Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices

Erich Wenger[1], Martin Feldhofer[1], Norbert Felber[2]

[1]Institute for Applied Information Processing and Communications,
Graz University of Technology
[2]Integrated Systems Laboratory,
Swiss Federal Institute of Technology Zürich
{erich.wenger,martin.feldhofer}@iaik.tugraz.at
felber@iis.ee.ethz.ch

Abstract. Hardware implementations for contactless devices like NFC or RFID tags face fierce constraints concerning the chip area and the power consumption. In this work, we present the low-resource hardware implementation of a 16-bit microprocessor that is able to efficiently perform Elliptic Curve Cryptography (ECC). The highly optimized design features the calculation of the Elliptic Curve Digital Signature Algorithm (ECDSA) using the standardized NIST curve in the finite field $\mathbb{F}_{p_{192}}$. We carefully selected the underlying algorithms to minimize the required memory resources while also keeping the required runtime within reasonable limits. In total, the microprocessor requires a chip area of 11686 gate equivalents and performs the ECDSA within 1377k clock cycles, which is to our knowledge the smallest implementation of ECDSA using the NIST P-192 curve published so far.

Keywords: Low-Resource Hardware Implementation, Microprocessor, Elliptic Curve Cryptography, ECDSA

1 Introduction

With the introduction of contactless technologies like Near Field Communication (NFC) and Radio Frequency Identification (RFID) the vision of the Internet of Things (IoT), where arbitrary objects communicate with each other, becomes reality. Until now, a major roadblock for large-scale use of this pervasive technology is the lack of security concepts and the missing low-resource implementations of cryptographic algorithms and protocols. With the introduction of security services like authentication and confidentiality the consumer acceptance can be improved. In our opinion, using strong cryptographic algorithms with a high security level and which are standardized is essential. This becomes comprehensible by looking at recently published attacks against proprietary solutions like the Mifare system [26].

A very suitable method of preventing attacks like forgery of goods is to use an authentication protocol with a digital signature scheme like the Elliptic Curve

Digital Signature Algorithm (ECDSA). Elliptic Curve Cryptography (ECC) can be used as a very efficient public-key scheme for low-resource processing because it requires far less resources than the widely known RSA scheme. With the ECDSA on a passive tag a proof-of-origin application can be easily established with the following approach. In principle, the passive tag stores a certificate in its non-volatile memory that has been signed by a central authority. First the reader retrieves the certificate from the tag and verifies the validity of the public key. Then it sends a random challenge to the tag. The tag uses its private key within the ECDSA to sign this challenge to prove the knowledge of the private key. The signature returned to the reader allows the verification of the authenticity of the tag.

Passively powered devices like NFC cards or RFID tags have fierce constraints concerning the chip area and power consumption. This is due to the low-cost requirement and the contactless operation, where the tag gets supplied from the electromagnetic field of a reader device. However, Elliptic-Curve operations with its underlying finite-field arithmetic is computational very intensive. Hence, a hardware module that performs ECDSA has to be highly optimized. In literature, implementations of ECDSA use one of three different design approaches. The first is to use an 8-, 16- or 32-bit general-purpose microprocessor [5, 9, 10, 13, 15]. In contactless devices where the application is fixed this approach is rather inefficient. The second possibility is to implement instruction-set extensions to an existing processor. The work of Großschädl et al. [8] evaluates this method. The third approach is to implement dedicated hardware modules with the single purpose to calculate ECDSA [3, 7, 14, 23–25].

In this paper we incorporate the ideas of building an optimized dedicated hardware module with the flexible approach of a general-purpose processor by designing a customized processor for calculating ECDSA. This approach combines the advantages of both implementation concepts. First, it can be extended with further functionality like protocol handling and other administrative tasks in a contactless device, but it is optimized for low-resource requirements, especially low chip area. In order to achieve the ambitious design goal to build the smallest hardware module for calculating ECDSA using the standardized NIST curve in $\mathbb{F}_{p_{192}}$, the instruction set and the used algorithms have been optimized. For that optimization, a multiply-accumulate unit is used in the ALU of the processor, the memory access to the RAM macro has been improved and the used algorithms are very memory and time efficient.

This paper is composed to show which measures have been taken to implement the smallest ECDSA hardware module. Section 2 presents a system overview, explains why the NIST P-192 curve has been selected and why a the implementation of a microprocessors is the best approach. Section 3 shows the selected algorithms used for ECC calculations and the underlying finite-field operations. In Section 4, the architecture of the microprocessor is depicted and the results of the design are shown in Section 5. Finally, conclusions are drawn in Section 6.

2 System Overview

In the following, we answer a few questions that allow a better understanding of the system and give insights to the most important design decisions.

2.1 Why are prime fields used as underlying fields?

The Elliptic Curve Digital Signature Algorithm can be performed, using various different elliptic curves and parameters. Binary extension fields $\mathbb{F}_{2^m}$ are used for many area-optimized ASICs. Unfortunately for an ECDSA signature, prime-field $\mathbb{F}_p$ operations are needed for the final operations, independent whether binary or primary fields are used as underlying field of the elliptic curve. So there are two solutions for this problem: either build an ASIC that is optimized for binary fields and extend it to be capable of calculating the final prime-field operations, or use prime-field only ECDSA parameters. In the second case, only prime-field arithmetic is required throughout the algorithms. As a result, the number of necessary arithmetic hardware components can be minimized. We think that the smallest implementations are processors that only use prime fields.

2.2 Is a microprocessor required?

A common application for elliptic curve signatures are smart cards. A future application for elliptic curve signatures are RFID tags, which today only have dedicated finite state machines. Both of those technologies have defined wireless interfaces (e.g. ISO-14443 [19] or ISO-15693 [18]). Those protocols can easily be implemented using a microprocessor. By having a small microprocessor managing the protocol and a co-processor handling the ECDSA signature introduces the problem of redundant hardware. Using the microprocessor for both tasks reduces the total chip-area requirement of an RFID tag. We think that a design using a processor will be used for the future ECC-capable RFID tags.

2.3 Why is a custom designed microprocessor necessary?

A good starting point would be to modify an existing processor. But in order to have maximum security, flexibility and performance, a custom processor has been designed. As a result, the functionality and the instruction set of the processor can be chosen freely. The requirements for such a processor are listed in the order of their importance:

- **Security** is more important than any other attribute. Nobody wants to pay for an insecure design. In the presented design, security is achieved by selecting secure, standardized and performance-efficient algorithms. Measures to counteract side-channel attacks have been implemented.

- **Low-Area.** The most important factor for electronic bulk commodity, using RFID systems are the costs of the tag. The price is mainly defined by the used manufacturing process and the required chip area. Fulfilling the low area requirement is achieved by using a small set of registers and a rather small, 16-bit multiplier. In general, the memory is the largest module of an ECDSA design. So, in order to keep the memory requirements low, curve parameters with a small prime size are selected: NIST-P192.
- **Low-Power.** The power for RFID tags is supplied by an electromagnetic field that is transmitted by a reader. The available power of such an electromagnetic field is limited by international regulations and decreases proportional to the distance between the tag and the reader antenna. As a result, the available power for an RFID tag is very low. For the implemented processor shown in this paper, we have concentrated our efforts on the low-area requirement and used methods like clock gating (for the registers) and operand isolation (for the multiplier) to reduce the power consumption.
- **Reusability.** A processor that does not at least support a minimum set of operations is practically useless. Such a processor would not be very flexible and nobody would use it. In our design the operations provided by the Thumb [1] and AVR [2] instruction sets are used as references. Both use a small, 16-bit instruction word. The resulting instruction set is extended and optimized for ECDSA and still supports the most important general-purpose instructions.
- **Performance** is a very important factor for every design. The special properties of the selected NIST-P192 parameters, make a fast reduction procedure possible. To further increase the performance of the presented processor, the memory access has been optimized and the instruction set has been parallelized.

2.4 Why is the signature verification algorithm implemented?

First of all, it is important to understand that the shown design is optimized for the ECDSA signature generation algorithm. Adding the extra functionality of a signature verification is very cheap. Only a few extra assembler functions and a bit more memory is required. However, having a signature verification algorithm on an embedded tag, greatly extends the usability of such a tag. In the case of an RFID reader-and-tag scenario, the tag could proof the authenticity of the reader, before actually sending his own digital signature.

With the hope that all the important questions are answered at this point, a more detailed discussion of the used algorithms is possible.

3 Implemented Algorithms

In order to calculate an elliptic curve signature, various algorithms need to be implemented. We decided not to cover an introduction to elliptic curves, point and field arithmetic, but we want to encourage the interested reader to take a look into [12]. The elliptic curve digital signature and verification algorithms are shown in Algorithm 1 and Algorithm 2.

Algorithm 1 ECDSA signature generation.

```
Require: Domain parameters, private key $d$, message $m$.
Ensure: Signature $(r, s)$.
1: Select $k \in_R [1, n-1]$.
2: Compute $kP = (x_1, y_1)$ and convert $x_1$ to an integer $\bar{x_1}$.
3: Compute $r = \bar{x_1} \bmod n$. If $r = 0$ then go to step 1.
4: Compute $e = H(m)$.
5: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6: Return$(r, s)$.
```

Algorithm 2 ECDSA signature verification.

```
Require: Domain parameters, public key $Q$, message $m$, signature $(r, s)$.
Ensure: Acceptance or rejection of the signature.
1: Verify that $r$ and $s$ are integers in the interval $[1, n-1]$. If any verification fails
   then return("Reject the signature").
2: Compute $e = H(m)$.
3: Compute $w = s^{-1} \bmod n$.
4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5: Compute $X = u_1P + u_2Q$.
6: If $X = \infty$ then return("Reject the signature");
7: Convert the x-coordinate $x_1$ of $X$ to an integer $\bar{x_1}$; compute $v = \bar{x_1} \bmod n$.
8: If $v = r$ then return("Accept the signature");
   Else return("Reject the signature").
```

3.1 Scalar Point Multiplication

The majority (about 95%) of the ECDSA's execution time is spent on the elliptic curve point multiplication (line 2 in Algorithm 1 and line 5 in Algorithm 2). This point multiplication algorithm needs to be save from simple and differential power analysis attacks. So it is necessary to use a point multiplication scheme as it is shown in Algorithm 3. In this scheme, point addition and point doubling is performed for every bit of the scalar. The only difference between a one and a zero bit in the scalar are some address bits. Because of this small change, we posture that a simple power analysis is not possible. A differential power analysis is also not possible because the ephemeral key k is changed in every request. Nevertheless, information about the point multiplication is additionally masked with a method called Randomized Projective Coordinates introduced

Algorithm 3 SPA-safe point multiplication.

```
Require: $k = (k_{t-1}, ..., k_1, k_0)_2$, $k_{t-1} = 1$, $P \in E(\mathbb{F}_q)$.
Ensure: $kP$.
1: $Q[0] \leftarrow P$.
2: $Q[1] \leftarrow 2P$.
3: for $i$ from $t-2$ to 0 do
4:     $Q[1 \oplus k_i] \leftarrow Q[k_i] + Q[1 \oplus k_i]$
5:     $Q[k_i] \leftarrow 2Q[k_i]$
6: end for
7: Return($Q$).
```

by Coron [6]. This very cheap and effective countermeasure multiplies the initial base point $P = (X, Y, Z)$ with a random integer λ so that $P = (\lambda X, \lambda Y, \lambda Z)$.

The point multiplication, as it is presented in Algorithm 3 can be further improved with the following methods:

- The Montgomery ladder introduced by Izu et al. [20] can be used to calculate a multiplication without the y-coordinates of the points. Let x_1, x_2 be x-coordinate values of two points P_1, P_2 of an elliptic curve $E : y^2 = x^3 + ax + b$. Then the x-coordinate value x_3 of the sum $P_3 = P_1 + P_2$ is given by

$$x_3 = \frac{2(x_1 + x_2)(x_1 x_2 + a) + 4b}{(x_1 - x_2)^2} - x'_3 \tag{1}$$

  where x'_3 is the x-coordinate value of $P'_3 = P_1 - P_2$. On the other hand, the x-coordinate value of x_4 of the doubled point $P_4 = 2P_1$ is given by

$$x_4 = \frac{(x_1^2 - a)^2 - 8bx_1}{4(x_1^3 + ax_1 + b)}. \tag{2}$$

- The doubling and add operations can be combined into a single function. As a result, intermediate results do not have to be calculated separately.
- In order to calculate Equations (1) and (2) efficiently, the base point $P = (x, y)$ needs to be transformed to the standard projective representation $P = (X, Y, Z)$, where $x = X/Z$ and $y = Y/Z$. As a result the inversions needed in Equations (1) and (2) can be avoided during the double-and-add algorithm. Note that one final inversion is still needed for the reversion of the standard projection.
- The z-coordinates of both points can be represented by a common z-coordinate. As a result, an 192-bit memory register can be saved.

All those optimizations have also been applied by Hutter [17] to create an efficient double-and-add algorithm that only requires seven 192-bit registers. This algorithm is based on the formulas by Izu et al. [20] and requires 16 field multiplications and 16 field additions.

This efficient multiplication scheme is used for the signature as well as for the verification algorithm. For that, the original algorithm needed some modifications. First of all, it cannot be assumed that the most significant bit of u_1 or

u_2 is always set (as it is for signing). But with a simple loop, the parameter t of Algorithm 3 can be easily found. Secondly, the base point P of the signature algorithm is always fixed. In this case, $x'_3 = P_x$ and $Q[1] \leftarrow 2P$ are both fixed parameters that can be stored as constants. The verification algorithm performs a point multiplication with the public key Q. Q varies from signature to signature. Therefore x'_3 and the initial $Q[1]$ vary. As a result, additional data memory is required for the storage of x'_3. The initial point doubling $Q[1] \leftarrow 2P$ has to be done once per verification. This is implemented very efficiently using Equation 2 and standard projective coordinates, resulting in 2 multiplications, 2 squarings and 10 additions. Two more functions are needed for a successful verification: a y-recovery and a point addition. The point addition as shown in Equation 1 cannot be used, because it assumes a constant difference between P_1 and P_2. So, the more general formula

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \tag{3}$$

which also uses the y-coordinates of the points is used. As a result a y-recovery has to be performed after each point multiplication. The y-recovery is done within the projected coordinates and needs 9 multiplications, 2 squarings, 9 additions and *no inversion.* The point addition algorithm needs 8 multiplications, 2 squarings and 4 additions. It takes advantage of the fact that only the x-coordinate is needed for the verification algorithm.

Up to now, it has been assumed that efficient field operations are available. Those operations are described in the following subsection.

3.2 Finite-Field Arithmetic

The finite-field operations build the basics for the point multiplication and are the most time-critical algorithms. They are performed modulo the prime $p_{192} = 2^{192} - 2^{64} - 1$. Whereas the field addition and subtraction operations can hardly be optimized, there are a lot of possibilities to improve the field multiplication. Most importantly, we designed our processor with a multiply-accumulate unit as central arithmetic logic unit. As a result, a fast product scanning multiplication algorithm can be applied. Secondly, contradicting the low-area requirement, all loops are unfolded. By doing so, no cycles for in/decreasing memory reference registers or variable range checks are wasted. Thirdly, the reduction of the upper half of the multiplication result is done in-place. This behavior has hardly any effect on the execution time but reduces the requirement for any temporary memory. Table 1 shows the big difference between the native and the highly optimized instruction set. For the native implementation, very simple instructions have been used. By modifying those instructions and parallelizing data load and calculation instructions, the runtime has been improved by a factor of 2.5. Furthermore, by using the available work registers as Cache, the runtime has been reduced to 328 cycles. This renders very effective because a single-port memory is used due to the the low area requirement.

At this point it is interesting to note that the squaring operation has been omitted. By adding such a function to the design, the runtime improves by 7.6%, whereas the area increases by 15.5%. Hence, we decided to use the standard multiplication for squaring. Most of that area increase is resulted by having a program memory with more entries.

Further necessary operations are field addition, multiplication and inversion algorithms that use a general integer n as modulo (lines 3-5 in Algorithm 1 and 3-4 in Algorithm 2). The prime used for those operations is not as nice as p_{192}. More general algorithms need to be used. For the multiplication, the Finely Integrated Product Scanning Form (FIPS) of a Montgomery multiplication algorithm by Koc et al. [22] is used. Table 1 shows a short comparison of three different implementations: Using a native instruction set with unfolded loops, using the optimized instruction set for the same implementation and using the optimized instruction set and loops in order to reduce the code size. Because the Montgomery Multiplications are less relevant concerning the runtime, the smallest implementation is used.

Method	Cycles	Code Size
NIST P-192		
Native IS	997	997
Optimized IS	401	401
Registers as Cache	**328**	**328**
Montgomery Multiplication		
Native IS	1580	1580
Optimized IS	651	651
Optimized Loops	**2021**	**84**

Table 1. Comparison of different multiplication functions. IS stands for Instruction Set. The bold implementations have been used.

The slowest of all algorithms is the inversion algorithm. This algorithm is needed for the reversion of the projective point coordinates (inversion of the z-coordinate) and the inversion of k and s in the ECDSA algorithm. For that a Montgomery inverse algorithm (see [21]) with final Montgomery multiplication (see [11]) is used. Because the speed is of minor importance, the focus has been on a small (program-memory efficient) implementation.

Generally, its a balancing act to choose which resources are spent on performance and which hardware is rather omitted. The next section covers a discussion of the used hardware architecture.

4 Hardware Architecture

All the previously described algorithms are implemented on a brand new processor architecture (see Figure 1). This processor is fully custom designed to perfectly suit the elliptic curve algorithms. Because of the aspired RFID application, the processor is ought to run on a relatively low frequency (less than 20MHz) and be as area efficient as possible. The design is based on a Harvard architecture (separated program and data memory). Investigation showed that a 16-bit word size is a good compromise between a low-area design and still being able to compute an ECDSA signature within a reasonable time.

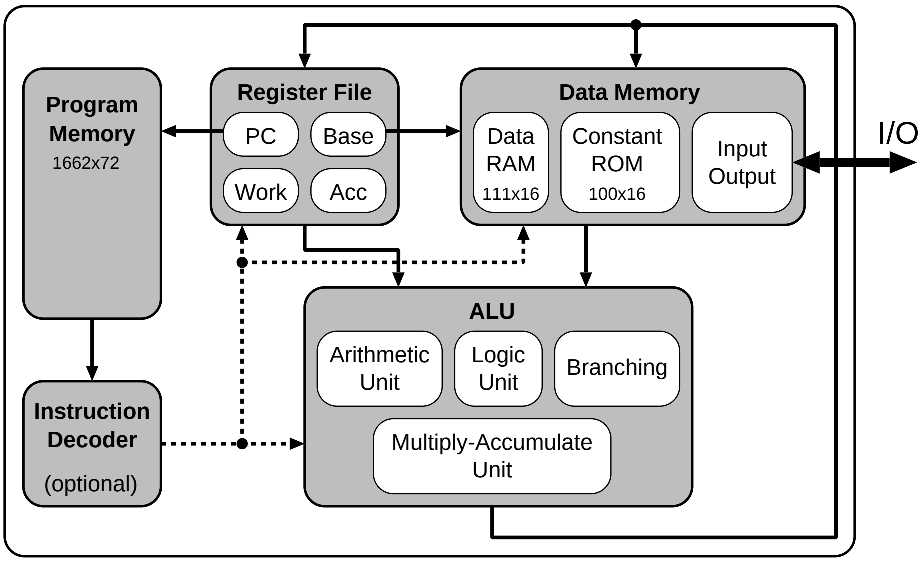

Fig. 1. Block diagram of the implemented processor with register file, ALU, data memory and program memory with the instruction decoder.

Many implementations (see [3, 14]) have the problem of an excruciating large data memory. They make use of synthesized dual-port memories. Although a dual-port memory improves the runtime of a signature generation by a factor of two, it has been decided to use a much smaller, single-port RAM macro. As a result, the data memory is implemented very area efficiently.

The data memory is split in three parts. One is the already mentioned RAM macro. A second part is a look-up table for constants. The third part are memory mapped inputs and outputs (I/O). Those I/O modules can be any memory logic necessary for the design. In the case of an RFID tag an interface such as ISO-14443 or ISO-15693 (see [19, 18]) would be such an I/O module.

The program memory is a synthesized look-up table. All 72 control signals are directly encoded in each entry of the table. Two other implementations with an optimized instruction set that use a 16-bit look-up table or a 16-bit ROM macro need more chip area. Those implementations would use the instruction decoder shown in Figure 1. The index for the selection of the control vector is the program counter (PC) of the CPU.

The CPU is equipped with a small register file. Like every other processor, it comes with a program counter and a stack pointer register. Because of the multiply-accumulate unit embedded within the ALU, an accumulator register is part of the design. This accumulator is actually implemented as three 16-bit registers. Further required registers are three 'base' registers, used to reference memory locations, and four work registers. Those work registers are the only registers without any special purpose. The CPU also contains status bits (Carry, Zero, Overflow, Negative) that are found in every microprocessor.

With this basic information, it is possible to look at some details. The most important feature of the processor is that it can process data loaded from the memory very efficiently. Unlike other designs, where the data from the memory needs to be stored in a register before it can be used, our processor can process the newly loaded data directly. This is especially important for the memory intensive field-multiplication algorithms. Another important feature is that the result of an ALU operation can be directly stored in a register *and* the memory. Such features in connection with the parallelized instruction set is the perfect environment for a fast and efficient calculation of an ECDSA signature. In this case, 'parallelized' means that several operations are encoded within one instruction and therefore they can be performed within the same cycle. As an example, a multiply-accumulate instruction that processes the read data can also request new data from the RAM within the same cycle. A short example that combines all those advantages is shown in Figure 2. The results of those optimizations has already been shown in Table 1.

1	**LD** A1
2	**MOVNF**
3	**LD** B11
4	**MOVNF**
5	**MULACC**
6	**LD** A2
7	**MOVNF**
8	**LD** B10
9	**MOVNF**
10	**MULACC**
11	**LD** A3
12	**MOVNF**
13	**LD** B9
14	**MOVNF**
15	**MULACC**
16	**STR** Acc0
17	**RSACC**

1	**LD** A1		
2	**MOVNF**	**LD** B11	
3	**LD** A2	**MULACC**	
4	**MOVNF**	**LD** B10	
5	**LD** A3	**MULACC**	
6	**MOVNF**	**LD** B9	
7	**MULACC**	**STR** Acc0	**RSACC**

Fig. 2. Part of a field multiplication algorithm. It stores and right shifts the result of the following equation: $A1 \cdot B11 + A2 \cdot B10 + A3 \cdot B9$. On the left hand side is the un-optimized code. On the right hand side is the resulting optimized code. **MOVNF** stores the previously loaded data into a register.

5 Results

The results of our implementation are summarized in Table 2. For achieving these results, we used the UMC-L180 technology in connection with the synthesis tool Design-Compiler from Synopsys. As already mentioned, the signature verification algorithm can be easily implemented based on the signature generation algorithm. Hence, we compare the sign-only version with the sign-and-verify implementation in the following.

Part	**Sign**	**Verify** (+ Sign)	Difference
Execution Time [kCycles]:			
Field multiplication	1003	2013	+100.7%
Field add/subtract	196	392	+100.0%
Montgomery inverse	95	95	-
Total	1377	2645	+92.1%
Number of words/entries:			
Data RAM	111	149	+34.2%
Program memory	1662	2321	+39.7%
Constants	100	148	+48.0%
Area [Gate Equivalents]:			
Program memory	4092	5441	+33.0%
CPU	4354	4354	-
Data RAM	2553	2988	+17.0%
Other	687	837	+21.8%
Total	11686	13620	+16.5%

Table 2. Comparison of two processor implementations. One is used for the ECDSA signature generation, the other one for the verification of an ECDSA signature.

Because the verification algorithm needs two point multiplications, the execution time differs greatly. Almost twice as much cycles are required for the field multiplications and additions used within the verification algorithm. The total cycles spent on the Montgomery inversion is roughly the same because both implementations need two inversions. Although larger memories and look-up tables are required for the verification design, this results in a small increase of the total area requirement by only 12.4%.

Relative to the total runtime of the signature generation algorithm, 73% of the runtime is spent on the field multiplications, 14% on the field additions and subtractions and 6.9% on two executions of the Montgomery inverse. Also an investigation on the used instructions has been made. The five most used commands (of 46 in total) are in connection with the memory. Four of them are parallelized instructions. Compared to dedicated co-processors, the **CALL** and **RET** instructions are a sign of overhead of the presented processor. Both instructions cover 44412 cycles, which is three percent of the total runtime. In other words, the runtime overhead of the presented processor is very small compared to a dedicated co-processor.

In many other ASIC designs, the control logic is the smallest part. However, the largest part of our processor is the program memory. It takes up 4092GE of the total chip area. This is because the implemented ECDSA signature algorithms needs a lot of different sub-functions. The major part of the CPU is the ALU with 2608GE.

	Area [GE]	Cycles [kCycles]	ECC Curve	VLSI technology	Processor
Kumar 2006 [23]	15094	430	B-163	AMI C35	NO
Hein 2008 [14]	13685	306	B-163	UMC L180	NO
Yong Ki Lee 2008 [24]	12506	276	B-163	UMC L130	YES
Bock 2008 [4]	12876	80	B-163	220nm	NO
Leinweber 2009 [25]	8756	191	B-163	IBM L130	YES
Auer 2009 [3]	24750	1031	P-192	AMS C35	NO
Fürbass 2007 [7]	23656	500	P-192	AMS C35	NO
Hutter 2010 [17]	19115	859	P-192	AMS C35B4	YES
This work	**11686**	**1377**	**P-192**	**UMC L180**	**YES**

Table 3. Comparison of our implementation with related work.

Table 3 gives a comparison of our work with various other publications. All of those implementations have been optimized for low area and low power requirements. Some implementations use the binary extension field $\mathbb{F}_{2^{163}}$ (according to NIST). This field provides a big advantage over the NIST prime fields $\mathbb{F}_{p_{192}}$. The computational complexity of a field multiplication is a lot lower. Hence the total number of cycles required for a point multiplication is a lot simpler. So it is not unexpected that the designs by Kumar, Hein, Yong Ki Lee, Bock and Leinweber (see [4, 14, 23–25]) are better in terms of area and execution time. But none of those designs implemented a full ECDSA signature. All of them concentrated on an efficient point multiplication.

Our implementation covers the ECDSA and SHA-1 algorithms and even so, it is 40% smaller than the smallest NIST P-192 implementation by Fürbass [7].

Because our created processor does not only outperform the shown dedicated hardware modules we also compare our work with implementations on other processor platforms. The comparison shown in Table 4 only lists papers which provide results for a NIST P-192 point multiplication.

	Processor	Word size [bit]	Cycles [kCycles]
Gura 2004 [10]	AVR ATmega128	8	9920
Hu 2004 [16]	Trimedia TM1300	32	2955
This work		**16**	**1377**

Table 4. Comparison of the performance of different processors.

Our processor performs better than the implementation by Gura et al. [10] on an 8-bit AVR processor. This probably is due to the small word size of that processor. But our design is also twice as fast than the implementation by Hu et al. [16] on a 32-bit Trimedia processor. This is especially spectacular, because the word size of the Trimedia processor is twice the word size of our implementation.

Using the tool First Encounter, a power simulation of the placed and routed design resulted in a power dissipation of $193\mu W$ at a frequency of $1.695MHz$. This is an eight of the carrier frequency used for a ISO-14443 RFID tag.

6 Conclusion

In this paper we have presented the hardware design of a low-resource microprocessor that has been optimized for the implementation of Elliptic Curve Cryptography. We have investigated the required algorithms to perform ECDSA signature generation and verification for the NIST P-192 curve which is defined over the prime field $\mathbb{F}_{p_{192}}$. The 16-bit microprocessor in Harvard architecture features a multiply-accumulate unit and has optimized access to the data memory in order to perform fast finite-field operations and hence allows fast ECC point multiplication. The processor has a chip area of 11686 gate equivalents including program memory, data memory, register file and ALU. It requires for the calculation of one ECDSA signature 1377k clock cycles. Our solutions outperforms all published software solutions and dedicated hardware modules in terms of chip area and allows an efficient calculation of ECDSA in authentication applications for contactless devices.

Acknowledgements.

We would like to thank the team working at the Microelectronics Design Center at the Swiss Federal Institute of Technology Zürich (especially Frank Gürkaynak and Beat Muheim) for their helpful comments concerning chip design and the team at the Institute for Applied Information Processing and Communications at Graz University of Technology (especially Michael Hutter) for their help concerning the elliptic curve algorithms and the related power analysis counter measures.

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).

References

1. ARM Corporation. 16-bit Thumb Instruction Set. Available online at `http://infocenter.arm.com/help/topic/com.arm.doc.qrc0006e/QRC0006_UAL16.pdf`, May 2010.
2. Atmel Corporation. 8-bit AVR Instruction Set. Available online at `http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf`, May 2008.

3. A. Auer. Scaling hardware for electronic signatures to a minimum. Master's thesis, TU Graz, October 2008.
4. H. Bock, M. Braun, M. Dichtl, E. Hess, J. Heyszl, W. Kargl, H. Koroschetz, B. Meyer, and H. Seuschek. A Milestone Towards RFID Products Offering Asymmetric Authentication Based on Elliptic Curve Cryptography. Invited talk at RFIDsec 2008, July 2008.
5. M. K. Brown, D. R. Hankerson, J. C. L. Hernández, and A. J. Menezes. Software Implementation of the NIST Elliptic Curves Over Prime Fields. In D. Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographers' Track at the RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2001.
6. 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.
7. F. Fürbass and J. Wolkerstorfer. ECC Processor with Low Die Size for RFID Applications. In *Proceedings of 2007 IEEE International Symposium on Circuits and Systems*. IEEE, IEEE, May 2007.
8. 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.
9. J. Guajardo, R. Blümel, U. Krieger, and C. Paar. Efficient Implementation of Elliptic Curve Cryptosystems on the TI MSP430x33x Family of Microcontrollers. In K. Kim, editor, *Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001, Cheju Island, Korea, February 13-15, 2001. Proceedings*, volume 1992 of *Lecture Notes in Computer Science*, pages 365–382. Springer, 2001.
10. 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.
11. G. Hachez and J.-J. Quisquater. Montgomery Exponentiation with no Final Subtractions: Improved Results. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000, Second International Workshop Worcester, MA, USA, August 1718, 2000 Proceedings*, volume 1965, pages 91–100. Springer Berlin / Heidelberg, August 2000.
12. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004.
13. T. Hasegawa, J. Nakajima, and M. Matsui. A Practical Implementation of Elliptic Curve Cryptosystems over GF(p) on a 16-Bit Microcomputer. In H. Imai and Y. Zheng, editors, *Public Key Cryptography (PCK 1998)*, volume 1431 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 1998.
14. D. Hein, J. Wolkerstorfer, , and N. Felber. ECC is Ready for RFID - A Proof in Silicon. In *Workshop on RFID Security 2008 (RFIDsec08)*, July 2008.

15. Y. Hu, Q. Li, and C. Kuo. Efficient implementation of elliptic curve cryptography (ECC) on VLIW-micro-architecture media processor. In *2004 IEEE International Conference on Multimedia and Expo, 2004. ICME'04*, volume 2, 2004.
16. Y. Hu, Q. Li, and C. C. J. Kuo. Efficient implementation of elliptic curve cryptography (ecc) on vliw-micro-architecture media processor. In *ICME*, pages 879–882, 2004.
17. M. Hutter, M. Feldhofer, and T. Plos. An ECDSA Processor for RFID Authentication. In B. Ors, editor, *Workshop on RFID Security - RFIDsec 2010, 6th Workshop, Istanbul, Turkey, June 7-9, 2010, Proceedings*, Lecture Notes in Computer Science. Springer, 2010.
18. International Organisation for Standardization (ISO). ISO/IEC 15693-3: Identification cards - Contactless integrated circuit(s) cards - Vicinity cards – Part 3: Anticollision and transmission protocol, 2001.
19. International Organization for Standardization (ISO). ISO/IEC 14443: Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards, 2000.
20. T. Izu, B. Möller, and T. Takagi. Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks. In A. Menezes and P. Sarkar, editors, *INDOCRYPT*, volume 2551 of *Lecture Notes in Computer Science*, pages 296–313. Springer, 2002.
21. B. Kaliski. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.
22. Ç. K. Koç, T. Acar, and B. S. K. Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3):26–33, June 1996.
23. S. S. Kumar and C. Paar. Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In *Workshop on RFID Security 2006 (RFIDSec06), July 12-14, Graz, Austria*, 2006.
24. Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-Curve-Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, November 2008.
25. L. Leinweber, C. Papachristou, and F. Wolff. Efficient Architectures for Elliptic Curve Cryptography Processors for RFID. *International Conference on Computer Design (ICCD)*, 2009.
26. K. Nohl, D. Evans, Starbug, and H. Plötz. Reverse-Engineering a Cryptographic RFID Tag. In *USENIX Security Symposium, San Jose, CA, USA, 31 July, 2008, Proceedings*, pages 1–9. USENIX, 2008.