# Design of High Speed Long Calculation Units for Public-Key Cryptography


*Jean-Pierre Seifert*

Infineon Technologies Corporation

Security & Chip Card ICs

Technical Innovation

D-81669 Munich

Germany

## *Motivation - Why to think about new long integer arithmetic units for smart card ICs?*

- ➔ First generation long integer units were designed to support RSA
- ➔ Longer RSA parameters desired
- ➔ RSA is loosing its dominant role
- ➔ Increasing interest in public key schemes based on elliptic curves
- ➔ New attacks unknown when the current units were designed

requirements

- ➔ Support of RSA up to 2048 Bits
- ➔ Support of elliptic curves over finite fields up to 256 Bits
- ➔ Immunity against recent attacks
- ➔ Flexibility and scalability of the design

## *What are the arithmetic requirements due to RSA?*

**RSA**

Crucial cryptographic operation:

$a, b, N \to a^b \bmod N$

***Reduction to more elementary operations***

modular squarings , modular multiplications

$c \to c^2 \bmod N;$

$c, a \to c \times a \bmod N$

**Registers:**

small number of very long registers (4 x 2048 Bits)

# *What are the arithmetic requirements due to elliptic curves?*

**Elliptic curve** ***E*****: $y^2 = x^3 + ax + b$ over GF(p) or E: $y^2 = x^3 + ax + b$ over GF($2^k$)**

Crucial cryptographic operation:
k, P $\rightarrow$ k $\times$ P, where k is a secret integer, P = ($x_P$, $y_P$) is a point on ***E***.

***Reduction to elementary operations on elliptic curve***

point doublings / additions of different points
$Q \rightarrow 2 \times Q$ $Q,P \rightarrow Q + P$

***Reduction to elementary modular arithmetic***

a lot of modular additions, modular subtractions, modular multiplications, and
at least one modular inversion

**Registers:**
large number of rather short registers (8 x 256 Bits)
random access to operands necessary

## *What is "modular arithmetic" and how to do it?*

| | |
|---|---|
| ***Given:*** | a positive integer N, the modulus,<br>integers a and b with $0 \le a, b < N$ |
| ***Modular addition:*** | Find the integer c with $0 \le c < N$ such that $c \equiv a+b \bmod N$ |
| ***Modular multiplication:*** | Find the integer c with $0 \le c < N$ such that $c \equiv a \times b \bmod N$ |

***Standard techniques for modular multiplication:***

- *Multiply a and b and divide the result by N.*
- *Interleave steps necessary for multiplication with steps for modular reduction*

***The methods for modular arithmetic depend strongly on the chosen representation of integers:***

*Basis* $\mathbf{B} = \{B_{m-1},\ldots,B_0\}$
*integer a* $\Leftrightarrow$ *vector* $(a_{m-1}, \ldots, a_0)$ of coordinates $a_j$ with respect to $B_j$

## *Some design approaches*

- **The chinese remainder approach**
  - *Basis* $\mathbf{B} = \{B_{m-1}, \ldots, B_0\}$ consists of coprime integers
  - additions, multiplications mod $B_{m-1}, \ldots, \mathrm{mod}\ B_0$ in parallel
  - modular reduction difficult

- **The full-parallel-multiplier-approach**
  - *Basis* $\mathbf{B} = \{B_{m-1}, \ldots, B_0\}$ consists of powers of a fixed radix B
  - widely  a software approach

- **The serial-parallel-multiplier approach**
  - *Basis* $\mathbf{B} = \{B_{m-1}, \ldots, B_0\}$ consists of powers of a fixed radix B

# *Algorithmic description of interleaved modular multiplication*

Given : Basis $\mathbf{B} = \{B_{m-1}, ..., B_0\}$ with $B_j = B^j$, for some radix $B = 2^k$ and
$a = (a_{m-1}, ..., a_0)$, $b = (b_{m-1}, ..., b_0)$, $c = (c_{m-1}, ..., c_0)$

Input: Operands a, b of length m, radix B, modulus N

Output: $c = a \cdot b \bmod N$

1) $c \leftarrow 0$;
2) for $i \leftarrow m - 1$ downto 0 do
3) $\quad c \leftarrow c \cdot B + a_i \cdot b$
4) $\quad c \leftarrow c \bmod N$

interleaved reduction

**Basic design options for a modular multiplication device**

*The full-parallel-multiplier approach:*
large radix $B = 2^k$ (i.e. $k = 16$)

**core component: k-bit parallel-multiplier**

- calculates products $a_i \cdot b$ in blocks of length k
- reduced number of loop iterations

*The serial-parallel-multiplier approach:*
small radix $B = 2^k$ (i.e. $k = 1,..,3$)

**core component: fast parallel adder**

- small number of products $a_i \cdot b$
- addition of intermediate results in one step

## *The serial-parallel-multiplier approach*

**Method**

- ➔ multiplication is broken down to shifts and additions
- ➔ modular reduction interleaved with steps for multiplication

**Basic constituents**

- ➔ fast parallel adder
- ➔ algorithm for modular reduction

## *The problem of fast parallel addition*

Given : Basis **B** = $\{B_{m-1}, ..., B_0\}$ with $B_j = B^j$ , for some fixed radix B, set of digits **Z.**

Integers $a = (a_{m-1}, ..., a_0)$ and $b = (b_{m-1}, ..., b_0)$ , $a_i, b_i \in Z$.

Problem: Design a device that determines $s = (s_m, ..., s_0)$ , where $s = a + b$

| | |
|---|---|
| Area: | as small as possible |
| Time: | as short as possible |
| Scalability: | as good as possible |

*Non-redundant set **Z** of digits:*

**Z** = {0,1,....,B -1}

representation of integer is unique

- ➔ **Carry-ripple-adder**
- ➔ **Carry-look-ahead-adder**
- ➔ **Carry-completition-adder**

*Redundant set **Z** of digits:*

#(**Z**) > B

representation of integer is not unique

- ➔ **Carry-save-adder**
- ➔ **Delayed-carry-adder**
- ➔ **RSD-adders**

## *Comparision of parallel adders (i)*

Basis **B** = $\{B_{m-1}, ..., B_0\}$ with $B_j = B^j$ , for radix B=2, set of digits: **Z** = {0,1}

➔ **Carry-ripple-adder**

| | | |
|---|---|---|
| Area: | O(m) | m full-adder-cells |
| Time: | O(m) | |
| Scalability: | good | |

➔ **Carry-look-ahead-adder**

| | |
|---|---|
| Area: | $O(m \cdot \log m)$ |
| Time: | O(log m) |
| Scalability: | difficult |

➔ **Carry-completition-adder**

| | |
|---|---|
| Area: | $O(m \cdot \log m)$ |
| Time: | O(log m) |
| Scalability: | difficult |

## *The carry-ripple-adder*

Given : Basis **B** $= \{B_{m-1}, ..., B_0\}$ with $B_j = B^j$, for the radix $B = 2$, set of digits **Z** $= \{0,1\}$

Integers $a = (a_{m-1}, ..., a_0)$ and $b = (b_{m-1}, ..., b_0)$, $a_i, b_i \in Z$.
$s = (s_m, ..., s_0)$ , where $s = a + b$

Addition rule: $s_i = a_i \oplus b_i \oplus c_i$, where $c_0 = 0$
$c_{i+1} = a_i \times b_i \vee a_i \times c_i \vee b_i \times c_i$
$s_m = c_m$

Properties:

| | |
|---|---|
| Area: | O(m), |
| Time: | O(m) |
| Scalability: | good |

Evaluation: completely inadequate for cryptographic applications

## *Comparision of parallel adders (II)*

Basis **B** = $\{B_{m-1}, \ldots, B_0\}$ with $B_j = B^j$ , for radix B=2, set of digits: **Z** = {0,1}

➔ **Carry-Save-adder**

Area: O(m) m full-adder-cells

Time: O(1)

Scalability: good

➔ **Delayed-carry-adder**

Area: O(m) m full-adder-cells + m half-adder-cells

Time: O(1)

Scalability: good

➔ **RSD-adders**

Area: O(m)

Time: O(1)

Scalability: good

# *The carry-save-adder or (3,2)-counter or 3-operand-adder*

Given : Basis **B** $= \{B_{m-1}, ..., B_0\}$ with $B_j = B^j$ , for radix B = 2.

| | |
|---|---|
| set of digits **Z** first operand: | **Z** = {0,1} |
| second operand | pair of binary digits from **Z** |
| sum: | pair of binary digits from **Z** |

**Input:** $a = (a_{m-1}, ..., a_0)$,
$b = (b_{m-1}, ..., b_0)$, $c=(c_{m-1}, ..., c_0)$

**Output:** $s = (s_{m-1}, ..., s_0)$ and $c' =(c'_{m-1}, ..., c'_0)$ such that

$$s + c' = a + b + c$$

Addition rules.

$$s_i = a_i \oplus b_i \oplus c_i$$
$$c'_{i+1} = a_i \times b_i \vee a_i \times c_i \vee b_i \times c_i$$

Properties: Area: O(m) , scales easily
Time: O(1)

The carry-save-adder is a straightforward derivate of the carry-ripple-adder.

## *The panic-adder or a fast-average-case adder*

**Idea:** On average, the longest carry chain when adding two m-bit numbers is of length

$$\log_2 m.$$

**Thus:**

— Divide m bits into blocks of length b, i.e., m/b blocks.

— Choose b large enough such that there is on average no carry between consecutive blocks.

— Realize the blocks of length b as carry-look-ahead-adders.

— A block gets into panic, if an incoming carry would travel through it.

**Clue:** No time needed for carry propagation on average, but only some little extra time in the unlikely case that a block gets into panic.

## *Probability that a block gets into panic*

probability for panic in block of length b = $(1/2)^b$

probability for no panic in l blocks of length b = $( 1 - (1/2)^b )^l$

probability for panic in at least 1 out of l blocks = $1 - ( 1 - (1/2)^b )^l$

## *The panic adder*

Given : Basis $\mathbf{B} = \{B_{m-1}, \ldots, B_0\}$ with $B_j = B^j$ , for the radix B = 2, set of digits $\mathbf{Z} = \{0,1\}$

Integers $a = (a_{m-1}, \ldots, a_0)$ and $b = (b_{m-1}, \ldots, b_0)$, $a_i, b_i \in Z$
$s = (s_m, \ldots, s_0)$, where $s = a + b$

Addition rule:

as described before

Properties:

| | |
|---|---|
| Area: | O(m), |
| Time: | O(1) |
| Scalability: | very good |

Evaluation: Extraordinarily suited for cryptographic applications as operands are long, thus resulting in a good average time.

# *Methods for interleaved modular reduction*

**Problem:** Given the modulus N and integers a and b with $0 \leq a, b < N$.
Find a + b mod N.

**Methods to solve the problem:**

➔ **Comparision of sizes and subtraction**

$c \leftarrow a + b$

if $c \geq N$ then $c \leftarrow c - N$

Problem: The comparision " is $c \geq N$ ? " can be difficult

➔ **Omura reduction**

For a fixed power $2^s > N$ the integer $2^s - N$ is stored

This replaces comparision of sizes by overflow detection

➔ **Montgomery reduction**

avoids completely operations based on estimates of sizes

Replaces operations with N by operations with the radix B

Drawback: conversion to special represention of integers is necessary

## *Comparision of some serial-parallel-designs*

**The ZDN-Unit**

Concept due to H. Sedlak (1988)

Characteristics:

- use of Booth´s algorithm to reduce the number of partial products
- special modular reduction based on an easy comparison with (2/3)·N
- execution of multiplication and reduction simultaneously
- use of a special three-operand-adder via carry-save-adder and panic adder

Time for one modular multiplication: ~ (1/2.8)·m clock cycles in practice
= (1/3)·m clock cycles in theory

**The Brickell-design**

Characteristics:

- based on a delayed-carry-adder
- modular reduction similar to Omura's approach

Time for one modular multiplication: m + 7 clock cycles

## *Comparision of some serial-parallel-designs (II)*

**Radix-2 and radix-4 RSD-units**

N. Takagi and S. Yajima (1992)

Characteristics:

- based on RSD-representation of integers
- modular reduction similar to Omura‘s approach

| | | |
|---|---|---|
| <u>Time for one modular multiplication:</u> | m clock cycles | radix B = 2 |
| | m/2 clock cycles | radix B = 4 |

**Radix-8 RSD-unit**

extrapolation based on Takagi -Yajima-design

Characteristics:

- based on RSD-representation to radix B = 8
- modular reduction similar to Omura‘s approach

<u>Time for one modular multiplication:</u> m/3 clock cycles

# *Comparision of some serial-parallel-designs (III)*

Comparing area and time for modular multiplication for some serial-parallel units:

| | ZDN-unit | Brickell | RSD-2 | RSD-4 | RSD-8 |
|---|---|---|---|---|---|
| total area | 1 | 1.8 | 2 | 3 | < 5 |
| time for modular multiplication | 1 | 2.8 | 2.8 | 1.4 | 0.9 |

- Data for the ZDN-unit are normalized to 1
- Area for Brickell, RSD-2 and RSD-4 extrapolated to modern chip technology
- Area estimate for RSD-8 based on a preliminary synthesis with automatic design tool

## *Conclusion*

- We described various design approaches for long integer arithmetic units
- The serial-parallel-multiplier seems to be the adequate approach
- Under all serial-parallel designs under consideration the ZDN-unit offers the best ratio between area consumption are performance