Christoph Puttmann & Jamshid Shokrollahi (2007). MPSoC-coupled Hardware Accelerator for Ellipic Curve Cryptography. In *Proceedings of the Western European Workshop on Research in Cryptology (WEWoRC'07)*, Stefan Lucks, Ahmad Reza Sadeghi & Christopher Wolf, editors, 1–10. Bochum. URL http://2007.weworc.org/index.html.

Resource Efficiency of Instruction Set Extensions for Elliptic Curve Cryptography

Christoph Puttmann[1] and Jamshid Shokrollahi[2]

[1] Heinz Nixdorf Institute, University of Paderborn, Germany
puttmann@hni.upb.de
[2] B-IT, University of Bonn, Germany
jamshid@bit.uni-bonn.de

Abstract. In this paper we present a holistic methodology for automated evaluation of instruction set extensions. We propose a two-stage framework for analyzing the resource efficiency of extending an instruction set. With emphasis to elliptic curve cryptography, several instruction set extensions are implemented to a 32 bit RISC microprocessor and synthesized in a state of the art 65 nm standard cell CMOS technology. The achieved performance improvement is analyzed in respect to the hardware costs in terms of chip area and power consumption. In order to accelerate algorithms for elliptic curve cryptography, we focus on the optimization of finite field multiplication. Our approach is based on the Karatsuba algorithm over binary fields $GF(2^m)$.

Keywords: instruction set extension, elliptic curve cryptography, binary field multiplication, resource efficiency

1 Introduction

Finite field arithmetic is a very important issue in public key cryptography and especially in elliptic curve cryptography (ECC). Among other operations in finite field arithmetic, multiplication is the most important one. The reason for this is, because multiplication is most resource consuming operation and is required very often in cryptographic algorithms. The finite fields that are used in cryptography are generally either prime finite fields or binary fields, which are of the form $\mathbb{F}_{2^n}$ and can be represented as vector spaces over $\mathbb{F}_2$ using several possible bases. The cost of finite field arithmetic depends on the selected basis. There are two popular bases, namely normal and polynomial bases. On the one hand, normal basis representation has the advantage that squaring can be done using only a circular shift. On the other hand, this representation has a relatively expensive multiplication cost and hence is used for applications, which need more squaring operations than multiplications, e.g., exponentiation in finite fields. Although squaring is slightly more expensive than in normal basis, in elliptic curve cryptography generally polynomial bases are used, because multiplication is much easier than in other representations. Multiplication

in polynomial basis consists of a polynomial multiplication followed by a modular reduction. The latter can be done very efficiently, when 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.

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 32 bit RISC processor, the N-Core [1], in order to accelerate the performance of elliptic curve cryptography on binary finite fields. More precisely, we evaluate instruction set extensions in respect to their resource efficiency. For this purpose, we present a two-stage framework with automated workflow for instruction set extension (ISE).

Our analysis is based on the multiplication of polynomials of degree 233 over $\mathbb{F}_2$. Among others, the finite field $\mathbb{F}_{2^{233}}$ has been suggested for ECC [2] by the National Institute of Standards and Technology (NIST). There are several proposals in the literature for multiplication of two polynomials (see [3] or [4] 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 $n = 232$ and has a cost of $O(n^{log_2 3})$. There also exists 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. Up to our knowledge, this is the first work that studies the Karatsuba method in respect to resource efficiency of hardware modifications based on instruction set extensions.

The paper is organized as follows. Section 1.1 is devoted to the discussion of different works in the literature, which concern ISE for finite field arithmetic and cryptography. In Section 2 we review polynomial multiplication methods and their implementation aspects. Section 3 describes the hardware architecture of the N-Core processor and introduces our automated workflow for ISE. Implementation issues of the instructions set extensions and analysis of their resource efficiency is presented in Section 4. Finally, Section 5 concludes the paper.

1.1 Related Work

There are several papers about instruction set extensions and their applications in elliptic curve cryptography over binary finite fields. The authors of [5] 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 [6] a dual field adder (DFA) module is implemented into 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 [7] 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, [8] discusses 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 [9] also the automatically generation of instruction set extension is considered. In our work, we consider the increase in performance of the Karatsuba method by means of integrating both, application-specific and automatically generated instruction set extensions, to the processor and analyze the resource efficiency. In this context, resource efficiency considers not only the execution time in terms of clock cycles, but also the hardware costs in terms of chip area and power consumption. Furthermore, the impact of code size reduction through instruction set extension is evaluated.

2 Polynomial Arithmetic

Having a compact representation for data structures will not only help in saving memory, but also can reduce operation time, since on most processors accessing the memory is one of the most time consuming operations. On w-bit processors, polynomials over $\mathbb{F}_2$ are saved in a 2^w-ary representation, i.e. ,

$$a(x) = \sum_{i=0}^{n-1} a_i x^i = \sum_{j=0}^{m} A_j x^{wj} = \sum_{j=0}^{m} X^j, \tag{1}$$

where $m = \lceil n/d \rceil$, $X = x^w$, and A_j, for $0 \leq j < m$, are polynomials of degree $w-1$ over $\mathbb{F}_2$.

Since we use a 32 bit RISC processor, in our case we choose $w = 32$. To multiply two w-bit polynomials, there are several algorithm, e.g. , the *shift-and-xor* method,

known in literature [10]. However, we use the optimized algorithm from the number theory library (NTL) for the 32 bit multiplication at word-level (see [11]). In the 32 bit method that is shown in Algorithm 1 the size of the lookup-table `A` can be adjusted according to the number of registers or cache. In addition, the loop containing Lines 8 to 18 has been unrolled for efficiency reasons.

Algorithm 1 word-level multiplication

Input: Two 32 bit words `a` and `b`, containing input polynomials of degree at most 31
Output: The 64 bit product `c`

1: $\texttt{A}[0] \leftarrow 0$
2: $\texttt{A}[1] \leftarrow \texttt{a}$
3: $\texttt{A}[2] \leftarrow \texttt{a} \ll 1$
4: $\texttt{A}[3] \leftarrow \texttt{A}[2] \oplus \texttt{a}$
5: $\texttt{A}[4] \leftarrow \texttt{A}[2] \ll 1$
6: $\texttt{A}[5] \leftarrow \texttt{A}[4] \oplus \texttt{a}$
7: $\texttt{A}[6] \leftarrow \texttt{A}[4] \oplus \texttt{A}[2]$
8: $\texttt{A}[7] \leftarrow \texttt{A}[6] \oplus \texttt{a}$ ▷ The array `A` contains $p(x)\texttt{a}$ for $p(x) \in \mathbb{F}_2[x]$ from 0 to $x^2 + x + 1$
9: $\texttt{c} \leftarrow \texttt{A}[\texttt{b} \ \& \ \texttt{0x0007}]$
10: $\texttt{b} \leftarrow \texttt{b} \gg 3$
11: $\texttt{c} \leftarrow \texttt{c} \oplus \texttt{A}[\texttt{b} \ \& \ \texttt{0x0007}]$
12: $\texttt{b} \leftarrow \texttt{b} \gg 3$
13: $\texttt{c} \leftarrow \texttt{c} \oplus \texttt{A}[\texttt{b} \ \& \ \texttt{0x0007}]$
14: $\texttt{b} \leftarrow \texttt{b} \gg 3$
15: $\texttt{c} \leftarrow \texttt{c} \oplus \texttt{A}[\texttt{b} \ \& \ \texttt{0x0007}]$
16: $\texttt{b} \leftarrow \texttt{b} \gg 3$
17: $\texttt{c} \leftarrow \texttt{c} \oplus \texttt{A}[\texttt{b} \ \& \ \texttt{0x0007}]$
18: $\texttt{b} \leftarrow \texttt{b} \gg 3$
19: $\texttt{c} \leftarrow \texttt{c} \oplus \texttt{A}[\texttt{b}]$

As you will see in Section 4, Algorithm 1 turned out to be very efficient on our processor for word-level multiplication of binary finite fields. The field multiplication in $\mathbb{F}_{2^{233}}$ is realized by extending Algorithm 1 using the Karatsuba method. This means, we multiply two polynomials $A(X) = \sum_{j=0}^{m-1} A_j X^j$ and $B(X) = \sum_{j=0}^{m-1} B_j X^j$ in which A_j and B_j are polynomials of degree at most 31. Originally, the Karatsuba method was introduced for multiplication of long integers by [12]. The three coefficients of the product $(A_1X + A_0)(B_1x + B_0) = A_1B_1X^2 + (A_1B_0 + A_0B_1)X + A_0B_0$ are "classically" computed with 4 multiplications and 1 addition from the four input coefficients A_1, A_0, B_1, and B_0. The Karatsuba method uses only 3 multiplications and 4 additions by applying the following formula:

$$(A_1X + A_0)(B_1X + B_0) = A_1B_1X^2 + ((A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0)x + A_0B_0. \quad (2)$$

Since in binary field arithmetics a multiplication needs much more resources than an addition, which equals a *XOR* operation, the Karatsuba method reduces the overall complexity. In our case, where the polynomial degree m is large, each of the polynomials A_0, A_1, B_0, and B_1 can be chosen as polynomials of degree $\lceil m/2 \rceil$. This process can be repeated until each of the coefficients is a polynomial of degree 31. Recursive application of the Karatsuba method decreases the number of operations from $O(m^2)$ of the classical method to $O(m^{\log_2 3})$.

3 Processor Architecture and ISE Framework

The N-Core represents a 32 bit RISC microprocessor and is specified at register transfer level (RTL) in the hardware description language VHDL [1]. The typical load-store architecture of the processor (cf. Fig. 1) provides a three-stage pipeline that comprises the operations *fetch*, *decode* and *execute*. The N-Core features two register banks containing 16×32 bit registers each. The additional register bank allows the N-Core immediate thread switching without storing the register content, e.g., for fast interrupt handling. In contrast to the 32 bit wide data path, all instruction words of the processor element have a fix length of 16 bit. Therefore, a high code density is reached and instruction memory can be reduced. Not only efficient memory utilization, but also a small size in terms of chip area and low power consumption (see Section 4) make the N-Core particularly suitable for embedded systems with limited resources, e.g., smartcards.

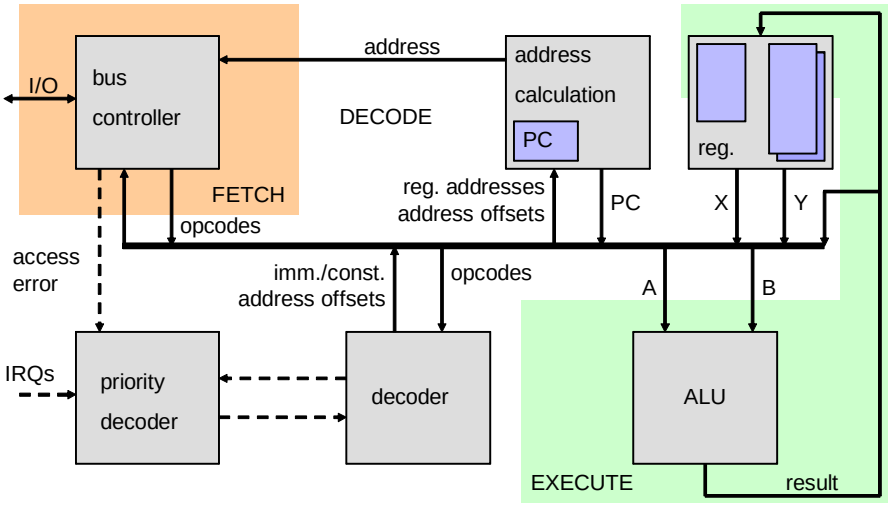

Fig. 1. N-Core architecture featuring a three-stage pipeline.

Although the N-Core uses 16 bit instruction words, there are still 11 % of free opcode space left. This allows us to implement additional instructions in order to accelerate a software task. We propose a two-stage workflow for instruction set extension as depicted in Fig. 2.

The first stage is called compiler-related workflow. At this stage, we use a formal model of the N-Core processor, which is described in the Unified Processor Specification Language (UPSLA). Based on this formal processor specification a corresponding ANSI-C compiler as well as a cycle accurate simulator can be automatically generated [13]. Furthermore, the simulator features powerful profiling capabilities that allow a detailed analysis of instruction distribution and memory access. In this way, a software application can easily be analyzed in respect to frequently executed instruction sequences. After the software profiling, selected instruction sequences can be combined to so called *super instructions*. Instead of executing an instruction sequence of two (*instruction pair*) or more instructions (*instruction block*), the super instruction completes the same operation in less clock cycles. In this way, not only instruction memory space is saved, but also the software application is accelerated. After the new super instruction is specified in the formal model of the processor, an adapted compiler and simulator can be generated on the fly. Recompiling the application program automatically applies the new super instructions. To analyze the performance improvement in terms of execution cycles, the software can be again simulated and profiled, respectively.

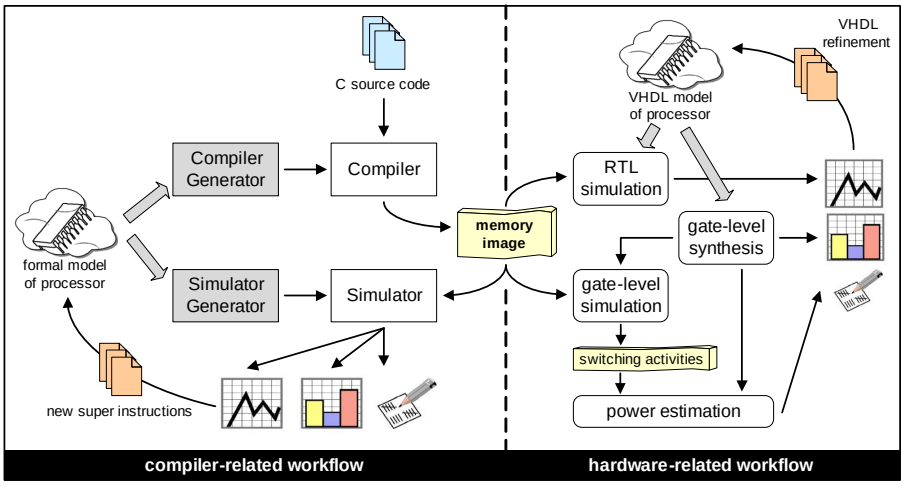

Fig. 2. Automated two-stage framework of instruction set extension.

After the instruction set has been optimized, we move to the hardware-related workflow in order to analyze the chip area and power consumption. Therefore, we have to implement the specified super instructions at register transfer level of our VHDL processor model. The compiled memory image is reused for RTL simulation to verify the functionality of the adapted processor. Chip area and power estimation are evaluated on gate-level with back annotation of switching activities, respectively.

4 Implementation and Analysis

Since the multiplication represents the most time consuming part of ECC methods, the evaluation of our ISE framework is based on the finite field multiplication in binary fields (cf. Section 2). Therefore, we start our analysis with a software implementation of Algorithm 1 that multiplies two 32 bit binary field polynomials. By applying our ISE-framework (cf. Fig. 2) we easily obtain detailed profiling information of the multiplication algorithm. After analyzing the distribution of instruction pairs, we choose to implement the new super instructions `LSRIxor`, `LSLIxor` and `LSRIandadd`. Due to the nature of Algorithm 1, the combination of shifting (*Logical Shift Right/Left Immediate*) and *XOR* operation is quite obvious for super instructions. The operation `LSRIandadd` combines even three regular instructions to one super instruction. Hence, the following operations are executed in one instead of three clock cycles: The content of register X is shifted to the right by the specified immediate value. Afterwards, a logical *AND* operation with the immediate value 28 is performed. Finally, the value of the *stack pointer* is added (*ADD*) and the result is written back to register X. This super instruction is frequently used in the multiplication algorithm for calculating address offsets. In the following, the N-Core processor that contains the three new super instructions is referred to as *ISE 1*.

Whereas ISE 1 only combines existing instructions to super instructions, the next variant of instruction set extension (referred to as *ISE 1*) introduces a new instruction, called `MULGF2`. This instruction multiplies two 32 bit binary field polynomials by using existing functions of the arithmetic logical unit (ALU). Multiplication with ISE 2 is realized bit by bit utilizing the shift unit and xor function of the ALU and can be compared to `MULGFS` instruction in [5].

The third type of instruction set extension (ISE 3) also supports the `MULGF2` instruction, but a dedicated hardware unit for the binary field multiplication is implemented instead of using existing ALU functions for the multiplication. Therefore, not only the instruction decoder is modified, but also the ALU has to be extended.

Table 1 shows the synthesis results for all three versions of instruction set extensions. The synthesis results are constraint for a clock frequency of 200 MHz and based on a 65 nm standard cell CMOS technology at typical operating conditions.

Table 1. Absolute values and relative increase of chip area based on 65 nm standard cell technology.

	total processor area		instruction decoder		arithmetic logical unit	
	absolute [µm²]	relative [%]	absolute [µm²]	relative [%]	absolute [µm²]	relative [%]
N-Core	57,189.96	0.00	3502.80	0.00	16,439.76	0.00
N-Core (ISE 1)	57,518.64	0.57	3507.84	0.14	16,137.01	-1.88
N-Core (ISE 2)	59,289.84	3.54	3520.08	0.49	18,260.64	9.97
N-Core (ISE 3)	64,734.48	11.65	3394.80	-3.18	22,924.08	28.29

Since using a top down design flow, the critical path may change due to our hardware modifications. As a result, the chip area of some components can also decrease, which is the case for ALU of ISE 1 and for the instruction decoder of ISE 3. Because already existing instructions are combined for ISE 1, the chip area increases slightly about 0.6 % in total. Also ISE 2 requires only a medium increase (3.5 %) of chip size, which corresponds to chip costs, respectively. Therefore, the worst impact on chip cost is observed by ISE 3, which is due to the dedicated hardware multiplier inside the ALU.

Figure 3 depicts the power consumption of the main N-Core components for a word-level multiplication at 200 MHz clock frequency. Again, ISE 3 requires the most resources and consumes nearly 20 % more power than the original N-Core, which is due to the high switching activity of the dedicated hardware multiplier. In contrast, ISE 2 even saves power compared to all other implementations, because

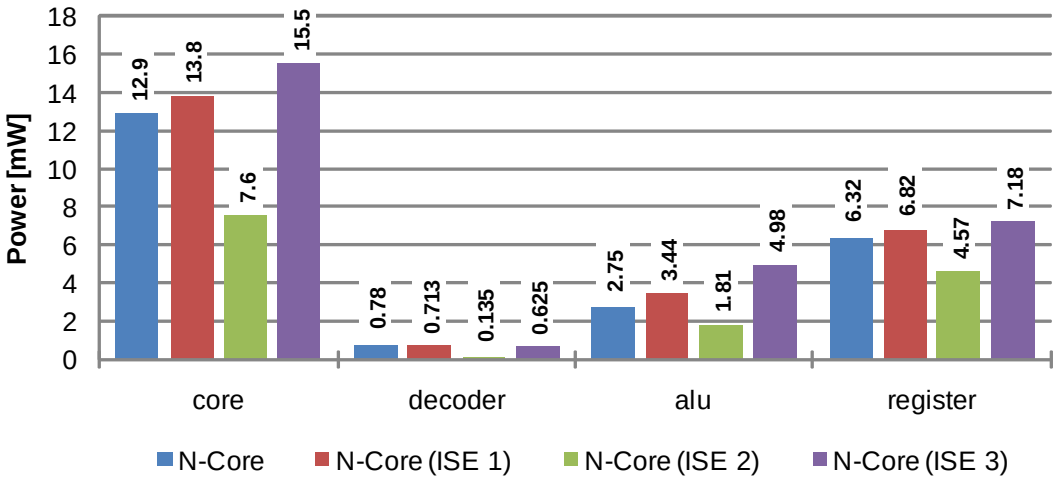

Fig. 3. Power consumption of main N-Core components for word-level multiplication at 200 MHz.

Table 2. Execution time in terms of clock cycles and code size for word and field multiplication.

	word multiplication		field multiplication	
	clock cycles	code size	clock cycles	code size
N-Core	166	268 byte	6498	956 byte
N-Core (ISE 1)	130	196 byte	5545	880 byte
N-Core (ISE 2)	36	2 byte	2060	576 byte
N-Core (ISE 3)	2	2 byte	1169	576 byte

mainly the shift and xor unit are used, which operate very power efficient. Using the super instructions of ISE 1 results in a slightly higher power consumption of roughly 7 % compared to the original N-Core.

The execution time in terms of clock cycles as well as the code size is shown in Table 2. A word-level multiplication using Algorithm 1 takes 166 clock cycles at the unmodified N-Core and requires 268 byte of instruction memory. The field multiplication over finite binary field $\mathbb{F}_{2^{233}}$ using the Karatsuba method consumes 6498 clock cycles and 956 byte of memory, respectively. With little effort of instruction combining (ISE 1) a speedup of 1.28 can be achieved for word-level multiplication and a speedup of 1.17 for the field multiplication, respectively. For calculating the word-level product bit by bit using ISE 2, a speedup of 4.61 is achieved. The Karatsuba method for field multiplication can be accelerated by factor 3.15 by applying ISE 2. The dedicated hardware multiplier requires only 2 clock cycles for a word multiplication, which equals a speedup of 83.0 or 5.56 for field multiplication, respectively. Because ISE 2 and ISE 3 use the `MULGF2` instruction, the code size for word-level multiplication equals 2 byte. As a result, nearly 40 % of instruction memory can be saved for the field multiplication algorithm, when applying ISE 2 or ISE 3.

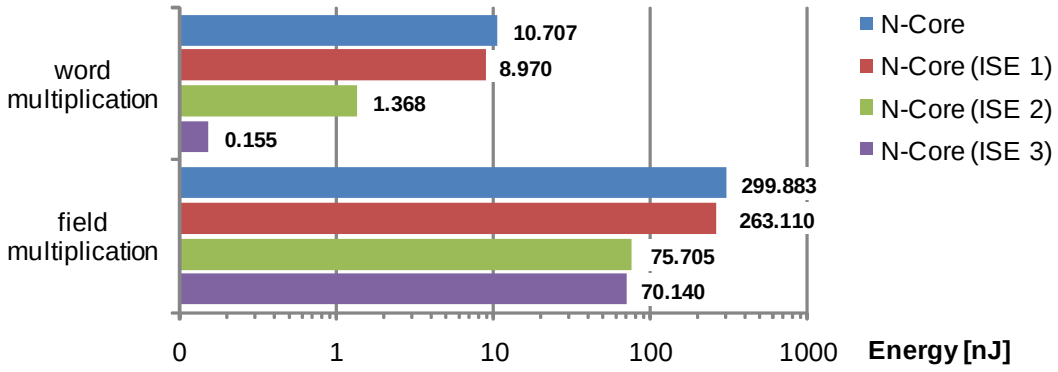

Fig. 4. Energy consumption for word and field multiplication at 200 MHz clock frequency.

Figure 4 shows the energy consumption for word-level and binary field multiplication at 200 MHz clock frequency. Whereas ISE 1 saves approximately 15 % of energy when calculating either a word-level multiplication or binary field multiplication, the impact of ISE 2 and ISE 3 is more significant. Because ISE 3 comprises the binary field hardware multiplier, a word-level multiplication is done in 2 clock cycles and therefore consumes only about 0,16 nJ of energy at 200 MHz clock frequency. At word-level ISE 2 consumes almost 9 times more energy for the multiplication, which is still only 13 % of the energy consumption of the original N-Core. In contrast, ISE 3 requires nearly the same energy as ISE 2 for a field multiplication over $\mathbb{F}_{2^{233}}$. This is due to the overhead of instructions other than multiplication, which do not benefit from the dedicated hardware multiplier.

5 Conclusion

In this paper, we have analyzed different types of instruction set extension in respect to their resource efficiency. In this context, we have considered resource efficiency as a trade-off between performance in terms of execution time (clock cycles) and hardware costs by means of chip area and power consumption. Furthermore, we have proposed an two-stage framework that features an automated workflow for ISE. Using our ISE framework, we have introduced three variants of ISE for accelerating ECC algorithms. Thereby, we have concentrated on binary field multiplication over $\mathbb{F}_{2^{233}}$. We have started with word-level multiplication of 32 bit polynomials, which were extended to field multiplication by applying the Karatsuba method.

The analysis of combining instructions (ISE 1), introducing new instructions that use existing function units (ISE 2) and implementing dedicated hardware for binary field multiplication (ISE 3) supplies a detailed overview of pros and cons of different types of instruction set extension. In respect to resource efficiency, ISE 2 provides the best trade-off between performance improvement and hardware costs. While the chip area increases only slightly (3.6 %), a speedup of roughly 4.6 is achieved for word-level multiplication. By applying the Karatsuba method, ISE 2 leads to an overall speedup of nearly 3.2 for a binary finite field multiplication over $\mathbb{F}_{2^{233}}$. Furthermore, ISE 2 saves approximately 75 % of energy compared to the original software implementation running on the N-Core.

In our future work, we will evaluate additional ISE for scalar multiplication in binary finite fields. Especially the modular reduction and squaring operation are promising further speedup possibilities. Moreover, we want to analyze hardware accelerator units that are externally coupled to the processor.

References

1. Langen, D., Niemann, J.C., Porrmann, M., Kalte, H., Rückert, U.: Implementation of a risc processor core for soc designs - fpga prototype vs. asic implementation. In: Proceedings of the IEEE-Workshop: Heterogeneous reconfigurable Systems on Chip (SoC), Hamburg, Germany (2002)
2. National Institute of Standards and Technology (NIST): Recommended elliptic curves for federal government use. In: Digital Signature Standard (DSS). Volume FIPS 186-2. U.S. Department Of Commerce (2000) 24–48
3. Knuth, D.E.: The Art of Computer Programming, vol. 2, Seminumerical Algorithms. 3rd edn., Reading MA (1998) First edition 1969.
4. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Second edn., Cambridge, UK (2003) First edition 1999.
5. Tillich, S., 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, Springer Verlag (2004) 282–295
6. Großschädl, J., Kamendje, G.A.: Instruction set extension for fast elliptic curve cryptography over binary finite fields GF(2^m). In Deprettere, E., Bhattacharyya, S., Cavallaro, J., Darte, A., Thiele, L., eds.: Proceedings of the 14th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2003), IEEE Computer Society Press (2003) 455–468
7. Großschädl, J., Savaş, E.: Instruction set extensions for fast arithmetic in finite fields GF(p) and GF(2^m). In Joye, M., Quisquater, J.J., eds.: Cryptographic Hardware and Embedded Systems — CHES 2004. Volume 3156 of Lecture Notes in Computer Science., Springer Verlag (2004) 133–147
8. Kumar, S.S., 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, August 30-September 1, 2004, Proceedings. Volume 3203 of Lecture Notes in Computer Science., Springer (2004) 586–595
9. Atasu, K., Pozzi, L., Ienne, P.: Automatic application-specific instruction-set extensions under microarchitectural constraints. In: DAC '03: Proceedings of the 40th conference on Design automation, New York, NY, USA, ACM Press (2003) 256–261
10. Hankerson, D., Menezes, A.J., Vanstone, S.A.: Guide to Elliptic Curve Cryptography. (Springer Professional Computing). Springer, Berlin (2004)
11. Shoup, V.: NTL: A library for doing number theory (http://www.shoup.net/ntl)
12. Karatsuba, A.A., Ofman, Y.: Multiplication of multidigit numbers on automata. Soviet Physics Doklady **7** (1963) 595–596
13. Kastens, U., Le, D.K., Slowik, A., Thies, M.: Feedback driven instruction-set extension. SIGPLAN Not. **39**(7) (2004) 126–135