Seminar of cryptology in Caen — October 27, 2011

Hardware implementation of the arithmetic of fields of characteristic 2 and 3

Nicolas Estibals

CARAMEL project-team, LORIA, Nancy Université / CNRS / INRIA

Nicolas.Estibals@loria.fr

Introduction

- Curve-based cryptography relies on finite fields
  - Among them: small characteristic fields (2 and 3)
    - Example: pairings on supersingular elliptic curves
- Need for hardware implementations
  - Embedded systems (RFID, smart card, sensors, . . .)
  - High-performance cryptographic coprocessor (bank servers, . . .)
    - No native support in CPUs (Partial support in Intel AVX instruction set)

Introduction

- Curve-based cryptography relies on finite fields
  - Among them: small characteristic fields (2 and 3)
    - Example: pairings on supersingular elliptic curves
- Need for hardware implementations
  - Embedded systems (RFID, smart card, sensors, . . .)
  - High-performance cryptographic coprocessor (bank servers, . . .)
    - No native support in CPUs (Partial support in Intel AVX instruction set)
- Reconfigurable hardware (FPGA)
  - great prototyping platform
  - flexibility when increasing security is needed

Introduction

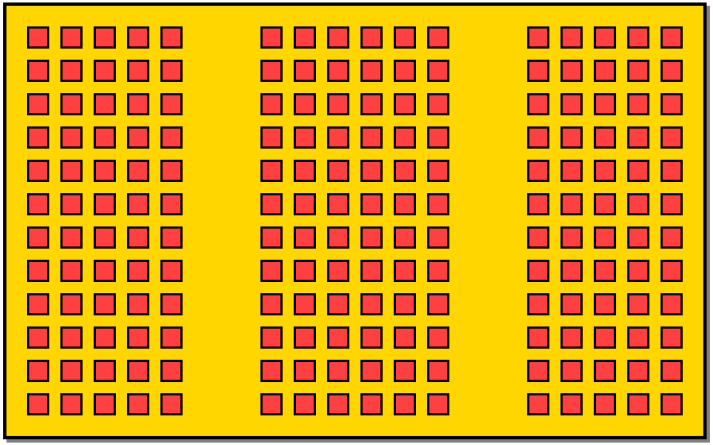
- Curve-based cryptography relies on finite fields
  - Among them: small characteristic fields (2 and 3)
    - Example: pairings on supersingular elliptic curves
- Need for hardware implementations
  - Embedded systems (RFID, smart card, sensors, . . .)
  - High-performance cryptographic coprocessor (bank servers, . . .)
    - No native support in CPUs (Partial support in Intel AVX instruction set)
- Reconfigurable hardware (FPGA)
  - great prototyping platform
  - flexibility when increasing security is needed
- Join work with:
  - Jean-Luc Beuchat, LCIS, University of Tsukuba, Japan.
  - Jérémie Detrey, CARAMEL project-team, INRIA Nancy Grand-Est, France.

FPGA architecture

- FPGAs are composed of:

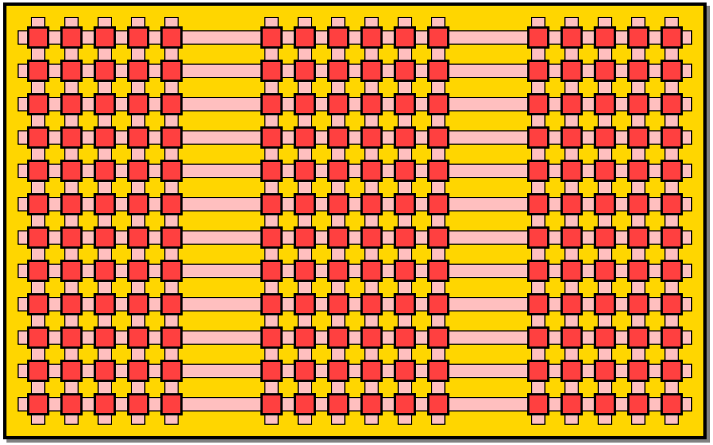
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells



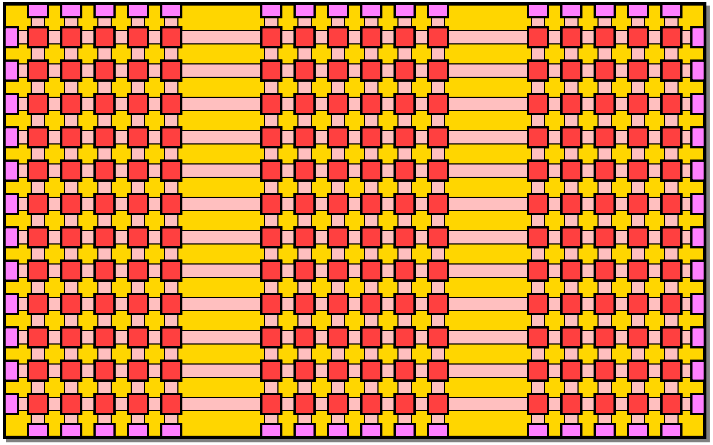
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix



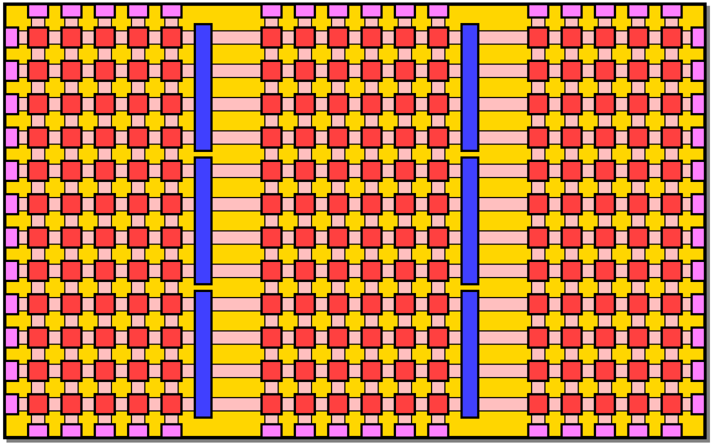
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells



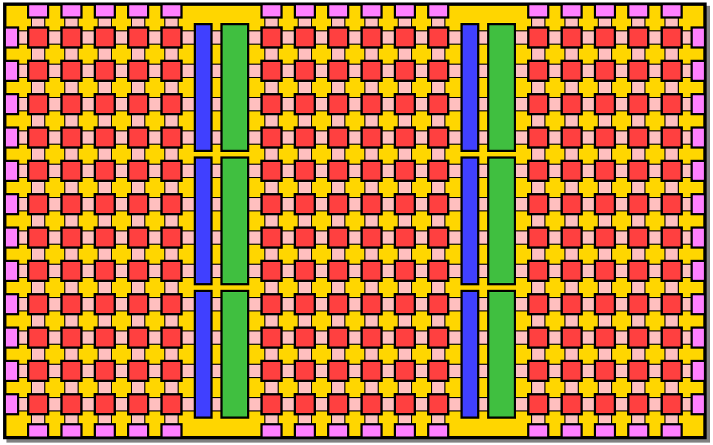
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks



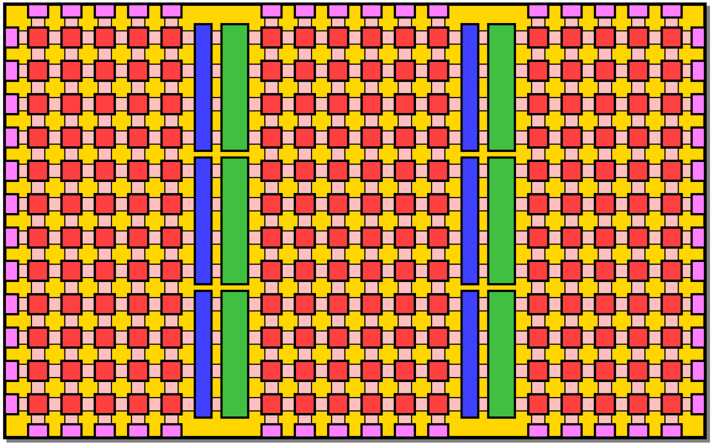
FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers



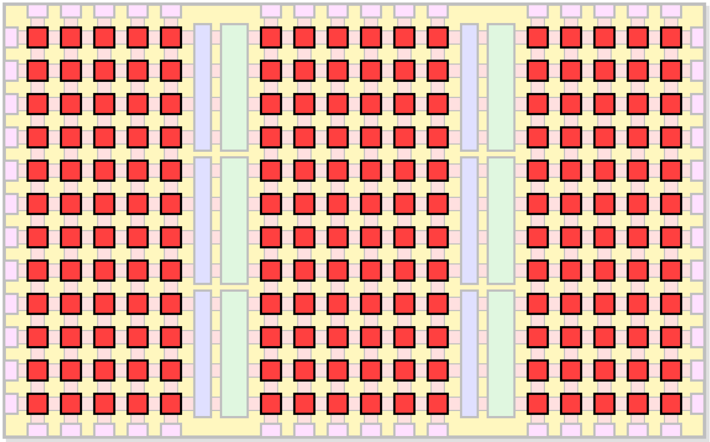
FPGA architecture

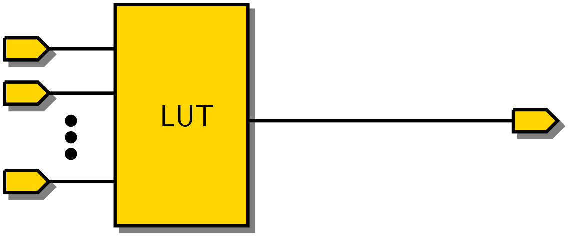
- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.



FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.

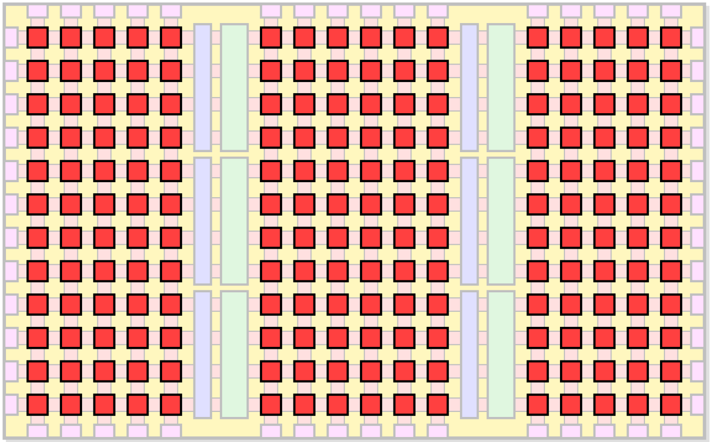


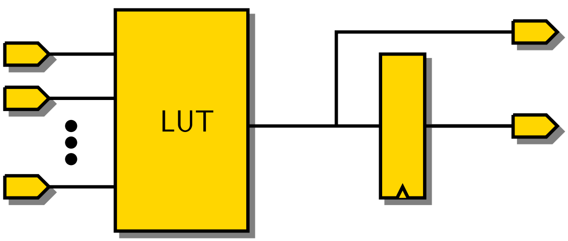


- Inside a logic cell:
  - connections to the routing matrix
  - programmable lookup-tables
    - ⋆ 4 inputs, 1 output
    - ⋆ 6 inputs, 1 output
    - ⋆ 6 inputs, 2 outputs

FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.

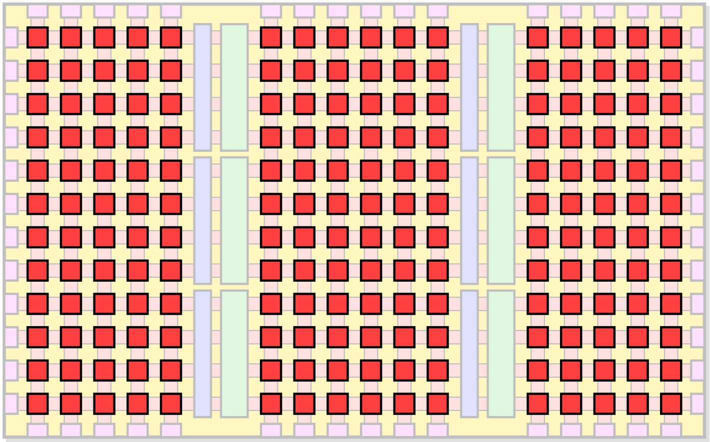


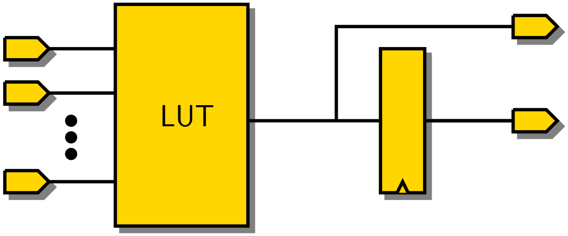


- Inside a logic cell:
  - connections to the routing matrix
  - programmable lookup-tables
    - ⋆ 4 inputs, 1 output
    - ⋆ 6 inputs, 1 output
    - ⋆ 6 inputs, 2 outputs
  - optional registers
    - ⋆ free pipelining

FPGA architecture

- FPGAs are composed of:
  - programmable logic cells
  - a configurable routing matrix
  - input/output cells
  - embedded memory blocks
  - small embedded multipliers
  - etc.





- Inside a logic cell:
  - connections to the routing matrix
  - programmable lookup-tables
    - 4 inputs, 1 output
    - 6 inputs, 1 output
    - 6 inputs, 2 outputs
  - optional registers
    - free pipelining
  - more logic for fast carry-propagation

Outline of the talk

- Small characteristic finite fields
- Multiplication algorithms and hardware implementation
- A finite field coprocessor
- Finite fields of composite extension degree

Representation of the elements

- Small characteristic: $p = 2$ or 3
- Polynomial basis:
  - $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree m
  - $\mathbb{F}_{p^m}$ represented by $\mathbb{F}_p[x]^{\leq(m-1)}$
  - Reduction modulo $f(x)$ possibly required

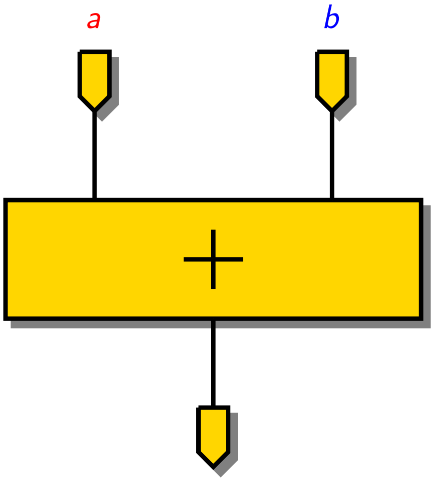
Representation of the elements

- Small characteristic: $p = 2$ or 3
- Polynomial basis:
  - $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree m
  - $\mathbb{F}_{p^m}$ represented by $\mathbb{F}_p[x]^{\leq(m-1)}$
  - Reduction modulo $f(x)$ possibly required
- Operations in the field
  - Addition
  - Frobenius automorphism: $(.)^p$
  - Multiplication
  - Inversion
    - Itoh & Tsujii algorithm (Fermat's little theorem)
    - or Extended Euclidean algorithm

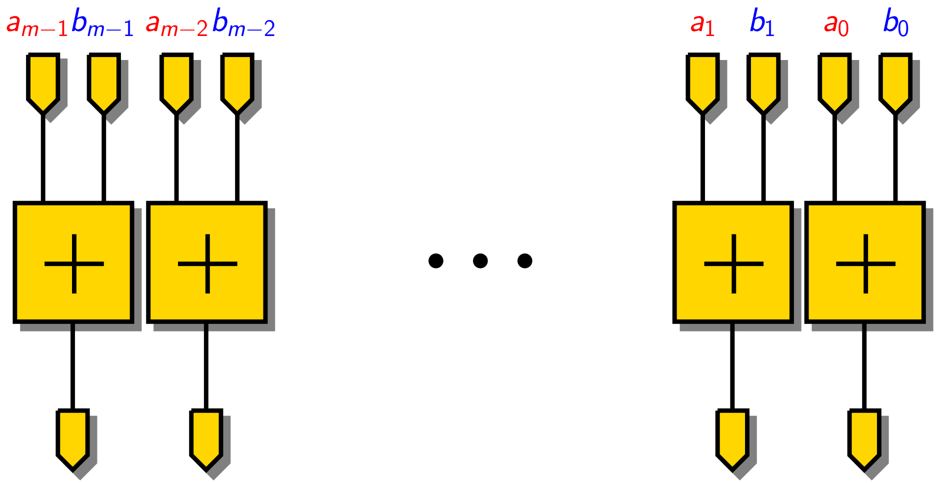
Representation of the elements

- Small characteristic: $p = 2$ or 3
- Polynomial basis:
  - $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f(x))$
  - $f(x)$ irreducible polynomial of degree m
  - $\mathbb{F}_{p^m}$ represented by $\mathbb{F}_p[x]^{\leq(m-1)}$
  - Reduction modulo $f(x)$ possibly required
- Operations in the field
  - Addition
  - Frobenius automorphism: $(.)^p$
  - Multiplication
  - Inversion
    - ⋆ Itoh & Tsujii algorithm (Fermat's little theorem)
    - ⋆ or Extended Euclidean algorithm

Hardware implementation of addition

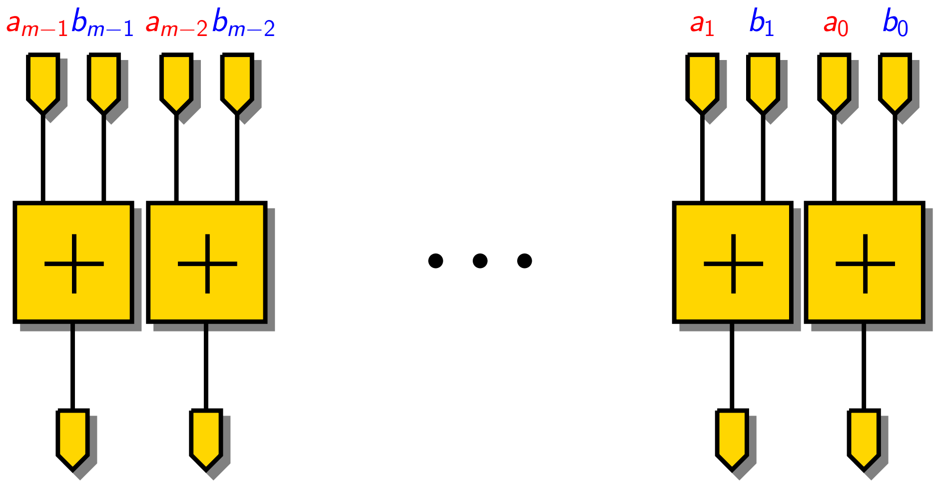


Hardware implementation of addition



- Add coefficient-wise

Hardware implementation of addition



- Add coefficient-wise
- Representation of coefficient
  - $\mathbb{F}_2$: two values $\rightarrow$ one wire, addition is the XOR boolean operator
  - $\mathbb{F}_3$: three values $\rightarrow$ borrow-save representation on two wires
    - $\star$ Opposite of a coefficient: swap the two wires

Computation of the Frobenius automorphism

$$a^p \equiv \left(a_{m-1}x^{m-1} + \cdots + a_1 x + a_0\right)^p \pmod{f(x)}$$

- Raising to the p-th power

Computation of the Frobenius automorphism

$$a^p \equiv a_{m-1}x^{p\cdot(m-1)} + \cdots + a_1x^p + a_0 \pmod{f(x)}$$

- Raising to the p-th power
- Linear operation
  - Since $\binom{p}{i} \equiv 0 \pmod{p}$ when $i \neq 0$, non-linear terms disappear
- Need reduction
  - reduce each $x^{p\cdot i}$
  - linear combination of the coefficients

Computation of the Frobenius automorphism

$$a^p \equiv (a_{\sigma_{m-1}(0)} + a_{\sigma_{m-1}(1)} + \cdots)x^{p-1} + \cdots \pmod{f(x)}$$

- Raising to the p-th power
- Linear operation
  - Since $\binom{p}{i} \equiv 0 \pmod{p}$ when $i \neq 0$, non-linear terms disappear
- Need reduction
  - reduce each $x^{p \cdot i}$
  - linear combination of the coefficients
- f with low Hamming weight
  - tri- or pentanomials
  - each coefficient of the results is the sum of few coefficients

Computation of the Frobenius automorphism

$$a^p \equiv (a_{\sigma_{m-1}(0)} + a_{\sigma_{m-1}(1)} + \cdots)x^{p-1} + \cdots \pmod{f(x)}$$

- Raising to the p-th power
- Linear operation
  - Since $\binom{p}{i} \equiv 0 \pmod{p}$ when $i \neq 0$, non-linear terms disappear
- Need reduction
  - reduce each $x^{p \cdot i}$
  - linear combination of the coefficients
- f with low Hamming weight
  - tri- or pentanomials
  - each coefficient of the results is the sum of few coefficients
- Hardware implementation
  - Selection of coefficient is free (just wiring!)
  - Same cost as for a few additions
  - Depending on LUTs' size, one LUT per coefficient might be enough

Itoh & Tsujii algorithm

- Fermat's little theorem

$$a^{-1} \equiv a^{p^m - 2} \pmod{f}$$

Itoh & Tsujii algorithm

- Fermat's little theorem

$$a^{-1} \equiv a^{p^m-2} \pmod{f}$$

- Computation of a^{p^m-2} only needs:
  - $(m-1)$ Frobenius automorphism applications
  - few multiplications
  - an inversion in $\mathbb{F}_p$

Itoh & Tsujii algorithm

- Fermat's little theorem
$$a^{-1} \equiv a^{p^m-2} \pmod{f}$$
- Computation of a^{p^m-2} only needs:
  - $(m-1)$ Frobenius automorphism applications
  - few multiplications
  - an inversion in $\mathbb{F}_p \rightarrow$ identity for $p = 2$ or 3

Itoh & Tsujii algorithm

- Fermat's little theorem

$$a^{-1} \equiv a^{p^m-2} \pmod{f}$$

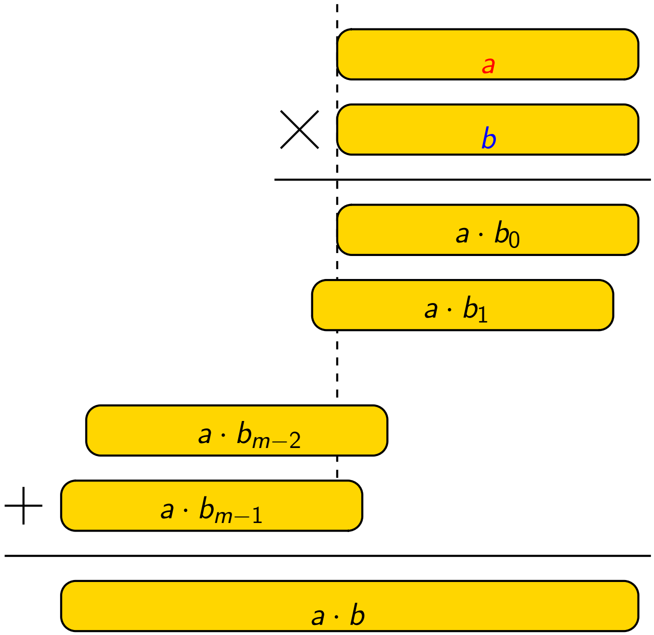
- Computation of a^{p^m-2} only needs:
  - $(m-1)$ Frobenius automorphism applications
  - few multiplications
  - an inversion in $\mathbb{F}_p \rightarrow$ identity for $p = 2$ or 3
- No need for supplementary hardware

Outline of the talk

- Small characteristic finite fields
- Multiplication algorithms and hardware implementation
- A finite field coprocessor
- Finite fields of composite extension degree

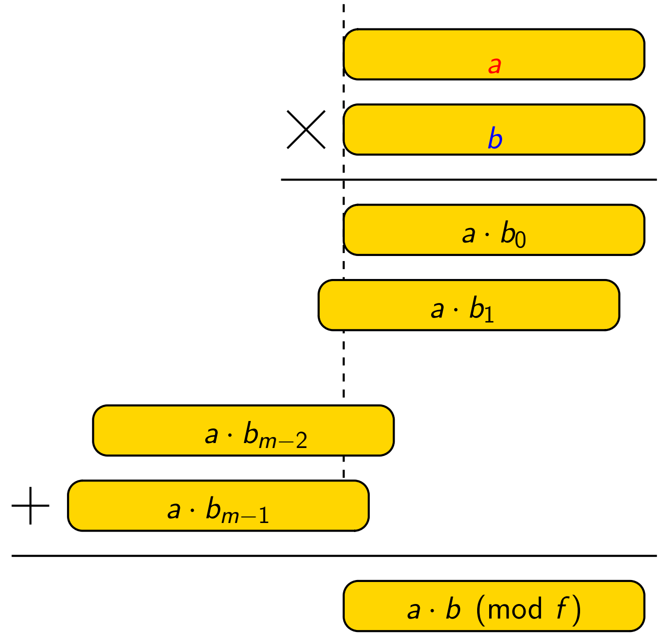
Naive algorithm

- Schoolbook algorithm
  - express each partial product
  - add them



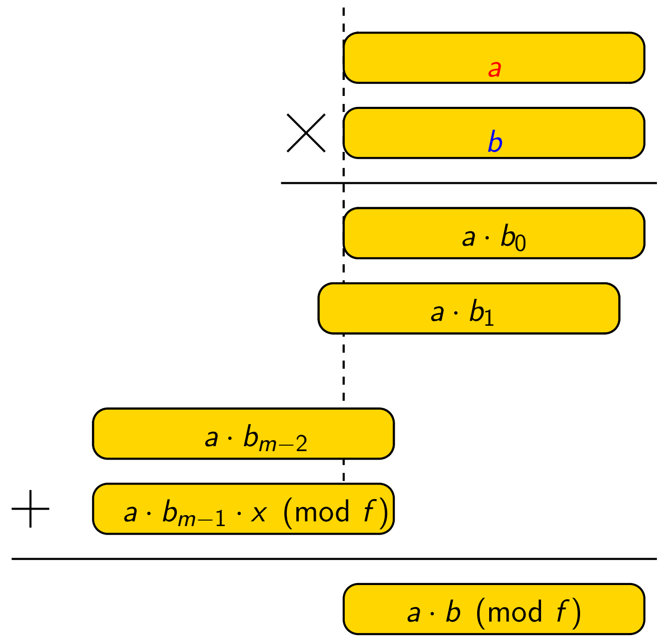
Naive algorithm

- Schoolbook algorithm
  - express each partial product
  - add them
- Reduction modulo f needed



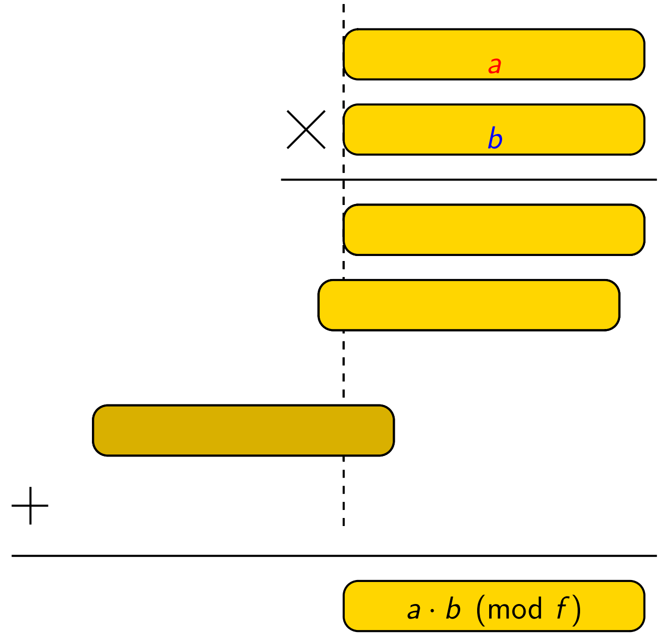
Naive algorithm

- Schoolbook algorithm
  - express each partial product
  - add them
- Reduction modulo f needed
- Reduce each partial product sequentially



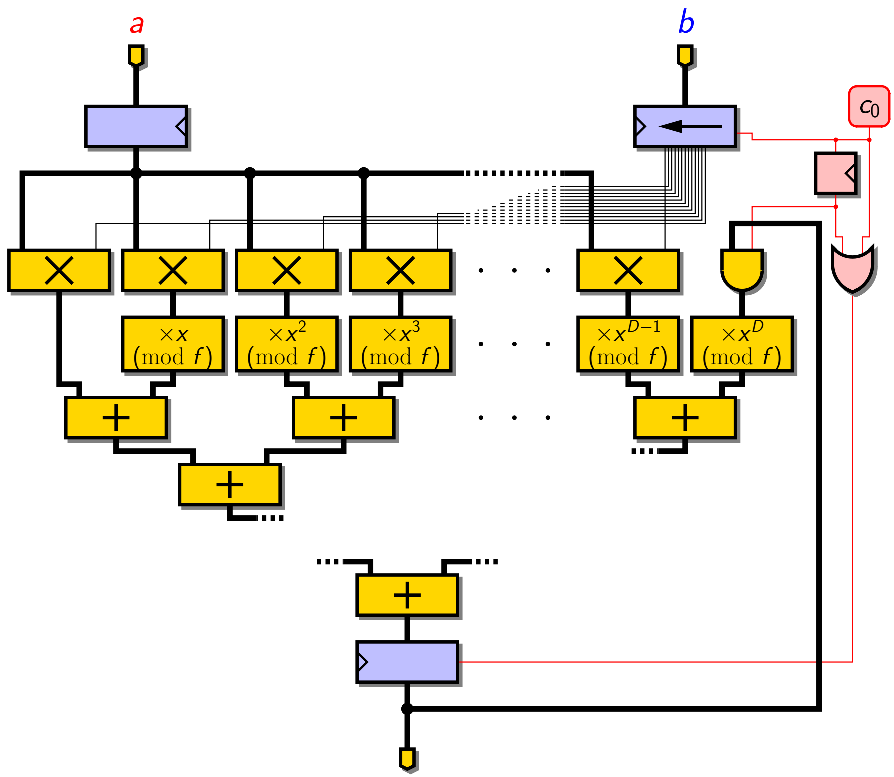
Naive algorithm

- Schoolbook algorithm
  - express each partial product
  - add them
- Reduction modulo f needed
- Reduce and accumulate each partial product sequentially



Parallel-serial multiplier

- Operand a is treated in parallel
- Operand b is treated D coefficients per cycle
- Need $\lceil m/D \rceil$ cycles to complete the product



Karatsuba algorithm

A $\qquad\qquad$ B

$A \cdot B$

Karatsuba algorithm

A_H A_L B_H B_L

$A_H B_H X^{2n} + (A_H B_L + A_L B_H) X^n + A_L B_L$

Karatsuba algorithm

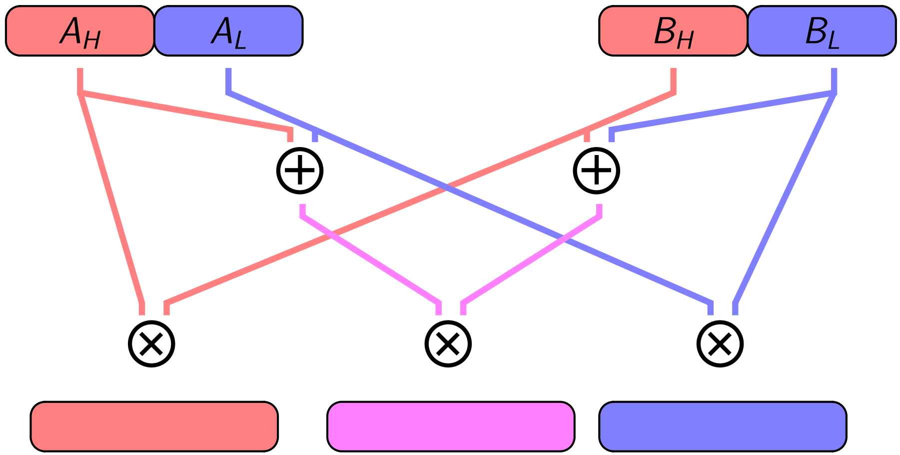
A_H A_L B_H B_L

$$A_H B_H X^{2n} + (A_H B_L + A_L B_H) X^n + A_L B_L$$

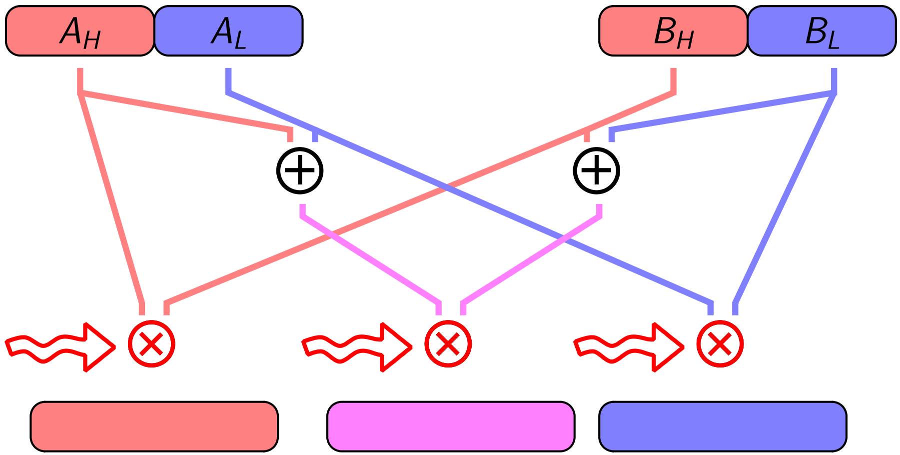
$$ab' + a'b = (a + a')(b + b') - ab - a'b'$$

$$A_H B_H X^{2n} + ((A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L) X^n + A_L B_L$$

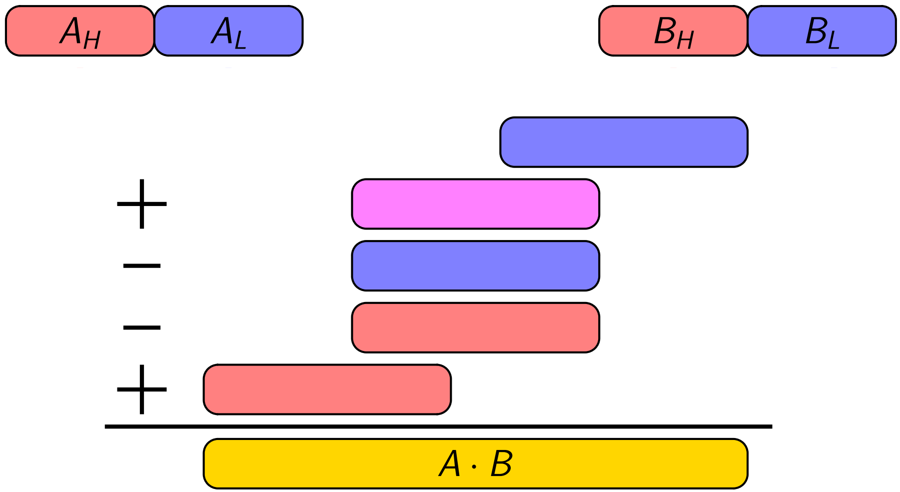
Karatsuba algorithm



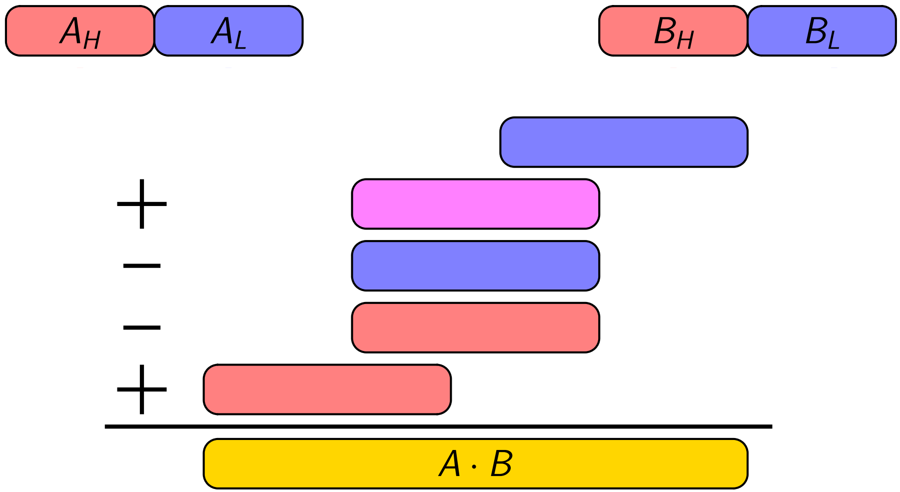
Karatsuba algorithm



Karatsuba algorithm

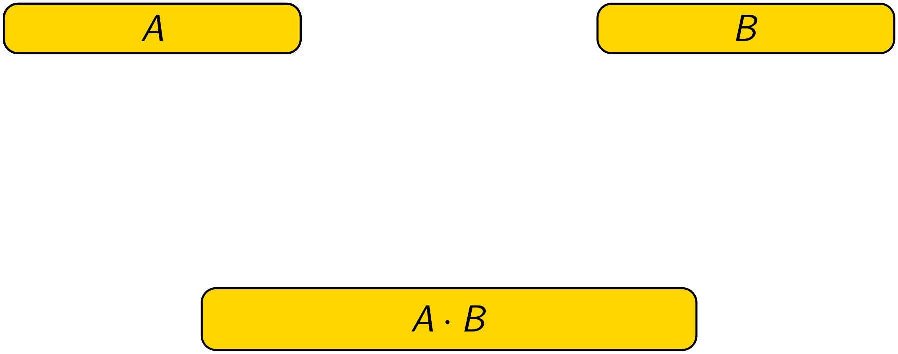


Karatsuba algorithm

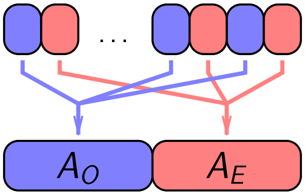


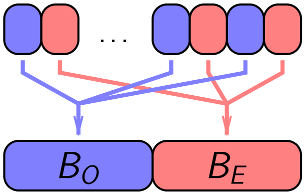
- 3-way Karatsuba
  - split operands in three parts
  - only 6 subproducts needed

Odd-even split for Karatsuba multiplication



Odd-even split for Karatsuba multiplication





$$(A_O B_O X^2 + A_E B_E) + X(A_O B_E + A_E B_O)$$

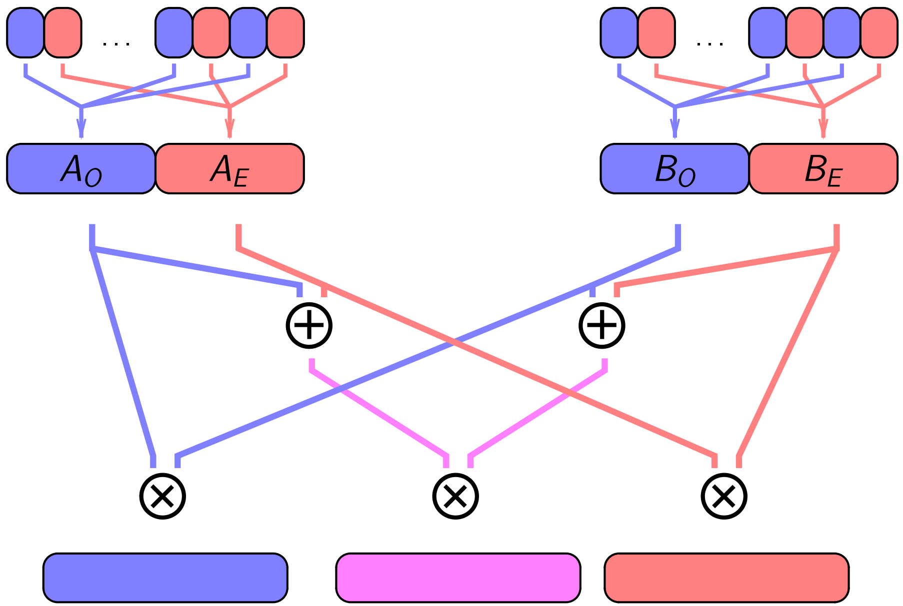
Odd-even split for Karatsuba multiplication

$$(A_O B_O X^2 + A_E B_E) + X(A_O B_E + A_E B_O)$$

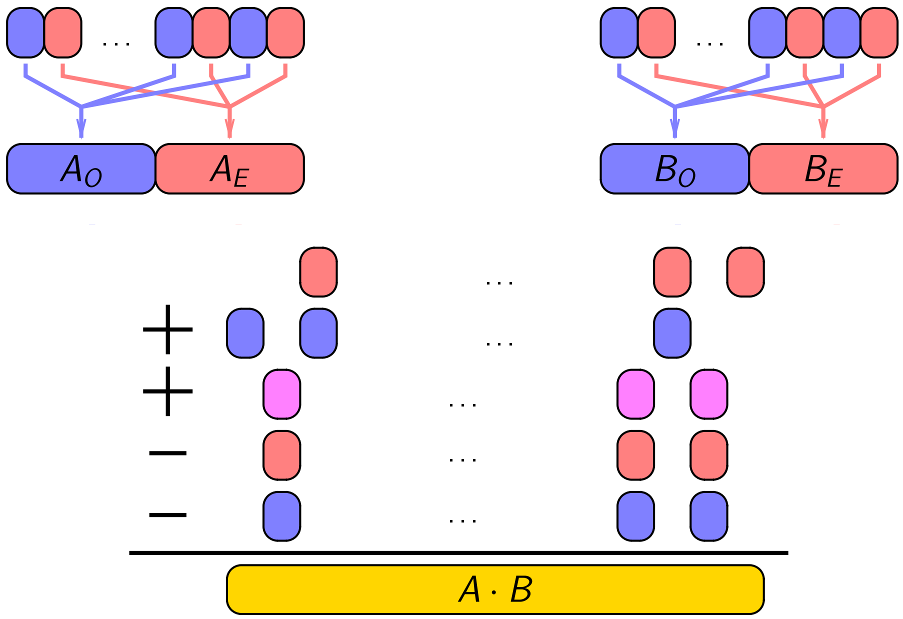
$$ab' + a'b = (a + a')(b + b') - ab - a'b'$$

$$(A_O B_O X^2 + A_E B_E) + X((A_O + A_E)(B_O + B_E) - A_O B_O - A_E B_E)$$

Odd-even split for Karatsuba multiplication

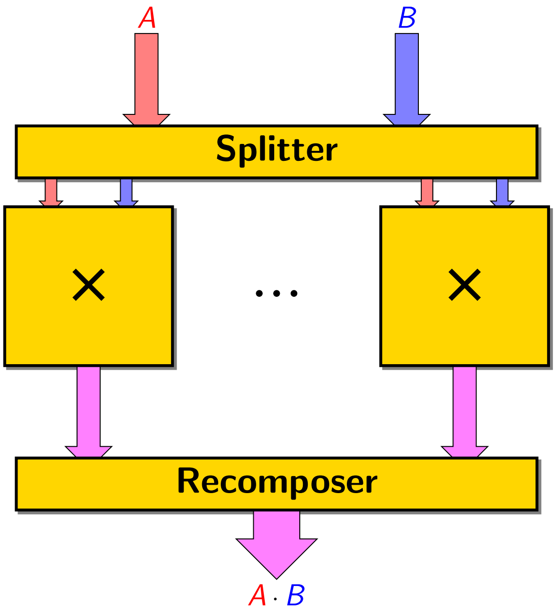


Odd-even split for Karatsuba multiplication



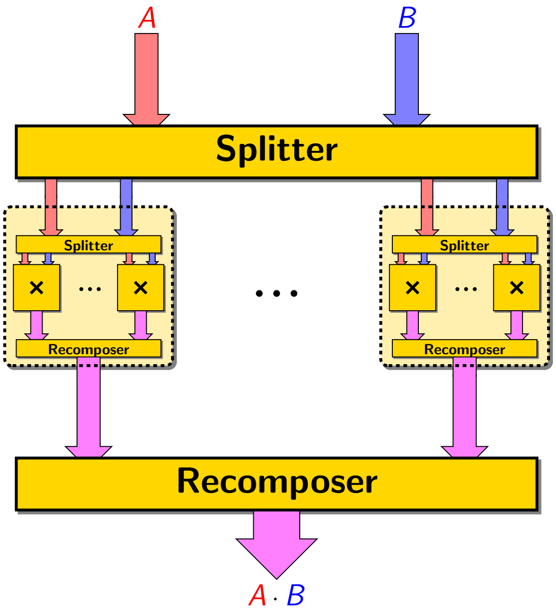
Fully parallel pipelined Karatsuba multiplier

- Karatsuba-like algorithm:
  - split the operands
  - compute the subproducts
  - recompose the result
- Fully parallel evaluation of the subproducts



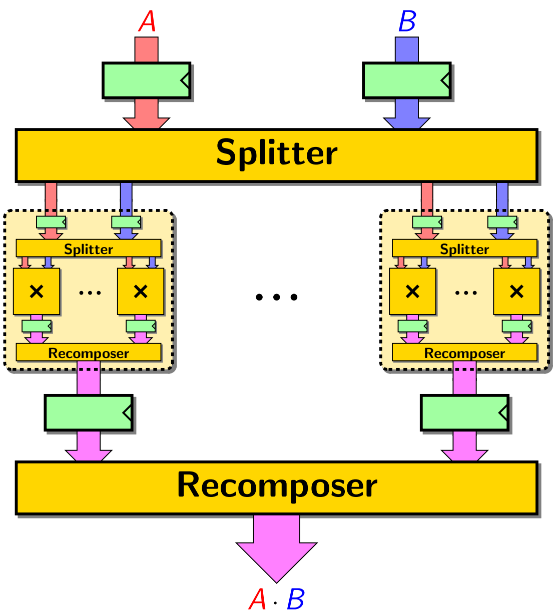
Fully parallel pipelined Karatsuba multiplier

- Karatsuba-like algorithm:
  - split the operands
  - compute the subproducts
  - recompose the result
- Fully parallel evaluation of the subproducts
- Recursive scheme
  - eventually use different multiplication algorithms
  - end with the quadratic paper-and-pencil algorithm



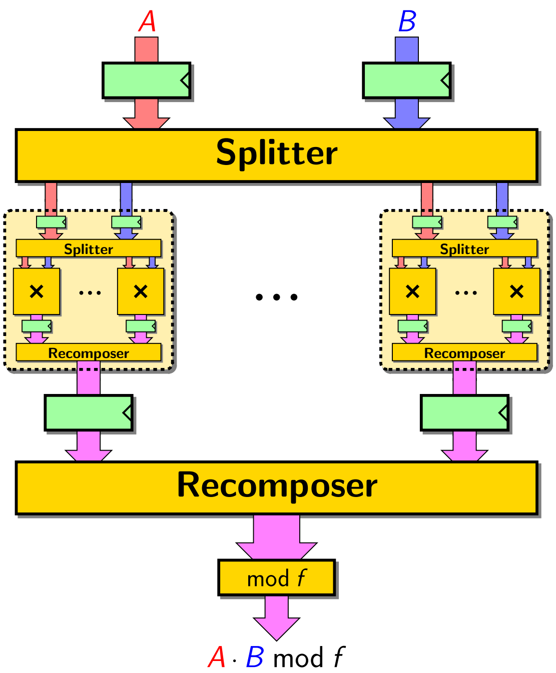
Fully parallel pipelined Karatsuba multiplier

- Karatsuba-like algorithm:
  - split the operands
  - compute the subproducts
  - recompose the result
- Fully parallel evaluation of the subproducts
- Recursive scheme
  - eventually use different multiplication algorithms
  - end with the quadratic paper-and-pencil algorithm
- Pipelined
  - with the help of optional registers
  - cut the critical path
  - increase the frequency



Fully parallel pipelined Karatsuba multiplier

- Karatsuba-like algorithm:
  - split the operands
  - compute the subproducts
  - recompose the result
- Fully parallel evaluation of the subproducts
- Recursive scheme
  - eventually use different multiplication algorithms
  - end with the quadratic paper-and-pencil algorithm
- Pipelined
  - with the help of optional registers
  - cut the critical path
  - increase the frequency
- Final reduction modulo f



An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \quad \rightarrow \quad \sim$ 380-bit field

An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \quad \rightarrow \quad \sim$ 380-bit field
- Recursion choice

Polynomial size	Used algorithm
239	3-way Karatsuba with odd-even trick
80	2-way Karatsuba with odd-even trick
40	2-way Karatsuba with odd-even trick
20	2-way Karatsuba with odd-even trick
10	2-way Karatsuba with odd-even trick
5	2-way Karatsuba with odd-even trick
3	quadratic multiplication

An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \quad \rightarrow \quad \sim$ 380-bit field
- Recursion choice

Polynomial size	Used algorithm
239	3-way Karatsuba with odd-even trick
80	2-way Karatsuba with odd-even trick
40	2-way Karatsuba with odd-even trick
20	2-way Karatsuba with odd-even trick
10	2-way Karatsuba with odd-even trick
5	2-way Karatsuba with odd-even trick
3	quadratic multiplication

- Choose to have 7 pipeline stages

An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \quad \rightarrow \quad \sim$ 380-bit field
- Recursion choice

Polynomial size	Used algorithm
239	3-way Karatsuba with odd-even trick
80	2-way Karatsuba with odd-even trick
40	2-way Karatsuba with odd-even trick
20	2-way Karatsuba with odd-even trick
10	2-way Karatsuba with odd-even trick
5	2-way Karatsuba with odd-even trick
3	quadratic multiplication

- Choose to have 7 pipeline stages
- Post-place-and-route estimation for Xilinx Virtex-II Pro
  - $\sim$ 50000 slices (2 LUTs 4 $\rightarrow$ 1 per slice)
  - 200 MHz
  - computes $200 \cdot 10^6$ products per second

An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \quad \rightarrow \quad \sim$ 380-bit field
- Recursion choice

Polynomial size	Used algorithm
239	3-way Karatsuba with odd-even trick
80	2-way Karatsuba with odd-even trick
40	2-way Karatsuba with odd-even trick
20	2-way Karatsuba with odd-even trick
10	2-way Karatsuba with odd-even trick
5	2-way Karatsuba with odd-even trick
3	quadratic multiplication

- Choose to have 7 pipeline stages
- Post-place-and-route estimation for Xilinx Virtex-II Pro
  - $\sim$ 50000 slices (2 LUTs 4 $\rightarrow$ 1 per slice)
  - 200 MHz
  - computes $200 \cdot 10^6$ products per second
- Comparison with parallel-serial multiplier with $D = 16$
  - $\sim$ 8700 slices
  - 180 MHz
  - computes $12 \cdot 10^6$ products per second

An example of multipliers over $\mathbb{F}_{3^{239}}$

- $\mathbb{F}_{3^{239}} = \mathbb{F}_3[X]/(X^{239} - X^5 + 1) \quad \rightarrow \quad \sim$ 380-bit field
- Recursion choice

Polynomial size	Used algorithm
239	3-way Karatsuba with odd-even trick
80	2-way Karatsuba with odd-even trick
40	2-way Karatsuba with odd-even trick
20	2-way Karatsuba with odd-even trick
10	2-way Karatsuba with odd-even trick
5	2-way Karatsuba with odd-even trick
3	quadratic multiplication

- Choose to have 7 pipeline stages
- Post-place-and-route estimation for Xilinx Virtex-II Pro
  - $\sim$ 50000 slices (2 LUTs 4 $\rightarrow$ 1 per slice)
  - 200 MHz
  - computes $200 \cdot 10^6$ products per second
  - 4000 products per second and per slice
- Comparison with parallel-serial multiplier with $D = 16$
  - $\sim$ 8700 slices
  - 180 MHz
  - computes $12 \cdot 10^6$ products per second
  - 1400 products per second and per slice

Outline of the talk

- Small characteristic finite fields
- Multiplication algorithms and hardware implementation
- A finite field coprocessor
- Finite fields of composite extension degree

Designing a finite field coprocessor

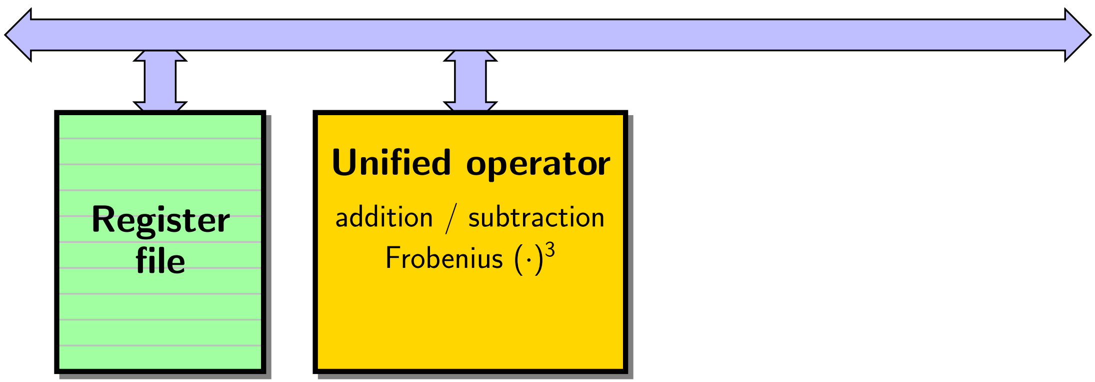
- Determine the specific needs of operations of your computation

Designing a finite field coprocessor

- Determine the specific needs of operations of your computation
- Example: final exponentiation in pairing computation

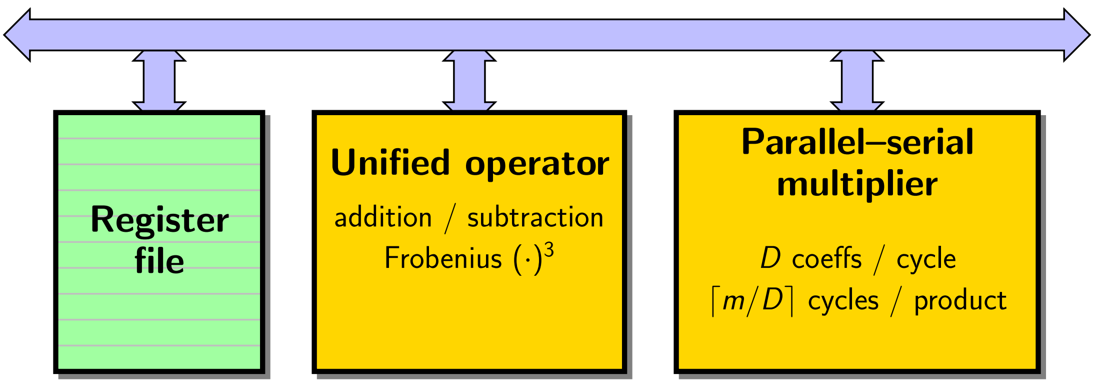
Designing a finite field coprocessor

- Determine the specific needs of operations of your computation
- Example: final exponentiation in pairing computation
  - Low silicon footprint design



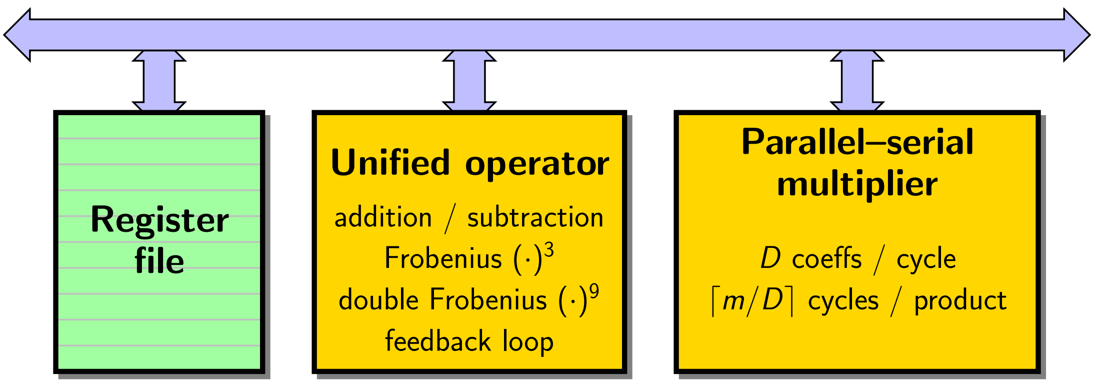
Designing a finite field coprocessor

- Determine the specific needs of operations of your computation
- Example: final exponentiation in pairing computation
  - Low silicon footprint design
  - Many multiplications



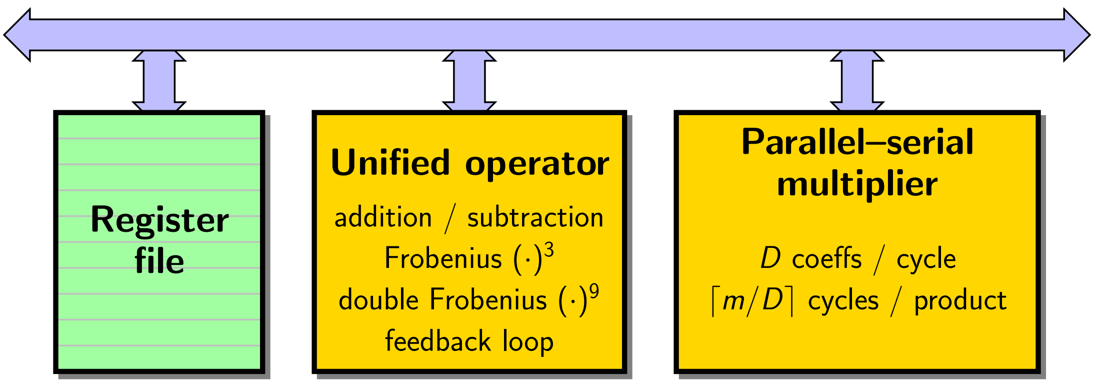
Designing a finite field coprocessor

- Determine the specific needs of operations of your computation
- Example: final exponentiation in pairing computation
  - Low silicon footprint design
  - Many multiplications
  - Long chains of Frobenius automorphism application

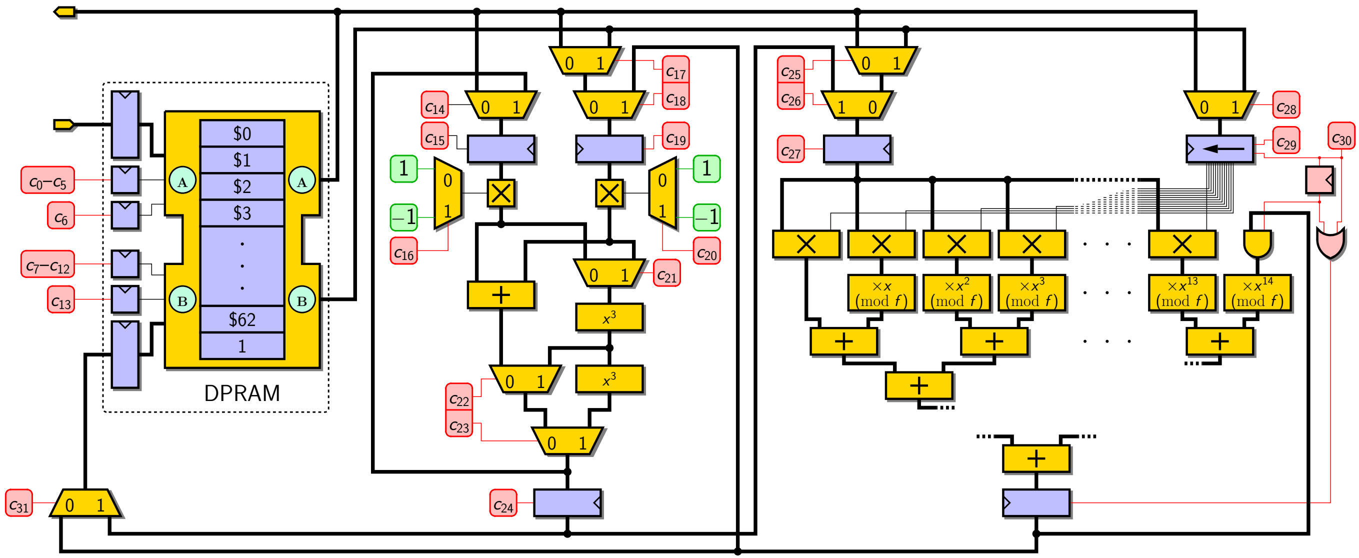


Designing a finite field coprocessor

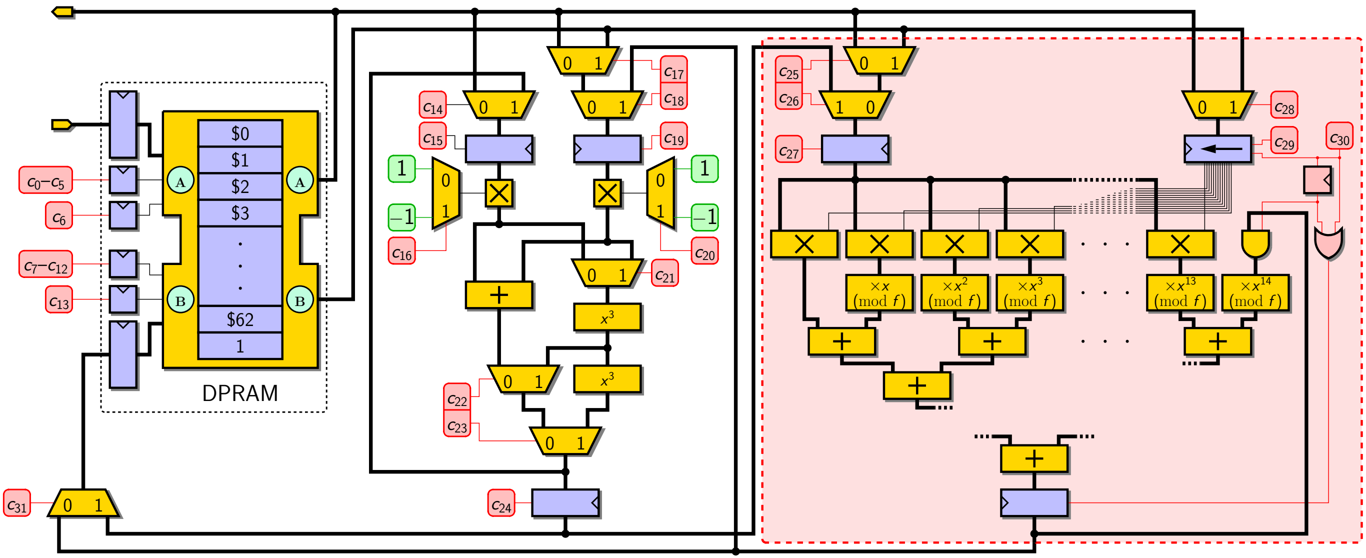
- Determine the specific needs of operations of your computation
- Example: final exponentiation in pairing computation
  - Low silicon footprint design
  - Many multiplications
  - Long chains of Frobenius automorphism application
  - Only one inversion



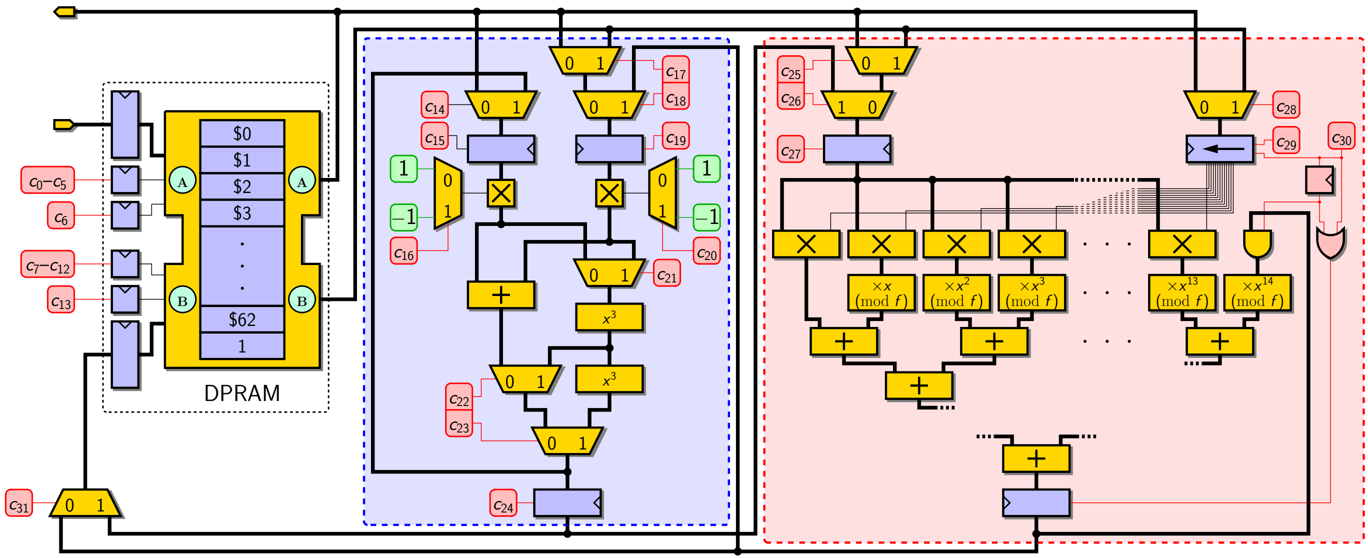
Detailed architecture of the coprocessor (char. 3)



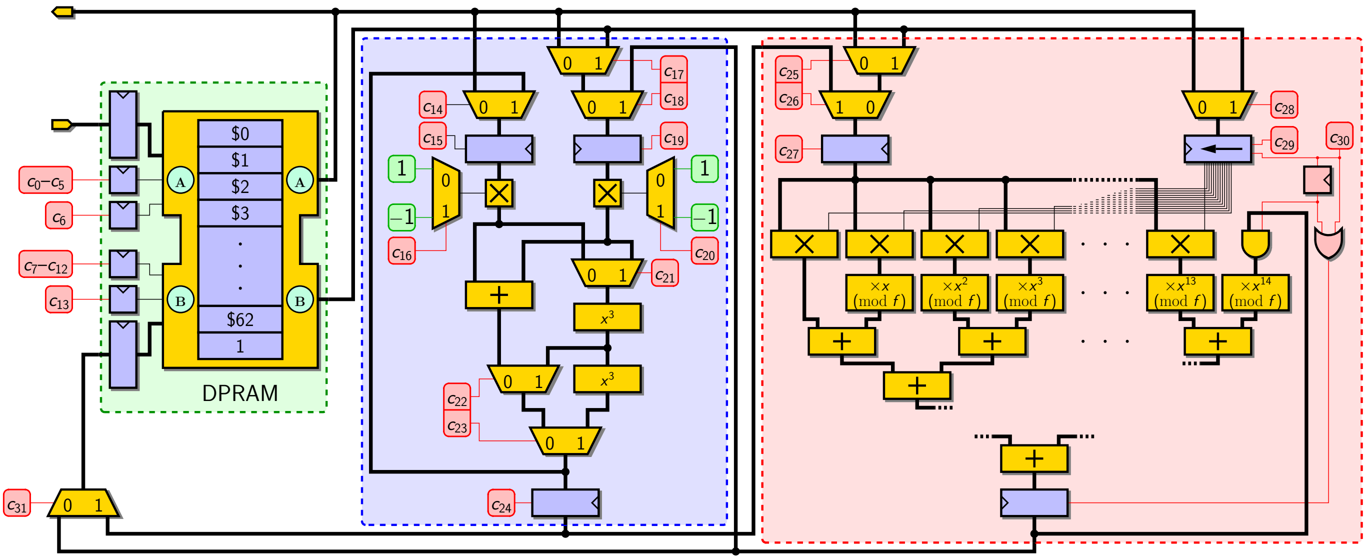
Detailed architecture of the coprocessor (char. 3)



Detailed architecture of the coprocessor (char. 3)



Detailed architecture of the coprocessor (char. 3)



Outline of the talk

- Small characteristic finite fields
- Multiplication algorithms and hardware implementation
- A finite field coprocessor
- Finite fields of composite extension degree

Composite extension degree

- Needed field might have a composite extension degree
- Tower field construction:
  - $\mathbb{F}_{p^{m \cdot n}} = \mathbb{F}_{p^m}[y]/(g(y))$
  - g irreducible polynomial of degree n

Composite extension degree

- Needed field might have a composite extension degree
- Tower field construction:
  - $\mathbb{F}_{p^{m \cdot n}} = \mathbb{F}_{p^m}[y]/(g(y))$
  - g irreducible polynomial of degree n
- Reducing the datapath
  - design a coprocessor for $\mathbb{F}_{p^m}$
  - program it to implement arithmetic of $\mathbb{F}_{p^{m \cdot n}}$
  - reduce area of the design
- Operations
  - same algorithms
  - coefficients are now in $\mathbb{F}_{p^m}$

Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step
- Apply reduction modulo g
  - some subproducts may not be needed after reconstruction step

Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step
- Apply reduction modulo g
  - some subproducts may not be needed after reconstruction step
- Toom–Cook algorithms
  - evaluate and interpolate at some points
  - $\mathbb{F}_p$ does not provide enough interpolation points
  - hardly usable in this case

Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step
- Apply reduction modulo g
  - some subproducts may not be needed after reconstruction step
- Toom–Cook algorithms
  - evaluate and interpolate at some points
  - $\mathbb{F}_p$ does not provide enough interpolation points
  - hardly usable in this case
- CRT-based algorithms
  - evaluate the product modulo some irreducible polynomials
  - reconstruct the result thanks to CRT

Some other multiplication algorithms

- Subproducts between field elements, not polynomials
  - no overlapping at reconstruction step
- Apply reduction modulo g
  - some subproducts may not be needed after reconstruction step
- Toom–Cook algorithms
  - evaluate and interpolate at some points
  - $\mathbb{F}_p$ does not provide enough interpolation points
  - hardly usable in this case
- CRT-based algorithms
  - evaluate the product modulo some irreducible polynomials
  - reconstruct the result thanks to CRT
- Montgomery's Karatsuba-like formulae
  - *ad hoc* formulae for degree 4, 5 and 6 polynomials
- Algorithmic search for optimal formulae
  - Work in progress with J. Detrey, R. Barbulescu and P. Zimmermann

Choosing multiplication algorithm

- Evaluate the cost of the different algorithms
  - choice depend on the hardware implementation of $\mathbb{F}_{p^m}$
  - additions not always negligible

Choosing multiplication algorithm

- Evaluate the cost of the different algorithms
  - choice depend on the hardware implementation of $\mathbb{F}_{p^m}$
  - additions not always negligible

Multiplication in $\mathbb{F}_{3^{m \cdot 5}}$

Algorithm	$\times$	$+$	Ratio
Schoolbook	25	24	0.96
One-level Karatsuba (Montgomery's trick)	21	29	1.38
Recursive Karatsuba	15	39	2.60
Recursive Karatsuba (Montgomery's trick)	14	43	3.07
Montgomery's Karatsuba-like	13	54	4.153
CRT-based	12	53	4.42

Choosing multiplication algorithm

- Evaluate the cost of the different algorithms
  - choice depend on the hardware implementation of $\mathbb{F}_{p^m}$
  - additions not always negligible

Multiplication in $\mathbb{F}_{3^{m \cdot 5}}$

Algorithm	$\times$	$+$	Ratio
Schoolbook	25	24	0.96
One-level Karatsuba (Montgomery's trick)	21	29	1.38
Recursive Karatsuba	15	39	2.60
Recursive Karatsuba (Montgomery's trick)	14	43	3.07
Montgomery's Karatsuba-like	13	54	4.153
CRT-based	12	53	4.42

Multiplication in $\mathbb{F}_{2^{m \cdot 7}}$

Algorithm	$\times$	$+$	Ratio
Schoolbook	49	48	0.98
One-level Karatsuba (Montgomery's trick)	40	52	1.30
Recursive Karatsuba	25	51	2.04
Recursive Karatsuba (Montgomery's trick)	23	76	3.30
Montgomery's Karatsuba-like	22	84	3.818
CRT-based	22	88	4.05

Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic

Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic
- Many criterion for optimization
  - area: low cost devices

Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic
- Many criterion for optimization
  - area: low cost devices
  - speed: if security should not introduce latency

Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic
- Many criterion for optimization
  - area: low cost devices
  - speed: if security should not introduce latency
  - area-speed tradeoff: high-throughput application

Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic
- Many criterion for optimization
  - area: low cost devices
  - speed: if security should not introduce latency
  - area-speed tradeoff: high-throughput application
- Multiplication is the critical operation
  - many implementation strategies

Conclusion

- Accelerators for curve-based cryptography rely on finite field arithmetic
- Many criterion for optimization
  - area: low cost devices
  - speed: if security should not introduce latency
  - area-speed tradeoff: high-throughput application
- Multiplication is the critical operation
  - many implementation strategies
- Need for algorithms/hardware codesign

Thank you for you attention!

Questions?