Evaluating 16-Bit Processors for Elliptic Curve Cryptography

Erich Wenger, Mario Werner

Evaluating 16-bit Processors for Elliptic Curve Cryptography

Erich Wenger and Mario Werner

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
Erich.Wenger@iaik.tugraz.at, M.Werner@student.tugraz.at

Abstract. In a world in which every processing cycle is proportional to used energy and the amount of available energy is limited, it is especially important to optimize source code in order to achieve the best possible runtime. In this paper, we present a side-channel secure C framework performing elliptic curve cryptography and improve its runtime on three 16-bit microprocessors: the MSP430, the PIC24, and the dsPIC. To the best of our knowledge we are the first to present results for the PIC24 and the dsPIC. By evaluating different multi-precision and field-multiplication methods, and hand-crafting the performance critical code in Assembler, we improve the runtime of a point multiplication by a factor of up to 5.41 and the `secp160r1` field-multiplication by 6.36, and the corresponding multi-precision multiplication by 7.91 (compared to a speed-optimized C-implementation). Additionally, we present and compare results for four different standardized elliptic curves making our data applicable for real-world applications. Most spectacular are the performance results on the dsPIC processor, being able to calculate a point multiplication within 1.7 – 4.9 MCycles.

Keywords: Elliptic Curve Cryptography, ECC, Prime Field, MSP430, PIC24, dsPIC, Assembler Optimization.

1 Introduction

Public-key cryptography is an active research area with a lot of applications. The applications range from desktop computing, over (wireless) smart cards down to low-energy sensor networks. On desktop computers one can use special instructions or graphic processors in order to implement asymmetric cryptography. Even nowadays mobile phones are equipped with powerful 32-bit processors making the job of implementing public-key cryptography easier. However, there are a lot of circumstances which call for cheap, energy saving, and fast solutions.

Especially applications utilizing small, embedded processors are interesting for elliptic curve cryptography (ECC). An ECC implementation for such a microprocessor must be resource-conscious, aware of practically dangerous power, timing, and fault attacks and still deliver a runtime, which is fast enough for real-world applications. Additionally, many applications require the use of standard-

ized elliptic curves [1, 5, 27] that exceed a defined security threshold. Previous work [20, 29, 31, 36] did only consider a part of those requirements.

This paper focuses on a fast and secure implementation of ECC on three embedded 16-bit processors. By optimizing performance critical field operations in Assembler, we drive the PIC24 [22], dsPIC [23], and MSP430 [32] to their limits. To the best of our knowledge we are the first to report runtime results on the PIC24 and dsPIC. Furthermore we present runtime results on the MSP430 using standardized NIST curves [27]. Especially by taking advantage of the 16-bit multiply-accumulate unit of the dsPIC, we were able to improve its runtime for a point multiplication by a factor of up to 5.41, a field multiplication by 6.36, and a big-integer multiplication by 7.91 (compared to a speed-optimized C-implementation). Thus we reduced a 160-bit multi-precision multiplication to 180 cycles, with 100 cycles being the theoretical minimum.

Our framework is written and verified in C, uses a side-channel aware Montgomery Ladder including y-recovery (formulae by Hutter *et al.* [16]), and verifies the resulting point in order to check for fault attacks (see Ebeid *et al.* [9]). Additionally a projective point randomization [7] was used in order to strengthen the side-channel resistance of the Montgomery ladder. Having fixed our high-level algorithms, we investigate different big-integer multiplication methods (operand-scanning, product-scanning, and hybrid) and field-multiplication methods (Barrett, Montgomery, and fast-NIST-reduction) on the three processors. Our results can be the foundation for choosing suitable embedded processors (*e.g.* smart cards or sensor networks), for the investigation of important ECC-related features for future microprocessors, and for follow-up research on custom ECC-hardware designs.

The remainder of the paper is structured as followed. Section 2 discusses relevant related work. Section 3 gives an introduction to elliptic curve cryptography and the used algorithms. Whereas Section 4 gives a short introduction into the MSP430 processor, Section 5 discusses the performance optimizations made for the Microchip processors. Section 6 gives a comparison of all results and Section 7 concludes the work.

2 Related Work

In the last years, a lot of research has been done on implementing elliptic curve cryptography on embedded devices. Most notable is the work by Gura *et al.* [13], who compared elliptic curve cryptography and RSA for the 8-bit ATmega128 processor [2]. For achieving high performance, they introduced the hybrid multiplication method. The results by Gura *et al.* for the ATmega128 processor have been later improved by Uhsadel *et al.* [33], Scott *et al.* [29], and Liu *et al.* [21]. Further results for ATmega128 processors have been shown by Szczechowiak *et al.* [31]. Using the preceding work of Scott and Szczechowiak [29], Szczechowiak *et al.* [31] presented a comparison of ECC and pairings for the ATmega128 and MSP430 processors. Those processors are specially interesting, because they are used for the Wireless Sensor Network nodes MICA2 [8] and Tmote Sky [26]. They

compared the performance of the sensor nodes for the NIST K-163 Koblitz curve over $GF(2^{163})$ and a custom curve with $p = 2^{160} - 2^{112} + 2^{64} + 1$ over $GF(p)$. Their results are based on MIRACL [30], a Multiprecision Integer and Rational Arithmetic C/C++ Library.

Similar work with focus on sensor nodes has been published by Liu *et al.* [20]. They used the by SECG standardized `secp160r1` prime field [4]. Note that for security reasons this curve has been removed from the latest version of the standard [5].

In 2009, Yan *et al.* [36] used the 32-bit DSP processor TMS320C6416 from Texas Instruments to implement the `secp160r1` and `secp224r1` elliptic curves. This very powerful processor fits only partly within the scope of this paper, but uses 16-bit multipliers internally. It operates at 1 GHz, has 64 32-bit registers and 1,024 KB L2 cache.

In contrast to the MSP430 and TMS320C6416 processors, we use the ECC-processors by Kern *et al.* [18], Hutter *et al.* [15], and Wenger *et al.* [35] as references. Those papers present specially optimized semi-custom ASIC designs that use 16-bit multipliers. This makes them perfectly suitable for a comparison with the performance of the dsPIC processor.

The design by Kern *et al.* [18] generates an ECDSA signature within 511 kCycles for the elliptic curve `secp160r1`. Specially notable is their fast reduction technique taking advantage of two carry registers.

Hutter *et al.* [15] are using an 8-bit RISC microcontroller supporting 32 instructions, which is used for higher-level control-flow operations. In order to process ECC efficiently, they use eight microcode ROM tables. Those tables control a 16-bit multiply-accumulate unit and a dual-port 16-bit RAM. For NIST P-192 prime field, without reduction a multiplication only needs 168 clock cycles. This is very close to the optimum of $N^2 = 12^2 = 144$ clock cycles. The design performs ECDSA and has a chip area of only 19,115 gate equivalents.

The focus of the work of Wenger *et al.* [35] was to reduce the area footprint (11,686 gate equivalents) of their processor. They use a custom 16-bit RISC processor, featuring 46 instructions and 12 registers. The most notable feature of their Arithmetic Logic Unit (ALU) is a 16-bit multiply-accumulate unit. In order to reduce the disadvantage from their single-port RAM, they parallelized and combined the arithmetic and memory access instructions. Hence the run time of the same 192-bit multi-precision multiplication needs 252 clock cycles, which is 33 % longer than the design of [15].

3 Elliptic-Curve Cryptography

Elliptic-Curve Cryptography (ECC) is defined over the Weierstrass equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

with $a_{i=1,2,3,4,6}, x, y \in K$. K defines the finite field. If a combination of x and y fulfills Equation (1), it is noted a point $P = (x, y)$ on the elliptic curve. Using formulas for the addition and doubling of points, a point multiplication $Q = kP$

can be derived. Finding k, if Q and P are given, is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP).

Throughout this paper, K is a prime field $\mathbb{F}_p$. $n = \lceil log_2(p) \rceil$ bits or $N = \lceil n/W \rceil$ words are needed to represent an element of $\mathbb{F}_p$. W is the word size of the processor used. The binary representation of a field element a can be stored in an array $A = (A[N-1], \ldots, A[2], A[1], A[0])$ of N W-sized words. The least significant bit of a is stored in the rightmost bit of $A[0]$.

3.1 Algorithms Used

Every point operation uses the underlying field operations, discussed in the following subsection. Speeding up those point operations has been a major goal of a lot of researchers [3, 14]. Unfortunately most of those formulas are vulnerable to power, timing and fault-analysis attacks [10, 34]. So it is important to use a method that is less vulnerable to such attacks. Such a method is the Montgomery Ladder. This method performs a point addition and doubling for every key bit during a point multiplication. We applied the formula by Hutter *et al.* [16] which only requires 7 n-bit registers. Their formula needs 12 multiplications, 4 squarings and 17 additions per key bit and uses a Montgomery Ladder with common-Z coordinates. By randomly initializing this Z-coordinate [7], an additional, computationally cheap resistance against side-channel attacks can be added. In order to make fault attacks more difficult, the y-coordinate is recovered within the projective coordinates and a point verification [9] is performed.

Having fixed the high-level point-multiplication formula, one can concentrate on the fast and efficient implementation of field operations.

3.2 Modular Multiplication

The most time-critical field operation is the field multiplication [14] and the most time-critical part of the field multiplication is the multi-precision multiplication. Although there exist different multi-precision multiplication methods, for every multiplication, N^2 partial products are needed in order to get a $2N$-word result. Such methods are the operand-scanning, the product-scanning [6], and the by Gura *et al.* [13] introduced hybrid method.

The operand-scanning method is shown on the left of Figure 1. During operand scanning one row is calculated by multiplying one word of the first operand with all words of the second operand. The resulting partial products are immediately added to $C[k]$. Thus this method needs $2N^2 + N$ read operations and $N^2 + N$ write operations. Those are $3N^2 + 2N$ memory operations in total.

By rearranging the fixed number of partial products, the product-scanning method (see right part Figure 1) can be derived. Here the intermediate results are sorted column-wise. By using an accumulator [6, 12], the intermediate results can be summed up very efficiently. For this method N^2 read operations are needed to access $A[i]$ and $B[j]$ and only $2N$ write operations are needed to store $C[k]$. So $2N^2 + 2N$ memory operations are needed in total.

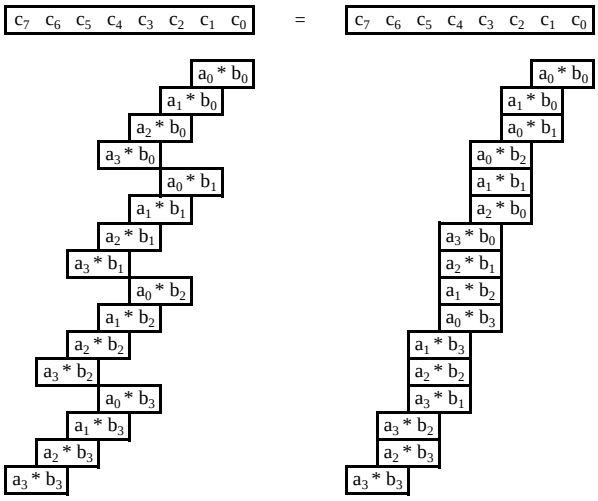

Fig. 1. Partial products during multi-precision multiplication. (left: Operand Scanning, right: Product Scanning)

In order to implement the operand and product-scanning multiplication methods, very few registers are needed. Gura *et al.* [13] takes advantage of the large register-set of embedded processors and combines the operand and product-scanning approaches. In an inner loop, d^2 intermediate products are processed row-wise (operand scanning) and in an outer loop, the intermediate sums are added column-wise (product scanning). The larger the parameter d is chosen, the more intermediate products are processed row-wise (d^2) and the more registers are needed ($3d + \lceil log_2(N/d)/W \rceil$) to implement this method efficiently. The advantage of the hybrid-multiplication method is that it can reduce the number of memory operations to $2\lceil N^2/d \rceil + 2N$.

3.3 Field Multiplication

The used multi-precision multiplication method has a significant influence on the runtime of a point and field multiplication. A field multiplication adds a reduction to the multi-precision multiplication. There are three very popular ways to perform this reduction: Barrett reduction, Montgomery reduction, and fast reduction (NIST reduction). We implemented all three methods and the results convinced us to do no further investigation into the Barrett reduction.

The fast reduction method is based on the special design of the primes used within the standards (*e.g.* NIST P-192; $p_{192} = 2^{192} - 2^{64} - 1$). This special Mersenne-like prime allows a reduction by only using addition and shift operations. In practice this method is very fast, because additions are much faster than multiplications. However there are Mersenne-like primes with a lot of (*e.g.* NIST P-256; $p_{256} = 2^{256} - 2^{128} - 2^{96} + 2^{32} - 1$) or odd (*e.g.* `secp160r1`;

$p_{160} = 2^{160} - 2^{31} - 1$) exponents, which have a significant influence on the runtime.

A method which is not dependent on the kind of the used prime is the Montgomery multiplication [25]. This method need some special set-up and back transformation of the field elements, but when there are a lot of operations involved, this method definitively pays off. First a $R = 2^{WN}$ is selected at design-time, making sure that $R > p$ and $gcd(R, p) = 1$. The field elements are transformed by

$$\tilde{a} \equiv aR \pmod{p}, \text{ and} \tag{2}$$

$$\tilde{b} \equiv bR \pmod{p}, \tag{3}$$

and further $\tilde{a}$ and $\tilde{b}$ are used instead of a and b. A multiplication is performed as follows:

$$\tilde{c} \equiv Mont(\tilde{a}, \tilde{b}) \equiv \tilde{a}\tilde{b}R^{-1} \equiv (aR)(bR)R^{-1} \equiv abR \equiv cR \pmod{p}. \tag{4}$$

Utilizing the Montgomery multiplication, the transformation of the field elements can be carried out easily:

$$Mont(a, R^2) \equiv aR^2R^{-1} \equiv aR \equiv \tilde{a} \pmod{p}, \tag{5}$$

$$Mont(\tilde{c}, 1) \equiv (cR)R^{-1} \equiv c \pmod{p}. \tag{6}$$

In practice the multi-precision multiplication and Montgomery reduction are interleaved. For the important implementation specific aspects of the Montgomery multiplication we refer to Koç *et al.* [19].

Every one of the previously discussed multiplication and reduction methods has advantages and disadvantages on certain microprocessors and the used prime. Those are discussed in the following sections.

4 Texas Instruments MSP430

The 16-bit MSP430 processor [32] by Texas Instruments is a very popular RISC processor. Especially its vast field of application makes it interesting (*e.g.* Tmote Sky [26]). The processor has 16 fully-addressable, single-cycle CPU registers that can be used with 27 instructions (24 additional instructions are emulated). Four of those registers are special purpose registers (program counter, stack pointer, status register, and constant generator), so not all registers can be used for an Assembler-optimized elliptic-curve implementation. A very important module for the multi-precision multiplication algorithm is the multiplier. The MSP430 does not have a multiplication instruction. Depending on the series, it only offers a 16-bit or 32-bit memory-mapped hardware multiplier. In order to perform a 16-bit multiplication, four memory accesses are necessary (write the two operand words and read the two resulting words).

For the MSP430, we implemented the three major multi-precision multiplication operations, discussed in Section 3.2 in Assembler. The advantage of the

operand-scanning method is that in average only three memory accesses are necessary for a single integer multiplication. Remember that one operand stays constant during the calculation of a row. The product-scanning multiplication method takes advantage of the `MAC` instruction. This instruction uses the result register as accumulator and provides a carry bit. So for every partial product, the two operands have to be written and only the carry flag has to be processed. For the hybrid multiplication results in Section 6, we used an implementation with $d = 2$.

In order to summarize our results for the MSP430, the biggest disadvantage in terms of ECC is that its hardware multiplier is memory mapped. Having a processor with a multiplication instruction improves ECC significantly. Such processors are the PIC24 and dsPIC.

5 Microchip PIC24 and dsPIC

The PIC24 and dsPIC microcontroller families by Microchip [22, 23] are 16-bit RISC processors that are using a modified Harvard architecture. They are widely used for motor control, signal processing, and intelligent sensor applications. These controllers use 24-bit instruction words with a variable length opcode field. Almost all instructions can be executed in a single cycle. Only commands which change the program flow, table operations and the double word move instruction take two or more cycles. Similar to the MSP430, the used PIC processors have 16 16-bit registers labeled as W0 to W15. W14 and W15 are used as frame and stack pointer.

The dsPIC processors facilitate different features that make them perfectly suited for public-key operations. First of all, the processor comes with a Digital Signal Processing (DSP) engine. This engine provides two 40-bit accumulators and a corresponding Arithmetic Logic Unit (ALU). This ALU is equipped with a multiply and accumulate (`MAC`) instruction. Additionally a second Address Generation Unit (AGU) is integrated. The primary address generation unit is called X, the secondary is called Y. Some of the DSP instructions can utilize these two AGUs and therefore fetch two operands at the same time while processing the data in the registers. However, not the whole address space is available to both AGUs. The X unit can read and write to all addresses while the Y unit can only read from a certain device specific region. Nevertheless it is possible to improve the performance considerable by taking the hassle of placing the data at the correct position in memory.

In order to minimize the overhead of loop constructs, special loop instructions named `REPEAT` and `DO` are provided. While `DO` needs two cycles to set up the loop and can execute several instructions multiple times, `REPEAT` needs one cycle and operates on a single command only. Another feature is the possibility to post increment or decrement pointers when indirect memory addressing is used. With this capability, the pointer arithmetic can be sped up significantly.

When performance is not the first priority using PIC24 processors is an option. Beside the missing DSP engine and the lack of the `DO` instruction the cores

of the PIC24 and the dsPIC processors are the same. Due to this compatibility we can give detailed results on the impact of the DSP engine concerning performance.

In order to find the fastest implementation for this processor architecture we start by presenting a generic product scanning algorithm which takes advantage of the DSP instructions. As the code of the inner loop is the most important code segment concerning speed, the following subsection focuses on this part. In the next step additional ideas to further optimize the multi-precision multiplication for speed are explained.

5.1 Generic Product Scanning on the dsPIC

The inner loop of the product scanning algorithm consists of one 16-bit multiplication and one addition (32-bit plus overflow handling). Additionally two load operations are required to fetch the values of $A[i]$ and $B[j]$ from memory. Furthermore the increment or decrement of i and j has to be handled as well. Usually this operations would involve two registers for the operand values, two registers for the multiplication product and three registers for the result of the addition.

As we have the DSP extension we take advantage of one of the 40-bit accumulators. The multiply-accumulate (`MAC`) instruction is used to multiply two 16-bit operands and to add the result to the accumulator. This means that we can handle the multiplication and the addition with the necessary precision in one clock cycle.

Now that we know how the data can be processed efficiently we still need to find a performant way to fetch the operand values. This can be implemented with the `MAC` instruction as well. As this DSP command possesses the capability to pre-fetch values by utilizing the two AGUs, it is possible to load both operands during one clock cycle.

By the use of pointers to address the current operands, the update of i and j can be implemented through applying post increments and/or decrements to those pointers. Due to the fact that this pointer arithmetic is also supported by this DSP operation the complete inner loop can be reduced to a single `MAC` instruction plus `REPEAT` command.

The result of this optimization can be seen in Figure 2 in Lines 14 to 16. At first the number of iterations for the `REPEAT` is calculated by subtracting the current loop count in DCOUNT (addressed via pointer in W0) from the total loop count (stored in W3). Afterwards the loop is set up using `REPEAT` followed by the `MAC` instruction which should be repeated. This `MAC` invocation reads as follows: Multiply the operands stored in *a_temp* and *b_temp* and add them to accumulator A ($A \leftarrow A + W5 * W6$). At the same time fetch the next *a_temp* via AGU X by dereferencing *aPtr_temp* ($W5 \leftarrow [W8]$). Simultaneously, the next *b_temp* is fetched via AGU Y by dereferencing *bPtr_temp* ($W6 \leftarrow [W10]$). The pointers are updated by incrementing *aPtr_temp* ($[W8]+=2$) and decrementing *bPtr_temp* ($[W10]-=2$) by two.

```
1  ; W0 = pointer to the DCOUNT register      ; W7 = pointer to the ACCAL register
2  ; W2 = result pointer                      ; W8 = temporary pointer (aPtr_temp)
3  ; W3 = integer length in words (length)    ; W9 = operand pointer (aPtr)
4  ; W4 = temp register (tmp)                 ; W10 = temporary pointer (bPtr_temp)
5  ; W5 = value of operand a (a_temp)         ; W11 = operand pointer (bPtr)
6  ; W6 = value of operand b (b_temp)
7
8        DO W3, loop                          ; main loop, iterates length+1 times
9        MOV W9,W8                                  ; aPtr_temp = aPtr
10       SUB W11,#2,W10                             ; bPtr_temp = bPtr - 2
11 ; pre-fetch operands for the inner loop          ; a_temp = *aPtr_temp++
12       MOVSAC A, [W8]+=2, W5, [W11]+=2, W6        ; b_temp = *bPtr++
13 ; calculate iteration count and execute the loop
14       SUB W3,[W0],W4
   ; tmp = length - DCOUNT
15       REPEAT W4
16       MAC W5*W6, A, [W8]+=2, W5, [W10]-=2, W6
17 ; store the resulting word and shift the accumulator
18       MOV [W7],[W2++]                            ; *resPtr++ = ACCAL
19 loop: SFTAC A, #16
```

Fig. 2. One of the two loops in the generic product scanning implementation.

As the `MAC` instruction always operates with the values which are already stored in *a_temp* and *b_temp*, it is mandatory to initialize them before this inner loop is executed. These fetches are done by using the `MOVSAC` operation in Line 12.

By looking at the pre-fetch behavior it can be seen that the last `MAC` operation loads operands which are not needed for the algorithm. By unrolling this last `MAC` invocation from the `REPEAT` it is possible to use the last pre-fetch to initialize *a_temp* and *b_temp* with the needed data for the next iteration. By doing this, the `MOVSAC` operation in the outer loop can be omitted.

5.2 Unrolled Product Scanning on the dsPIC

By unrolling the code for a fixed n-bit integer multiplication, advantage of the constant input size can be taken to improve the pointer arithmetic. The temporary pointers can be omitted when an appropriate sequence for the partial products is chosen as the post-in/decrement functionality of the `MAC` instruction is sufficient to update the pointers. The resulting zig-zag like pattern is nicely visualized in Figure 3. A part of the resulting source code is shown in Figure 4. Observe that the multiplicands for the multiplication are loaded in the preceding operations $W5 \leftarrow [W9]$ and $W6 \leftarrow [W11]$. In the once more preceding operation, the pointer registers are updated using the post in/decrement feature: $[W9]- = 2$, $[W11]+ = 2$.

At this point, we have to note that there is a prerequisite which has to be fulfilled. The pre-fetch mechanism requires that the operands are in different address spaces (X and Y). We ensure this by copying the input integers into

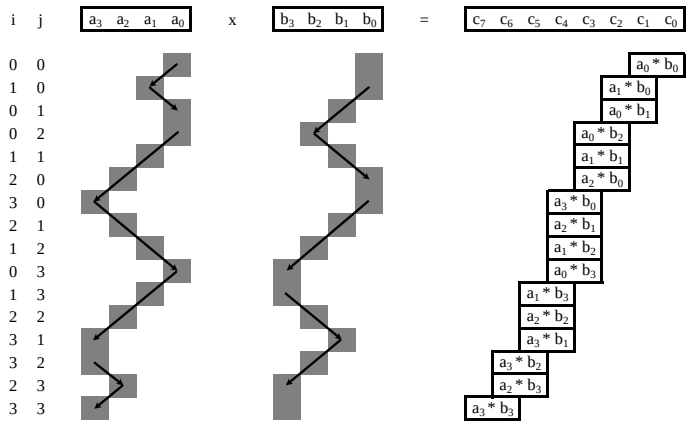

Fig. 3. Note the zig-zag product scanning multiplication method, which is specially suited for the dsPIC. (e.g. four word multi-precision multiplication)

temporary variables at the beginning of the multiplication. Although these copy operations are quite cheap (15 cycles for one 160-bit integer) it is possible to omit them by placing the operands in the correct memory region before the function call. As the generic implementation uses the pre-fetch technique too, the same constraint applies.

5.3 Montgomery Multiplication on the dsPIC

Like proposed by Koç *et al.* [19], various implementations of the Montgomery multiplication are possible which mainly differ in the underlying multiplication method (operand scanning, product scanning, ...) and the degree of integration of the reduction step. As product scanning provides the best results on the dsPIC architecture we have chosen the Finely Integrated Product Scanning (FIPS)

```
1     MAC W5*W6, A, [W9]-=2, W5, [W11]+=2, W6
2     MAC W5*W6, A, [W9], W5, [W11]+=2, W6
3     MAC W5*W6, A, [W9]+=2, W5, [W11]-=2, W6
4     MOV [W7],[W2++]
5     SFTAC A, #16
6
7     MAC W5*W6, A, [W9]+=2, W5, [W11]-=2, W6
8     MAC W5*W6, A, [W9]+=2, W5, [W11]-=2, W6
9     MAC W5*W6, A, [W9]+=2, W5, [W11], W6
10    MAC W5*W6, A, [W9]-=2, W5, [W11]+=2, W6
11    MOV [W7],[W2++]
12    SFTAC A, #16
```

Fig. 4. Unrolled calculation of the result word two and three.

method for our implementation. The interleaving of multiplication and reduction makes it necessary to implement the whole Montgomery multiplication using Assembler. Otherwise no decent performance can be gained following the FIPS approach. Through the high similarity of the product-scanning and the FIPS method most of the optimizations presented in the last sections can be applied as well, resulting in a fast implementation of the field multiplication.

6 Comparison Results

For the results shown in this section, we avoided to select the parameters for ECC ourselves. Instead we chose curves that have been standardized [1, 5, 27]. NIST [28] gave recommendations about the security margin for future applications. They recommend to use elliptic curves that exceed 160 bits. Also SECG removed the 160-bit curve in their latest release of the SEC standard [5]. Nevertheless, we present results for the `secp160r1` curve for comparison with related work. Additionally, the relative performance results shown in Subsection 6.1 are also applicable for larger elliptic curves. Those elliptic curves are discussed in Subsection 6.2.

For the generation of the results, for the MSP430 we used the IAR Embedded Workbench 5.30 [17] with the maximum optimizer settings available '-Ohs'. All results have been generated for the MSP430F1611 device which comes with an embedded 16-bit multiplier.

For the PIC24 and dsPIC results, we used the simulator of the MPLAB IDE v8.63 [24], the MPLAB C30 v3.25 compiler with settings '-o3', '-funroll-loops', and the PIC24FJ96GA006 and dsPIC30F6015 devices.

For the following comparisons it should be noted that we neglect parameters such as chip area, power consumption and cost factors, because they significantly differ from processor to processor. However energy is defined as product of power and time. So by minimizing the runtime, we also optimize the energy consumption.

6.1 Relative Performance

In Table 1, we compare the run times for big-integer multiplication, field multiplication and point multiplication using the `secp160r1` curve [4]. For all presented platforms, we started with a reference implementation in C. In C, the operand-scanning multiplication outperforms the product-scanning multiplication by 12.6 % (MSP430) and 16.4 % (PIC24). Consequently, the point multiplication is 8.2 % and 13.5 % faster. By manually writing the multi-precision multiplication in Assembler, the run time was reduced by a factor of 1.53 to 2.42. Implementing the hybrid multiplication method with $d = 2$ on the MSP430 improved the runtime by just 4.8 %. By only using 3.1 kbytes of program memory and 274 bytes of stack, this implementation is very resource friendly. Unrolling the product-scanning multiplication method improved the multi-precision multiplication by another 15.1 %, but came at the cost of additional 1.5 kbytes (47 %) more program code.

Table 1. Comparison of the multi-precision multiplication, the field multiplication and the ECC point multiplication for `secp160r1`.

Implementation			Multi-Prec. Mult.	$\mathbb{F}_{p_{160}}$ Mult.	Point Mult.
MSP430	C	op. sc.	4,103	6,069	16,985,654
MSP430	C	pr. sc.	4,699	6,665	18,512,606
MSP430[a]	ASM	op. sc.	2,583	4,127	11,380,361
MSP430[a]	ASM	pr. sc.	1,945	3,489	9,745,805
MSP430[a]	ASM	hybrid	1,851	3,395	9,504,977
MSP430[a]	ASM + unrolled	pr. sc.	**1,570**	**3,112**	**8,779,931**
PIC24	C	op. sc.	1,423	2,393	6,703,476
PIC24	C	pr. sc.	1,702	2,675	7,753,292
PIC24[b]	ASM	op. sc.	**929**	**1,909**	**5,463,648**
PIC24[b]	ASM	pr. sc.	1,031	2,011	5,739,732
dsPIC[c]	ASM + DO/REP.	op. sc.	622	998	2,840,921
dsPIC[c]	ASM + DO/REP.	pr. sc.	727	1,104	3,127,253
dsPIC[c]	ASM + DSP	op. sc.	546	923	2,648,377
dsPIC[c]	ASM + DSP	pr. sc.	267	644	1,932,431
dsPIC[c]	ASM + unrolled	pr. sc.	**180**	557	1,709,537
dsPIC[c]	ASM + DSP	Montgomery	—	554	1,696,433
dsPIC[c]	ASM + unrolled	Montgomery	—	**376**	**1,239,281**
MSP430	Liu *et al.* [20]				12,645,040
MSP430[d]	Scott *et al.* [29, 31]		1,746	2,736	5,760,000
TMS320	Yan *et al.* [36]		150	290	810,000
ASIC	Kern *et al.* [18]	pr. sc.		167	511,864

[a] Multi-precision addition and subtraction were manually unrolled in Assembler.
[b] Only multi-precision multiplication was manually written in Assembler.
[c] Multi-precision addition, subtraction, and shift operation were manually written in Assembler.
[d] Did not use `secp160r1`.

Up to this point, the results on the PIC24 processor are identical to the results of the dsPIC processor. In the following steps, we utilize the special features of the dsPIC to further improve the performance. By making use of the `DO` and `REPEAT` commands, the run-time of a single multi-precision multiplication was reduced by 29 % to 33 %. A more significant performance improvement was achieved by utilizing the DSP part of the dsPIC processor. A speedup of 2.72 has been achieved for the product-scanning multiplication method. Using the same methodology, the operand-scanning multiplication method improved by only 1.14. So only when we take advantage of the DSP-unit, the product-scanning method is (2.04 times) faster. A further experiment showed that by unrolling the Assembler code and performing the product-scanning in a zig-zag-like fashion, the run time could be further reduced by 32.6 %. At this point it should be noted that although the performance of the multi-precision multiplication has been improved by a factor of 7.91 (unrolled DSP vs. best C version), the performance of the field multiplication improved by 4.30 and the point mul-

tiplication improved by 3.92. A reason for that is the slow reduction modulo $p_{160} = 2^{160} - 2^{31} - 1$. The term 2^{31} results into a relatively slow shift operation. So we investigated the Montgomery multiplication technique. Utilizing the FIPS multiplication method from Koç *et al.* [19], the field multiplication was improved by 14.0 %. Unrolling the Assembler code resulted in the fastest field multiplication, just needing 376 cycles. This is 32.5 % faster than the fastest combination of unrolled product-scanning and fast reduction.

By investigating the related results for the MSP430, it becomes obvious that in comparison to Liu *et al.* [20], our point multiplication method is 25 % faster even though they used a memory-hungry sliding-window point multiplication method.

The hybrid multi-precision multiplication method by Scott and Szczechowiak *et al.* [29, 31] is 6 % faster than our multiplication method, because they unrolled their hybrid multiplication. However compared to our unrolled product-scanning method they are 11 % slower. The differences within the field and point multiplication come from a different elliptic curve and sliding-window point multiplication formula used. Because the sliding window technique needs additional memory, they need 2.9 kbytes of data memory and 31.3 kbytes of program memory, which is more than 10 times the resources we need.

The results on the dsPIC processor made us confident enough to compare them with the powerful TMS320C6416 processor (Yan *et al.* [36]) as well as the custom designed ASIC by Kern *et al.* [18]. The implementation by Yan *et al.* [36] using the mighty TMS320C6416 processor[1] and the custom designed ASIC ought to be faster. But the difference is only a factor of 1.53 – 2.42.

With this comparison we showed that the field multiplication utilizing a memory-mapped multiplication unit is more than 2 times slower. Also the advantages of having `DO/REPEAT` and DSP instructions have been discussed. Utilizing the full potential of the Montgomery multiplication method, the point multiplication has been improved by a factor of 5.41 versus the fastest C-only implementation.

6.2 Scaling of Performance

The last subsection limited the comparison to the `secp160r1` curve. This subsection extends our focuses to the NIST [27] standardized elliptic curves P-192, P-224, and P-256.

The most remarkable feature within Table 2 is the influence of the chosen field prime into the run time of the point multiplication. On the dsPIC processor, the performance of the FIPS Montgomery field-multiplication is better for the `secp160r1` and NIST P-256 fields, but the fast reduction technique utilizing Mersenne-like primes are faster for the NIST P-192 and P-224 prime fields.

The Assembler optimizations on the MSP430 resulted in a speedup of 1.93 to 2.34. The larger the used prime field, the larger is the achieved speedup. Also on the PIC24 architecture, speedups between 1.23 and 1.42 have been achieved.

[1] 64 32-bit registers. Eight independent functional units. 1 GHz.

Table 2. Point multiplication for different field parameters in kCycles.

	Implementation		`secp160r1`	**P-192**	**P-224**	**P-256**
MSP430	C	op. sc.	16,986	23,405	35,531	47,455
MSP430	ASM	hybrid	9,505	11,949	18,464	23,973
PIC24	C	op. sc.	6,703	8,985	13,781	18,992
PIC24	ASM	op. sc.	5,464	6,754	10,138	13,379
dsPIC	ASM + DSP	pr. sc.	1,932	**2,178**	**2,880**	5,079
dsPIC	ASM + DSP	Mont.	**1,696**	2,528	3,582	**4,879**
MSP430	Liu *et al.* [20]		12,645			
C6416	Yan *et al.* [36]		810		1,690	
ASIC	Kern *et al.* [18]	pr. sc.	512			
ASIC	Wenger *et al.* [35]	pr. sc.		1,377		
ASIC	Hutter *et al.* [15]	pr. sc.		783		

Utilizing the dsPIC, the biggest speedup factors, ranging from 3.89 up to 4.79, have been achieved.

Our results improved the work of Liu *et al.* [20] on the MSP430 processor. Again we compared our results with the powerful TMS320C6416 processor as well as some custom designed ASICs. The TMS320C6416 performs the point multiplication only 1.7 – 2.1 times faster, which is quite small compared to the processing power of the TMS320C6416. Also the ASIC designs ought to be faster. Although those design focused on area-optimizations the authors [15, 18, 35] achieved good run time results. Consequently the performance differs by a factor of 1.58 – 3.31.

7 Conclusion

In this paper we presented, evaluated, and optimized an elliptic curve point multiplication for three different processors using four different elliptic curves. Starting with a C reference implementation (which includes several countermeasures against possible attacks), we were able to improve the runtime by simply rewriting the performance critical field operations in Assembler. Therefore we evaluated the most commonly used big-integer and field multiplication methods. We achieved a speedup for a single point multiplication of 1.93 – 2.34 on the MSP430, 1.23 – 1.42 on the PIC24 and 3.89 – 5.41 on the dsPIC. Especially impressing is the possible speedup of 7.91 for a single 160-bit multiplication on the dsPIC which is 10.5 times faster than fastest corresponding operation on the MSP430.

By the best of our knowledge, we are the first to present results for ECC on the PIC24 and dsPIC and those results are especially interesting for future applications in which embedded processors are required.

Acknowledgements

This work has been supported by the Austrian Government through the research program FIT-IT Trust in IT Systems under the project number 825743 (project PIT). Finally we would like to thank Manfred Großmann for some fruitful discussions.

References

1. American National Standards Institute (ANSI). AMERICAN NATIONAL STANDARD X9.62-2005. Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.
2. Atmel Corporation. 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash. Available online at `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`, August 2007.
3. D. Bernstein and T. Lange. Explicit-formulas database. www.hyperelliptic.org/EFD.
4. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0. Available online at `http://www.secg.org/`, September 2000.
5. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0. Available online at `http://www.secg.org/`, January 2010.
6. P. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, October 1990.
7. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99, First International Workshop, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.
8. Crossbow Technology, Inc. MICAz Wireless Measurement System. Available online at `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`.
9. N. Ebeid and R. Lambert. Securing the Elliptic Curve Montgomery Ladder Against Fault Attacks. In *Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2009, Lausanne, Switzerland, 2009, Proceedings*, pages 46–50, September 2009.
10. J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-Art of Secure ECC Implementations: A Survey on known Side-Channel Attacks and Countermeasures. In *Hardware-Oriented Security and Trust - HOST 2010, In 3rd IEEE International Symposium, California, USA, June 13-14, 2010, Proceedings.*, pages 76–87. IEEE, 2010.
11. M. Großmann. Optimize Elliptic Curve Cryptography for MSP430 Processor. Bachelor Thesis at Graz University of Technology, May 2011.
12. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 133–147. Springer, August 2004.

13. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132. Springer, 2004.
14. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004.
15. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In S. B. O. Yalcin, editor, *Workshop on RFID Security – RFIDsec 2010, 6th Workshop, Istanbul, Turkey, June 7-9, 2010, Proceedings*, volume 6370 of *Lecture Notes in Computer Science*, pages 189–202. Springer, 2010.
16. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In A. Nitaj and D. Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 Fourth International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 170–187. Springer, 2011.
17. IAR Systems. IAR Embedded Workbench. Available online at `http://www.iar.com/`, 2011.
18. T. Kern and M. Feldhofer. Low-Resource ECDSA Implementation for Passive RFID Tags. In *17th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2010), December 12-15th, 2010, Athens, Greece, Proceedings*, pages 1236–1239. IEEE, 2010.
19. Ç. K. Koç, T. Acar, and B. S. K. Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3):26–33, June 1996.
20. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *International Conference on Information Processing in Sensor Networks - IPSN 2008, April 22-24, 2008, St. Louis, Missouri, USA, Proceedings.*, pages 245–256, St. Louis, MO, April 2008.
21. Z. Liu, J. Großschädl, and I. Kizhvatov. Efficient and Side-Channel Resistant RSA Implementation for 8-bit AVR Microcontrollers. In *Workshop on the Security of the Internet of Things - SOCIOT 2010, 1st International Workshop, November 29, 2010, Tokyo, Japan, Proceedings*. IEEE Computer Society, 2010.
22. Microchip. PIC24FJ128GA010 Family Data Sheet. Available online at `http://www.microchip.com`, October 2009. DS39747E.
23. Microchip. dsPIC30F6010A/6015 Data Sheet. Available online at `http://www.microchip.com`, March 2011. DS70150E.
24. Microchip. MPLAB Integrated Development Environment. Available online at `http://www.microchip.com`, 2011.
25. P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44:519–521, 1985.
26. Moteiv. The Moteiv Wireless Sensor Networks Website. `http://www.moteiv.com/`.
27. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009. Available online at `http://www.itl.nist.gov/fipspubs/`.
28. National Institute of Standards and Technology (NIST). SP800-57 Part 1: DRAFT Recommendation for Key Management: Part 1: General. Available online at `http://csrc.nist.gov/publications/drafts/800-57/Draft_SP800-57-Part1-Rev3_May2011.pdf`, May 2011.

29. M. Scott and P. Szczechowiak. Optimizing Multiprecision Multiplication for Public Key Cryptography. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2007/299, 2007.
30. Shamus Software. Multiprecision Integer and Rational Arithmetic C/C++ Library. Available online at `http://www.shamus.ie/`, 2011.
31. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In R. Verdone, editor, *Wireless Sensor Networks 5th European Conference, EWSN 2008, Bologna, Italy, January 30-February 1, 2008. Proceedings.*, volume 4913 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2008.
32. Texas Instruments. MSP430C11x1 - Mixed Signal Microcontroller. Available online at `http://focus.ti.com`, 2008.
33. L. Uhsadel, A. Poschmann, and C. Paar. Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes. In F. Stajano, C. Meadows, S. Capkun, and T. Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks 4th European Workshop, ESAS 2007, Cambridge, UK, July 2-3, 2007. Proceedings.*, volume 4572 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2007.
34. C. D. Walter. Simple Power Analysis of Unified Code for ECC Double and Add. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2004.
35. E. Wenger, M. Feldhofer, and N. Felber. Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices. In Y. Chung and M. Yung, editors, *WISA*, volume 6513, pages 92–106. Springer, 2010.
36. H. Yan, Z. J. Shi, and Y. Fei. Efficient Implementation of Elliptic Curve Cryptography on DSP for Underwater Sensor Networks. In *7th Workshop on Optimizations for DSP and Embedded Systems (ODES- 7)*, pages 7–15, March 2009.