



KNIGHTCORE: AN ULTRA-LIGHTWEIGHT NTT-BASED POLYNOMIAL MULTIPLIER FOR ML-KEM ON RESOURCE-CONSTRAINED PLATFORMS

Bardia Taghavi¹, Reza Azarderakhsh¹, Mehran Mozaffari Kermani²

¹Florida Atlantic University, Boca Raton, FL, USA

²University of South Florida, Tampa, FL, USA

Oct 13, 2025

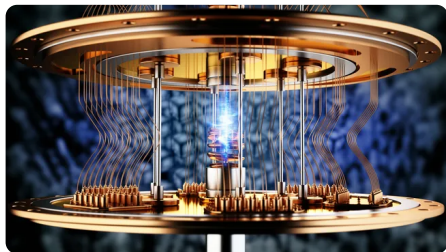
- 1 Introduction & Motivation
 - The Quantum Threat & The PQC Solution
 - The Hardware Challenge for ML-KEM
 - ML-KEM Overview
- 2 Background
 - Number Theoretic Transforms (NTT) in ML-KEM
 - Point-Wise Multiplication (PWM) in ML-KEM
 - Related Work
- 3 KNightCore Architecture
 - Our Contributions
 - Constant-Geometry (CG)
 - Key Innovation: PWM via Twiddle Factor Reuse
 - Proposed Architecture Overview
- 4 Results & Comparison
 - Implementation Results
- 5 Conclusion & Future Work

INTRODUCTION & MOTIVATION

THE QUANTUM THREAT & THE PQC SOLUTION

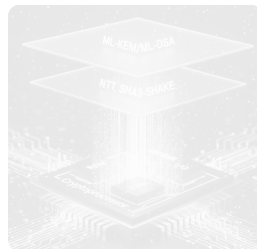
The Threat

- ✧ The rise of quantum computers threatens the security of current public-key cryptography (e.g., RSA, ECC).
- ✧ Systems relying on factoring or discrete log problems will become insecure.
- ✧ This impacts nearly all digital security, from financial transactions to secure communications.



The Solution

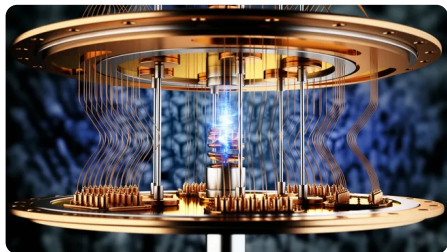
- ✧ Post-Quantum Cryptography (PQC) provides new algorithms resistant to quantum attacks.
- ✧ NIST has standardized ML-KEM (FIPS 203) for quantum-resistant key establishment.
- ✧ ML-KEM's security is based on the Module Learning With Errors (Module-LWE) problem.



THE QUANTUM THREAT & THE PQC SOLUTION

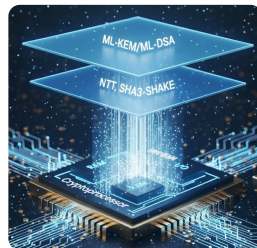
The Threat

- ✧ The rise of quantum computers threatens the security of current public-key cryptography (e.g., RSA, ECC).
- ✧ Systems relying on factoring or discrete log problems will become insecure.
- ✧ This impacts nearly all digital security, from financial transactions to secure communications.



The Solution

- ✧ Post-Quantum Cryptography (PQC) provides new algorithms resistant to quantum attacks.
- ✧ NIST has standardized ML-KEM (FIPS 203) for quantum-resistant key establishment.
- ✧ ML-KEM's security is based on the Module Learning With Errors (Module-LWE) problem.



THE HARDWARE CHALLENGE FOR ML-KEM

The Bottleneck: Polynomial Multiplication

- ✧ The performance of ML-KEM hardware is overwhelmingly dominated by polynomial arithmetic.
- ✧ These operations occur in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $n = 256$ and $q = 3329$.
- ✧ The core operation is a three-stage process:
 1. Forward Number Theoretic Transform (NTT)
 2. Point-Wise Multiplication (PWM)
 3. Inverse Number Theoretic Transform (iNTT)

The Goal

Design an **ultra-lightweight, unified hardware core** for all three operations, specifically for resource-constrained platforms like IoT devices and smart cards where area, power, and DSP blocks are scarce.

THE HARDWARE CHALLENGE FOR ML-KEM

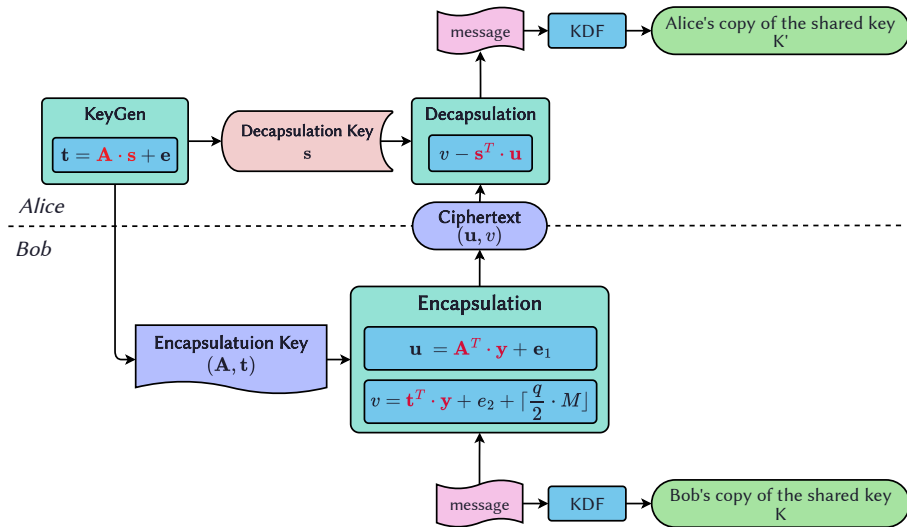
The Bottleneck: Polynomial Multiplication

- ✧ The performance of ML-KEM hardware is overwhelmingly dominated by polynomial arithmetic.
- ✧ These operations occur in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $n = 256$ and $q = 3329$.
- ✧ The core operation is a three-stage process:
 1. Forward Number Theoretic Transform (NTT)
 2. Point-Wise Multiplication (PWM)
 3. Inverse Number Theoretic Transform (iNTT)

The Goal

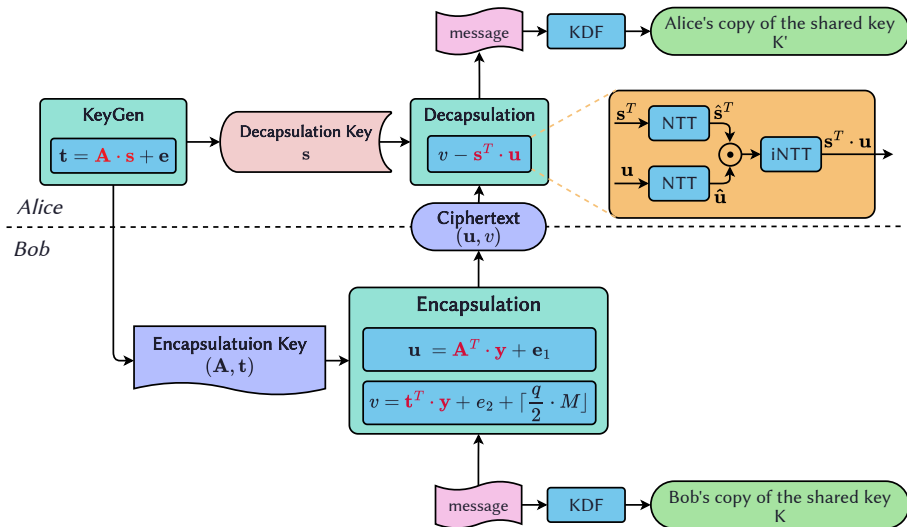
Design an **ultra-lightweight, unified hardware core** for all **three operations**, specifically for resource-constrained platforms like IoT devices and smart cards where area, power, and DSP blocks are scarce.

NTT & PWM IN ML-KEM



Overview of ML-KEM

NTT & PWM IN ML-KEM



Overview of ML-KEM

BACKGROUND



NUMBER THEORETIC TRANSFORMS (NTT) IN ML-KEM

- ✧ NTT is an efficient algorithm used to compute polynomial multiplication, reducing complexity from $O(n^2)$ to $O(n \log n)$.
- ✧ ML-KEM uses an **incomplete NTT**, which transforms n coefficients into $n/2$ degree-1 polynomials of form:

$$[\hat{a}_{2j}, \hat{a}_{2j+1}] = \hat{a}_{2j} + \hat{a}_{2j+1} \cdot x$$

- ✧ This reduces the number of computation stages from $\log_2 n$ to $\log_2 n - 1$.

POINT-WISE MULTIPLICATION (PWM) IN ML-KEM

Point-Wise Multiplication (PWM)

- ✧ After the NTT, coefficients are multiplied point-wise.
- ✧ For the incomplete NTT, this step is complex, requiring 5 modular multiplications and 2 additions for each pair of output coefficients.
- ✧ Implementing this efficiently without a large hardware footprint is a major design challenge.

PWM Equation for ML-KEM

$$\begin{aligned}\hat{c}_{2j} &= \hat{a}_{2j} \cdot \hat{b}_{2j} + \hat{a}_{2j+1} \cdot \hat{b}_{2j+1} \cdot \zeta^{2\text{brv}_l(j)+1} \pmod{q}, \\ \hat{c}_{2j+1} &= \hat{a}_{2j} \cdot \hat{b}_{2j+1} + \hat{a}_{2j+1} \cdot \hat{b}_{2j} \pmod{q}.\end{aligned}$$

RELATED WORK: A BALANCING ACT

- ✧ Hardware acceleration for PQC has been widely studied, with designs trading off performance for area.
- ✧ **High-Speed Designs:** Often use [multiple DSP blocks](#) and [parallel butterfly units](#) to minimize latency, but at a high resource cost.
- ✧ **Lightweight Designs:** Focus on area efficiency by eliminating DSPs and reducing memory. Some achieve this but may require [more BRAMs](#), or [more LUTs](#) due to unoptimized modular multiplication.
- ✧ **The Gap:** Many solutions use separate hardware for transformation (NTT) and multiplication (PWM), which is inefficient for compact designs.

*KNightCore aims to bridge this gap by creating a **single, unified core** that is both DSP-free and highly area-efficient, based on our previous work, **LightNTT**.*

RELATED WORK: A BALANCING ACT

- ✧ Hardware acceleration for PQC has been widely studied, with designs trading off performance for area.
- ✧ **High-Speed Designs:** Often use **multiple DSP blocks** and **parallel butterfly units** to minimize latency, but at a high resource cost.
- ✧ **Lightweight Designs:** Focus on area efficiency by eliminating DSPs and reducing memory. Some achieve this but may require **more BRAMs**, or **more LUTs** due to unoptimized modular multiplication.
- ✧ **The Gap:** Many solutions use separate hardware for transformation (NTT) and multiplication (PWM), which is inefficient for compact designs.

*KNightCore aims to bridge this gap by creating a **single, unified core** that is both DSP-free and highly area-efficient, based on our previous work, **LightNTT**.*

KNIGHTCORE ARCHITECTURE

OUR CONTRIBUTIONS

1. Unified, Constant-Geometry Datapath

A single, deeply pipelined butterfly unit is architected to execute **forward NTT**, **inverse NTT**, and **point-wise multiplication** on the same hardware, governed by a simple control mechanism.

2. Novel PWM via Twiddle Factor Reuse

The main innovation: a PWM method that reuses the existing NTT twiddle factor ROM and datapath, completely **eliminating the need for dedicated PWM multipliers and storage**.

3. Ultra-Lightweight, DSP-Free Implementation

The unified design achieves an exceptionally low resource footprint, making it one of the most area-efficient ML-KEM accelerators to date.

OUR CONTRIBUTIONS

1. Unified, Constant-Geometry Datapath

A single, deeply pipelined butterfly unit is architected to execute **forward NTT**, **inverse NTT**, and **point-wise multiplication** on the same hardware, governed by a simple control mechanism.

2. Novel PWM via Twiddle Factor Reuse

The main innovation: a PWM method that reuses the existing NTT twiddle factor ROM and datapath, completely **eliminating the need for dedicated PWM multipliers and storage**.

3. Ultra-Lightweight, DSP-Free Implementation

The unified design achieves an exceptionally low resource footprint, making it one of the most area-efficient ML-KEM accelerators to date.

OUR CONTRIBUTIONS

1. Unified, Constant-Geometry Datapath

A single, deeply pipelined butterfly unit is architected to execute **forward NTT**, **inverse NTT**, and **point-wise multiplication** on the same hardware, governed by a simple control mechanism.

2. Novel PWM via Twiddle Factor Reuse

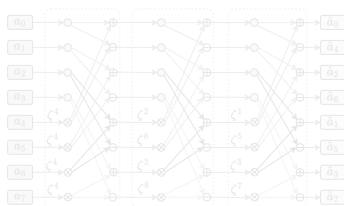
The main innovation: a PWM method that reuses the existing NTT twiddle factor ROM and datapath, completely **eliminating the need for dedicated PWM multipliers and storage**.

3. Ultra-Lightweight, DSP-Free Implementation

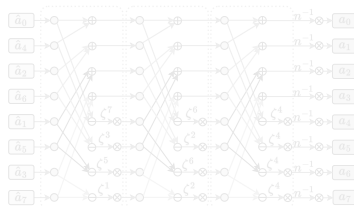
The unified design achieves an exceptionally low resource footprint, making it one of the most area-efficient ML-KEM accelerators to date.

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.



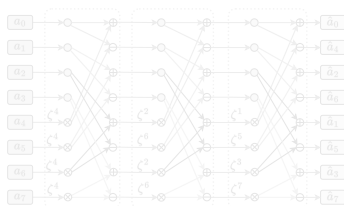
(a) NTT



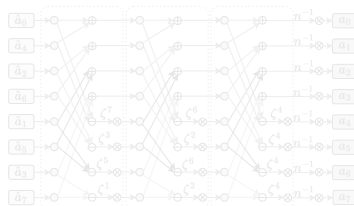
(b) iNTT

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.



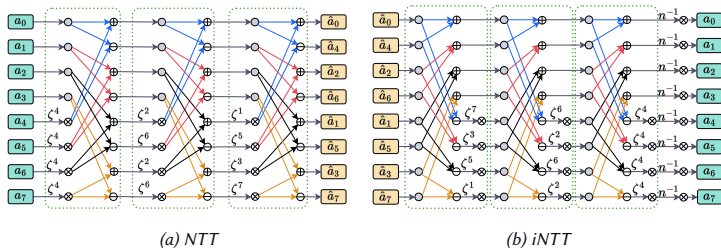
(a) NTT



(b) iNTT

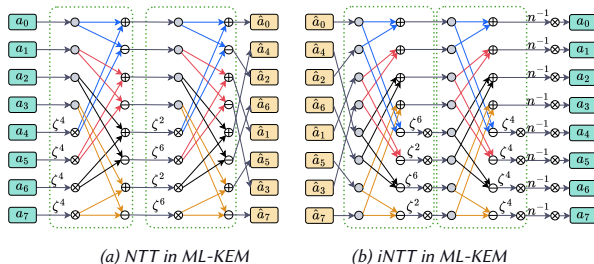
CONSTANT-GEOMETRY (CG)

- ✧ 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.



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.



CONSTANT-GEOMETRY (CG) NTT DATAFLOW

The spacing between the pairs of butterflies is fixed:

- ✧ **NTT:** Reads from a_i and $a_{n/2+i}$ (spacing is $n/2$), writes into a_{2i} and a_{2i+1} (spacing is 1).
- ✧ **iNTT:** Reads from a_{2i} and a_{2i+1} (spacing is 1), writes into a_i and $a_{n/2+i}$ (spacing is $n/2$).

Incomplete NTT Exception

- ✧ **NTT:** The last stage of the NTT writes into $[a_{2i}, a_{2i+2}]$ and then $[a_{2i+1}, a_{2i+3}]$ (spacing is 2).
- ✧ **iNTT:** The first stage of the iNTT reads from $[a_{2i}, a_{2i+2}]$ and then $[a_{2i+1}, a_{2i+3}]$ (spacing is 2).

CONSTANT-GEOMETRY (CG) NTT DATAFLOW

The spacing between the pairs of butterflies is fixed:

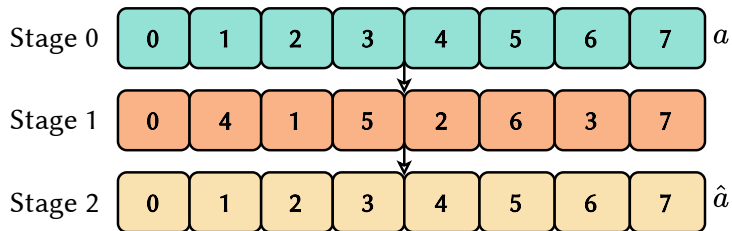
- ✧ **NTT:** Reads from a_i and $a_{n/2+i}$ (spacing is $n/2$), writes into a_{2i} and a_{2i+1} (spacing is 1).
- ✧ **iNTT:** Reads from a_{2i} and a_{2i+1} (spacing is 1), writes into a_i and $a_{n/2+i}$ (spacing is $n/2$).

Incomplete NTT Exception

- ✧ **NTT:** The last stage of the NTT writes into $[a_{2i}, a_{2i+2}]$ and then $[a_{2i+1}, a_{2i+3}]$ (spacing is 2).
- ✧ **iNTT:** The first stage of the iNTT reads from $[a_{2i}, a_{2i+2}]$ and then $[a_{2i+1}, a_{2i+3}]$ (spacing is 2).

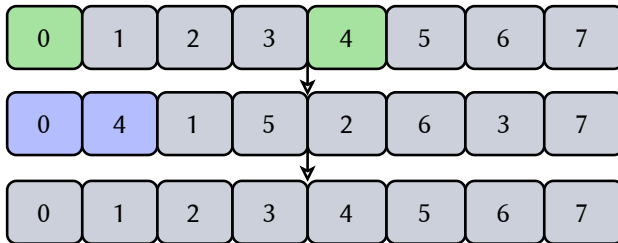
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ Overview:



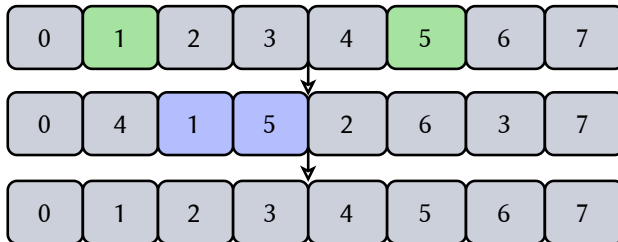
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

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



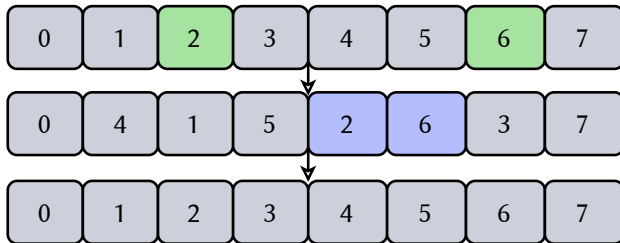
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

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



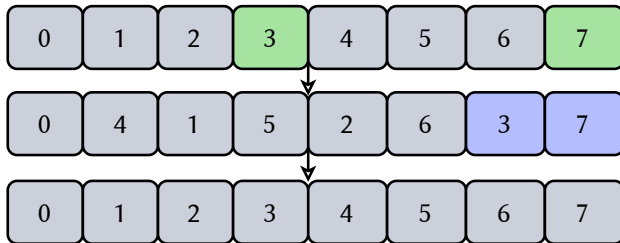
CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ **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.

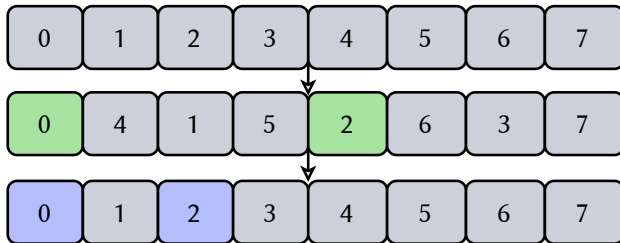


Note

These iterations are exactly the same as the first iterations in all other stages in constant-geometry architecture, **except for the last stage.**

CONSTANT-GEOMETRY (CG) NTT DATAFLOW

✧ **Stage 2 (last stage) - Iteration 1:** Reads from 0, 4; Writes into 0, 2.



KEY INNOVATION: PWM VIA TWIDDLE FACTOR REUSE

The PWM stage for incomplete NTT requires multiplication by twiddle factors of the form $\zeta^{2\text{brv}(j)+1}$.

The Challenge

Storing these extra twiddle factors would require additional ROM. Implementing the 5 required multiplications in parallel would require significant hardware.

The KNightCore Solution

We discovered a mathematical property that allows us to generate the required PWM twiddle factors from the existing NTT twiddle factors stored in ROM:

$$\begin{aligned}\zeta^{2\text{brv}_l(2i)+1} &\equiv \zeta^{\text{brv}_l(n/4+i)} \pmod{q}, \\ \zeta^{2\text{brv}_l(2i+1)+1} &\equiv -\zeta^{\text{brv}_l(n/4+i)} \pmod{q}.\end{aligned}$$

This allows us to reuse the NTT's **ROM**, and also the multiplier, adder, and subtractor to perform the entire PWM operation with **zero additional hardware**.

KEY INNOVATION: PWM VIA TWIDDLE FACTOR REUSE

The PWM stage for incomplete NTT requires multiplication by twiddle factors of the form $\zeta^{2\text{brv}(j)+1}$.

The Challenge

Storing these extra twiddle factors would require additional ROM. Implementing the 5 required multiplications in parallel would require significant hardware.

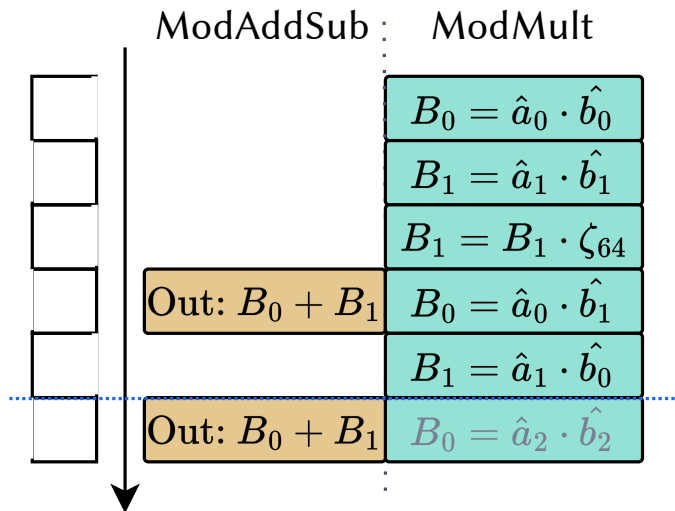
The KNightCore Solution

We discovered a mathematical property that allows us to generate the required PWM twiddle factors from the existing NTT twiddle factors stored in ROM:

$$\begin{aligned}\zeta^{2\text{brv}_l(2i)+1} &\equiv \zeta^{\text{brv}_l(n/4+i)} \pmod{q}, \\ \zeta^{2\text{brv}_l(2i+1)+1} &\equiv -\zeta^{\text{brv}_l(n/4+i)} \pmod{q}.\end{aligned}$$

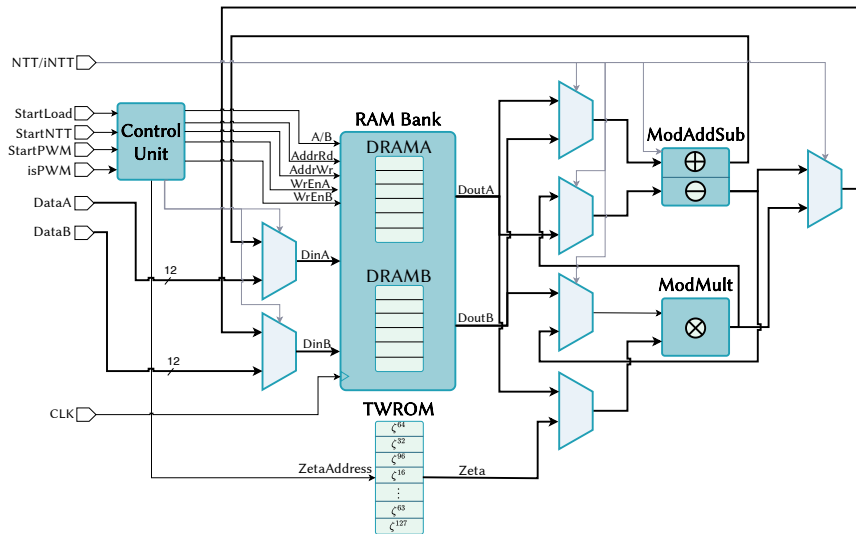
This allows us to reuse the NTT's **ROM**, and also the multiplier, adder, and subtractor to perform the entire PWM operation with **zero additional hardware**.

PWM PIPELINE



PWM Pipeline

PROPOSED ARCHITECTURE OVERVIEW



Hardware Block Diagram of KNightCore

PROPOSED ARCHITECTURE OVERVIEW: KEY COMPONENTS

- ✧ **Unified Core:** A single datapath with a *single butterfly unit* handles NTT, iNTT, and PWM.
- ✧ **Memory:** A ping-pong RAM Bank ($2 \times 256 \times 12$ bits) ensures conflict-free memory access, eliminating pipeline stalls.
- ✧ **Twiddle Factor ROM:** A single small ROM (128×12 bits) stores twiddle factors that are reused for all three operations.

RESULTS & COMPARISON

IMPLEMENTATION RESULTS ON ARTIX-7

✧ KNightCore was implemented on a Xilinx Artix-7 FPGA (xc7a100csg324).

Resource Utilization

- ✧ **LUTs:** 650
- ✧ **FFs:** 130
- ✧ **BRAMs:** 1.5
- ✧ **DSPs:** **0**
- ✧ **Dynamic Power Consumption:** 0.058 mW

Performance @ 274 MHz

- ✧ **NTT/iNTT:**
 - ▷ 924 cycles
 - ▷ 3.37 μ s
- ✧ **PWM:**
 - ▷ 643 cycles
 - ▷ 2.35 μ s

COMPARATIVE ANALYSIS

Paper	PWM?	#LUTs	#FFs	#DSPs	#BRAMs	CC (NTT/PWM)	Freq. (MHz)	Latency (μ s) (NTT/PWM)	ATP
Yaman et al.	Yes	948	352	1	2.5	904/3359	190	4.76/17.68	4.09
Sun & Bai	Yes	1070	1071	5	10.5	306/-	296	1.03/-	3.11
Ni et al.	Yes	1875	1669	4	0	228/145	294	0.78/0.49	1.28
Zhang et al.	Yes	609	640	2	4	490/536	257	1.91/2.08	2.39
Liu et al.	No	632	402	3	2.5	1033	335	3.08	3.9
Nguyen et al.	No	541	680	0	4	461	417	1.11	0.84
Javeed & Gregg	No	676	633	0	0	2016	307	6.57	1.63
KNightCore	Yes	650	130	0	1.5	924/643	274	3.37/2.35	1.28

- ❖ KNightCore achieves an Area-Time Product (ATP) of 1.28, matching the efficiency of a high-speed design Ni et al. but without using any DSPs.
- ❖ Compared to other DSP-free designs like Nguyen et al., KNightCore includes the crucial PWM functionality while using significantly fewer BRAMs (1.5 vs. 4).

COMPARATIVE ANALYSIS

Paper	PWM?	#LUTs	#FFs	#DSPs	#BRAMs	CC (NTT/PWM)	Freq. (MHz)	Latency (μ s) (NTT/PWM)	ATP
Yaman et al.	Yes	948	352	1	2.5	904/3359	190	4.76/17.68	4.09
Sun & Bai	Yes	1070	1071	5	10.5	306/-	296	1.03/-	3.11
Ni et al.	Yes	1875	1669	4	0	228/145	294	0.78/0.49	1.28
Zhang et al.	Yes	609	640	2	4	490/536	257	1.91/2.08	2.39
Liu et al.	No	632	402	3	2.5	1033	335	3.08	3.9
Nguyen et al.	No	541	680	0	4	461	417	1.11	0.84
Javeed & Gregg	No	676	633	0	0	2016	307	6.57	1.63
KNightCore	Yes	650	130	0	1.5	924/643	274	3.37/2.35	1.28

- ✧ KNightCore achieves an Area-Time Product (ATP) of **1.28**, matching the efficiency of a high-speed design Ni et al. but without using any DSPs.
- ✧ Compared to other DSP-free designs like Nguyen et al., KNightCore includes the crucial PWM functionality while using significantly fewer BRAMs (**1.5** vs. **4**).

CONCLUSION & FUTURE WORK

CONCLUSION

- ✧ This paper presented **KNightCore**, an ultra-lightweight, unified arithmetic core for polynomial multiplication in ML-KEM.
- ✧ By leveraging a Constant-Geometry datapath and a **novel method for PWM that reuses existing resources**, KNightCore performs all necessary operations in a single, compact pipeline.
- ✧ Our implementation on an Artix-7 FPGA demonstrates exceptional resource efficiency:
 - ▷ **0 DSPs** and only **1.5 BRAMs**.
 - ▷ Low power consumption of **0.058 W**.
 - ▷ Highly competitive Area-Time Product (ATP) of **1.28**.
- ✧ KNightCore is a potent and validated solution for resource-constrained platforms like IoT devices and smart cards that require efficient post-quantum cryptography.

System Integration

Integrate KNightCore into a full System-on-Chip (SoC) for secure applications, such as smart cards and embedded Secure Elements (eSEs).

Security Enhancements

Develop and incorporate countermeasures against side-channel attacks (e.g., power and timing analysis), which are critical for physically accessible devices.

Architectural Exploration

- ❖ Adapt the KNightCore architecture to support other module-lattice-based cryptographic schemes with a unified hardware core.
- ❖ Further optimize control logic to reduce latency and solidify its suitability for next-generation secure devices.

System Integration

Integrate KNightCore into a full System-on-Chip (SoC) for secure applications, such as smart cards and embedded Secure Elements (eSEs).

Security Enhancements

Develop and incorporate countermeasures against side-channel attacks (e.g., power and timing analysis), which are critical for physically accessible devices.

Architectural Exploration

- ❖ Adapt the KNightCore architecture to support other module-lattice-based cryptographic schemes with a unified hardware core.
- ❖ Further optimize control logic to reduce latency and solidify its suitability for next-generation secure devices.

System Integration

Integrate KNightCore into a full System-on-Chip (SoC) for secure applications, such as smart cards and embedded Secure Elements (eSEs).

Security Enhancements

Develop and incorporate countermeasures against side-channel attacks (e.g., power and timing analysis), which are critical for physically accessible devices.

Architectural Exploration

- ✧ Adapt the KNightCore architecture to support other module-lattice-based cryptographic schemes with a unified hardware core.
- ✧ Further optimize control logic to reduce latency and solidify its suitability for next-generation secure devices.

ACKNOWLEDGMENT

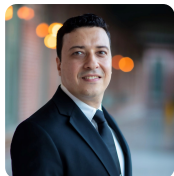
Our Team:



Bardia Taghavi,
PhD Candidate
Florida Atlantic University, USA.
staghavi2024@fau.edu



Reza Azarderakhsh,
Professor
Florida Atlantic University, USA.
razarderakhsh@fau.edu



Mehran Mozaffari Kermani,
Associate Professor
University of South Florida, USA.
mehran2@usf.edu

Sponsors:



Thank you for your attention!

Any questions?

staghavi2024@fau.edu