Systolic Array Architectures for Number Theoretic Transform (NTT)

Nedal Abu Haltam

May 26, 2025

Abstract

This work explores hardware-accelerated implementations of the Number Theoretic Transform (NTT) using systolic array architectures. NTT is pivotal in lattice-based cryptographic protocols, offering exact integer arithmetic essential for secure and efficient post-quantum cryptographic systems. We investigate 5 1D systolic array designs and single 2D sytolic array design that leverage the structured and repetitive nature of NTT computations to enhance throughput and scalability. By analyzing various implementation strategies and their computational characteristics, this study aims to contribute toward high-performance and hardware-efficient NTT modules suitable for FPGA and ASIC platforms.

1 Introduction

The Number Theoretic Transform (NTT) has emerged as a key computational kernel in modern cryptographic algorithms, particularly in lattice-based cryptography. With the growing demand for post-quantum secure systems, such as Kyber and Dilithium, efficient and reliable computation of the NTT has become critical. Unlike the Fast Fourier Transform (FFT), which uses floating-point arithmetic and is prone to precision errors, NTT operates entirely in the integer domain, offering exact results—a vital requirement in cryptographic computations.
As cryptographic workloads are increasingly deployed on embedded and resource-constrained hardware, accelerating NTT using specialized hardware becomes essential. This project investigates the use of 1D systolic array architectures to perform NTT operations efficiently. Systolic arrays are particularly attractive for such tasks due to their regular structure, local communication, and high parallelism, making them well-suited for FPGA and ASIC implementations.

2 Background: Number Theoretic Transform (NTT)

The NTT is a discrete transform analogous to the Discrete Fourier Transform (DFT), but it operates over a finite field defined by a prime modulus p. It transforms an input vector $a = [a_0, a_1, \ldots, a_{n-1}]$ into an output vector $A = [A_0, A_1, \ldots, A_{n-1}]$, where each output is computed as:

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

Here, ω is a primitive n-th root of unity in the field $\mathbb{Z}_p$, and n is typically a power of two. The NTT is used to accelerate polynomial multiplication by transforming the input polynomials into the frequency domain, multiplying pointwise, and then applying the inverse transform.
A straightforward (naïve) implementation of the NTT has a time complexity of $O(n^2)$, as illustrated in typical pseudocode. However, for large inputs, this becomes computationally intensive. Fast algorithms, similar in spirit to the Cooley–Tukey FFT, reduce this to $O(n \log n)$, and these can be mapped efficiently to hardware-friendly systolic array architectures to exploit parallelism and pipelining.

3 Significance & Application Domain

- Crucial in Cryptography
  - Used in lattice-based cryptography, including post-quantum cryptography schemes like Kyber and Dilithium (both NIST finalists).
- No Precision Issues
  - Unlike FFT, which can suffer from floating-point inaccuracies, NTT's use of integers ensures exact results, which is critical in cryptographic applications.
- Well-suited for Hardware Acceleration
  - Because it uses only integer operations (addition, multiplication, modulo), it's easier to implement efficiently on FPGAs and ASICs.

4 Literature Review

A Complete Beginner Guide to the Number Theoretic Transform (NTT)

- **Link:** https://eprint.iacr.org/2024/585.pdf
- This work provides an introduction to the NTT algorithm, including different implementation strategies.

Design of Novel Systolic Array based NTT for CRYSTALS-Kyber Scheme

- **Link:** https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10808929
- Achieves approximately 23 ns for a 512-point NTT.
- Implements a 1D-systolic array using the Vivado synthesis tool.
- Demonstrates that the computational efficiency of a polynomial multiplier can be significantly improved to $\mathcal{O}(n)$ by modifying the systolic architecture to include a series of n multiply-accumulate units (MACs).

Hardware Design of an n-Coefficient NTT-Based Polynomial Multiplier

- **Link:** https://www.researchgate.net/publication/271205312_Hardware_Design_of_an_NTT-Based_Polynomial_Multiplier
- Achieves 4.35 ns latency for a 512-point NTT.
- Implemented purely in Verilog using the Quartus II tool.

Exploring Energy Efficient Quantum-Resistant Signal Processing Using Array Processors

- **Link:** https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9053653
- Achieves 10.75 μs for a complete polynomial multiplication using NTT-based multipliers.
- Describes that each multiplication requires executing the NTT block three times: bit-reversal, forward NTT, and point-wise multiplication with pre-computed twiddle factors.
- Utilizes Vivado HLS 2018.2 targeting Artix-7 and Zynq UltraScale+ FPGAs.

Optimizing Systolic Array-based NTT Accelerators

- **Link:** https://d197for5662m48.cloudfront.net/documents/publicationstatus/248660/preprint_pdf/92e991c886d74cfb0087585b44a3bde7.pdf
- Utilizes Gemmini and ZigZag frameworks for design space exploration in systolic-array-based NTT accelerator optimization.

Accelerating Polynomial Multiplication for RLWE using Pipelined FFT

- **Link:** https://eprint.iacr.org/2023/1815.pdf
- Explores three implementation approaches for FFT-based polynomial multiplication.
- Uses Vivado HLS for synthesis and evaluation.

5 Problem Formulation

Algorithm 1 Trivial $O(n^2)$ Number Theoretic Transform (NTT)

Require: Input array $InputVector[0 \dots n-1]$, prime modulus p, primitive nth root of unity $root \in \mathbb{Z}_p$, and n (should be a power of two)
Ensure: Output array $OutputVector[0 \dots n-1]$, where

$$OutputVector[i] = \sum_{j=0}^{n-1} InputVector[j]\, \omega^{ij} \mod p, \quad i = 0, 1, \dots, n-1.$$

for $i = 0$ to $n-1$ **do**
  $OutputVector[i] \leftarrow 0$
  for $j = 0$ to $n-1$ **do**
    $power \leftarrow j \times i \mod n$
    $\omega \leftarrow \text{mod\_pow}(root, power, p)$
    $OutputVector[i] \leftarrow (OutputVector[i] + (InputVector[j] \times \omega) \bmod p) \bmod p$
  end for
end for
return $OutputVector$

6 Begin The Systematic Methodology for Processor Array Design

6.1 2-D Computation Domain

The Computation Domain $\mathcal{D}$ is a plane in the 2-D integer space $\mathbb{Z}^2$, let $\boldsymbol{p}$ be a point $\in \mathcal{D}$, then $\boldsymbol{p}$ has the the following form:

$$\boldsymbol{p} = \begin{bmatrix} i \\ j \end{bmatrix}$$

In this work, we desire to map our 2-D computation domain problem to a 1-D systolic array.

6.2 Target Processor Array architecture

The Time complexity of the NTT algorithm is $\mathcal{O}(n^2)$, so we desire to design a 1D systolic array to accelerate such algorithm

6.3 Dependence Matrices and the Null Space Basis Vectors

The Dependence Matrix of the InputVector (IV) is

$$Dep_{IV} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

, and the null space basis vector (NSBV) of IV is,

$$e_{IV} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

The Dependence Matrix of the OutputVector (OV) is

$$Dep_{OV} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

, and the (NSBV) of OV is,

$$e_{OV} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The Dependence Matrix of the Twiddle Factors (TF) is

$$Dep_{TF} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

, and the (NSBV) of TF is,

$$e_{TF} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

So the TF is always broadcasted.

6.4 Data Scheduling

The scheduling function is a linear function that assigns a time value for each point $\boldsymbol{p} \in \mathcal{D}$ and is given by

$$t(\boldsymbol{p}) = s\boldsymbol{p}$$

We have two indices, i and j, to choose whether to pipeline or broadcast. In the following subsections, we shall explore all possible scheduling vectors and the associated projection direction vectors (PDV). The following information applies uniformly to all subsections unless otherwise stated

- let s be a scheduling vector $s = \begin{bmatrix} s_1 & s_2 \end{bmatrix}$
- let $\boldsymbol{p} \in \mathcal{D},\ \boldsymbol{p} = \begin{bmatrix} i \\ j \end{bmatrix}$

6.4.1 IV is pipelined and OV is pipelined

IV is pipelined so

$$se_{IV} \neq 0$$

therefore

$$\begin{bmatrix} s_1 & s_2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \neq 0$$

This implies that $s_1 \neq 0$, we choose $s_1 = 1$, OV is also pipelined so

$$se_{OV} \neq 0$$

therefore

$$\begin{bmatrix} 1 & s_2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \neq 0$$

Which implies that $s_2 \neq 0$, we choose $s_2 = 1$ The resultant scheduling vector s for the case of both variables to be pipelined is

$$s_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

To get the possible PDV, we must satisfy the inequality.y

$$sd \neq 0$$

Where d is a PDV, the following vectors are valid PDV we choose the vectors that have a single one in their entry.s

$$d_{11} = \begin{bmatrix} 1 & 0 \end{bmatrix}^t$$

$$d_{12} = \begin{bmatrix} 0 & 1 \end{bmatrix}^t$$

$$d_{13} = \begin{bmatrix} 1 & 1 \end{bmatrix}^t$$

The same goes for the rest of the subsections

6.4.2 IV is pipelined and OV is broadcasted

$$s_2 = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$d_{21} = \begin{bmatrix} 1 & 0 \end{bmatrix}^t$$

6.4.3 IV is broadcasted and OV is pipelined

$$s_3 = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$d_{31} = \begin{bmatrix} 0 & 1 \end{bmatrix}^t$$

6.4.4 IV is broadcasted and OV is broadcasted

For this case, we got

$$s_4 = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

This means that all the computations are done in one clock cycle. We will see in the Design Space Exploration section how we can construct such a systolic array that can perform the entire computation in one cycle .

6.5 Summary

Table 1 summarizes the data scheduling done above

Scheduling Vectors	Associated PDV vectors	IV	OV	TF
$s_1 = [1\ \ 1]$	$d_{11} = [1\ \ 0]$ $d_{12} = [0\ \ 1]$	Pipelined	Pipelined	Broadcasted
$s_2 = [1\ \ 0]$	$d_{21} = [1\ \ 0]$	Pipelined	Broadcasted	Broadcasted
$s_3 = [0\ \ 1]$	$d_{31} = [0\ \ 1]$	Broadcasted	Pipelined	Broadcasted
$s_4 = [0\ \ 0]$	No PDV	N/A for now	N/A for now	N/A for now

Table 1: Scheduling Vectors and their associated PDV vectors

6.6 Projection Operation

In the following subsections, we are going to derive the projection matrix, project the point $\boldsymbol{p} \in \mathcal{D}$ in the projected computation domain, and project the NSBV for the variables to see along which axis the variable is pipelined/broadcasted. Note for the variable TF, it is always localized because, regardless of the projection matrix $\boldsymbol{P}$, the projected NSBV will always equal zero.

$$\bar{e_{TF}} = \boldsymbol{P} e_{TF} = 0$$

6.6.1 PDV d_{11}

The projection matrix

$$\boldsymbol{P}_{11} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

The projected point

$$\bar{\boldsymbol{p}} = \boldsymbol{P}_{11} \boldsymbol{p} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = j$$

The projected NSBVs

$$\bar{e_{IV}} = \boldsymbol{P}_{11} e_{IV} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

This means that the IV is localized.

$$\bar{e_{OV}} = \boldsymbol{P}_{11} e_{OV} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$$

This means that the OV is going to be pipelined along the j-$axis$. The scheduling function

$$t(\boldsymbol{p}) = s\boldsymbol{p} = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i + j$$

6.6.2 PDV d_{12}

The projection matrix

$$\boldsymbol{P}_{12} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The projected point

$$\bar{\boldsymbol{p}} = \boldsymbol{P}_{12}\boldsymbol{p} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i$$

The projected NSBVs

$$e_{\bar{IV}} = \boldsymbol{P}_{12} e_{IV} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$

This means that the IV is pipelined along the *i-axis*

$$e_{\bar{OV}} = \boldsymbol{P}_{12} e_{OV} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0$$

This means that the OV is localized. The scheduling function

$$t(\boldsymbol{p}) = s\boldsymbol{p} = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i + j$$

6.6.3 PDV d_{21}

The projection matrix

$$\boldsymbol{P}_{21} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

The projected point

$$\bar{\boldsymbol{p}} = \boldsymbol{P}_{21}\boldsymbol{p} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = j$$

The projected NSBVs

$$e_{\bar{IV}} = \boldsymbol{P}_{21} e_{IV} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

This means that the IV is localized.

$$e_{\bar{OV}} = \boldsymbol{P}_{21} e_{OV} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$$

This means that the OV is going to be broadcast along the *j-axis*. The scheduling function

$$t(\boldsymbol{p}) = s\boldsymbol{p} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i$$

6.6.4 PDV d_{31}

The projection matrix

$$\boldsymbol{P}_{31} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The projected point

$$\bar{\boldsymbol{p}} = \boldsymbol{P}_{31}\boldsymbol{p} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i$$

The projected NSBVs

$$e_{\bar{IV}} = \boldsymbol{P}_{31} e_{IV} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$

This means that the IV is broadcast along the *i-axis*

$$e_{\bar{OV}} = \boldsymbol{P}_{31} e_{OV} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0$$

This means that the OV is localized. The scheduling function

$$t(\boldsymbol{p}) = s\boldsymbol{p} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = j$$

7 Design Space Exploration

In this section, we will dive deep into each of the five designs and see how each one of them performs the necessary computation differently.

7.1 1D Systolic Array Designs

For all the designs, when we refer to IV0, we mean the input at index 0 of the input vector, and its index with it, and when we refer to OV0 we mean a register that will eventually hold the output at this index and it's index with it.

7.1.1 Design 1

For design one, the size of the systolic array is N elements (i.e., the size of the input vector). The input vector is localized, and the output is accumulated throughout the pipeline. This systolic array takes I + J time steps to complete the computation

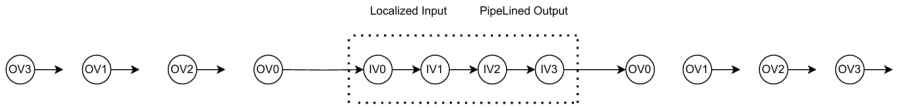

Figure 1: Systolic Array architecture schematic

7.1.2 Design 2

Design 2 has the same number of time steps needed, because the scheduling vector is common between them. The difference is that the output is localized; therefore, gathering the output will be different. See, there will be an internal register that multiplies and accumulates its value until it's done

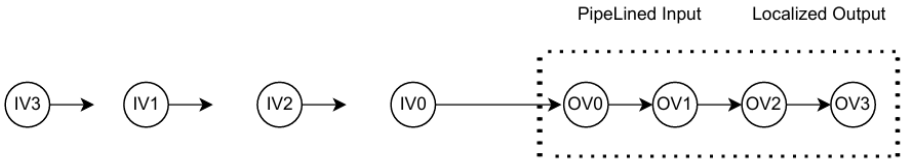

Figure 2: Systolic Array architecture schematic

7.1.3 Design 3

Design 3 has the output scattered in a broadcast fashion and then gathered using an extra adder that adds the values from all the processing elements and then saves them somewhere else.

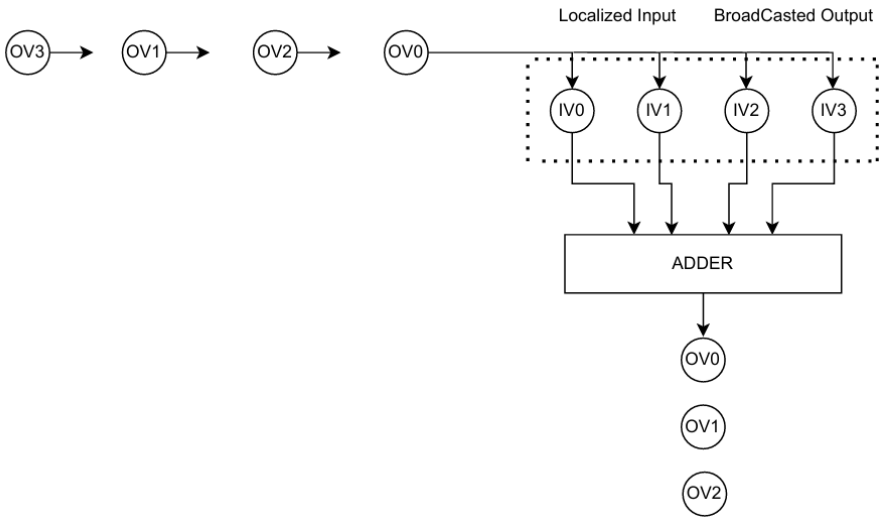

Figure 3: Systolic Array architecture schematic

7.1.4 Design 4

The last 1D design has the opposite of the previous design, such that the input is broadcast along the systolic array and the output is localized and accumulated in each processing element in a register

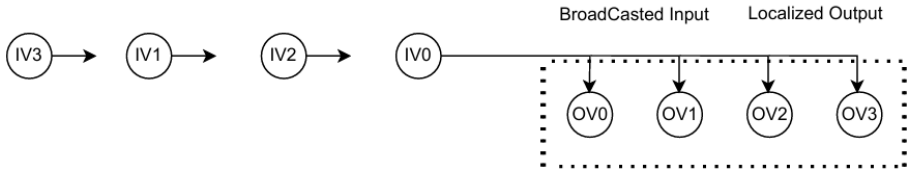

Figure 4: Systolic Array architecture schematic

7.2 The 2D Systolic Array Design

7.2.1 Design 5

This design is unique. The original problem is a 2D problem implemented in a single nested loop. constructing a **2D** systolic array is non-trivial. The only choice we have so that we can satisfy the condition of zero time steps as obtained previously, $t(p) = 0$, the input and the output should be broadcast so every processing element has the needed operands to perform its partial computation. After that, each element of the output vector comes from a line of processing elements that are added together. So, referring to Figure 5, the first element of the output vector comes from adding all the results of the processing elements along the first vertical line, so the addition is vertical-wise.

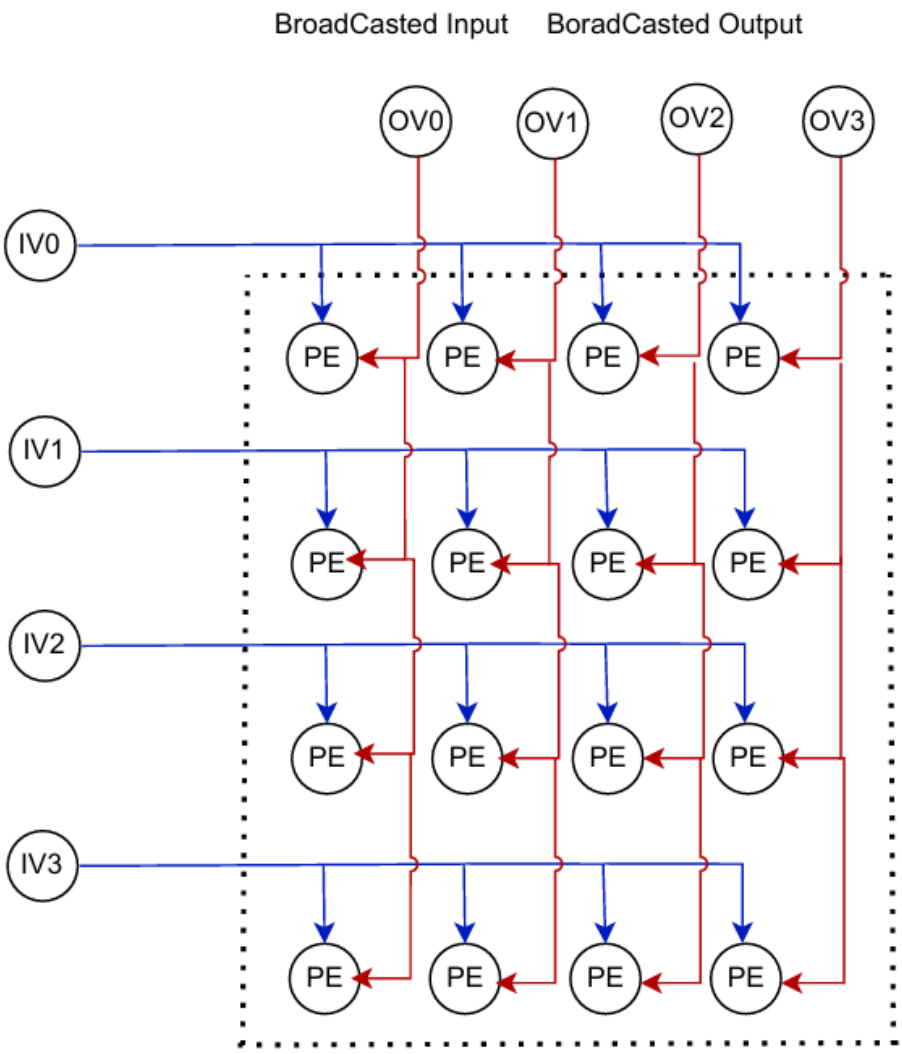

Figure 5: Systolic Array architecture schematic

Finally, for functional verification and to make sure that the designs mentioned above are true and do the right computation. A Verilog code was written for each design and simulated and verified through a test bench for a specified input, and prints the resultant Output vector to the standard output so you can see it, and of course, a waveform file will be generated to observe the signal values. A waveform screenshots were taken. Everything will be in a folder named Design

8 Conclusion

This study emphasizes the significance of systolic array architectures in optimizing the performance of the Number Theoretic Transform, a core operation in post-quantum cryptographic schemes. By leveraging the regularity and modularity of 1D systolic arrays, we can effectively parallelize and pipeline NTT computations, leading to improved computational efficiency and suitability for hardware acceleration. And as we saw, we discovered a new systolic array design, which turns out to be a 2D design that computes the NTT in one cycle. These insights pave the way for practical, high-throughput implementations on FPGAs and ASICs, facilitating secure and fast cryptographic operations in real-world applications.

9 References

References

[1] Hamid Nejatollahi, Sina Shahhosseini, Rosario Cammarota, and Nikil Dutt. Exploring energy efficient quantum-resistant signal processing using array processors. In *ICASSP 2020 - 2020*

IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 1539–1543, 2020.

[2] Keerthija Puli and Vikramkumar Pudi. Design of novel systolic array based ntt for crystals-kyber scheme. In *2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pages 539–543, 2024.

[3] CP Rentería-Mejía and J Velasco-Medina. Hardware design of an n-coefficient ntt-based polynomial multiplier.

[4] Ardianto Satriawan, Rella Mareta, and Hanho Lee. A complete beginner guide to the number theoretic transform (NTT). Cryptology ePrint Archive, Paper 2024/585, 2024.

[5] Eike Schultz, Saleh Mulhem, Lukas Groth, Mladen Berekovic, and Rainer Buchty. Optimizing systolic array-based ntt accelerators. *Authorea Preprints*, 2025.

[6] Neil Thanawala, Hamid Nejatollahi, and Nikil Dutt. Accelerating polynomial multiplication for RLWE using pipelined FFT. Cryptology ePrint Archive, Paper 2023/1815, 2023.

[4] [2] [3]
[1] [5] [6]