ISSN , Vol. 0, No. 0, pp. 1–16. DOI:

Designing Efficient and Flexible NTT Accelerators

Ahmet MALAL

Middle East Technical University, Ankara, Turkey
ASELSAN, Ankara, Turkey
ahmetmalal@aselsan.com.tr

Abstract. The Number Theoretic Transform (NTT) is a powerful mathematical tool with a wide range of applications in various fields, including signal processing, cryptography, and error correction codes. In recent years, there has been a growing interest in efficiently implementing the NTT on hardware platforms for lattice-based cryptography within the context of NIST's Post-Quantum Cryptography (PQC) competition. The implementation of NTT in cryptography stands as a pivotal advancement, revolutionizing various security protocols. By enabling efficient arithmetic operations in polynomial rings, NTT significantly enhances the speed and security of lattice-based cryptographic schemes, contributing to the development of robust homomorphic encryption, key exchange, and digital signature systems.
This article presents a new implementation of the Number Theoretic Transform for FPGA platforms. The focus of the implementation lies in achieving a flexible trade-off between resource usage and computation speed. By strategically adjusting the allocation of BRAM and DSP resources, the NTT computation can be optimized for either high-speed processing or resource conservation. The proposed implementation is specifically designed for polynomial multiplication with a degree of 256, accommodating coefficients of various bit sizes. Furthermore, the uniform factor graph method was utilized as an alternative to the Cooley-Tukey graph method, resulting in a notable simplification of BRAM addressing procedures. This adaptability renders it suitable for cryptographic algorithms like `CRYSTALS-Dilithium` and `CRYSTALS-Kyber`, which use 256-degree polynomials.

Keywords: Number Theoretic Transform · Post-Quantum Cryptography · Hardware Cryptography · FPGA Implementation · Polynomial Multiplication

1 Introduction

Quantum computers provide a severe threat to the information security industry's fast-evolving landscape. The security of sensitive data and communication is put at risk by the possibility of these computer systems breaking established encryption techniques. The necessity to develop new cryptographic techniques that can prevent both classical and quantum attacks has given rise to the area of post-quantum cryptography with the arrival of practical quantum computers.

The National Institute of Standards and Technology (NIST) has organized a comprehensive standardization process for post-quantum cryptography in response to the escalating significance of quantum-based security threats and their potential to conventional cryptographic systems. Several algorithms are submitted from all over the world. These candidates include various mathematical techniques, such as lattice-based cryptography, code-based cryptography, multivariate polynomial cryptography, and more. Through an inclusive and detailed assessment, NIST seeks to identify cryptographic methods that exhibit robust resistance against potential quantum attacks. After an extensive standardization process,

PQC competition's winners were announced in July 2022. Lattice-based cryptographic algorithms have become strong candidates in post-quantum cryptography competitions because they offer a good balance of security and speed [ADPS16, DLL+17, BDK+18, DKRV18]. `CRYSTALS-Dilithium` was selected as a post-quantum secure digital signature algorithm, while `CRYSTALS-Kyber` was chosen as the public key encryption and key establishment algorithm [DLL+17, BDK+18].

Lattice-based cryptography employs complex mathematical structures known as lattices to build robust security measures for data protection [BBK16]. Within this framework, polynomial multiplication is a fundamental operation that contributes to the effectiveness and efficiency of cryptographic algorithms. These cryptographic methods balance protecting sensitive information and maintaining computational performance by integrating polynomial multiplication with lattice-based techniques [BDK+18, ABB+20]. Homomorphic encryption represents another crucial application for polynomial multiplication. This advanced encryption technique allows for computations to be performed on encrypted data without needing to decrypt it first [GH10, FV12].

The Number Theoretic Transform is a powerful and efficient polynomial multiplication technique. It leverages number theory principles to efficiently compute polynomial products by transforming the polynomial multiplication problem into a problem in the complex number domain. By converting the convolution operation into a point-wise multiplication operation in the transformed domain, NTT significantly reduces the computational complexity, making it an essential tool for applications requiring efficient polynomial multiplication. Its effectiveness in lattice-based cryptography and related fields highlights the critical role NTT plays in enhancing the performance of cryptographic algorithms and mathematical computations.

Given that polynomial multiplication is pivotal in both homomorphic encryption and lattice-based cryptographic schemes, there's a significant interest among researchers in this field. This increase in interest has led to the development of various implementation techniques aimed at achieving optimized solutions on both software and hardware platforms [NDG19, MÖS20, AHY22, CYY+22]. While these advances not only increase the efficiency of polynomial multiplications, they also contribute enormously to the efficiency of cryptographic algorithms [BHK+22].

Our contributions: We have developed a novel and flexible architecture designed specifically for implementing both forward and inverse NTT operation on FPGA platforms. Our focus of the implementation was achieving a flexible trade-off between resource usage and computation speed.

- We introduce a new FPGA implementation architecture for NTT operation. We have developed a *run-time* configurable butterfly unit capable of supporting Cooley-Tukey, Gentleman-Sande, and pointwise multiplication operations simultaneously. This design feature provides us the flexibility to use the same butterfly unit for both forward and inverse NTT computations, as well as point-wise multiplications during *run-time*.

- The number of butterfly units used in the design is configurable and can be set at *compile-time*. This parameter directly impacts the utilization of Block RAM (BRAM) and Digital Signal Processors (DSPs) within the design. Consequently, depending on specific requirements, it can be configured for either high-speed computation or resource conservative design. Using more butterfly units results in enhanced processing speed but costs a higher consumption of BRAM and DSP resources.

- We have used a *run-time* configurable Montgomery modular multiplier, capable of effectively multiplying coefficients of polynomials in R up to 31-bit in the butterfly units. Instead of using Cooley-Tukey or Gentleman-Sande methods, we preferred to use constant-geometry [Pea68] method for easy to address BRAMs. Our design

offers an adaptable and generic form for performing NTT operations on polynomials with a degree of 256, supporting coefficients up-to 31 bits.

Structure of the paper: This paper is structured as follows: Section 2 provides a brief overview of the PQC (Post-Quantum Cryptography) competition, lattice problems, and selected algorithms. In Section 3 our work is described in depth by providing a comprehensive description of the processes and techniques. Section 4 presents our FPGA results and compares them with those of other researchers. Finally in section 5, we conclude our study and discuss future work.

2 Background

In this section, we provide introductory information about the PQC competition, focusing on well-known lattice-based problems, and the winners of these competitions, namely `CRYSTALS-Kyber` and `CRYSTALS-Dilithium`. Then, we will emphasize the significance of the NTT structure in these contexts.

2.1 PQC Competition

The rapid enhancement of quantum computing technologies has raised significant security concerns for widely-used public key cryptosystems, such as `RSA` and `ECDSA`, which are vulnerable to quantum attacks. To prevent these emerging threats, NIST initiated a competition in 2016. This global challenge aimed to standardize quantum-secure algorithms for digital signature algorithm and key encapsulation mechanisms (KEM). A wide variety of applications were submitted to the competition with 69 different algorithms from all over the world. These entries exhibited a variety of quantum-secure structures, including lattice-based, code-based, multivariate-based, hash-based, and isogeny-based cryptosystems.

Following three rounds of evaluation and elimination, the competition led to significant developments in the field of quantum-secure cryptography. On July 5, 2022, NIST announced the standardization of `CRYSTALS-Kyber`, a lattice-based algorithm, as the chosen Key Encapsulation Mechanism (KEM) algorithm [BDK+18]. Additionally, `CRYSTALS-Dilithium`, `Falcon`, and `SPHINCS+` were declared as the winners for standard digital signature algorithm [DLL+17].

Except for `SPHINCS+`, which is a hash-based algorithm, all other algorithms selected by NIST were of the lattice-based. The result of this competition clearly showed that lattice-based systems were the more preferred systems.

2.2 Lattice Problems

Lattice-based cryptography is an area of study in cryptographic research that builds security primitives upon the hardness of computational problems in lattices. These problems, often related to the difficulty of finding the shortest vector in a lattice or its closest variant, have been shown to be resistant against quantum attacks, made lattice-based schemes potential candidates for post-quantum cryptography. Both `CRYSTALS-Dilithium` and `CRYSTALS-Kyber`, the winners of PQC competition, are based on computational problems in lattices [BDK+18], [DLL+17].

Definition 1 (Learning with errors (LWE) [Reg05])**.** The problem is stated in 2005 by Regev. Given a polynomial number of samples in the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$, it becomes computationally infeasible to find the secret vector $\mathbf{s} \in \mathbb{Z}_q^n$. In this formulation, the vector $\mathbf{a} \in \mathbb{Z}_q^n$ is selected uniformly at random, and the error term e is taken from a chosen error distribution ϕ.

Cryptosystems based on the LWE [LPR13] problem often require extensive matrix operations, leading to high computational costs and larger key sizes. To decrease larger key sizes, Ring-LWE [LS15] and Module-LWE are introduced.

Definition 2 (Ring-LWE)**.** The Ring-LWE problem is proposed in 2013 by Lyubashevsky. Let us define $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ as the ring of polynomials, with n being a power of 2. The Ring-LWE problem is formulated as follows: given pairs of the form $(a, a \cdot s + e)$, it is computationally infeasible to recover the secret polynomial $s \in R_q$. Here, the polynomial $a \in R_q$ is chosen uniformly at random, and the error polynomial e's coefficients are small, comes from the error distribution ϕ.

Definition 3 (Module-LWE)**.** The Module-LWE problem is introduced in 2015. The Module-LWE problem is characterized by the difficulty in deducing a secret vector $s \in R_q^k$ given samples in the form $(\mathbf{a}, \mathbf{a}^T\mathbf{s} + \mathbf{e})$. In this scenario, the vector $\mathbf{a} \in R_q^k$ is selected uniformly at random from the ring R_q^k, and the error polynomial $\mathbf{e}$ is composed of coefficients that are small, derived from the error distribution ϕ. This formulation of the Module-LWE problem highlights the challenge in recovering the secret vector $\mathbf{s}$ under these conditions.

Both `CRYSTALS-Dilithium` and `CRYSTALS-Kyber` are algorithms that are based on the Module-LWE problem.

2.3 `CRYSTALS-Dilithium`

The `CRYSTALS-Dilithium` algorithm has been selected by NIST as a standard quantum-secure digital signature algorithm, with the announcement made in July 2022. The pseudocode template of the algorithm is presented in Algorithm 1. As previously mentioned, this algorithm is based on the Module-Learning with Errors problem. Dilithium operates within the ring $\mathbb{Z}_q[X]/(X^{256} + 1)$, where q is set to $2^{23} - 2^{13} + 1$, which is 8380417.

The polynomials utilized in the algorithm consist of 256 coefficients, each belonging to $\mathbb{Z}_q$. Each coefficient can be accommodated within a 23-bit register. Consequently, each polynomial can be represented as a product of $256 \times 23 = 5888$ bits, which is equivalent to 736 bytes.

`CRYSTALS-Dilithium` algorithm consists of extensive matrix multiplications. For instance, the multiplication As_1, where A is a matrix in $R_q^{k \times l}$, and s_1 is a vector in S_η^l. The values of k and l vary according to the desired security level.

$$(k, l) = \left\{ \begin{array}{ll} (4, 4), & \text{if security level} = 1 \\ (6, 5), & \text{if security level} = 3 \\ (8, 7), & \text{if security level} = 5 \end{array} \right\}$$

Assuming we are operating at security level 5, A contains $8x7 = 56$ polynomials, each with 256 coefficients. These coefficients are elements of $\mathbb{Z}_q$. The multiplication of A and s_1 comes with a huge computational cost.

Let A be a $k \times l = (8, 7)$ matrix and s_1 be a vector of length $l = 7$ in S_η. The matrix multiplication As_1 is given by:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1l} \\ a_{21} & a_{22} & \cdots & a_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kl} \end{pmatrix}, \quad s_1 = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_l \end{pmatrix}$$

The result of the multiplication $A \cdot s_1$ is:

$$t = A \cdot s_1 = \begin{pmatrix} a_{11} \cdot s_1 + a_{12} \cdot s_2 + \cdots + a_{17} \cdot s_7 \\ a_{21} \cdot s_1 + a_{22} \cdot s_2 + \cdots + a_{27} \cdot s_7 \\ \vdots \\ a_{81} \cdot s_1 + a_{82} \cdot s_2 + \cdots + a_{87} \cdot s_7 \end{pmatrix}$$

There are 56 distinct polynomial multiplication calculations required to determine the value of t. As can be observed, the algorithm involves numerous other multiplication operations as well, many of which are highlighted in the pseudo-code 1.

Both the speed of the algorithm and the resource utilization are completely influenced by polynomial multiplications. Consequently, accelerating the multiplication operation plays a critical role in enhancing the overall efficiency of the algorithm.

2.4 `CRYSTALS-Kyber`

The `CRYSTALS-Kyber` algorithm has been selected by NIST as a standard quantum-secure public-key encryption and key-establishment algorithm, with the announcement made in July 2022. The pseudocode template of key generation, encryption and decryption of the algorithm is presented in Algorithm 2. As previously mentioned, this algorithm is

Algorithm 1 Template for `CRYSTALS-Dilithium` Digital Signature Algorithm [DLL+17]

function GEN
  $A \leftarrow R_q^{k \times l}$
  $(s_1, s_2) \leftarrow S_\eta^l \times S_\eta^k$
  $t := As_1 + s_2$ ▷ **NTT Multiplication**
  return $(pk = (A, t), sk = (A, t, s_1, s_2))$
end function

function SIGN(sk, M)
  $z := \perp$
  while $z = \perp$ **do**
    $y \leftarrow S_{\gamma_1 - 1}^l$
    $w_1 := \text{HighBits}(Ay, 2\gamma_2)$ ▷ **NTT Multiplication**
    $c \in B_{60} := H(M \| w_1)$
    $z := y + cs_1$ ▷ **NTT Multiplication**
    if $\|z\|_\infty \geq \gamma_1 - \beta$ **or**
      $\|\text{LowBits}(Ay - cs_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$ **then** ▷ **NTT Multiplication**
        $z := \perp$
    end if
  end while
  return $\sigma = (z, c)$
end function

function VERIFY$(pk, M, \sigma = (z, c))$
  $w_1' := \text{HighBits}(Az - ct, 2\gamma_2)$ ▷ **NTT Multiplication**
  if $\|z\|_\infty < \gamma_1 - \beta$ **and** $c = H(M \| w_1')$ **then**
    return *True*
  else
    return *False*
  end if
end function

also based on the Module-Learning with Errors problem, just like `CRYSTALS-Dilithium`. `CRYSTALS-Kyber` operates within the ring $\mathbb{Z}_q[X]/(X^{256}+1)$, where q is set to 3329.

The polynomials utilized in the algorithm consist of 256 coefficients, each belonging to $\mathbb{Z}_q$. Each coefficient can be accommodated within a 12-bit register. Consequently, each polynomial can be represented as a product of $256 \times 12 = 3072$ bits, which is equivalent to 384 bytes.

Algorithm 2 Template for `CRYSTALS-Kyber` Key Encapsulation Mechanism [BDK+18]

function KYBER.CPA.KEYGEN
  $\rho, \sigma \leftarrow \{0,1\}^{256}$
  $A \sim R_q^{k \times k} := \mathrm{Sam}(\rho)$
  $(s, e) \sim \beta_\eta^k \times \beta_\eta^k := \mathrm{Sam}(\sigma)$
  $t := \mathrm{Compress}_q(\ \mathrm{A}\ \mathrm{s}\ + e, d_t)$ ▷ **NTT Multiplication**
  $pk := (t, \rho), sk := s$
end function

function KYBER.CPA.ENC($pk := (t, \rho), m \in \mathcal{M}$)
  $r \leftarrow \{0,1\}^{256}$
  $t := \mathrm{Decompress}_q(t, d_t)$
  $A \sim R_q^{k \times k} := \mathrm{Sam}(\rho)$
  $(r, e_1, e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \mathrm{Sam}(r)$
  $u := \mathrm{Compress}_q(\ A^T r\ + e_1, d_u)$ ▷ **NTT Multiplication**
  $v := \mathrm{Compress}_{q_2}(\ t^T r\ + e_2 + \frac{q}{2} \cdot m, d_v)$ ▷ **NTT Multiplication**
  return $c := (u, v)$
end function

function KYBER.CPA.DEC($sk := s, c = (u, v)$)
  $u := \mathrm{Decompress}_q(u, d_u)$
  $v := \mathrm{Decompress}_q(v, d_v)$
  return $\mathrm{Compress}_q(v - \ s^T u\ , 1)$ ▷ **NTT Multiplication**
end function

As highlighted, `CRYSTALS-Kyber` algorithm also consists of huge matrix multiplications, which both time and resource consuming.

2.5 NTT and NTT-parameters of Winners

Similar to the Fast Fourier Transform (FFT), NTT operates on sequences of numbers but focuses on modular arithmetic within a given number system. Let p be a prime number, and ω be a primitive n-th root of unity modulo p, where n is the length of the input sequence. The NTT converts a sequence $\{a_0, a_1, \ldots, a_{n-1}\}$ into another sequence $\{A_0, A_1, \ldots, A_{n-1}\}$, where

$$A_k = \sum_{j=0}^{n-1} a_j \cdot \omega^{jk} \pmod{p}$$

This transformation enables efficient operations in the frequency domain, allowing for the analysis and manipulation of data in a different representation. Lets assume that two polynomials a $= \{a_0, a_1, \ldots, a_{n-1}\}$ and b $= \{b_0, b_1, \ldots, b_{n-1}\}$ will be multiplied. The time complexity of polynomial multiplication is $\mathcal{O}(n^2)$ if schoolbook technique is used. Utilizing NTT technique necessitates the execution of the forward NTT operation on polynomial a,

followed by its application on polynomial b. Subsequently, a point-wise multiplication is performed on the coefficients of polynomials a and b. Upon completion of these operations, the inverse NTT is employed. Consequently, the polynomial multiplication of a and b is effectively computed. NTT provides $\mathcal{O}(n \log n)$ time complexity, which is much more efficient than schoolbook technique.

Polynomial multiplication is generally based on two different parameters:

- n: the degree of the polynomial ring
- k: the bit length of the coefficient modulus

While homomorphic encryption schemes use large n and k parameters for polynomial multiplication, ranging from n = 1024 to n = 32768 and k = 14 to 60. Dilithium uses *(n,k)* = (256,23); Kyber uses *(n,k)* = (256,12).

Post-quantum cryptographic algorithms and homomorphic encryption schemes use NTT operation for the algorithm's efficiency. Therefore, several NTT implementations are designed and proposed with different implementation concerns for several platforms. *Run-time* reconfigurable NTT multiplication is proposed with supporting six different parameter sets [MÖS20] for FPGA platforms. Instead of applying the Cooley-Tukey algorithm, the constant-geometry-based NTT operation is implemented for ASIC platforms [BUC19]. There also several optimized NTT implementations for software [AHY22] and GPU [OEM+21] platforms.

3 The Proposed Architecture for NTT

In this section, we provide a detailed exploration of our design, focusing on its key components and overall architecture. We begin with Montgomery multiplication. Following this, we delve into the internal structure of the butterfly unit and our proposed NTT architecture.

3.1 Montgomery Multiplication

Montgomery multiplication is a technique commonly used in modular arithmetic and cryptographic operations, particularly in modular exponentiation and elliptic curve cryptography. It aims to accelerate modular multiplications by reducing the number of expensive modular divisions. While several modular multiplication algorithms exist, including the Barrett and Karatsuba reduction methods, we selected the Montgomery algorithm due to its fulfillment with our design criteria. In Montgomery multiplication, the input operands are first transformed into a new representation known as the Montgomery domain, which simplifies the modular reduction step. The core of the algorithm involves a series of additions and bit-wise operations.

Since we work with signals in the FPGA, we can work on numbers of any bit length we want. While each coefficient is stored in a 32-bit register for Dilithium parameters in the software, we used 23-bit registers for FPGA. This choice allowed us to set the Montgomery constant as 2^{23} instead of 2^{32}. As a result, when we transform a polynomial into the NTT domain, we obtain results that differ from the reference C-Code provided by the Dilithium designers [DLL+17]. However, when we perform the complete multiplication, we achieve the same result.

3.2 Proposed Butterfly Unit

The butterfly unit is a fundamental building block in the NTT operation. It performs the essential complex multiplication and addition operations that contribute to the transformation of input data into NTT-domain coefficients. We have designed a pipelined-butterfly

unit such that it can be used both in forward and inverse NTT operation. Additionally, our proposed butterfly unit provides pointwise multiplication. Figure 1 shows the internal structure of the proposed butterfly unit. It receives four input parameters, denoted as a, b, w, $mode$, and produces two outputs, a' and b'. a and b are the coefficients of the polynomial, where w is the corresponding twiddle factor. $mode$ is used for encoding behaviour of the module. It is 2-bit, where "00" denotes forward NTT, "01" denotes inverse NTT, and "10" denotes point-wise multiplication. We used 7-stage pipeline delay in butterfly units to increase the clock frequency.

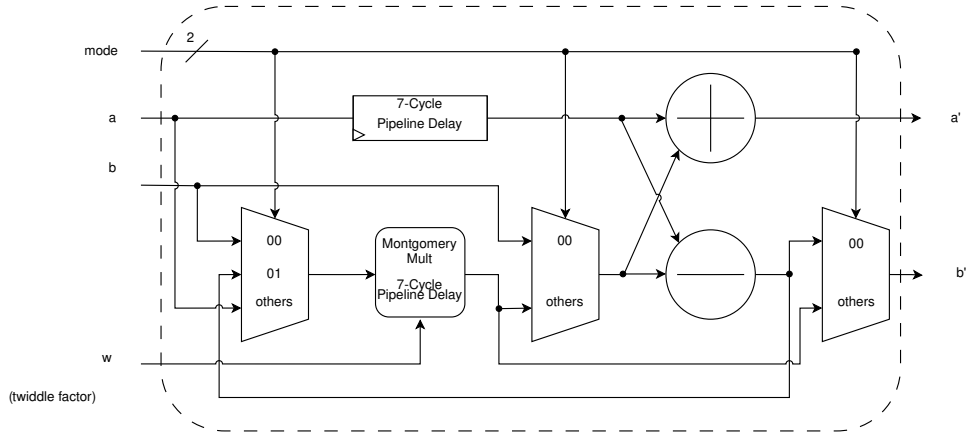

Figure 1: The Proposed Butterfly Unit

3.3 Cooley-Tukey Algorithm

The Cooley-Tukey algorithm is a widely used method for efficiently computing the NTT and its inverse NTT. It employs a divide-and-conquer approach by recursively breaking down a NTT of any composite size $N = N_1 \times N_2$ into smaller NTTs of sizes N_1 and N_2. This iterative process continues until the base case of $N = 2$ is reached, at which point the NTT computation can be efficiently carried out using butterfly operations. These butterfly operations involve combining the results of the smaller NTTs to produce the final NTT result. The Cooley-Tukey algorithm significantly reduces the number of complex multiplications required compared to the straightforward computation of the NTT. Figure 2 shows the flow of the algorithm for a polynomial of degree 8.

3.4 Constant-Geometry based NTT

Instead of using the Cooley-Tukey method, we used constant-geometry based graph to perform NTT operation. The Cooley-Tukey method is a good practice for software platforms since it has a recursive design. However, implementing recursive methods commonly brings more cost to hardware platforms. In every stage, the butterfly unit input comes from different addresses of BRAM. We would need to construct an address arranger to rearrange the addresses, which also comes with resource costs. However, every stage rearrangements are the same in a constant-geometry based graph [Pea68]. It also provides flexibility on number of butterfly units. Based on our choice of the number of butterfly units, the number of BRAMs increases correspondingly; however, this accelerates NTT operation. Figure 3 shows the flow of the algorithm for a polynomial of degree 8.

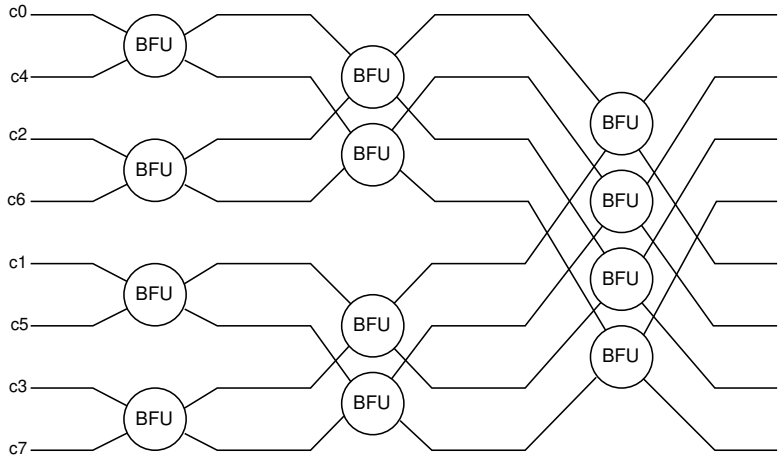

Figure 2: Cooley-Tukey Graph based NTT operation for a polynomial degree 8.

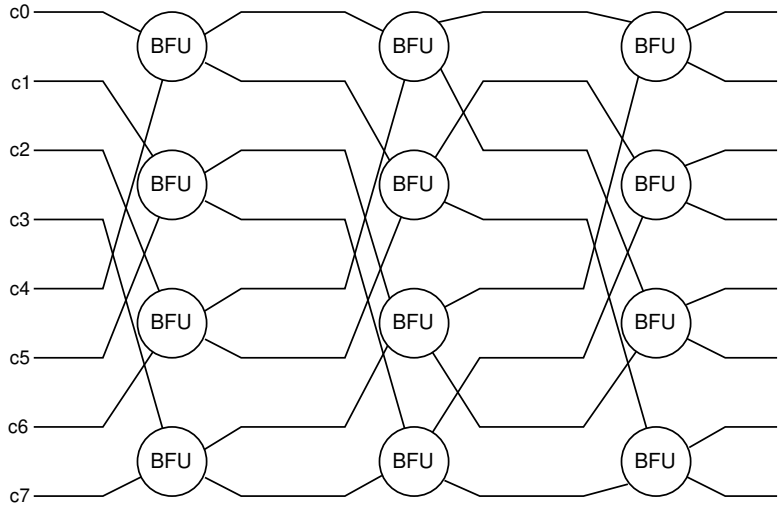

Figure 3: Constant-Geometry based NTT operation for a polynomial degree 8.

3.5 NTT Memory Architecture

To make the best use of our butterfly units, we had to increase the number of Block RAMs (BRAMs) by two times. We used true-dual port BRAMs for this expansion. These BRAMs allow us to access two different addresses at the same time in just one cycle. We planned this setup so that the coefficients of each polynomial are evenly spread out over the first half of the BRAMs. In every computational cycle, we take one coefficient from the top part of a BRAM and another from the middle. These two coefficients are then sent to their own butterfly units for processing. After that, the outputs from the butterfly units are directly saved into the second half of the BRAMs.

We have developed pipelined butterfly units to enhance performance. The latency of each butterfly unit is a 7-cycle. Additionally, reading and writing to the corresponding BRAM incurs an extra 4-cycle delay. Thus, in each stage, the total cost of utilizing a butterfly unit is 11-cycle. Overall cycle count for one stage depends on the number of butterfly units used in the design. For polynomials with 256 coefficients, there are 8 stages. Additionally, there is a 3-cycle overhead for initializing and finalizing the NTT operation.

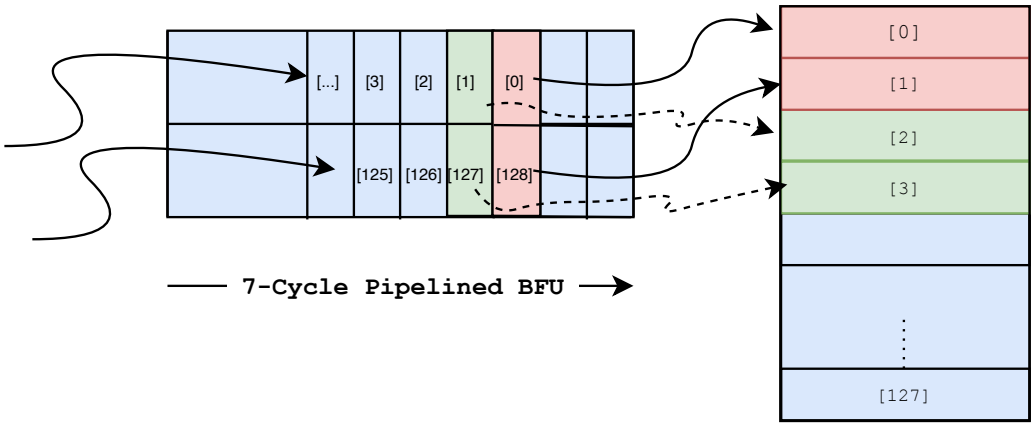

Figure 4: The internal process of butterfly units, where number of butterfly unit (η) is 2.

The derived formula for forward NTT is as follows:

$$8 \cdot \left(\frac{256}{2 \cdot \#BFU} + 11 \right) + 3 = \frac{1024}{\#BFU} + 91 \text{ cycles.}$$

In the inverse NTT process, we need to do pointwise multiplication at the end of the 8th stage. The number of cycles this multiplication takes depends on how many butterfly units (BFUs) we use. Our butterfly units are designed to handle pointwise multiplication along with forward and inverse modular operations efficiently. This means that the cycles needed for multiplying each coefficient is calculated as $\frac{256}{\#\text{BFU}}$. 11-cycle come from butterfly unit latency. There's also an extra 3-cycle needed at the beginning and end of the inverse NTT operation for start and exit states.

The formula we use for inverse NTT is as follows:

$$\begin{aligned} 8 \cdot \left(\frac{256}{2 \cdot \#BFU} + 11 \right) + \left(\frac{256}{\#BFU} \right) + 11 + 3 &= \frac{1024}{\#BFU} + \frac{256}{\#BFU} + 102 \\ &= \frac{1280}{\#BFU} + 102 \text{ cycles.} \end{aligned}$$

Algorithm 3 presents the pseudo-code we developed. To explain in more detailed, we have shown scenarios when two and four butterfly units are used. Our design supports any number of butterfly units power of two.

3.5.1 2-Butterfly Unit Scenario

In Figure 5, we illustrate the internal structure of the NTT operation, stage by stage. Consider the case where we perform the forward NTT operation on a polynomial with 256 coefficients, using 2 butterfly units (#BFU). The first 128 coefficients are stored in the first BRAM, and the next 128 in the second BRAM. The first BFU takes its inputs from the first elements of the first and second BRAMs. The second BFU's inputs are the 64^{th} elements of these BRAMs. This pattern is shown in detail in Figure 5. The results from the BFUs are then sent to the corresponding BRAMs. Specifically, the first BFU sends its results to the third BRAM, and the second BFU's outputs go to the fourth BRAM. In subsequent stages, the BFU inputs are taken from the third and fourth BRAMs, alternating in each stage. After eight stages, the forward NTT operation on the polynomial is complete. As depicted, only four BRAMs are utilized for storing coefficients, with an additional BRAM reserved for the twiddle factors.

Algorithm 3 Constant Geometry Based NTT Operation Algorithm

Require: Polynomial $a(x) \in \mathbb{R}_q$, η number of used butterfly unit and n-th primitive root of unity $\omega_n \in \mathbb{Z}_q$
Ensure: Polynomial $a'(x) \in \mathbb{R}_q$ such that $a'(x) = NTT(a(x))$

$x \leftarrow 0$
for i `in 0 to` $\eta - 1$ **do**
  for j `in 0 to` $256/\eta - 1$ **do**
    $RAM[i][j] \leftarrow a_{i*(256/\eta)+j}$ ▷ Fills the RAMs with coefficients
    $RAM[i+\eta][j] \leftarrow 0$ ▷ Generate other RAMs, Overall we need 2η RAMs
  end for
end for
for i `in 0 to 511` **do**
  $W[i] \leftarrow \omega_n^i 2^{\lceil log_2 q \rceil} \bmod q$ ▷ Calculates twiddle factors
end for ▷ NTT operation starts
for s `in 0 to 7` **do** ▷ Keeps the stage counter
  for j `in 0 to` $(256/2\eta) - 1$ **do**
    for i `in 0 to` $(\eta/2) - 1$ **do**
      $\omega_0 \leftarrow W[2^{7-s}(1 + 2(bitRev((i * 256/\eta) \bmod 2^s, s)))]$
      $\omega_1 \leftarrow W[2^{7-s}(1 + 2(bitRev((i * 256/\eta + 256/(2\eta)) \bmod 2^s, s)))]$
      $a_0 \leftarrow RAM[i][j]$, $b_0 \leftarrow RAM[i + \eta/2][j]$
      $a_1 \leftarrow RAM[i][j + 256/\eta]$, $b_1 \leftarrow RAM[i + \eta/2][j]$
      $c_0, d_0 \leftarrow BFU(a_0, b_0, w_0)$, $c_1, d_1 \leftarrow BFU(a_1, b_1, w_1)$ ▷ Butterfly Operation
      $RAM[2i + \eta][2x] \leftarrow c_0$
      $RAM[2i + \eta][2x + 1] \leftarrow d_0$
      $RAM[2i + 1 + \eta][2x] \leftarrow c_1$
      $RAM[2i + 1 + \eta][2x + 1] \leftarrow d_1$
      $x \leftarrow x + 1$
    end for
  end for
  for i `in 0 to` $\eta/2 - 1$ **do**
    $RAM[i] \leftarrow RAM[i + \eta]$
  end for
end for
for i `in 0 to` $\eta - 1$ **do**
  for j `in 0 to` $256/\eta - 1$ **do**
    $a'_{i*(256/\eta)+j} \leftarrow RAM[i][j]$ ▷ Constructs the output polynomial
  end for
end for

3.5.2 4-Butterfly Unit Scenario

In Figure 6, we showed the internal structure of the NTT operation stage by stage. Let us consider the scenario where the forward NTT operation of a polynomial comprising 256 coefficients is undertaken, employing a designated butterfly unit count (#BFU) of 4. The initial 64 coefficients will be allocated to the first BRAM, followed by the subsequent 64 coefficients to the second BRAM, and so forth, maintaining this pattern. The input of the first BFU is the first element of the first BRAM and the first element of the third BRAM. The inputs of the second BFU will be the 32^{nd} elements of the first BRAM and 32^{nd} elements of the third BRAM. The pattern is detailed in Figure 6. The results of BFUs will be directed to the corresponding BRAMs. For the first BFU, the results will be directed to the fifth BRAM. The output of the second BFU will be moved to the sixth BRAM. In the next stage, the inputs of the BFUs will come from the fifth, sixth, seventh,

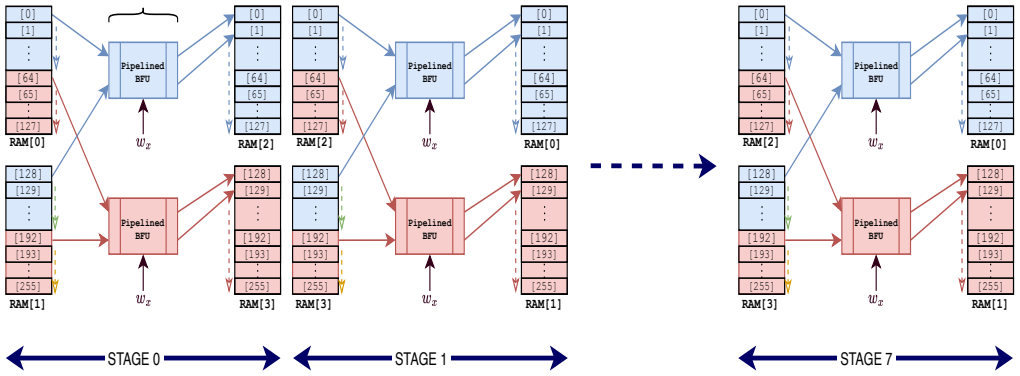

Figure 5: Internal Structure of NTT Operation, where number of butterfly unit (η) is 2.

and eighth BRAMs. The BRAMs will be used vice-versa in each stage. After eight stages, the forward NTT operation of the given polynomial will be computed.

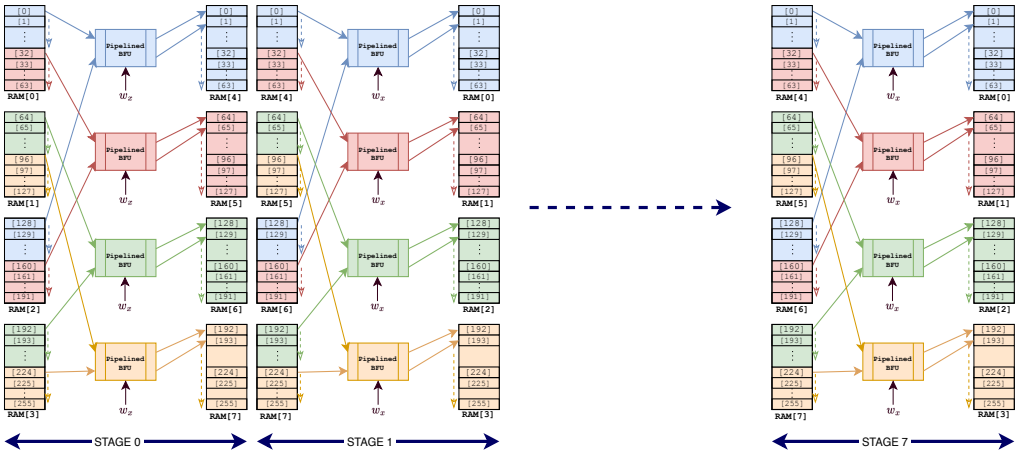

Figure 6: Internal Structure of NTT Operation, where number of butterfly unit (η) is 4.

4 Results

This section presents the results of our proposed memory-speed trade-off implementation of NTT operation within the `CRYSTALS-Dilithium` algorithm. Our approach is distinct in its ability to be used according to specific requirements, balancing between memory efficiency and fast performance. While our implementation may not claim the title of being the most compact or the fastest in absolute terms, its true strength lies in its versatile configurability. This adaptability allows us to optimize the design based on the specific needs of the application, whether prioritizing a smaller memory footprint or seeking enhanced processing speed.

Our results showcase that, in certain configurations, our design achieves greater compactness compared to some existing implementations, while in other setups, it outperforms competitors in speed. Our implementation holds its ground when compared with other research in the field, demonstrating comparable, if not superior, performance in various aspects. The comparative analysis presented here not only underscores the relative strengths of our design but also its potential as a flexible solution adaptable to a wide range of

cryptographic applications.By striking a balance between compactness and speed, our design becomes a crucial design in the FPGA-based NTT operation.

Table 1: Performance Results of The Proposed Design for Forward NTT Operation, where (n,k) = (256,23) on Virtex UltraScale+

#BFU	LUT	FF	BRAM	DSP	Cycles
2	1601	699	5	10	512+91 = 603
4	3080	1279	10	20	256+91 = 347
8	5496	2463	20	40	128+91 = 219
16	11558	4912	40	80	64+91 = 155
:	:	:	:	:	:

Table 1 and Table 2 provide a comprehensive overview of the outcomes obtained from our implementations. In Table 1, we present the results of the forward NTT implementation, focusing on memory consumption, execution time, and achieved throughput. Our proposed trade-off mechanism allows for a fine adjustment between memory utilization and processing speed, catering to diverse application requirements. The variation in memory-speed trade-off is systematically explored, highlighting the flexibility of our approach. Additionally, Table 2 illustrates the outcomes of the inverse NTT implementation, mirroring a similar analysis with a specific emphasis on the trade-off between memory utilization and computational efficiency.

Table 2: Performance Results of The Proposed Design for Inverse NTT Operation, where (n,k) = (256,23) on Virtex UltraScale+

#BFU	LUT	FF	BRAM	DSP	Cycles
2	2405	794	4	10	512+128+102=742
4	4734	1462	8	20	256+64+102=422
8	7864	2796	16	40	128+32+102=262
16	15885	5559	32	80	64+16+102=182
:	:	:	:	:	:

Table 3: Comparative FPGA Resource and Performance Table, for Dilithium NTT parameters (n,k) = (256,23)

Design	LUT	FF	BRAM	DSP	Cycles	Frequency(MHz)	Platform
[ZZW+22]	1919	1301	2	8	296	96.9	Artix-7
[MCL+22]	799	328	4.5	2	1405	172	Artix-7
[ZHL+21]	2044	N/A	16	N/A	1170	216	Artix-7
[LSG21]	524	759	17	1	533	311	Virtex-7
[NDG19]	1899	2041	2	8	294	445	Zynq UltraScale+
[RMJ+21]	1798	2532	3.5	48	502	637	Virtex UltraScale+
[BNG21]	4509	3146	16	0	300	N/A	Virtex UltraScale+
This Work,**Our Smallest**	**1601**	699	5	10	603	142.8	Virtex UltraScale+
This Work,**Our Fastest**	11558	4912	40	80	**155**	142.8	Virtex UltraScale+

To compare the effectiveness of our proposed approach, we provide a comparative analysis in Table 3. This table compares our results with other prominent researchers in the field. By doing so, we gain valuable insights into the performance benchmarks of existing methodologies. The comparison covers aspects such as memory consumption, execution time, and balancing strategies, highlighting the improvements achieved by our proposed memory speed balancing implementation.

5 Conclusion and Future Work

Overall, the results presented in this section underscore the significance of our contribution. Our memory-speed trade-off implementation of the NTT operation offers a customizable balance between resource utilization and computational swiftness. By configuring the number of butterfly units used in the design, we generated various of implementations. We compared our smallest and fastest results with the results of other researchers. The results we obtained show that our flexible design is comparable to other results. The comparative analysis shows our approach's competitiveness within the landscape of lattice-based cryptography, aligning with the objectives of the NIST PQC initiative. These findings collectively emphasize the potential of our methodology to pave the way for efficient and robust NTT solution.

As part of future work, we aim to expand the scope of our research in several directions. Our NTT implementation is designed to operate on polynomials with $n = 256$, supporting coefficients of up to 31-bit. In the context of this study, performance metrics were obtained for only Dilithium $(n, k) = (256, 23)$ parameters. First, we plan to obtain performance results for a wider range of NTT parameters, including those relevant to Kyber and other cryptographic algorithms. Additionally, we intend to delve deeper into the optimization of the Montgomery multiplier to enhance its frequency performance. Finally, for the purpose of comprehensive comparison and evaluation, we will consider performing our NTT implementation on different FPGA platforms. These future efforts will not only contribute to a more comprehensive understanding of the algorithm's flexibility and efficiency but also pave the way for its broader applicability in various computational contexts.

6 Acknowledgement

The author would like to thank Erdem Alkım for the meaningful discussion and his comments during the preparation of this study.

References

[ABB+20] Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Juliane Krämer, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qtesla. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020, Rome, Italy, October 19-22, 2020, Proceedings, Part I*, volume 12146 of *Lecture Notes in Computer Science*, pages 441–460. Springer, 2020.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016.

[AHY22] Erdem Alkim, Vincent Hwang, and Bo-Yin Yang. Multi-parameter support with ntts for ntru and ntru prime on cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):349–371, Aug. 2022.

[BBK16] Nina Bindel, Johannes Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. Cryptology ePrint Archive, Paper 2016/415, 2016.

[BDK+18] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367, 2018.

[BHK+22] Hanno Becker, Vincent Hwang, Matthias J. Kannwischer, Bo-Yin Yang, and Shang-Yi Yang. Neon NTT: faster dilithium, kyber, and saber on cortex-a72 and apple M1. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):221–244, 2022.

[BNG21] Luke Beckwith, Duc Tri Nguyen, and Kris Gaj. High-performance hardware implementation of crystals-dilithium. In *International Conference on Field-Programmable Technology, (IC)FPT 2021, Auckland, New Zealand, December 6-10, 2021*, pages 1–10. IEEE, 2021.

[BUC19] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):17–61, 2019.

[CYY+22] Xiangren Chen, Bohan Yang, Shouyi Yin, Shaojun Wei, and Leibo Liu. CFNTT: scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):94–126, 2022.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*, volume 10831 of *Lecture Notes in Computer Science*, pages 282–305. Springer, 2018.

[DLL+17] Leo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals – dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Paper 2017/633, 2017.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[GH10] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. *IACR Cryptol. ePrint Arch.*, page 520, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

[LSG21] Georg Land, Pascal Sasdrich, and Tim Güneysu. A hard crystal - implementing dilithium on reconfigurable hardware. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 210–230. Springer, 2021.

[MCL+22] Gaoyu MAO, Donglong CHEN, Guangyan LI, Wangchen DAI, Abdurrashid Ibrahim SANKA, Çetin Kaya KOÇ, and Ray C. C." CHEUNG. High-performance and configurable sw/hw co-design of post-quantum signature crystals-dilithium. *ACM Transactions on Reconfigurable Technology and Systems*, 16(3), oct 2022. Research Unit(s) information for this publication is provided by the author(s) concerned.

[MÖS20] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savas. FPGA implementation of a run-time configurable ntt-based polynomial multiplication hardware. *Microprocess. Microsystems*, 78:103219, 2020.

[NDG19] Duc Tri Nguyen, Viet Ba Dang, and Kris Gaj. A high-level synthesis approach to the software/hardware codesign of ntt-based post-quantum cryptography algorithms. In *International Conference on Field-Programmable Technology, FPT 2019, Tianjin, China, December 9-13, 2019*, pages 371–374. IEEE, 2019.

[OEM+21] Ozgun Ozerk, Can Elgezen, Ahmet Can Mert, Erdinc Ozturk, and Erkay Savas. Efficient number theoretic transform implementation on gpu for homomorphic encryption. Cryptology ePrint Archive, Paper 2021/124, 2021.

[Pea68] Marshall C. Pease. An adaptation of the fast fourier transform for parallel processing. *J. ACM*, 15(2):252–264, apr 1968.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[RMJ+21] Sara Ricci, Lukas Malina, Petr Jedlicka, David Smékal, Jan Hajny, Peter Cíbik, Petr Dzurenda, and Patrik Dobias. Implementing crystals-dilithium signature scheme on fpgas. In Delphine Reinhardt and Tilo Müller, editors, *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, pages 1:1–1:11. ACM, 2021.

[ZHL+21] Zhen Zhou, Debiao He, Zhe Liu, Min Luo, and Kim-Kwang Raymond Choo. A software/hardware co-design of crystals-dilithium signature scheme. *ACM Trans. Reconfigurable Technol. Syst.*, 14(2):11:1–11:21, 2021.

[ZZW+22] Cankun Zhao, Neng Zhang, Hanning Wang, Bohan Yang, Wenping Zhu, Zhengdong Li, Min Zhu, Shouyi Yin, Shaojun Wei, and Leibo Liu. A compact and high-performance hardware architecture for crystals-dilithium. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):270–295, 2022.