RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography

Tim Fritzmann[1], Georg Sigl[1], and Johanna Sepúlveda[2]

[1] Technical University of Munich,

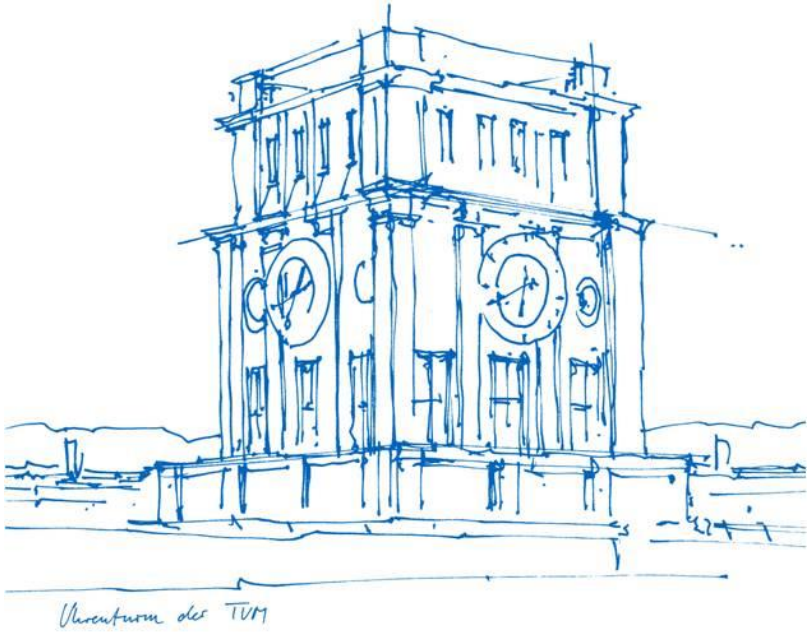
Chair of Security in Information Technology

[2] Airbus Defence and Space

14 – 17 September 2020

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung



Post-Quantum Cryptography

Lattice-based cryptography is largest class!

Implementation Goals:

HW/SW Co-Design

Content

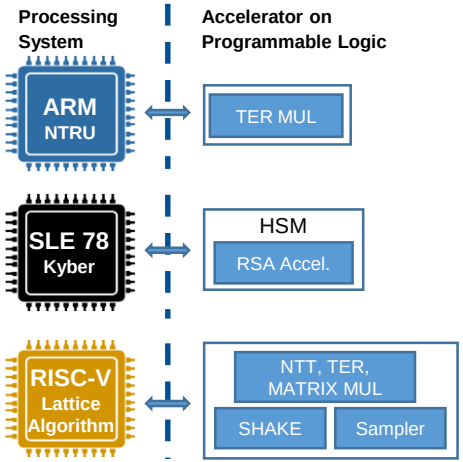
- HW/SW Codesign
- Contributions
- Polynomial Arithmetic
- Polynomial Sampling
- Integration into RISC-V
- Evaluation

HW/SW Codesign - Bottlenecks

- Which operations should be accelerated?
- **RLWE problem:**
  - **Ring arithmetic:** $\mathcal{R} = \mathbb{Z}_q/\langle \phi_n(x) \rangle$ with $\phi_n(x) = x^n + 1$
  - **RLWE instance:** $b = a \cdot s + e$

Bottlenecks are the polynomial multiplication and sampling!

HW/SW Co-Design



2018: Fritzmann et al., Efficient hardware/software co-design for NTRU
2019: Farahmand et al., Evaluating the potential for hardware acceleration of four NTRU-based key encapsulation mechanisms using software/hardware codesign

2019: *Albrecht et al., Implementing RLWE-based schemes using an RSA co-processor*

2019: *Fritzmann et al., Towards reliable and secure post-quantum co-processors based on RISC-V*
2019: *Banerjee et al., Sapphire: A configurable crypto-processor for post-quantum lattice-based protocol*
2020: *Wang et al., Parameterized hardware accelerators for lattice-based cryptography and their application to the HW/SW co-design of qTESLA*
...

➡ **Loosely Coupled Co-Processors**

Disadvantages loose coupling:

- Connection to a bus system leads to a high communication overhead
- Inflexibility when trying to decrease communication overhead
- High area consumption (I/O buffers, control circuit)

Content

- HW/SW Codesign
- Contributions
- Polynomial Arithmetic
- Polynomial Sampling
- Integration into RISC-V
- Evaluation

Contributions

- Investigation of tightly coupled accelerators for NewHope, Kyber and Saber

No complex bus communication	High flexibility	Small area
ISA must be extended	Core must be modified	

- Previous tightly coupled accelerators only accelerate modular arithmetic[1)]
- Powerful accelerators for <u>all</u> bottlenecks in this work
  - Vectorized modular arithmetic
  - NTT computations (Twiddle factor computations, vectorized butterfly operations, …)
  - Hash and sampling computations
- RISC-V integration and 29 new instructions

[1)] *Alkim et al., ISA extensions for finite field arithmetic - accelerating Kyber and NewHope on RISC-V*

Content

- ❑ HW/SW Codesign
- ❑ Contributions
- ❑ Polynomial Arithmetic
- ❑ Polynomial Sampling
- ❑ Integration into RISC-V
- ❑ Evaluation

Polynomial Multiplication

- Number Theoretic Transform (NTT)
- $\boldsymbol{c} = NTT^{-1}\big(NTT(\boldsymbol{a}) \circ NTT(\boldsymbol{s})\big)$

NTT:

$$\hat{a}_i = \sum_{j=0}^{n-1} \gamma_n^j \cdot \omega_n^{ij} \cdot a_j$$

Pre-Processing Twiddle Factors Coefficients

$\mathbf{NTT^{-1}}$:

$$a_i = n^{-1} \cdot \gamma_n^{-i} \sum_{j=0}^{n-1} \omega_n^{-ij} \cdot \hat{a}_j$$

Post-Processing

Number Theoretic Transform (NTT)

- ❑ Hybrid approach (NewHope and Kyber)
  - Large polynomial length ($n = 512$ and $n = 1024$):
    - On-the-fly Twiddle factor computation (reduces precomputations)
    - Address controller for $NTT_{br\rightarrow no}^{CT}$ and fast access FPR
  - Small polynomial length ($n = 256$):
    - LUT based approach (avoid *post-processing)*
    - $NTT_{no\rightarrow br}^{CT}$ and $INVNTT_{br\rightarrow no}^{GS}$ (*avoid bit-reversal)*

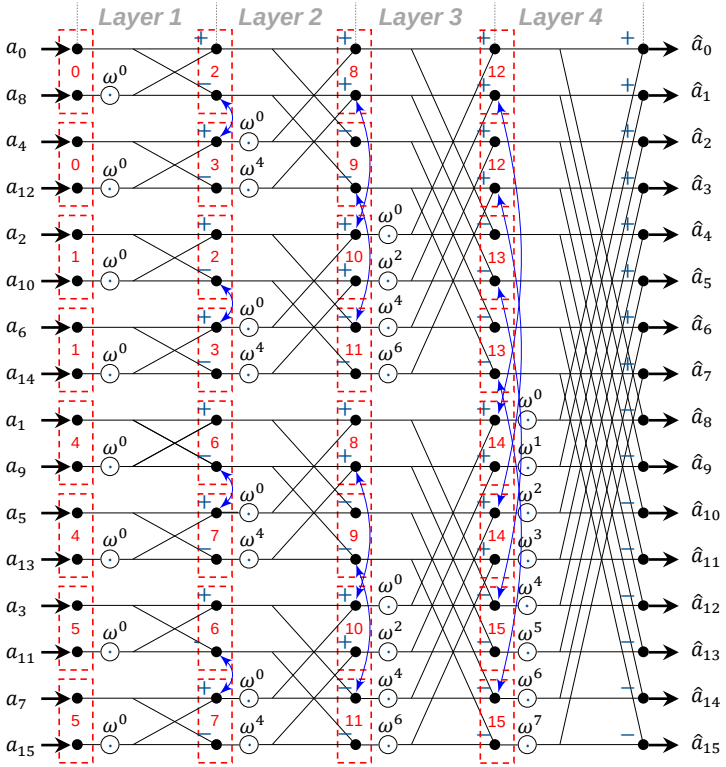
- ❑ NTT for general modulus (Saber)
  - Avoid precision *errors:* $q' > n \cdot q^2$
  - Let $q'_1, \ldots q'_k$ be co-prime and $\mathrm{q}' = \prod_{i=1}^{k} q'_i$
  - Recombination with CRT

Number Theoretic Transform (NTT)

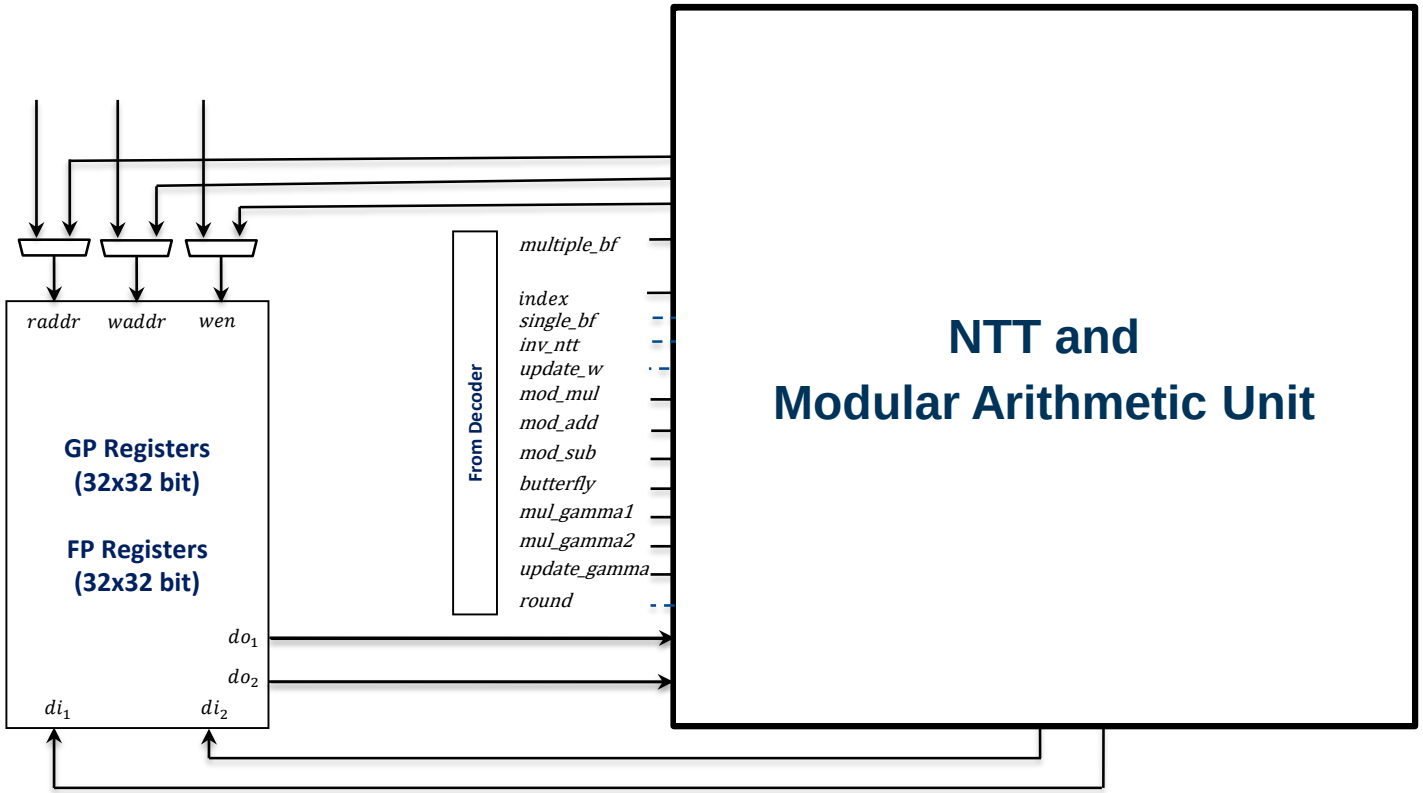
Optimizations:

- Calculate powers of ω on-the-fly
- Store 2 coeffs. in 1 register
- Swap coeffs. in HW
- Compute 2 BF operations in parallel
- Calculate next layer before finalizing previous

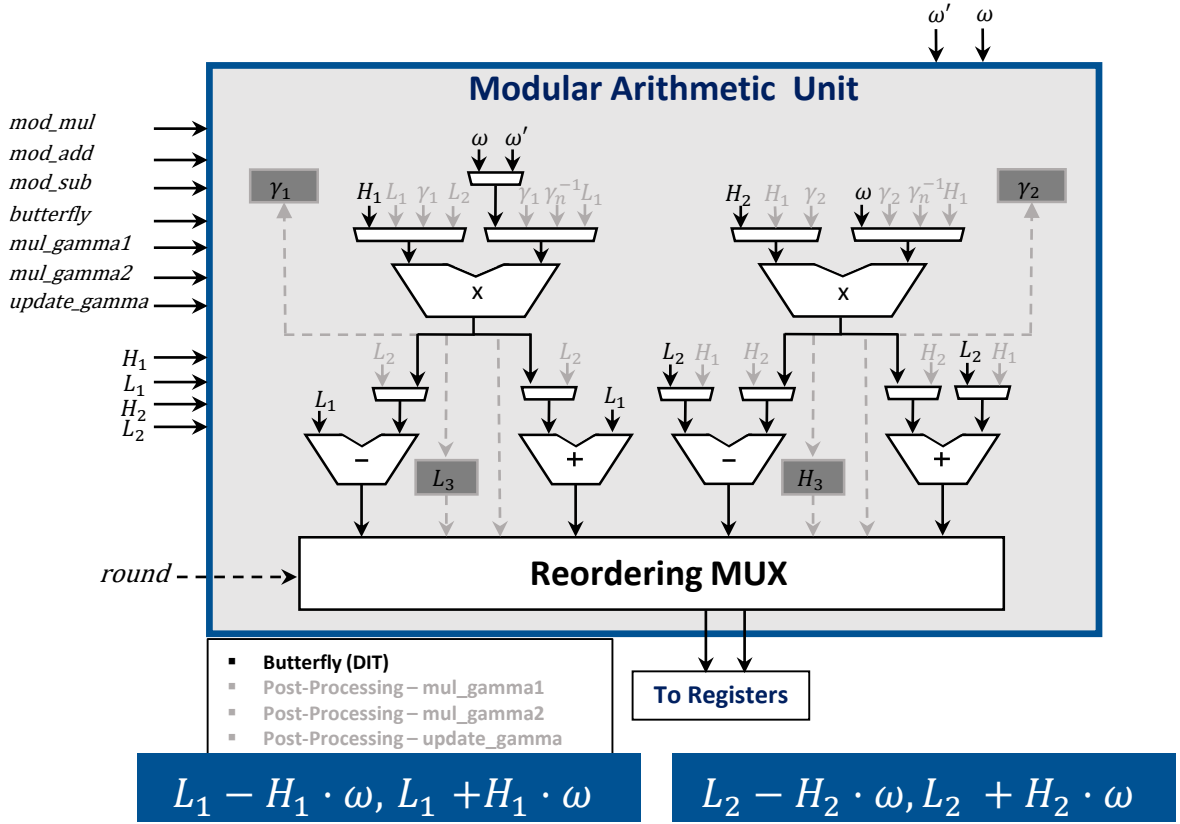
Register Content	Load coeffs.	BF0	BF1	BF0	BF1	Load coeffs.	…
R0	a_0, a_8	a_0, a_8		a_0, a_4		a_1, a_9	…
R1	a_4, a_{12}		a_4, a_{12}	a_8, a_{12}		a_5, a_{13}	…
R2	a_2, a_{10}	a_2, a_{10}			a_2, a_6	a_3, a_{11}	…
R3	a_6, a_{14}		a_6, a_{14}		a_{10}, a_{14}	a_7, a_{15}	…



NTT and Modular Arithmetic Unit



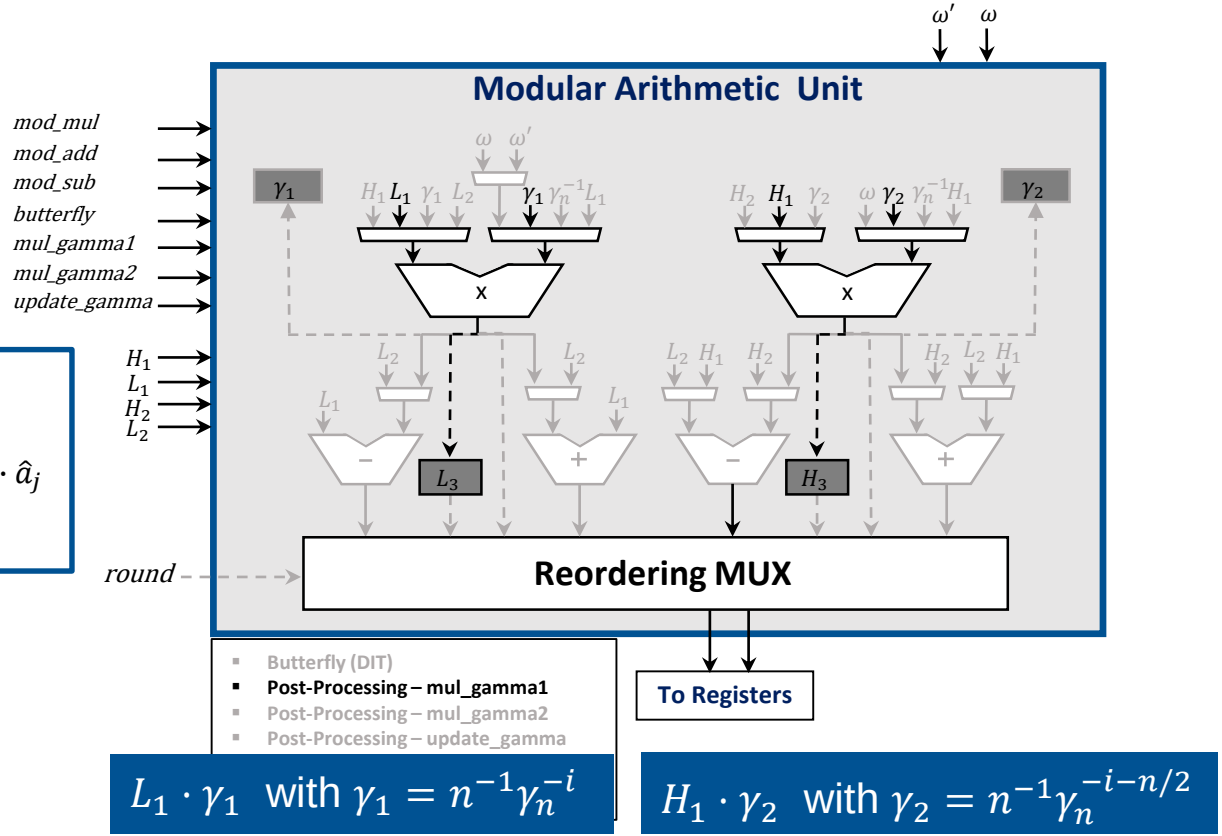
Modular Arithmetic Unit



Modular Arithmetic Unit

Remember NTT^{-1}:

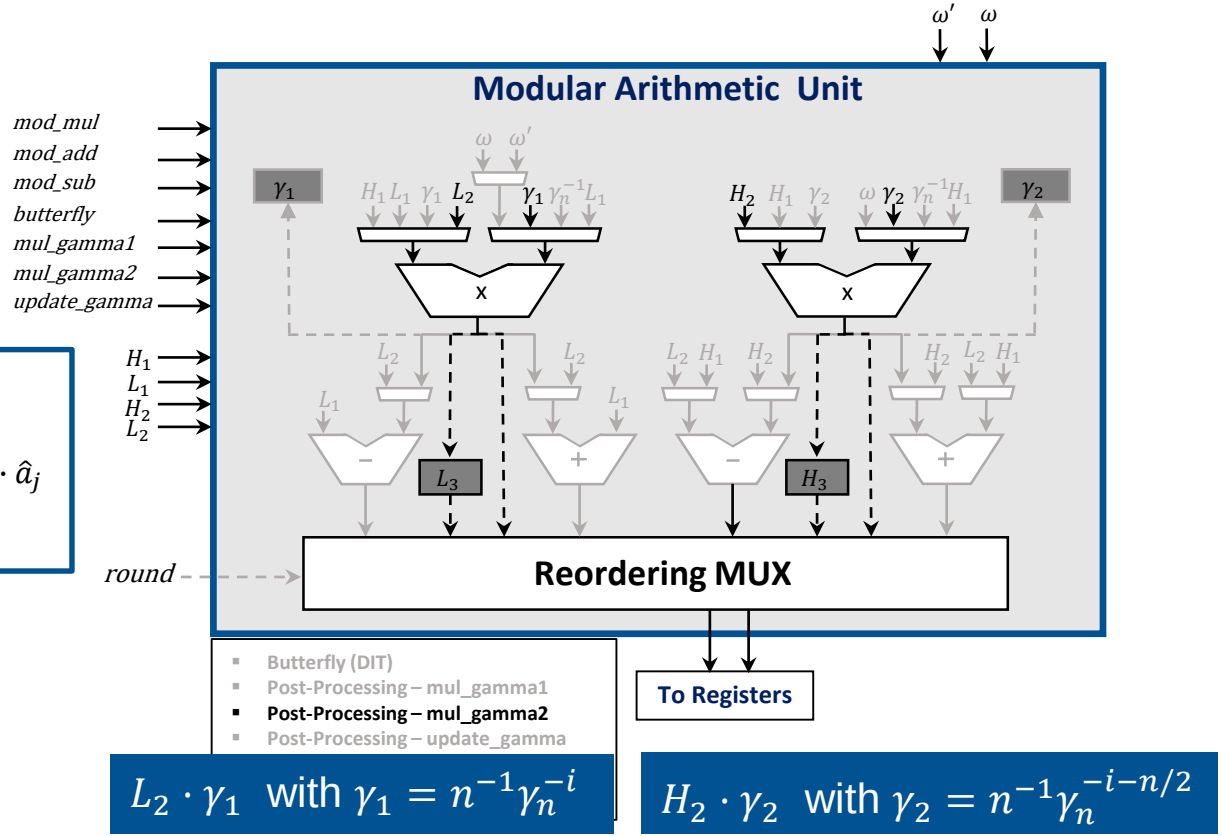
$$a_i = \underbrace{n^{-1} \cdot \gamma_n^{-i}}_{\text{Post-Processing}} \sum_{j=0}^{n-1} \omega_n^{-ij} \cdot \hat{a}_j$$



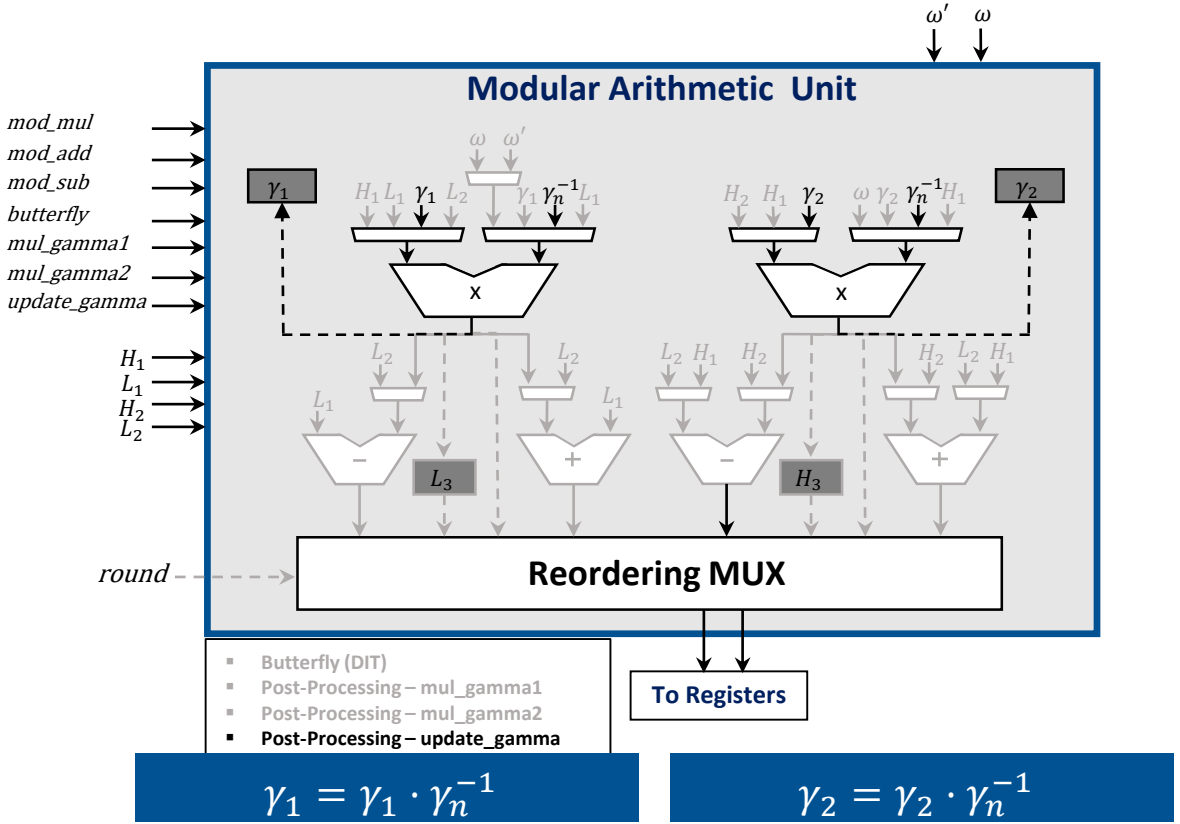
Modular Arithmetic Unit

Remember NTT^{-1}:

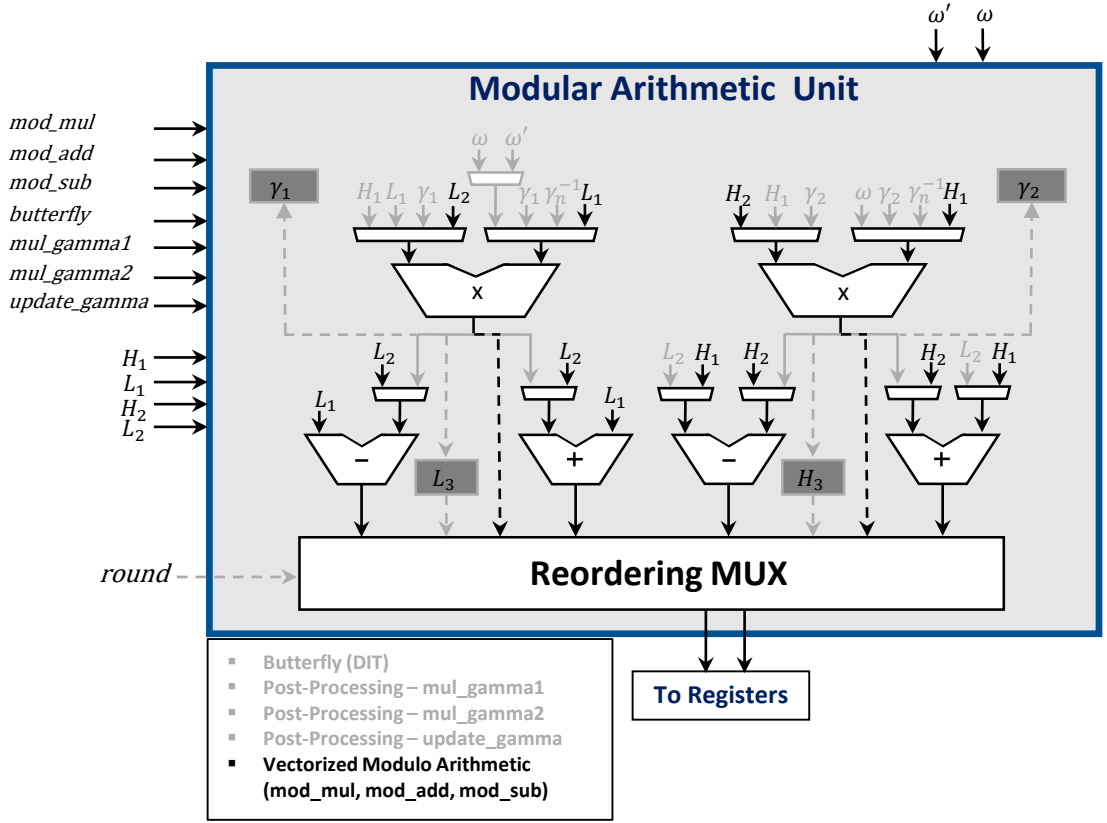
$$a_i = \underbrace{n^{-1} \cdot \gamma_n^{-i}}_{\text{Post-Processing}} \sum_{j=0}^{n-1} \omega_n^{-ij} \cdot \hat{a}_j$$



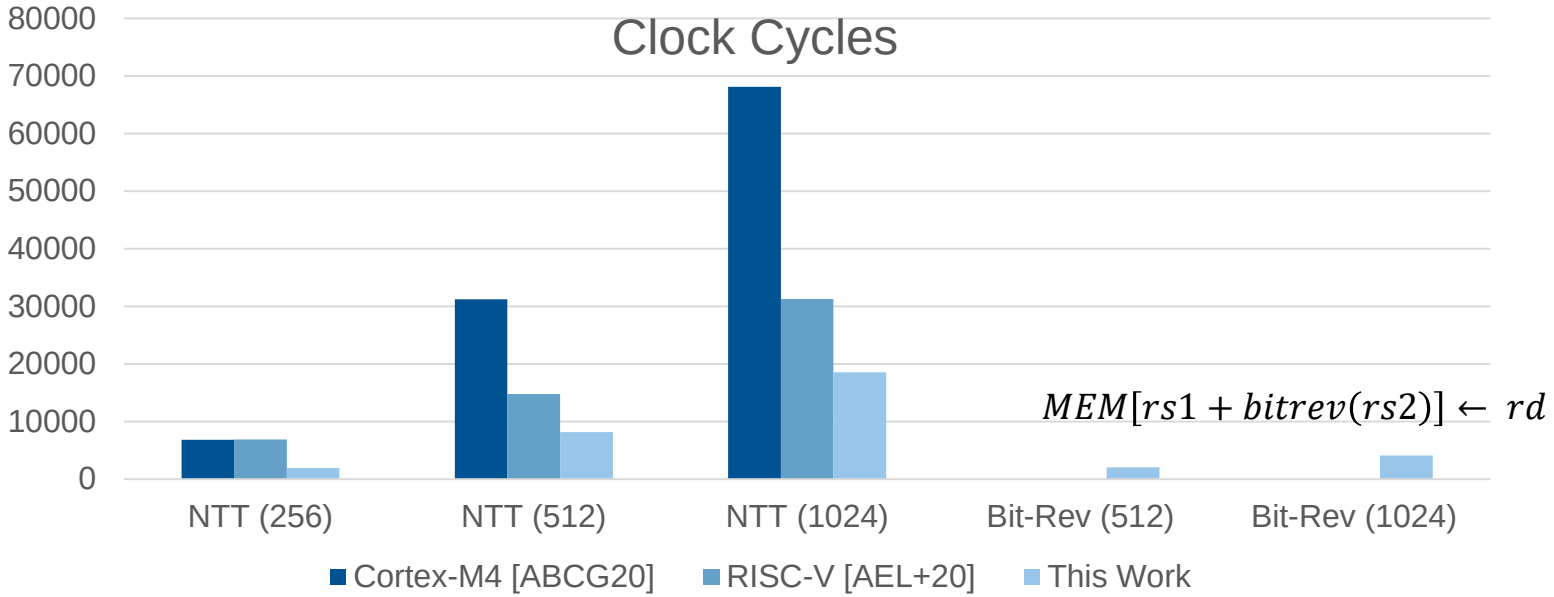
Modular Arithmetic Unit



Modular Arithmetic Unit



NTT Results



Precomputations:
From 7168 to 44 bytes (NewHope-1024)
From 3584 to 40 bytes (NewHope-512)

Karatsuba-based Multiplication

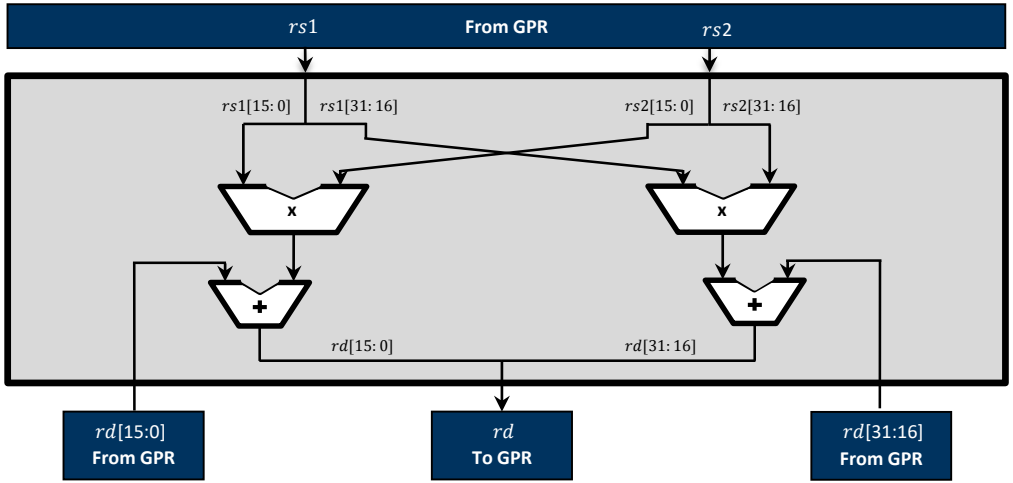
❑ Karatsuba splitting

- Split length-m polynomials $\boldsymbol{a}$ and $\boldsymbol{b}$ into length-m/2
- Lower part ($\boldsymbol{a^l}$, $\boldsymbol{b^l}$); and higher part ($\boldsymbol{a^h}$, $\boldsymbol{b^h}$)
- $\boldsymbol{c = ab = a^l b^l + \left(a^l b^h + a^h b^l\right)x^{m/2} + a^h b^h x^m}$ → 4 multiplications
- $\boldsymbol{c = a^l b^l + \left((a^l + b^h)(b^l + b^h) - a^l b^l - a^h b^h\right)x^{m/2} + a^h b^h x^m}$ → 3 different multiplications

❑ pq.mac (vectorized modular multiply accumulate)

- $rd[15:0] = (rs1[15:0] \cdot rs2[15:0] + rd[15:0])\ mod\ q'$
- $rd[31:16] = (rs1[31:16] \cdot rs2[31:16] + rd[31:16])\ mod\ q'$

PQ.MAC



$$rd[15:0] = (rs1[15:0] \cdot rs2[15:0] + rd[15:0])\ mod\ q'$$

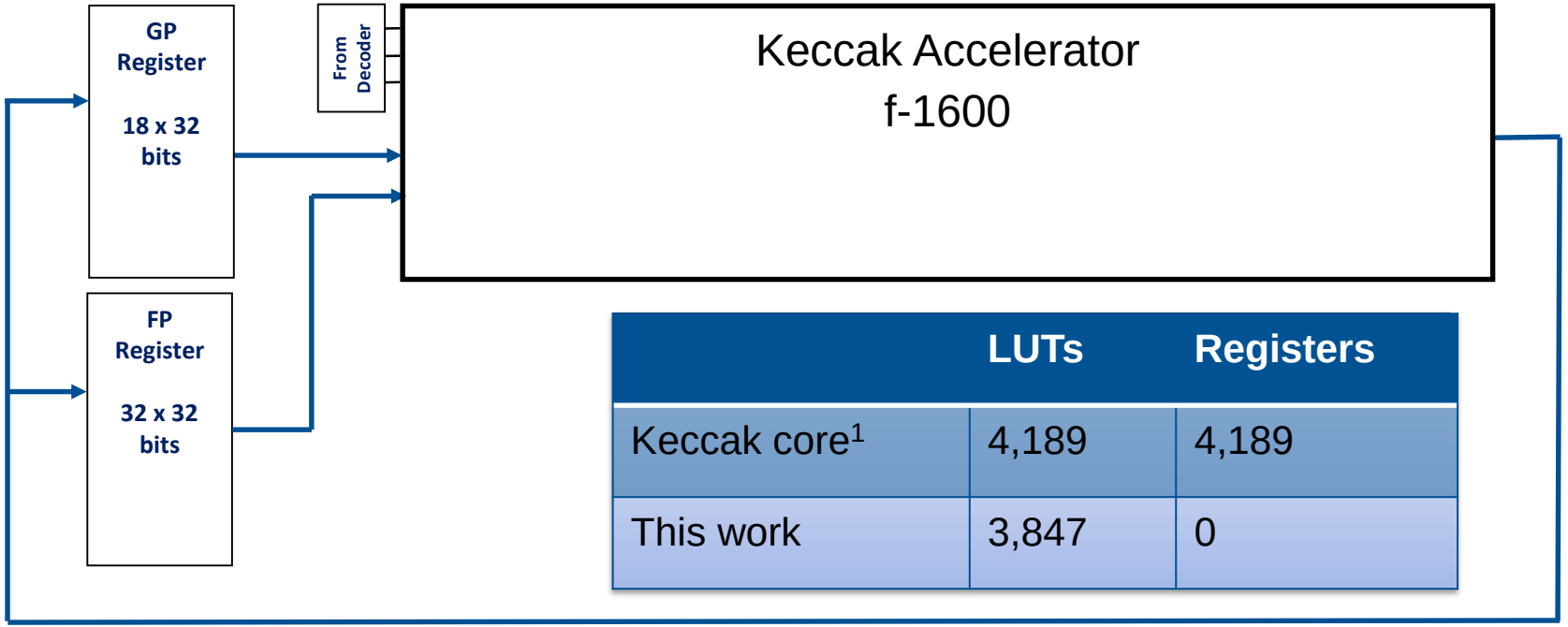
$$rd[31:16] = (rs1[31:16] \cdot rs2[31:16] + rd[31:16])\ mod\ q'$$

Multiplication 104,074 to 71,349 cycles
(31 % improvement)

Content

- HW/SW Codesign
- Contributions
- Polynomial Arithmetic
- Polynomial Sampling
- Integration into RISC-V
- Evaluation

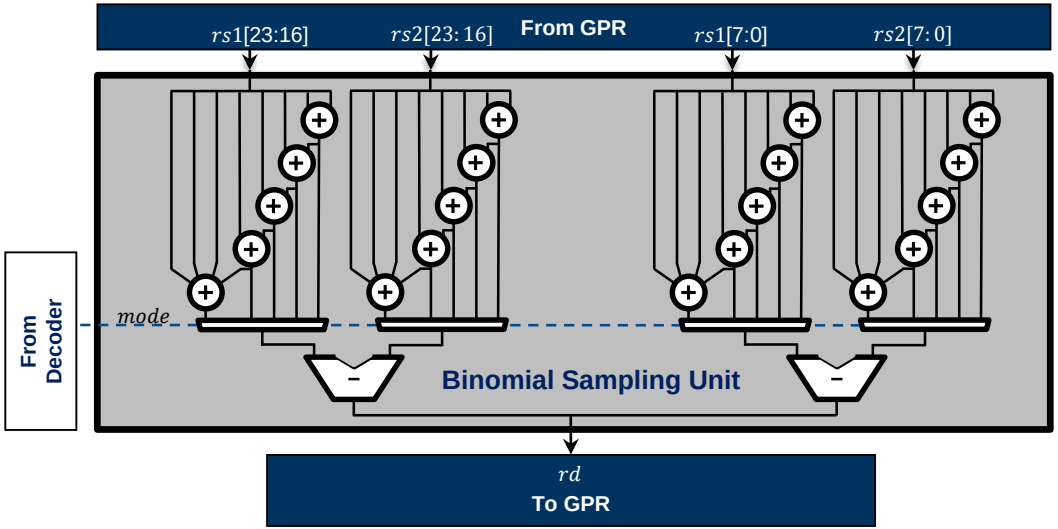
Keccak



	LUTs	Registers
Keccak core[1]	4,189	4,189
This work	3,847	0

[1] https://keccak.team/hardware.html

Binomial Sampling Unit

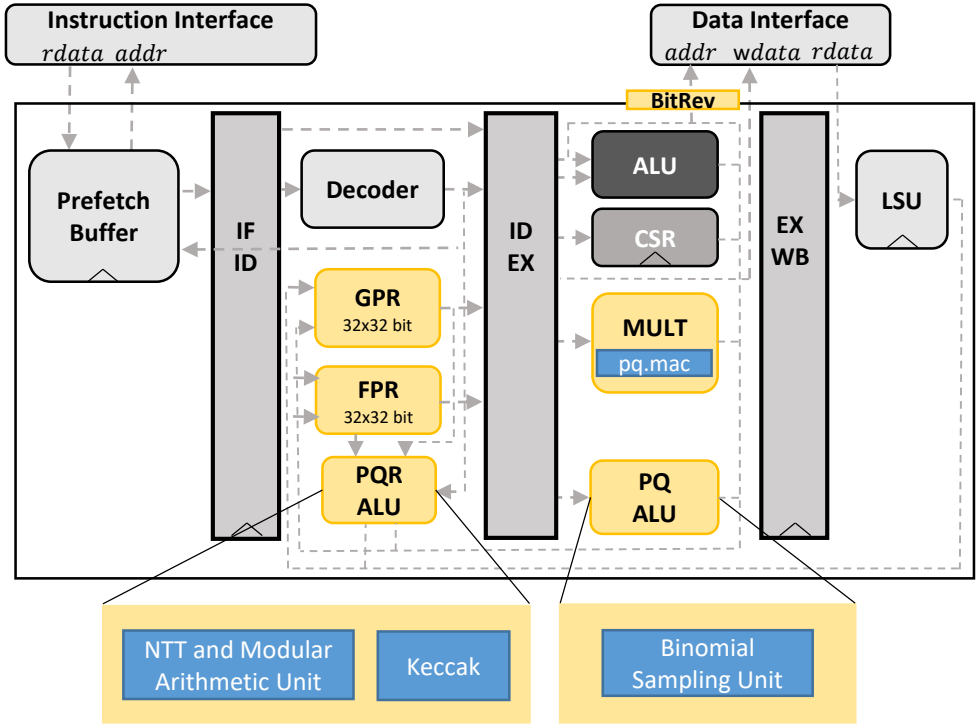


$$\Psi_k = \sum_{i=0}^{k-1} (b_i - b'_i) \mod q, \text{ where } b_i, b'_i \in \{0, 1\}$$

Content

- HW/SW Codesign
- Contributions
- Polynomial Arithmetic
- Polynomial Sampling
- Integration into RISC-V
- Evaluation

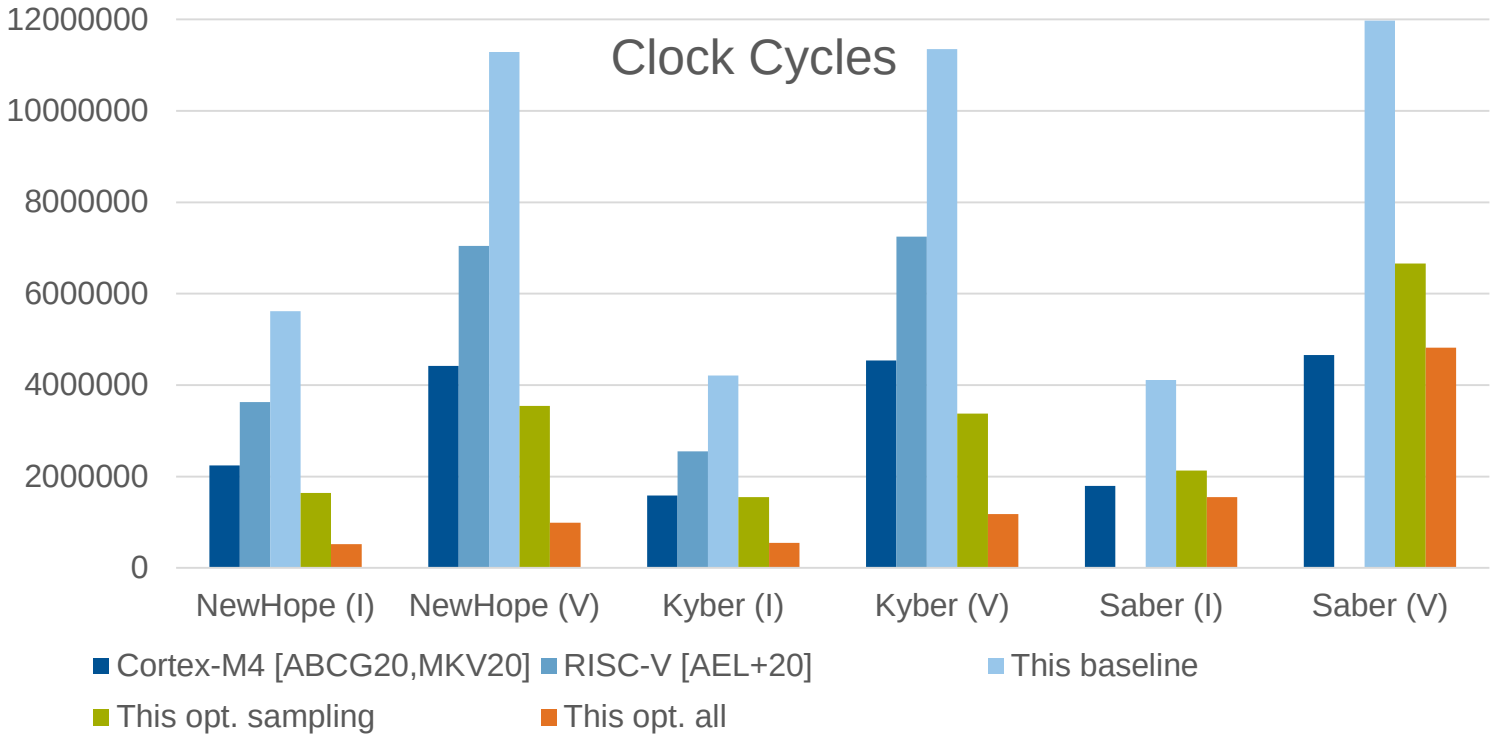
RISQ-V – Accelerators Integration



Content

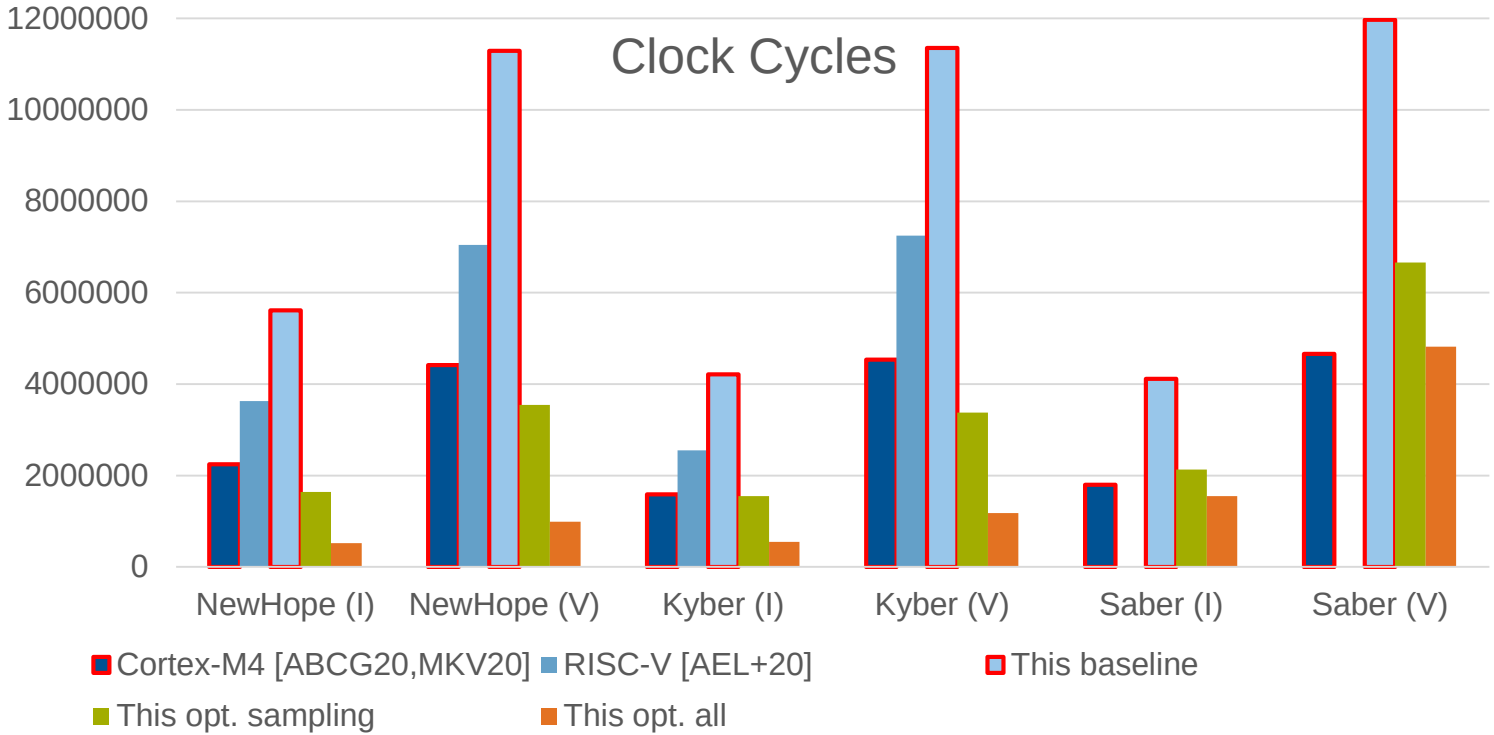
- HW/SW Codesign
- Contributions
- Polynomial Arithmetic
- Polynomial Sampling
- Integration into RISC-V
- Evaluation

Evaluation


Clock Cycles
12000000
10000000
8000000
6000000
4000000
2000000
0
NewHope (I)
NewHope (V)
Kyber (I)
Kyber (V)
Saber (I)
Saber (V)
Cortex-M4 [ABCG20,MKV20]
RISC-V [AEL+20]
This baseline
This opt. sampling
This opt. all

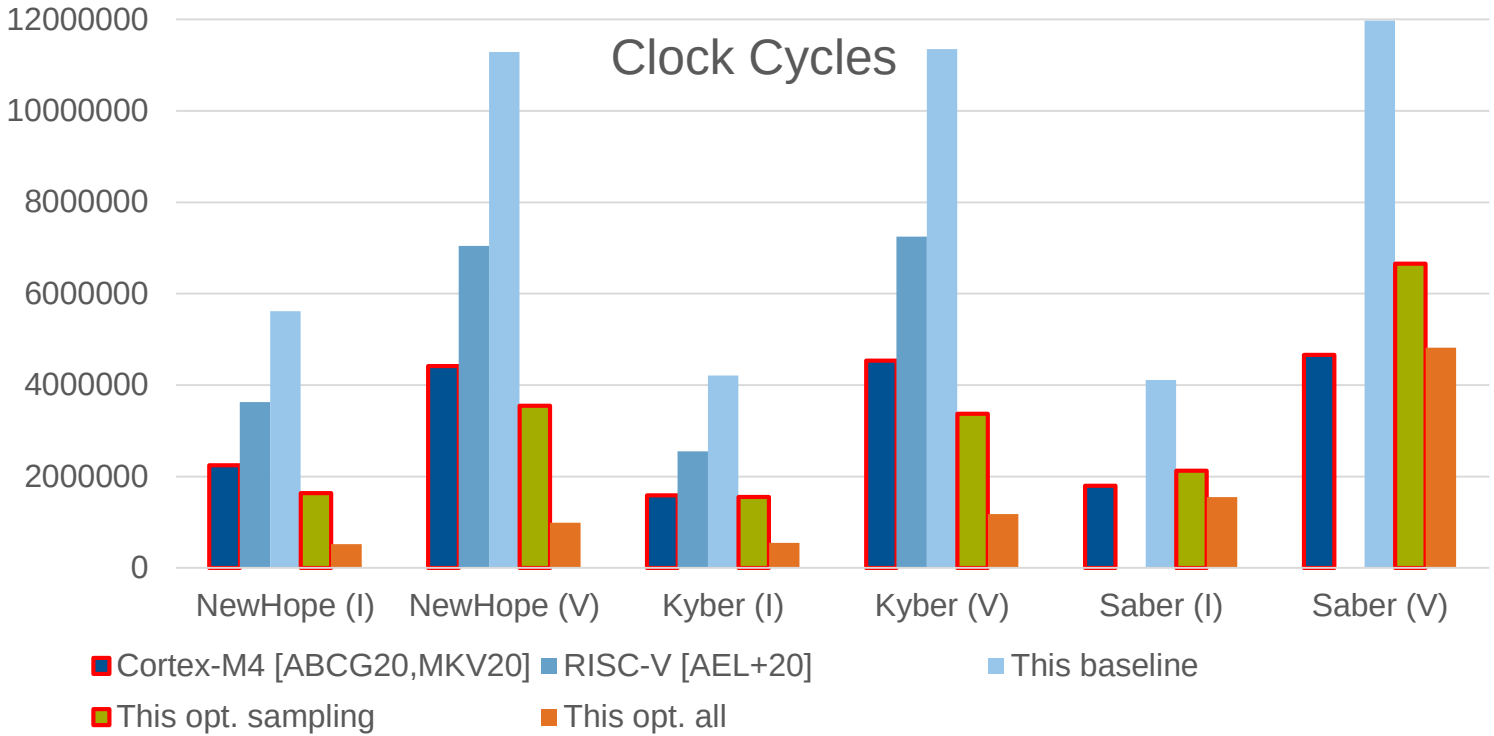
Evaluation

Large gap between baseline and optimized Cortex-M4



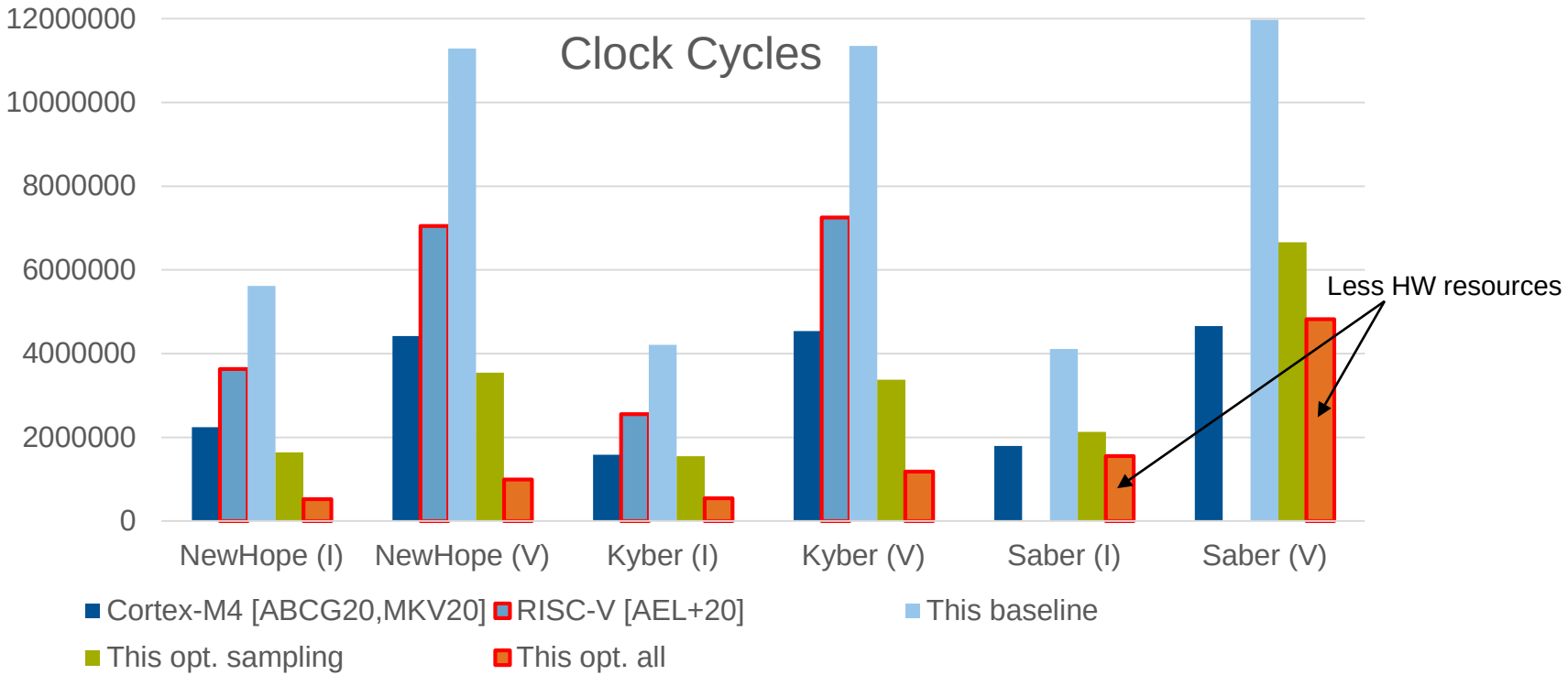
Evaluation

Optimize sampling alone brings good improvement



Evaluation

Stronger accelerators and accelerators for all bottlenecks are important



Evaluation ASIC

	Cell Count	Cell Area Combinatorial	Cell Area Sequential	Cell Area Memory
Original Platform	36,173	78,676 [μm]	92,304 [μm]	669,346[μm]
RISQ-V	57,413	143,198 [μm]	102,273 [μm]	669,346 [μm]

- Increase of cell count by 1.6
- Reduced energy consumption by factors of up to:
  - Up to 9.5 NewHope
  - Up to 7.7 Kyber
  - Up to 2.1 Saber

Conclusion

- RISQ-V: an enhanced RISC-V architecture that integrates powerful tightly coupled accelerators
- ISE for modular arithmetic, Keccak and binomial sampling
- Reuse existing hardware resources and strategies for decreasing memory access rate

Thank you for your attention!