FPGA Acceleration of Number Theoretic Transform

Tian Ye[1](✉), Yang Yang[2], Sanmukh R. Kuppannagari[2], Rajgopal Kannan[3], and Viktor K. Prasanna[2]

[1] Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA
tye69227@usc.edu

[2] Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089, USA
{yyang172,kuppanna,prasanna}@usc.edu

[3] US Army Research Lab, Playa Vista, CA 90094, USA
rajgopal.kannan.civ@mail.mil

Abstract. Fully Homomorphic Encryption (FHE) is a technique that enables arbitrary computations on encrypted data directly. Number Theoretic Transform (NTT) is a fundamental component in FHE computations as it allows faster polynomial multiplication. However, it is computationally intensive and requires acceleration for practical deployment of FHE. The latency and throughput of existing NTT hardware designs are limited by the complex data communication pattern between adjacent NTT stages and the modular arithmetic operations. In this paper, we propose a parameterized architecture for NTT on FPGA. The architecture can be configured for a given polynomial degree, modulus and target hardware in order to optimize the latency and/or throughput. We develop a novel low latency fully pipelined modular arithmetic logic to implement the NTT core, the key computational unit of NTT. Streaming permutation network is used to reduce the data communication complexity between NTT stages. We implement the proposed architecture for various polynomial degrees, moduli, and data parallelism on state-of-the-art FPGAs. Experimental results show that our architecture configured to perform 4096 polynomial degree NTT achieves up to 1.29× and 4.32× improvement in latency and throughput respectively over state-of-the-art designs on FPGA.

Keywords: Number theoretic transform · Parallel computing · FPGA

1 Introduction

Fully Homomorphic Encryption (FHE) provides a solution to utilize cloud platforms in a trusted and secure manner by directly performing computations on

T. Ye and Y. Yang—Equal contribution.

© Springer Nature Switzerland AG 2021
B. L. Chamberlain et al. (Eds.): ISC High Performance 2021, LNCS 12728, pp. 98–117, 2021.
https://doi.org/10.1007/978-3-030-78713-4_6

encrypted data [17]. Polynomial multiplication is one of the most time-consuming operations in FHE applications [28]. Naive implementation of polynomial multiplication results in $O(N^2)$ time complexity, where N is the degree of the polynomial. Number Theoretic Transform (NTT) has been proposed to reduce the complexity to $O(N \log N)$. Profiling results from [28] show that NTT is a primary bottleneck in FHE based applications such as FHE-Convolutional Neural Networks accounting for 55.2% of the execution time. Therefore, high performance implementation of NTT will have a critical impact of FHE based applications.

FPGAs have gained a lot of traction due to their immense flexibility and high energy efficiency. They are being widely adopted in cloud platforms where they are attached to the data center nodes to design highly customized, domain specific accelerators [13,27]. The logic density and compute throughput of state-of-the-art FPGAs have increased dramatically in recent years [18,35]. They also provide fine-grained memory access to high bandwidth on-chip SRAMs and external DRAMs. These features make them a logical choice for accelerating compute intensive applications such as NTT.

However, it is non-trivial to efficiently utilize the abundant FPGA resources for NTT to achieve low latency and high throughput. First, NTT requires complex data communication between computation stages due to loop-dependent permutation stride[1] in the algorithm [14]. Previous FPGA implementations have used all-to-all connections to facilitate communications [22,29,32]. The routing complexity increases quadratically with the data processing rate per cycle [31]. The design in [24] fully unrolls all the computation stages and uses fixed-function switch to reduce complexity. The wiring length as well as the interconnect area doubles from stage i to stage $i + 1$. In addition, due to the high polynomial degree in FHE applications [2], input coefficients often are not available concurrently. This adds an extra layer of complexity as the communication pattern also changes for different input data beats in the same computation stage. Second, designing low latency NTT cores to execute the key computation operation of NTT is challenging due to high resource requirements of modular arithmetic. Arbitrary modular arithmetic requires division operations that are expensive in FPGAs. Although division avoiding reduction algorithms [6,23] for arbitrary fixed modulus have been developed, they require additional multiplications, thereby incurring high latency. Lastly, based on the application, we may seek to minimize the resource consumption, maximize performance, or optimize some weighted combination of these. Thus, a parameterized design is desirable.

In this paper, we design an FPGA-based fully pipelined high performance NTT architecture. The architecture is parameterized and can be configured to support a wide range of polynomial degrees, moduli, and data parallelism. We use data parallelism, parallel input and output coefficients per cycle, to control the required I/O bandwidth for a given implementation. These parameters can be chosen at design time to meet latency and throughput requirements as well as the device resource constraints. To improve throughput, our design fully unrolls all the NTT computation stages. We employ streaming permutation net-

[1] Given a stride S, a permutation stride is defined as reordering an m-element data vector such that elements with distance of S are shifted into adjacent locations.

work (SPN) to reduce the routing complexity between NTT stages [10]. SPN reduces routing complexity by trading expensive long wires and switches with more pipeline stages. It can scale to large data parallelism with lower cost in terms of wiring and interconnect area compared to other types of interconnect such as crossbar. To obtain low latency NTT core with modular multiplication, our design supports any prime modulus q that is produced by choosing positive integers i and j to satisfy the property $2^j \equiv 2^i - 1 \pmod q$ (henceforth referred to as the *modulus property*). For such a modulus q, the modulo operation can be replaced by repeated additions, subtractions and shift operations (Sect. 3.3). As a result, an NTT core with low latency and low resource requirements can be realized. Note that the algorithm proposed in [37] is designed only for $q = 2^{14} - 2^{12} + 1$. In this work, we generalize the algorithm to support any q that satisfies the modulus property.

The key contributions of this paper are:

- We design a parameterized NTT architecture on FPGA that can support a wide-range of polynomial degrees, moduli, and data parallelism. Given the polynomial degree and the hardware resource constraints, our architecture can be configured to obtain high throughput and low latency.
- We utilize streaming permutation network to support various data parallelism and to reduce the data communication complexity between NTT stages. This technique enables our architecture to be fully unrolled and pipelined for all the NTT stages, which leads to high throughput.
- To obtain low latency, we develop a compact NTT core that can perform modular arithmetic operations without any multiplication. Our NTT core design can be used to generate a collection of algorithms for different moduli as required by the given application.
- We implement our architecture for various polynomial degrees, moduli and data parallelism on state-of-the-art FPGAs. It can be configured to perform 512, 1024, 2048 and 4096 polynomial degree NTT in less than 0.57 µs, 0.76 µs, 1.03 µs and 1.99 µs respectively. By further increasing data parallelism, throughput of 43.0, 20.6, 9.2 and 1.7 million transforms per second is achieved for 512, 1024, 2048 and 4096 polynomial degree NTT respectively.
- Our design achieves superior throughput while also improving the latency compared with state-of-the-art designs on the same hardware. We improve the latency up to 1.29× and the throughput up to 4.32×.

2 Related Work

NTT Acceleration: Recent work [22,29,32] focus on optimizing memory layout to enable parallel and conflict-free memory access between NTT stages. However, in these designs require all-to-all connection between NTT cores and intermediate data memory, which limits the scalability. Throughput is also reduced due to reusing the same set of NTT cores across all the stages. A systolic array approach for NTT acceleration is presented in [25]. This architecture fully unrolls all

the NTT stages. However, data parallelism in NTT is not explored and computation in each NTT stage is serialized. The NTT hardware proposed in [5,26] is limited to a specific setting. Nejatollahi et al. use processing-in-memory technology to accelerate NTT [24]. The design unrolls all the NTT computation stages and all the input coefficients to improve parallelism, but latency and throughput are affected by the long computation cycles in Processing In-Memory (PIM) technology. [3,19,20,30] use CPU or GPU to accelerate NTT, but the optimizations in these designs cannot be applied due to the differences in the underlying architecture. The work in [19] executes several NTTs concurrently to exploit massive GPU parallelism. In such a design, reducing the batch size does not lead to reduced latency. Thus, this design is not suitable for our scenario where in addition to throughput, latency for a single NTT computation needs to be minimized.

Modular Multiplication: This is one of the key operations of NTT and many works have focused on its efficient implementation. In [21], the modulo algorithm allows the output to be slightly greater than the modulus q. This optimization avoids division operations, but still requires additional large-latency multiplications. [22] implemented an iterative modulo operation based on Montgomery reduction [23]. This can be resource-consuming as each iteration has a multiplication. In [29], modular multiplication is based on Barrett reduction [6]. It requires pre-computations depending on the twiddle factors. This consumes more on-chip storage. Also, their algorithm requires two additional multiplications. [37] designed an architecture for $q = 2^{14} - 2^{12} + 1$ that avoids any multiplication in the modular reduction. This results in low latency and reduced resource requirements. However, many applications of NTT have large coefficients, and thus need larger q. Therefore, our work extends this design for any prime q that satisfies $2^j \equiv 2^i - 1 \pmod{q}$ for some positive integers i and j. Please Sects. 3.3 and 4.2 for details.

To the best of our knowledge, existing work on FPGAs does not account for the performance impact of the interstage data. As a result, the scalability and achievable performance are limited. In contrast, we use streaming permutation network to enable a fully unrolled and pipelined architecture with variable data parallelism. We further develop low latency modular arithmetic unit that supports arbitrary modulus q satisfying the modulus property.

3 Background

3.1 Fully Homomorphic Encryption (FHE)

Homomorphic encryption is a practical approach for privacy-preserving computation using lattice-based cryptography [17]. It allows direct computations, including addition, scaling and multiplication, on ciphertext without access to the original data. There are a variety of encryption schemes, e.g., BGV [8], BFV [16] and CKKS [12]. For all these encryption schemes, both the plaintext and the ciphertext are high-degree polynomials, typically ranging from 2^{10} to

2^{15} [2]. The security level is quantified by two parameters, the degree of polynomials and the width of the selected modulus. Both are critical to the performance of homomorphic computations. Typical parameters for different security levels can be found in [2].

3.2 Number Theoretic Transform (NTT)

Polynomial multiplication is one of the most computationally expensive operation of homomorphic encrypted computations [28]. The complexity of multiplying two polynomials of degree N is $O(N^2)$. To reduce the complexity to $O(N \log N)$, number-theoretic transform (NTT) is used [1].[2] This simplifies the polynomial multiplication into N coefficient-wise multiplications.

Algorithm 1: Number Theoretic Transform

```
    Input: Coefficients A = (A[0], A[1], ..., A[n − 1]) and twiddle factors in
           bit-reversed order φ = (φ[0], φ[1], ..., φ[n − 1])
    Output: A ← NTT(A) in bit-reversed order
 1  for (m ← n/2; m ≥ 1; m ← m/2) do
 2      for (i ← 0; i < n/(2m); i ← i + 1) do
 3          S ← φ[n/(2m) + i]
 4          for (j ← 0; j < m; j ← j + 1) do
 5              U ← A[2m · i + j]
 6              V ← S · A[2m · i + j + m] mod q
 7              A[2m · i + j] ← U + V mod q
 8              A[2m · i + j + m] ← U − V mod q
 9          end
10      end
11  end
```

In Algorithm 1, each iteration of the outer loop is called a stage. The algorithm has $\log N$ sequential stages as the outer loop in Line 1, and each stage has $N/2$ independent instances of Line 5–8 that can be computed in parallel. Each instance of Line 5–8 takes two coefficients as input, performs modular arithmetic and updates the two coefficients. Modular arithmetic includes modular multiplication, addition and subtraction. Note that the computational pattern of the NTT algorithm is similar to that of the FFT algorithm. However, NTT performs modular arithmetic on integer coefficients as opposed to FFT which performs arithmetic on complex numbers. As performing modular arithmetic is computationally expensive, existing FFT implementations such as [11,36] cannot be trivially extended to accelerate NTT.

3.3 Modular Reduction

As NTT limits the coefficients to be in a finite ring of integers, modular computations are required in the algorithm. Modular addition and subtraction are

[2] All logs in this paper are to base 2.

trivial, which require only one more addition or subtraction. In contrast, it is non-trivial to design an efficient modular multiplication, as division and modulo operations are expensive on FPGAs. Typical modular reduction algorithms, e.g., Barrett reduction [6] and Montgomery reduction [23], replace the expensive division operation with multiplication operation when the modulus is preconfigured. A recent work [21] proposes a relaxation that allows the output to be slightly longer than the modulus, which speeds up the reduction. However, they still require resource-consuming multiplication operation. Instead, we utilize the design proposed in [37] that has additions and subtractions only. We generalize it from $q = 12289$ to any q that satisfies $2^j \equiv 2^i - 1 \pmod{q}$. Algorithm 2 is an example of the reduction algorithm for $q = 2^{28} - 2^{16} + 1$ that avoids additional multiplications. The algorithm is hardcoded for a specific value of q. We also provide designs similar to Algorithm 2 for a collection of different q values. One of them can be selected and embedded into NTT cores at design time.

Algorithm 2: Reduction for $q = 2^{28} - 2^{16} + 1$

Input: 56-bit integer $z[55:0]$
Output: $y = z \bmod q$

1. $c \leftarrow z[55:52] + z[51:40] + z[39:28]$
2. $d \leftarrow z[55:52] + z[55:40] + z[55:28]$
3. $e \leftarrow c[13:12] + c[11:0]$
4. $f \leftarrow ((e[12] + e[11:0]) << 16) - (e[12] + c[13:12])$
5. $y \leftarrow f + z[27:0]$
6. **if** $y \geq q$ **then**
7. | $y \leftarrow y - q$
8. **end**
9. $y \leftarrow y - d$
10. **if** $y < 0$ **then**
11. | $y \leftarrow y + q$
12. **end**

3.4 Challenges in Accelerating NTT

The two main challenges in NTT are the implementation of the NTT core and the interstage connection network. The key component of the NTT core is modular multiplication which is usually resource-consuming and slow, as discussed in Sect. 3.3. An efficient design for the NTT core, especially modular multiplication, is necessary to reduce the latency and resource consumption. Our design of the low-latency NTT core is described in Sect. 4.2. The interconnection between the NTT stages is a butterfly network, which has high complexity in terms of wiring length and area. For a naïve butterfly network, the wiring length and the interconnect area doubles from stage i to stage $i+1$ [33]. This is prohibitively large. Existing works proposed several ways to address the challenge. The designs proposed in [22,29] fold all the NTT stages and reuse the same set of NTT cores for all the stages. They simplify the interconnection by data reading and writing in the on-chip memory. However, this method results in low throughput due to the folding of all the stages. In this paper, we support a fully unrolled and pipelined

design. We use a streaming permutation network [10] as interconnection with a low resource requirement.

4 Accelerator Design

4.1 Design Methodology

Key NTT parameters such as the degree of polynomial and the modulus width are often chosen by considering not only the level of security, but also the latency, throughput, and hardware resource constraints. As a result, these parameters can differ considerably across homomorphic encrypted (HE) applications [3,4,15]. It is desirable to design the hardware architecture such that it can be configured to run NTT with different settings easily. In addition, due to the high polynomial degree in HE-based applications [2], processing all the input coefficients concurrently requires high hardware resource and I/O bandwidth. Common loop tiling technique is often used to fold input coefficients into smaller groups. Each group is then processed in a streaming manner. This technique reduces I/O bandwidth and hardware resources, but permuting streaming data is challenging as data elements need to be moved across both spatial and temporal dimensions. The design in [25] only processes two coefficients per cycle in each NTT computation. This greatly simplifies the data permutation but leads to inefficient utilization of the available bandwidth. Other NTT hardware implementations on FPGA use carefully designed intermediate data layout in on-chip SRAM to reduce memory access conflicts [22,29]. Complex routing and arbitration logic are needed to permute data in each NTT computation stage. As a result, the number of NTT cores is limited in these designs, which impacts the NTT latency and throughput.

Our NTT hardware architecture is constructed by fully unrolling and pipelining all the NTT computations stages. NTT input and output are folded and processed in a streaming fashion to satisfy I/O bandwidth constraint and to reduce resource consumption. Key NTT algorithmic and architecture settings are exposed as parameters, allowing hardware re-configuration to support various NTT use cases. We define the following parameters that can be specified at design time to customize our architecture:

- Polynomial Degree (N): Application parameter. It determines the polynomial degree for number theoretic transform. Our architecture supports any polynomial degree. For HE-based computation, N is typically a power-of-two number and between 2^{10} to 2^{15} [2,12].
- Modulus (q): Application parameter. Modulus used in Algorithm 1. We support prime modulus q that is produced by choosing positive integers i and j to satisfy the property $2^j \equiv 2^i - 1 \pmod{q}$. i and j determine the bit width of the modulus and the polynomial coefficients.
- Data Parallelism (p): Architecture parameter. It determines the number of coefficients being processed per cycle in each NTT computation stage ($2 \leq p \leq N$). Higher data parallelism improves latency and throughput but requires more I/O bandwidth and FPGA resources. To reduce design complexity, p is restricted to be a power-of-two number.

– Pipeline Parallelism: Architecture parameter. It determines the unrolled NTT computation stages in the NTT hardware. In this paper, we fix this parameter to $\log N$.

The proposed architecture receives p input coefficients per cycle. After a fixed delay, it starts to produce p output coefficients per cycle. Our design does not have restriction on the location of the input and output data. They can be from external DRAM or other IP blocks inside the FPGA. We utilize direction connection permutation (when permutation stride is less than p) and streaming permutation network (SPN) [10] (when permutation stride is greater than or equal to p) to facilitate data communication between NTT stages (Sect. 4.3). Given parameter N and p, our architecture instantiates $\log N$ computation stages, and each stage contains $p/2$ NTT cores. Figure 1 and 2 present the top-level architecture of 16-point NTT with $p = 4$ and $p = 8$ respectively. There are 4 NTT computation stages, and 3 permutation networks are needed in the design.

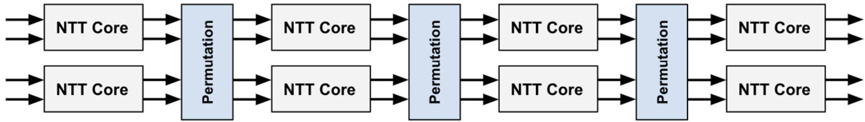

Fig. 1. Top level architecture of 16-point NTT with $p = 4$.

In contrast to prior work, SPN avoids the complex routing and arbitration logic between NTT computation stages by trading expensive long wires and switches with multi-stage parallel routing networks. The highly scalable and extensible SPN can be configured to realize arbitrary stride permutation between its input and output. To reduce the modular arithmetic latency, customized NTT core is developed to replace costly multiplications in modulo operations with additions, subtractions, and shift operations (Sect. 4.2). The utilization of streaming permutation network and low latency NTT core allows us to fully unroll and pipeline all the NTT computation stages to obtain low latency and high throughput.

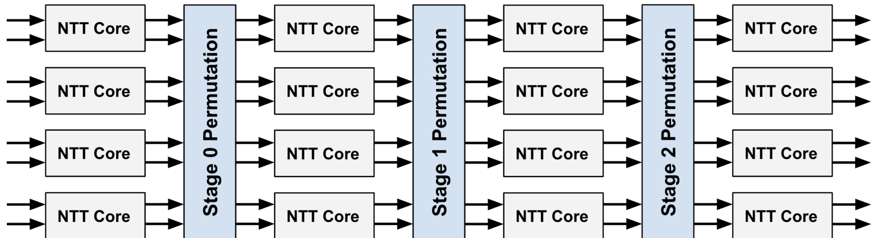

Fig. 2. Top level architecture of 16-point NTT with $p = 8$.

The parameterized architecture opens up design space trade-offs concerning latency, throughput, I/O bandwidth constraint, resource consumption, and application requirement. Applications need to consider N and q accordingly in order to achieve certain level of security [2,12]. For example, with $N = 1024$, it is recommended to use 27-bit modulus q in order to achieve 128-bit security [2]. Larger polynomial degree (N) needs more NTT computation stages, thereby increasing the latency. Higher data parallelism (p) can speed up the NTT latency and throughput, but it has implications on routing resources and I/O bandwidth requirements. Given application parameters N and q, the largest data parallelism (p) can be determined by considering the following constraints:

- I/O Bandwidth: Each input and output coefficient is of size $\lceil \log q \rceil$ bits. Given available input and output bandwidth BW, p can be chosen such that $2 \times p \times \lceil \log q \rceil \times Fmax = BW$, where $Fmax$ is the FPGA design frequency and a factor of 2 is to account for both input and output.
- FPGA Resources: The architecture is implemented under limited LUT, BRAM and DSP resources. Each NTT core requires LUT and DSP resources. Due to fully unrolling all the NTT stages, there are $p/2 \log N$ NTT cores. SPN consumes LUT and BRAM resources, there are $(\log N - \log p)$ stages using SPN since those stages have permutation stride greater than or equal to p.

Our parameterized architecture provides users with the flexibility to configure N, p, and q with a variety of options.

4.2 NTT Core

The NTT core is used to perform the inner loop body of the NTT algorithm that receives two coefficients as inputs and generates two coefficients as outputs. The key component of the NTT core is the module for modular multiplication. To perform the modulo q operation efficiently, we use a design similar to the one proposed in [37]. Moreover, we generalize it by providing designs for a collection of prime q values. Those q are produced by choosing positive integers i and j to satisfy the property $2^j \equiv 2^i - 1 \pmod{q}$. To illustrate the algorithm, we use $q = 2^{28} - 2^{16} + 1$ as an example. In this example, all coefficients are 28 bits, and thus the multiplication result is up to 56 bits. Denote the 56-bit number as $z[55:0]$, and it can be reduced in the following way for the first step:

$$\begin{aligned} z[55:0] &= 2^{28} \cdot z[55:28] + z[27:0] \\ &= (2^{16} - 1) \cdot z[55:28] + z[27:0] \end{aligned} \tag{1}$$

Essentially, any occurrence of 2^{28} is replaced by $2^{16} - 1$. The reduction can be repeated until the result is less than 2^{28}. The entire algorithm is illustrated in Algorithm 2. It only includes additions, subtractions and bit-wise operations without multiplications, so the latency and resource consumption are low.

Although the algorithm is highly dependent on the value of q, different values of q have similar algorithms. Specifically, the algorithms for other q still have

the same kinds of operations as Algorithm 2, but they have different numbers of inputs in Line 1–2 and bit widths of the inputs in Line 1–5. For a given q, by customizing the number of inputs and bit widths, we can obtain low latency NTT cores using Algorithm 2. The latency and resource utilization for various q are evaluated in Sect. 5.4.

Note that only supporting q values with the modulus property does not make the NTT design less applicable. From the perspective of polynomial multiplications, the modulus q only needs to be a prime greater than the maximum coefficient of the input polynomial. There are many eligible prime q satisfying the property. For example, it can be verified that there are over 100 such q ranging from 14 bits to 60 bits. Therefore, for any given polynomial, we can easily choose the smallest q greater than all the coefficients.

4.3 Permutation Network

Parallel input data is required to be permuted before being processed by the subsequent NTT cores, since each computation stage has a different stride (S). As described in Algorithm 1, S can be formulated as $S_i = 2^{\log N - i - 1}$, where i is the computation stage and satisfies $0 \leq i < \log N$. The last stage has stride equal to 1, so no permutation is needed.

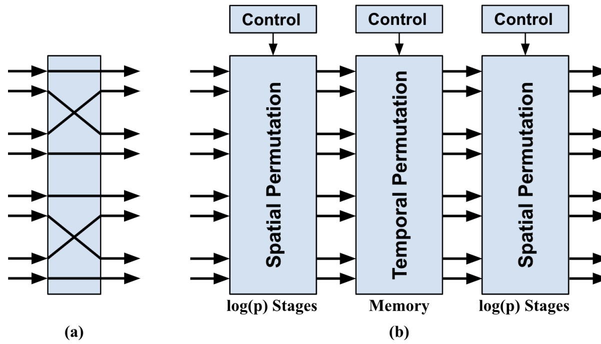

Fig. 3. Permutation network of the 16-point NTT with $p = 8$. (a) Direct connection permutation with $S = 2$. (b) Streaming permutation network.

Due to the fully unrolled design, each stage has a fixed permutation pattern and does not require dynamic re-configuration. Our architecture employs two types of permutation modules, as shown in Fig. 3. With $S < p$, later stages of the NTT computation, only spatial permutation is needed—shuffling data within the p inputs in the same cycle. We use a direct connection permutation with fixed wiring for these stages. This type of permutation module achieves low latency but cannot permute data in the time dimension across different input beats. For earlier stages, with $S \geq p$, streaming permutation network in [10] is used to re-arrange input data from different cycles in a streaming fashion. As shown in

Fig. 3(b), the datapath consists of two p-to-p spatial permutation networks and one temporal permutation network. Spatial permutation, reordering within the p data inputs in the same cycle, is realized using the classic Benes network [7]. In the middle, temporal permutation uses on-chip SRAM to rearrange data across different cycles. The control logic, which includes routing information and memory read/write addresses, are generated statically at IP core configuration time.

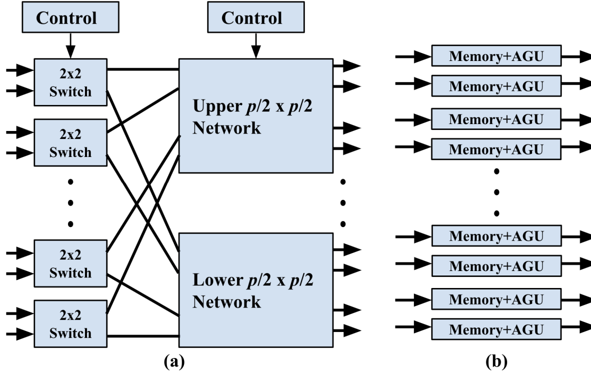

Fig. 4. Microarchitecture of the spatial and temporal permutation sub-network in streaming permutation network. (a) Spatial permutation. (b) Temporal permutation.

Figure 4 shows the microarchitecture of the spatial and temporal sub-networks. As shown in Fig. 4(a), spatial permutation network is implemented using Benes network [7]. A Benes network is a multi-stage routing network, with the first and last stage each has $p/2$ 2×2 switches. In the middle, there are two $p/2 \times p/2$ sub-networks, and each can be decomposed into the three-stage Benes network recursively. Compared to a naive crossbar interconnect, which requires $O(p^2)$ connections, each spatial permutation network has $(p/2) \cdot \log p$ 2×2 switches. Thus, streaming permutation network in our design asymptotically has lower complexity. Moreover, wiring length in the network does not change with permutation stride [9]. Each 2×2 switch has one control bit to route inputs to the upper or lower sub-networks respectively.

Figure 4(b) illustrates the design of temporal permutation network. It has p dual-port memory blocks and p address generation units (AGU). AGU produces the control signals and addresses to the memory block it connects to. Each AGU issues memory read and write addresses independently, thereby achieving temporal permutation across data received in different cycles.

As N data points stream through the interconnect with p per cycle, they are first permuted spatially by the first spatial permutation network, then the data are written into the p memory blocks. Finally, p data points with stride S_i are read out per cycle and permuted again by the second spatial permutation network. Since the architecture parameters – N and p – are fixed at run-time, the

configurations for all the 2×2 switches and the AGUs can be determined offline and remain valid as long as N and p don't change. We store this information in FPGA's on-chip memory. More details about the routing algorithm can be found in [10].

5 Experiments and Results

5.1 Experimental Setup

In this section, we present a detailed evaluation of the proposed NTT architecture. All the designs are implemented using SystemVerilog on Virtex-7 XC7VX690 and XC7VX980 FPGA. The XC7VX690 device has 433,200 LUTs, 866,400 Flip-Flops and 1,470 BRAMs; the XC7VX980 device has 612,000 LUTs, 1,224,000 Flip-Flops and 1,500 BRAMs. Both devices have 3,600 DSPs. We use Xilinx Vivado 2020 to perform synthesis, place and route.

Our flexible and scalable architecture gives users a wide range of design options based on the application requirement and resource availability. We evaluated the performance and resource utilization of our designs by varying the polynomial degree (N) and data parallelism (p). We use $\langle x, y \rangle$ to denote a design with $N = x$ and $p = y$. Based on the widely used NTT parameters [2,3], we conducted experiments with NTT polynomial degree $N = 512$, 1024, 2048. We conducted a sweep over a range of available bandwidth to our designs by choosing p from 32 to 128. The metrics for performance analysis are NTT latency in μs and throughput in *polynomials transformed per second*. The performance metrics were measured by running post place and route simulations. The resource utilization is reported in terms of usage of LUTs, Flip-flops, BRAMs, and DSPs.

5.2 Performance Evaluation

Table 1 shows the measured NTT performance for various polynomial degrees (N) and data parallelism (p) on XC7VX980 FPGA. In this set of experiments, we fixed the modulus and the polynomial coefficients to 28 bits, which is commonly used by prior work [3,22]. Note that our architecture can easily support modulus with different bit width as described in Sect. 4.2. End-to-end latency from receiving the first input data to producing the last output coefficient is in the range of 0.57 µs to 1.29 µs. Higher polynomial degree incurs higher latency due to the additional NTT computation stages and also due to increased I/O time for a given p. Our architecture can achieve a high operating frequency between 210 MHz and 220 MHz for $p = 32$ and $p = 64$ designs. For a given N, $p = 128$ design consumes the least number of clock cycles. Due to the increased parallelism in each stage, majority of the NTT stages in $p = 128$ design uses fixed direct connection instead of streaming permutation network (Sect. 4.3), which reduces the latency in terms of clock cycles. However, $p = 128$ has many more NTT cores than $p = 32$ and $p = 64$ designs. It increases the resource consumption significantly and poses challenge during the place and route phase.

Table 1. Measured performance and resource utilization of complete NTT designs on XC7VX980 FPGA

Design	Latency	Throughput	LUT	FF	BRAM	DSP
$\langle 512, 32 \rangle$	0.66	13,750,000	82,498	89,688	64	576
$\langle 512, 64 \rangle$	0.57	26,625,000	161,703	171,472	96	1,152
$\langle 512, 128 \rangle$	0.60	43,000,000	303,040	316,920	156	2,304
$\langle 1024, 32 \rangle$	0.91	6,781,250	94,394	104,846	80	640
$\langle 1024, 64 \rangle$	0.76	13,125,000	187,283	204,162	128	1,280
$\langle 1024, 128 \rangle$	0.82	20,625,000	360,765	391,945	234	2,560
$\langle 2048, 32 \rangle$	1.29	3,390,625	107,293	120,718	96	704
$\langle 2048, 64 \rangle$	1.03	6,468,750	212,835	237,719	160	1408
$\langle 2048, 128 \rangle$	1.18	9,250,000	418,214	468,230	312	2,816

As a result, we observe that the frequency drops to 150 MHz - 170 MHz, and the overall latency for $p = 128$ is higher than $p = 64$ for the same value of N.

Sustained throughput in terms of *polynomials transformed per second* is also shown in Table 1. Different from the latency results, $p = 128$ designs perform the best from throughput perspective. Although frequency of the designs with $p = 128$ drops almost 25% compared with the other designs, $p = 128$ has $2\times$ and $4\times$ the processing rate compared to $p = 64$ and $p = 32$ respectively. The processing rate increase helps offset the frequency drop in this case. In the best-case scenario ($N = 512$), our architecture can transform more than 40 million *polynomials per second* in a streaming fashion.

The results on latency and throughput verify the scalability and flexibility of our architecture. Since our architecture is fully pipelined and can process p inputs per cycle, different data parallelism (p) requires different bandwidth to stream the input and the output coefficients. Table 2 shows the required I/O bandwidth in order to fully utilize the hardware pipeline when performing $N = 1024$ NTT on VX980 FPGA. p is the primary factor that influences the required bandwidth. Since p also means the design generates p output coefficients per cycle, the same amount of input bandwidth is needed on the output side. For $p = 128$, our design requires a sustained total bandwidth of 170 GB/s. This bandwidth can be made available if the polynomials are stored in on-chip SRAM (i.e., produced by other IP cores within the FPGA) or in high bandwidth external memory such as DDR4 or HBM. When adopting our architecture for different applications, p is an important parameter for bandwidth allocation.

5.3 Resource Utilization

The resource utilization of our implementations is reported in Table 1. Since each stage requires $p/2$ NTT cores, and there are $\log N$ stages in total, higher data parallelism (p) demands more hardware resources. On the other hand, with more parallel inputs per cycle, fewer stages need streaming permutation network

Table 2. I/O bandwidth required to fully utilize the proposed NTT accelerator with $N = 1024$, 28-bit input and output coefficients

Device	Data parallelism (p)	Input bandwidth [GB/s]	Output bandwidth [GB/s]
VX980	32	27.8	27.8
VX980	64	53.8	53.8
VX980	128	84.5	84.5

(Sect. 4.3), which helps reduce the resource consumption. We observe that the reduction in streaming permutation network is less than the increase in the resource demand with more NTT cores.

Overall, Table 1 shows that optimizing different metrics can lead to different design configurations. Given a polynomial degree N, one may choose to use $p = 64$ as the latency optimized design, $p = 128$ as the throughput optimized design, and $p = 32$ as the design that requires the least hardware resources (Sect. 5.2). An optimal design should be obtained by having a holistic view on the system requirements in terms of latency, throughput, resource availability and application requirements.

5.4 Evaluation of NTT Core and Streaming Permutation Network

NTT Core: As the modulus q affects the modulo algorithm in the NTT cores, we evaluate how the resource utilization and achieved frequency for NTT cores vary for various q. We performed experiments on a standalone NTT core for $\lceil \log q \rceil = 16, 27, 28$ and 32. The results are shown in Table 3.

Note that this experiment is performed on a single NTT core instead of the entire architecture, so the frequency shown in Table 3 are higher than the ones in the integrated experiments. Except for the 27-bit case, smaller q values consume less resources in terms of LUTs and FFs and achieve higher frequency. Note that $q = 2^{27} - 2^{21} + 1$ utilizes more resources than $q = 2^{28} - 2^{16} + 1$; this is because it needs to sum up 5 inputs for the first two steps, in contrast to the 3 inputs for the 28-bit case as shown in Line 1–2 of Algorithm 2. Due to the same reason, the 27-bit case has a lower frequency. However, the variance of the frequency for all the cases is not significant. Also, all of them have the same latency of 5 cycles, so the impact of q on the overall performance is minimal.

Streaming Permutation Network: We evaluate the latency and resource consumption of the streaming permutation network by synthesis, place and route each streaming permutation network in $\langle 1024, 32 \rangle$, $\langle 1024, 64 \rangle$, and $\langle 1024, 128 \rangle$ designs as a standalone module on VX980 FPGA. Input and output coefficients are 28-bit wide. Each streaming permutation network has two spatial permutation sub-networks, which has $\log p$ stages, and one temporal permutation sub-network. Figure 5 shows the resource utilization of streaming permutation network for $p = 32, 64, 128$ with permutation stride $S = 512$. For a given p, the

Table 3. FPGA resource utilization on VX690 for NTT core with various moduli

Modulus q	LUT	FF	DSP	Latency	Frequency
16 bits $(2^{16} - 2^{12} + 1)$	246	206	1	5 cycles	281 MHz
27 bits $(2^{27} - 2^{21} + 1)$	485	424	4	5 cycles	262 MHz
28 bits $(2^{28} - 2^{16} + 1)$	458	376	4	5 cycles	274 MHz
32 bits $(2^{32} - 2^{20} + 1)$	534	479	4	5 cycles	270 MHz

resource consumption is very similar for different S, therefore we omit the details for other strides in the interest of space. We observe close to linear increase in resource consumption with the increase in data parallelism. The majority of BRAM resources consumed by the streaming permutation network is due to the temporal permutation network. It requires p independent memory blocks, each of which is mapped to 1 BRAM18 resource configured as simple dual-port mode. Each BRAM18 memory stores at most $1024/p$ data inputs. The BRAM resource reported in the Fig. 5 is based on BRAM36 resource, each BRAM36 contains 2 BRAM18 blocks. As p increases to 128, BRAMs are also used to store the configuration tables in the spatial permutation sub-networks. As a result, it requires 14 extra BRAMs. LUTs are mainly used by configuration tables and the AGUs.

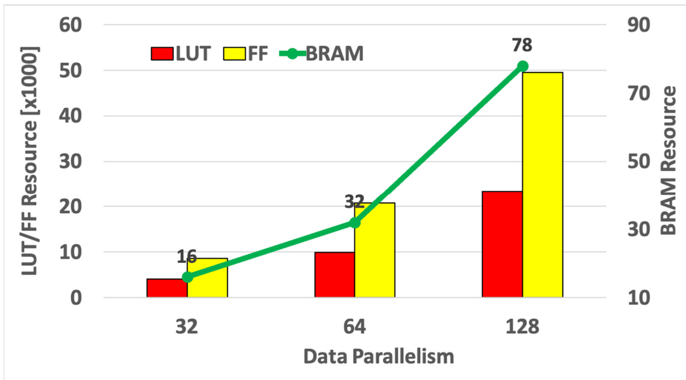

Fig. 5. Resource utilization for streaming permutation network $S = 512$ in $\langle 1024, 32\rangle$, $\langle 1024, 64\rangle$, and $\langle 1024, 128\rangle$ designs. 28-bit per input and output coefficients are used.

Figure 6 shows the latency in cycles and frequency in MHz for each streaming permutation network, measured from the time the first input is received to the time the first output is produced. Latency is between 15 cycles to 30 cycles, depending on the values of p and S. The latency of spatial permutation subnetwork only grows logarithmically, as there are $\log p$ stages in each spatial permutation sub-network. For a given p, as S varies, the spatial permutation latency

does not change. Temporal permutation latency increases with S because the hardware needs to wait for more data inputs before it can generate the first output. But temporal permutation resource consumption does not change much with different S. Good scalability is also observed with regard to frequency, we observe 340 MHz for p =32,64 designs and 315 MHz for p = 128.

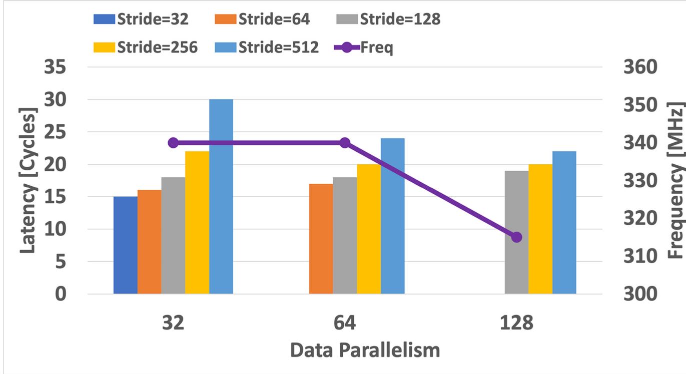

Fig. 6. Latency of streaming permutation network in $\langle 1024, 32\rangle$ design. 28-bit per input and output coefficients are used.

5.5 Comparison with Prior Work

We compare our design with existing implementations on FPGA [22,25,29] in terms of the consumed resources, latency and throughput. Recall that the latency is the duration between receiving of the first input at an input port and generating of the last output at an output port. The throughput is the number of transformed polynomials per second. We use our performance results obtained on XC7VX690 FPGA, which is the same device used in [22]. Table 4 shows the comparison.

Both designs in [22,29] fold all the stages and reuse the same set of processing elements for all the stages. Without unrolling all the stages, this leads to much lower throughput than ours. Complex routing and arbitration logic is needed between NTT cores and intermediate data buffer. For modulo arithmetic, their designs offer more flexibility than ours by supporting all moduli with a general modular multiplication design. However, this design choice requires more DSPs per NTT core and has higher latency.

In [29], the authors only reported resources and performance for $N = 4096$ and $\lceil \log q \rceil = 52$ with at most 32 NTT cores on Intel FPGAs. For a fair comparison, we compute the utilization of Xilinx DSPs for their NTT core for $\lceil \log q \rceil = 28$. According to Algorithm 1 in [29], their NTT core includes three 30×30 partial multiplications, two of which only output lower 30 bits and one of which outputs higher 30 bits. The former can be implemented by three 15×15 multiplications and the latter needs four 15×15 multiplications. Thus, the entire

Table 4. Comparison with prior work

Design	[22]	[25]	This paper $p = 32$	This paper $p = 64$	[29][a]	This paper $p = 32$
Platform	VX690	Zynq UltraScale+	VX690	VX690	VX690	VX690
N	1024	1024	1024	1024	4096	4096
$\lceil \log q \rceil$	28	16	28	28	28	28
LUT	132K	3K	94.4K	187.2K	–	117.3K
FF	59K	3K	104.5K	205.5K	–	135.2K
BRAM	96	29	80	128	–	189
DSP	448	58	640	1280	320	768
Freq. [MHz]	125	183	215	212	300	224
Energy [μJ]	–	12.52	9.4	14	–	22.9
Latency [μs]	2	101.84	0.92	0.75	2.56	1.99
Throughput	500,000	98,193	6,718,750	13,250,000	390,625	1,687,500
Throughput per DSP	1,116	1,693	10,498	10,352	1,220	2,197
Throughput per LUT	3.78	32.7	71.17	70.77	–	14.38

[a] The performance and resource utilization of [29] are extrapolated.

NTT core includes ten 15 × 15 multiplications, which needs 10 DSPs according to [34]. We also verified this by actually implementing their NTT core on Xilinx VX690. We choose the largest design with 32 NTT cores from [29]. We assume that their design can still achieve 300 MHz frequency as they reported, which is an optimistic upper bound. The estimated latency, throughput and throughput per DSP are shown in the last two columns of Table 4. Our design for $N = 4096$ with $p = 32$ achieves 4.32× improvement in throughput, 1.80× improvement in throughput per DSP and 1.29× improvement in latency compared with the design in [29].

The design in [22] has the same N and $\lceil \log q \rceil$ as our sample design, and they also use VX690 FPGA as the target platform. Even though our architecture fully unrolls all the NTT stages, our design with $p = 32$ still has similar hardware cost compared with theirs. This is mainly due to the resource-efficient NTT cores in our architecture. We can achieve superior performance due to increased FPGA frequency and fully pipelined design. Our design with $p = 32$ achieves 2.17× improvement in latency and 9.41× in throughput per DSP compared with the design in [22].

The design in [25] fully unrolls and pipelines all the $\log N$ stages. We calculate their throughput assuming their systolic array is fully pipelined. Note that this gives an upper bound on their throughput. Different from our approach, data communication complexity is greatly simplified in their design as each systolic processing element only processes two coefficients per cycle in each NTT stage. This can also lead to under-utilization of I/O bandwidth. Their design can be mapped to our architecture by setting $p = 2$. The performance of their design is reduced significantly as the amount of parallelism is small. However, their design

consumes very small amount of resources, which is beneficial for devices with limited resources or power constraints.

In addition to the prior work shown in Table 4, a ReRAM-based ASIC architecture is proposed in [24] to accelerate NTT. It performs fine-grained computations using the PIM technology. The architecture consumes very low energy [24]. The simulated design runs at 910 MHz. This requires a sustained I/O bandwidth of 2.3 GB/s for $N = 1024$. However, the arithmetic operation in each stage requires $O(\log^2 q)$ cycles. In the VLSI model [33], the time complexity (latency) is $O(\log N \log^2 q)$ and the area is $O(N \log N \log q + N^2)$. The second term of the area is for the interconnection between stages. For $N = 1024$ and $\lceil \log q \rceil = 16$, the design simulation in [24] shows latency of 83.12 μs and throughput of 553 thousand transforms per second. Note that for $N = 1024$ and $\lceil \log q \rceil = 28$, our design with $p = 32$ has latency of 0.92 μs and throughput of 6.7 million transforms per second.

6 Conclusion

In this paper, we designed an FPGA architecture for NTT with configurable parameters including polynomial degree, modulus and data parallelism. We utilized streaming permutation network as interconnection between each stage to reduce complexity. We also developed a low-latency design for modulo operations. The experiments for polynomials of degree 4096 showed that our design achieves 4.32× throughput compared with the state-of-the-art design on FPGA while also improving the latency by 1.29×. Thus, our design can be used to implement both high throughput NTT intensive workloads as well as low latency NTT inference workloads such as privacy preserving ML inference.

In the future, we will make our design more flexible by allowing reconfiguration of polynomials with variable degrees and moduli at runtime. Also, we will develop a design space exploration tool for trade-off analysis on performance, resource consumption and application requirements.

Acknowledgement. This work has been sponsored by the U.S. National Science Foundation under grant numbers OAC-1911229 and CNS-2009057. Equipment grant by Xilinx is greatly appreciated.

References

1. Aho, A.V., Hopcroft, J.E.: The Design and Analysis of Computer Algorithms. Pearson Education India (1974)
2. Albrecht, M., et al.: Homomorphic encryption security standard. Tech. rep. (2018)
3. Alkim, E., Barreto, P.S.L.M., Bindel, N., Kramer, J., Longa, P., Ricardini, J.E.: The lattice-based digital signature scheme qTESLA. In: ACNS (2020)
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange: a new hope. In: USENIX SEC (2016)
5. Banerjee, U., Ukyab, T.S., Chandrakasan, A.P.: Sapphire: a configurable crypto-processor for post-quantum lattice-based protocols. In: TCHES (2019)

6. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: CRYPTO 1986 (1987)
7. Beneš, V.E.: Optimal rearrangeable multistage connecting networks. Bell Syst. Tech. J. **43**(4), 1641–1656 (1964)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: ITCS (2012)
9. Chen, R., Park, N., Prasanna, V.K.: High throughput energy efficient parallel FFT architecture on FPGAs. In: HPEC (2013)
10. Chen, R., Prasanna, V.K.: Automatic generation of high throughput energy efficient streaming architectures for arbitrary fixed permutations. In: FPL (2015)
11. Chen, R., Le, H., Prasanna, V.K.: Energy efficient parameterized fft architecture. In: 23rd International Conference on Field programmable Logic and Applications, pp. 1–7. IEEE (2013)
12. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: Selected Areas in Cryptography - SAC (2018)
13. Chiou, D.: The microsoft catapult project. In: IISWC (2017)
14. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**, 297–301 (1965)
15. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. Tech. Rep. MSR-TR-2016-3 (2016)
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012)
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC (2009)
18. Intel: Stratix 10 MX FPGAs. https://www.intel.com/content/www/us/en/products/programmable/sip/stratix-10-mx.html
19. Kim, S., Jung, W., Park, J., Ahn, J.: Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUS. In: IEEE International Symposium on Workload Characterization (IISWC), pp. 264–275. IEEE Computer Society, Los Alamitos (2020)
20. Lee, W.K., Akleylek, S., Yap, W.S., Goi, B.M.: Accelerating number theoretic transform in GPU platform for qTESLA scheme. In: ISPEC (2019)
21. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Cryptology and Network Security (2016)
22. Mert, A.C., Karabulut, E., Öztürk, E., Savaş, E., Becchi, M., Aysu, A.: A flexible and scalable NTT hardware: applications from homomorphically encrypted deep learning to post-quantum cryptography. In: DATE (2020)
23. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**, 519–521 (1985)
24. Nejatollahi, H., Gupta, S., Imani, M., Rosing, T.S., Cammarota, R., Dutt, N.: CryptoPIM: in-memory acceleration for lattice-based cryptographic hardware. In: DAC (2020)
25. Nejatollahi, H., Shahhosseini, S., Cammarota, R., Dutt, N.: Exploring energy efficient quantum-resistant signal processing using array processors. In: ICASSP (2020)
26. Nguyen, D.T., Dang, V.B., Gaj, K.: A high-level synthesis approach to the software/hardware codesign of NTT-based post-quantum cryptography algorithms. In: ICFPT (2019)
27. Putnam, A., et al.: A reconfigurable fabric for accelerating large-scale datacenter services. In: ISCA (2014)

28. Reagen, B., et al.: Cheetah: optimizing and accelerating homomorphic encryption for private inference. In: IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 26–39. IEEE (2020)
29. Riazi, M.S., Laine, K., Pelton, B., Dai, W.: HEAX: an architecture for computing on encrypted data. In: ASPLOS (2020)
30. Seiler, G.: Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. report 2018/039 (2018)
31. Serpanos, D.N., Wolf, T.: Architecture of Network Systems (2011)
32. Sinha Roy, S., Turan, F., Jarvinen, K., Vercauteren, F., Verbauwhede, I.: Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In: HPCA (2019)
33. Ullma, J.D.: Computational Aspects of VLSI (1984)
34. Xilinx: 7 Series FPGAs Data Sheet: Overview. https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
35. Xilinx: Xilinx UltraScale+ HBM FPGAs. https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus-hbm.html
36. Yu, C.L., Kim, J.S., Deng, L., Kestur, S., Narayanan, V., Chakrabarti, C.: FPGA architecture for 2D discrete fourier transform based on 2d decomposition for large-sized data. J. Signal Process. Syst. **64**(1), 109–122 (2011)
37. Zhang, N., Yang, B., Chen, C., Yin, S., Wei, S., Liu, L.: Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT. In: TCHES (2020)