TU Graz

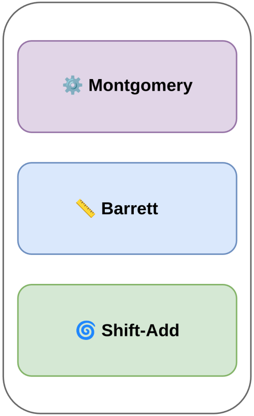
SCIENCE
PASSION
TECHNOLOGY

Hardware Evaluation of Modular Multiplication methods

Teodora Alexandrescu

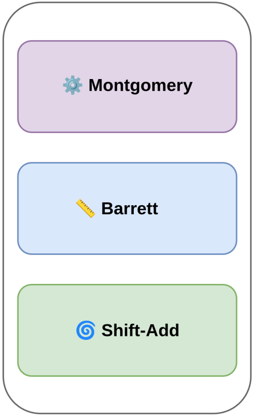
Supervisor: Aikata Aikata

Modular Reduction Algorithms



- Dilithium & Kyber moduli for proof-of-concept
- Area and performance results
- Suggest appropriate implementation strategies

Modular Reduction Algorithms



- Dilithium & Kyber moduli for proof-of-concept
- Area and performance results
- Suggest appropriate implementation strategies

Applications of Modular Multiplication

- Schemes based on LWE (or MLWE, RLWE)

$$\mathbf{A} \cdot \mathbf{s} + e \approx \mathbf{b} \pmod{q}$$

- Post-quantum schemes: Kyber, Dilithium, Falcon $\rightarrow$ MLWE/ RLWE

Applications of Modular Multiplication

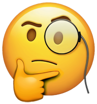
- Schemes based on LWE (or MLWE, RLWE)

$$\mathbf{A} \cdot \mathbf{s} + e \approx \mathbf{b} \pmod{q}$$

- Post-quantum schemes: Kyber, Dilithium, Falcon $\rightarrow$ MLWE/ RLWE

Why Optimize Modular Reduction?

- Schoolbook reduction: long division by q ($x \mod q$)
- Expensive operation on digital platforms
- Solution?



Montgomery Reduction

- Montgomery Form
  - $\bar{x} = xR \ mod \ Q$
- Replace division with shifts by **R**.

Algorithm 1 Montgomery Reduction (REDC)

Require: $T, Q, R = 2^n$, where $\gcd(R, Q) = 1$.
Output: $TR^{-1} \mod Q$

1: $m \leftarrow (T \mod R)Q' \mod R$
2: $t \leftarrow (T + m \cdot Q)/R$
3: **if** $t \geq Q$ **then**
4: $\quad t \leftarrow T - Q$
5: **end if**
6: **return** t

Barrett Reduction

- Approximates q
- Precomputation term μ
- Leverage precomputed term μ to avoid division

Algorithm 3 Barrett Reduction

Require: $X, Y, M,\ X, Y < M$
Output: $X \cdot Y \mod M$

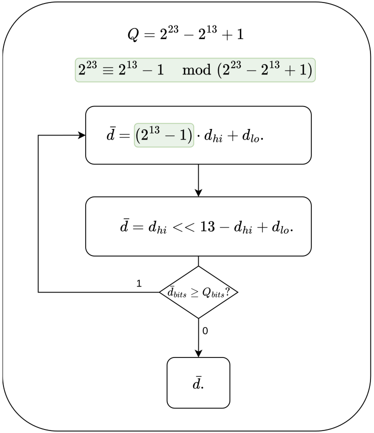
1: $T \leftarrow X \cdot Y$
2: $\mu \leftarrow \lfloor (1/M) \cdot 2^{2k} \rfloor$
3: $q \leftarrow \lfloor (T \cdot \mu)/2^{2k} \rfloor$
4: $r \leftarrow T - q \cdot M$
5: **if** $r \geq M$ **then**
6: $\quad r \leftarrow r - M$
7: **end if**
8: **return** r

- In practical applications, such as lattice-based PQC, often the moduli used are almost powers of two
- General Montgomery and Barrett techniques do not exploit the pseudo-Mersenne form of the moduli

- In practical applications, such as lattice-based PQC, often the moduli used are almost powers of two
- General Montgomery and Barrett techniques do not exploit the pseudo-Mersenne form of the moduli

Shift-Add Reduction

- *Special form modulus*
  - Exploit modular equivalences
  - Compute modulus with *shifts and additions*, **recursively**.
- Each fold
  - Shrinks the size of the number
  - Maintains congruence modulo q



Can we avoid runtime recursion?

- Precompute recursive approach with pen & paper and *avoid runtime recursion* altogether

$$d[23:0] = 2^{12} d[23:12] + d[11:0]$$

$$\boxed{use\ 2^{12} \equiv 2^9 + 2^8 - 1.}$$

$$= 2^9 d[23:12] + 2^8 d[23:12] - d[23:12] + c[11:0]$$

$$\ldots$$

$$\boxed{\begin{aligned} &= 2^9(d[23:15] + d[23:16] + d[14:12]) + \\ &\quad 2^8(d[23:15] + d[23:16] + d[15:12]) - \\ &\quad d[23:15] - d[23:16] - d[23:12] + d[11:0]. \end{aligned}}$$

Precomputed Shift-Add Kyber

$$d[45:0] = 2^{23} d[45:23] + d[22:0]$$

$$\boxed{use\ 2^{23} \equiv 2^{13} - 1.}$$

$$= (2^{13} - 1) d[45:23] + d[22:0]$$

$$\ldots$$

$$\boxed{\begin{aligned} &= 2^{13}(d[32:23] + d[42:33] + d[45:43]) - \\ &\quad d[45:23] - d[45:33] - d[45:43]) + d[22:0]. \end{aligned}}$$

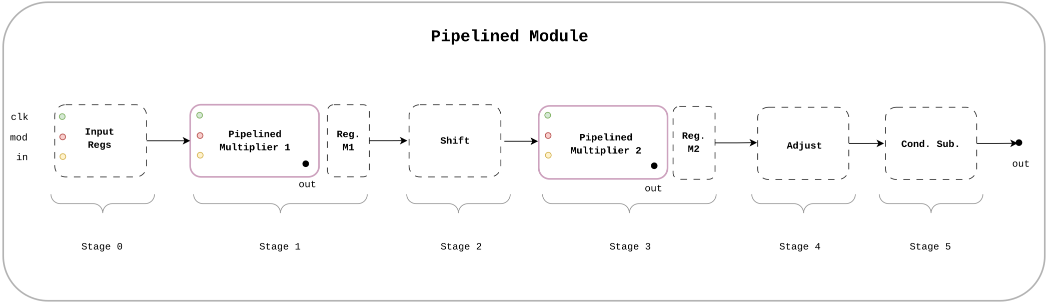
Precomputed Shift-Add Dilithium

Design & Implementation

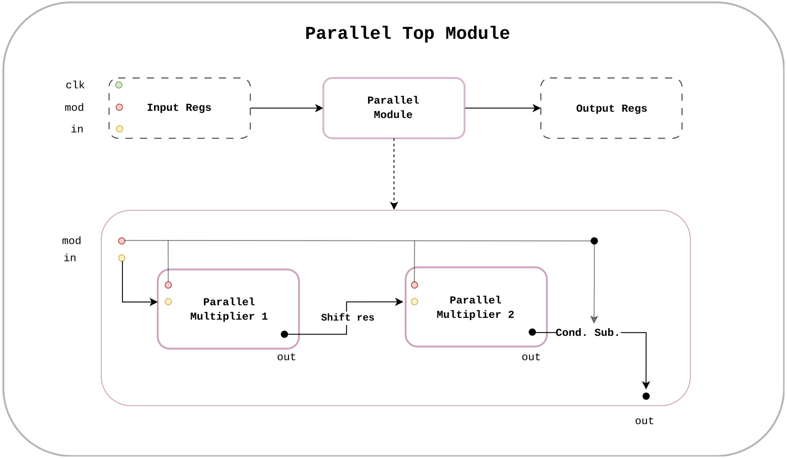
- Approaches and targets
  - *Pipelined*: achieve high throughput
  - *Digit-parallel*: achieve results using few clock cycles
  - *Digit-/Bit-serial*: saves area, but latency is high
- Batch of 64 inputs
- Hardware Description Language: SystemVerilog
- Synthesis Tool: **Xilinx Vivado 2024.2**
- Testing Board: **Artix-7 AC701 Evaluation Platform**

Pipelined

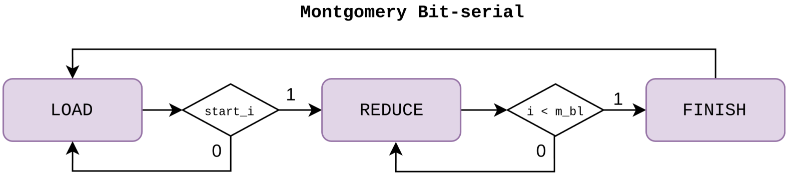
- Approach used by both Barrett & Montgomery



Digit-parallel

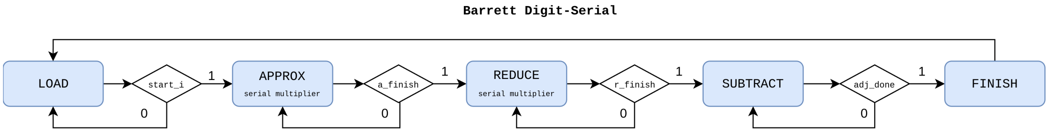


Bit-serial



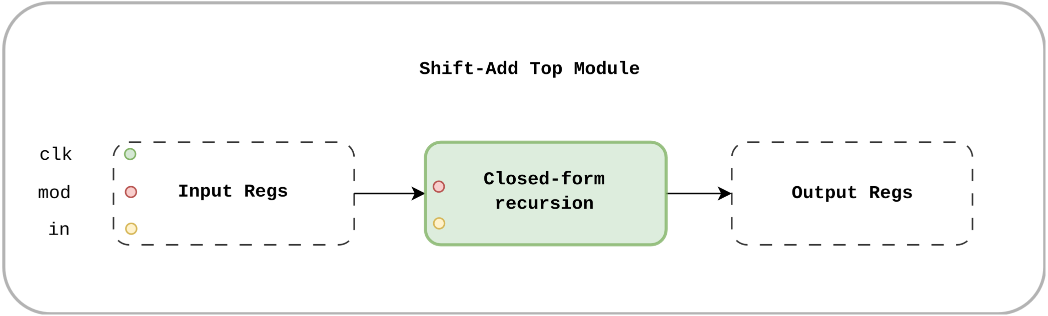
- FSM driven design
- Bit-by-bit processing of an input

Digit-serial



- FSM driven design
- Uses digit-serial modules with *16×16-bit MAC units* for iterative multiplication

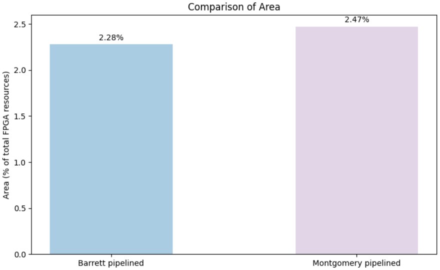
Closed-form Shift-Add

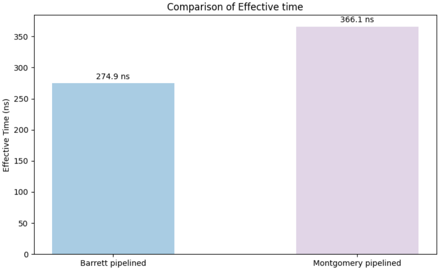


- Implements closed-form logic in one submodule
- Input and output are just registered in the top module

Theory aside - let's put these devices under test!

Pipelined

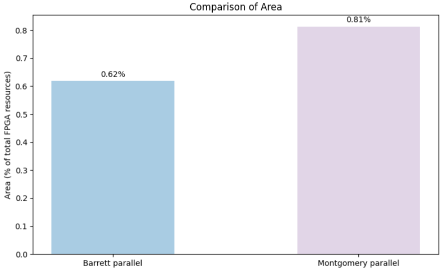


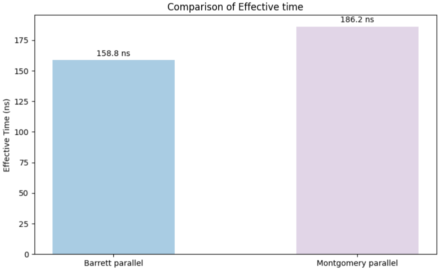


	LUT	FF	DSP
Barrett	1392	2306	26
Montgomery	1606	2441	26

	Frequency (MHz.)	Throughput (outs/s)
Barrett	156.39	152.76
Montgomery	117.46	114.73

Digit-parallel

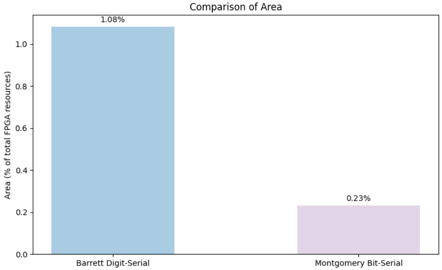


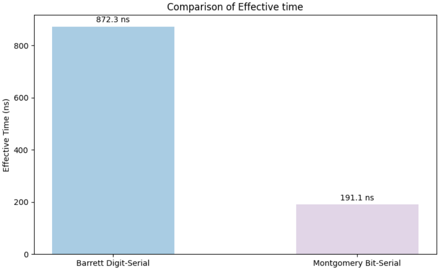


	LUT	FF	DSP
Barrett	960	129	26
Montgomery	1606	2441	26

	Frequency (MHz.)	Throughput (outs/s)
Barrett	31.70	6.34
Montgomery	26.85	5.37

Digit-/Bit-Serial

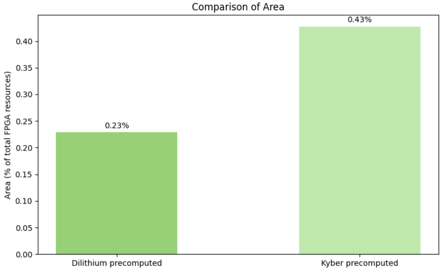


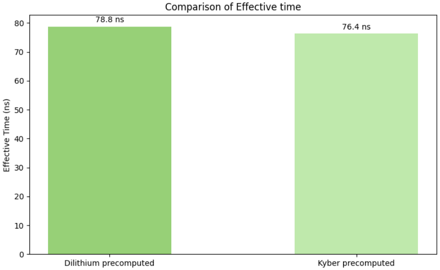


	LUT	FF	DSP
Barrett	948	1008	2
Montgomery	403	75	0

	Frequency (MHz.)	Throughput (outs/s)
Barrett	120.36	1.14
Montgomery	130.80	4.69

Closed-Form Shift-Add





	LUT	FF	DSP
Dilithium	299	111	0
Kyber	631	138	0

	Frequency (MHz.)	Throughput (outs/s)
Dilithium	50.78	10.15
Kyber	53.06	10.61

Conclusion

- Modular multiplication is often the *performance bottleneck* in cryptographic algorithms
- High performance solutions demand further research

Conclusion

- Modular multiplication is often the *performance bottleneck* in cryptographic algorithms
- High performance solutions demand further research

Conclusion

- Modular multiplication is often the *performance bottleneck* in cryptographic algorithms
- High performance solutions demand further research

TU Graz

SCIENCE
PASSION
TECHNOLOGY

Hardware Evaluation of Modular Multiplication methods

Teodora Alexandrescu

Supervisor: Aikata Aikata