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

Tim Fritzmann\textsuperscript{1}, Georg Sigl\textsuperscript{1}, and Johanna Sepúlveda\textsuperscript{2}

\textsuperscript{1} Technical University of Munich, Chair of Security in Information Technology
\textsuperscript{2} Airbus Defence and Space

14 – 17 September 2020
Post-Quantum Cryptography

Implementation Goals:

- Resources
- Flexibility
- Performance

HW/SW Co-Design

Lattice-based cryptography is largest class!

Announcement
Submission
Round 2
Round 3
Draft Standards

2016
2017
2019
2020
2022/2024

Tim Fritzmann, TU Munich
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: \( R = \mathbb{Z}_q / \langle \phi_n(x) \rangle \) 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)

Tim Fritzmann, TU Munich
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

- Previous tightly coupled accelerators only accelerate modular arithmetic

- Powerful accelerators for all 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)
- \( c = NTT^{-1}(NTT(a) \circ NTT(s)) \)

NTT:
\[
\hat{a}_i = \sum_{j=0}^{n-1} \gamma_n^j \cdot \omega_n^{ij} \cdot a_j
\]
Pre-Processing  Twiddle Factors  Coefficients

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 $q' = \prod_{i=1}^{k} q'_i$
  - Recombination with CRT
Number Theoretic Transform (NTT)

Optimizations:
- Calculate powers of $\omega$ 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

<table>
<thead>
<tr>
<th>Register</th>
<th>Load coeffs.</th>
<th>BF0</th>
<th>BF1</th>
<th>BF0</th>
<th>BF1</th>
<th>Load coeffs.</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>$a_0, a_8$</td>
<td></td>
<td>$a_0, a_4$</td>
<td></td>
<td>$a_1, a_9$</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>R1</td>
<td>$a_4, a_{12}$</td>
<td></td>
<td></td>
<td>$a_6, a_{12}$</td>
<td></td>
<td>$a_5, a_{13}$</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>$a_2, a_{10}$</td>
<td></td>
<td>$a_2, a_6$</td>
<td></td>
<td>$a_3, a_{11}$</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>R3</td>
<td>$a_6, a_{14}$</td>
<td></td>
<td>$a_{10, 14}$</td>
<td></td>
<td>$a_7, a_{15}$</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
NTT and Modular Arithmetic Unit

GP Registers (32x32 bit)

FP Registers (32x32 bit)

Rad dr  waddr  wen

d1

d2

d3

d4

From Decoder

multiple_bf

index

single_bf

inv_ntt

update_w

mod_mul

mod_add

mod_sub

butterfly

mul_gamma1

mul_gamma2

update_gamma

round

Tim Fritzmann, TU Munich
Modular Arithmetic Unit

- mod_mul
- mod_add
- mod_sub
- butterfly
- mul_gamma1
- mul_gamma2
- update_gamma

Butterfly (DIT)
- Post-Processing – mul_gamma1
- Post-Processing – mul_gamma2
- Post-Processing – update_gamma

To Registers

$\begin{align*}
L_1 &= H_1 \cdot \omega, \quad L_1 + H_1 \cdot \omega \\
L_2 &= H_2 \cdot \omega, \quad L_2 + H_2 \cdot \omega
\end{align*}$

Tim Fritzmann, TU Munich
Modular Arithmetic Unit

Remember NTT\(^{-1}\):
\[ a_i = n^{-1} \cdot \gamma_n^{-i} \sum_{j=0}^{n-1} \omega_n^{-ij} \hat{a}_j \]

Post-Processing

Butterfly (DIT)
- Post-Processing – mulGamma1
- Post-Processing – mulGamma2
- Post-Processing – update Gamma

To Registers

\[ L_1 \cdot \gamma_1 \text{ with } \gamma_1 = n^{-1} \gamma_n^{-i} \]
\[ H_1 \cdot \gamma_2 \text{ with } \gamma_2 = n^{-1} \gamma_n^{-i-n/2} \]
Modular Arithmetic Unit

Remember NTT⁻¹:

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

Post-Processing

\[ \omega' = \omega \]

\[ H_2 \cdot \gamma_2 \quad \text{with} \quad \gamma_2 = n^{-1} \gamma_n^{-i-n/2} \]

\[ L_2 \cdot \gamma_1 \quad \text{with} \quad \gamma_1 = n^{-1} \gamma_n^{-i} \]

Post-Processing

- Butterfly (DIT)
- Post-Processing – mul_gamma1
- Post-Processing – mul_gamma2
- Post-Processing – update_gamma

To Registers
Modular Arithmetic Unit

\[ \gamma_1 = \gamma_1 \cdot \gamma_n^{-1} \]

\[ \gamma_2 = \gamma_2 \cdot \gamma_n^{-1} \]

- Butterfly (DIT)
- Post-Processing – \text{mul\_gamma1}
- Post-Processing – \text{mul\_gamma2}
- Post-Processing – \text{update\_gamma}

Tim Fritzmann, TU Munich
Modular Arithmetic Unit

- Butterfly (DIT)
- Post-Processing – mul_gamma1
- Post-Processing – mul_gamma2
- Post-Processing – update_gamma
- Vectorized Modulo Arithmetic
  (mod_mul, mod_add, mod_sub)

Tim Fritzmann, TU Munich
NTT Results

Clock Cycles

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

- **Karatsuba splitting**
  - Split length-$m$ polynomials $a$ and $b$ into length-$m/2$
  - Lower part $(a^l, b^l)$; and higher part $(a^h, b^h)$
  - $c = ab = a^l b^l + (a^l b^h + a^h b^l)x^{m/2} + a^h b^h x^m$
  - $c = a^l b^l + ((a^l + b^h)(b^l + b^h) - a^l b^l - a^h b^h)x^{m/2} + a^h b^h x^m$

- **pq.mac (vectorized modular multiply accumulate)**
  - $rd[15:0] = (rs1[15:0] \cdot rs2[15:0] + rd[15:0]) \mod q'$
  - $rd[31:16] = (rs1[31:16] \cdot rs2[31:16] + rd[31:16]) \mod q'$

4 multiplications

3 different multiplications

Tim Fritzmann, TU Munich
\[ \text{rd}[15:0] = (\text{rs1}[15:0] \cdot \text{rs2}[15:0] + \text{rd}[15:0]) \mod q' \]

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

Multiplication 104,074 to 71,349 cycles

(31 % improvement)
Content

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

Keccak Accelerator f-1600

<table>
<thead>
<tr>
<th>LUTs</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keccak core¹</td>
<td>4,189</td>
</tr>
<tr>
<td>This work</td>
<td>3,847</td>
</tr>
</tbody>
</table>

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

Instruction Interface
\[ rdata \rightarrow addr \]

Data Interface
\[ addr \rightarrow wdata \rightarrow rdata \]

Prefetch Buffer

IF ID

Decoder

ID EX

EX WB

LSU

GPR 32x32 bit

FPR 32x32 bit

PQR ALU

ALU

CSR

MULT

pq,mac

PQ ALU

NTT and Modular Arithmetic Unit

Keccak

Binomial Sampling Unit
Content

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

Large gap between baseline and optimized Cortex-M4

Clock Cycles

Tim Fritzmann, TU Munich
Optimize sampling alone brings good improvement

Clock Cycles

Evaluation

Tim Fritzmann, TU Munich
Evaluation

Stronger accelerators and accelerators for all bottlenecks are important

Clock Cycles

- 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

Less HW resources
## Evaluation ASIC

<table>
<thead>
<tr>
<th></th>
<th>Cell Count</th>
<th>Cell Area Combinatorial [μm]</th>
<th>Cell Area Sequential [μm]</th>
<th>Cell Area Memory [μm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Platform</td>
<td>36,173</td>
<td>78,676</td>
<td>92,304</td>
<td>669,346</td>
</tr>
<tr>
<td>RISQ-V</td>
<td>57,413</td>
<td>143,198</td>
<td>102,273</td>
<td>669,346</td>
</tr>
</tbody>
</table>

- 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!