter.

B. An Optimization to Rader's Algorithm

Rader's algorithm requires $m-1$ additions to compute $X_0 = \sum_{i=0}^{m-1} x_i$ and $m-1$ additions to add x_0 to each element of the convolution result. This number can be reduced to just two additions. We observe that

$$
A_0 = \sum_{q=0}^{N-2} x_{g^q} = \sum_{i=1}^{N-1} x_i, \tag{17}
$$

so X_0 can be computed as $X_0 = x_0 + A_0$. If $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$, then adding x_0 to C_0 before the inverse NTT corresponds to adding x_0 to each element of $\mathbf{c}$. Algorithm 1 describes our implementation of Rader's algorithm with these modifications. Note that we exclude the final reversed re-indexing step.

TABLE I
BOOTSTRAPPABLE VALUES FOR PARAMETER m. GENERATED USING `FIND_BOOTSTRAPPABLE_PARAMS.M` OF [19].

cyclotomic ring m	65535	49981	42799	35113	21845
m's factorization	$257 \cdot 17 \cdot 5 \cdot 3$	$331 \cdot 151$	$337 \cdot 127$	$73 \cdot 37 \cdot 13$	$257 \cdot 17 \cdot 5$
number of slots	2048	1650	2016	864	1024

TABLE II
POSSIBLE MODULI FOR PFA-RADER FFT AND BLUESTEIN FFT

	PFA-Rader		Bluestein	
m	p	n-bits	p	n-bits
65 535	$16\,776\,960k + 1$	31	$8\,589\,803\,520k + 1$	40
49 981	$49\,481\,190k + 1$	33	$6\,551\,109\,632k + 1$	41
42 799	$43\,141\,392k + 1$	32	$5\,609\,750\,528k + 1$	41
35 115	$82\,169\,100k + 1$	34	$4\,602\,593\,280k + 1$	39
21 845	$5\,592\,320k + 1$	30	$1\,431\,633\,920k + 1$	38

Algorithm 1 Optimized Rader's Algorithm

```
RADER($\mathbf{x}$)
    $\mathbf{x}' \leftarrow$ RaderPermutation($[x_1, x_2, \ldots, x_{m-1}]$)
    $\mathbf{A} \leftarrow \mathcal{N}(\mathbf{x}')$
    $X_0 \leftarrow A_0 + x_0$
    $\mathbf{C} \leftarrow \mathbf{B} \odot \mathbf{A}$
    $C_0 \leftarrow C_0 + x_0$
    $[X_1, X_2, \ldots, X_{m-1}] \leftarrow \mathcal{N}^{-1}(\mathbf{C})$
    return $\mathbf{X}$
```

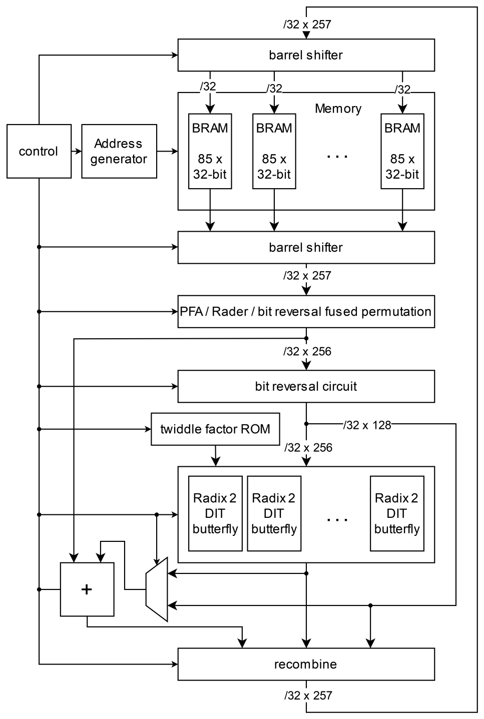

Fig. 2. Proposed PFA+Rader NTT hardware architecture

IV. HARDWARE ARCHITECTURE

In the remainder, we describe a high-throughput hardware NTT implementation that is optimized for the 21845-th cyclotomic polynomial. As mentioned before, our implementation uses a combination of the Prime-Factor, Rader, and Cooley-Tukey algorithms, decomposing $21845 = 257 \times 17 \times 5$. Since FHE applications typically use a single parameter set, there is no need to support multiple choices of m within a single architecture. Therefore, our implementation focuses on the specific optimized choice $m = 21845$, but the proposed methodology is general and can be extended to produce optimized circuits for other choices for m.

A. Architecture Overview

Figure 2 shows an overview of the hardware architecture. The architecture exploits data parallelism by performing operations on large vectors. The data is streamed through a fully pipelined datapath with twenty-seven pipeline stages. The processor is fully pipelined, and the design achieves a clock frequency of 250 MHz after FPGA place & route. There are no wait states, such that all processing elements achieve optimal utilization.

Vectors are composed of 257 32-bit words, where 257 corresponds to the largest dimension in the resulting 257×85 matrix obtained through PFA. Each vector can represent either one row of the matrix or three columns. As the vectors pass through the datapath, either one recursive stage of the Cooley-Tukey FFT algorithm or 128 pointwise multiplications can be computed. Each functional unit in the architecture is designed to handle a throughput of one vector per cycle. The inner NTT unit consists of 128 radix-2 butterfly units. These units can be configured to perform either a butterfly operation or only a multiplication and modular reduction.

The overall architecture can work with up to 46 different moduli of 32 bits, this is necessary for modulus switching in the BGV scheme. Only forward NTTs are supported, but an inverse NTT can be computed by performing a re-indexing in software that puts the elements in the order as shown in Equation 6.

To achieve the required memory bandwidth, we use 257 individually addressable memory banks. The rows of the matrix are stored in a staggered manner, ensuring that each element of a row or column resides in a different memory bank. An address generator determines the read and write addresses, while barrel shifters eliminate and reapply the offset caused by row staggering.

In the following sections, we will describe the control flow, memory, permutation hardware, and inner Cooley-Tukey NTT implementation in more detail.

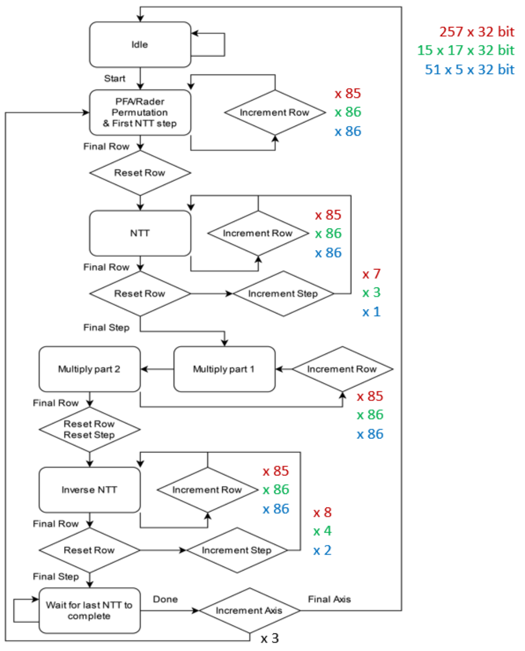

Fig. 3. Control state diagram of the PFA+Rader algorithm executions.

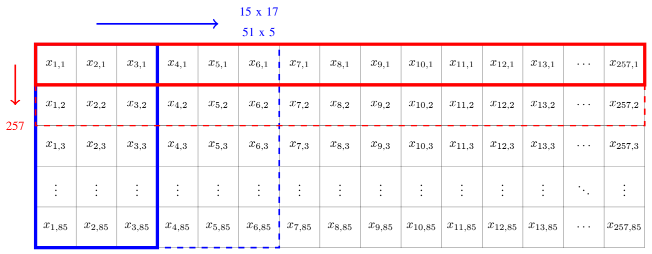
15 x 17
51 x 5
257
$x_{1,1}$ $x_{2,1}$ $x_{3,1}$ $x_{4,1}$ $x_{5,1}$ $x_{6,1}$ $x_{7,1}$ $x_{8,1}$ $x_{9,1}$ $x_{10,1}$ $x_{11,1}$ $x_{12,1}$ $x_{13,1}$ $\cdots$ $x_{257,1}$
$x_{1,2}$ $x_{2,2}$ $x_{3,2}$ $x_{4,2}$ $x_{5,2}$ $x_{6,2}$ $x_{7,2}$ $x_{8,2}$ $x_{9,2}$ $x_{10,2}$ $x_{11,2}$ $x_{12,2}$ $x_{13,2}$ $\cdots$ $x_{257,2}$
$x_{1,3}$ $x_{2,3}$ $x_{3,3}$ $x_{4,3}$ $x_{5,3}$ $x_{6,3}$ $x_{7,3}$ $x_{8,3}$ $x_{9,3}$ $x_{10,3}$ $x_{11,3}$ $x_{12,3}$ $x_{13,3}$ $\cdots$ $x_{257,3}$
$x_{1,85}$ $x_{2,85}$ $x_{3,85}$ $x_{4,85}$ $x_{5,85}$ $x_{6,85}$ $x_{7,85}$ $x_{8,85}$ $x_{9,85}$ $x_{10,85}$ $x_{11,85}$ $x_{12,85}$ $x_{13,85}$ $\cdots$ $x_{257,85}$

Fig. 4. Illustration of the matrix layout and iterations during PFA+Rader algorithm executions.

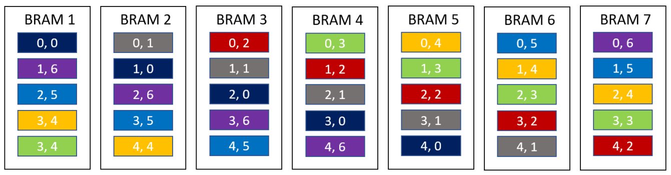

Fig. 5. Illustration of staggered rows in memory, each color represents a column in the PFA matrix.

B. Control Flow

Figure 3 illustrates the state diagram of the controller, which governs the processing of a three-dimensional grid with dimensions $257 \times 17 \times 5$ obtained through PFA. The outer loop in the controller contains seven distinct states, including the initial idle state. Additionally, three state variables are present: *axis*, *row*, and *step*. *axis* indicates the traversal direction within the grid, *row* indicates the currently processed row index, and *step* corresponds to the Cooley-Tukey algorithm stage. The output of the controller depends on both the current state and the current axis.

Each stage of the NTT computation is performed on all rows along one axis before moving on to the next stage. Iterating over the NTT stages in the outer loop and over the rows in the inner loop eliminates data dependencies that would cause pipeline stalls. The iteration order is different for the Multiply part 1 and Multiply part 2 states, which encompass the point-wise multiplication in Rader's Algorithm 1. Iterating over the rows in the outer loop is more efficient for these operations since there are no data dependencies. Moreover, some of the data required for part 2 would be overwritten by the result of part 1, this is avoided by performing both operations for the same row in subsequent cycles.

The pipeline stalls in the final state, where we wait for all NTTs along the current axis to complete. This stalling is necessary when transitioning from rows to columns because there is a data dependency between each row and column.

Figure 4 details the iteration through the matrix. The three-dimensional $257 \times 17 \times 5$ matrix is stored in memory as a two-dimensional 257×85 matrix, where each column represents a flattened 17×5 matrix. The highlighted blue and red regions indicate the rows and columns processed in one iteration step, respectively. First, we iterate over every row and subsequently over every column. Three columns are processed simultaneously in each step, except in the last iteration step, where only two columns are processed. The dashed regions indicate the sections of the matrix that will be processed in the next iteration step. It is important to note that columns undergo two separate iterations: in the first iteration, the rows of the flattened 17×5 matrices are processed, and in the second iteration, the columns of these matrices are processed.

C. Memory Design

The PFA FFT re-expresses the NTT of size $m = 21845$ as a two-dimensional NTT over a 257×85 matrix. This matrix is stored in 257 Block RAMs (BRAM) on the FPGA so that a vector of 257 32-bit words can be read from or written to the memory in every cycle.

Accessing matrix elements in parallel poses a memory challenge. Specifically, storing columns in separate BRAMs enables parallel row access but not parallel column access, while storing rows in separate BRAMs enables parallel column access but not parallel row access. To address this, a staggered memory arrangement is used. These permuted arrangements have been used before for power-of-two NTT implementations in FHE [20], [21], but, to the best of our knowledge, never for the PFA NTT.

Rows are stored offset by one BRAM from each other, enabling parallel access to both rows and columns. Figure 5 gives a visual example using a 7×5 matrix. Each color corresponds to a distinct matrix column, highlighting separate BRAM storage for both rows and column elements.
To facilitate reading from and writing to memory with the required offsets, two circular shift circuits are needed: one at the write ports and another at the read ports of the memory. These circuits enable circularly shifting 32-bit words across

lanes in a vector of length 257. A barrel shifter is used to achieve a throughput of one vector per cycle.

D. Combining Permutations

The NTTs of size 85 over the columns are re-expressed as a two-dimensional NTT over a 17×5 matrix. This translates to two distinct permutations in our hardware: one that groups the elements of every matrix row together and one that groups the columns of the matrix.

The NTTs of size 257, 17 and 5 are computed using Rader's algorithm. In Rader's algorithm, the first element of each row (with sizes 257, 17, and 5) in the vector needs to be removed. A permutation removes the first points from the NTT vectors and fills the gaps with subsequent elements. The removed first points are collected and placed at the end of the vector. An additional permutation is required for the re-indexing in Rader's algorithm. This permutation is different for each NTT size (257, 17, and 5). Finally, a bit-reversal permutation is necessary because the decimation in time Cooley-Tukey algorithm expects the initial input in a bit-reversed order.

In summary, there are many permutations involved within the combination of the PFA and Rader algorithms. In our architecture, we propose to combine permutations to limit their limit number. This optimization reduces the total number of permutations to only six. To select one permutation from the reduced set of six, a 6-to-1 multiplexer is used.

E. Vectorized Cooley-Tukey FFT

The NTTs of sizes $N-1$ within Rader's algorithm (256, 16, and 4), are computed using the Cooley-Tukey FFT algorithm. This inner Cooley-Tukey FFT algorithm, specifically the radix-2 decimation-in-time (DIT) variant, works by breaking down a Discrete Fourier Transform (DFT) of a larger size $2n$ into two smaller transforms of size n. These smaller transforms are then combined using mathematical operations called "butterflies," which involve computing smaller DFTs of size 2 that are multiplied with roots of unity known as "twiddle factors". Figure 6 shows a butterfly diagram for an 8-point FFT algorithm, which illustrates how the inputs and outputs are connected for each stage during the computation. The same hardware used for computing an FFT of size $2n$ can also be used for computing two FFTs of size n by omitting the final stage of the FFT and adjusting the twiddle factors. This property allows us to reuse the hardware of a 256-point FFT, for computing multiple 16-point and 4-point FFTs in parallel.

The Cooley-Tukey NTT requires the input to be bit-reversed. The bit-reversal circuit is simple. To achieve a throughput of one vector per cycle, it is implemented as a series of eight bit-reversal permutations of increasing lengths: $\{2, 4, 8, 16, 32, 64, 128, 256\}$. Multiplexers are used to select either the permuted vector or the non-permuted vector.

F. Butterfly Units

We used 128 butterfly units to achieve a throughput of one vector per cycle. The design of a butterfly unit is depicted in Figure 7.

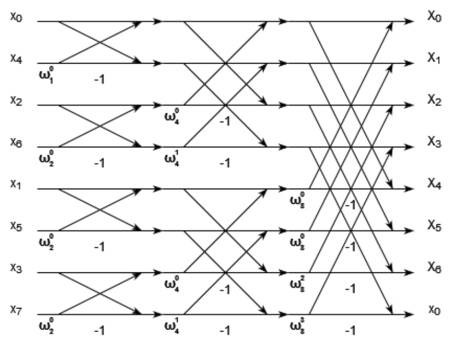

Fig. 6. Butterfly diagram for radix-2 DIT FFT algorithm

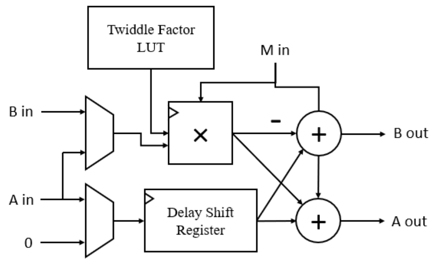

Fig. 7. Implementation of radix-2 DIT FFT butterfly

The butterfly unit makes use of a word-level Montgomery modular multiplier implementation from Mert et al. [9] The multiplier is fully pipelined and has a latency of nineteen cycles. A shift register is used to match the latency of the modular multiplier for the other input. Twiddle factors are precomputed and stored in a lookup table.

The convolution in Rader's algorithm requires a pointwise product to be computed. We repurpose existing butterfly unit multipliers for this task to optimize resource utilization. One of the factors in each pointwise product is a precomputed constant, conveniently stored within the FFT twiddle factor lookup table. To perform multiplication without the final addition, we add a multiplexer at input A that allows a zero input selection. Another multiplexer is added at input B to enable multiplication of either input A or input B with a constant factor.

V. Evaluation and Results

A. Latency Simulation Results

We measure the latency of architecture using waveform simulations. Simulation involves writing testvector data into the BRAMs from a file, followed by the full NTT computation. Testvectors are generated using HElib and checked for correctness. Simulation shows that a 21845-point NTT can

Frequency (MHz)	CLB LUT	CLB Register	BRAM	DSP
250	224074	169900	300	1024

TABLE III
Alveo U250 Implementation results

TABLE IV
COMPARISON TO OTHER NTT IMPLEMENTATIONS. BOTH NON-POWER-OF-TWO (LEFT) AND POWER-OF-TWO NTTS (RIGHT) ARE INCLUDED.

Design	**This work**	[12]	[7]	[8]	[9]	[10]	[11]
m	**21845**	8193/4369	32768	8192	4096	512	1024
Coefficient size (bits)	**32**	25	32	54	60	13	16
Platform	**Alveo U250**	Virtex-7	Virtex-7	Stratix 10 GX 2800	Virtex-7	Virtex 6	Artix-7
Frequency (MHz)	**250**	250	250	300	125	278	45.47
Cycles	**2987**	5825	12725	768	972	2304	18537
CLB LUTs	**224k**	76.2k	219k	142k	99.3k	1536	2908
CLB Registers	**170k**	-	90.7k	387k	-	953	170
BRAM	**429**	62	193	725	176	3	0
DSPs	**1024**	256	768	320	929	1	9
$\tilde{l}$ (µs)	**0.44**	2.9/5.8	1.0	0.14	0.86	45	815
ADP: $\tilde{l} \times$ LUTs	**87k**	219k/440k (**2.5×/5.1×**)	232k	21k	86k	70k	2.4M
ADP: $\tilde{l} \times$ DSPs	**398**	734/1480 (**1.8×/3.7×**)	814	47	803	45	7338

be computed in 2987 clock cycles. This number is consistent regardless of the data or modulus used. Of these total clock cycles, 2392 clock cycles are spent in computing the inner FFTs using Cooley-Tukey's algorithm, 514 clock cycles in pointwise multiplication for Rader's algorithm, and 78 cycles are pipeline stalls, which corresponds to only 2.6% of the total execution time.

B. Implementation Results

Table III shows the implementation area and frequency results. These results include the entire NTT but without BRAM interface. Of the used BRAMs, 257 are used as RAM, and 171 BRAMs are used as ROM to store the twiddle factors. The Alveo U250 FPGA contains BRAM tiles, which can implement either one large BRAM or two smaller BRAMs. The ROM BRAMs use one tile each, the RAM BRAMs use 128.5 tiles. Therefore, utilization reports show 300 BRAMs used.

A major challenge in the efficient place & route of our design using default settings in Vivado is SLL (Super Long Line) congestion. SLLs connect signals between die slices that are called Super Logic Regions (SLR) on the FPGA. There are only a limited number of these SLL connections available, and the router has difficulty with timing closure when a high portion of these SLLs is used. To address this problem, we eliminated using SLLs entirely by confining all the logic to one SLR. This single SLR placement solves congestion issues and allows our design to achieve 250 MHz clock frequency after place & route.

At a frequency of 250 MHz, a 21845-point NTT can be computed in 11.9 microseconds, achieving a speed that is ninety times faster than when the same NTT computation is performed on a conventional CPU platform using HElib. Conducting the same 21845-point NTT computation on an Intel Core i7 processor, with 12 threads at 4 GHz, results in a computation time of 1.10 milliseconds.

C. Comparison to Other Implementations

Few efforts have been made in hardware acceleration of NTTs with non-power-of-two lengths. Hsu and Shieh [13] theoretically analyze the use of the Prime-factor and Rader algorithms but do not develop an actual hardware implementation. To the best of our knowledge, Wu et al.'s VLSI implementation of the Bluestein algorithm [12] is the only non-power-of-two implementation of an NTT. Their implementation makes use of a 64-bit modulus, which is significantly larger than the 22 to 25-bit coefficient size, in order to avoid modular reductions during computation. We will focus mainly on Wu et al.'s implementation, since the other implementations are specifically tailored for power-of-two length NTTs [7]–[11]. These power-of-two implementations are included in our discussion solely for reference, and should not be considered suitable for our target application of BGV with non-power-of-two cyclotomics.

The implementation of Wu et al. targets BGV with $m = 8193$ or $m = 4369$. We note that, in contrast to our implementation, these smaller values for m are not bootstrappable (Table I). They are only suitable for levelled FHE applications, limiting their usability significantly.

To facilitate performance comparison across different NTT sizes, Table IV presents the latency for one forward NTT computation normalized to $m = 1024$ and $k = 32$, where k is the coefficient size in bits. We assume quasilinear scaling with m due to the FFT operation's time complexity. Linear performance scaling is assumed for k as lowering the small moduli's bit-width in the BGV scheme roughly corresponds to a proportional increase in the number of vectors in DoubleCRT representation. The normalized latency $\tilde{l}$ is calculated as

$$\tilde{l} = \text{latency} \cdot \frac{\text{Reference NTT complexity}}{\text{Other design complexity}} \quad (18)$$

$$= \text{latency} \cdot \frac{32 \cdot 1024 \log_2 1024}{k \cdot m \log_2 m} \quad (19)$$

Table IV also includes Area-Delay Product (ADP) of the normalized latency with the number of LUTs and DSPs. For this ADP figure of merit combining area and performance, lower values indicate a better performance. It's important to note that this performance estimation does not consider other potential limitations.

In our implementation, we achieve a performance in the ADP figure of merit, $\tilde{l} \times$ LUTs/DSPs, that is more than double when compared to the implementation by Wu et al., particularly when evaluating a 8193-point NTT. This scenario represents the optimal case for Wu et al.'s implementation, as this specific parameter selection eliminates the need for padding within Bluestein's algorithm. However, the practical use case for BGV referenced by Wu et al. uses the parameter choice $m = 4369$. In this more realistic scenario, our design shows a fivefold increase in performance. For completeness, we note that our design does not incorporate hardware-based reversed re-indexing, resulting in a scrambled NTT result. This reversed re-indexing step can be performed in linear time and is not necessary for element-wise operations, but is required when we convert back from DoubleCRT to polynomial coefficient representation.

For fairness, we also note that our implementation targeting the Alveo U250 is a more advanced process node (16nm) than the Virtex-7 (28nm) that is popular in prior work. For FHE applications, the Alveo U250 is a more interesting target, as it is readily available in cloud platforms — such as Amazon's AWS — as a datacenter accelerator card. Whereas LUT/DSP utilisation is typically nearly identical between these two devices, we estimate that prior work would run at slightly higher frequencies on the Alveo U250.

VI. Conclusion

In this work, we presented a hardware architecture for efficient non-power-of-two Number Theoretic Transform (NTT) computations, targeting fully homomorphic encryption using the BGV scheme. Our design focuses on the 21845-th cyclotomic polynomial, which is a practical parameter for BGV. The non-power-of-two NTT architecture employs a combination of the Prime-factor and Rader FFT algorithms. We used efficient arithmetic and algorithmic optimization techniques and leveraged parallel processing, pipelining, and optimized memory management to achieve high performance. The simulation and implementation results have demonstrated its competitive performance as compared to existing implementations for both power-of-two and non-power-of-two NTT sizes.

Acknowledgment

This work was supported by CyberSecurity Research Flanders with reference number VR20192203. In addition, this work is supported in part by the European Commission through the Horizon 2020 research and innovation program Belfort ERC Advanced Grant 101020005 and by the Defence Advanced Research Projects Agency (DARPA) under contract No. HR0011-21-C-0034 DARPA DPRIVE BASALISC. Michiel Van Beirendonck is funded by FWO as Strategic Basic (SB) PhD fellow (project number 1SD5621N).

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *41st ACM STOC*, M. Mitzenmacher, Ed. ACM Press, May / Jun. 2009, pp. 169–178.

[2] S. Halevi and V. Shoup, "Bootstrapping for HElib," Cryptology ePrint Archive, Report 2014/873, 2014, https://eprint.iacr.org/2014/873.

[3] N. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," Cryptology ePrint Archive, Report 2011/133, 2011, https://eprint.iacr.org/2011/133.

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *Electron. Colloquium Comput. Complex.*, vol. TR11-111, 2011. [Online]. Available: https://eccc.weizmann.ac.il/report/2011/111

[5] S. Halevi and V. Shoup, "Design and implementation of HElib: a homomorphic encryption library," Cryptology ePrint Archive, Report 2020/1481, 2020, https://eprint.iacr.org/2020/1481.

[6] L. Bluestein, "A linear filtering approach to the computation of discrete fourier transform," *IEEE Transactions on Audio and Electroacoustics*, vol. 18, no. 4, pp. 451–455, 1970.

[7] E. Öztürk, Y. Doröz, E. Savas, and B. Sunar, "A custom accelerator for homomorphic encryption applications," *IEEE Trans. Computers*, vol. 66, no. 1, pp. 3–16, 2017. [Online]. Available: https://doi.org/10.1109/TC.2016.2574340

[8] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: High-performance architecture for computation on homomorphically encrypted data in the cloud," Cryptology ePrint Archive, Report 2019/1066, 2019, https://eprint.iacr.org/2019/1066.

[9] A. C. Mert, E. Karabulut, E. Öztürk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Computers*, vol. 71, no. 11, pp. 2829–2843, 2022. [Online]. Available: https://doi.org/10.1109/TC.2020.3017930

[10] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE based cryptoprocessor," Cryptology ePrint Archive, Report 2013/866, 2013, https://eprint.iacr.org/2013/866.

[11] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," Cryptology ePrint Archive, Report 2020/446, 2020, https://eprint.iacr.org/2020/446.

[12] S.-Y. Wu, K.-Y. Chen, and M.-D. Shieh, "Efficient vlsi architecture of bluestein's fft for fully homomorphic encryption," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 2242–2245.

[13] H. Hsu and M. Shieh, "VLSI architecture of polynomial multiplication for BGV fully homomorphic encryption," in *IEEE International Symposium on Circuits and Systems, ISCAS 2020, Sevilla, Spain, October 10-21, 2020.* IEEE, 2020, pp. 1–4. [Online]. Available: https://doi.org/10.1109/ISCAS45731.2020.9181192

[14] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," Cryptology ePrint Archive, Report 2016/504, 2016, https://eprint.iacr.org/2016/504.

[15] R. Agarwal and C. Burrus, "Fast convolution using fermat number transforms with applications to digital filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 22, no. 2, pp. 87–97, 1974.

[16] I. J. Good, "The relationship between two fast fourier transforms," *IEEE Trans. Computers*, vol. 20, no. 3, pp. 310–317, 1971. [Online]. Available: https://doi.org/10.1109/T-C.1971.223236

[17] C. Rader, "Discrete fourier transforms when the number of data samples is prime," *Proceedings of the IEEE*, vol. 56, no. 6, pp. 1107–1108, 1968.

[18] M. Parker and M. Benaissa, "Unusual-length number-theoretic transforms using recursive extensions of rader's algorithm," *IEE Proceedings - Vision, Image and Signal Processing*, vol. 142, pp. 31–34(3), February 1995. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ip-vis_19951689

[19] R. Geelen and F. Vercauteren, "Bootstrapping for BGV and BFV revisited," *J. Cryptol.*, vol. 36, no. 2, p. 12, 2023. [Online]. Available: https://doi.org/10.1007/s00145-023-09454-6

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

[21] R. Geelen, M. Van Beirendonck, H. V. L. Pereira, B. Huffman, T. McAuley, B. Selfridge, D. Wagner, G. Dimou, I. Verbauwhede, F. Vercauteren, and et al., "Basalisc: Programmable hardware accelerator for bgv fully homomorphic encryption," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 4, p. 32–57, Aug. 2023. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/11157