



Hardware Implementation of Number Theoretic Transformation for PQC on FPGA

Debapriya Basu Roy

SETTLOR LAB

Department of Computer Science & Engineering
Indian Institute of Technology Kanpur, India

This talk is based on

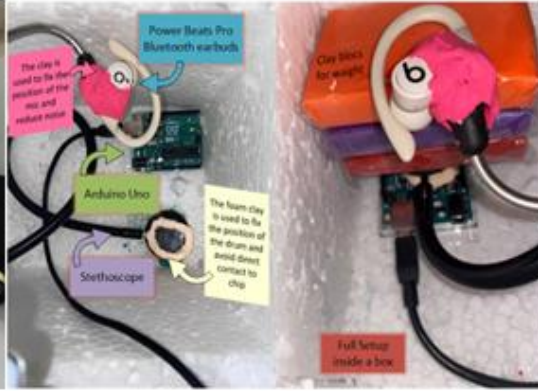
KiD Unified NTT for Kyber & Dilithium · VLSID 2024 **Winograd for NTT** Higher-radix, low-latency NTT · IEEE TCAS-I 2024

Lightweight Radix-4 NTT Design exploration for ML-KEM & ML-DSA · FPL 2026

Secure Embedded and Smart Things Laboratory (SETTLOR), CSE, IITK



Side-Channel secure Efficient hardware for PQC



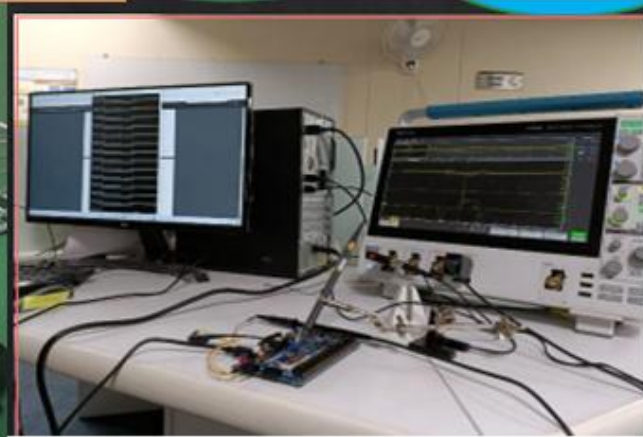
Acoustic Side Channel Attacks



Screen Gleaning for OTP extraction



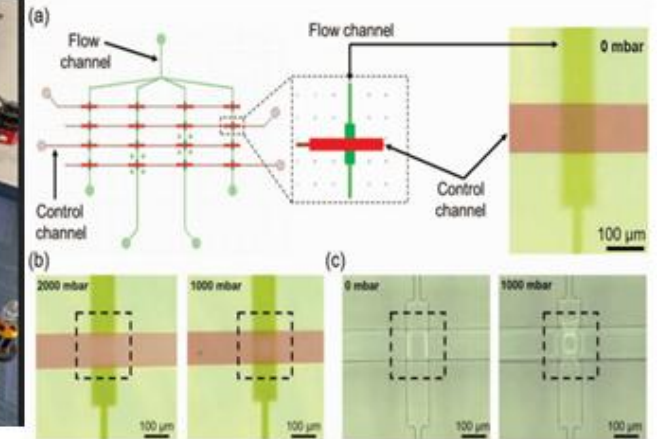
Far Field Side Channel Attacks



EM Side Channel Attacks

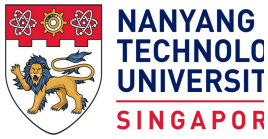


Swarm of Drones Security



Bio Chip Security

SETTLOR Team and Research Collaborators



Roadmap of the Talk

- 1 Background**
- 2 KiD · A unified radix-2 core**
One datapath shared by both schemes. Covered briefly — the idea, the mechanism, the numbers.
- 3 Winograd for NTT · Higher radix, fewer multipliers**
Winograd radix-8 and radix-16 butterflies for Dilithium, FALCON and Kyber, and why they beat Cooley-Tukey where it matters.
- 4 Lightweight radix-4 · The FPGA design space exploration**
BRAM-free radix-4, a 2-multiplier variant, a multiplier-free design via Zech's logarithm, and what each configuration costs.
- 5 Summary**
The three works as three points in one design space.

From the Quantum Threat to Deployed Standards

2017 – 2026

Quantum hardware — Progress

2018

Google Bristlecone: a 72-qubit superconducting processor.



2019

Google claims quantum supremacy on a sampling task.



2020

China claims quantum supremacy with the Jiuzhang photonic machine.



2023

IBM Condor: 1,121 physical qubits — scale, but still no error correction.



Dec 2024

Google Willow (105 qubits): first below-threshold surface code. Logical error falls as the lattice grows from 3×3 to 7×7.



2026

The logical-qubit era. Microsoft + Quantinuum: 12 logical qubits at $\sim 2 \times 10^{-3}$. IonQ Tempo: 64 algorithmic qubits. IBM Kookaburra: $\sim 4,158$ qubits.

NIST post-quantum standardisation — from 69 candidates to fixed targets

2017

Round 1 opens with 69 accepted submissions.



2019

Round 2: 26 candidates remain.



2020

Round 3: 15 candidates remain.



2022

Round 4: 8 submissions carried forward; SIKE broken classically.



Aug 2024

FIPS 203 (ML-KEM), FIPS 204 (ML-DSA) and FIPS 205 (SLH-DSA) published as final standards.



2025 – 27

HQC selected (Mar 2025) as a fifth, code-based KEM. FIPS 206 / FN-DSA draft Aug 2025, final expected late 2026 or early 2027. IR 8547:

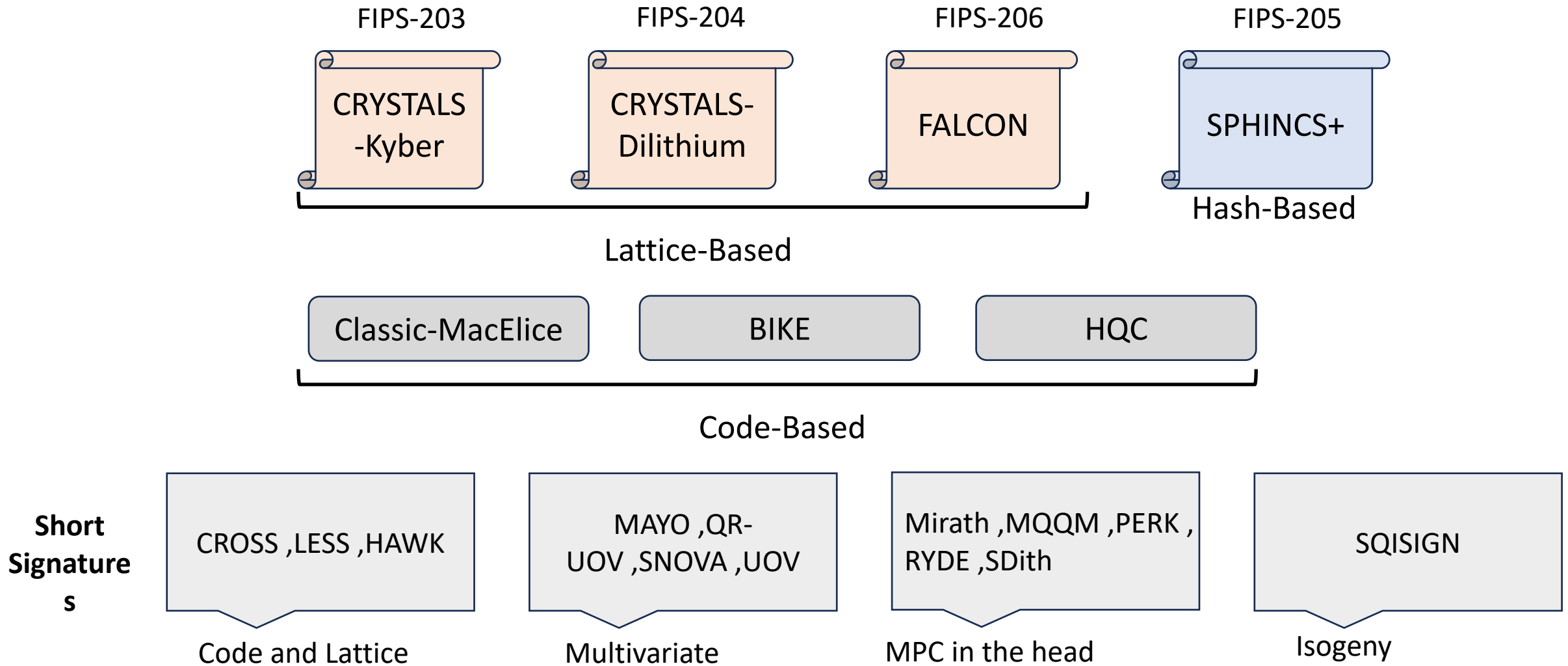
Why this matters here

No cryptographically relevant quantum computer exists yet, but harvest-now-decrypt-later and a 2030 deprecation deadline make integration of PQC an immediate requirement.

Sources: NIST CSRC post-quantum cryptography standardisation project; NIST IR 8547; NSA CNSA 2.0; public vendor announcements, 2018–2026.

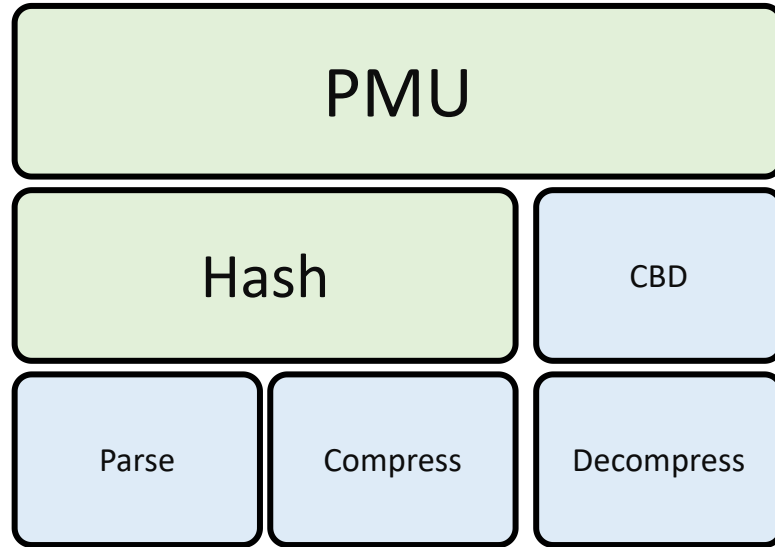
NIST PQC Standardization

No known attacks using classical or quantum computers.

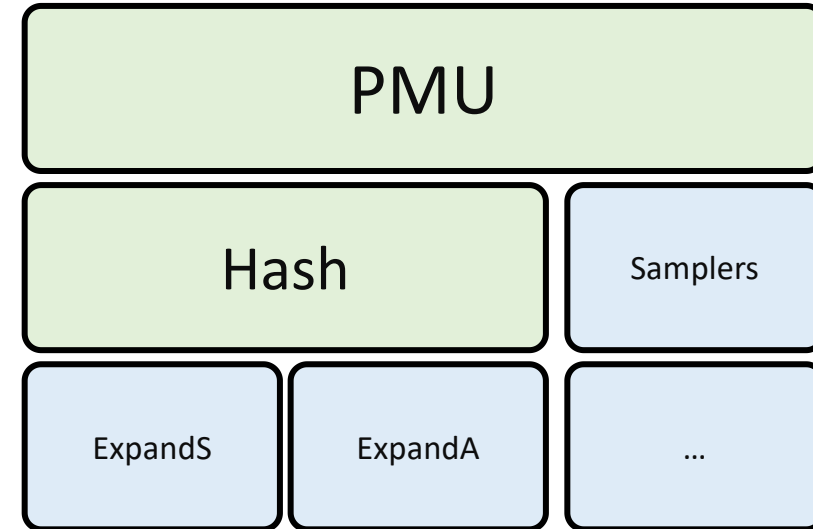


A Hardware Designer's Perspective towards designing Kyber / Dilithium

Kyber



Dilithium



- Faster and Lightweight Design.
- Majority of the operations involves **Polynomial Multiplication Unit and Hashing**.
- For large degree polynomial multiplication, **Number Theoretic Transformation (NTT)** is used.

NTT Multiplication in Kyber / Dilithium

CRYSTALS-Kyber

Polynomial multiplication in Kyber is equivalent to negative wrapped convolution with modulus $q = 3329$ and $n = 256$. But 512^{th} primitive root of unity does not exist for Kyber; rather, Kyber has 256^{th} root of unity. So, Kyber performs incomplete NTT i.e. NTT of odd and even coefficients are computed independently

CRYSTALS-Dilithium

Polynomial multiplication in CRYSTALS-Dilithium is also performed in the ring $\mathbb{Z}_q[x]/(X^n + 1)$ where $q = 8380417$ and $n = 256$. In this case, 512^{th} root of unity exists, and we can deploy complete NTT for Dilithium.

Pointwise Multiplication in NTT domain for CRYSTALS-Kyber

$$\hat{h}_{2i} = \hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1} \cdot \zeta^{2br(i)+1} \quad \hat{h}_{2i+1} = \hat{f}_{2i}\hat{g}_{2i+1} + \hat{f}_{2i+1}\hat{g}_{2i}$$

$$\text{PWM0} : s_0 = \hat{f}_{2i} + \hat{f}_{2i+1}, \quad s_1 = \hat{g}_{2i} + \hat{g}_{2i+1}, \quad m_0 = \hat{f}_{2i} \cdot \hat{g}_{2i}, \quad m_1 = \hat{f}_{2i+1} \cdot \hat{g}_{2i+1}$$

$$\text{PWM1} : s_2 = m_0 + m_1, \quad m_2 = s_0 \cdot s_1, \quad m_3 = m_1 \cdot \zeta^{2br(i)+1}, \quad \hat{h}_{2i} = m_0 + m_3, \quad \hat{h}_{2i+1} = m_2 - s_2$$

Components in NTT/ NTT⁻¹/ PWM

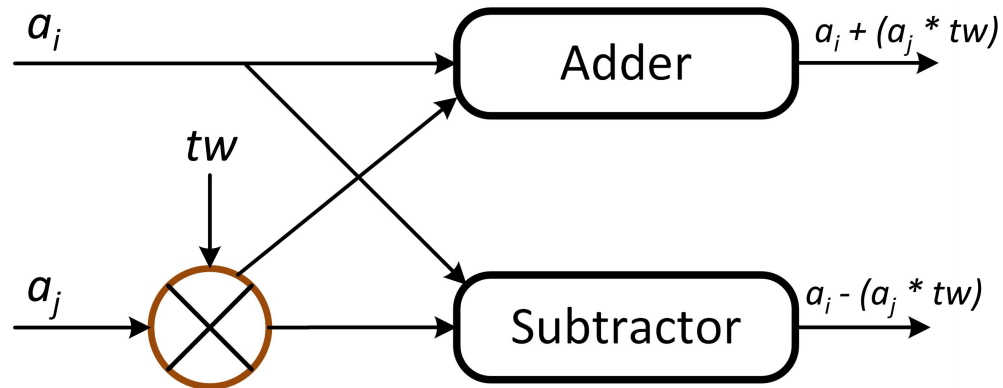
Modular
Adder

Modular
Subtractor

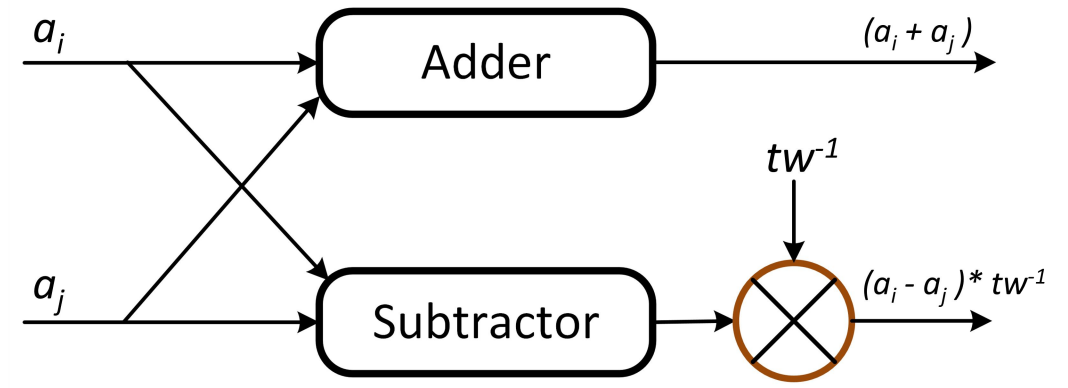
Modular
Multiplier

Memory Unit

Barrett/ Montgomery/ K-Red



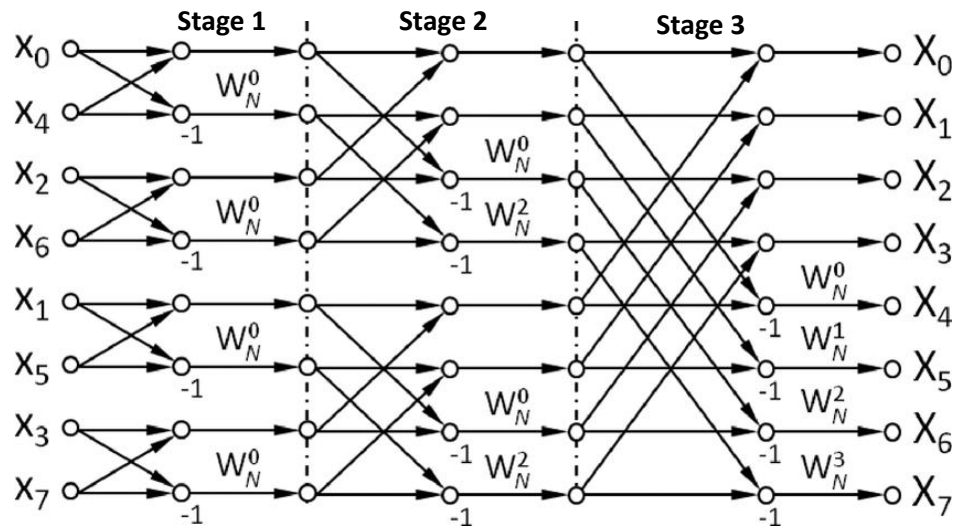
Cooley-Tukey NTT Butterfly Unit



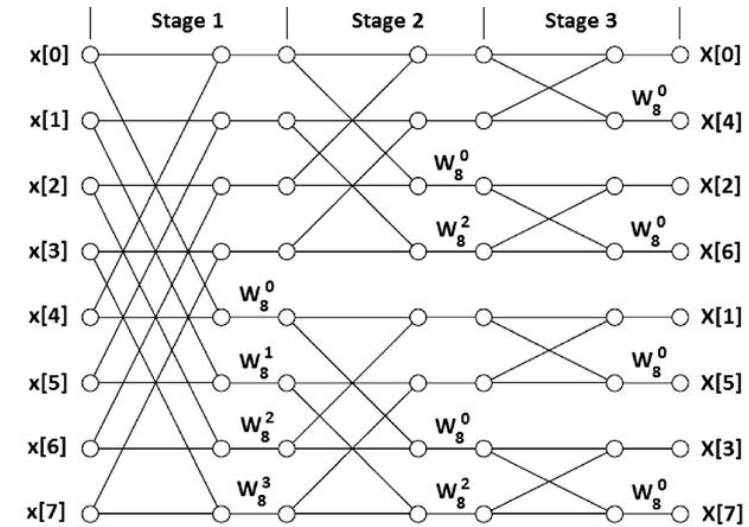
Gentleman-Sande NTT⁻¹ Butterfly Unit

- Radix-2 Cooley-Tukey / Gentleman-Sande Architecture.
- Coefficients are fetched and processed through the BFUs iteration wise.

Challenges in Designing Efficient NTT Architecture



Radix 2 DIT Signal Flow for 8 point NTT, Note the bit reversal of the input



Radix 2 DIF Signal Flow for 8 point NTT, note the bit reversal of the output.

Pipelined
Memory
Access

Avoiding rearrangement
of coefficients between
NTT/ NTT-1

Low Overhead
Modular
Multiplier

Supporting
NTT/NTT⁻¹/
PWM

Multiplication
with N^{-1} during
NTT⁻¹

The observation

ML-KEM: 12-bit coefficients ($q = 3329$) ML-DSA: 23-bit coefficients ($q = 8380417$). Two Kyber coefficients occupy almost exactly the space of one Dilithium coefficient, so a single 24-bit datapath can support either two Kyber BFUs or one Dilithium BFU.

Share the adder, not with muxes

One 24-bit adder/subtractor acts as two Kyber lanes or one Dilithium lane. A control bit at position 12 decides whether the carry crosses the boundary or stops there — which keeps the FPGA fast-carry chain intact, where earlier unified work lost it to multiplexers.

Size the multiplier once

Two DSP blocks configured as 23×12 give either two independent Kyber coefficient multiplications, or one Dilithium multiplication once the partial products are summed. Nothing sits idle in either mode.

Reduce with the prime's structure

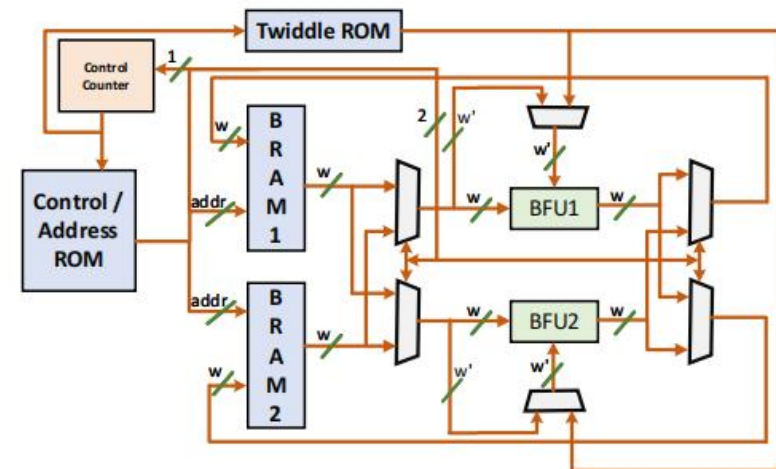
Montgomery reduction specialised to $q = 2^{11} + 2^{10} + 2^8 + 1$ and $q = 2^{23} - 2^{13} + 1$, so the constant multiplications collapse to shifts and adds: 71 LUTs and 1 DSP per Kyber multiplication.

Let the memory map do the work

Two simple dual-port RAMs with a ping-pong address sequence, precomputed offline into a ROM. Conflict-free and stall-free while pipeline depth $\leq (\text{coefficient memory depth})/2$.

Result

A single core covering both standardised lattice schemes: 1384 LUT, 2 DSP and 4.5 BRAM at 387 MHz on Zynq UltraScale+ — a better area-delay product than the existing unified design, and than the two standalone cores it replaces.



Scaling the Framework: Trade-offs in Area and Latency

	Design 1	Design 2	Design 3
Design Configuration	2 Kyber / 1 Dilithium BFUs	4 Kyber / 2 Dilithium BFUs	8 Kyber / 4 Dilithium BFUs
Hardware Footprint (LUTs / DSP / BRAM)	1384 / 2 / 4.5	2893 / 4 / 4.5	5909 / 8 / 5.5
Latency (Kyber / Dilithium NTT cycles)	448 / 1024	224 / 512	112 / 256
Frequency Constraint	387 MHz	342 MHz	294 MHz

Winograd for NTT: A Case Study on Higher-Radix and Low-Latency Implementation of NTT for Post Quantum Cryptography on FPGA (TCAS-I)

Suraj Mandal, Debapriya Basu Roy

Department of Computer Science & Engineering

Indian Institute of Technology Kanpur



Design Challenge

Radix-2/4 Scaling

Results in unacceptably high latency for time-critical cryptographic applications.

Parallel BFU Replication

Reduces latency but introduces massive additive increments in area and memory overhead.

Higher-Radix CT/GS

Exponentially increases circuit complexity, leading to longer critical paths and stalling.

The ATP Trap

Higher-radix Cooley-Tukey designs inevitably yield an unfavourable Area-Time Product (ATP).

Why Winograd? Trade Multipliers for Adders

Cooley-Tukey does not scale to higher radix

- Raising the radix cuts latency, but the multiplication count and the critical path rise with it: a radix-16 Cooley-Tukey stage needs $32 \times N/16$ modular multiplications, 17 of them inside the butterfly.
- Deeper butterflies lengthen the critical path, and pipeline depth is capped at $(\text{memory depth})/R$ — so the design cannot be pipelined out of the problem.
- The outcome in the literature: higher-radix Cooley-Tukey designs post a worse area-time product than the slower radix-2 designs they were meant to beat.

Winograd restructures the butterfly

- Several layers of Hadamard gates surround a single multiplication layer, instead of interleaving multiplications through every stage.
- Radix-16 needs 13 internal multiplications instead of 17, and radix-8 needs only 4 — $28 \times N/R$ and $11 \times N/R$ in total.
- A modular multiplier costs a DSP pair or hundreds of LUTs; an adder costs a fraction of that. Trading multiplications for adders is a favourable exchange.
- This is the first work to apply Winograd's FFT technique to NTT for lattice-based post-quantum cryptography.

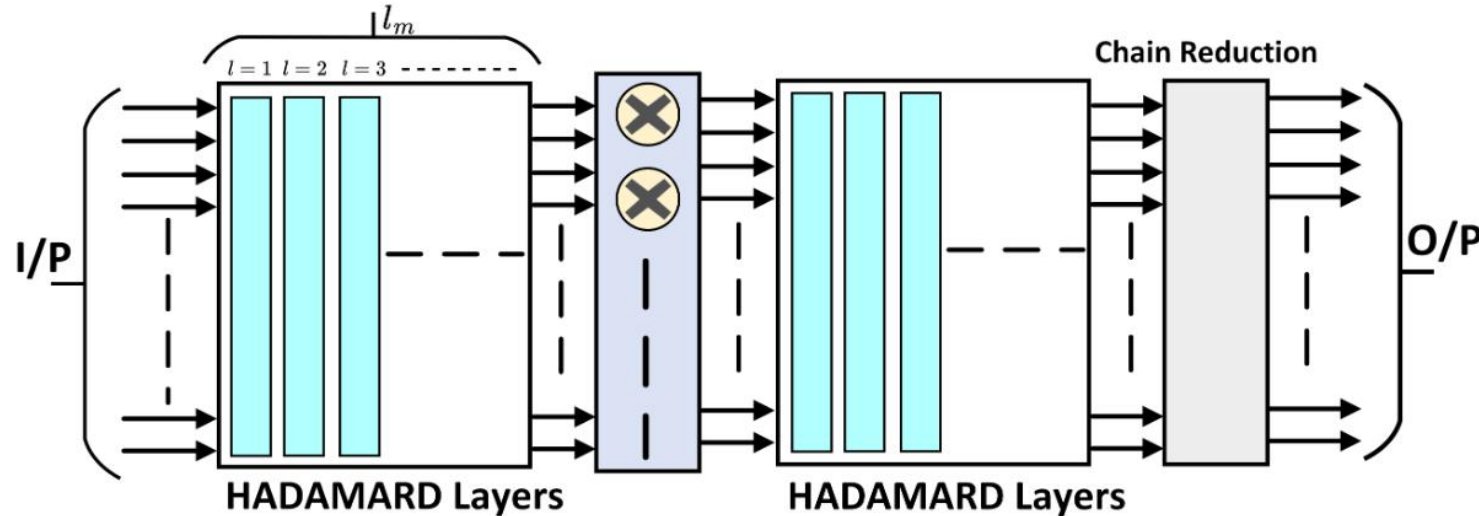
The price, and how it is paid. Winograd needs more modular adders and subtractors, and every Hadamard layer widens the operands by one bit. Both are absorbed: the redesigned Hadamard gate is cheaper than the generic one (47 LUTs at 578 MHz against 77 LUTs at 346 MHz), and the multipliers and the chain-reduction layer are sized for the widened operands.

Measured on the radix-16 Dilithium NTT unit

12141 LUTs and 50 DSPs at 254 MHz, against 17296 LUTs and 64 DSPs at 163 MHz for the Cooley-Tukey/Gentleman-Sande equivalent — an area-time product of 2772 versus 6154, at the same 58-cycle latency.

Winograd's BFU for Higher Radix NTT

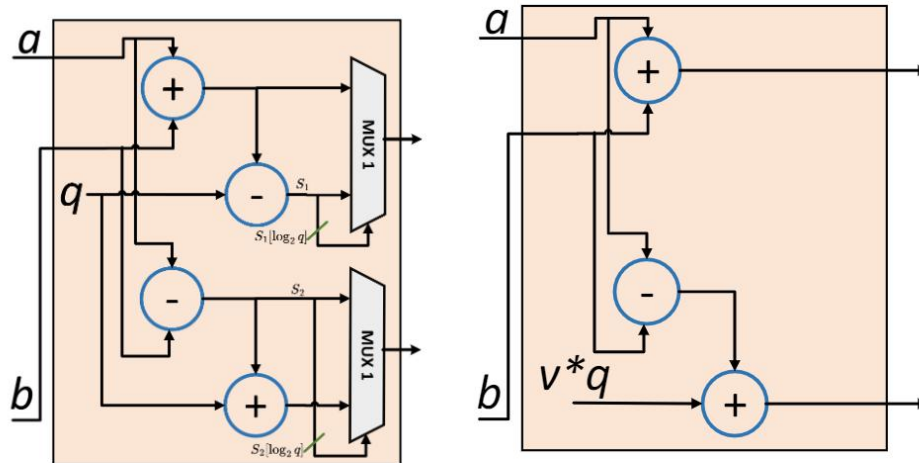
Generic Winograd BFU



Winograd's BFU requires fewer number of modular multipliers than Cooley-Tukey or Gentleman-Sande for higher radix BFU

What we proposed:

- 1) Modified Hadamard layer
- 2) Modified K-RED multiplier
- 3) Chain Reduction Module



(a) Generic Hadamard Gate

(b) Our Modified Hadamard Gate

Modified Hadamard Gate: No modular reduction

Hadamard Gate	Width of q ($\lceil \log_2 q \rceil$)	LUT	Frequency (Mhz)
Generic	23	77	346
Proposed		47	578

K-RED Algorithm for Modular Multiplication for Dilithium

Algorithm 6 K-RED Algorithm

- 1: **Input:** $C = a \times b$
- 2: **Output:** $K \times C \bmod q$
- 3: $d_0 = C[m : 0]$, $d_1 = C \gg m$
- 4: return $(K \times d_0 - d_1)$

Algorithm 7 Modified K-RED Algorithm for Dilithium

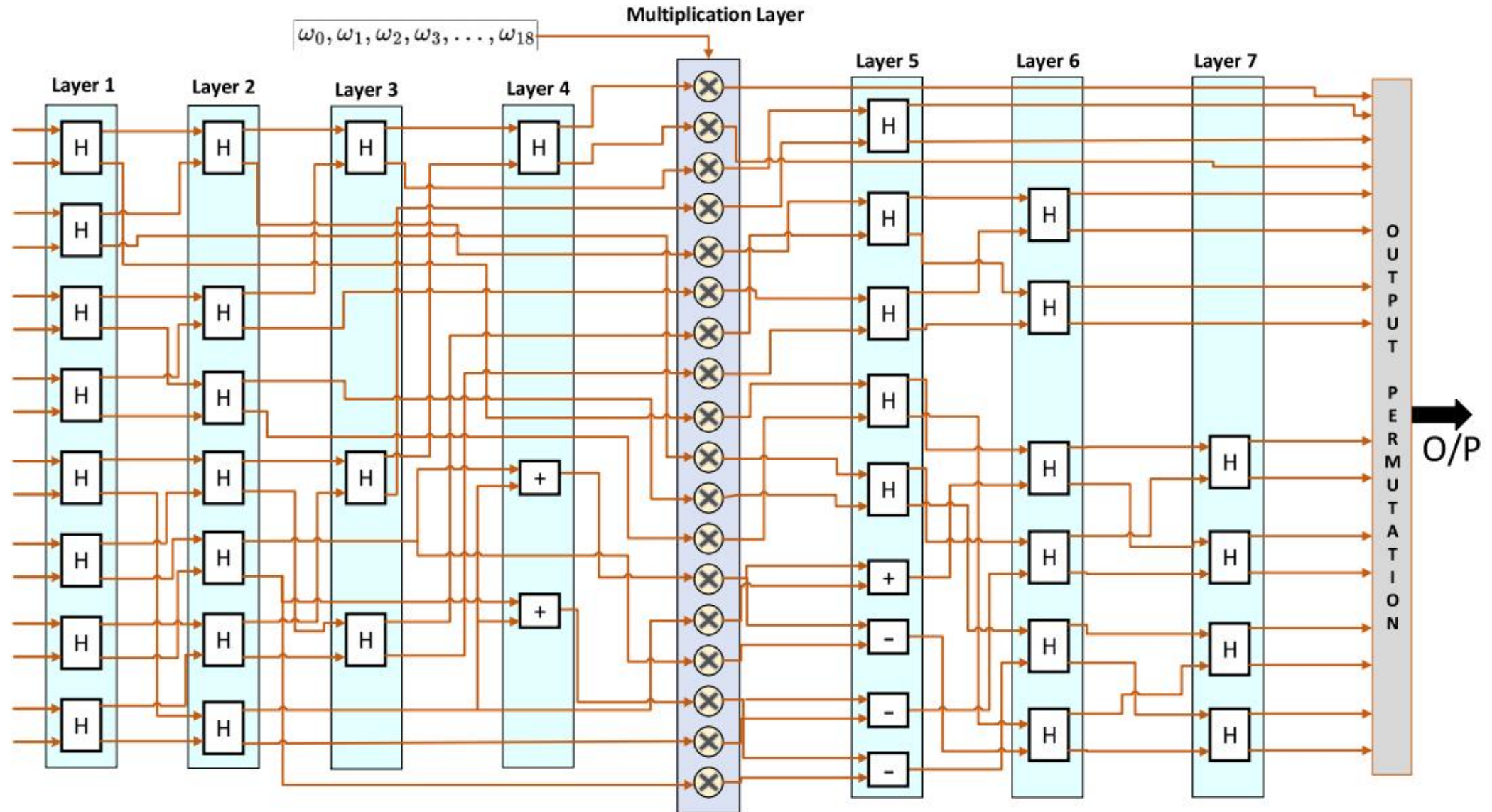
- 1: **Input:** $C = a[22 : 0] \times b[26 : 0]$
- 2: **Output:** $K'_{Dilithium} \times C \bmod q$, where $K'_{Dilithium} = 256^{-1}$
- 3: $c_{32} = C \ll 5$, $cl = c_{32}[22 : 0]$
- 4: $ch_1 = c_{32} \gg 23$, $ch_2 = ch_1 \gg 23$, $ch_3 = ch_1[22 : 0]$
- 5: $c_1 = cl - ch_1 + (ch_2 \ll 26) - (ch_2 \ll 13) + (ch_3 \ll 13)$
- 6: return $K - RED(c_1)$

Reduction Method	DSP	LUT	FF	Frequency (Mhz)
Montgomery [13]	2	235	42	285
Solinas [10]	2	260	36	N/A
Proposed K-RED	2	155	128	303

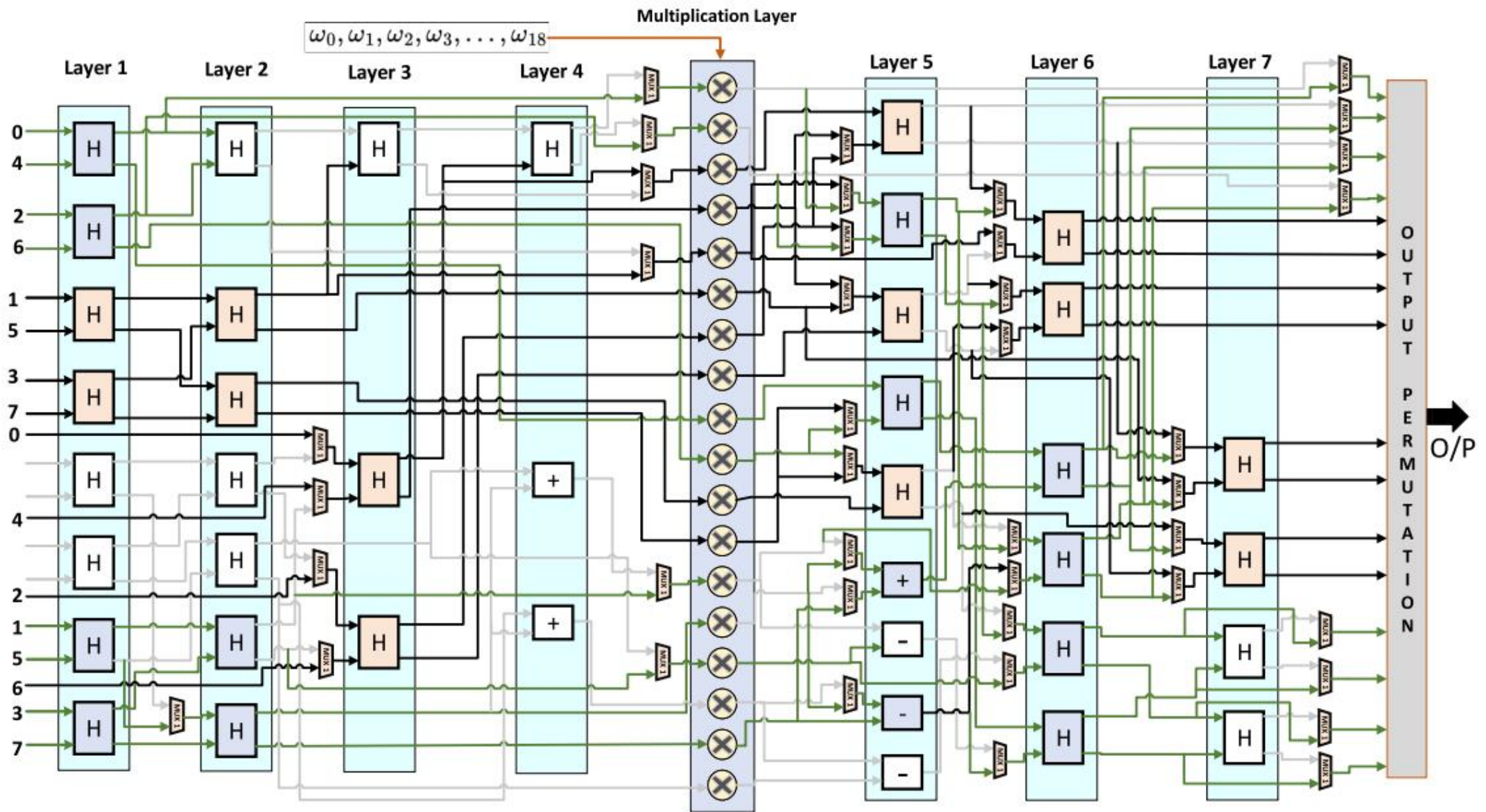
Result: Low Area Modular Multiplier with low critical path compared to traditional coefficient multiplier

- The generic version of the K-RED Algorithm takes two inputs a and b and produces output $(K \times C) \bmod q$ where $q = (K \times 2^m) + 1$.
- The modified K-RED algorithm for Dilithium produces $(K'_{Dilithium} \times C) \bmod q$ where $K'_{Dilithium} = 256^{-1}$.
- Due to the optimizations applied on Winograd's radix-16 BFU, for Dilithium, proposed modular multiplier is designed to handle a multiplication of one 23-bit operand coming from the twiddle factors and another 27-bit operand.
- During NTT, we have modified each twiddle factors by multiplying with $K'^{-1}_{Dilithium}$ to nullify the effect of $K'_{Dilithium}$.
- During PWM operation, $K'_{Dilithium}$ is multiplied while performing the reduction. So, we don't need multiplication with 256^{-1} during inverse NTT

Winograd Radix-16 BFU used for Dilithium (degree 256)

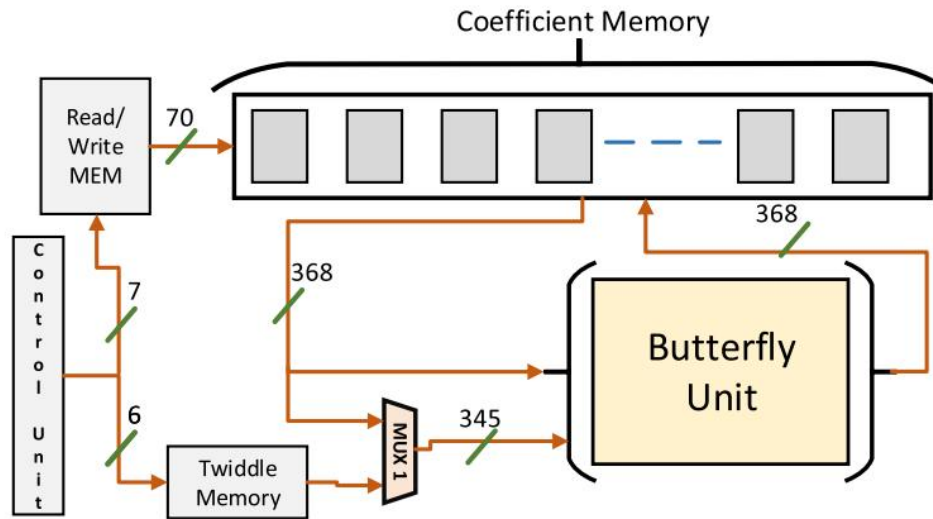


Winograd Mixed Radix-16/8 BFU used for Kyber (degree 128)

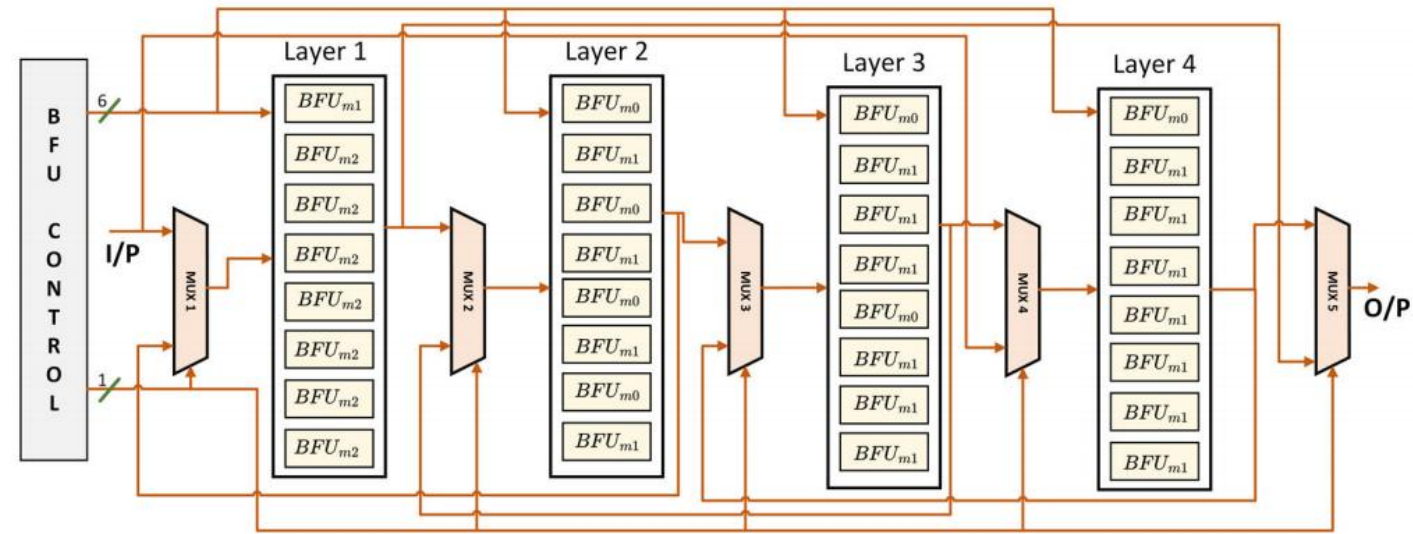


Winograd vs Cooley-Tukey

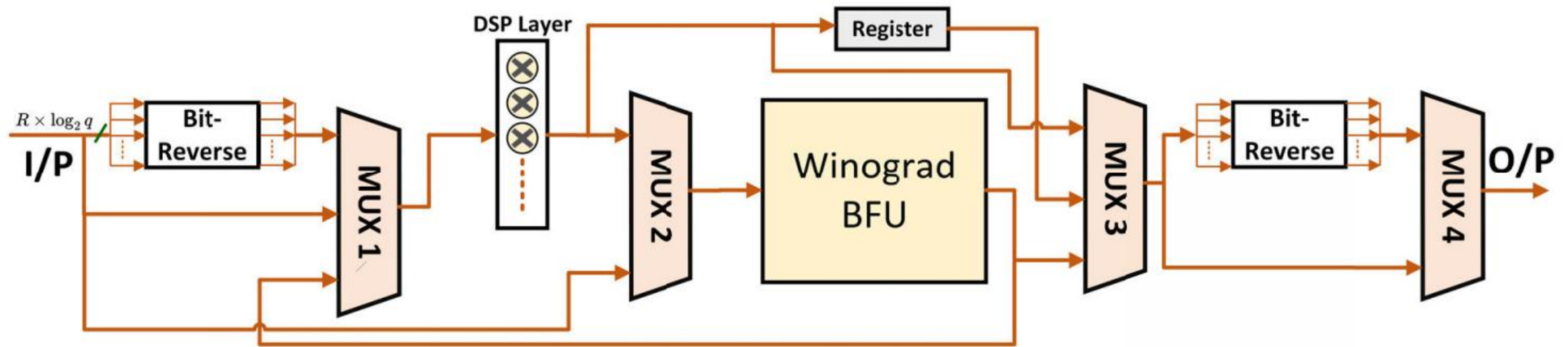
Radix-16 NTT multiplication unit architecture.



Radix-16 Cooley-Tukey/Gentleman-Sande BFU.



Winograd BFU configured for NTT/NTT-1 /PWM.



Winograd vs Cooley-Tukey

BFU	DSP	LUT	FF	Frequency (Mhz)	Latency			ADP-LUT	ADP-DSP
					NTT	NTT	PWM	NTT/NTT	NTT/NTT
Cooley-Tukey	64	17296	10218	163	58	58	44	6154	23
Winograd	50	12141	7941	254				2772	12

Result for CRYSTALS-Dilithium

[illegible]

Result for FALCON

NTT Type	Design	Board	LUT	slice	FFs	BRAM	DSP	Latency (NTT)	Latency (INTT)	Latency (PWM)	Freq.(MHz)	AT
Radix-2	[24]	Artix-7	11000	2750	5182	12	64	318	391	1100	140	20784
			63000	15750	18000	48	256	126	141	428	126	41350
Radix-8 (Winograd)	TW	Zynq 7000	5267	1520	1871	7	12	228	228	76	217	2858
		ZCU+ 102	4995	839	1582						269	1728
		Artix-7	5490	1540	1781						144	4338

Result for CRYSTALS-Kyber

NTT Type	Design	Board	LUT	Slice	FFs	BRAM	DSP	Latency (NTT)	Latency (INTT)	Latency (PWM)	Freq.(MHz)	AT
Radix-2/4	[22]	zynq7000	5293	1323	5518	-	34	127	127	-	265	2263
	[23]	Artix-7	1740	575	643	5	4	224	224	64	200	1092
	[11]	Kintex U+	24186	3023	14756	24	48	100		-		3912
Radix-8/16 (Winograd)	TW	Zynq 7000	9824	2745	4752	12	29	58	58	44	181	1809
		ZCU 102+	9083	1520	3839						252	1017
		Artix-7	9985	2983	4580						132	2585

Result for Homomorphic Encryption for 4096 degree Polynomial

NTT Type	Design	Board	LUT	Slice	FFs	BRAM	DSP	Latency (NTT)	Latency (INTT)	Latency (PWM)	Freq.(MHz)	AT
Radix-2	[26]	Virtex U+	7906	988	3973	24	32	3080	3080	-	271	47598
Radix-16	TW		24341	3798	11928	35	104	800	800	544	253	44895

Winograd for NTT: The Workflow

01

Construct the butterfly

Derive Winograd radix-8 and radix-16 BFUs: layers of Hadamard gates wrapped around one multiplication layer, rather than Cooley-Tukey's cascade of multiply-add stages.



02

Redesign the Hadamard gate

Compute $a + b$ and $a - b + v \cdot q$, with v chosen per layer so every output stays positive. This removes the modular reduction and the multiplexer from each gate.



03

Absorb the bit growth

Each layer widens the operands by a bit, so the multiplier accepts $(\lceil \log_2 q \rceil + k) \times \lceil \log_2 q \rceil$ inputs, and a chain-reduction layer returns the outputs to mod q at the end.

04

Specialise the modular reduction

Modified K-RED for Dilithium, with $K' = 256^{-1}$ so NTT^{-1} gets its n^{-1} for free; Montgomery K-RED for FALCON and Kyber, avoiding conversion into and out of the Montgomery domain.



05

Feed it without stalling

R coefficient BRAMs, read/write addresses precomputed into a ROM, and a stage-wise shifted mapping — with stalls inserted only where the pipeline depth exceeds $(\text{memory depth})/R$.



06

Validated on 3 Architectures

Radix-16 for Dilithium ($n = 256$), radix-8 for FALCON ($n = 512$), and a radix-16 BFU reconfigurable as two parallel radix-8 units for Kyber's incomplete NTT.

58 clock cycles per NTT for Dilithium, with a lower area-time product than both the radix-16 Cooley-Tukey equivalent and the existing radix-4 designs. The same framework is then carried to FALCON, to Kyber's mixed-radix incomplete NTT, to 4096-degree FHE parameters, and to a 45 nm ASIC synthesis.

Design Exploration of Lightweight Radix-4 NTT Multiplication Unit for ML-KEM and ML-DSA on FPGA (FPL 2026)

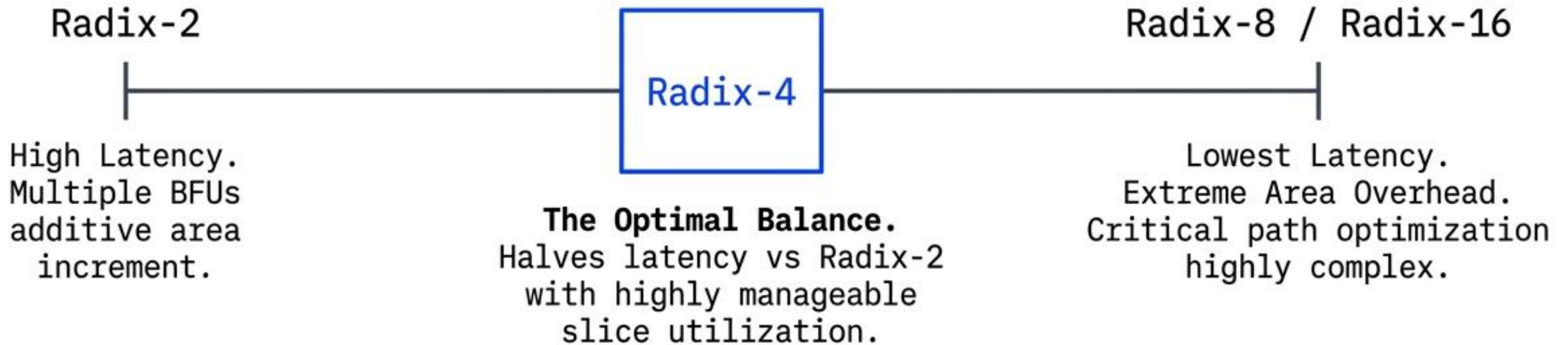
Suraj Mandal, Neha Chaurasia, Debapriya Basu Roy

Department of Computer Science & Engineering

Indian Institute of Technology Kanpur



Motivation & Problem Statement



Contributions



Lightweight, BRAM/DSP-free radix-4 BFU

- Unified NTT / NTT^{-1} / PWM datapath for ML-KEM and ML-DSA. Achieves the lowest area-delay product (ATP) of any design in the literature.



Two-multiplier radix-4 variant

- Trades 2 of the 4 multipliers for a simple ping-pong memory scheme — a low-BRAM option for memory-constrained deployments.



First multiplier-free NTT via Zech's logarithm

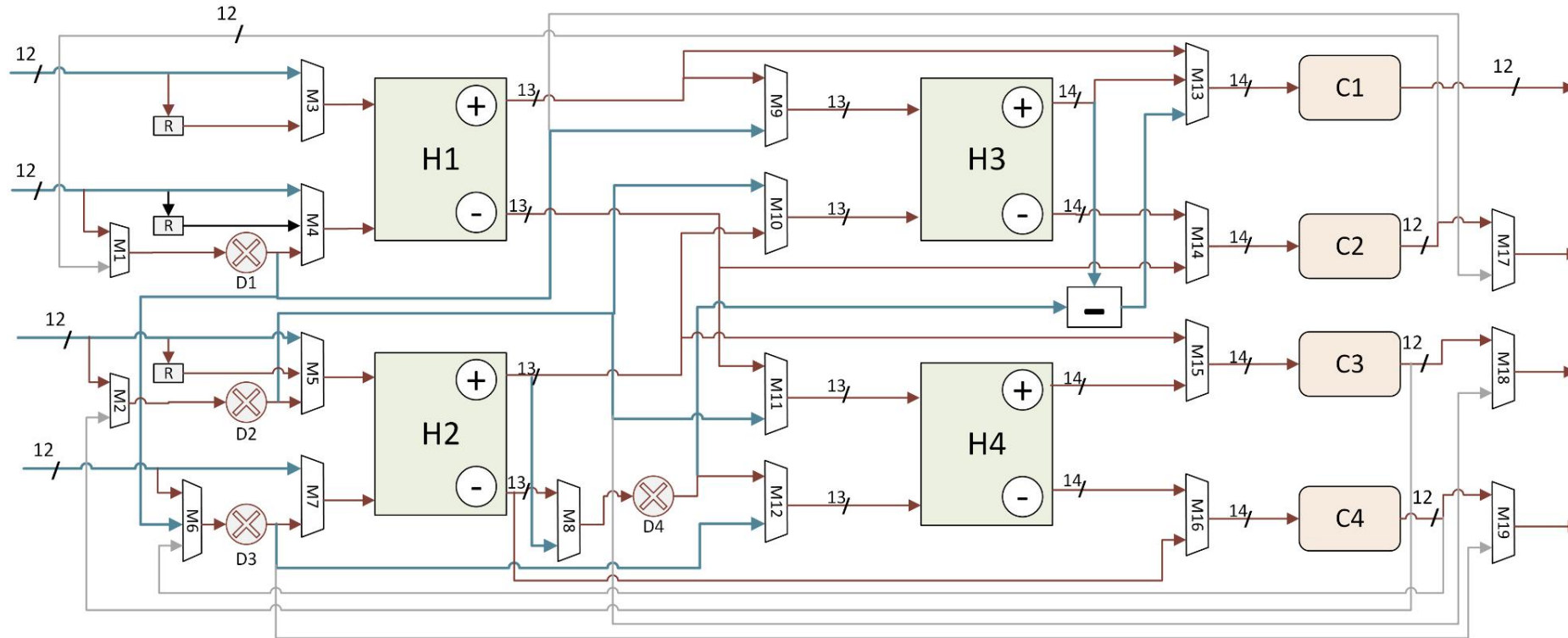
- Replaces modular multiplication with modular addition/subtraction for ML-KEM — eliminating DSP blocks entirely.



Systematic BRAM / DSP / LUT design-space study

- A comprehensive framework and trade-off analysis across resource configurations for radix-4 NTT on FPGA.

Architecture I — BRAM/DSP-Free Radix-4 BFU

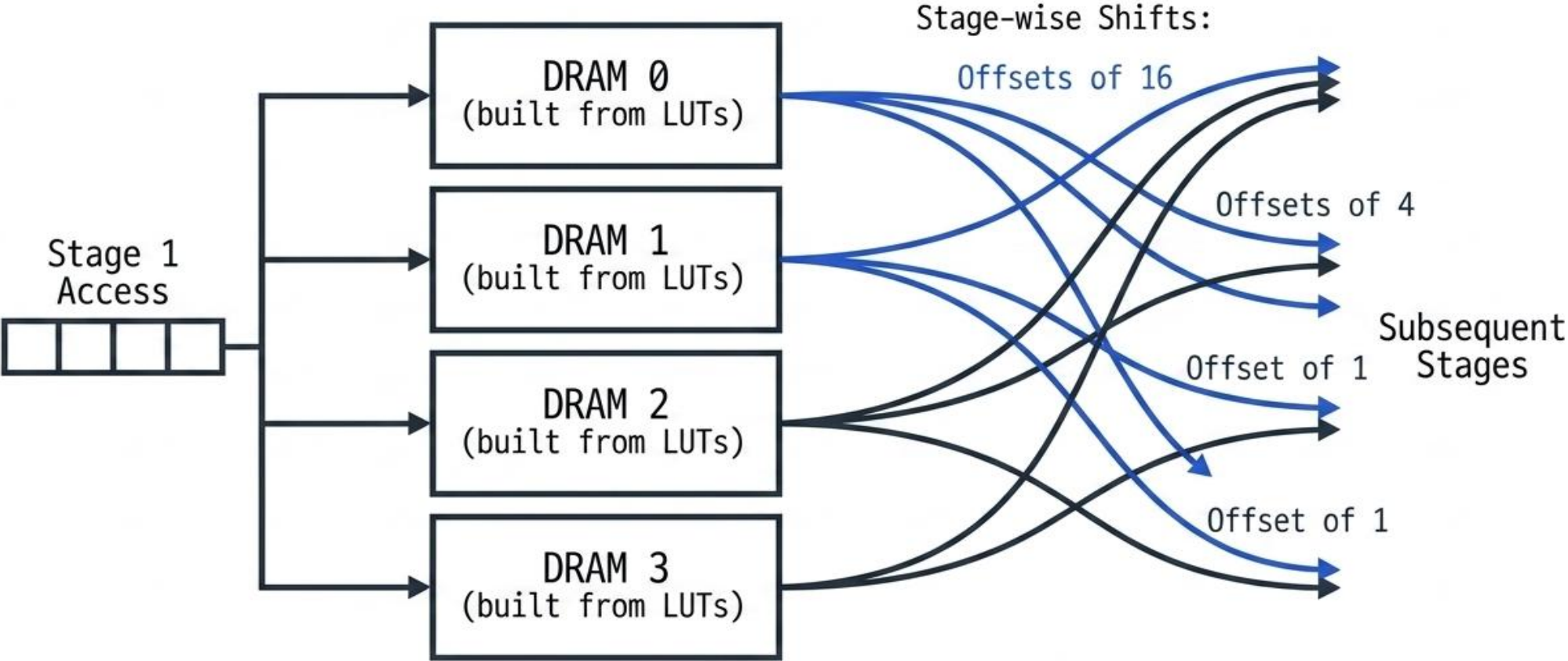


Multiplexer Network (M1-M19): Dynamically routes for NTT vs PWM modes.

DSP Optimization: 4 parallel multipliers (D1-D4).

Chain Reduction nodes (C1-C4) locking final outputs to Z_q .

Conflict-Free Memory Mapping Scheme



Design-Space Exploration: The Area-Time Product

$$ATP = (LUT + DSP \times 400 + BRAM \times 800) \times (Latency / Frequency)$$

Four resource configurations explored per target, trading BRAMs $\leftrightarrow$ DSPs $\leftrightarrow$ LUTs:

C1

DSPs only, 0 BRAM

C2

DSPs + BRAMs

C3

0 DSP, 0 BRAM (LUT-based
mult.)

C4

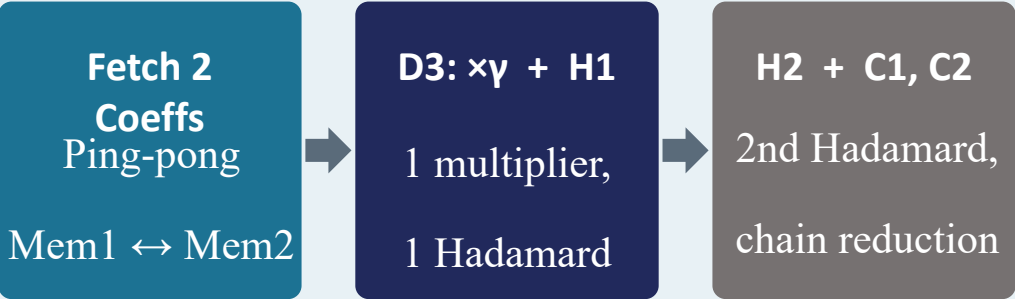
Fewer DSPs + BRAMs

Findings

- ML-KEM: the DSP-free, BRAM-free configuration wins — its 14×12-bit multiply is cheap in LUTs (~154 LUTs), beating a full DSP block
- ML-DSA: the BRAM-free config with 4 DSPs wins — the 25×6-bit portion of its coefficient multiply is cheaper in LUTs than in a second DSP
- General guidance: choose DSP-based configs when resources are free (DSPs are optimized for multiplication); choose LUT-only configs when DSPs/BRAMs are reserved for other accelerators

Alternative Architecture: The 2-Multiplier Design

2-Multiplier BFU Datapath (per pass)

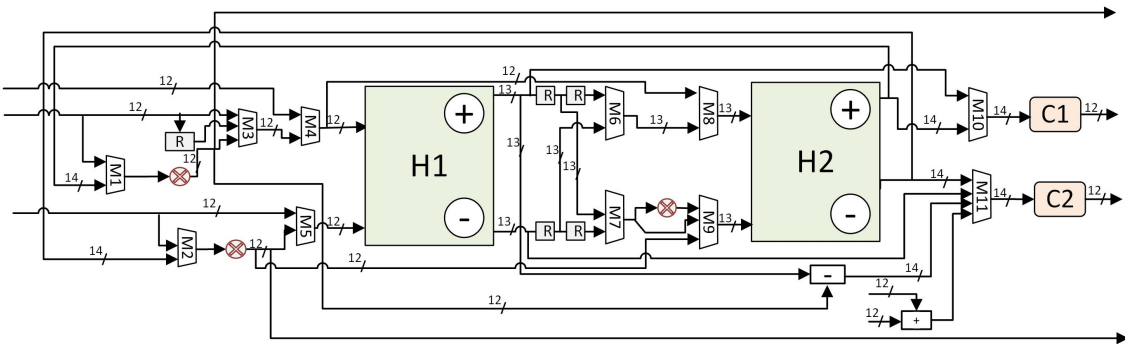


- Halves multiplier count (2 vs. 4) by processing 2 coefficients per pass instead of 4.
- Trade-off: reads stall until the previous pass's writes finish — breaks full pipelining, ~2× latency.
- Best fit: edge devices where BRAM is strictly capped and area must be minimized.

Area Overhead (Table II)

Design	LUT	FF	BRAM	DSP	Freq
ML-KEM*	1153	409	1	2	370 MHz
ML-DSA*	1939	665	2	4	278 MHz
ML-KEM	830	371	2.5	2	333 MHz
ML-DSA	1390	667	4.5	4	270 MHz

- Latency: 556 / 256 / 556 cycles (NTT / PWM / NTT⁻¹) for ML-KEM; 556 / 128 / 556 for ML-DSA — roughly 2× the 4-multiplier design due to pipeline stalls.



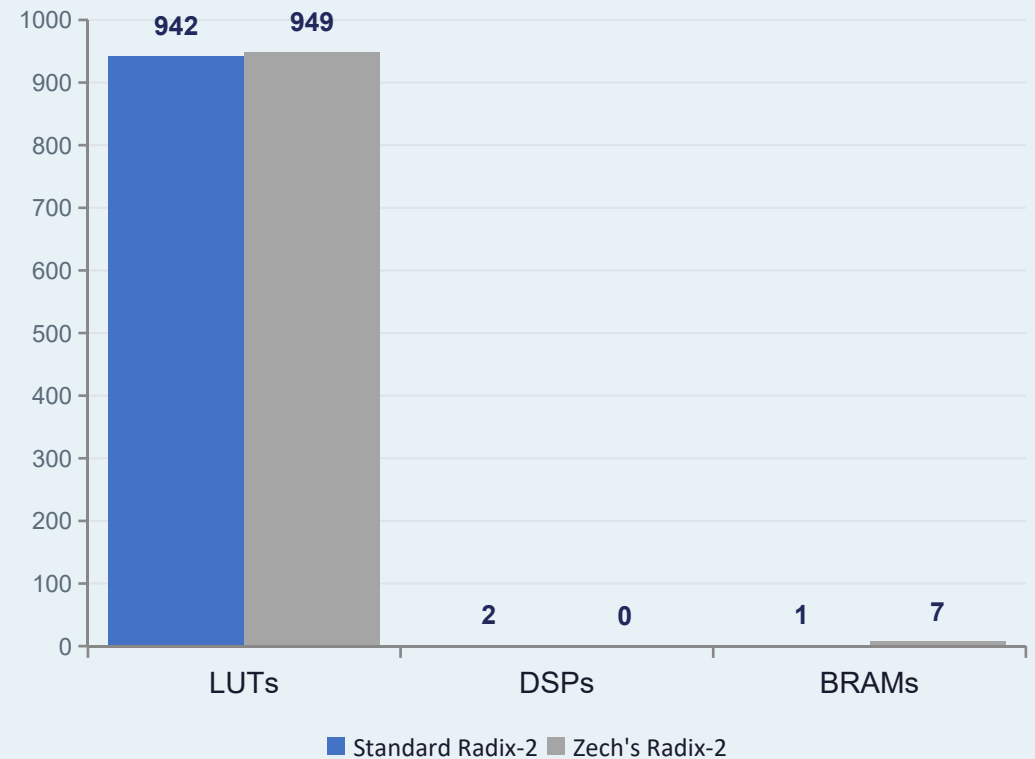


First for NTT hardware

- Represents coefficients as powers of a primitive element α : $x = \alpha^i \mod q$
- Multiplication becomes addition: $(i+j) \mod (q-1)$ — eliminates modular multipliers entirely (only adder/subtractor needed)
- Coefficient width limitation: viable for ML-KEM (12-bit) via a compact BRAM mapping table — infeasible for ML-DSA (23-bit) due to exponential memory scaling
- First reported multiplier-free NTT architecture in the literature

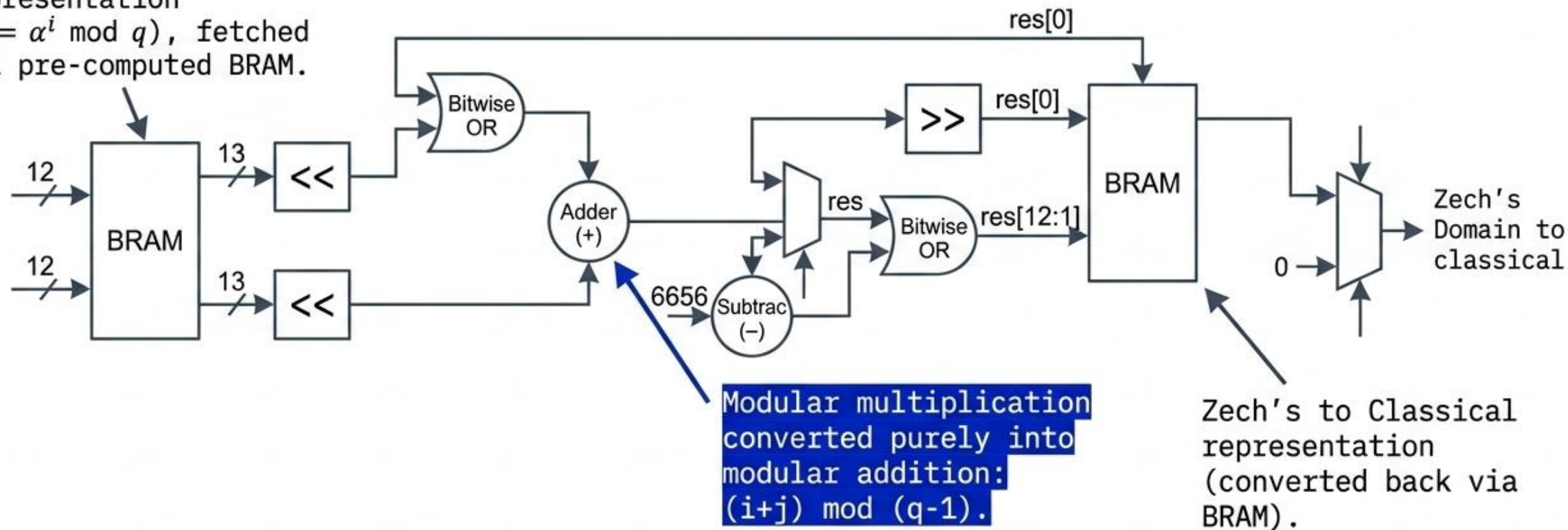
Resource Profiling: Memory Swap

Standard Radix-2 vs. Zech's Radix-2 (ML-KEM)



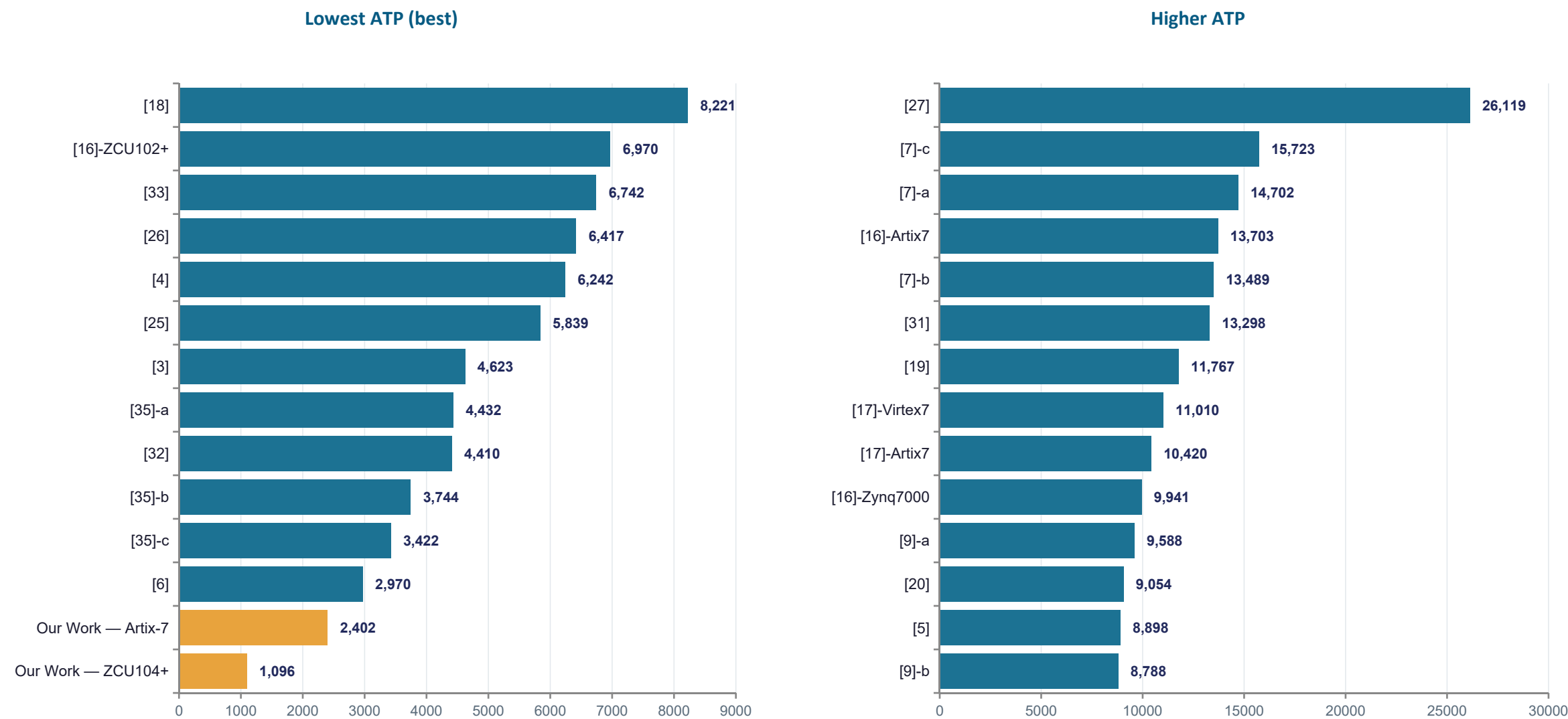
Architecture III: Multiplier-Less Zech's Logarithm (ML-KEM)

Classical to Zech's representation
($x = \alpha^i \bmod q$), fetched
via pre-computed BRAM.

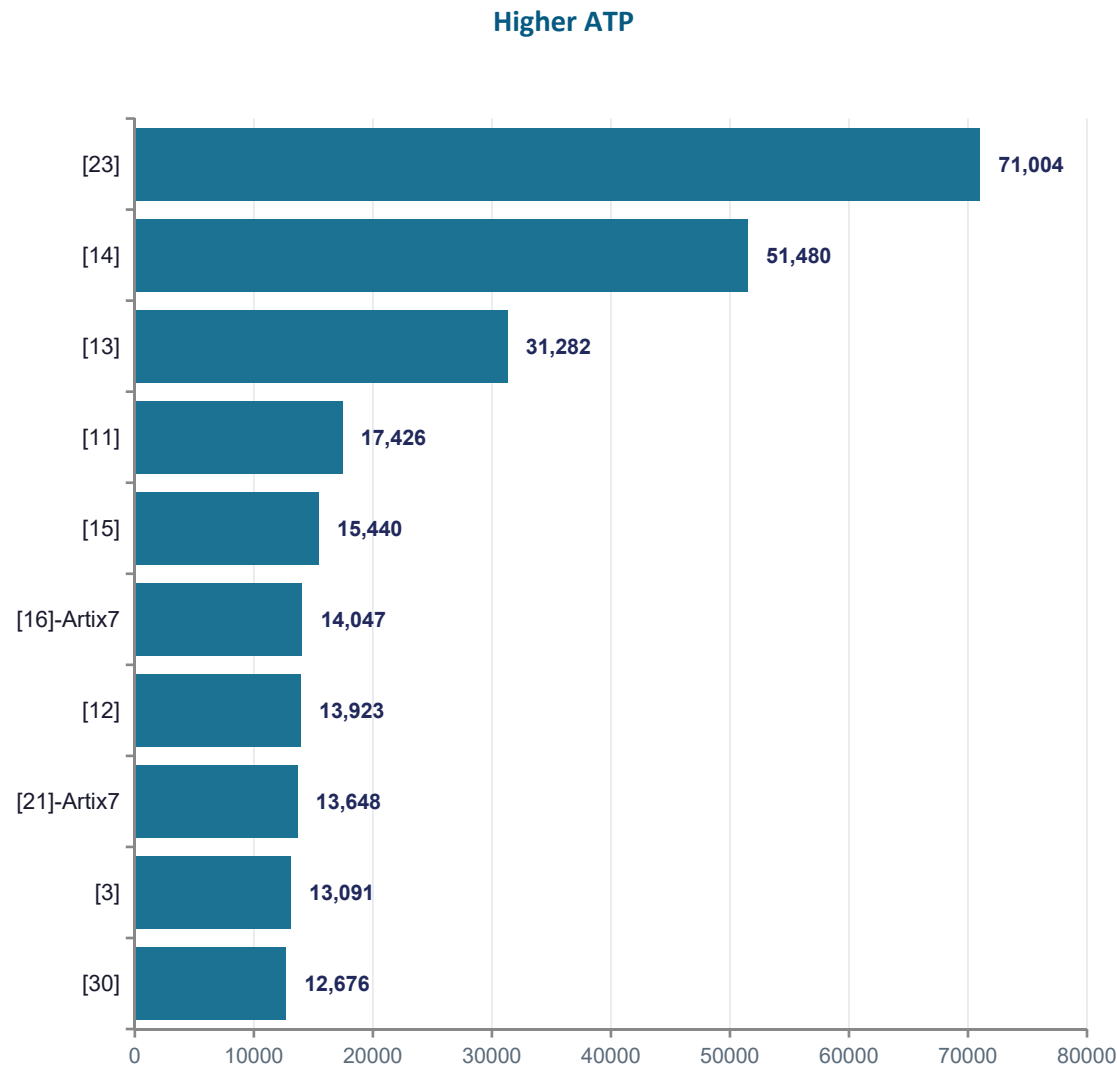
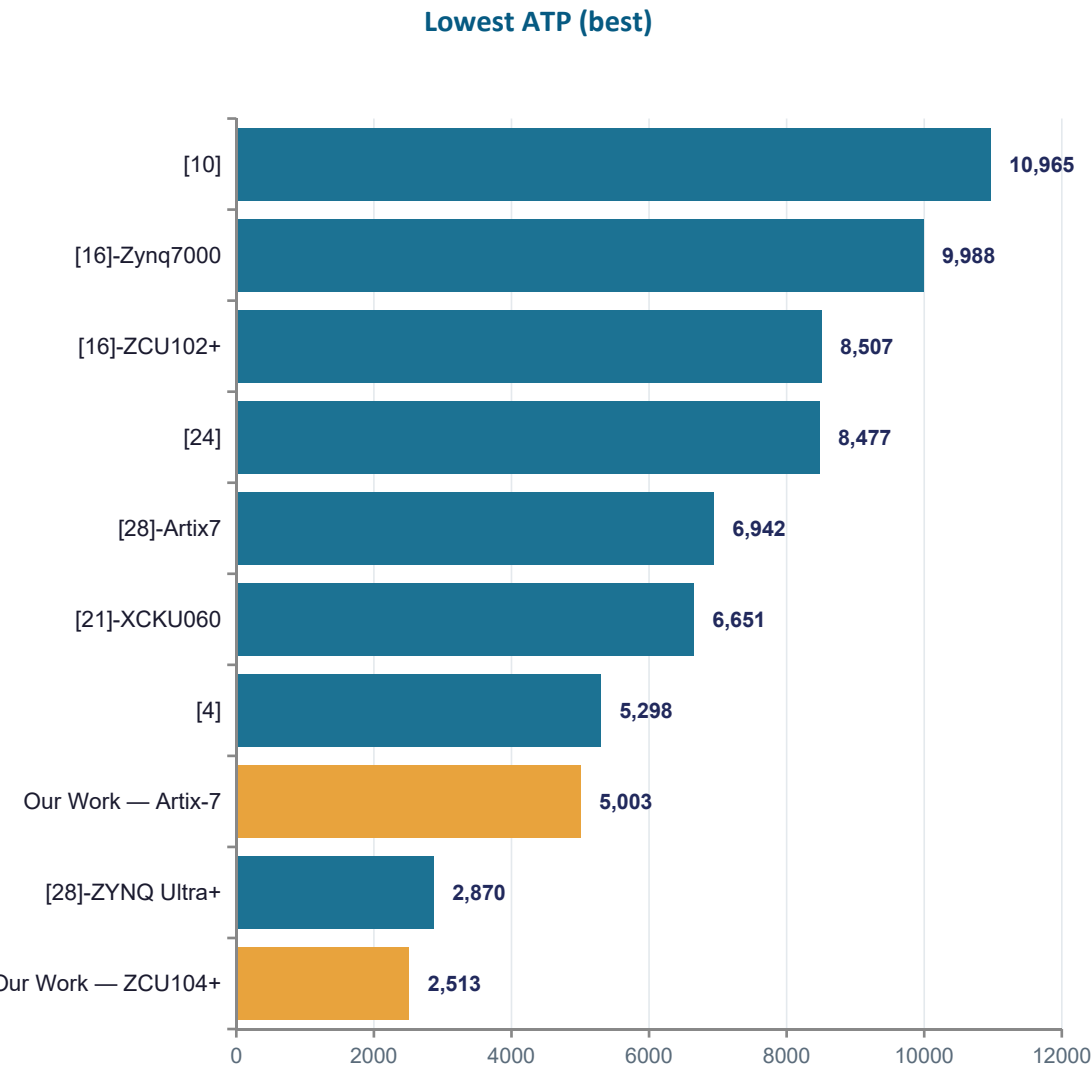


Coefficient width limitation: Viable for ML-KEM (12-bit) via BRAM mapping, but infeasible for ML-DSA (23-bit) due to exponential memory mapping scale.

ATP Comparison — (ML-KEM)



ATP Comparison — (ML-DSA)



Architectural Selection

BRAM-Free (High Performance)

Mechanism: Shifting addressing over DRAMs.

Ideal For: Standard implementations demanding minimum Area-Time Product. Time-critical applications.

Constraint: Highest DSP utilization (**4** for KEM, **8** for DSA).

2-Multiplier (Low Memory)

Mechanism: Ping-Pong dual-port mapping.

Ideal For: Edge devices where BRAM is strictly capped and area must be minimized.

Constraint: **2x** latency penalty due to read/write pipeline stalls.

Zech's Logarithm (Multiplier-Less)

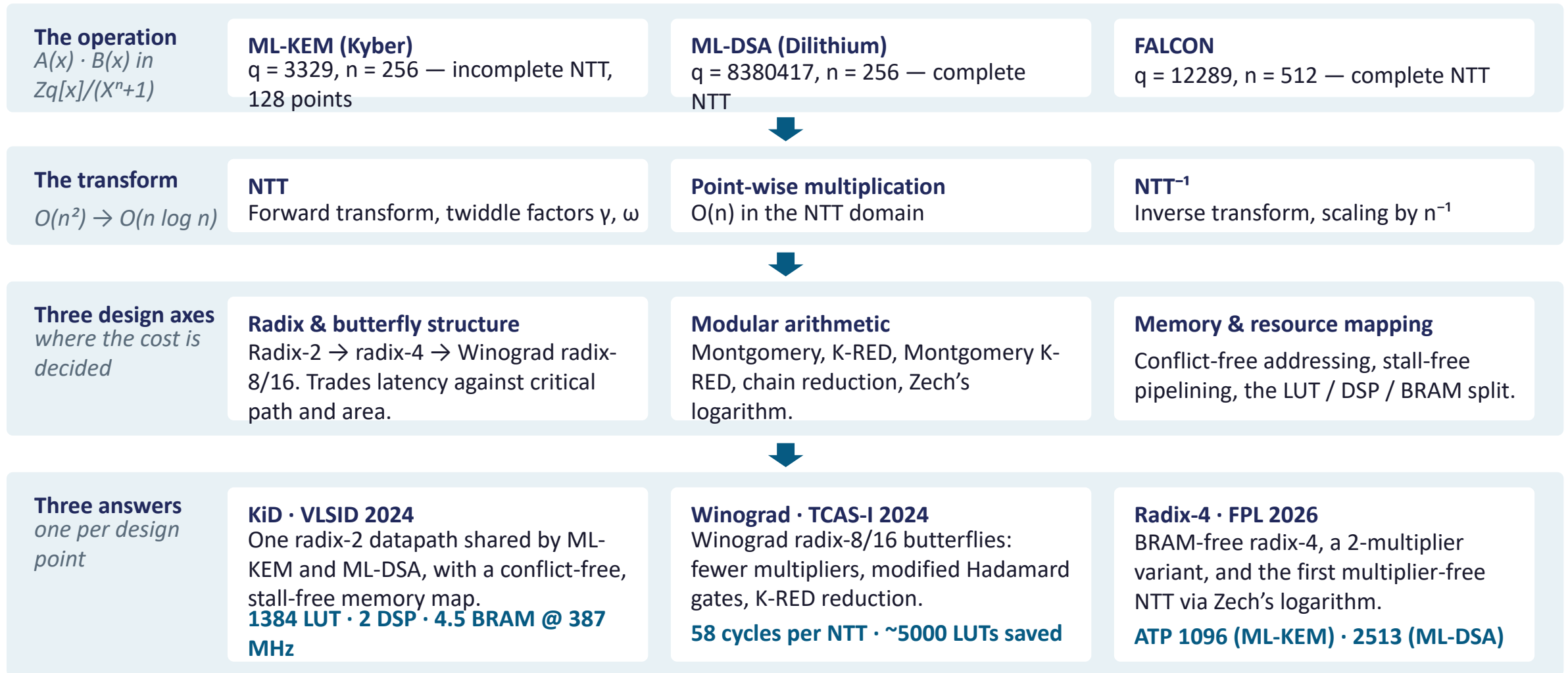
Mechanism: Modular addition via logarithmic mapping.

Ideal For: DSP-starved environments handling ML-KEM exclusively.

Constraint: High BRAM mapping required.
Incompatible with ML-DSA 23-bit width.

FPGA resources are not a monolith; architecture must map specifically to target constraints.

The Complete Flow: Polynomial Multiplication with an Optimised NTT Core



The same transform, three points in one design space — the lowest area-delay product comes from choosing radix, butterfly structure and FPGA resource mix together, not one at a time.

Thank you!

Feel free to reach us at surajmandal@cse.iitk.ac.in , nchaurasia25@cse.iitk.ac.in , dbroy@cse.iitk.ac.in

Backup Slides

- PQC background: LWE, a worked LWE PKE example, CRYSTALS-Kyber algorithms
- Full state-of-the-art comparison table and reference list
- KiD in detail: memory addressing, Montgomery multiplier, modular adder/subtractor, unified core, full results
- Winograd: Montgomery K-RED, radix-8 FALCON and mixed radix-16/8 Kyber butterflies, NTT/NTT⁻¹/PWM configuration
- Radix-4: detailed signal flow, chain reduction, per-configuration ATP tables, full comparison tables

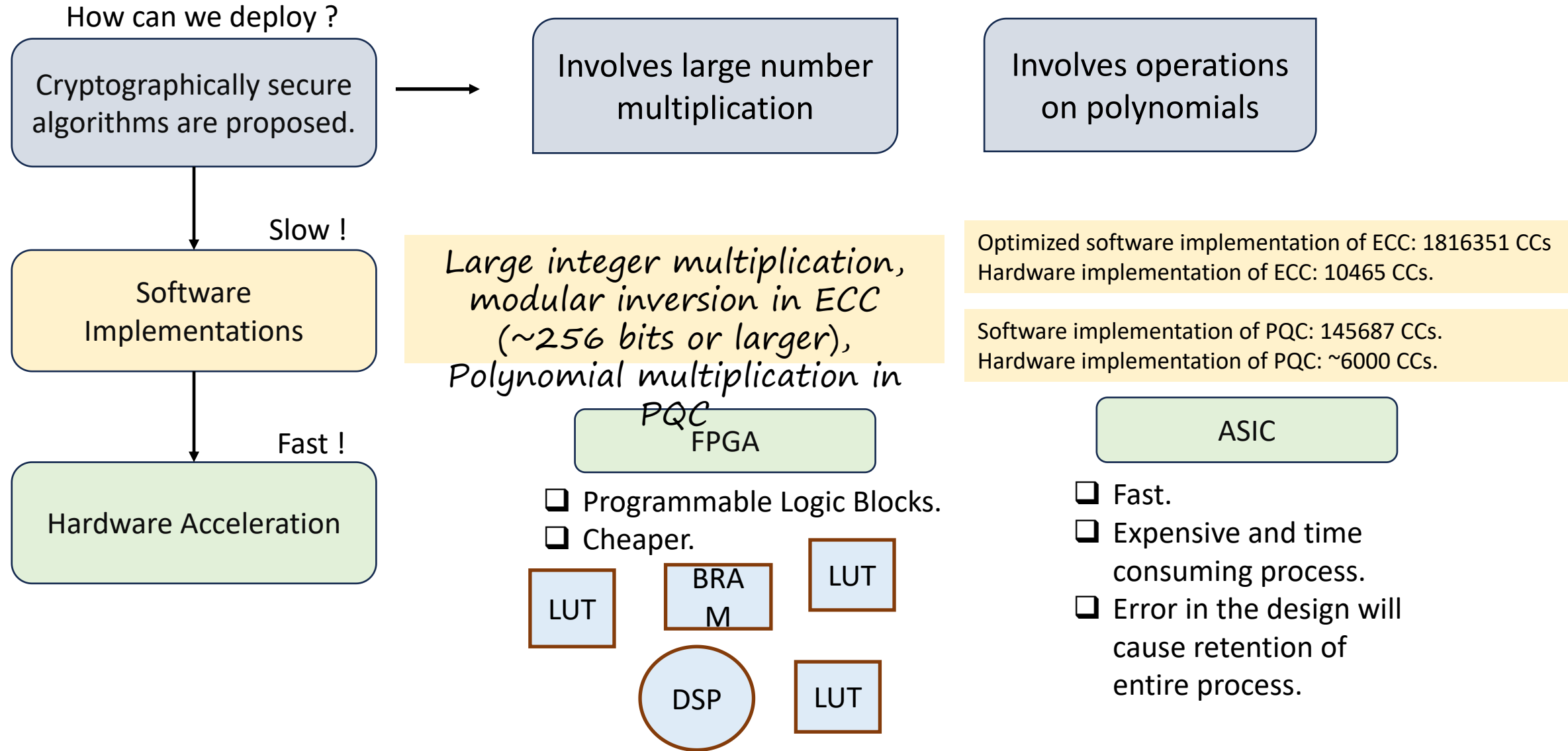
State-of-the-art Works on designing Efficient NTT Multiplication Unit

	PQC Scheme	Features in NTT Multiplication unit	BFU Unit
[1]	Dilithium	Reduces stalls between consecutive stages of NTT. Implemented 2 layers of NTT at a time.	Cooley-Tukey/Gentleman-Sande architecture.
[2]		Utilizes the DSP blocks.	Pre-computed twiddle factors.
[3]		R2DMC radix-2 Multi path delay commutator.	Optimized R2DMC based BFU unit.
[4]		Uses two dual port ROMs.	Cooley-Tukey/Gentleman-Sande based design.
[5]		Radix-4 butterfly unit.	Radix-4 Cooley-Tukey/Gentleman-Sande.
[6]	CRYSTALS-Kyber, CRYSTALS-Dilithium, FALCON, New-Hope	NTT multiplier for CRYSTALS-Kyber, CRYSTALS-Dilithium, FALCON, New-Hope	Unified NTT butterfly unit for lattice based PQC.
[7]	Kyber	Unified NTT multiplication for Saber and Dilithium.	Unified NTT butterfly unit for Saber and Dilithium.
[8]		No rearrangement between NTT and NTT ⁻¹ .	Reconfigurable BFUs to execute NTT/NTT-1/PWM.
[9]		Reduced area and memory consumption	Cooley-Tukey/Gentleman-Sande based reconfigurable architecture.
[10]		Ping-pong memory access.	Cooley-Tukey/Gentleman-Sande based reconfigurable architecture.
[11]		Pipelined NTT multiplication.	Performs NTT/NTT-1/PWM mode.
[12]		Optimized storage requirement.	Reconfigurable butterfly unit.
[13]		Mixed radix-2/4 NTT multiplication unit.	Reconfigurable mixed radix-2/4 BFU able to perform NTT/NTT ⁻¹
[14]		Mixed radix-2/4 NTT multiplication unit.	Reconfigurable mixed radix-2/4 BFU able to perform NTT/NTT-1/PWM
[15]	FALCON	Reconfigurable NTT multiplication unit for FALCON	Reconfigurable butterfly unit able to perform NTT/NTT-1/PWM 40

References

1. Beckwith, Luke, Duc Tri Nguyen, and Kris Gaj. "High-performance hardware implementation of crystals-dilithium." *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2021.
2. Land, Georg, Pascal Sasdrich, and Tim Güneysu. "A hard crystal-implementing dilithium on reconfigurable hardware." *International Conference on Smart Card Research and Advanced Applications*. Cham: Springer International Publishing, 2021.
3. Zhao, Cankun, et al. "A compact and high-performance hardware architecture for CRYSTALS-Dilithium." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022): 270-295.
4. Gupta, Naina, et al. "Lightweight hardware accelerator for post-quantum digital signature CRYSTALS-Dilithium." *IEEE Transactions on Circuits and Systems I: Regular Papers* (2023).
5. Wang, Tengfei, et al. "Efficient implementation of Dilithium signature scheme on FPGA SoC platform." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.9 (2022): 1158-1171.
6. Derya, Kemal, et al. "CoHA-NTT: A configurable hardware accelerator for NTT-based polynomial multiplication." *Microprocessors and Microsystems* 89 (2022): 104451.
7. Aikata, Aikata, et al. "A unified cryptoprocessor for lattice-based signature and key-exchange." *IEEE Transactions on Computers* 72.6 (2022): 1568-1580.
8. Xing, Yufei, and Shuguo Li. "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021): 328-356.
9. Bisheh-Niasar, Mojtaba, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography." *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2021.
10. Zhang, Cong, et al. "Towards efficient hardware implementation of NTT for kyber on FPGAs." *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021.
11. Dang, Viet Ba, Kamyar Mohajerani, and Kris Gaj. "High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber." *IEEE Transactions on Computers* 72.2 (2022): 306-320.
12. Bisheh-Niasar, Mojtaba, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. "Instruction-set accelerated implementation of CRYSTALS-Kyber." *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.11 (2021): 4648-4659.
13. Duong-Ngoc, Phap, and Hanho Lee. "Configurable mixed-radix number theoretic transform architecture for lattice-based cryptography." *IEEE Access* 10 (2022): 12732-12741.
14. Guo, Wenbo, and Shuguo Li. "Highly-efficient hardware architecture for crystals-kyber with a novel conflict-free memory access pattern." *IEEE Transactions on Circuits and Systems I: Regular Papers* (2023).

Hardware Acceleration



Introduction to Classical Cryptography

Clever Person With Powerful Computer

Alice

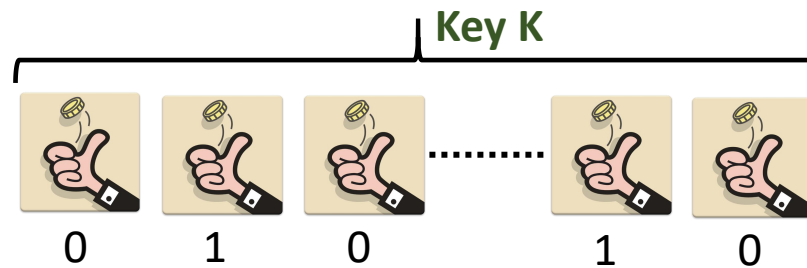


Bob



M

Public Network



Symmetric Key Cryptography

Uniformly
Random Bits

Public Key Cryptography

Public Key P_1

Private key P_2

Factorization

$$N = p \cdot q$$

p and q are large primes.

Discrete Logarithm Problem

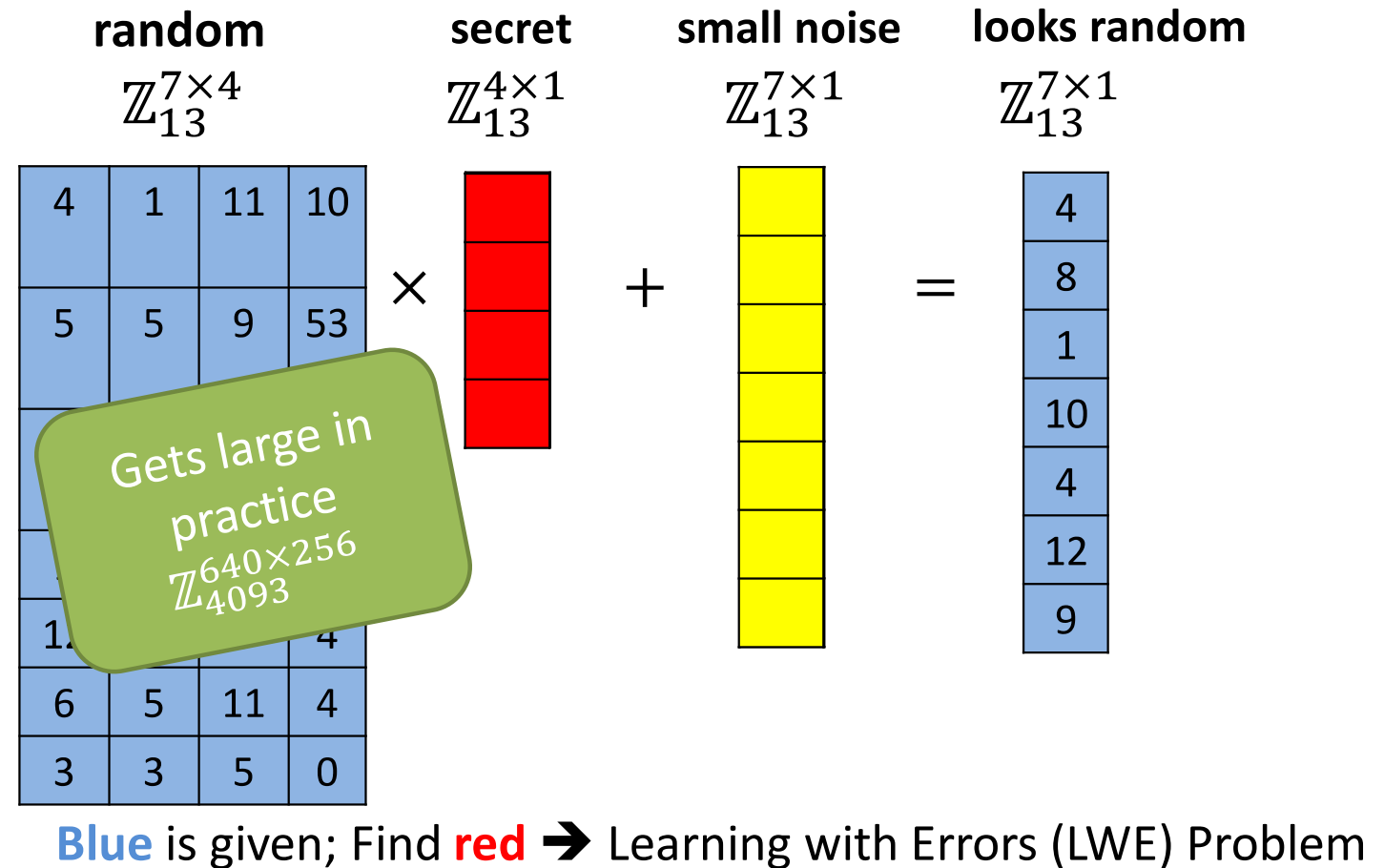
Given g^x , find x .

G is a cyclic group.

Learning With Error (LWE)

- Let $\{v_1, v_2, v_3, \dots, v_n\} \in R^m$ be a set of linearly independent vectors. The lattice $\mathbf{L}$ generated by $\{v_1, v_2, v_3, \dots, v_n\}$ is the set of integer linear combinations of $\{v_1, v_2, v_3, \dots, v_n\}$.

$$L = \{a_1 v_1 + \dots + a_n v_n \mid a_1, \dots, a_n \in \mathcal{Z}\}$$



Source: Tim Güneysu, Tutorial@CHES 2017 - Taipei

An Example of LWE PKE ($k=2$, $q=17$, and $n=4$)

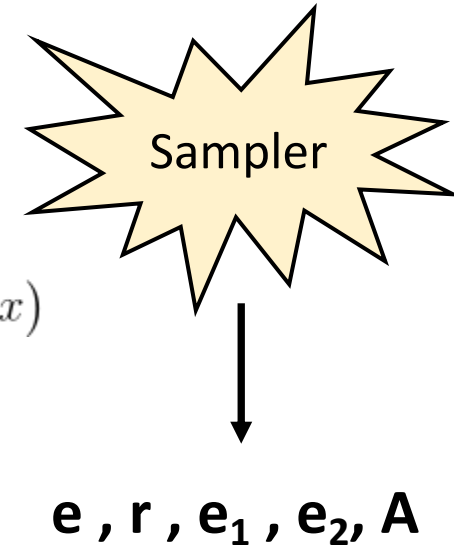
Key-Generation

$$\mathbf{t} = \mathbf{A} * \mathbf{s} + \mathbf{e}$$

Private: $\mathbf{s}$ | Public: $\mathbf{t}, \mathbf{A}$

$$\mathbf{A} = \begin{pmatrix} 6x^3 + 16x^2 + 16x + 11 & 9x^3 + 4x^2 + 6x + 3 \\ 5x^3 + 3x^2 + 10x + 1 & 6x^3 + x^2 + 9x + 15 \end{pmatrix} \quad \mathbf{s} = (-x^3 - x^2 + x, -x^3 - x) \quad \mathbf{e} = (x^2, x^2 - x)$$

$$\mathbf{t} = (16x^3 + 15x^2 + 7, 10x^3 + 12x^2 + 11x + 6)$$



Encryption

$$\mathbf{u} = \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$$

$$\mathbf{v} = \mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + \mathbf{m}$$

To encrypt a message m_b (11), we have to convert it to binary representation and then multiply with $q/2$.

$$m_b = 1x^3 + 0x^2 + 1x^1 + 1x^0 = x^3 + x + 1$$

$$\mathbf{m} = \left\lfloor \frac{q}{2} \right\rfloor \times m_b = 9x^3 + 9x + 9$$

$$\mathbf{r} = (-x^3 + x^2, x^3 + x^2 - 1)$$

$$\mathbf{e}_1 = (x^2 + x, x^2)$$

$$\mathbf{e}_2 = -x^2 - x$$

$$\begin{aligned} \mathbf{u} &= \mathbf{A}^T \mathbf{r} + \mathbf{e}_1 \\ &= (11x^3 + 11x^2 + 10x + 3, 4x^3 + 4x^2 + 13x + 11) \end{aligned}$$

$$\begin{aligned} \mathbf{v} &= \mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + \mathbf{m} \\ &= (7x^3 + 6x^2 + 8x + 15) \end{aligned}$$

Continue...

Decryption

$$\mathbf{m}_n \equiv \mathbf{v} - \mathbf{s}^T \mathbf{u}$$

$$m_n = e^T r + e_2 + m + s^T e_1 = 7x^3 + 14x^2 + 7x + 5$$

7 – Closest to 9 (q/2) or 1

14 – Closest to 17 (q) or 0

7 – Closest to 9 (q/2) or 1

5 – Closest to 9 (q/2) or 1

Decrypted Message: 1 0 1 1

CRYSTALS-Kyber

Algorithm 4 KYBER.CPAPKE.KeyGen(): key generation

Output: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$

```
1:  $d \leftarrow \mathcal{B}^{32}$ 
2:  $(\rho, \sigma) := G(d)$ 
3:  $N := 0$ 
4: for  $i$  from 0 to  $k - 1$  do
5:   for  $j$  from 0 to  $k - 1$  do
6:      $\hat{A}[i][j] := \text{Parse}(\text{XOF}(\rho, j, i))$ 
7:   end for
8: end for
9: for  $i$  from 0 to  $k - 1$  do
10:   $s[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$ 
11:   $N := N + 1$ 
12: end for
13: for  $i$  from 0 to  $k - 1$  do
14:   $e[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$ 
15:   $N := N + 1$ 
16: end for
17:  $\hat{s} := \text{NTT}(s)$ 
18:  $\hat{e} := \text{NTT}(e)$ 
19:  $\hat{t} := \hat{A} \circ \hat{s} + \hat{e}$ 
20:  $pk := (\text{Encode}_{12}(\hat{t} \bmod^+ q) \parallel \rho)$ 
21:  $sk := \text{Encode}_{12}(\hat{s} \bmod^+ q)$ 
22: return  $(pk, sk)$ 
```

Algorithm 5 KYBER.CPAPKE.Enc(pk, m, r): encryption

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$

Input: Message $m \in \mathcal{B}^{32}$

Input: Random coins $r \in \mathcal{B}^{32}$

Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

```
1:  $N := 0$ 
2:  $\hat{t} := \text{Decode}_{12}(pk)$ 
3:  $\rho := pk + 12 \cdot k \cdot n/8$ 
4: for  $i$  from 0 to  $k - 1$  do
5:   for  $j$  from 0 to  $k - 1$  do
6:      $\hat{A}^T[i][j] := \text{Parse}(\text{XOF}(\rho, i, j))$ 
7:   end for
8: end for
9: for  $i$  from 0 to  $k - 1$  do
10:   $r[i] := \text{CBD}_{\eta_1}(\text{PRF}(r, N))$ 
11:   $N := N + 1$ 
12: end for
13: for  $i$  from 0 to  $k - 1$  do
14:   $e_1[i] := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$ 
15:   $N := N + 1$ 
16: end for
17:  $e_2 := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$ 
18:  $\hat{r} := \text{NTT}(r)$ 
19:  $u := \text{NTT}^{-1}(\hat{A}^T \circ \hat{r}) + e_1$ 
20:  $v := \text{NTT}^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + \text{Decompress}_q(\text{Decode}_1(m), 1)$ 
21:  $c_1 := \text{Encode}_{d_u}(\text{Compress}_q(u, d_u))$ 
22:  $c_2 := \text{Encode}_{d_v}(\text{Compress}_q(v, d_v))$ 
23: return  $c = (c_1 \parallel c_2)$ 
```

Continue ...

Algorithm 6 KYBER.CPAPKE.Dec(sk, c): decryption

Input: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

Output: Message $m \in \mathcal{B}^{32}$

1: $\mathbf{u} := \text{Decompress}_q(\text{Decode}_{d_u}(c), d_u)$

2: $v := \text{Decompress}_q(\text{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$

3: $\hat{\mathbf{s}} := \text{Decode}_{12}(sk)$

4: $m := \text{Encode}_1(\text{Compress}_q(v - \text{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u})), 1))$

$\triangleright m := \text{Compress}_q(v - \mathbf{s}^T \mathbf{u}, 1))$

5: **return** m

$$\begin{aligned}
 v - \mathbf{s}^T \cdot \mathbf{u} &= t^T + \mathbf{e}_2 + m - \mathbf{s}^T (A^T r + \mathbf{e}_1) \\
 &= (As + \mathbf{e})^T r + \mathbf{e}_2 + m - (As)^T r + \mathbf{s}^T \cdot \mathbf{e}_1 \\
 &= \underline{(As)^T r - (As)^T r} + \underline{\mathbf{e}^T \cdot r + \mathbf{e}_2 + \mathbf{s}^T \mathbf{e}_1} + m \\
 &= m + (\text{small})
 \end{aligned}$$

	n	k	q	η_1	η_2	(d_u, d_v)	δ
Kyber512	256	2	3329	3	2	(10, 4)	2^{-139}
Kyber768	256	3	3329	2	2	(10, 4)	2^{-164}
Kyber1024	256	4	3329	2	2	(11, 5)	2^{-174}

Security Category	Attack Type
1	Key search on a block cipher like AES-128.
2	Collision search on a 256-bit hash function like SHA256.
3	Key search on a block cipher like AES-192.

Number Theoretic Transformation

$Poly_A : (A_0, A_1, A_2, \dots, A_{n-1})$

$n = 256$

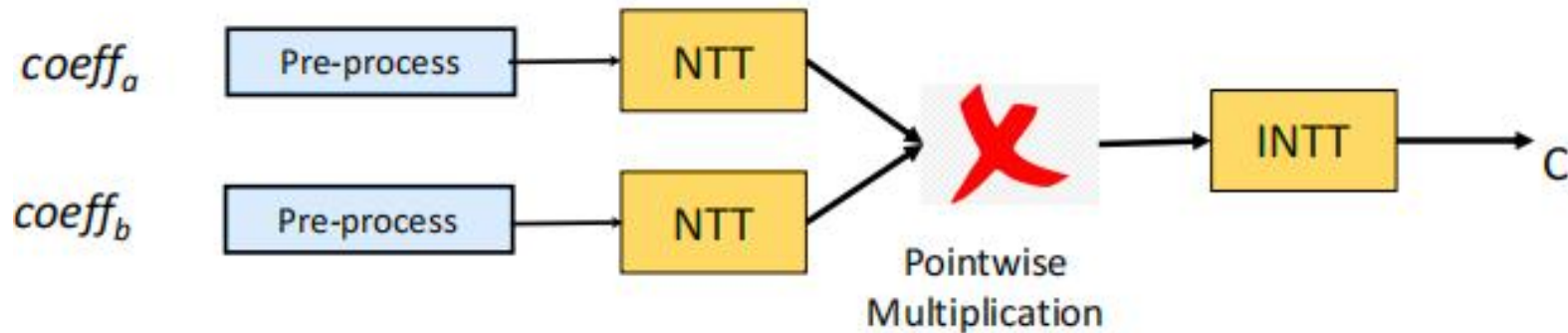
$Poly_B : (B_0, B_1, B_2, \dots, B_{n-1})$

Polynomial Multiplication Methods

Schoolbook Multiplication $O(n^2)$

Karatsuba Multiplication $O(n^{\log_2 3})$

Number Theoretic Transformation $O(n \log n)$



Convolution: The convolution product of two polynomial a,b are same as multiplying the two polynomials

Multiplication is defined as $NTT^{-1}(NTT(A).NTT(B))$

Continue...

Setup:

- Input: sequence of n positive integers
- Choose M such that $1 \leq n < M$ and every input value is in the range $[0, M)$.
- Choose an integer $k \geq 1$ and define $q = kn + 1$ such that $q \geq M$. It can be shown that we can make sure that q is prime.
- As q is prime, the multiplicative group Z_q has $q-1 = kn$ number of elements. Then this group have at least one generator g .
- Define $w = g^k \bmod q$. This w will be the primitive n^{th} root of unity

Positive Wrapped Convolution

$$C(x) = A(x) \times B(x) \bmod (x^n - 1) = \sum_{i=0}^{n-1} c_i x^i$$

$$\text{where } c_i = \left(\sum_{j=0}^i a_j \cdot b_{i-j} + \sum_{j=i+1}^n a_j \cdot b_{n+i-j} \right) \bmod q$$

Negative Wrapped Convolution with NTT

$$\tilde{A}_j = \sum_{i=0}^{n-1} \gamma^i \cdot \omega_n^{ij} \cdot A_i, A_j = \frac{1}{n} \cdot \gamma^{-j} \cdot \sum_{i=0}^{n-1} \omega_n^{-ij} \cdot \tilde{A}_j$$

Positive Wrapped Convolution with NTT

$$NTT_j^A = \sum_{i=0}^{n-1} \omega_n^{ij} \cdot A_i, NTT_j^{-1A} = \frac{1}{n} \sum_{i=0}^{n-1} \omega_n^{ij} \cdot A_i$$

Why NTT is Faster

Algorithm 1 Decimation in Time NTT Algorithm

```
1: Input:  $P_N = \{a_1, a_2, a_3, \dots, a_n\}$   $P_N \in \mathbb{Z}_q$ 
2: Output:  $\text{NTT}(P)$ 
3:  $P = \text{bitreverse}(P_N)$ 
4: for  $m \leftarrow 2$  to  $n$  do
5:    $\omega = 1, \omega_m = \omega_n^{n/m}$ 
6:   for  $j \leftarrow 0$  to  $\frac{m}{2}$  do
7:     for  $k \leftarrow 0$  to  $n$  do
8:        $u_1 = P[k+j], t_1 = P[k+j+\frac{m}{2}] \cdot \omega$ 
9:        $P[k+j] = (u_1 + t_1) \bmod q$ 
10:       $P[k+j+\frac{m}{2}] = (u_1 - t_1) \bmod q$ 
11:       $k = k + m$ 
12:   end for
13:    $\omega = (\omega_m \cdot \omega) \bmod q$ 
14:    $j = j + 1$ 
15: end for
16:  $m = 2 * m$ 
17: end for
```

- Both Decimation in Time and Decimation of Frequency algorithm has a complexity of $O(n \log n)$
- They apply divide and conquer approach to achieve this acceleration

- The pointwise multiplication has a complexity of $O(n)$ only
- The multiplication and addition operations involved in these two algorithms are done in the field $\mathbb{Z}_q$

Algorithm 2 Decimation in Frequency NTT^{-1} Algorithm

```
1: Input:  $\hat{P} = \{b_1, b_2, b_3, \dots, b_n\}$   $P \in \mathbb{Z}_q$ 
2: Output:  $\text{NTT}^{-1}(\hat{P})$ 
3: for  $m \leftarrow n$  to  $1$  do
4:    $j_1 = 0, h = \frac{m}{2}$ 
5:   for  $i \leftarrow 0$  to  $h$  do
6:      $j_2 = j_1 + t - 1$ 
7:      $\text{index} = h + i$ 
8:      $\text{Omega} = (\omega^{-1})^{(\text{bitreverse}(\text{index}))}$ 
9:     for  $j \leftarrow j_1$  to  $j_2$  do
10:       $u_1 = \hat{P}[j], t_1 = \hat{P}[j+t] \cdot \omega$ 
11:       $\hat{P}[j] = (u_1 + t_1) \bmod q$ 
12:       $\hat{P}[j+t] = ((u_1 - t_1) \cdot \text{Omega}) \bmod q$ 
13:       $j = j + 1$ 
14:   end for
15:    $j_1 = j_1 + 2 * t$ 
16:    $i = i + 1$ 
17: end for
18:  $m = \frac{m}{2}$ 
19: end for
20: for  $i \leftarrow 0$  to  $n$  do
21:    $\hat{P}[i] = \hat{P}[i] \cdot \frac{1}{n}$ 
22:    $i = i + 1$ 
23: end for
```

Modular Multipliers

$a=2238, b=1276, (a \cdot b) \% 3329 = 2735$

Montgomery

Algorithm	Mont_Reduc(R,P,T): Montgomery Reduction
	Input: R, P, T . P is the prime, $R = 2^k > P$, $0 \leq T < PR$, $\gcd(P, R) = 1$
	Output: $TR^{-1} \bmod P$
1	Compute P' such that $RR^{-1} - PP' = 1$ (can be computed using extended Euclidean algorithm);
2	$m = T \times P' \bmod R$;
3	$t = (T + mP)/R$;
4	if $t \geq P$ then
5	$t = t - P$;
6	end
7	return t ;

Barrett

Algorithm	Barrett modular multiplication.
	Input: A modulus $M \in \mathbb{N}_{\geq 2}$ of length $k = \lfloor \log_b M \rfloor + 1$ in base $b \in \mathbb{N}_{\geq 2}$, the integer reciprocal $\mu = \lfloor b^{2k}/M \rfloor$ of M , and integers $x, y \in \mathbb{Z}_M$.
	Output: $(x \cdot y) \bmod M$.
1:	$z \leftarrow x \cdot y$
2:	$\tilde{q} \leftarrow \lfloor \lfloor z/b^{k-1} \rfloor \mu / b^{k+1} \rfloor$
3:	$r \leftarrow z - \tilde{q} \cdot M$
4:	while $r \geq M$ do
5:	$r \leftarrow r - M$ ▷ Extra reduction
6:	end while
7:	return r

K-Red

Algorithm	K-RED Algorithm
1:	Input: $C = a \times b$
2:	Output: $K \times C \bmod q$
3:	$d_0 = C[m : 0], d_1 = C \gg m$
4:	return $(K \times d_0 - d_1)$

$K = 2^8 \cdot 13 + 1$

$K\text{-Red}(a, b) = 2265,$
where $K=13$

$\text{Montgomery}(a, b) = 475$

Where $R = 2^{12}$, and $R^{-1} = 2704$

$(2704 \cdot 2735) \% 3329 = 475$

$\text{Barrett}(a, b) = 2735$

Where $k=12$ and $\mu=5039$

$(13 \cdot 2735) \% 3329 = 2265$

KiD: What Problem Are We Solving?

The setting

A complete post-quantum public-key stack needs both a KEM and a signature scheme, so a device deploying NIST PQC will run ML-KEM and ML-DSA side by side. Both are defined over $\mathbb{Z}_q[x]/(X^n+1)$ with $n = 256$, and in both the polynomial multiplication unit dominates the area and the runtime. Instantiating a separate NTT core for each scheme pays for the same silicon twice.

The observation that makes sharing possible

ML-KEM coefficient: 12 bits ($q = 3329$)

ML-DSA coefficient: 23 bits ($q = 8380417$)

Two Kyber coefficients occupy almost exactly the space of one Dilithium coefficient — so one 24-bit datapath is either two Kyber lanes or one Dilithium lane.

The goal

- One NTT multiplication unit that performs NTT, NTT^{-1} and PWM for both ML-KEM and ML-DSA.
- Reconfigurable at run time, with no duplicated arithmetic and no idle logic in either mode.
- Fully pipelined — no stalls between NTT stages.
- Scalable: the same core instantiated with 2, 4 or 8 Kyber butterflies.
- Optimised for area-delay product, not for peak frequency alone.

Why radix-2 for a unified core

ML-DSA has a 512th root of unity and supports a complete NTT, so radix-4 would suit it well. ML-KEM does not: only a 256th root exists, the NTT is incomplete, and the odd and even coefficients are transformed independently. Radix-2 is therefore the largest structure both schemes share — the common denominator on which a genuinely unified datapath can be built. Higher radix is revisited later in this talk, per scheme rather than shared.

KiD: Four Design Principles

Everything in the architecture follows from one rule: share a resource only where sharing costs less than duplicating it.

01

Share at the bit level, not with multiplexers

One 24-bit adder/subtractor serves as two Kyber lanes or one Dilithium lane. A control bit inserted at position 12 decides whether the carry propagates across the boundary or is stopped there.

Keeps the FPGA fast-carry chain intact — earlier unified work used multiplexers and lost it.

02

Size the multiplier once, use it twice

Two DSP blocks configured as 23×12 give either two independent Kyber coefficient multiplications or, with the partial products summed, one Dilithium multiplication.

The same DSPs are busy in both modes; nothing sits idle.

03

Reduce using the structure of the prime

Montgomery reduction specialised to $q = 2^{11} + 2^{10} + 2^8 + 1$ for Kyber and $q = 2^{23} - 2^{13} + 1$ for Dilithium, so the constant multiplications become shifts and adds.

71 LUTs + 1 DSP per Kyber multiplication; cheaper than the Solinas reduction used in prior unified work.

04

Make the memory map, not the control unit, do the work

Two simple dual-port RAMs with a ping-pong read/write address sequence, generated offline and hard-coded in a ROM rather than computed by logic.

Conflict-free and stall-free as long as pipeline depth $\leq$ (coefficient memory depth)/2.

What this buys

A single core covering both standardised lattice schemes at 1384 LUT, 2 DSP and 4.5 BRAM running at 387 MHz — a better area-delay product than the existing unified design, and better than the two standalone cores it replaces.

Unified NTT Multiplication Unit for CRYSTALS-Kyber and CRYSTALS-Dilithium (VLSI-Design 2024)

Suraj Mandal, Debapriya Basu Roy

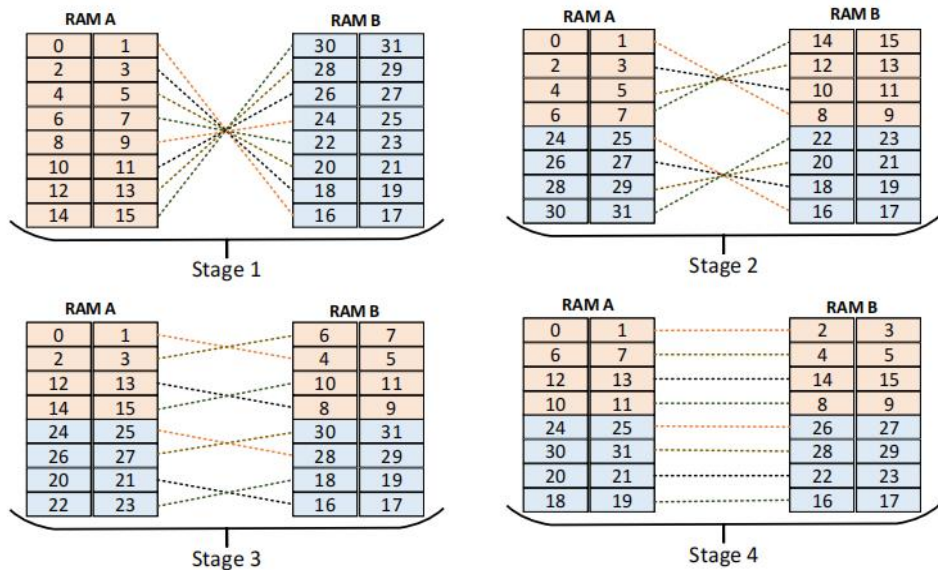
Department of Computer Science & Engineering

Indian Institute of Technology Kanpur



Memory Addressing Scheme

Pipelined Memory Access without Stalling



Algorithm 1 Pipelined Memory Addressing for NTT/NTT⁻¹

```

1: procedure ADDRESSING( ch, d)
2:   if ch==1 then
3:     start = 1, end =  $\log_2 d$ 
4:   else
5:     start =  $\log_2 d$ , end = 1
6:   end if
7:   for i=start to end do
8:      $m = 2^i$ ,  $p_2 = \frac{m}{2}$ 
9:     for j=0 to N by  $p_2$  do
10:       $k_1 = j$ ,  $k_2 = p_2 - 1 + j$ 
11:      for k=0 to  $p_2$  do
12:        if k is even then
13:          addressA =  $k_1$ , addressB =  $k_2$ 
14:        else if k is odd then
15:          addressA =  $k_2$ , addressB =  $k_1$ 
16:           $k_1 = k_1 + 1$ ,  $k_2 = k_2 - 1$ 
17:        end if
18:      end for
19:    end for
20:  end for
21:  return (addressA, addressB)
22: end procedure

```

To achieve a fully pipelined Radix-2 NTT/INTT architecture, the memory mapping scheme must execute $\log_2 n$ stages without stalling.



$$\text{Pipeline Depth} \leq (\text{Depth of Coefficient Memory}) / 2$$

Design of Efficient Montgomery Multiplier

Algorithm 1. Montgomery Reduction

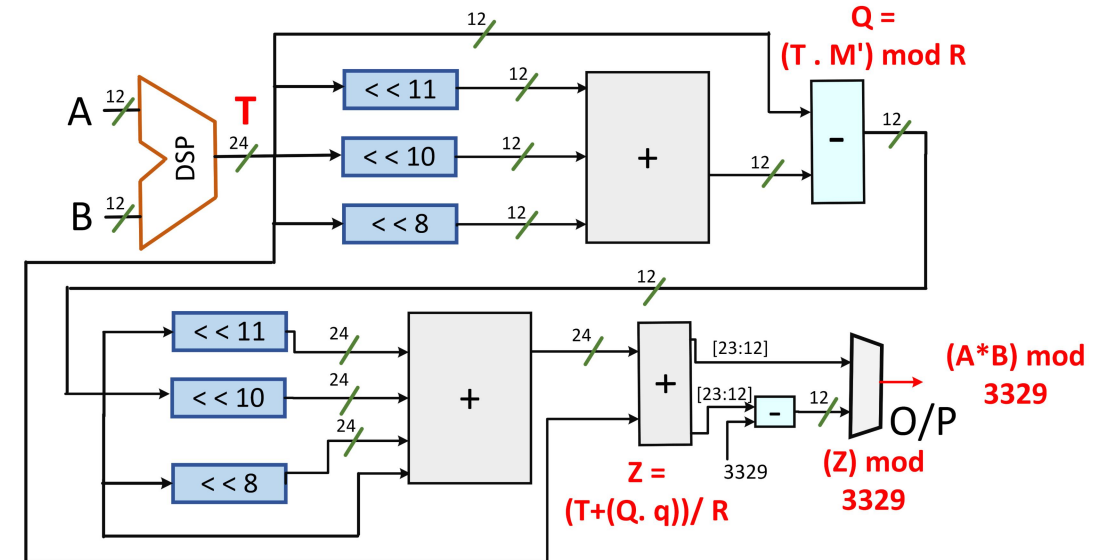
Require: An m -bit modulus M , Montgomery radix $R = 2^m$, two m -bit operands A and B , and m -bit pre-computed constant $M' = -M^{-1} \bmod R$

Ensure: Montgomery product ($Z = (A \cdot B) \cdot R^{-1} \bmod M$)

- 1: $T \leftarrow A \cdot B$
 - 2: $Q \leftarrow T \cdot M' \bmod R$
 - 3: $Z \leftarrow (T + Q \cdot M) / R$
 - 4: **if** $Z \geq M$ **then** $Z \leftarrow Z - M$ **end if**
 - 5: **return** Z
-

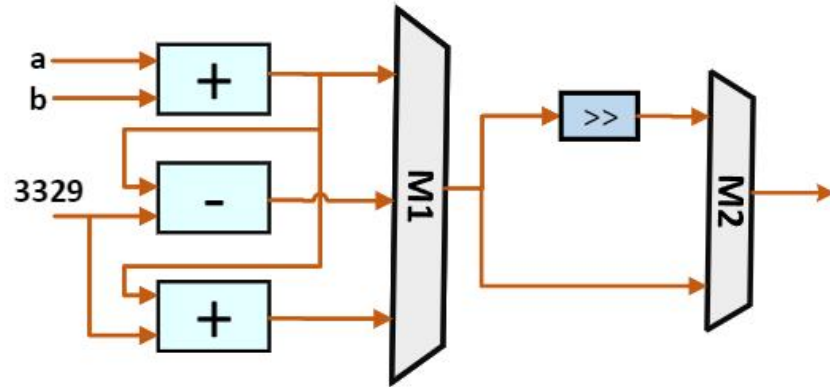
- ✓ Prime structure $M = 2^{11} + 2^{10} + 2^8 + 1$
- ✓ Structure of $M' = 2^{11} + 2^{10} + 2^8 - 1$
- ✓ Omits the use of DSP blocks for multiplications.

Montgomery Multiplier with one DSP

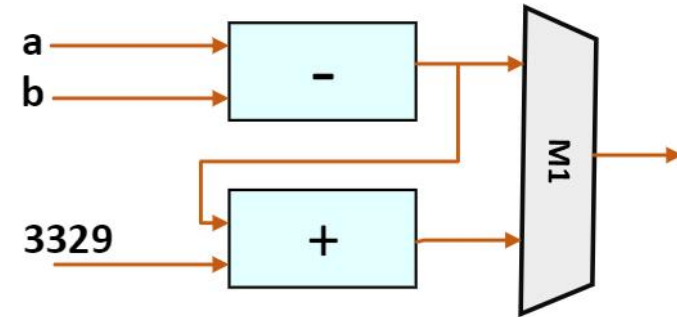


Modular Adder/Subtractor

Modular Adder

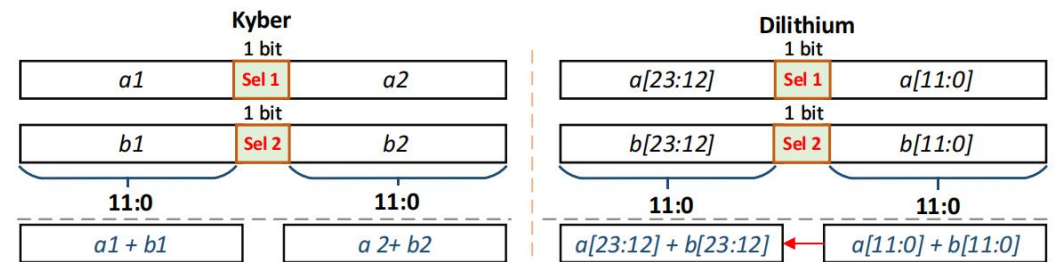


Modular Subtractor



- ✓ $M1$ and $M2$ in modular adder is configured to perform $2^{-1} \cdot (a+b)$ during NTT^{-1} .
- ✓ In modular subtractor, multiplication with 2^{-1} has been precomputed with twiddle factors.

Shared Adder for Kyber and Dilithium



Proposed Unified NTT Multiplication Unit

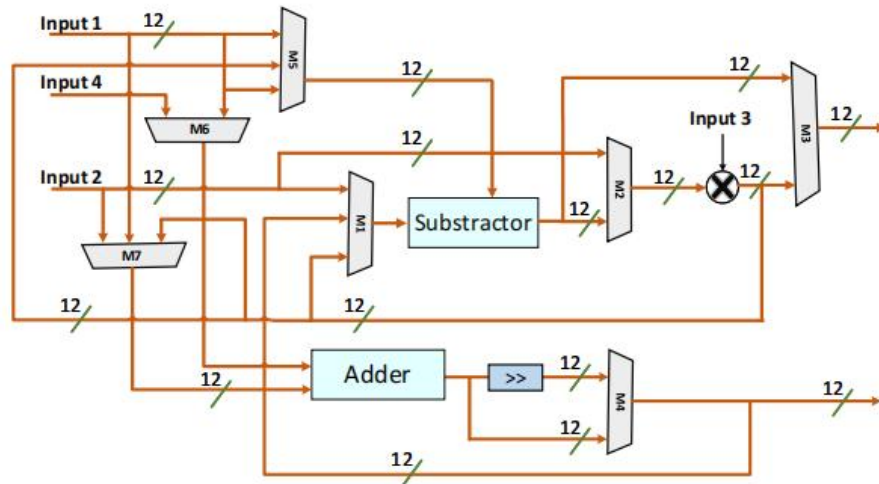
Our Proposal

Design 1: The first design is based on 2 radix-2 BFUs for Kyber that can also be used as 1 radix-2 BPU of Dilithium.

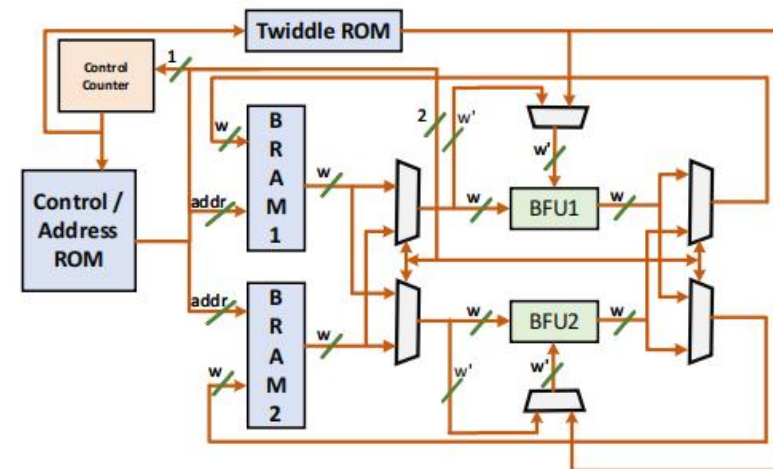
Design 2: The second design is based on 4 radix-2 BFUs of Kyber that can be used as 2 radix-2 BFUs of Dilithium.

Design 3: Our third design is based on 8 radix-2 BFUs of Kyber that can be used as 4 radix-2 BFUs of Dilithium.

BFU Unit



NTT Architecture



Experimental Results

Results for Standalone Implementation

NTT Type		Board	LUT	FFs	BRAM	DSP	Latency			Freq. (Mhz)	ADP-LUT			ADP-DSP	ADP-BRAM
							NTT	INTT	PWM		NTT	INTT	PWM		
Kyber	[3]	Artix-7	801	717	2	4	324	324	-	222	1169	1169	-	6	3
	[17]		360	145	2	3	940	1203	1289	115	2943	3766	4035	25	16
	[17]		737	290	4	6	474	602	1289	115	3038	3858	8261	25	16
	[2]		1579	1058	3	2	448	448	256	161	4394	4394	2511	6	8
	[5]		880	999	1.5	2	448	448	256	222	1776	1776	1015	4	3
	[4]		609	640	4	2	490	490	-	256	1166	1166	-	8	15
	TW		799	916	2	2	448	448	256	310	1155	1155	660	3	3
	TW	ZCU+ 102	698	865			448	448	256	500	625	625	357	2	2
Dilithium	[8]	Artix-7	9018	6292	2	16	256	256	-	250	9234	9234	-	16	2
	[10]	Zynq 7000	2386	932	2	8	256	256	64	217	2815	2815	704	9	2
	[9]	Artix-7	524	759	1	17	533	536	-	311	898	903	-	29	2
	[11]		2759	2037	7	4	512*	512*	128*	163	8666	8666	2167	13	22
	[11]	ZCU+	2759	2037	7	4	512*	512*	128*	391	3613	3613	903	5	9
	TW	Zynq 7000	698	771	2.5	2	1024	1024	256	279	2562	2562	640	7	9
		Artix-7	690	771			1024	1024	256	273	2588	2588	647	8	9
		ZCU+ 102	724	769			1024	1024	256	413	1795	1795	449	5	6

Results for Unified Implementation

NTT Type		Board	LUT	FFs	BRAM	DSP	K/D	Latency			Freq. (Mhz)	ADP-LUT			ADP-DSP	ADP-BRAM
								NTT	INTT	PWM		NTT	INTT	PWM		
Unified	[1]	ZCU+ 102	3487	1918	3*	4	K	224	224	128	270	2893	2893	1653	3	2
							D	512	512	128	270	6612	6612	1653	8	6
	TW		1384	1220	4.5	2	K	448	448	256	387	1602	1602	916	2	5
							D	1024	1024	256	387	3662	3662	916	5	12
			2893	2356	4.5	4	K	224	224	128	342	1895	1895	1083	3	3
							D	512	512	128	342	4331	4331	1083	6	7
			5909	3376	5.5	8	K	112	112	64	294	2251	2251	1286	3	2
							D	256	256	64	294	5145	5145	1286	7	5
	Artix-7	1315	1280	4.5	2	K	448	448	256	263	2240	2240	1280	3	8	
						D	1024	1024	256	263	5120	5120	1280	8	18	
		3105	2389	4.5	4	K	224	224	128	200	3478	3478	1987	4	5	
						D	512	512	128	200	7949	7949	1987	10	12	
		6201	3562	5.5	8	K	112	112	64	165	4209	4209	2405	5	4	
						D	256	256	64	165	9621	9621	2405	12	9	

Montgomery K-RED Algorithm for FALCON and Kyber

Algorithm 8 Modified K-RED Algorithm for FALCON

```
1: Input:  $C = a[13 : 0] \times b[15 : 0]$ 
2: Output:  $K'_{FALCON} \times C \bmod q$ , where  $K'_{FALCON} = 512^{-1}$ 
3:  $c_8 = C \ll 3$ 
4:  $cl_1 = c_8[13 : 0]$ ,  $ch_1 = c_8 \gg 14$ ,  $ch_{11} = (ch_1 \ll 12) - ch_1$ 
5:  $cl_2 = ch_{11}[13 : 0]$ ,  $ch_2 = ch_{11} \gg 14$ ,  $ch_{21} = (ch_2 \ll 12) - ch_2$ 
6:  $cl_3 = ch_{21}[13 : 0]$ ,  $ch_3 = ch_{21} \gg 14$ ,  $ch_{31} = (ch_3 \ll 12) - ch_3$ 
7:  $cl_4 = ch_{31}[13 : 0]$ ,  $ch_4 = ch_{31} \gg 14$ ,  $ch_{41} = (ch_4 \ll 12) - ch_4$ 
8:  $cl_5 = ch_{41}[13 : 0]$ ,  $ch_5 = ch_{41} \gg 14$ ,  $ch_{51} = (ch_5 \ll 12) - ch_5$ 
9:  $c_1 = ch_{51} + c_1 + cl_2 + cl_3 + cl_4 + cl_5$ 
10: return K-RED( $c_1$ )
```

Algorithm 9 K-RED Montgomery Modular Multiplication for FALCON

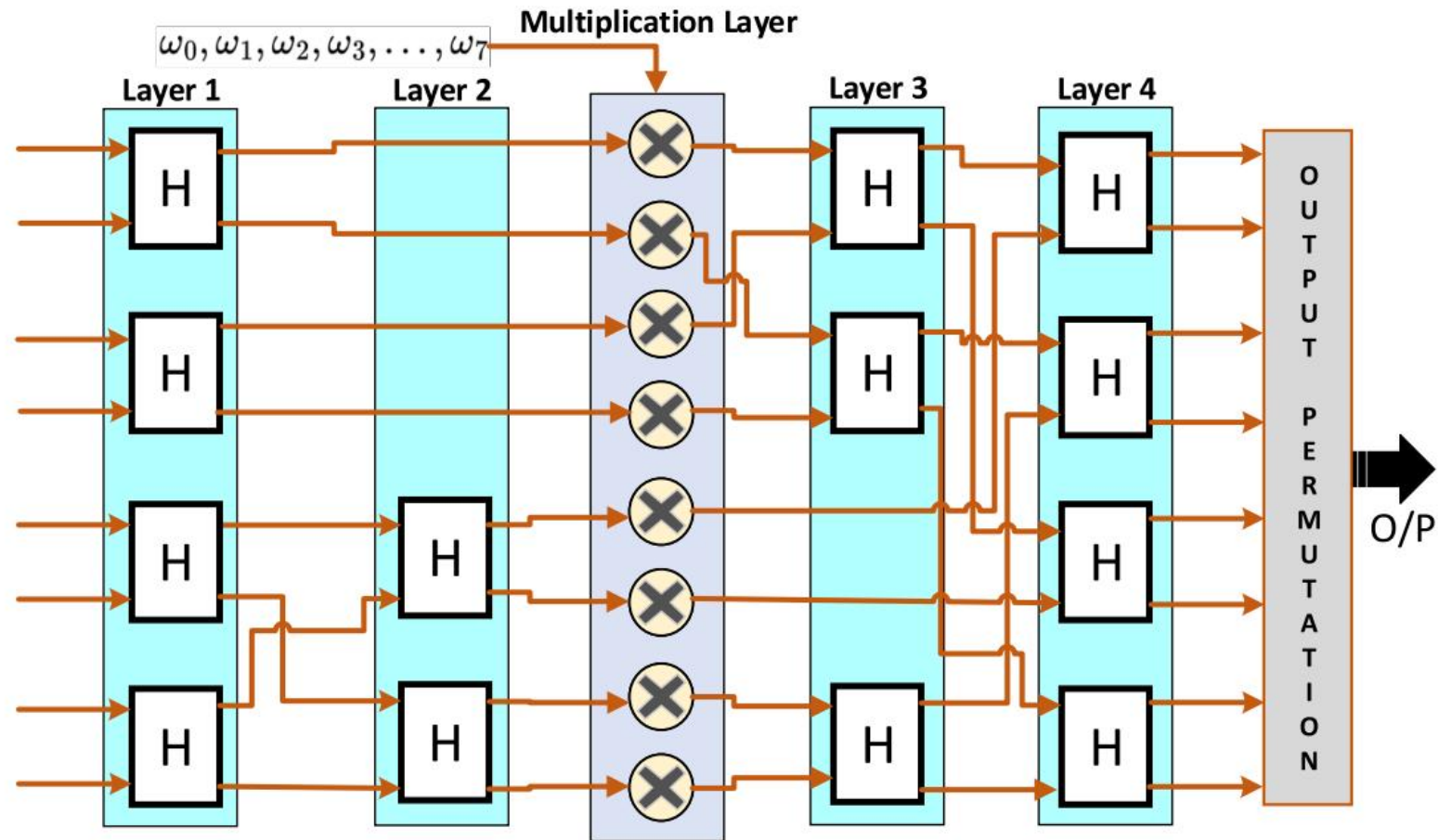
```
1: Input:  $t = a[13 : 0] \times b[15 : 0]$ , where  $a, b \notin \text{Montgomery Domain}$ 
2: Output:  $t.R^{-1} \bmod q$ 
3:  $t_d = (t \ll 14) - (t \ll 12) - t$ ,  $m = t_d[17 : 0]$ 
4:  $mult_n = (m \ll 14) - (m \ll 12) + m$ 
5:  $res_1 = (t + mult_n) \gg 18$ 
6: if  $res_1 > \text{prime}$  then
7:    $result = res_1 - \text{prime}$ 
8: end if
9: return  $result$ 
```

- For both FALCON and Kyber, modified K-RED did not perform well due to the increased operand width.
- We proposed a completely new multiplication algorithm combining Montgomery multiplication with K-RED method

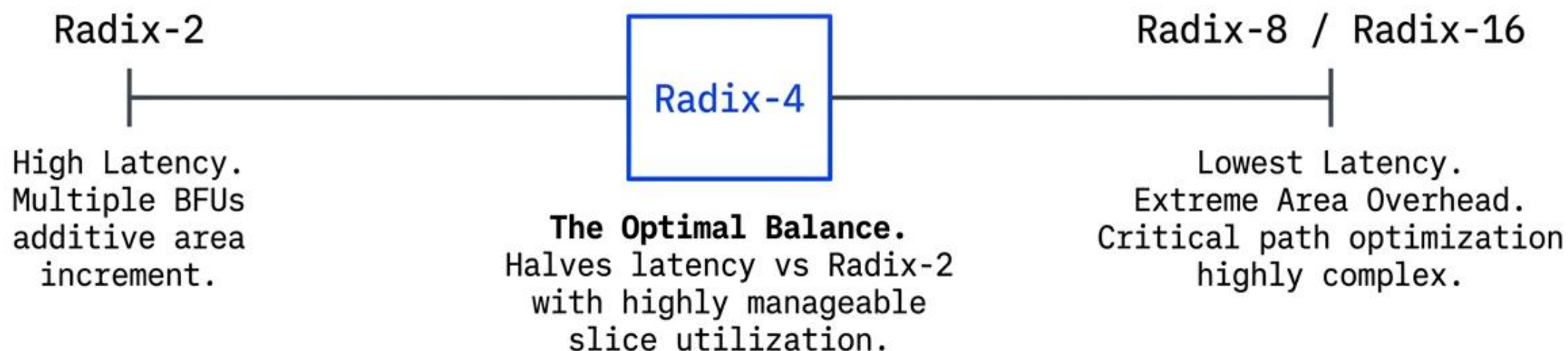
Reduction Method	DSP	LUT	FF	Frequency (Mhz)
Montgomery	1	96	42	294
K-RED	1	226	160	173

Result: Low Area Modular Multiplier with low critical path compared to traditional coefficient multiplier

Winograd Radix-8 BFU used for FALCON (degree 512)



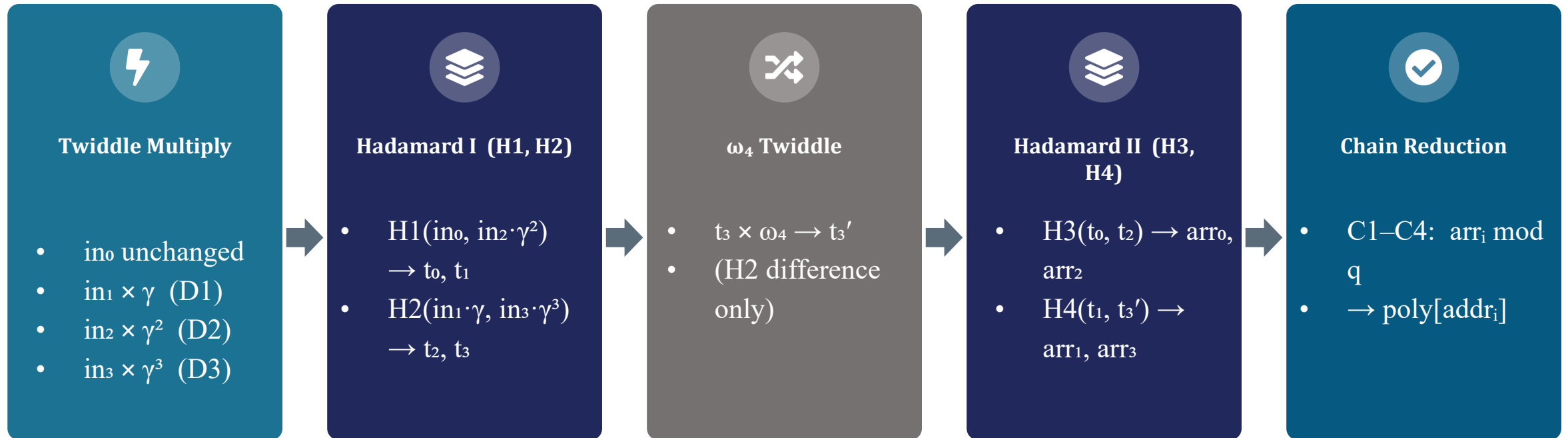
The Radix Selection: Balancing Latency and Area Overhead



Algorithm Distinction:

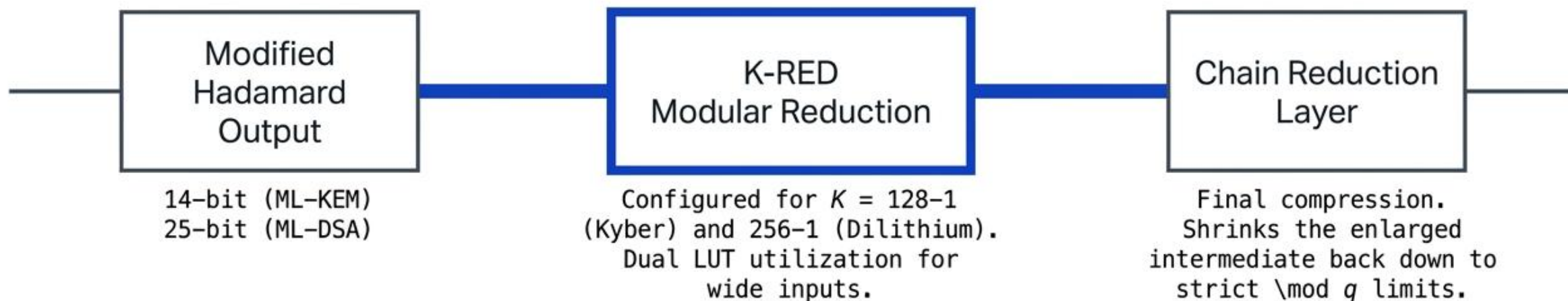
ML-DSA uses standard NTT ($n=256$).
ML-KEM uses incomplete NTT ($n=128$)
alongside a radix-2 final layer due to
the absence of a 512^{th} root of unity.

Our Proposed Radix-4 BFU — Detailed Signal Flow

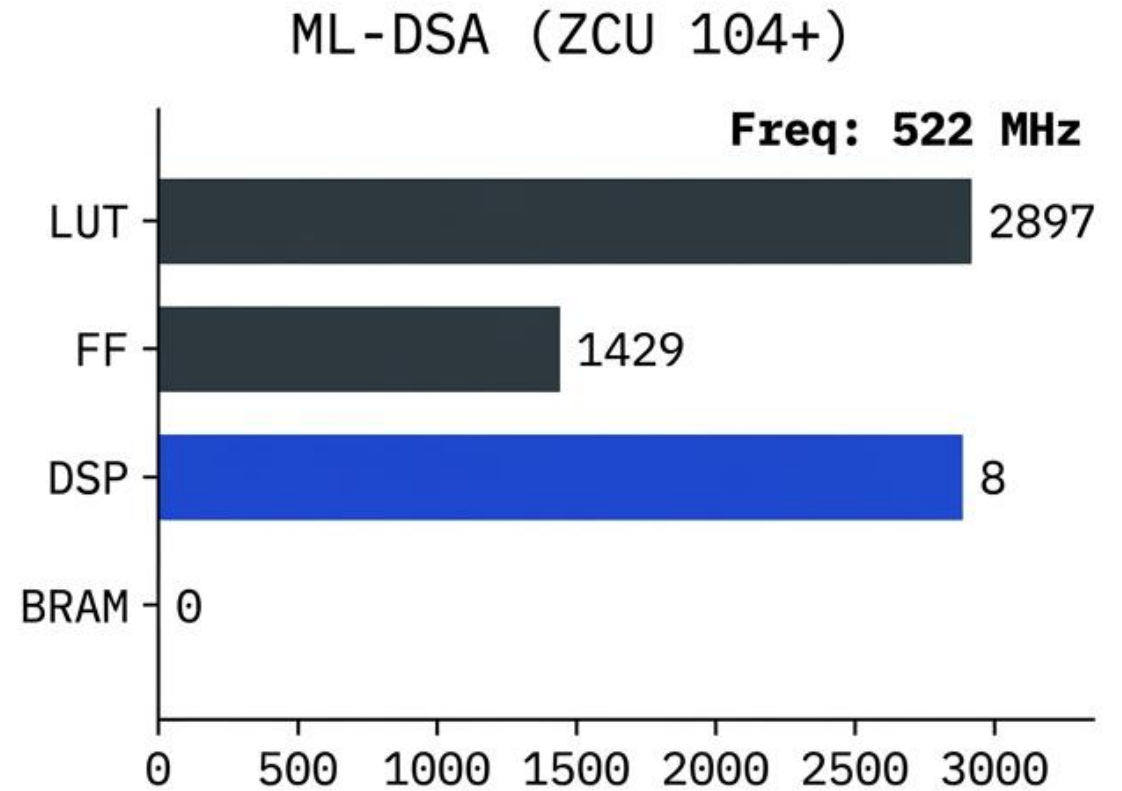
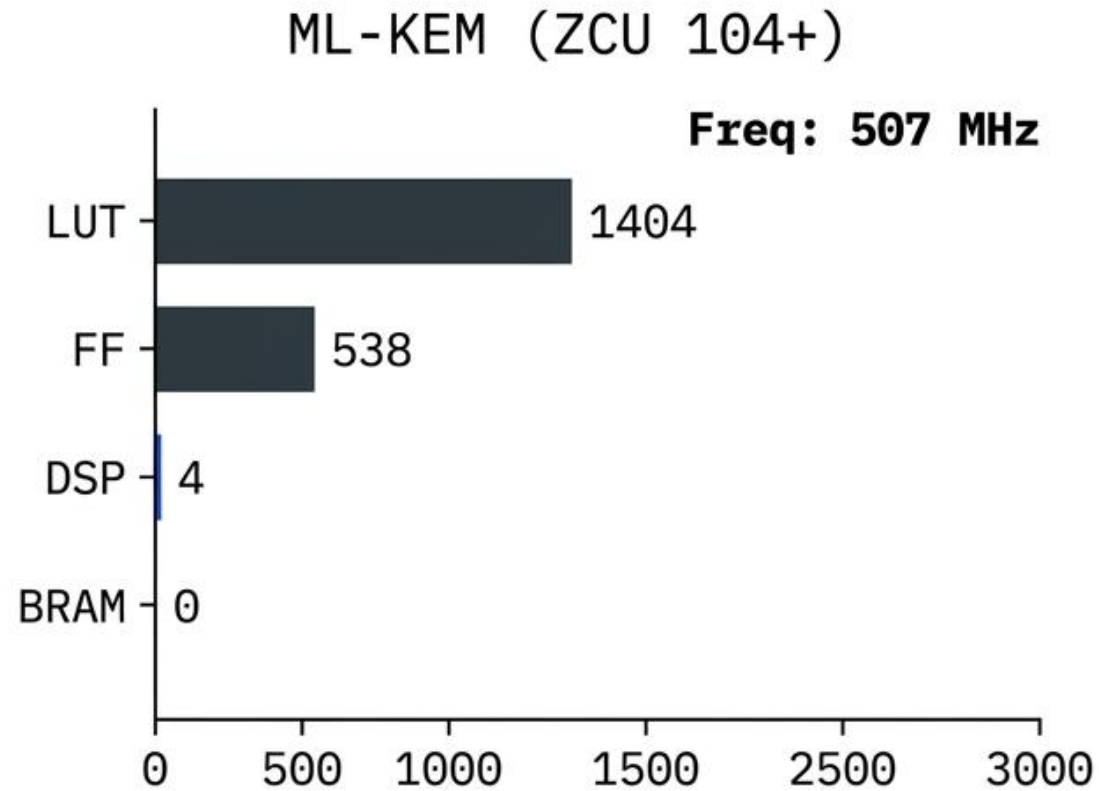


- Mode-select: 19 multiplexers (M1–M19) reroute this same datapath for NTT^{-1} (twiddle multiply moves after Hadamard II) and PWM, and for ML-KEM's radix-2 final layer.
- Used K-RED modular reduction: Configured for $K = 128^{-1}$ (Kyber) and 256^{-1} (Dilithium).

Resolving Bit-Width Expansion: K-RED and Chain Reduction



Resource Profiling: BRAM-Free Implementation



Finding Out the Optimal Solution for Achieving lowest ATP :

Config.	Platform	LUT	FF	BRAM	DSP	Latency	Freq (MHz)	ATP
						NTT/PWM/INTT		
C_1	ZCU 104+	1404	538	0	4	256/128/256	507	1516
C_2		1265	588	2	4		358	3192
C_3		1713	1006	0	0		400	1096
C_1	Artix-7	1416	680	0	4		256	3016
C_2		1376	679	2	4		227	5160
C_3		1924	886	0	0		205	2402

Our Proposed Radix-4 NTT Multiplication Units for Kyber in Different Configurations. C1 : Uses 4 DSP with no BRAMs. C2 : Uses 4DSPs with 2 BRAMs, C3 : Uses no DSPs and no BRAMs

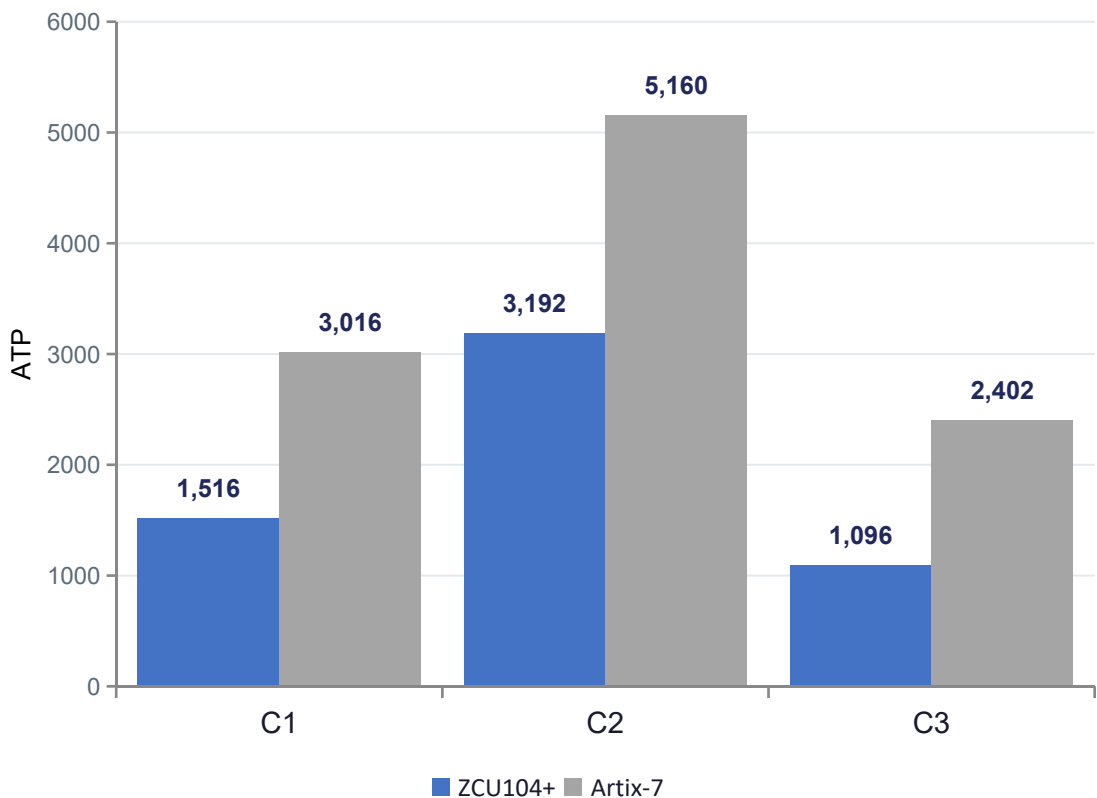
Config.	Platform	LUT	FF	BRAM	DSP	Latency	Freq (MHz)	ATP
						NTT/PWM/INTT		
C_1	ZCU 104+	2897	1429	0	8	256/64/256	522	2990
C_2		2598	1329	4	8		527	4370
C_3		3426	1719	0	4		512	2513
C_4		3149	1769	4	4		500	4069
C_1	Artix-7	3014	1423	0	8		273	5827
C_2		2760	1423	4	8		277	8465
C_3		3403	1710	0	4		256	5003
C_4		3264	2024	4	4		238	8674

Our Proposed Radix-4 NTT Multiplication Units for Dilithium in Different Configurations. C1 : Uses 8 DSP with no BRAMs. C2 : Uses 8DSPs with 4 BRAMs, C3 : Uses 4 DSPs with 0 BRAMs, C4 : Uses 4 DSPswith 4 BRAMs.

Finding the Optimal Configuration for Lowest ATP

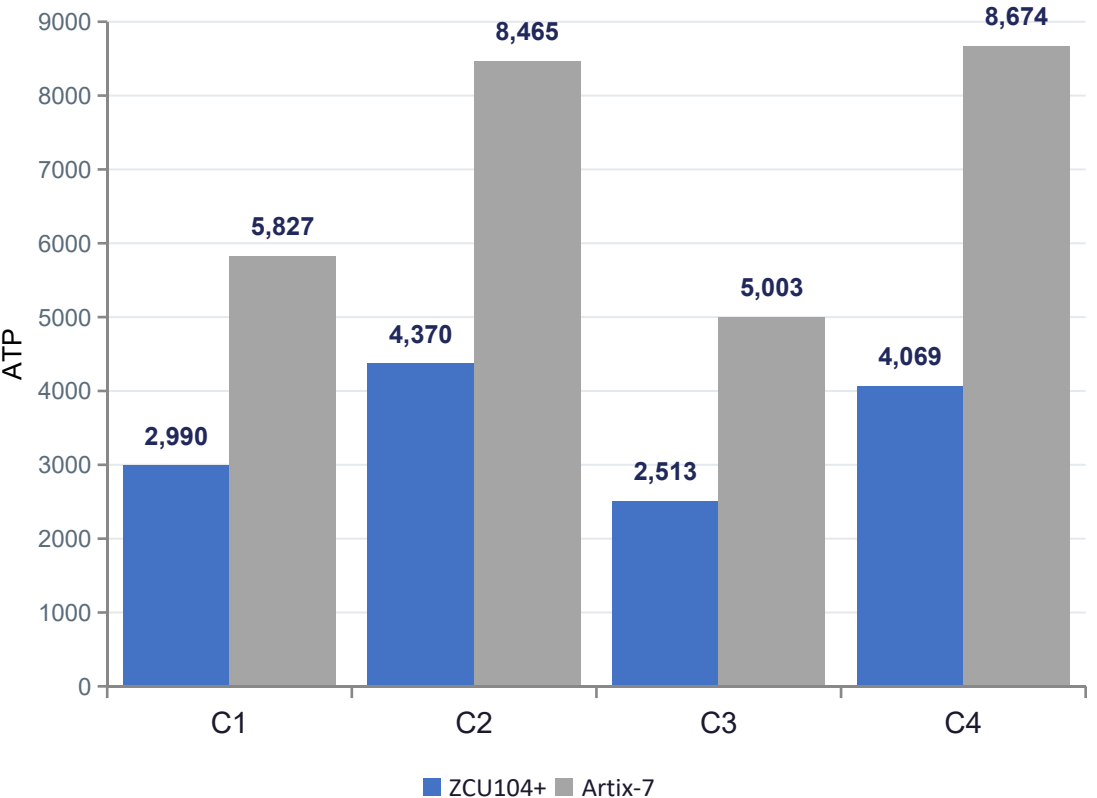
ATP across all explored configurations (C1–C4) on both target platforms — lower is better

ML-KEM (Kyber)



Lowest: C3 on ZCU104+ — ATP 1096 (0 DSP, 0 BRAM)

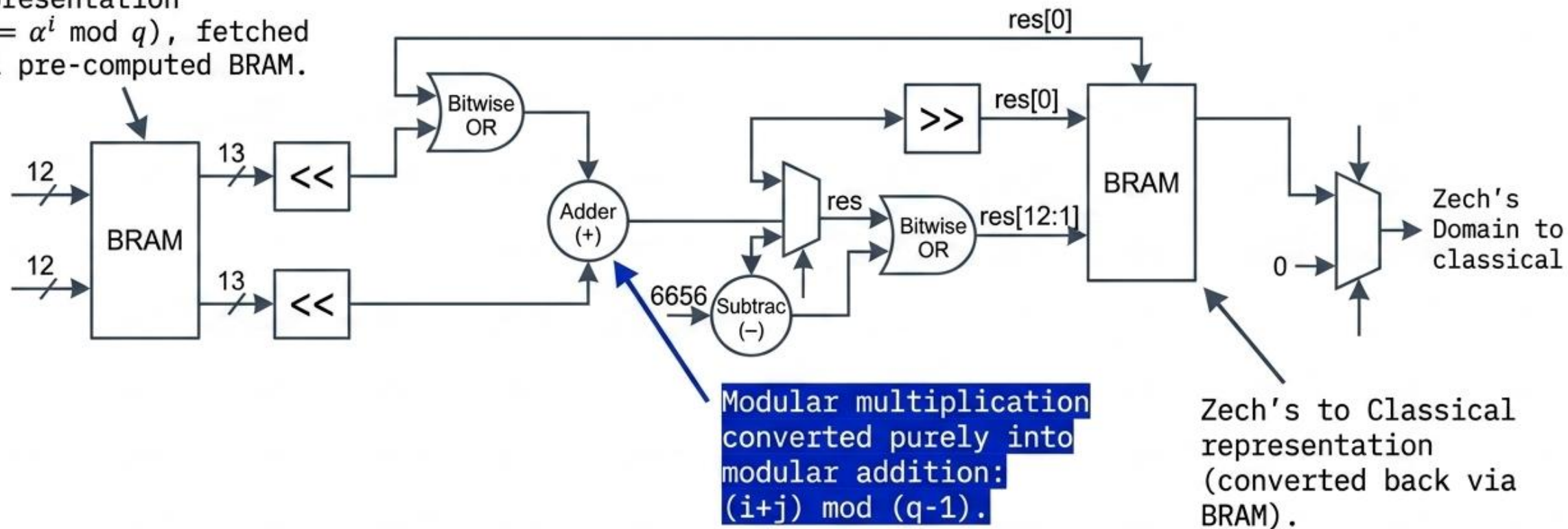
ML-DSA (Dilithium)



Lowest: C3 on ZCU104+ — ATP 2513 (4 DSP, 0 BRAM)

Architecture III: Multiplier-Less Zech's Logarithm (ML-KEM)

Classical to Zech's representation
($x = \alpha^i \bmod q$), fetched
via pre-computed BRAM.



Coefficient width limitation: Viable for ML-KEM (12-bit) via BRAM mapping, but infeasible for ML-DSA (23-bit) due to exponential memory mapping scale.

Comparison Result for ML-KEM

Work	Radix	Board	LUT	FF	DSP	BRAM	Freq. (Mhz)	Latency (NTT)	ATP
[31]	Radix-2	Artix-7	1579	1058	2	3	161	448	13298
[25]			801	717	4	2	222	324	5839
[9]			609	640	3	4	256	490	9588
[26]			880	999	3	1.5	229	448	6417
[3]			799	916	2	2	310	448	4623
[27]			737	290	6	4	115	474	26119
[4]			4834	4683	0	1	250	277	6242
[5]			2543	792	4	9	232	182	8898
[6]			1154	1031	2	0	300	456	2970
[7]			690	419	1	2.5	190	904	14702
			1782	1027	4	9	182	232	13489
			6393	3604	16	33	172	69	15723
[35]			676	633	0	0	307	2016	4432
			1137	1176	0	0	306	1009	3744
			2081	2237	0	0	304	504	3422
			[9]	609	640	2	4	257	490
[32]			1463	426	2	0	235	458	4410
[33]			1416	1074	2	1.5	227	448	6742
[19]	Radix-2/4	Virtex-7	982	1151	5	10.5	296	306	11767
[17]			982	1151			306	296	11010
		Artix-7	1070	1071				278	10420
[20]	Radix-2/4	Zynq 7000	5293	5518	34	0	265	127	9054
[18]		Artix-7	1740	643	4	5	200	224	8221
[16]	Radix-8/16	Zynq 7000	9824	4752	29	12	181	58	9941
		ZCU 102+	9083	3839			252		6970
		Artix-7	9985	4580			132		13703
Our Work	Radix-2/4	ZCU 104+	1713	1006	0	0	400	256	1096
		Artix-7	1924	886	0	0	205	256	2402

TABLE VI: Comparison Table for ML-KEM

Comparison Result for ML-DSA

Work	Radix	Board	LUT	FF	DSP	BRAM	Freq. (Mhz)	Latency (NTT)	ATP
[11]	Radix-2	Artix-7	9018	6292	16	2	250	256	17426
[12]			524	759	17	1	311	533	13923
[23]			1919	1301	8	2	96.9	1024	71004
[3]			690	771	2	2.5	273	1024	13091
[13]			2759	2037	4	7	163	512	31282
[4]			7451	5275	0	0	180	128	5298
[10]			4187	4387	18	0	297	286	10965
[14]			11000	5182	64	12	140	156	51480
[28]			755	844	2	0	237	1058	6942
		ZYNQ UltraScale+	765	844	2	0	577	1058	2870
[15]		XCKU060	5478	4955	12	6	250	256	15440
[29]		ZCU 102+	2341	1048	4	1	200	-	-
[30]			3821	2970	20	5	322	258	12676
[24]	Radix-4	Zynq7000	2386	932	8	2	217	256	8477
[21]		XCKU-060	1552	1867	6	5	324	271	6651
		Artix-7	1557	1652			158	271	13648
[16]	Radix-16	Artix-7	12720	7834	50	14.5	183	58	14047
		ZCU 102+	11963	7187			297	58	8507
		Zynq 7000	12141	7941			254	58	9988
Our Work	Radix-4	ZCU 104+	3426	1719	4	0	512	256	2513
		Artix-7	3403	1710			256		5003

TABLE VII: Comparison Table for ML-DSA

Summary: Three Designs, One Design Space

Across the three works the same question is asked at three different points of the radix / structure / resource space: what does one NTT coefficient really have to cost on an FPGA?

VLSID 2024

KiD

Unified radix-2 NTT

- One BFU serving both Kyber (12-bit) and Dilithium (23-bit) coefficients
- Shared adder/subtractor exploiting the FPGA fast-carry chain
- Conflict-free, fully pipelined dual-port memory mapping

Unified core, 2 Kyber / 1 Dilithium BFU

1384 LUT · 2 DSP · 4.5 BRAM @ 387 MHz — better ADP than the existing unified design

IEEE TCAS-I 2024

Winograd for NTT

Radix-8 / 16 butterfly

- First use of Winograd's FFT structure for NTT in lattice-based PQC
- Modified Hadamard gate ($a-b+v\cdot q$) removes in-layer modular reduction
- K-RED and Montgomery K-RED multipliers absorb n^{-1} for free

Radix-16 Winograd, Dilithium

58 cycles per NTT — ~5000 fewer LUTs than the radix-16 Cooley-Tukey BFU

FPL 2026

Lightweight Radix-4

Resource design space

- BRAM-free radix-4 BFU on distributed RAM with chain reduction
- 2-multiplier variant for memory-constrained deployments
- First multiplier-free NTT via Zech's logarithm (ML-KEM)

Best configuration, ZCU104+

ATP 1096 (ML-KEM) and 2513 (ML-DSA) — lowest reported in the literature

Takeaway

Latency, area and memory are not independent knobs. Radix, butterfly structure and the LUT / DSP / BRAM mix have to be chosen together — that joint choice, not any single optimisation, is what delivers the lowest area-delay product for NTT on FPGA.