Adv. Radio Sci., 8, 295–305, 2010
www.adv-radio-sci.net/8/295/2010/
doi:10.5194/ars-8-295-2010

Advances in Radio Science

Resource efficiency of hardware extensions of a 4-issue VLIW processor for elliptic curve cryptography

T. Jungeblut[1], **C. Puttmann**[1], **R. Dreesen**[2], **M. Porrmann**[1], **M. Thies**[2], **U. Rückert**[3], **and U. Kastens**[2]

[1]University of Paderborn, System and Circuit Technology, Germany
[2]University of Paderborn, Department of Computer Science, Germany
[3]Bielefeld University, Cognitive Interaction Technology, Germany

Abstract. The secure transmission of data plays a significant role in todays information era. Especially in the area of public-key-cryptography methods, which are based on elliptic curves (ECC), gain more and more importance. Compared to asymmetric algorithms, like RSA, ECC can be used with shorter key lengths, while achieving an equal level of security. The performance of ECC-algorithms can be increased significantly by adding application specific hardware extensions.

Due to their fine grained parallelism, VLIW-processors are well suited for the execution of ECC algorithms. In this work, we extended the fourfold parallel CoreVA-VLIW-architecture by several hardware accelerators to increase the resource efficiency of the overall system. For the design-space exploration we use a dual design flow, which is based on the automatic generation of a complete C-compiler based tool chain from a central processor specification. Using the hardware accelerators the performance of the scalar multiplication on binary fields can be increased by the factor of 29. The energy consumption can be reduced by up to 90%. The extended processor hardware was mapped on a current 65 nm low-power standard-cell-technology. The chip area of the CoreVA-VLIW-architecture is 0.24 mm^2 at a power consumption of 29 mW/MHz. The performance gain is analyzed in respect to the increased hardware costs, as chip area or power consumption.

Correspondence to: T. Jungeblut
(tj@hni.upb.de)

1 Introduction

Nowadays, security is a vital issue to everyone, who wants to exchange confidential or private information electronically. Especially in the area of public key schemes, elliptic curve cryptography (ECC) gained increasing popularity. Compared to other asymmetric algorithms like RSA, methods based on elliptic curves provide an equivalent security level while using smaller key sizes. Therefore, ECC requires less memory and is well suited for embedded devices.

The performance of algorithms for ECC can be significantly increased by modifying the hardware architecture that executes these algorithms. However, this performance improvement can only be achieved at the expense of additional costs in terms of chip area or power consumption, respectively. We define the trade-off between performance and costs as resource efficiency. In this paper, we analyze different modifications of a VLIW processor, in order to accelerate the performance of elliptic curve cryptography over binary finite fields. More precisely, we evaluate instruction set extensions (ISE) as well as dedicated hardware accelerator blocks in respect to their resource efficiency. For this purpose, we present a holistic methodology for the automated evaluation of instruction set extensions. In order to accelerate algorithms for elliptic curve cryptography, we focus on the optimization of finite field arithmetic.

Finite field arithmetic is a very important issue in public key cryptography and especially in elliptic curve cryptography, as stated in Hankerson et al. (2004). It is generally accepted that multiplication is one of the most resource consuming operations in finite field arithmetic because it is required very often in cryptographic algorithms. The finite fields that are used in elliptic curve cryptography are typically binary finite fields, which are of the form $\mathbb{F}_{2^m}$ and can be represented as vector spaces over $\mathbb{F}_2$ using polynomial basis representation. The multiplication in polynomial basis consists of a polynomial multiplication followed by a

Published by Copernicus Publications on behalf of the URSI Landesausschuss in der Bundesrepublik Deutschland e.V.

modular reduction. The latter can be done very efficiently by using a sparse irreducible polynomial that generates the finite field. Therefore, the cost of multiplication in finite field arithmetic is dominated by the cost of the polynomial multiplication. A less critical but also important function is the squaring operation in finite fields. Therefore, we consider both, the finite field multiplication as well as the finite field squaring operation, as focus of our optimization methods.

Our analysis is based on the multiplication and squaring of polynomials of length 233 over $\mathbb{F}_2$. Among others, the finite field $\mathbb{F}_{2^{233}}$ has been suggested by the ECC National Institute of Standards and Technology (NIST) (2000). There are several proposals in the literature for multiplication of two polynomials (see Knuth (1998) or von zur Gathen and Gerhard (2003) for a comprehensive list). Each solution is appropriate for some range of polynomial degrees and is generally selected according to the degree and implementation platform. The most popular method to multiply polynomials is the classical method, whose cost is quadratic in the degree of polynomials involved. Depending on the polynomial degrees, other methods are asymptotically less expensive than the classical algorithm. The point where an algorithm gets better than an asymptotically more expensive method is called *crossover point*. We choose the Karatsuba method, which is appropriate for our polynomial degree $m = 232$ and has a cost of $O(m^{\log_2 3})$, cf. Hankerson et al. (2004). There also exist methods, e.g. the Cantor or FFT method, which have asymptotically lower costs than the Karatsuba method, but their crossover points are so high that they are only suitable for very large polynomial degrees. Hence, in this work we want to analyze the Karatsuba method in respect to the resource efficiency of hardware modifications based on instruction set extensions.

The rest of the paper is organized as follows. Section 1 is devoted to the discussion of different works in the literature, which concern ISE in general as well as optimizations for finite field arithmetic and cryptography in particular. In Sect. 2 we review elliptic curve arithmetic and algorithmic aspects. Section 3 describes our hardware extension methodology and the architecture of our VLIW-processor. The implementation results and analysis of the resource efficiency can be found in Section 4. Finally, Sect. 5 concludes the paper.

Related work

There are several publications about instruction set extensions and their applications in elliptic curve cryptography over binary finite fields. Tillich and Großschädl (2004) consider the integration of bit-level and word-level multiply and shift methods into a SPARC V2 core to increase its performance for multiplication in $\mathbb{F}_{2^{191}}$. In Großschädl and Kamendje (2003) a dual field adder (DFA) module augments a 16-bit RISC architecture to increase the performance of arithmetic in binary fields. In this way, the authors created a unifying multiplier for both numbers and binary polynomials. In Großschädl and Savaş (2004) a multiply and accumulate (MAC) structure is integrated into a MIPS32 core. MACs have been already used in DSP processors to increase the performance of finite impulse response (FIR) filters and the authors have used the similarities between polynomial multiplications and convolution methods to make use of these structures. Finally, Kumar and Paar (2004) discuss the performance gain achieved by integrating a polynomial multiplication unit into the data path of an 8-bit microcontroller, which is equipped with a reconfigurable module. All of the above methods use application-specific information about classical polynomial multiplication methods and find suitable instructions, mostly the so-called `MULGF2`, and measure the achieved improvement in execution time.

In our work, we consider the performance gain of the Karatsuba method by means of integrating both, application-specific and automatically generated instruction set extensions into the processor and analyze their resource efficiency. The difference from existing contributions is that we consider the energy consumption of the instruction set extensions. Also, we define resource efficiency not only as the execution time in terms of clock cycles, but also as the hardware costs in terms of chip area and power consumption. Furthermore, our contribution evaluates the impact of code size reduction through instruction set extension. An example for another processor implementations of application specific processors is presented by Tensilica. The Tensilica tools, as shown in Gonzalez (2000), offer the generation of a tool chain and a hardware description from a single processor specification. Their approach allows the extension of a predefined processor architecture by application-specific instructions. In contrast, our approach also exposes design parameters like instruction format, 2/3-address instructions, pipeline depth, forwarding circuit, branch prediction, etc. to the developer. This effectively allows the exploration of a larger design space.

2 Elliptic curve arithmetic

In this section the elliptic curve cryptography based on binary field arithmetic is introduced. The general equation for a non-supersingular elliptic curve E over the binary finite field $\mathbb{F}_{2^m}$ is given by equation:

$$E : y^2 + xy = x^3 + ax^2 + b \tag{1}$$

for appropriate parameters $a, b \in \mathbb{F}_{2^m}$. The set of points $(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$, which satisfy (1), together with the identity element $\mathcal{O}$, generate an additive abelian group. Let $\mathcal{P} = (x_p, y_p)$ and $\mathcal{Q} = (x_q, y_q)$ be two given points on the curve in (Eq. 1). The point addition $\mathcal{P} + \mathcal{Q}$ as well as the point doubling $\mathcal{P} + \mathcal{P}$ are two operations defined on the elliptic curve E, which can geometrically be represented by the tangent and chord operation, respectively. By applying the point addition and point doubling operations, we are able

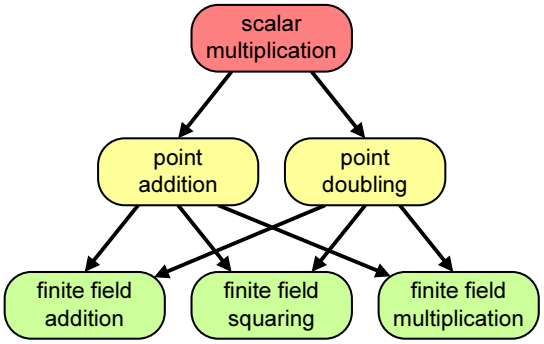

Fig. 1. Arithmetic hierarchy of elliptic curve cryptography.

to multiply an integer k with a point $\mathcal{P}$, which is the result of $k-1$ times adding the point $\mathcal{P}$ to itself. This operation is known as scalar or point multiplication $k \cdot \mathcal{P}$. Figure 1 depicts this hierarchical structure of arithmetic operations used for elliptic curve cryptography over finite fields.

2.1 Projective coordinates

Naturally, point addition and point doubling also require a field inversion when using affine coordinates (x, y). Since inversion is a very expensive operation compared to multiplication, addition and squaring in finite fields, we use projective coordinates. In standard projective coordinates the points on the elliptic curve are represented as a triple (X, Y, Z) in such a way that $x \to X/Z$ and $y \to Y/Z$. By using projective coordinates only one finite field inversion is required at the end of a scalar multiplication in order to transform the projective coordinates back to affine coordinates.

2.2 Montgomery ladder algorithm

In Montgomery (1987) a very efficient method to perform the scalar multiplication is presented, which was applied to elliptic curve cryptography by López and Dahab (1999). The method is known as montgomery ladder and is shown at point level in Algorithm 1. Since in every loop iteration the same operations are performed, namely one point addition and one point doubling, the montgomery ladder algorithm is shielded against timing attacks and simple power analysis attacks.

2.3 Polynomial basis representation

As finite field arithmetic represents the base operations, an efficient representation of the finite field elements in $\mathbb{F}_{2^m}$ is important. The polynomial basis representation can be described as a vector space of dimension m over the field $\mathbb{F}_2$ and is one of the most common representations in ECC. Field elements in polynomial basis representation are expressed as

Algorithm 1 The Montgomery ladder algorithm for scalar multiplication expressed at point level.

Input: A point $\mathcal{P}$ on the elliptic curve E, together with the binary representation of the scalar k as $(k_{i-1}k_{i-2}\ldots k_1k_0)_2$.
Output: $k \cdot \mathcal{P}$
$\mathcal{P}_1 \leftarrow \mathcal{P}$, $\mathcal{P}_2 \leftarrow 2\mathcal{P}$
for j **from** $i-2$ **downto** 0 **do**
  if $k_j = 1$ **then**
    $\mathcal{P}_1 \leftarrow \mathcal{P}_1 + \mathcal{P}_2$, $\mathcal{P}_2 \leftarrow 2\mathcal{P}_2$
  else
    $\mathcal{P}_2 \leftarrow \mathcal{P}_1 + \mathcal{P}_2$, $\mathcal{P}_1 \leftarrow 2\mathcal{P}_1$

binary polynomials of degree $m-1$ as follows:

$$a(x) = \sum_{i=0}^{m-1} a_i \cdot x^i \quad \text{with} \quad \mathrm{a_i} \in \{0,1\} \tag{2}$$

One benefit of binary fields is that the finite field addition is calculated by a carry-free XOR operation of corresponding coefficients. Finite field squaring can be achieved by shifting each bit a_i to a_{2i} and filling the gaps with zeros. Similar to finite field multiplication, the result is a binary polynomial of degree $2m-2$, which has to be reduced modulo an irreducible, sparse polynomial of degree m. However, the finite field multiplication is equivalent to the product of the corresponding polynomials, which is, compared to addition and squaring, the most computational intensive operation.

2.4 Karatsuba multiplication method

In order to reduce the complexity of polynomial multiplication, the method of Karatsuba and Ofman (1963) is applied. Whereas "classically" the coefficients of the product $(a_1x + a_0)(b_1x + b_0) = a_1b_1x^2 + (a_1b_0 \oplus a_0b_1)x + a_0b_0$ from the four input coefficients a_1, a_0, b_1, and b_0 are computed with 4 multiplications and 1 addition, the Karatsuba formula uses only 3 multiplications and 4 additions in binary fields:

$$\begin{aligned}&(a_1x + a_0)(b_1x + b_0) = \\ &a_1b_1x^2 + ((a_1 \oplus a_0)(b_1 \oplus b_0) \oplus a_1b_1 \oplus a_0b_0)x + a_0b_0.\end{aligned} \tag{3}$$

By applying the Karatsuba method to larger polynomials the costs of extra additions vanish compared to those of the saved multiplications and an asymptotical cost of $O(m^{1.59})$ compared to the classical cost of $O(m^2)$ is achieved (cf. Hankerson et al., 2004).

3 Hardware extension approach

In this section we present our architecture, which we use for the further design space exploration.

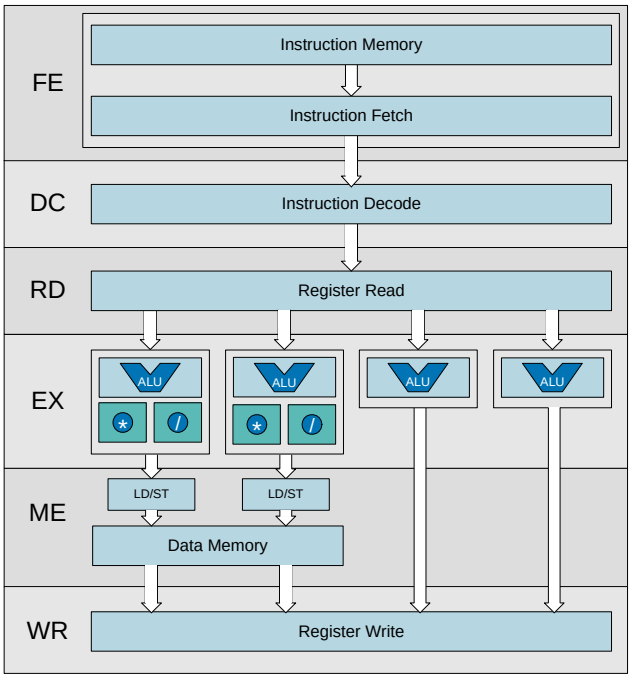

Fig. 2. 6-staged pipeline of the CoreVA processor.

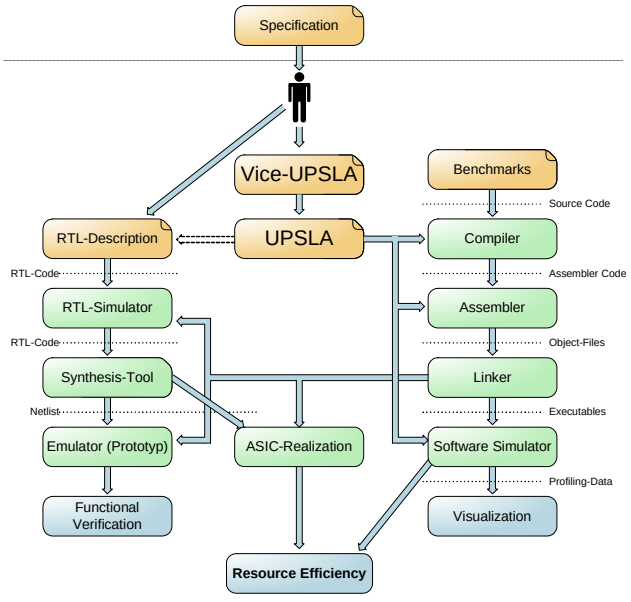

Fig. 3. Dual design-flow for design space exploration.

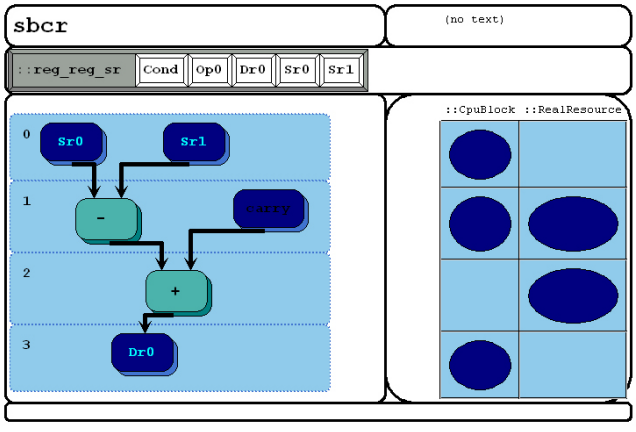

Fig. 4. Graphical user interface for the specification of the UPSLA reference description.

3.1 The CoreVA architecture

The CoreVA architecture represents a 4-issue VLIW architecture (cf. Jungeblut et al., 2007). Using the hardware description language VHDL, CoreVA is specified as a soft macro at register-transfer-level (RTL). The typical harvard architecture with separated instruction and data memory provides a six-staged pipeline (instruction fetch (FE), instruction decode (DE), register read (RE), execute (EX), memory (ME), and register write (WR), cf. Fig. 2).

Besides four arithmetic-logical-units (ALUs), two dedicated multiply-accumulate (MLA) units support fast multiplication. Divisions are accelerated by two dedicated division step units. The register file comprises 31 general purpose 32-bit registers, which can be accessed by all four issue slots. The byte-wise addressable memory supports 8-bit, 16-bit and 32-bit data transmissions using natural alignment and little-endian byte-ordering. The operations follow a two- and three-address format and are all executed in one clock cycle. Most instructions have a latency of one clock cycle, except branch, MLA and load operations, which have a latency of two clock cycles. In SIMD (single instruction, multiple data) mode, two 16-bit words can be processed in each functional unit (FU), which leads to an eightfold parallelism. Two 7-bit condition registers support conditional execution for scalar and SIMD operations. If not all FUs can be utilized, a stop bit in the opcode allows to omit empty trailing instruction slots. This leads to more compact code and reduces power consumption, which is very useful for embedded systems, e.g. smart cards. Still, 64% of the opcode space of the CoreVA architecture is free and can be used for instruction set extensions.

3.2 Reference-specification-based design-flow

We propose a dual design-flow, consisting of the automatic generation of a reference processor specification and an RTL-based hardware development (see Fig. 3).

The design-flow encompasses two domains. In the software domain, a formal processor model is described in the Unified Processor Specification Language (UPSLA) of Kastens et al. (2004), using a graphical user interface called *Vice-UPSLA* (cf. Fig. 4).

UPSLA is a non redundant specification, which allows the rapid generation of a complete tool chain consisting of a ANSI-C compiler, an assembler, a linker, various debugging tools, and an instruction set simulator (ISS). The instruction set simulator features powerful profiling capabilities that allow a detailed analysis of the application, e.g. instruction distribution, memory accesses and utilization of the forwarding circuits. Moreover, hot spots, representing frequently executed parts of the code, can be highlighted directly in the C-code (see Figs. 3 and 6).

In addition, frequently executed data-dependent instruction sequences can be displayed. Selected instruction sequences can be combined into so called *super instructions*. Instead of executing two (*instruction pair*) or more (*instruction block*) instructions consecutively, a super instruction performs the same operation in less clock cycles. This reduces execution time as well as code size and register pressure. Although super instructions sometimes increase power consumption, the reduced execution time leads to lower energy demand. After the super instructions are added to the formal model of the processor, the complete tool chain can automatically be re-generated within a few minutes. After recompiling the software, the developer can check the utilization of the super instructions and the reduction of code size. To analyze the improvement with respect to execution time, the re-compiled application can be simulated and profiled.

The hardware domain of the design-flow starts at the VHDL specification of the processor model. The super instructions are implemented at register-transfer-level (RTL). The compiler generates executables, which are reused in RTL simulation (*Mentor Graphics ModelSim*) to verify the functionality of the adapted processor. Standard cell synthesis with *Synopsys Design Compiler* is used to estimate area requirements. Switching activities from the simulation are back annotated to analyze gate-level power consumption. The evaluation of the resource consumption leads to a refinement of the RTL implementation and the adaption of the processor model (e.g. instruction encoding). These two steps are repeated until the timing and resource requirements are met. An alternative to tightly coupled instruction set extensions, is the integration of loosely coupled dedicated hardware accelerators. These modules can be used through memory-mapped-I/O (MMIO) access. For memory write accesses the address is decoded in the memory stage and the data word is send to the selected module. For read access the output of the respective hardware accelerator is passed to the processor pipeline and written back to the register file.

Due to the independent specifications (UPSLA vs. RTL), a mechanism is needed to check consistency between these two domains. In addition to commercial verification tools (e.g. Synopsys Formality), we apply a *Validation by Simulation* approach for hardware-software-co-design, which is presented in Jungeblut et al. (2007).

For the functional verification of the hardware implementation, we use the RAPTOR2000/X64 rapid prototyping environment (cf. Fig. 8, Porrmann et al., 2009). This modular system offers a large selection of FPGA daughter boards and physical interfaces (e.g. Ethernet), which enable the use in real environments.

The evaluation of our design-flow is based on the finite field arithmetic in binary fields (cf. Sect. 2). Finite field multiplication and finite field squaring represent two of the most time consuming operations in elliptic curve cryptography. Using our framework, we developed two types of hardware extensions: instruction set extensions and a set of hardware accelerators, which are accessible via MMIO.

For the instruction set extension, we analyzed the distribution of the instructions and combined data-dependent instructions into super instructions. The result is a set of three super instructions that are composed of two instructions each.

As hardware accelerators, we implemented two dedicated hardware extensions for the binary field multiplication and binary field squaring. In the next section, both types of extensions are analyzed. Thereby, we do not only consider the speedup in execution time and reduction of code size, but also examine the impact on chip area and power consumption.

4 Implementation and analysis

As a starting point for our design space exploration we have mapped the ECC algorithm introduced in Puttmann et al. (2008) to our CoreVA architecture, using our framework. Without manual optimizations, the parallelizing C-compiler compiled the ECC algorithm to the CoreVA architecture utilizing an average of two functional units. Execution time is 3.6M clock cycles, which equals a speedup of 3.1 compared to a single processor implementation presented in Puttmann et al. (2008). Hand-optimized assembler code enhances the utilization to an average of three functional units and a speedup of 3.7 compared to Puttmann et al. (2008). In the following, these software implementations are referred to as *C* and *ASM*.

Using our framework, we identified three new instruction set extensions: *logical shift left and exclusive or (LSLXOR)*, *logical shift right and exclusive or (LSRXOR)*, and *move bits (MVBITS)*. The first two obviously reflect the nature of the binary field multiplication, the last one that of the binary field squaring. Hence, these instructions are executed in one instead of two clock cycles. Changes to the hardware specification comprise only slight changes to the decoder and the functional units of the processor core. In the following, the CoreVA variant, that includes the three super instructions, is referred to as *CoreVA(ISE)*.

While the instruction set extensions combine existing instructions to super instructions, the second type introduces dedicated hardware accelerators to the processor core (cf. Fig. 9). The extensions are connected via an address decoder. The hardware description of the processor core it-

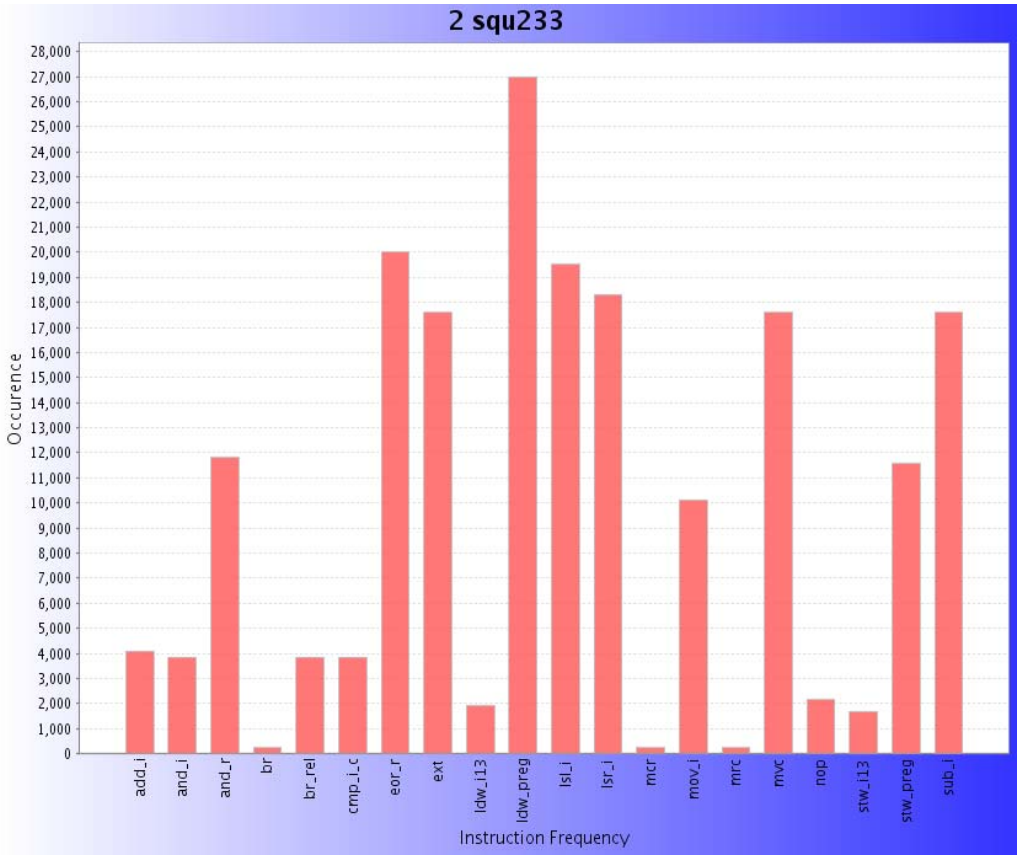

Fig. 5. Instruction distribution of finite field squaring algorithm.

```
0x006e  00000000    A[6] = A6;
0x006f  00000000    A[7] = A7;
0x0070  00000000
0x0071  00000000
0x0072  00000702        lo = A[b & 7];
0x0073  00002106        t = A[(b >> 3) & 7]; hi = t >> 29; lo ^= t << 3;
0x0074  00002106        t = A[(b >> 6) & 7]; hi ^= t >> 26; lo ^= t << 6;
0x0075  00002106        t = A[(b >> 9) & 7]; hi ^= t >> 23; lo ^= t << 9;
0x0076  00001404        t = A[(b >> 12) & 7]; hi ^= t >> 20; lo ^= t << 12;
0x0077  00001404        t = A[(b >> 15) & 7]; hi ^= t >> 17; lo ^= t << 15;
0x0078  00000702        t = A[(b >> 18) & 7]; hi ^= t >> 14; lo ^= t << 18;
0x0079  00001404        t = A[(b >> 21) & 7]; hi ^= t >> 11; lo ^= t << 21;
0x007a  00000702        t = A[(b >> 24) & 7]; hi ^= t >> 8; lo ^= t << 24;
0x007b  00000702        t = A[(b >> 27) & 7]; hi ^= t >> 5; lo ^= t << 27;
0x007c  00002808        t = A[b >> 30]; hi ^= t >> 2; lo ^= t << 30;
0x007d  00002400        if (a >> 31) hi ^= ((b & 0xb6db6db6UL) >> 1);
0x007e  00005088        if ((a >> 30) & 1) hi ^= ((b & 0x24924924UL) >> 2);
0x007f  00002106        c[0] = lo;     c[1] = hi;
0x0080  00000000    }
0x0081  00000000
```

Fig. 6. Hot spots highlighted directly in the C-code.

self remains unchanged. The extensions can be accessed by ordinary memory read and write operations (MMIO). A dedicated part of the address space is mapped to the respective accelerator module. Before computing the binary field multiplication and squaring, the input data has to be transferred to the extensions, which requires one clock cycle per

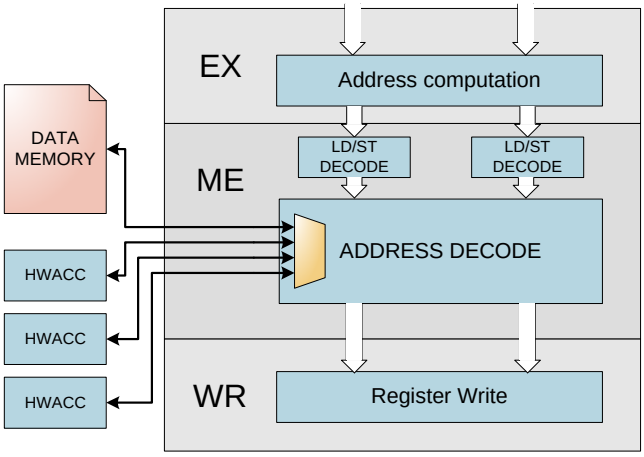

Fig. 7. Interface to dedicated hardware accelerators in the CoreVA architecture.

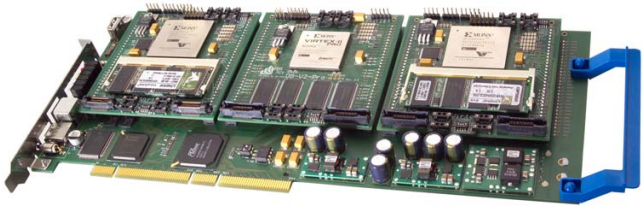

Fig. 8. Rapid prototyping environment RAPTOR2000/X64.

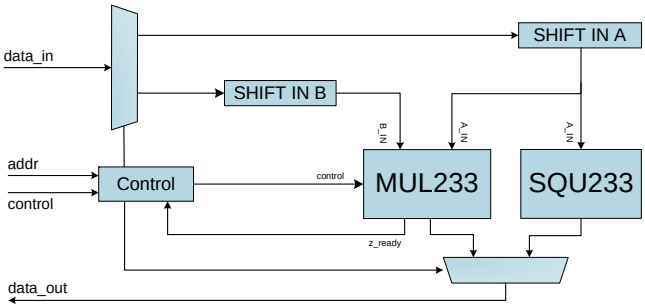

Fig. 9. Hardware accelerator for the scalar multiplication.

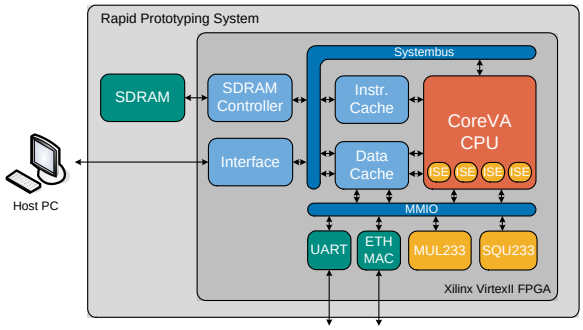

Fig. 10. Evaluation environment for the CoreVA processor using RAPTOR2000/X64.

32-bit word. The result can be read in two clock cycles, because of the latency of the load-word instruction. In the following, three implementations of the CoreVA processor are analyzed:

- *CoreVA(SQU233)* includes only one accelerator for field squaring
- *CoreVA(MUL233)* includes only one accelerator for field multiplication
- *CoreVA(FF233)* includes both accelerators for finite field arithmetic

4.1 Functional verification

The hardware extensions were functionally verified using our rapid prototyping environment RAPTOR2000/X64 (see Sect. 3).

The CoreVA processor was emulated on a Xilinx Virtex-2 6000 FPGA. Maximum clock frequency is 25 MHz. 32 KB on-chip-memory is used for the L1-Cache (instructions and data). Up to 4GB SDRAM can be used for memory intensive applications. Additionally, a 100 MBit/s ethernet interface is used to supply realistic input data. The CoreVA processor can be controlled by a graphical user interface. Via the RAPTOR interface (cf. Fig. 10) the on-chip memory and SDRAM can be accessed. The processor can run in full speed or in step-by-step mode to apply our *Validation by Simulation* methodology of Jungeblut et al. (2007).

4.2 Cost and performance analysis

After the functional verification the enhanced architecture was mapped to a 65 nm low power standard cell technology to analyze the area and power requirements. Table 1 shows the resource consumption of all four types of hardware extensions. The implementation achieves a clock frequency of 200 MHz and is based on a 65nm low power standard cell CMOS technology at worst case operating conditions.

As the optimization during synthesis is a non-deterministic process, utilization of standard cells can vary and the chip area of some components can even decrease, which is the case for the ISEs. The dedicated hardware extensions require an acceptable increase of area requirements (27.71% for MUL233, 1.92% for SQU233, cf. Fig. 11). The combination of both hardware accelerators leads to an impact of 29.87% compared to the original implementation. Figure 12 depicts the power consumption of the four optimized CoreVA implementations at 200 MHz. Again, the implementations using dedicated hardware accelerators cause the highest power consumption, whereas the instruction set extensions even cause a slightly lower switching activity.

Table 1. Resource consumption of different hardware configurations.

	Frequency [MHz]	Area [µm^2]	Relative [%]	Power [mW]	Relative [%]
CoreVA	200	237 019	100.00%	5.81	100.00%
CoreVA(ISE)	200	236 704	99.87%	5.80	99.83%
CoreVA(SQU233)	200	241 577	101.92%	6.07	104.48%
CoreVA(MUL233)	200	302 714	127.72%	7.30	125.65%
CoreVA(FF233)	200	307 825	129.87%	7.53	129.60%

Table 2. Execution cycles using different types of hardware acceleration for finite field arithmetic of characteristic 2.

	Scalar multiplication		Field multiplication		Field squaring		Word multiplication	
	Clock cycles	Relative [%]	Clock cycles	Relative [%]	Clock cycles	Relative [%]	Clock cycles	Relative [%]
CoreVA(C)	3 552 571	100.00	2111	100.00	353	100.00	44	100.00
CoreVA(C.ISE)	3408875	95.96	2018	95.59	344	97.45	40	90.91
CoreVA(ASM)	2 941 035	82.79	1839	87.12	185	52.41	33	75.00
CoreVA(ASM.ISE)	2 622 409	73.82	1636	77.50	162	45.89	26	59.09
CoreVA(SQU233)	3 129 187	88.08	2111	100.00	49	13.88	44	100.00
CoreVA(MUL233)	684 999	19.28	73	3.46	353	100.00	44	100.00
CoreVA(FF233)	260 615	7.34	73	3.46	49	13.88	44	100.00

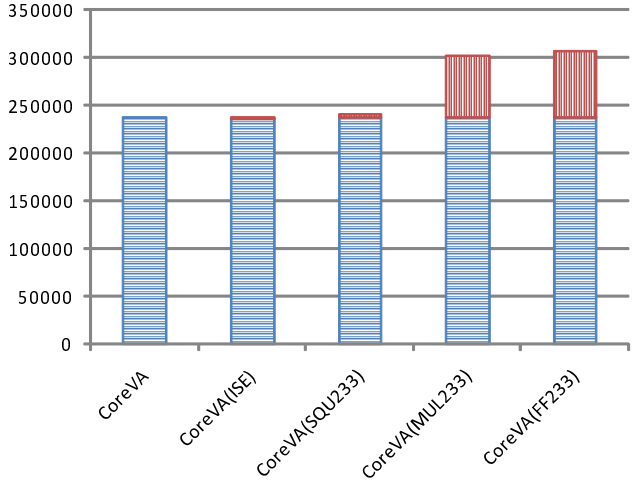

Fig. 11. Area requirements of hardware configurations [µm^2].

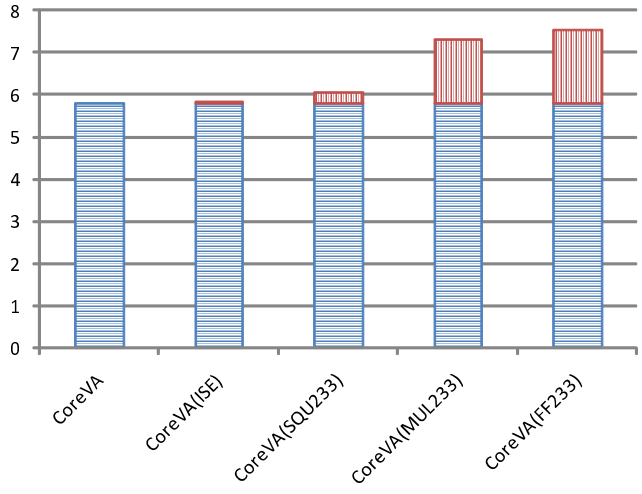

Fig. 12. Power dissipation of different hardware configurations [mW].

The execution time in terms of clock cycles is shown in Table 2. Without any additional hardware effort for the instruction set extensions, the performance of ASM word level multiplication is increased by 27%, ASM field multiplication by 12%, and ASM field squaring by 14% (C implementation: 10%, 5%, 3%). The scalar multiplication (ASM) is sped up by about 12% (C implementation: 4%). Using the *CoreVA(MUL233)* hardware extension, the field multiplication is computed in 73 clock cycles, which is equivalent to a speedup of 25 compared to the ASM implementation (C implementation: 29). Hardware acceleration with the *CoreVA(SQU233)* implementation reduces calculation time of the field squaring from 353 clock cycles (C), respectively 185 (ASM) to 49 clock cycles (speedup of 7.2 respectively 3.8). The last hardware extension combines both dedicated hardware extensions as one module, which reduces load/store operations to transfer input- and output-data. Using this implementation, we achieve a speedup of about 13.6 for the scalar multiplication compared to the soft-

Table 3. Code size using different types of hardware acceleration for finite field arithmetic of characteristic 2.

	Scalar multiplication		Field multiplication		Field squaring		Word multiplication	
	Code size [bytes]	Relative [%]	Code size [bytes]	Relative [%]	Code size [bytes]	Relative [%]	Code size [bytes]	Relative [%]
CoreVA(C)	10 336	100.00	2052	100.00	528	100.00	448	100.00
CoreVA(C.ISE)	8656	83.75	1940	94.54	496	93.94	336	75.00
CoreVA(ASM)	8752	84.67	2068	100.78	464	87.88	464	103.57
CoreVA(ASM.ISE)	8576	82.97	1924	93.76	432	81.82	320	71.43
CoreVA(SQU233)	8496	82.20	2052	100.00	208	39.39	448	100.00
CoreVA(MUL233)	6988	67.61	304	14.81	528	100.00	448	100.00
CoreVA(FF233)	6732	65.13	2052	100.00	208	39.39	448	100.00

ware C implementation (speedup of 11.2 compared to ASM software implementation). Compared to the hardware extended processor of Puttmann et al. (2008) this is equivalent to a speedup of 15.

Besides optimizing execution time by using hardware acceleration for the ECC algorithm, code size is also reduced significantly. Code size impacts the amount of memory needed, which is an expensive resource in embedded systems. Table 3 shows the code size of the word and field multiplication, the field squaring and the scalar multiplication using different types of hardware acceleration. Compared to the initial C implementation, code size of the scalar multiplication is reduced from 10 336 bytes to 8656 bytes using instruction set extensions. Using the *CoreVA(FF233)* dedicated hardware accelerator this effect is even more significant: 35% of instruction memory can be saved in this configuration. Power consumption depends on the memory size as well as the number of accesses, so resource efficiency can be increased by optimizing code size. As the total code size, which depends on all applications (plus operating system, etc.) can vary extremely in different environments and because the type of memory (single port or dual port) highly effects power consumption, we only consider the energy demand of processor architecture in this paper. Besides total power consumption, energy is an even more important quality measure for mobile devices like smart cards.

Despite the higher power consumption caused by the hardware extensions, energy will be decreased because of the reduced execution time. Figures 13–16 and Table 4 show the energy consumption of the original software implementations compared to the hardware accelerated versions. The darker bars show the energy consumption of the C implementation, the brighter ones reflect the assembler implementation. Control of the hardware accelerators is only implemented at C-level. Energy consumption of word level multiplication can be reduced by using the instruction set extensions. As the dedicated hardware accelerators are only applicable for field arithmetic, energy consumption increases. At field squaring level the dedicated *CoreVA(SQU233)* hardware accelerator causes the greatest improvement.

At field level, the dedicated *CoreVA(MUL233)* hardware accelerator causes the greatest improvement. Energy consumption for field multiplication reduces from 61 nJ to less than 2.7 nJ (96%) (cf. Fig. 14). Obviously, *CoreVA(SQU233)* can not be used for the field multiplication, so energy consumption increases because of the additional hardware integrated to the processor. Energy consumption of field squaring benefits especially from the ASM optimized algorithm in conjunction with the *CoreVA(ISE)* instruction set extensions. An energy reduction of about 50% can be achieved (cf. Fig 15). Using the *CoreVA(SQU233)* hardware extension reduces energy consumption of field squaring by a factor of 6.9. Again, energy consumption increases when using *CoreVA(MUL233)*, as it is not applicable to field squaring. Figure 16 shows the energy consumption for the scalar multiplication. Energy savings for the scalar multiplication range from 26% (ASM, *CoreVA(ISE)*) up to 90% for the *CoreVA(FF233)* hardware acceleration. Energy demand of the ASM implementation using *CoreVA(ISE)* instruction set extensions for both field multiplication and field squaring is even lower as using only the *CoreVA(SQU233)* hardware accelerator, which only impacts execution cycles of field squaring *CoreVA(SQU233)* only affects execution cycles of field squaring. As field multiplication is the domi-

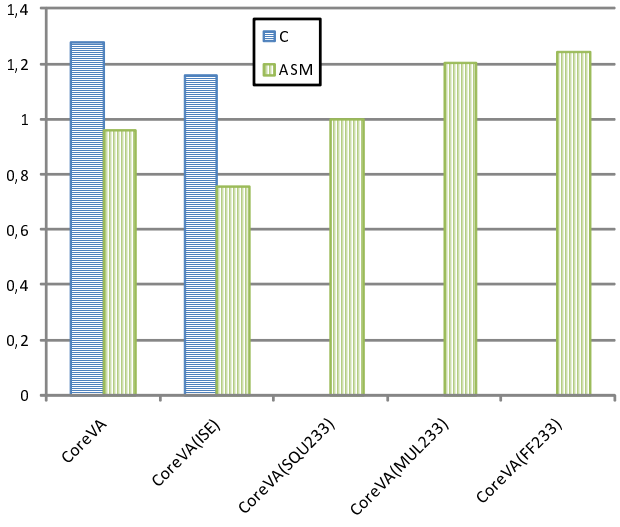

Fig. 13. Energy consumption of the word multiplication [mW].

Table 4. Energy consumption using different types of hardware acceleration.

	Scalar multiplication		Field multiplication		Field squaring		Word multiplication	
	Energy [nJ]	Relative [%]	Energy [nJ]	Relative [%]	Energy [nJ]	Relative [%]	Energy [nJ]	Relative [%]
CoreVA(C)	103 202.19	100.00	61.32	100.00	10.25	100.00	1.281	00.00
CoreVA(C.ISE)	98 857.38	95.79	58.52	95.43	9.98	97.28	1.16	90.75
CoreVA(ASM)	85 437.07	82.79	53.42	87.12	5.37	52.41	0.96	75.00
CoreVA(ASM.ISE)	76 049.86	73.69	47.44	77.37	4.70	45.81	0.75	58.99
CoreVA(SQU233)	94 970.83	92.02	67.12	109.46	1.79	17.44	1.20	94.23
CoreVA(MUL233)	20 789.72	20.14	2.22	3.61	5.61	54.75	1.00	78.36
CoreVA(FF233)	9812.15	9.51	2.75	4.48	1.84	17.99	1.24	97.20

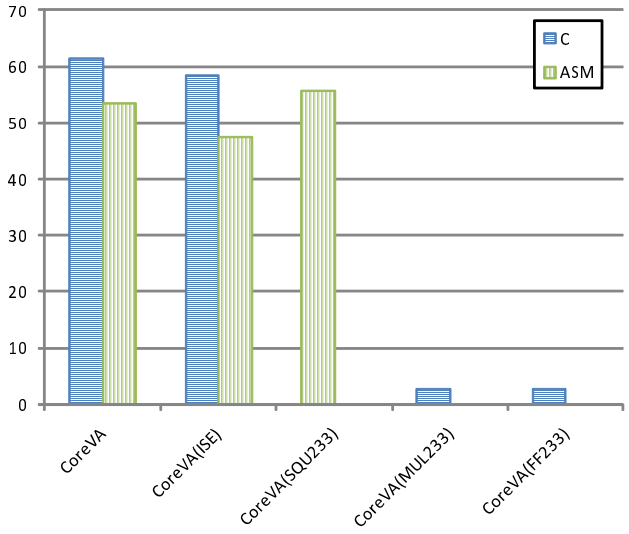

Fig. 14. Energy consumption of the field multiplication [mW].

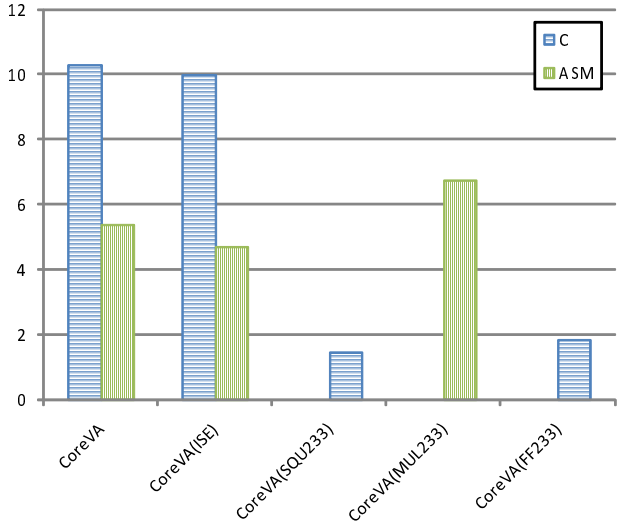

Fig. 15. Energy consumption of the field squaring [mW].

nating part of scalar multiplication energy consumption increases. Only the combination of both hardware extensions (*CoreVA(FF233)*) can improve energy efficiency for this algorithm.

5 Conclusions

We analyzed different types of hardware extensions in respect to resource efficiency, which we consider to be the ratio of performance in terms of execution time (clock cycles) and chip area and power consumption. These constituent characteristics have been presented in detail, whereas the resource efficiency itself should be computed in respect to the envisioned application scenario. We outlined our dual design-flow, consisting of the automatic generation of a complete Ansi-C-compiler tool chain based on a high level reference specification in the UPSLA language and a common RTL-based hardware development. Using this design-flow, we introduced three instruction set extensions and two dedicated hardware accelerators. Major components of the scalar multiplication over binary fields $\mathbb{F}_{2^{233}}$ were considered, starting from word multiplication up to field multiplication and field squaring. The tightly coupled instruction set extensions combine existing functional units, so resource consumption is negligible. The integration of loosely coupled hardware accelerators adds still reasonable resource overhead but results in an even higher speedup. In this case, area and power consumption increase by about 30%, whereas the execution time of the scalar multiplication decreases by 93% compared to the original software implementation. Both types of hardware extensions achieve an energy reduction, ranging from 27% to 90%.

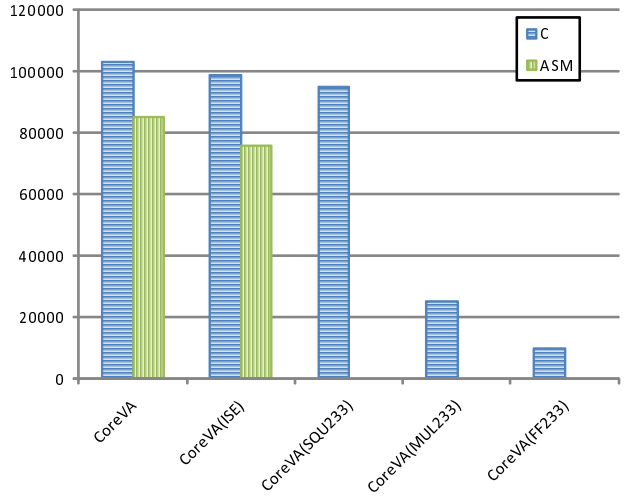

Fig. 16. Energy requirements of the scalar multiplication [mW].

Acknowledgements. Substantial parts of the research described in this paper were funded by the Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung – BMBF) registered there under grant numbers 01BU0661 (MxMobile) and 01BU0643 (Easy-C).

References

Gonzalez, R. E.: Xtensa: A Configurable and Extensible Processor, IEEE Micro, 20, 60–70, 2000.

Großschädl, J. and Kamendje, G.-A.: Instruction Set Extension for Fast Elliptic Curve Cryptography over Binary Finite Fields GF(2^m), in: Proceedings of the 14th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2003), edited by: Deprettere, E., Bhattacharyya, S., Cavallaro, J., Darte, A., and Thiele, L., 455–468, IEEE Computer Society Press, 2003.

Großschädl, J. and Savaş, E.: Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m), in: Cryptographic Hardware and Embedded Systems — CHES 2004, edited by Joye, M. and Quisquater, J.-J., vol. 3156 of Lecture Notes in Computer Science, 133–147, Springer Verlag, 2004.

Hankerson, D., Menezes, A. J., and Vanstone, S. A.: Guide to Elliptic Curve Cryptography. (Springer Professional Computing), Springer, Berlin, 2004.

Jungeblut, T., Dreesen, R., Porrmann, M., Rückert, U., and Hachmann, U.: Design Space Exploration for Resource Efficient VLIW-Processors, in: University Booth of the Design, Automation and Test in Europe (DATE) conference, Munich, Germany, 2007.

Karatsuba, A. A. and Ofman, Y.: Multiplication of multidigit numbers on automata, Soviet Physics Doklady, 7, 595–596, 1963.

Kastens, U., Le, D. K., Slowik, A., and Thies, M.: Feedback Driven Instruction-Set Extension, in: Proceedings of ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04), Washington, D.C., USA, 2004.

Knuth, D. E.: The Art of Computer Programming, vol. 2, Seminumerical Algorithms, Reading MA, 3rd edn., first edition 1969, 1998.

Kumar, S. S. and Paar, C.: Reconfigurable Instruction Set Extension for Enabling ECC on an 8-Bit Processor., in: Field Programmable Logic and Application, 14th International Conference , FPL 2004, Leuven, Belgium, 30 August–1 September 2004, Proceedings, vol. 3203 of Lecture Notes in Computer Science, 586–595, Springer, 2004.

López, J. and Dahab, R.: Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation, in: CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems, 316–327, Springer-Verlag, London, UK, 1999.

Montgomery, P. L.: Speeding the Pollard and elliptic curve methods of factorization, Mathematics of Computation, 48, 243–264, 1987.

National Institute of Standards and Technology (NIST): Digital Signature Standard (DSS), vol. FIPS 186-2, chap. Recommended elliptic curves for federal government use, 24–48, US Department Of Commerce, 2000.

Porrmann, M., Hagemeyer, J., Romoth, J., and Strugholtz, M.: Rapid Prototyping of Next-Generation Multiprocessor SoCs, in: In Proceedings of Semiconductor Conference Dresden, SCD 2009, 29–30, Dresden, Germany, 2009.

Puttmann, C., Shokrollahi, J., and Porrmann, M.: Resource Efficiency of Instruction Set Extensions for Elliptic Curve Cryptography, in: ITNG '08: Proceedings of the Fifth International Conference on Information Technology: New Generations, 131–136, IEEE Computer Society, Washington, DC, USA, 2008.

Tillich, S. and Großschädl, J.: A Simple Architectural Enhancement for Fast and Flexible Elliptic Curve Cryptography over Binary Finite Fields GF(2^m), in: Advances in Computer Systems Architecture, Lecture Notes in Computer Science, 282–295, Springer Verlag, 2004.

von zur Gathen, J. and Gerhard, J.: Modern Computer Algebra, Cambridge, UK, second edition, 2003.