An Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method

Jens Franke, Thorsten Kleinjung - University of Bonn
Christof Paar, Jan Pelzl - University of Bochum
Christine Priplata, Colin Stahlke - EDIZONE GmbH, Bonn
Martin Šimka – University of Košice

Outline

1. Motivation and Introduction
2. The Elliptic Curve Method
3. A Hardware Architecture for ECM
4. Results
5. Conclusions

1. Introduction

Why factor numbers?

- Security of RSA relies on difficulty to factor large composites

  $n = p \cdot q$, known n, what is p and q?

  (in practice: $n \sim 1024$ bit)

- Holy grail in cryptanalysis:

  „Find efficient method for factoring (large) integers.“

1. Introduction

Running time of certain factorization methods:

Method	Running time	
Lehman's algorithm	$O(N^{1/3})$	Running time depending on N
Multiple Polynomial Quadratic Sieve (MPQS)	$O(e^{((1+o(1))\ln N \ln\ln N)^{1/2}})$	
Continued Fraction (CF)	$O(e^{(c\ln N \ln\ln N)^{1/2}})$	
Number Field Sieve	$O(e^{c(\ln N)^{1/3}(\ln\ln N)^{2/3}})$	
Trial Division	$O(f\cdot(\log N)^2)$	Running time depending on f
Pollard's rho	$O(f^{1/2}(\log N)^2)$	
Elliptic Curve Method (ECM)	$O(e^{((2+o(1))\ln f \ln\ln f)^{1/2}}(\log N)^2)$	

(N: composite, f: nontrivial factor)

1. Introduction

Different algorithms for different purposes, e.g.,

- Best known method for factoring large integers: GNFS
  (WR in factoring random RSA modulus: 576 bit)

- Methods suited for factoring numbers of 100-200 bit, e.g.,
  - MPQS
  - ECM (small factors)
  - Trial division (very, very small factors)

1. Introduction

Observation for, e.g., GNFS:

- Smoothness tests of „medium sized“ integers required

Objective:

- Design special purpose hardware for smoothness tests
- Parameters (SHARK):
  - Factor numbers up to 200 bit with factors up to 40 bit
  - Target low area-time (AT) complexity
  - Technical feasability preferable

$\Rightarrow$ Elliptic Curve Method (ECM)

1. Introduction

Why ECM?

- Factor integers with relatively small factors (up to 40 bit)
- Almost ideal for hardware implementation:
  - Allows for low I/O
  - Requires little memory
  - Easy to parallelize
  - Closely related to Elliptic Curve Cryptography (ECC)

2. The Elliptic Curve Method

2. The Elliptic Curve Method

- Algorithm proposed by [H.W. Lenstra 1985]
- Principle based on Pollard's $(p\text{-}1)$-method:
  - given an elliptic curve E defined over $\mathbb{Z}/N\mathbb{Z}$ and a point $\mathrm{P} \in \mathrm{E}(\mathbb{Z}/N\mathbb{Z})$
  - compute point multiple $k \cdot \mathrm{P}$ and „hope" that

$$k \cdot \mathrm{P} = \mathcal{O} \in \mathrm{E}(\mathbb{Z}/p\mathbb{Z}) \quad \text{and} \quad k \cdot \mathrm{P} \neq \mathcal{O} \in \mathrm{E}(\mathbb{Z}/N\mathbb{Z})$$

(e.g., with $\mathrm{E}: y^2z = x^3 + axz^2 + bz^3$ and $\mathrm{Q} = k \cdot \mathrm{P} = (x_\mathrm{Q}, y_\mathrm{Q}, z_Q)$,
$z_\mathrm{Q} = 0 \bmod p$ and $z_\mathrm{Q} \neq 0 \bmod N$,
hence, $\gcd(z_\mathrm{Q}, N) = p$)

2. The Elliptic Curve Method

- Advantage over Pollard's (p-1)-method:
  - If no factor found, simply choose another curve
  - Easy to parallelize

2. The Elliptic Curve Method

Elliptic curves and point arithmetic:

- Use curves in Montgomery form:

$$By^2z = x^3 + Ax^2z + xz^2$$

- Point addition of P+Q involves P, Q and P-Q:

$$x_{P+Q} = z_{P-Q}[(x_P - z_P)(x_Q + z_Q) + (x_P + z_P)(x_Q - z_Q)]^2$$
$$z_{P+Q} = x_{P-Q}[(x_P - z_P)(x_Q + z_Q) - (x_P + z_P)(x_Q - z_Q)]^2$$

- Point duplication of P involves P, curve parameter A:

$$4x_p z_p = (x_P + z_P)^2 - (x_P - z_P)^2$$
$$x_{2P} = (x_P + z_P)^2 (x_P - z_P)^2$$
$$y_{2P} = 4x_P z_P[(x_P - z_P)^2 + 4x_P z_P(A+2)/4]$$

2. The Elliptic Curve Method

ECM Phase 1:

Compute $Q=k \cdot P$ with $k = \prod_{p \in \mathbb{P},\, p \le B_1} p^{e_p}$ and $e_p = [\log B_1 / \log p]$

– Use Montgomery ladder for point multiplication:
Given the triple $(P, nP, (n+1)P)$, we either compute

$$(P,\ 2nP\ ,(2n+1)P) \quad \text{or}$$
$$(P,(2n+1)P,2(n+1)P)$$

by one addition and duplication in Montgomery form.

– In the case of $z_P=1$, $10[\log_2 k]$ multiplications are required

2. The Elliptic Curve Method

ECM Phase 2:

Compute $p_i \cdot \mathrm{Q} \; \forall \; B_1 \leq p_i \leq B_2$ and check if $\gcd(z_{p\mathrm{Q}}, N) > 1$

Efficient method for phase 2:

- Precompute small table T of multiples $k \cdot \mathrm{Q}$
- Express all primes as $p_i = mD \pm k$ with $\mathrm{k} \in \mathrm{T}$
  (could compute $p_i \cdot \mathrm{Q} = mD \cdot \mathrm{Q} \pm k \cdot \mathrm{Q}$)
- Fact: $\gcd(z_{p\mathrm{Q}}, N) > 1$ iff $\gcd(x_{mD\mathrm{Q}} z_{k\mathrm{Q}} - x_{k\mathrm{Q}} z_{mD\mathrm{Q}}, N) > 1$
  (hence, only sequence of $mD\mathrm{Q}$ has to be computed)
- Compute product $\prod(x_{mD\mathrm{Q}} z_{k\mathrm{Q}} - x_{k\mathrm{Q}} z_{mD\mathrm{Q}})$ for all primes and perform a final gcd with N

2. The Elliptic Curve Method

Choice of „good“ parameters:

- Set of parameters deduced by software experiments:
  $B_1 = 960$, $B_2 = 57\,000$, $D = 30$
- Probability of success ~ 80% with 20 distinct curves per N
- Time complexity of ECM phases:
  - Phase 1: ~ 13740 modular multiplications/ squarings
  - Phase 2: ~ 24926 modular multiplications/ squarings
- Memory complexity: 21 registers of size of N per ECM unit

3. A Hardware Architecture for ECM

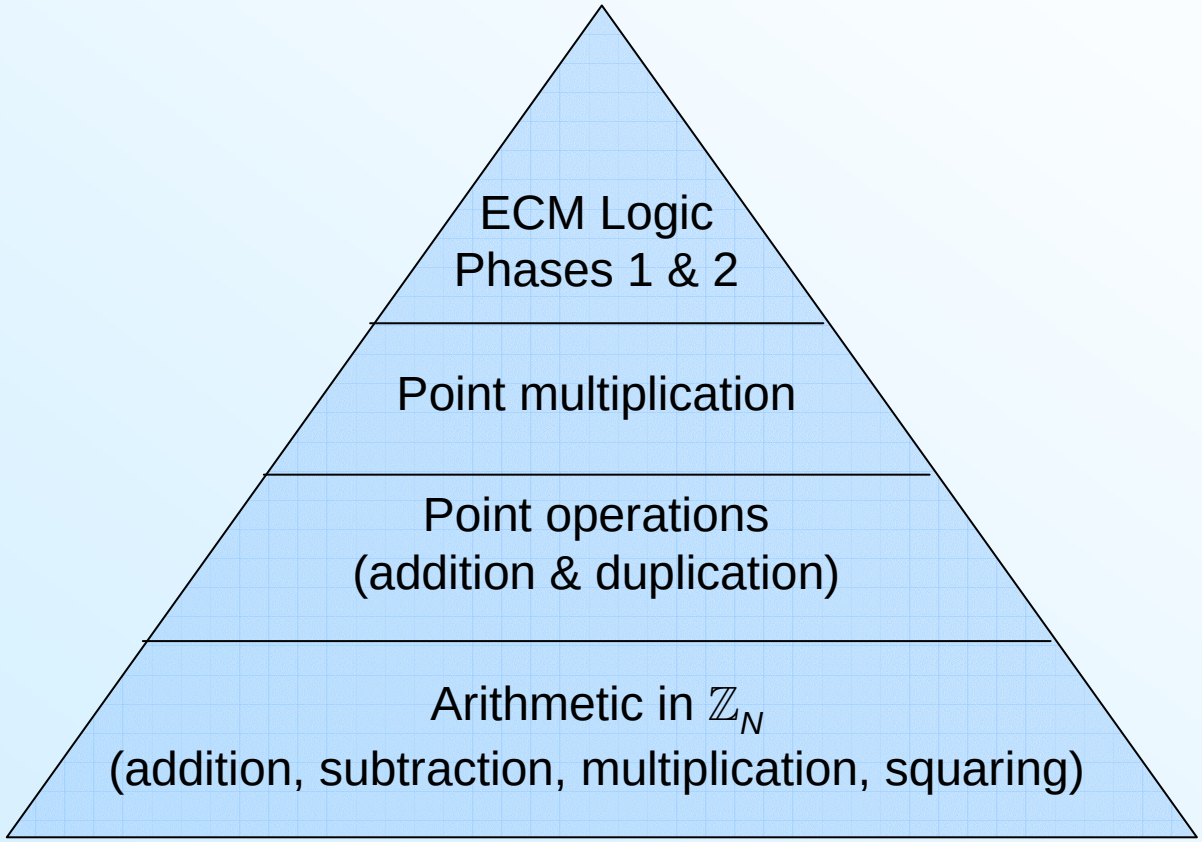
3. A Hardware Architecture for ECM

Design objectives:

- Low area-time (AT) product
- Low communication overhead
- Parallelizable architecture
- Easy to adopt to other bitlengths

3. A Hardware Architecture for ECM

Top down design of ECM:



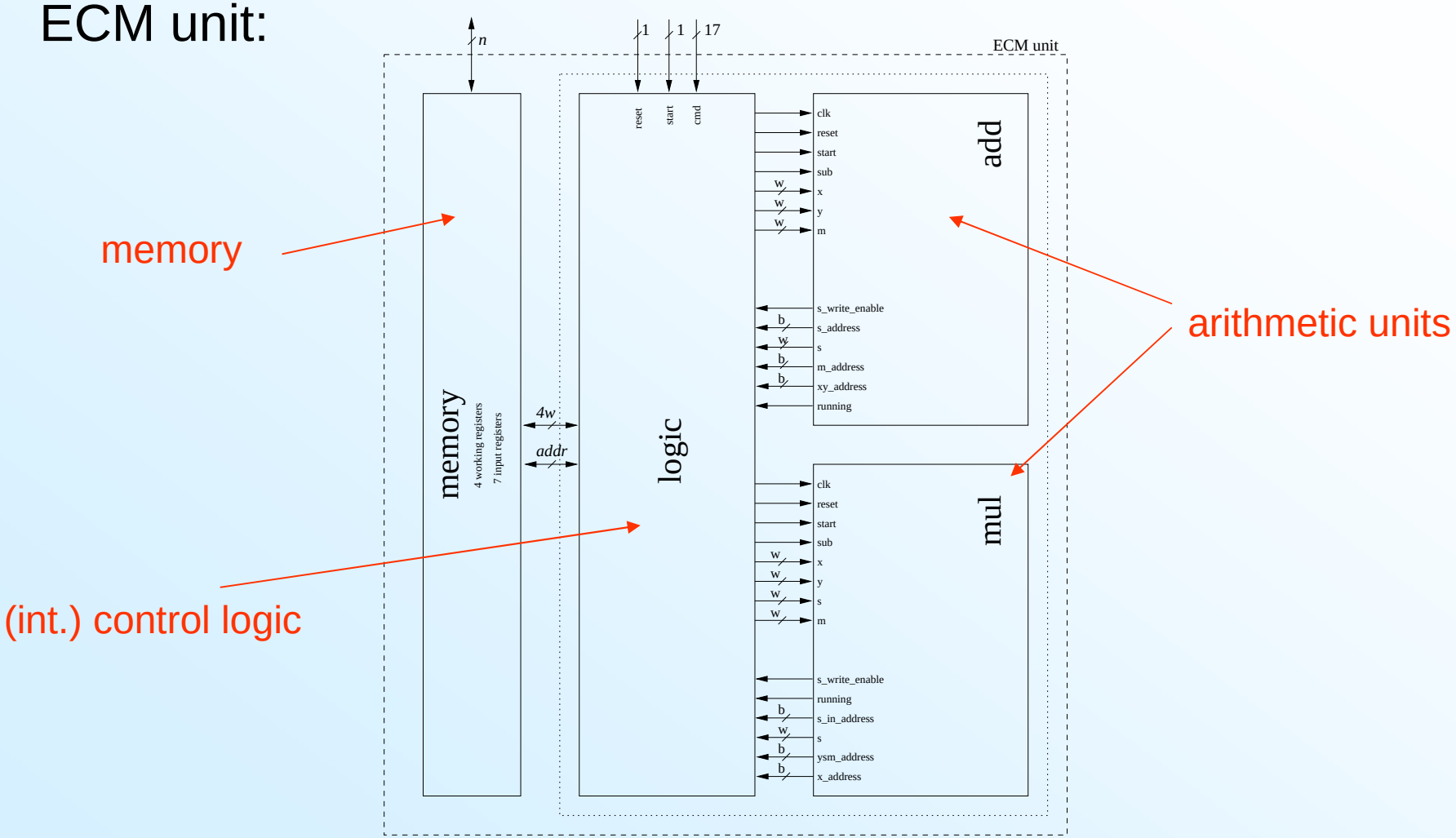
3. A Hardware Architecture for ECM

Specification of a single ECM unit:

- One ECM unit handles one curve (initialized in the beginning)
- Control of both phases by (external) central logic
  (control sequence is identical for every unit)
- Each unit has ist own memory
- Units implement arithmetic in $\mathbb{Z}_N$
  - addition + subtraction: Standard carry ripple adder (wordwise)
  - multiplication and squaring: efficient multiplier with pipelining-structure [Koç et al.]

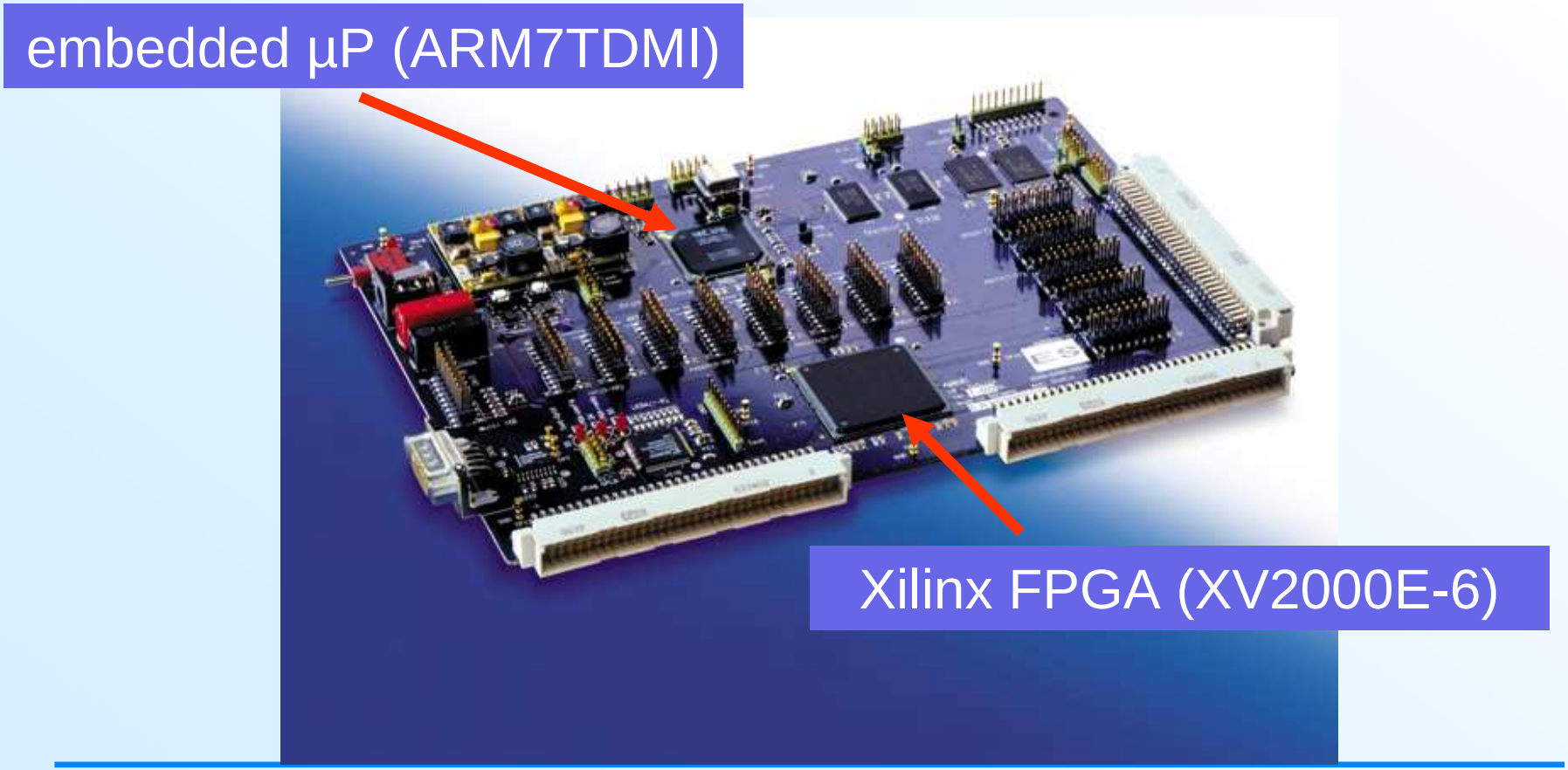
3. A Hardware Architecture for ECM

ECM unit:



4. Results

Hardware platform: System-on-Chip (SoC), 25 MHz



4. Results

Area-time specification of running implementation (no estimates!):

- Maximum frequency: 38.132 MHz
- Device utilization (FPGA):
  - 1122 CLBs (5.8%)
  - 44 Block RAMs (27%)
- Running times (25MHz):
  - Point addition (phase 1) 333µs
  - Point addition (phase 2) 397µs
  - Point duplication 330µs
  - Phase1 912ms
  - Phase2 1879ms

5. Conclusion & Outlook

- Area-time efficient hardware architecture for ECM
- Running FPGA implementation as proof of concept
- Realistic (and realizable) circuit for supporting, e.g., the GNFS (ASIC estimates see paper)

5. Conclusions & Outlook

Future work:

- Optimize control logic
- Improvements of basic Montgomery ladder
- Analyze parallel ECM in hardware
- Use CPU core in VHDL instead of embedded ARM µP
- ASIC simulation of ECM

5. Conclusions & Outlook

Thanks!