



LIGHTNTT: A TINY NTT/INTT CORE FOR ML-DSA FEATURING A CONSTANT-GEOMETRY PIPELINED DESIGN

Bardia Taghavi¹, Reza Azarderakhsh¹, Mehran Mozaffari Kermani²

¹Florida Atlantic University, Boca Raton, FL, USA

²University of South Florida, Tampa, FL, USA

Sep 1, 2025

1 Introduction

- The Quantum Threat & The Need for PQC
- The Bottleneck in Lattice-Based Cryptosystems
- The Solution: NTT
- The Challenge: NTT on Constrained Devices
- Our Solution: Introducing LightNTT

2 Architecture

- Core Architecture 1: Constant-Geometry (CG)
- Core Architecture 2: Efficient Memory
- Core Architecture 3: Pipelined Architecture
- Core Architecture 4: Optimized Modular Arithmetic
- Overall Architecture

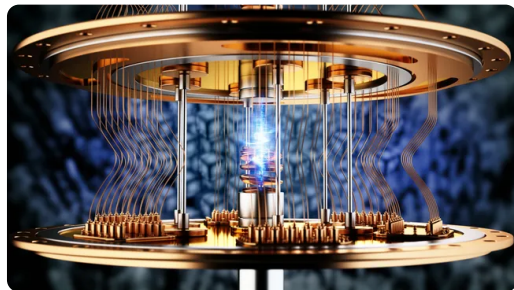
3 Implementation

- Implementation and Results
- Comparison with State-of-the-Art
- Scalability Through Parallelism
- Conclusion and Future Work

INTRODUCTION

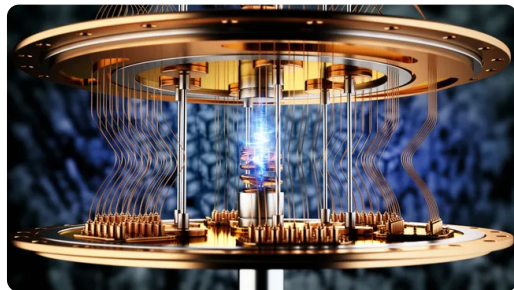
THE QUANTUM THREAT & THE NEED FOR PQC

- ✧ **The Threat:** Quantum computers running algorithms like Shor's can break the cryptography that secures our digital world today (e.g., RSA & ECC).
- ✧ **The Solution:** We must transition to Post-Quantum Cryptography (PQC), which is resistant to attacks from both classical and quantum computers.
- ✧ **A Leading Candidate:** Module-Lattice-Based Cryptography (MLBC) offers strong security and efficiency.



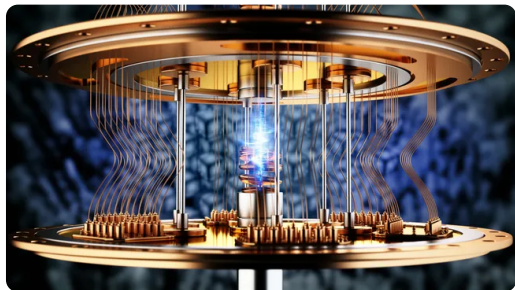
THE QUANTUM THREAT & THE NEED FOR PQC

- ✧ **The Threat:** Quantum computers running algorithms like Shor's can break the cryptography that secures our digital world today (e.g., RSA & ECC).
- ✧ **The Solution:** We must transition to Post-Quantum Cryptography (PQC), which is resistant to attacks from both classical and quantum computers.
- ✧ **A Leading Candidate:** Module-Lattice-Based Cryptography (MLBC) offers strong security and efficiency.



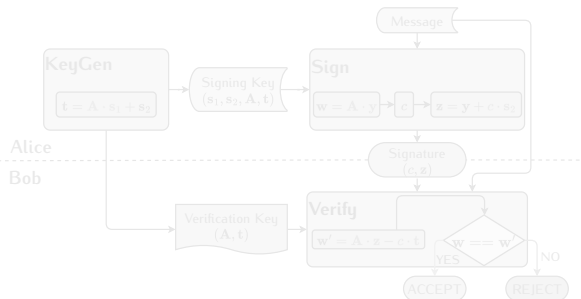
THE QUANTUM THREAT & THE NEED FOR PQC

- ❖ **The Threat:** Quantum computers running algorithms like Shor's can break the cryptography that secures our digital world today (e.g., RSA & ECC).
- ❖ **The Solution:** We must transition to Post-Quantum Cryptography (PQC), which is resistant to attacks from both classical and quantum computers.
- ❖ **A Leading Candidate:** Module-Lattice-Based Cryptography (MLBC) offers strong security and efficiency.



THE BOTTLENECK IN LATTICE-BASED CRYPTO

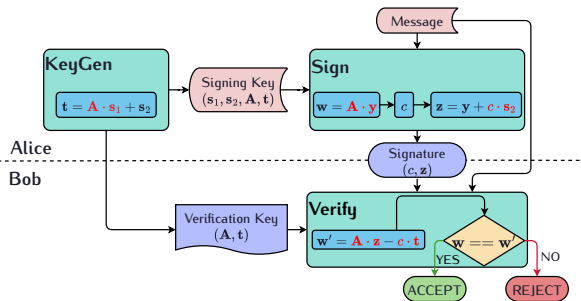
- ✧ MLBC schemes like **ML-DSA** rely heavily on polynomial multiplication.
 - ▷ MLBC schemes are based on lattices, where the basis vectors are from ring of polynomials $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $q = 8380417$ (23 bits) is a prime number and $n = 256$ is the degree of the polynomial for ML-DSA.
- ✧ This operation is the most computationally intensive part and can be a major performance bottleneck, especially on small devices.



Simplified High-Level Overview of ML-DSA

THE BOTTLENECK IN LATTICE-BASED CRYPTO

- ✧ MLBC schemes like **ML-DSA** rely heavily on polynomial multiplication.
 - ▷ MLBC schemes are based on lattices, where the basis vectors are from ring of polynomials $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $q = 8380417$ (23 bits) is a prime number and $n = 256$ is the degree of the polynomial for ML-DSA.
- ✧ This operation is the most computationally intensive part and can be a major performance bottleneck, especially on small devices.

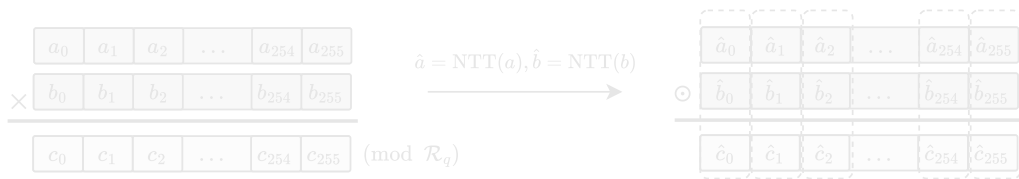


Simplified High-Level Overview of ML-DSA

THE SOLUTION: NTT

✧ **The Solution:** Number Theoretic Transform (NTT) is a highly efficient method for performing polynomial multiplication.

- ▷ It converts polynomials into a different domain where multiplication is much faster (pointwise).

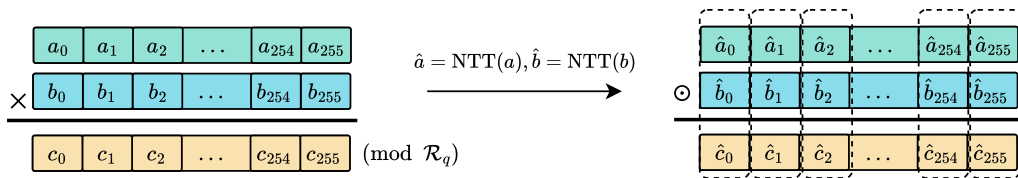


- ▷ Then, $c = \text{iNTT}(\hat{c})$ is the result of the polynomial multiplication.

THE SOLUTION: NTT

✧ **The Solution:** Number Theoretic Transform (NTT) is a highly efficient method for performing polynomial multiplication.

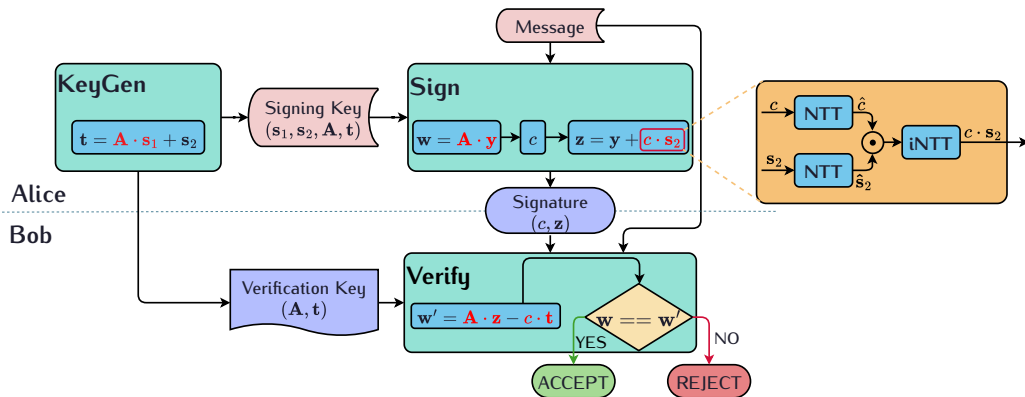
- ▷ It converts polynomials into a different domain where multiplication is much faster (pointwise).



- ▷ Then, $c = \text{iNTT}(\hat{c})$ is the result of the polynomial multiplication.

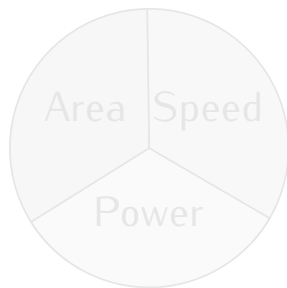
THE SOLUTION: NTT

- ✧ ML-DSA requires the use of NTT and iNTT for every polynomial multiplication.
- ✧ Steps are NTT transformation of polynomials and vectors of polynomials, pointwise multiplication (PWM) in NTT domain, and iNTT transformation.

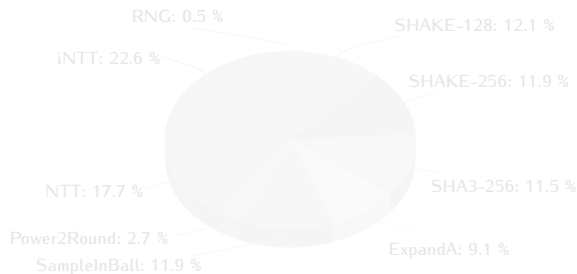


THE CHALLENGE: NTT ON CONSTRAINED DEVICES

- ✧ Implementing NTT in hardware is challenging for resource-constrained devices like **IoT sensors, smart cards, and embedded systems**.
- ✧ The core challenge is balancing the design trade-offs.



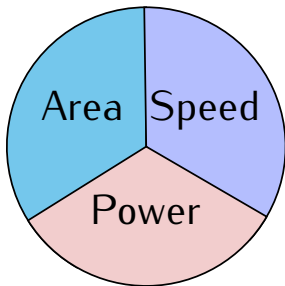
Design Trade-Offs



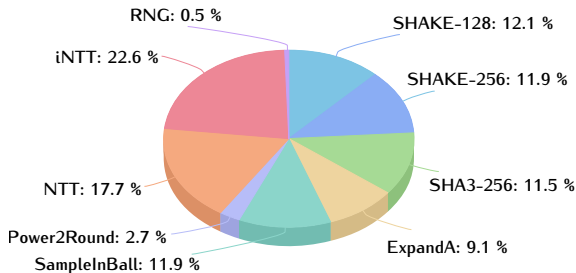
Dynamic Power Consumption of Different Blocks of ML-DSA

THE CHALLENGE: NTT ON CONSTRAINED DEVICES

- ✧ Implementing NTT in hardware is challenging for resource-constrained devices like **IoT sensors**, **smart cards**, and **embedded systems**.
- ✧ The core challenge is balancing the design trade-offs.



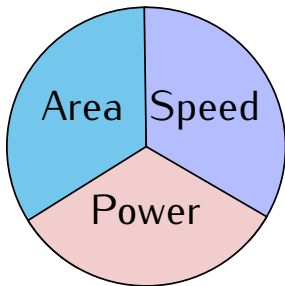
Design Trade-Offs



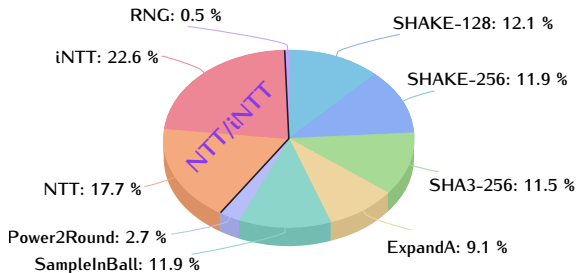
Dynamic Power Consumption of Different Blocks of ML-DSA

THE CHALLENGE: NTT ON CONSTRAINED DEVICES

- ✧ Implementing NTT in hardware is challenging for resource-constrained devices like **IoT sensors**, **smart cards**, and **embedded systems**.
- ✧ The core challenge is balancing the design trade-offs.



Design Trade-Offs



Dynamic Power Consumption of Different Blocks of ML-DSA

OUR SOLUTION: INTRODUCING LIGHTNTT

- ✧ This paper introduces **LightNTT**, a high-efficiency NTT/iNTT core specifically designed for **ML-DSA** on constrained devices.

- ✧ Key Design Principles:

Minimal Resource Use: ▷ Smallest possible footprint.	Constant-Geometry Dataflow: ▷ For simplified control and better security.
Deeply Pipelined Architecture: ▷ With a single butterfly unit to maximize throughput.	Optimized Modular Arithmetic: ▷ Using an efficient, hardware friendly, Barrett multiplier.

OUR SOLUTION: INTRODUCING LIGHTNTT

- ✧ This paper introduces **LightNTT**, a high-efficiency NTT/iNTT core specifically designed for **ML-DSA** on constrained devices.
- ✧ **Key Design Principles:**

Minimal Resource Use: ▷ Smallest possible footprint.	Constant-Geometry Dataflow: ▷ For simplified control and better security.
Deeply Pipelined Architecture: ▷ With a single butterfly unit to maximize throughput.	Optimized Modular Arithmetic: ▷ Using an efficient, hardware friendly, Barrett multiplier.

ARCHITECTURE



CORE ARCHITECTURE 1: CONSTANT-GEOMETRY (CG)

- ✧ In a typical NTT, the data access pattern changes at every stage, which complicates the control logic.
- ✧ Our Constant-Geometry (CG) architecture maintains a fixed memory access pattern throughout the computation.
- ✧ Benefits:
 - ▷ Radically simplifies control logic, saving area.
 - ▷ The regular data flow is inherently more resistant to memory-based side-channel attacks.

CORE ARCHITECTURE 1: CONSTANT-GEOMETRY (CG)

- ✧ In a typical NTT, the data access pattern changes at every stage, which complicates the control logic.
- ✧ Our **Constant-Geometry (CG)** architecture maintains a fixed memory access pattern throughout the computation.
- ✧ **Benefits:**
 - ▷ Radically simplifies control logic, saving area.
 - ▷ The regular data flow is inherently more resistant to memory-based side-channel attacks.

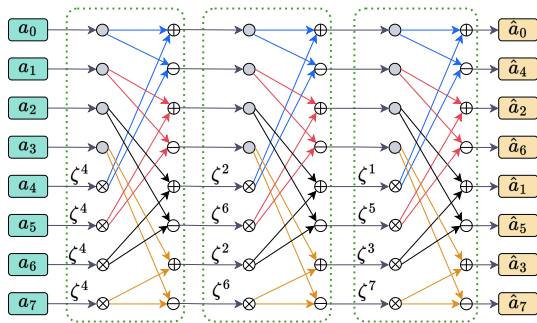
CORE ARCHITECTURE 1: CONSTANT-GEOMETRY (CG)

- ✧ In a typical NTT, the data access pattern changes at every stage, which complicates the control logic.
- ✧ Our **Constant-Geometry (CG)** architecture maintains a fixed memory access pattern throughout the computation.
- ✧ **Benefits:**
 - ▷ Radically simplifies control logic, saving area.
 - ▷ The regular data flow is inherently more resistant to memory-based side-channel attacks.

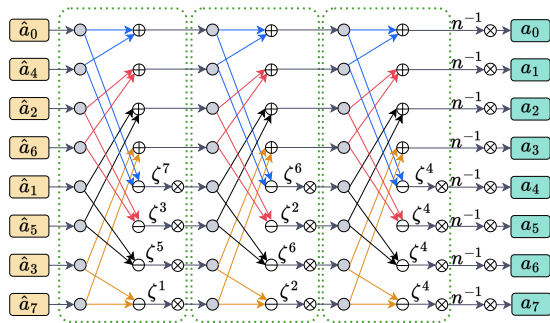
CORE ARCHITECTURE 1: CONSTANT-GEOMETRY (CG) DATAFLOW

✧ The spacing between the pairs of butterflies is fixed:

- ▷ NTT: Reads from a_i and $a_{n/2+i}$, writes into a_{2i} and a_{2i+1} .
- ▷ iNTT: Reads from a_{2i} and a_{2i+1} , writes into a_i and $a_{n/2+i}$.



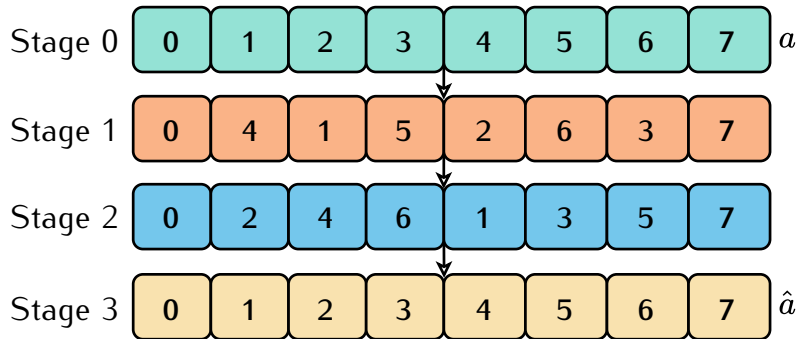
(a) NTT



(b) iNTT

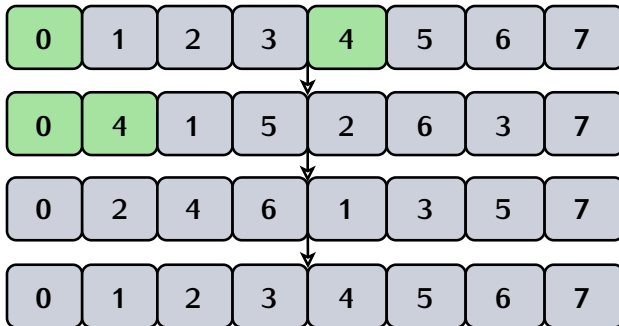
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ Overview:



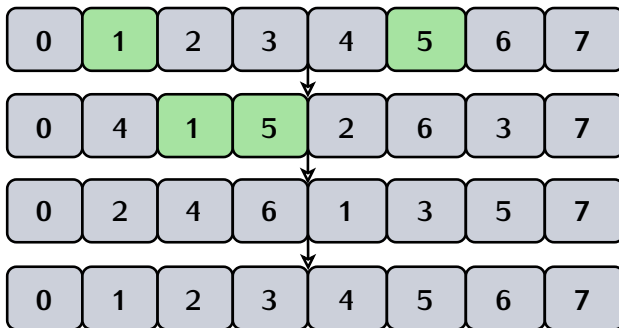
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ Stage 1 - Iteration 1: Reads from 0, 4; Writes to 0, 1.



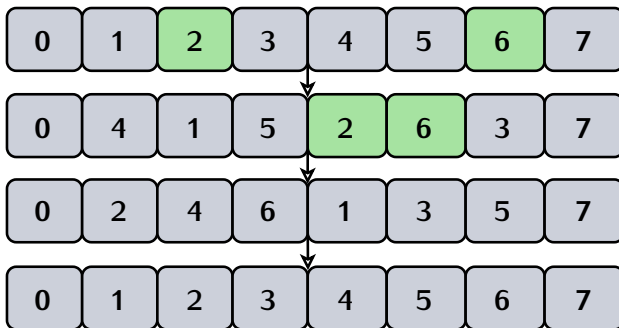
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ Stage 1 - Iteration 2: Reads from 1, 5; Writes into 2, 3.



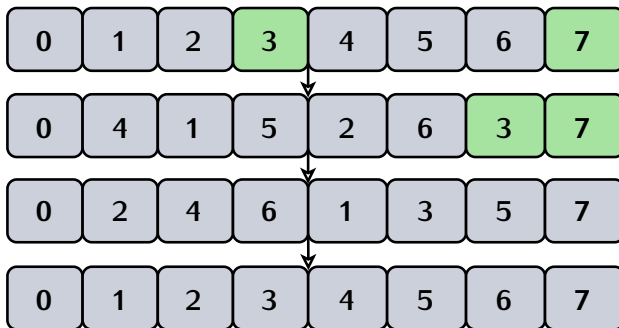
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ Stage 1 - Iteration 3: Reads from 2, 6; Writes into 4, 5.



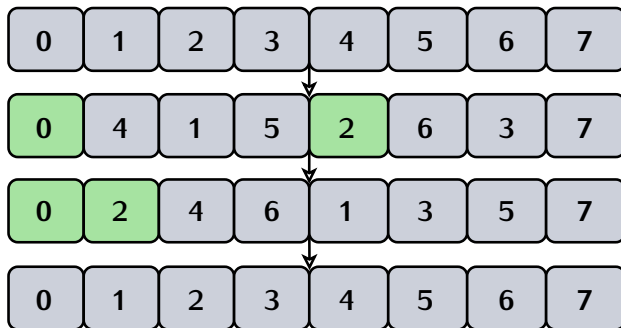
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ Stage 1 - Iteration 4: Reads from 3, 7; Writes into 6, 7.



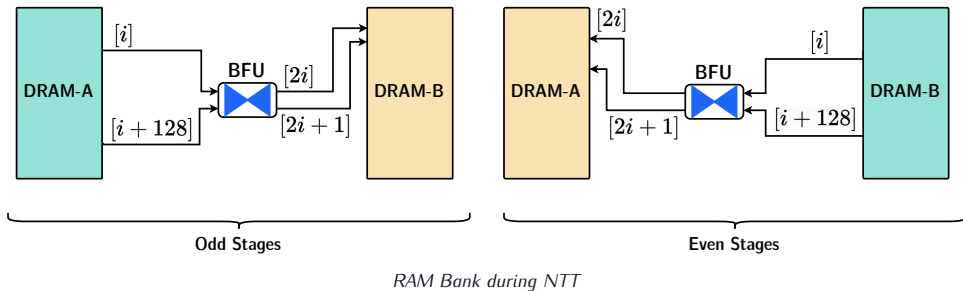
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ Stage 2 - Iteration 1: Reads from 0, 4; Writes into 0, 1, again.



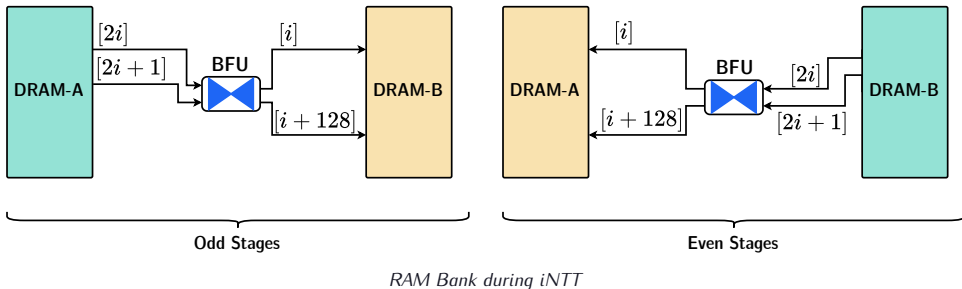
CORE ARCHITECTURE 2: EFFICIENT MEMORY

- ✧ To keep the pipeline full and avoid conflicts, we use a **ping-pong memory** architecture:
 - ▷ Two dual-port Block RAMs (BRAMs).
- ✧ While one BRAM is being read from to feed the butterfly unit, the other is being written to with the results from the previous stage. This process then alternates at the end of each stage.
- ✧ This ensures fast, conflict-free memory access that sustains the pipeline's efficiency.



CORE ARCHITECTURE 2: EFFICIENT MEMORY

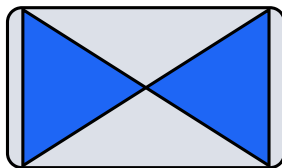
- ✧ To keep the pipeline full and avoid conflicts, we use a **ping-pong memory** architecture:
 - ▷ Two dual-port Block RAMs (BRAMs).
- ✧ While one BRAM is being read from to feed the butterfly unit, the other is being written to with the results from the previous stage. This process then alternates at the end of each stage.
- ✧ This ensures fast, conflict-free memory access that sustains the pipeline's efficiency.



CORE ARCHITECTURE 3: PIPELINED ARCHITECTURE

- ✧ The "*butterfly unit*" is the core computational unit in the NTT algorithm.
- ✧ We use a single, unified butterfly unit that handles both NTT and its inverse (iNTT), significantly reducing area.
- ✧ This unit is deeply pipelined, allowing us to process data continuously and achieve high performance with minimal hardware.
- ✧ Number of Clock Cycles:

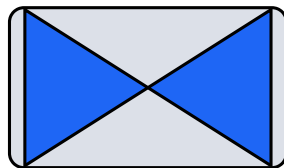
$$\left(\frac{n}{2} + 4\right) \cdot \log_2 n = 1056.$$



CORE ARCHITECTURE 3: PIPELINED ARCHITECTURE

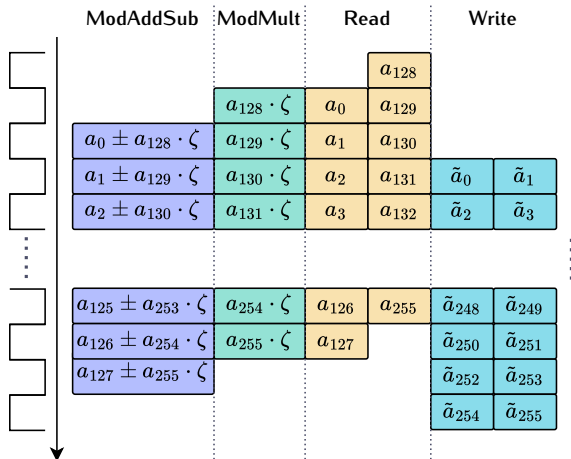
- ✧ The "*butterfly unit*" is the core computational unit in the NTT algorithm.
- ✧ We use a single, unified butterfly unit that handles both NTT and its inverse (iNTT), significantly reducing area.
- ✧ This unit is deeply pipelined, allowing us to process data continuously and achieve high performance with minimal hardware.
- ✧ Number of Clock Cycles:

$$\left(\frac{n}{2} + 4\right) \cdot \log_2 n = 1056.$$



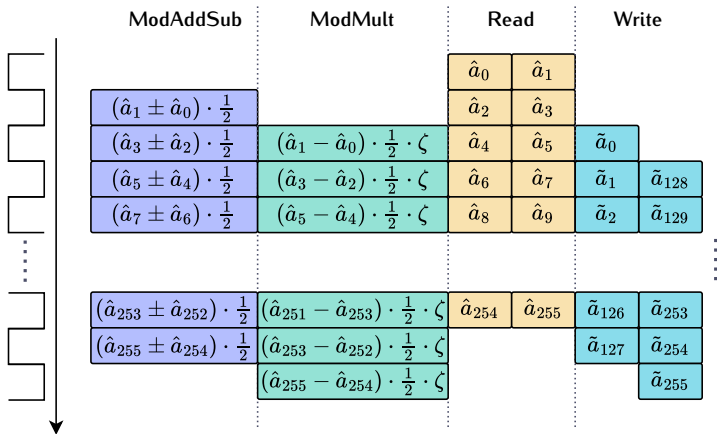
CORE ARCHITECTURE 3: PIPELINED ARCHITECTURE

- ✧ Pipeline during NTT: 3 clocks at first, 1 clock at the end, for the pipeline latency.
- ✧ The rest of the pipeline (n/2 steps) is done, each in one clock cycle.



CORE ARCHITECTURE 3: PIPELINED ARCHITECTURE

- ✧ Pipeline during iNTT: 3 clocks at first, 1 clock at the end, for the pipeline latency.
- ✧ The rest of the pipeline (n/2 steps) is done, each in one clock cycle.



CORE ARCHITECTURE 4: MODULAR ADDITION AND SUBTRACTION

- ✧ Modular addition and subtraction are quite straightforward.
- ✧ The only tricky part is multiplying by $2^{-1} \bmod q$ in iNTT after each stage.

Algorithm 1: Multiply by $2^{-1} \bmod q$

Input: $a \in \mathbb{Z}_q$

Output: $out = 2^{-1} \cdot a \bmod q$

1 $out \leftarrow a[0] ? a \gg 1 : (a + q) \gg 1$

- ▷ If a is even, just divide by 2.
- ▷ If a is odd, add q and then divide by 2.

CORE ARCHITECTURE 4: MODULAR ADDITION AND SUBTRACTION

- ✧ Modular addition and subtraction are quite straightforward.
- ✧ The only tricky part is multiplying by $2^{-1} \bmod q$ in iNTT after each stage.

Algorithm 1: Multiply by $2^{-1} \bmod q$

Input: $a \in \mathbb{Z}_q$

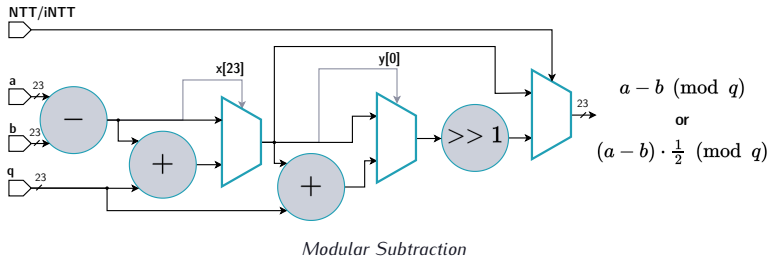
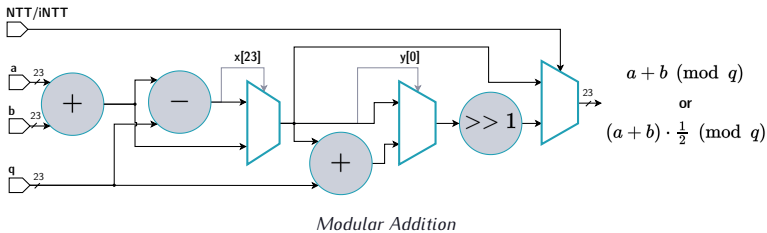
Output: $out = 2^{-1} \cdot a \bmod q$

$out \leftarrow a[0] ? a \gg 1 : (a + q) \gg 1$

- ▷ If a is even, just divide by 2.
- ▷ If a is odd, add q and then divide by 2.

CORE ARCHITECTURE 4: MODULAR ADDITION AND SUBTRACTION

✧ This hardware can be used for both NTT and iNTT.



CORE ARCHITECTURE 4: OPTIMIZED MODULAR MULTIPLICATION

- ✧ Modular multiplication is a critical and complex part of the butterfly unit.
- ✧ We implemented and compared five different algorithms: **Barrett**, **Montgomery**, **Solinas**, **Plantard**, and **K²-RED**.
- ✧ **Barrett** reduction emerged as the most efficient choice, offering the best balance of both **resource utilization** and **speed**, resulting in the lowest Area-Time Product (ATP) of 5.82.

CORE ARCHITECTURE 4: OPTIMIZED MODULAR MULTIPLICATION

- ✧ Modular multiplication is a critical and complex part of the butterfly unit.
- ✧ We implemented and compared five different algorithms: **Barrett**, **Montgomery**, **Solinas**, **Plantard**, and **K²-RED**.
- ✧ **Barrett** reduction emerged as the most efficient choice, offering the best balance of both **resource utilization** and **speed**, resulting in the lowest Area-Time Product (ATP) of 5.82.

CORE ARCHITECTURE 4: OPTIMIZED BARRETT MULTIPLICATION

$$\diamond \mu = \left\lfloor \frac{2^{2t}}{q} \right\rfloor = 2^{23} + 2^{13} + 2^3 - 1.$$

$$\diamond q = 2^{23} - 2^{13} + 1.$$

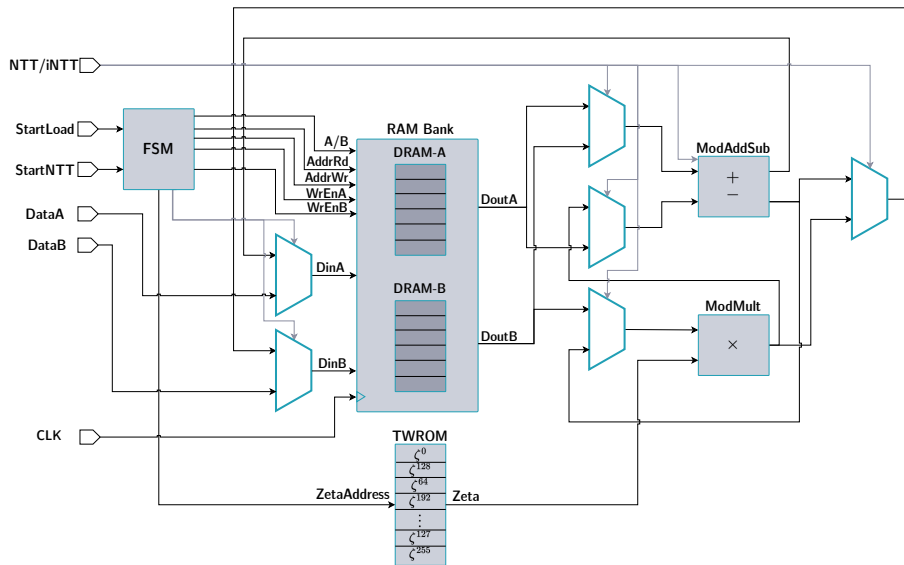
Algorithm 2: Hardware-friendly Optimized Barrett Modular Multiplication

Input: $a, b \in \mathbb{Z}_q, t = \lceil \log_2 q \rceil = 23$

Output: $out = a \cdot b \pmod{q}$

```
1  $product \leftarrow a \cdot b$ 
2  $x_1 \leftarrow product \gg (t - 2)$ 
3  $x_2 \leftarrow (x_1 \ll 23) + (x_1 \ll 13) + (x_1 \ll 3) - x_1$ 
4  $s \leftarrow x_2 \gg (t + 2)$ 
5  $out \leftarrow product - ((s \ll 23) - (s \ll 13) + s) - q$ 
6 if  $out < 0$  then
7    $out \leftarrow out + q$ 
```

OVERALL ARCHITECTURE



IMPLEMENTATION

IMPLEMENTATION AND RESULTS

- ❖ **FPGA Platform:** Artix-7 (XC7A100CSG324).
- ❖ **Target Scheme:** ML-DSA.
- ❖ **Number of Clock Cycles:** 1056, for both NTT and iNTT, in all cases:
 - ▷ Each BFU operation takes 1 clock cycle.
- ❖ The table below shows the results of NTT/iNTT core using different modular multipliers.

Modular Reduction	# LUTs	# FFs	# DSPs	# BRAMs	Freq. (MHz)	Latency (μ s)	Power (mW)	ATP
Barrett	590	158	2	2.5	302	3.49	77	5.82
Montgomery	616	158	2	2.5	277	3.81	66	6.46
Solinas Prime	692	158	2	2.5	280	3.77	74	6.68
Plantard	525	158	3	3	266	3.96	62	7.73
K ² -RED	581	158	7	2.5	257	4.11	68	11.96

IMPLEMENTATION AND RESULTS

- ❖ **FPGA Platform:** Artix-7 (XC7A100CSG324).
- ❖ **Target Scheme:** ML-DSA.
- ❖ **Number of Clock Cycles:** 1056, for both NTT and iNTT, in all cases:
 - ▷ Each BFU operation takes 1 clock cycle.
- ❖ The table below shows the results of NTT/iNTT core using different modular multipliers.

Modular Reduction	# LUTs	# FFs	# DSPs	# BRAMs	Freq. (MHz)	Latency (μ s)	Power (mW)	ATP
Barrett	590	158	2	2.5	302	3.49	77	5.82
Montgomery	616	158	2	2.5	277	3.81	66	6.46
Solinas Prime	692	158	2	2.5	280	3.77	74	6.68
Plantard	525	158	3	3	266	3.96	62	7.73
K ² -RED	581	158	7	2.5	257	4.11	68	11.96

COMPARISON WITH STATE-OF-THE-ART

- ✧ We measure overall efficiency using the Area-Time Product (ATP), where lower is better.
- ✧ LightNTT achieves an ATP of 5.82.
- ✧ This is approximately 29% better than comparable state-of-the-art small designs, and even better than larger, faster designs.

Paper	FPGA Platform	# LUTs	# FFs	# DSPs	# BRAMs	# CCs	Freq. (MHz)	Latency (μ s)	ATP
Shrivastava et al.	ZUS+ ¹	2287	1924	6	5	624	420	1.56	8.97
DiMatteo et al.	A7 ²	1302	571	9	3.5	1074	275	3.91	17.74
Malal	VUS+ ³	1601	699	10	5	603	142.8	4.22	23.00
Kundi et al.	ZUS+	3821	2970	20	5	258	322	0.80	9.06
Mandal & Roy	A7	690	771	2	2.5	1024	273	3.75	7.93
Geng et al.	V7 ⁴	1.2k	640	4	0	1034	246	4.2	10.59
Land et al.	A7	524	759	17	1	183	311	1.7	9.1
Nguyen et al.	A7	13804	11019	0	0	64	163	0.39	7.53
This Work	A7	590	158	2	2.5	1056	300	3.49	5.82

1: Zynq UltraScale+, 2: Artix-7, 3: Virtex UltraScale+, 4: Virtex-7

COMPARISON WITH STATE-OF-THE-ART

- ✧ We measure overall efficiency using the Area-Time Product (ATP), where lower is better.
- ✧ LightNTT achieves an ATP of 5.82.
- ✧ This is approximately 29% better than comparable state-of-the-art small designs, and even better than larger, faster designs.

Paper	FPGA Platform	# LUTs	# FFs	# DSPs	# BRAMs	# CCs	Freq. (MHz)	Latency (μ s)	ATP
Shrivastava et al.	ZUS+ ¹	2287	1924	6	5	624	420	1.56	8.97
DiMatteo et al.	A7 ²	1302	571	9	3.5	1074	275	3.91	17.74
Malal	VUS+ ³	1601	699	10	5	603	142.8	4.22	23.00
Kundi et al.	ZUS+	3821	2970	20	5	258	322	0.80	9.06
Mandal & Roy	A7	690	771	2	2.5	1024	273	3.75	7.93
Geng et al.	V7 ⁴	1.2k	640	4	0	1034	246	4.2	10.59
Land et al.	A7	524	759	17	1	183	311	1.7	9.1
Nguyen et al.	A7	13804	11019	0	0	64	163	0.39	7.53
This Work	A7	590	158	2	2.5	1056	300	3.49	5.82

1: Zynq UltraScale+, 2: Artix-7, 3: Virtex UltraScale+, 4: Virtex-7

SCALABILITY THROUGH PARALLELISM

- ✧ The small size of LightNTT is a significant advantage.
- ✧ For applications requiring higher throughput, multiple LightNTT cores can be instantiated to run in parallel.
- ✧ For example, using 4 parallel cores could achieve a very competitive throughput of nearly 6.8 Gbps. This is not feasible with larger, more area-intensive designs.

In addition to the applications that face limitations in area usage, such as *smart cards, IoT devices, wearable gadgets, etc.*

SCALABILITY THROUGH PARALLELISM

- ✧ The small size of LightNTT is a significant advantage.
- ✧ For applications requiring higher throughput, multiple LightNTT cores can be instantiated to run in parallel.
- ✧ For example, using 4 parallel cores could achieve a very competitive throughput of nearly 6.8 Gbps. This is not feasible with larger, more area-intensive designs.

In addition to the applications that face limitations in area usage, such as *smart cards, IoT devices, wearable gadgets*, etc.

CONCLUSION AND FUTURE WORK

✧ Conclusion:

- ▷ We presented LightNTT, a lightweight and efficient NTT/iNTT hardware core for ML-DSA.
- ▷ By combining a constant-geometry dataflow, a pipelined butterfly unit, and an optimized Barrett multiplier, we achieved a compelling balance between area, speed, and power.

✧ Future Work:

- ▷ **Security Analysis:** Formally investigate and quantify the design's resilience against side-channel attacks (SCA).
- ▷ **Broader Support:** Adapt the architecture to support other module-lattice-based cryptographic schemes.

ACKNOWLEDGMENT

Our Team:



Bardia Taghavi,
PhD Candidate
Florida Atlantic University, USA.



Reza Azarderakhsh,
Professor
Florida Atlantic University, USA.



Mehran Mozaffari Kermani,
Associate Professor
University of South Florida, USA.

Sponsors:



Thank you for your attention!

Any questions?