ECC Summer School 2004

Hardware Implementation of ECC

Jan Pelzl

Communications Security Group

Ruhr-Universität Bochum

http://www.crypto.rub.de

Overview

● Overview
Introduction
Hardware Design
Hardware for $GF(2^n)$
Hardware for $GF(p)$
Practical Performance

1. Introduction
   - Why Hardware Acceleration?
   - What is Hardware?
2. Hardware Design
   - Design Process
   - Example
3. Hardware for $GF(2^n)$
   - Number Representations
   - Addition, Subtraction
   - Squaring
   - Multiplication
   - Reduction
4. Hardware for $GF(p)$
   - Number Representations
   - Addition, Subtraction
   - Squaring, Multiplication
   - Reduction
5. Practical Performance
   - Examples

Why Hardware Acceleration?

● Overview

Introduction

○ Why Hardware Acceleration?

● What is Hardware?

● Tradeoff ASIC vs. FPGA

Hardware Design

Hardware for $GF(2^n)$

Hardware for $GF(p)$

Practical Performance

Hardware (HW) acceleration required if conventional processors too slow.

$\Rightarrow$ two scenarios:

1. High end server with many (thousands) PK operations per second
   $\longrightarrow$ Let HW do computational expensive PK operations
   $\longrightarrow$ Example: SSL - Server

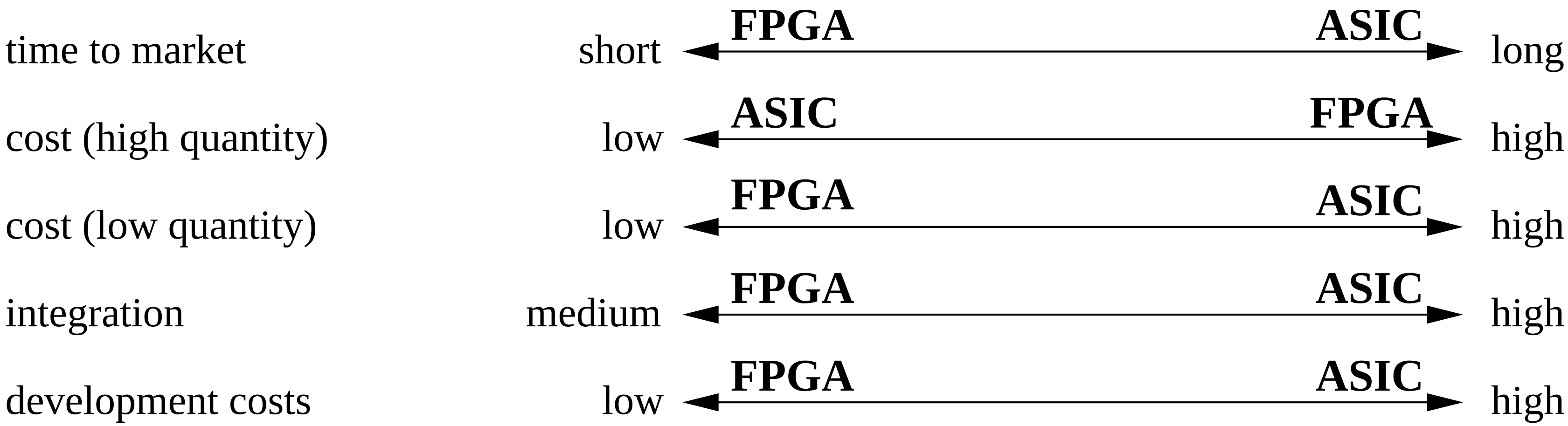
2. Embedded device with low power CPU (cheap)
   $\longrightarrow$ HW extension to enable for PK cryptography
   $\longrightarrow$ Examples: Mobiles, low-end PDA, smart cards

What is Hardware?

Electronic circuit designed mainly for a single purpose (aka special purpose hardware).

- Application Specific Integrated Circuit (ASIC) (=chip for one special application)
- Field Programmable Gate Array (FPGA) (=reprogrammable circuit, additional logic)

Tradeoff ASIC vs. FPGA



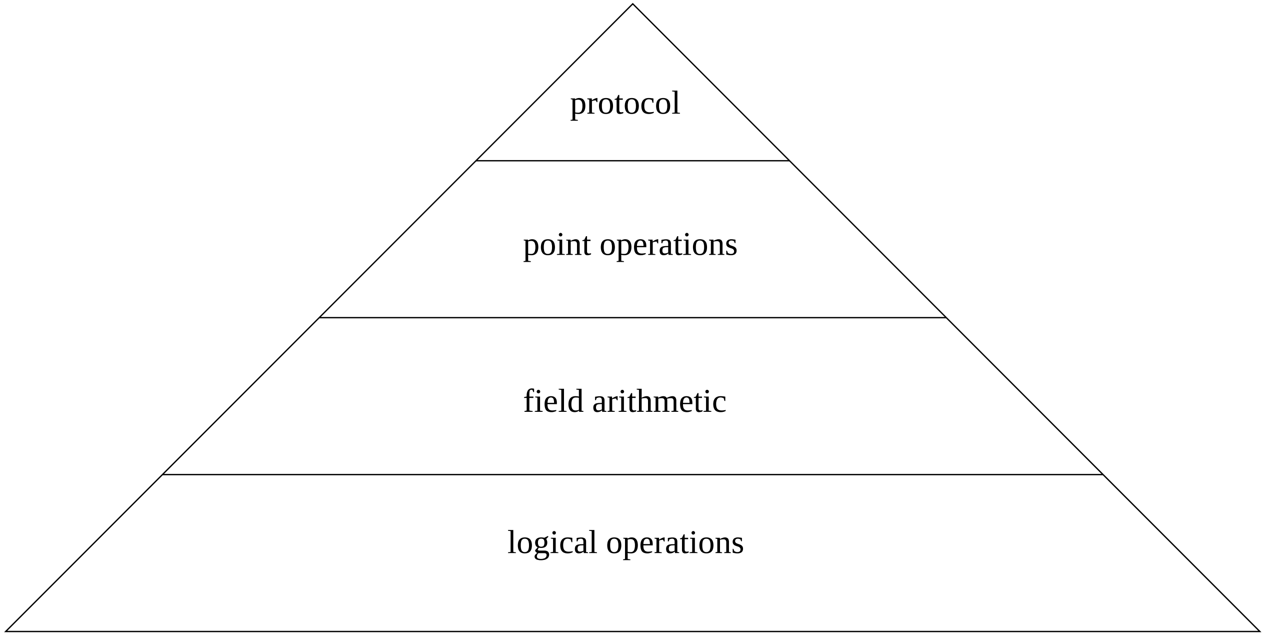
time to market	short	FPGA	ASIC	long
cost (high quantity)	low	ASIC	FPGA	high
cost (low quantity)	low	FPGA	ASIC	high
integration	medium	FPGA	ASIC	high
development costs	low	FPGA	ASIC	high

Figure 1: Comparison ASIC vs. FPGA

Design Process

Objective: Project structure and interaction of different functional units onto specific hardware.

E.g., Functionality of an elliptic curve cryptosystem:



protocol

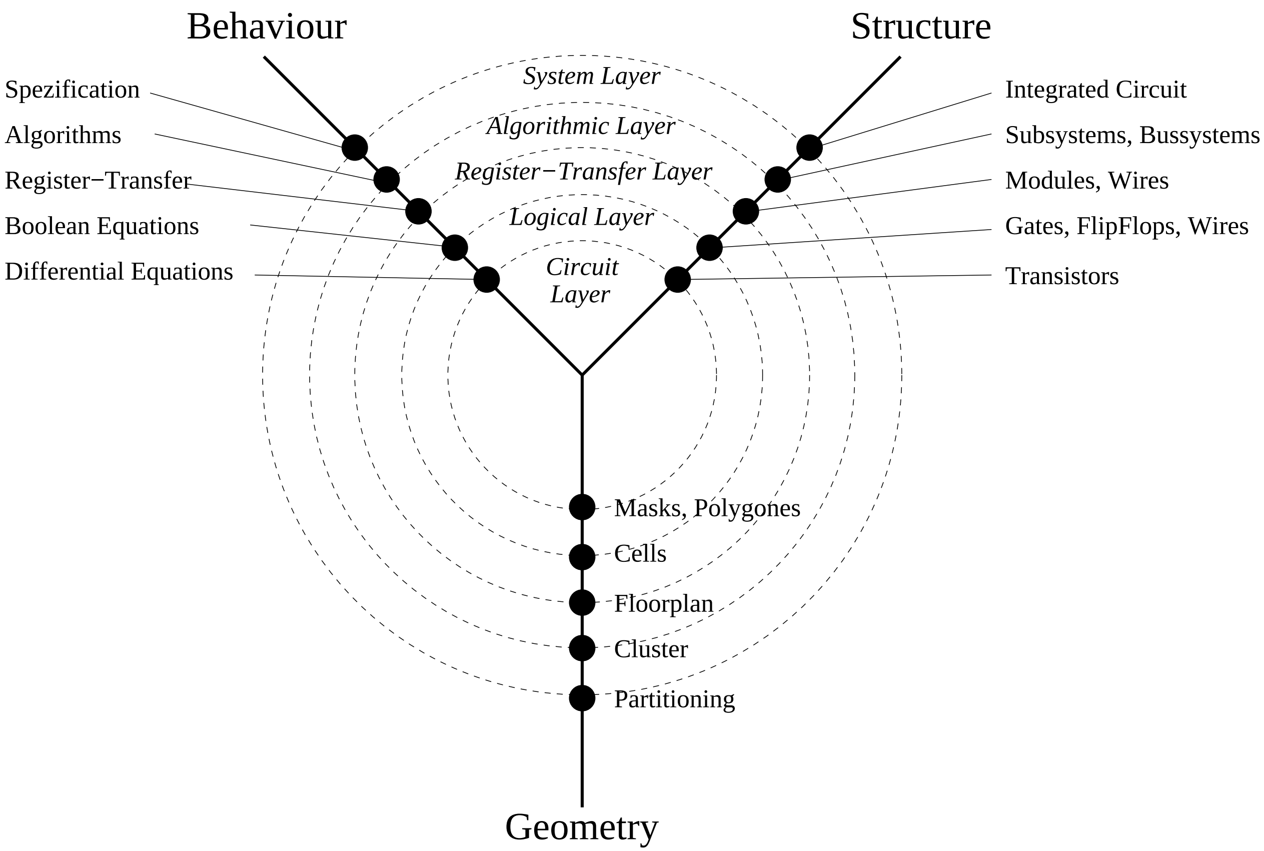
point operations

field arithmetic

logical operations

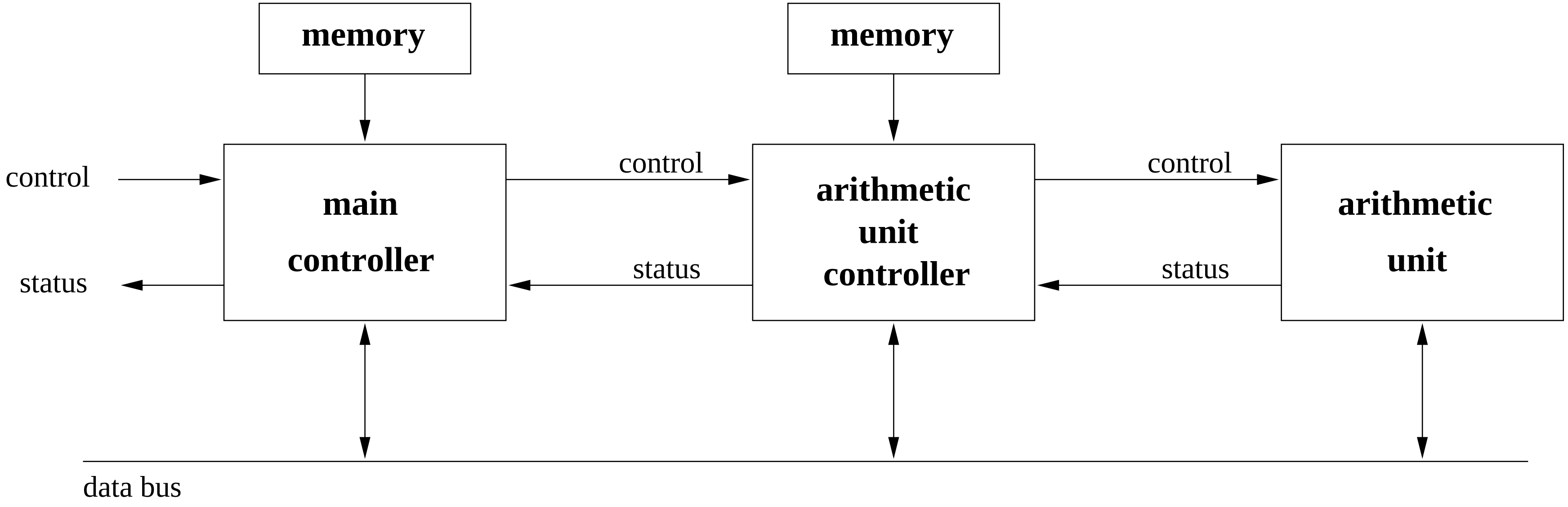
Design Process (2)

Different design methods available, e.g., use Gajski Y-Diagram to specify 3 approaches to the hardware:



Example

Processor Architecture for ECC (subsystems) [OP00]:



Number Representation

Most popular basis for numbers in $GF(2^n)$:

- Polynomial basis:
  - specified by an irreducible polynomial $f(z)$ modulo 2 (field poynomial)
  - bit string $(a_{n-1}a_{n-2}...a_1a_0)$ represents polynomial $a_{n-1}z^{n-1} + ... + a_1z + a_0$
  - field arithmetic is implemented as polynomial arithmetic modulo $f(z)$

- Normal basis:
  - specified by an element θ
  - bit string $(a_{n-1}a_{n-2}...a_1a_0)$ represents element $a_0\theta + a_1\theta^2 + a_2\theta^{2^2} + ... + a_{n-1}\theta^{2^{m-1}}$
  - for special classes of normal basis (Type T low-complexity normal basis), efficient implementations possible

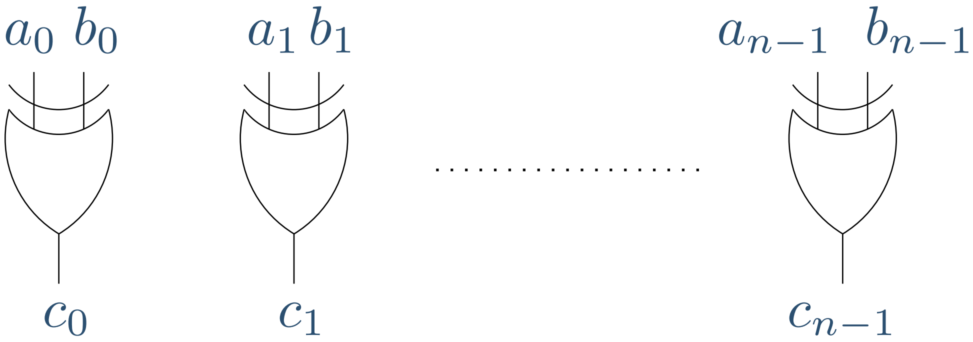
In HW, polynomial basis advantegeous. (focus here)

Addition, Subtraction

In Hardware,

- addition and subtraction over $GF(2^n)$ by bitwise XOR
- variables of arbitrary bit length can be processed simultaneously

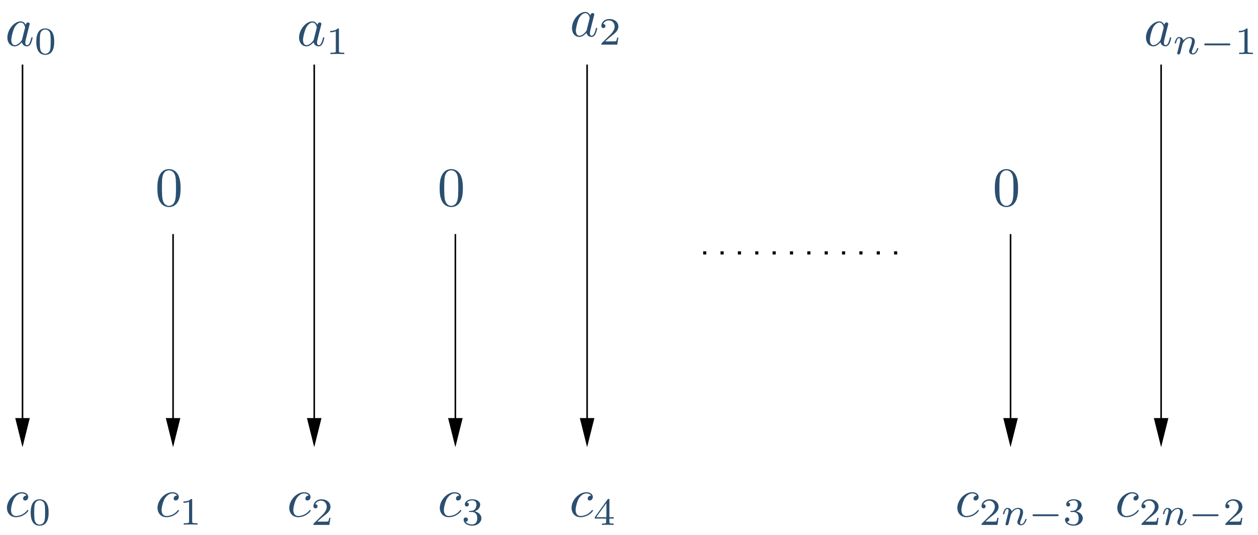
$$c = a \pm b = a \oplus b$$



Squaring

No complicated logic required, only routing of input bits to correct output positions:

$$c = a^2$$



Multiplication

Array type multiplication:

- n multiplicand bits are processed in parallel
- multiplier bits are processed one bit each step
- polynomial multiplication and reduction are interleaved each step

The array type multiplication can be performed in two major ways:

- Least Significant Bit first (LSB first)
- Most Significant Bit first (MSB first)

Multiplication (2)

Let $A = \sum_{j=0}^{n-1} a_j z^j$, $B = \sum_{j=0}^{n-1} b_j z^j$, and $C = \sum_{j=0}^{n-1} c_j z^j$ the product of A and B.

LSB first multiplication:

$$\begin{aligned} C &= AB \bmod f(z) \\ &= b_0 A + b_1 (A \cdot z \bmod f(z)) \\ &\quad + b_2 (A \cdot z^2 \bmod f(z)) \\ &\quad + ... + b_{n-1} (A \cdot z^{n-1} \bmod f(z)) \end{aligned}$$

MSB first multiplication:

$$\begin{aligned} C &= AB \bmod f(z) \\ &= (...(A \cdot b_{n-1} \cdot z \bmod f(z) \\ &\quad + A \cdot b_{n-2}) \cdot z \bmod f(z) \\ &\quad + ... + A \cdot b_1) \cdot z \bmod f(z) + A \cdot b_0 \end{aligned}$$

Multiplication (3)

Digit Serial Multiplier (DSM):

- n multiplicand bits are processed in parallel
- D bits of multiplier are processed simultaneously each step $\longrightarrow$ total number of steps: $\lceil n/D \rceil$
- polynomial multiplication and reduction are interleaved each step
- Most Significant Digit first (MSD fist) multiplier
- Least Significant Digit first (LSD fist) multiplier

(For more details on $GF(2^n)$ multiplier in HW, refer to [SP98].)

Reduction

- In case of addition/ subtraction: simply add $f(z)$ to intermediate results if larger than $f(z)$.
  $\longrightarrow$ can hardwire reduction polynomial

- In case of squaring by table lookup: shift and add $f(z)$ to eliminate all bit positions with ’1’.
  $\longrightarrow$ can hardwire reduction polynomial multiple times

- interleaved multiplication does not need additional reduction

Number Representation

● Overview

Introduction

Hardware Design

Hardware for $GF(2^n)$

Hardware for $GF(p)$

○ Number Representation
● Addition
● Addition (2)
● Addition (3)
● Addition (4)
● Addition (5)
● Subtraction
● Multiplication
● Multiplication (2)
● Multiplication (3)
● Multiplication (4)
● Squaring
● Reduction

Practical Performance

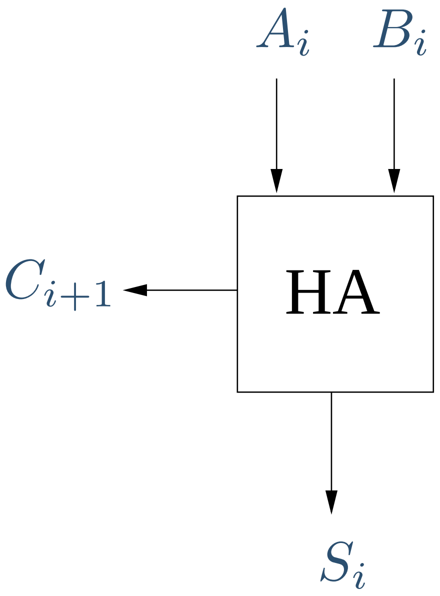
Can express $a \in GF(p)$ in different ways:

- binary form: $a = (a_{n-1}...a_1a_0)_2 = a_{n-1}2^{n-1} + ... + a_12 + a_0$
- redundant bit representation (e.g., for carry save adders)
  → uses more bits than necessary to represent number
- 2's complement for negative numbers:
  - Positive 2's complement numbers are represented as the simple binary.
  - Negative 2's complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

integer		2's complement
signed	unsigned	
5	5	0000 0101
-5	251	1111 1011

Addition

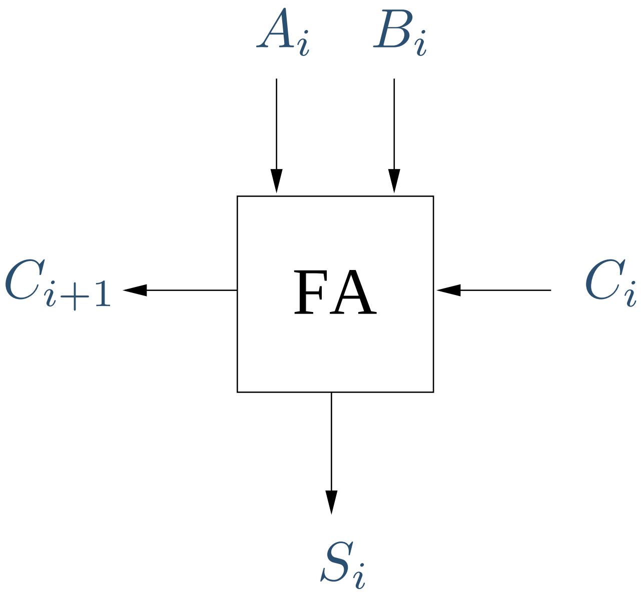
Simple adder for two input bits (half-adder):



A_i	B_i	S_i	C_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Addition (2)

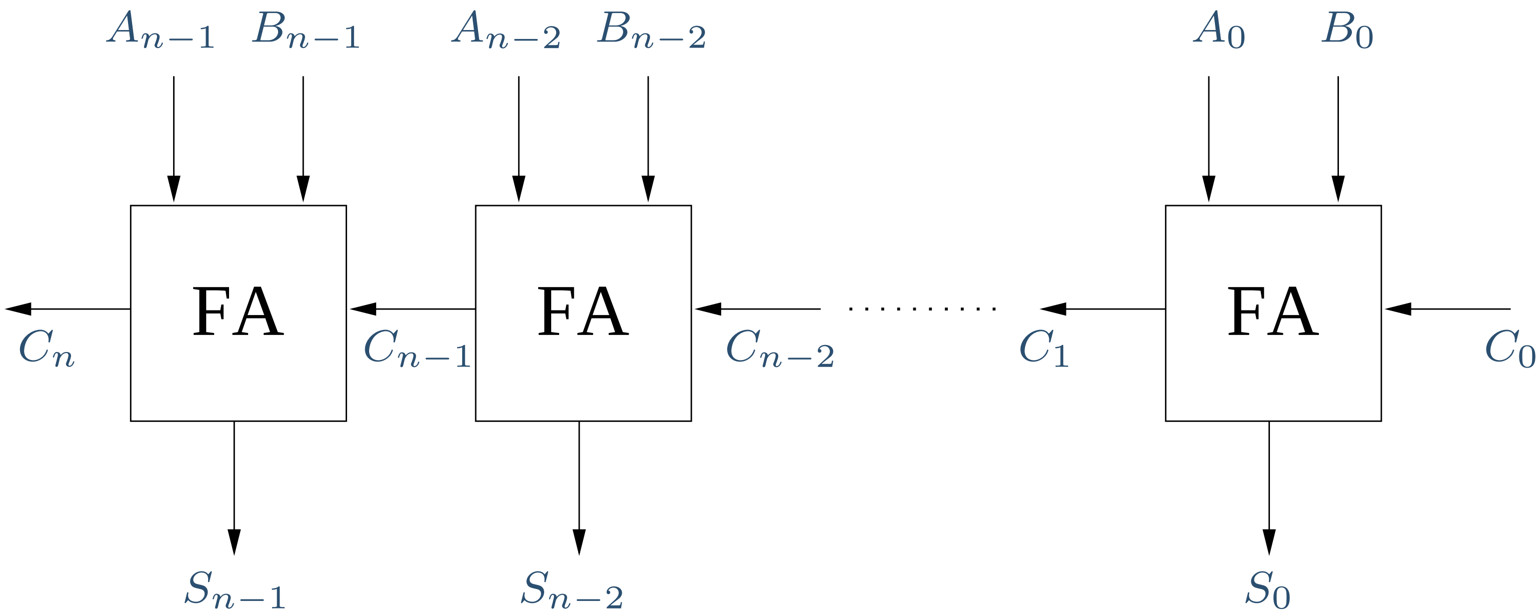
Simple adder for three input bits (full-adder):



A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

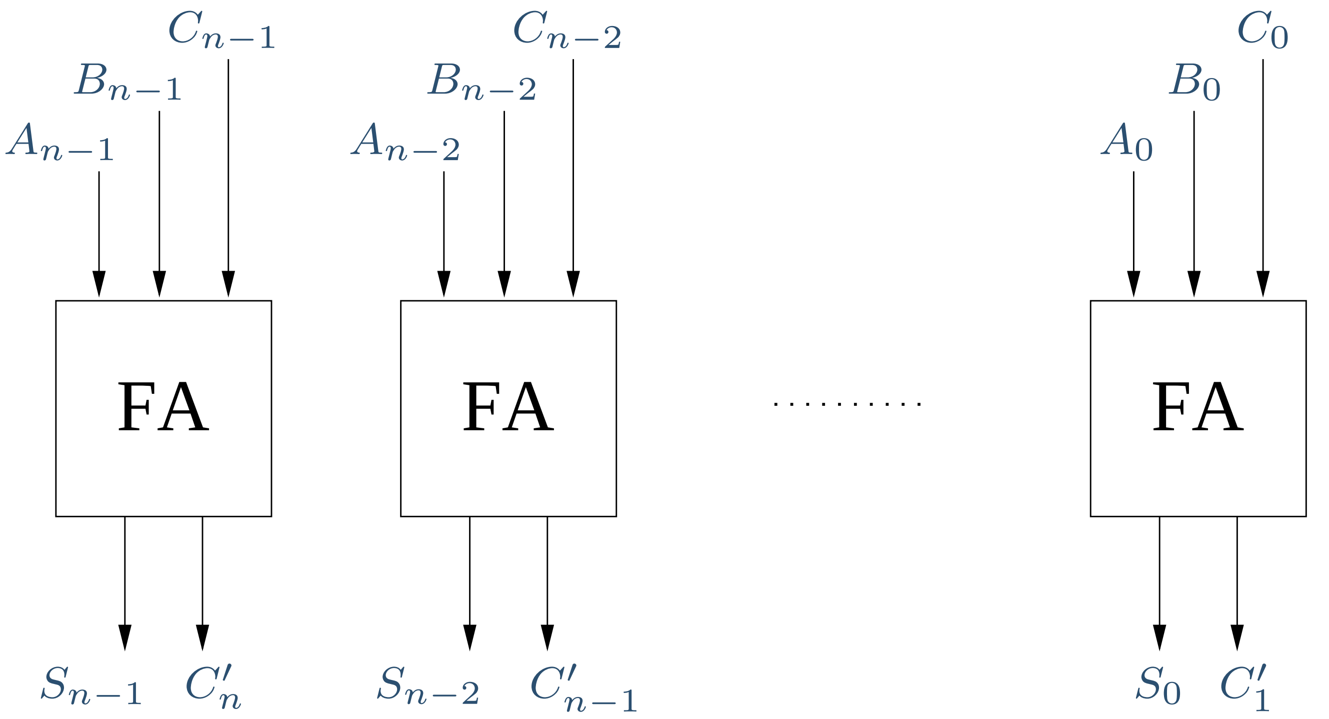
Addition (3)

Carry Propagate Adder (CPA): Latency $O(n)$



Addition (4)

Carry Save Adder (CSA): Latency $O(1)$



Addition (5)

Other methods for addition:

- Bit Serial Adder
  → compute one bit at a time with fulladder
- Carry Completion Sensing Adder
  → figures out when all carrys are computed
- Carry Look-Ahead Adder
  → extra logic to compute carry bits in advance
- Carry Delay Adder
  → two level CSA used for efficient multiplication

(See [Ç. K. Koç96] for overview of addition operations in HW.)

Subtraction

Implement subtraction with addition:

$$a - b = a + b'$$

where b' is $\overline{b} + 1$ and $\overline{b}$ the bit wise complement of b.
The highest carry bit af the addition is ignored.

Example: Let $a = 13_{10} = (1101)_2$ and $b = 6_{10} = (0110)_2$

$$\begin{array}{rrcl} & \overline{b} & = & (1001)_2 \\ & b' & = & \overline{b} + 1 = 1010 \\ \Rightarrow & a - b & = & a + b' \\ & & = & (1101)_2 + (1010)_2 \\ & & = & (0111)_2 = 7_{10} \end{array}$$

Multiplication

Different methods for modular multiplication:

- Schoolbook:
  → add intermediate results
  → reduce in the end
- Interleaved:
  → interleaved multiplication and reduction
  → keeps intermediate results small
- Montgomery:
  → alternative residue number system (RNS)
  → interleaved multiplication and reduction

● Overview

Introduction

Hardware Design

Hardware for $GF(2^n)$

Hardware for $GF(p)$

● Number Representation
● Addition
● Addition (2)
● Addition (3)
● Addition (4)
● Addition (5)
● Subtraction
● Multiplication
○ Multiplication (2)
● Multiplication (3)
● Multiplication (4)
● Squaring
● Reduction

Practical Performance

Multiplication (2)

Interleaved multiplication with shift-add multiplication algorithm:

```
Require: $A < 2^k$, $B < 2^k$, modulus $n$
Ensure: $P = A \cdot B$ such that $0 \leq P \leq 3n$
$P := 0$
for $i = 0$ to $k - 1$ do
    $P := 2P + A \cdot B_{k-1-i}$
    $P := P \bmod n$
end for
return $P$
```

Remark: Since $0 \leq P \leq 3n$ at most 2 subtractions of n are required in the end.

Multiplication (3)

Residue number system for Montgomery multiplication:

$$a \longmapsto a' = a \cdot r \pmod n \quad \text{where} \quad \gcd(r, n) = 1.$$

Montgomery product c' of residues a' and b' defined as

$$c' = a' \cdot b' \cdot r^{-1} \pmod n$$

Practice: Use $r = 2^k$ for fast computation (next slide)

Multiplication (4)

● Overview

Introduction

Hardware Design

Hardware for $GF(2^n)$

Hardware for $GF(p)$

● Number Representation
● Addition
● Addition (2)
● Addition (3)
● Addition (4)
● Addition (5)
● Subtraction
● Multiplication
● Multiplication (2)
● Multiplication (3)
○ Multiplication (4)
● Squaring
● Reduction

Practical Performance

Binary add-shift algorithm for Montgomery multiplication:

```
Require: $A' < 2^k, B' < 2^k$, modulus $n$, $r = 2^k$
Ensure: $U = A' \cdot B' \cdot 2^{-k}$ such that $0 \leq P < 2n$
U := 0
for $i = 0$ to $k - 1$ do
    $U := U + A_i \cdot B$
    if $U$ is odd then
        then $U := U + n$
    end if
    $U := U/2$
end for
```

Squaring

- in $GF(p)$ usually done with multiplication HW
- special HW can make use of special structure
  - reuse intermediate results
  - but: increase in area

Reduction

Reduction following addition/ subtraction ($c = a \pm b$):

- if $a + b \geq n$, compute $a + b - n$
- if $a - b < 0$ compute $a + n - b$

Reduction following multiplication/ squaring ($c = a^2$ or $c = a \cdot b$):

- use interleaved or montgomery multiplication (reduction included)
- else: compute remainder of c/n

Timings

Tradeoff time vs. area.
⇒ optimize area-time (AT) product

Timings for finite field/ ECC operations of some hardware platforms:

FPGA/ASIC	operation	time	reference
FPGA	512bit multiplication $GF(p)$	$2.37ms$	[BP99]
ASIC	160bit multiplication $GF(p)$	$4.1s$	[STK00]
FPGA	163bit ECC scalarmult. $GF(2^n)$	$144s$	[GCE$^+$01]
FPGA	191bit ECC scalarmult. $GF(p)$	$3ms$	[OP01]

Further Reading

- Excellent and brief overview of hardware adders and multipliers for arithmetic in $GF(p)$ given in [Ç. K. Koç96]. Very useful!
- For fields $GF(2^n)$, refer to [SP98]. An overview over different types of multipliers over extensions fields of characteristic two and their optimization is presented.

Literature

References

[BP99] T. Blum and C. Paar. Montgomery modular multiplication on reconfigurable hardware. In *Proceedigns of the 14th IEEE Symposium on Computer Arithmetic (ARITH-14)*, pages 70–77, 1999.

[Ç. K. Koç96] Ç. K. Koç. RSA Hardware Implementation. RSA Laboratories Technical Report TR-801, RSA Laboratories, Version 1.0 – April 19th, 1996.

[GCE$^+$01] N. Gura, S. Chang, H. Eberle, G. Sumit, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila. An End-to-End Systems Approach to Elliptic Curve Cryptography. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume LNCS 1965, pages 351–366. Springer-Verlag, 2001.

[OP00] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, volume LNCS 1965. Springer-Verlag, 2000.

[OP01] G. Orlando and C. Paar. A Scalable $GF(p)$ Elliptic Curve Processor Architecture for Programmable Hardware. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2001*, volume LNCS 2162, pages 348–363. Springer-Verlag, May 14-16, 2001.

[SP98] L. Song and K. K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, June 1998.

[STK00] E. Savas, A. F. Tenca, and C . K. Koç. A scalable and unified multiplier architecture for finite fields gf(p) and gf(2). In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, volume LNCS 1965, pages 281–296. Springer-Verlag, 2000.