A Hardware Processor Supporting Elliptic Curve Cryptography for Less than 9 kGEs

Erich Wenger, Michael Hutter

A Hardware Processor Supporting Elliptic Curve Cryptography for Less Than 9 kGEs

Erich Wenger and Michael Hutter

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{Erich.Wenger,Michael.Hutter}@iaik.tugraz.at

Abstract. Elliptic Curve Cryptography (ECC) based processors have gained large attention in the context of embedded-system design due to their ability of efficient implementation. In this paper, we present a low-resource processor that supports ECC operations for less than 9 kGEs. We base our design on an optimized 16-bit microcontroller that provides high flexibility and scalability for various applications. The design allows the use of an optimized RAM-macro block and reduces the complexity by sharing various resources of the controller and the datapath. Our results improves the state of the art in low-resource $\mathbb{F}_{2^{163}}$ ECC implementations (14 % less area needed compared to the best solution reported). The total size of the processor is 8,958 GEs for a 0.13 μm CMOS technology and needs 285 kcycles for a point multiplication. It shows that the proposed solution is well suitable for low-power designs by providing a power consumption of only 3.2 μW at 100 kHz.

Keywords: Low-Resource Hardware Implementation, Elliptic Curve Cryptography, Binary Extension Field, Embedded Systems.

1 Introduction

With the rapid development of more powerful and energy-saving devices, we unwittingly move towards the vision of the Internet of things. The required security services within this vision can be particularly achieved using Elliptic Curve Cryptography (ECC). This paper focuses on a low-resource hardware processor that provides ECC capabilities while meeting the low-area and low-power requirements of embedded systems.

There exist many proposals for low-resource ECC processors. Most of the processors operate on binary-field elliptic curves and use full-precision arithmetic to increase the performance of point multiplication [4, 13, 25, 35]. One of the most efficient solutions in terms of low-resource requirements has been reported by Lee et al. [26].

They presented a processor supporting a small elliptic curve over $\mathbb{F}_{2^{163}}$ which makes use of a tiny 8-bit microcontroller to handle higher-level protocol implementations. The ECC operation of $k \cdot P$ is performed by a separated Modular Arithmetic Logic Unit (MALU). The processor needs 12,506 GEs and 276 kcycles

to perform a point multiplication. However, the area estimations do not including program ROM and RAM to store intermediate results and the necessary secret scalar k. Similar datapath architectures have been reported by Batina et al. [2] and Sakiyama et al. [32]. Hein et al. [17] reported a very efficient co-processor (without microcontroller) for the same elliptic curve supporting multi-precision arithmetics. They applied a finite-state machine based control-engine needing 11,904 GEs including a standard-cell based RAM memory.

In this paper, we present a low-resource hardware processor that is based on a 16-bit multi-precision architecture and an area-optimized custom microcontroller. This combination allows several optimizations. First, it allows the use of an efficient RAM-macro block that reduces the area requirements for short-term memory significantly. Second, since both the microcontroller and the datapath use a 16-bit architecture, all resources are shared to minimize the area footprint of the processor. As an outcome, we present a complete solution including memory for short-term (RAM) as well as long-term storage (program ROM), controller, and datapath using a polynomial multiply-accumulate (MAC) unit. In addition, we present results of higher-level protocol implementations of the Elliptic Curve Digital Signature Algorithm (ECDSA) [30] and give results for digital signature generation as well as verification. For a point multiplication, our NIST B-163 based processor needs only 8,958 GEs in total and performs a point multiplication within 285 kcycles. We demonstrate that the proposed solution is also well suitable for low-resource embedded systems by providing a power consumption of only 3.2 μW at 100 kHz.

The rest of the article is structured as follows. In Section 2, a brief introduction into elliptic curve cryptography is given. In Section 3, we face the challenge of low-resource ECC hardware implementations and explore various design possibilities. We evaluate appropriate word sizes of a processor and analyze different memory types. Section 4 presents details about the hardware architecture of our processor. Details about the implementation are given in Section 5. In Section 6, the results are presented. Conclusions are drawn in Section 7.

2 Elliptic Curve Cryptography

Within Elliptic Curve Cryptography (ECC), not only a single number or polynomial is used, but a pair of those. Each pair (x, y) of such numbers that satisfy the general Weierstrass equation

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

is called a point on an elliptic curve. When a certain type of number is used, in our case binary polynomials within $GF(2^m)$, the Weierstrass equation can be reduced to

$$y^2 + xy = x^3 + ax^2 + b. \tag{2}$$

Among the most critical operation in terms of speed and security is the ECC point multiplication. The implementation of this multiplication has to be secure

against various implementation attacks such as side-channel and fault-analysis attacks. The Montgomery ladder [28, 21] provides very beneficial properties in this context. We therefore decided to use it for our design and applied the very fast group-operation formulas of López and Dahab [27]. The formulas are based on projective coordinates (which avoid expensive field inversions) that can be nicely combined with proposed countermeasures (see also the work of Junfeng Fan et al. [11]) such as randomized projective coordinates (RPC) [6] or point-validity checks [8].

We use the following notations throughout the paper (similar to [16]). Let $f(z) = z^m + r(z)$ denote an irreducible binary polynomial of degree m. The elements of $\mathbb{F}_{2^m}$ are binary polynomials of degree at most $m-1$. An addition of field elements is the usual addition of binary polynomials. Multiplication is performed modulo $f(z)$. A field element $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$ is associated with the binary vector $a = (a_{m-1}, \ldots, a_2, a_1, a_0)$ of length m. Furthermore, let $N = \lceil m/W \rceil$ be the number of words with width W needed to store $a(z)$. $A = (A[N-1], \ldots, A[2], A[1], A[0])$, where the rightmost bit of $A[0]$ is a_0, and the leftmost $(WN - m)$ bits of $A[N-1]$ are unused (always set to zero).

For further readings on ECC we refer to several books [1, 3, 16, 23] that discuss the topic extensively.

3 Design-Space Exploration

In this section, we will explore different hardware-design options to obtain best results for a low-resource ECC processor. The design goals have been to meet all requirements of embedded systems which are low area (due to the production costs), low power (due to a possible contactless operation), appropriate speed (required for certain applications), security and flexibility. Due to the latter requirement, we decided to base our design on a customized microcontroller. This has the advantage of being modular in terms of protocol implementations and modifications of already implemented solutions.

By following the principles of hardware/software co-design, it showed that the dominant factors of ECC processors are the finite-field hardware multiplier and the type and size of the applied data memory. In the following, we discuss these factors and explore the design space to find the best solution for our objectives.

3.1 The Hardware Multiplier

One of the most area consuming parts within the ALU of an ECC-hardware design is the finite-field multiplier. The size, speed, and power consumption of such a multiplier largely depends on the word size of the processor and the underlying finite field. Figure 1 shows the hardware architecture of a 4-bit multiplier for binary-field (carry-less multiplier), prime-field (integer multiplier), and dual-field arithmetic. The basic structure of all three types of multiplier is the same. Only the adder structure needs to be adopted.

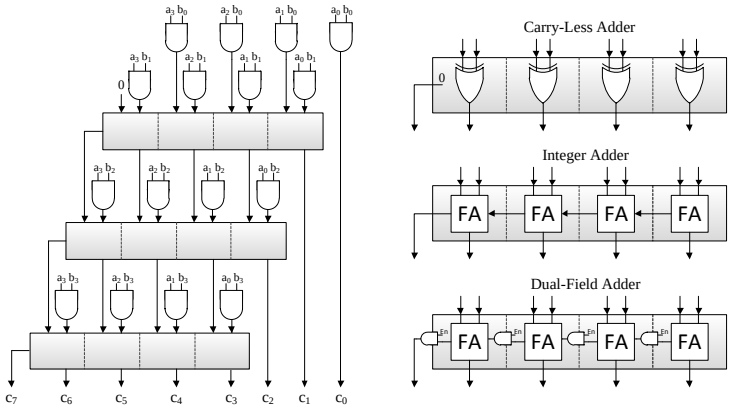

Fig. 1. General 4-bit multiplier structure to the left. Carry-less, integer, and dual-field adder (from top to bottom) on the right.

Table 1 shows the area evaluation of different hardware-multiplier types. We evaluated multipliers for prime-field, binary-field, and dual-field arithmetic for word sizes of 8, 16, 32, and 64 bits (on register-transfer level). For the evaluation we used the UMC-L130 CMOS technology where an AND gate needs 1.25 GEs, a XOR gate needs 2.75 GEs, and a full-adder cell needs 5.5 GEs.

Obviously the area requirement scales quadratically with the given word size and carry-less multipliers provide the lowest area footprint and lowest increase in area for all given word sizes. Runtime approximations for an ECC point multiplication showed that the word size of the carry-less multiplier must be at least 16 bits in order to achieve a sensible runtime.

Next to a carry-less multiplier, an integer multiplier is necessary to provide operations for higher-level protocols (*e.g.* ECDSA). Note that this multiplier is needed only very few times for most protocols (only four prime field multiplications are required for ECDSA signature generation, for instance). Thus, lower word sizes are acceptable since no significant reduction in speed is expected. We therefore decided to implement a 16-bit carry-less multiplier (to provide an appropriate speed for a point multiplication) and an 8-bit integer multiplier instead of a dual-field 16-bit multiplier (which needs 1,946 GEs). This would sum up to 1,226 GEs which is 720 GEs less than for a dual-field multiplier.

Table 1. Area evaluation of different hardware-multiplier types.

Finite Field	**Required adder cells per bit**	**8 bit [GE]**	**16 bit [GE]**	**32 bit [GE]**	**64 bit [GE]**
$GF(2^m)$	XOR	211	850	3,389	13,508
$GF(p)$	FA	376	1,616	6,688	26,336
Dual field	AND + FA	458	1,946	8,018	31,514

Table 2. Area evaluation of different 16×128-bit RAM architectures.

Type	**Port**	**Storage** [GEs]	**Logic** [GEs]	**Total** [GEs]
Std. cells (registers)	Single	10,281	2,941	13,926
Std. cells (latches)	Single	8,388	3,119	12,221
Macro S-RAM	Dual	-	-	6,737
Macro S-RAM[a]	Single	-	-	6,000
Macro register-file	**Single**	-	-	**2,955**

[a] Approximated based on UMC 180 nm technology.

3.2 The Memory Type and Architecture

One of the most area expensive chip components of ECC processors is the Random Access Memory (RAM). RAM is necessary to store intermediate values (*e.g.* point coordinates during point multiplication $k \cdot P$) and the secret scalar k. The size of the memory varies depending on the requirements of the ECC formulas (the formulas of López Dahab [27] need at least 5 registers of memory for full-precision architectures and 6 registers for multi-precision architectures due to the need of intermediate storage of in-place operations).

In Table 2, we compare different 16×128-bit RAM types concerning their area requirements. We compare standard-cell based implementations with dedicated RAM macro blocks synthesized in CMOS UMC-L130 technology. The standard-cell based RAM implementations (register and latch based) have been designed on RTL-level and synthesized using Cadence RTL compiler [5]. The RAM-macro blocks have been generated using the Standard Memory Compiler FSA0A Memaker 200901.1.1 by the Faraday Technology Corporation [12]. All except of one type of RAM provide a single read-port and a single write-port. There is one S-RAM macro that features a dual-port read/write interface.

It shows that the latch-based RAM is about 12 % smaller than the register-based RAM. This is because the size of a flip-flop is 5 GE and the size of a latch is 4 GE. This 25 % difference in area is debilitated because some additional registers and control logic is required so that the latch-based RAM works the same way as the register-based RAM. Adding a second read port to those RAMs would be relatively cheap in terms of chip area (it would require about 3,000 GEs in addition by introducing a second multiplexer at the output). Note that a dual-port memory would increase the performance of a multi-precision multiplication by a factor of about two.

From the two available single-port RAM macros, the register-file macro is about 50 % smaller than the S-RAM macro. The dual-port S-RAM macro, in contrast, is only 12 % larger than the single-port S-RAM macro, however, it is about 2.3 times larger than the register-file based RAM macro.

The register-file RAM macro provides best performance in our evaluation scenario. We performed several power simulations using Cadence Encounter and obtained similar results for the register-file RAM macro and the standard-cell

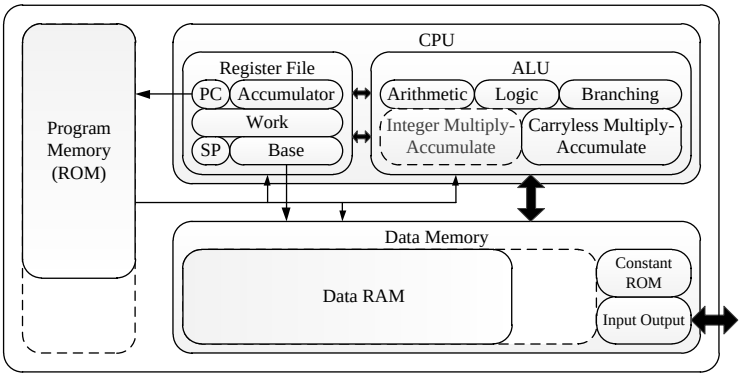

Fig. 2. High-level block diagram of the processor. Components for higher-level protocols are drawn with slashed lines (*i.e.* integer multiplier, program and data memory).

based RAM architectures. The main disadvantages of the register-file macro are the lack of a second read port (speed) and the limit of clock-synchronous read operations. The lack of a second read port can be compensated by using temporary working registers. The lack of an asynchronous read functionality can be balanced with a more difficult control logic.

4 Hardware Architecture

In this section, we introduce the hardware architecture of our processor. It is based on the microprocessor design called Neptun [34], which uses a Harvard architecture. This allows to fetch, decode, execute, and store data within the same clock cycle and allows low-area optimizations due to the choice of different memory types and sizes. Figure 2 shows the block diagram of the architecture. It is mainly composed of a Central Processing Unit (CPU) including register file and Arithmetic Logic Unit (ALU), and memories for program code, constants, and data.

4.1 Central Processing Unit (CPU)

The heart of the processor is the 16-bit CPU. It is composed of several internal registers and an ECC optimized ALU. The register file consists of a program counter (PC), a stack pointer (SP), three base registers, four working registers, and an accumulator register: The program counter is used as index for the program memory. The stack pointer (SP) is needed to store registers on the data memory. The stack is also used to store program-return addresses that are needed for function calls. In order to address certain base addresses within the data memory, three base registers are used. We integrated two source registers and one destination register. They are used together with a 4-bit offset to address data in the memory. The offset address is stored within a program word. We

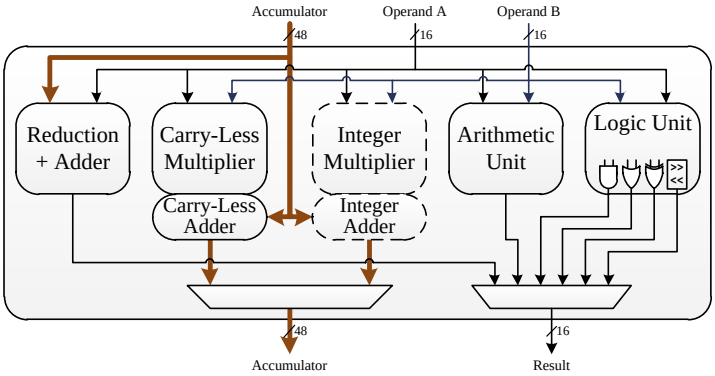

Fig. 3. High-level diagram of the arithmetic logic unit.

implemented four 16-bit working registers that can be used as general-purpose registers. The registers are needed for almost any ECC operation and are used to reduce the number of memory-read cycles within the finite-field multiplication. The accumulator register (ACC) is needed for the multiply-accumulate operation of the 163-bit multi-precision multiplication.

We integrated several optimizations to increase the performance of ECC operations. First, the ALU accesses data directly without loading it first into CPU registers (as it is in the case of conventional microcontrollers). In the first clock cycle, the data is addressed in the memory. In the second cycle, the data is processed by the ALU and the result is stored back in memory within the same clock cycle. This increases the performance of memory-access operations significantly. Second, loading and processing of data is done simultaneously by the processor. This avoids unnecessary idle cycles and improves the efficiency of multi-precision arithmetic operations. Those optimizations are described in more detail in [34].

Arithmetic Logic Unit (ALU). The arithmetic logic unit (ALU) mainly consists of a reduction-logic unit, a carry-less multiplier, an arithmetic unit (addition/subtraction), and a logic unit (supporting OR, AND, XOR, and shift operations). For higher-level protocols, an integer multiplier is needed in addition (drawn with dashed lines). Figure 3 shows a high-level diagram of the ALU. We also integrated an operand isolation technique for each submodule which reduces the power-consumption significantly.

4.2 Memory for Program, Data, and Constants

Our processor provides a long-term storage memory that mainly stores the program for ECC point multiplication. The memory provides 72 control signals and contains up to 1,800 entries depending on the implemented algorithms and higher-level protocols. Most of the control signals are used to control the dataflow

within the CPU. Best area results have been achieved by directly synthesizing the memory table as Read Only Memory (ROM) using standard cells. Experiments in which a 16-bit instruction set or a ROM macro have been introduced resulted in a larger area requirement.

For short-term data storage, we used a 16-bit RAM macro (register-file based) as discussed in Section 3. Note that in contrast to most processors reported in literature [4, 25, 26, 31], we include the number for the required storage of the secret scalar k. For an ECC point multiplication, 1,296 bits (81 entries) are necessary (we used a 16×84 macro in that case). For higher-level protocols, additional memory is needed (*e.g.* 1,536 bits for ECDSA signature generation (16×96 macro) and 2,384 bits for ECDSA signature verification (16×152 macro)).

ECC constants have been stored in a ROM. The ROM has been implemented as a look-up table and stores between 880 and 2,564 bits such as the x and y coordinate of the base point P, the ECC parameters a and b (see Equation (2)), and the irreducible polynomial $f(z)$.

The input/output of data has been realized via memory mapped I/O. Data can be written and read using a 16-bit parallel interface.

5 Implementation Details

In the following, we give details about the implemented carryless multiply-accumulate unit and the modular arithmetics in order to perform ECC operations.

5.1 Carry-Less Multiply-Accumulate Unit

The multi-precision multiplication over $\mathbb{F}_{2^{163}}$ has been realized following a multiply-accumulate (MAC) approach. There exist several publications that make use of MAC units to increase the performance of modular multiplication (see *e.g.* the work of [9, 14, 15, 17, 33]). We implemented the multiplication by a product-scanning form (often referred as Comba multiplication), where each partial product of $A[i] \cdot B[j]$ gets accumulated to a common sum (ACC_1, ACC_0), *i.e.* $(ACC_1, ACC_0) \leftarrow (ACC_1, ACC_0) + A[i] \cdot B[j]$.

Note that for the polynomial MAC unit the handling of carry propagation is not needed. Thus, the accumulator register needs a size of only $(2W - 1)$ bits.

We implemented several improvements to increase the performance. First, the entire multiplication algorithm has been unrolled so that no extra cycles are wasted for loop operations. Second, we reused the working registers as a memory cache to reduce the number of necessary load operations. With each working register used, the total number of read operations has been reduced by about $2N$. Third, we added a third word to the accumulator register (ACC_2, ACC_1, ACC_0) in order to allow efficient reduction of the accumulated sum. Thus, the MAC operation is performed on the words (ACC_2, ACC_1) instead of (ACC_1, ACC_0) and

Algorithm 1 Polynomial multiplication with interleaved reduction.

Require: Binary polynomials $a(z)$ and $b(z)$ of degree at most $m-1$.
Ensure: $c(z) = a(z) \cdot b(z) \bmod f(z)$.
1: $ACC \leftarrow 0$
2: **for** i **from** 0 **to** $N-1$ **do**
3: **for** each element of $\{(i,j)|i+j=k, 0 \leq i,j \leq N-1\}$ **do**
4: $(ACC_2, ACC_1) \leftarrow (ACC_2, ACC_1) + A[i] \cdot B[j]$.
5: **end for**
6: $C[k] \leftarrow ACC_1$.
7: $ACC \leftarrow ACC \gg W$.
8: **end for**
9: $ACC \leftarrow higher(ACC)$.
10: **for** k **from** t **to** $2N-2$ **do**
11: **for** each element of $\{(i,j)|i+j=k, 0 \leq i,j \leq t-1\}$ **do**
12: $(ACC_2, ACC_1) \leftarrow (ACC_2, ACC_1) + A[i] \cdot B[j]$.
13: **end for**
14: $C[k-N-1] \leftarrow C[k-N-1] + reduce(ACC)$.
15: $ACC \leftarrow ACC \gg W$.
16: **end for**
17: $C[N-1] \leftarrow lower(C[N-1]) + reduce(ACC)$.
18: $ACC \leftarrow ACC \gg W$.
19: $C[0] \leftarrow C[0] + reduce(ACC + higher(C[N-1])) \gg W$.
20: $C[N-1] \leftarrow lower(C[N-1]) \gg W$.
21: **Return**(c).

ACC_0 is used to store the previous intermediate result. A detailed description of the reduction method is given in the following subsection.

Algorithm 1 shows the algorithm of the implemented polynomial multiplication. The polynomials $a(z)$ and $b(z)$ get multiplied and the reduced result is stored in $c(z)$. In the lines 1 to 8, the lower N words of the result $c(z)$ are calculated. Note that in this phase the ACC_0 register is not used. In line 9, the lower $(m-W(N-1))$ bits of the accumulator need to be cleared. Those are the bits of the results that do not need to be reduced. The lines 10-16 calculate the higher N words of $c(z)$ and reduce them immediately. According to the recommended NIST irreducible polynomial B-163 $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$, the reduction function (line 14) can be written as

$$reduce(ACC) = \Big(ACC \gg (W+3) + ACC \gg W + ACC \gg (W-3) + ACC \gg (W-4)\Big) \wedge (2^W - 1). \quad (3)$$

Finally, in lines 17-20 the rest of the accumulator and the higher bits of $C[N-1]$ get reduced.

Polynomial NIST B-163 Reduction Logic. We make use of the recommended NIST irreducible polynomial B-163 to perform a very efficient modular

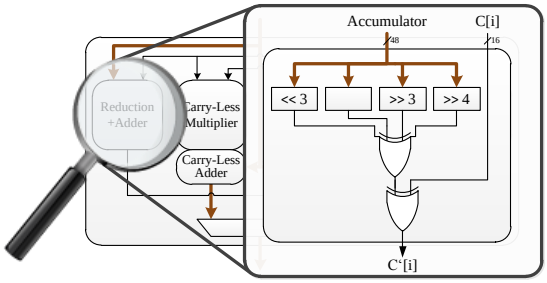

Fig. 4. Using the dedicated reduction logic, the content of the accumulator is reduced and stored in $C'[i]$ (hold in data memory) with index i.

reduction for modular multiplication and squaring. The reduction logic is shown in Figure 4. We hard-wired the output of the appropriate accumulator register according to Equation (3). The reduction logic takes the output of the 48-bit accumulator register, performs 4×16 XOR operations and the result is added with the intermediate result $C[i] = C[k - N - 1]$ (see line 14 in Algorithm 1). After the addition (XOR), the variable $C[i]$ is updated with $C'[i]$ in the data memory. Only one clock cycle is needed to reduce the intermediate result of the accumulator and sum of partial products, respectively. Figure 4 shows the dedicated reduction logic.

It should be noted that although the reduction logic has been specially optimized for NIST B-163, the CPU is capable of handling arbitrary irreducible polynomials. Thus requirements such as flexibility and extendability are ensured.

5.2 Modular Arithmetic

Modular Addition. The simplest operation is the modular addition. It is a simple XOR operation. Neither a carry flag nor a finite-field reduction need to be considered. Modular addition over $\mathbb{F}_{2^{163}}$ needs 35 clock cycles on our processor.

Modular Multiplication. Modular multiplication has been realized using the carryless multiply-accumulate unit described in Section 5.1. Our processor needs 222 clock cycles for a 163-bit multiplication.

Modular Squaring. Modular squaring can be performed very efficiently. The binary representation of the polynomial can be easily squared by inserting a 0 between each consecutive bit of the polynomial, e.g. $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$ would results in $a(z)^2 = a_{m-1}z^{2m-2} + \cdots + a_2z^4 + a_1z^2 + a_0$. This can be realized with only a few additional hardware components. The polynomial-reduction logic can be reused for squaring. One modular squaring needs 41 clock cycles on our processor and thus is 5.4 times faster than a modular multiplication.

Modular Inversion. Modular inversion is required to transform the projective coordinates back into affine. For this operation, we made use of Fermat's

Table 3. Size and power estimations of our processor for different CMOS technologies using Latch-based RAMs.

Technology	Area $[\mu m^2]$	NAND Gate $[\mu m^2]$	Total Area $[GE]$	Power $[\mu W$@1MHz$]$	Leakage $[\mu W]$
AMS c35b4	693,948	54.600	12,710	696.3	0.63
UMC f180GII	139,469	9.374	14,878	107.1	0.53
UMC f130SP	71,745	5.120	14,013	31.4	1.37
UMC f090SP	39,550	3.136	12,612	70.1	54.32

little theorem [20] that states that $a = a^{2^m} \bmod f(z) \, \forall a \in \mathbb{F}_{2^m}$. As a result, $a^{-1} \equiv a^{2^m-2} \bmod f(z)$. This exponentiation can be performed using 162 squaring and only 9 multiplications for the NIST B-163 binary field. As a result 11,031 cycles are needed for an inversion.

6 Results

We synthesized our processor using different CMOS technologies from various manufacturers. For synthesis, we used the Cadence RTL compiler [5] Version v08.10. Table 3 shows the total area and power-consumption estimation of the processor using latch-based RAMs[1] (described in Section 3.2). The power-consumption estimations were made using Cadence Encounter Version v08.10. All obtained area results are within a 20 % margin. In view of power consumption, best performance had been obtained for the UMC-L130 technology. For all following approximations we used register-based RAM macros.

In Table 4, the area and power requirements for individual chip components are listed. The memory needs most of the area which is 5,399 GEs. The CPU needs 3,556 GEs in total where only 849 GEs are used for the carry-less multiplier. The total size of the processor sums up to 8,958 GEs.

In Table 5, we compare our results with related work. There exist many publications of ECC processors over $\mathbb{F}_{2^{163}}$. Most of those processors use full-precision arithmetic to perform the point multiplication. For a fair comparison, we listed the results of the authors for different digit sizes (d=1...8). All implementations need between 10,392 GEs and 16,247 GEs of chip area and between 47 and 430 kcycles for the computation of $k \cdot P$. Our implementation needs 8,958 GEs of area which is 1,434 GEs less area than the best reported solution. This is an area improvement by about 14 %. The number of needed clock cycles can be compared with the full-precision solutions with d=1. The power and energy consumption is very low and fulfills most requirements of embedded-system designs.

6.1 Results for Higher-Level Protocol Implementations

As a higher-level protocol, we implemented the Elliptic Curve Digital Signature Algorithm (ECDSA) [30]. In addition to a point multiplication over the binary

[1] We did not have access to RAM macros for all those technologies.

Table 4. Size and power consumption of individual chip components.

Component	Area [GE]	Area [%]	Power [μW@1MHz]	Power [%]
Memory	5,399	60.27	11.57	35.77
Program memory	2,471	27.58	4.24	13.10
Data RAM	2,528	28.22	4.66	14.41
Constant ROM	256	2.56	1.62	5.01
CPU	3,556	39.70	18.93	58.54
ALU	1,837	20.51	11.05	34.16
Carry-less multiplier	849	9.48	2.30	7.12
Logic unit	348	3.88	2.15	6.65
Arithmetic unit	93	1.04	0.37	1.15
Register Set	875	9.77	1.48	4.58
Total Area	**8,958**	**100.00**	**32.34**	**100.00**

field $\mathbb{F}_{2^{163}}$, ECDSA needs a hash function and several prime-field arithmetic operations to generate and verify a digital signature. As a hash function, we implemented the 160-bit SHA-1 algorithm according to ISO/IEC FIPS-180-3 [29]. Replacing the SHA-1 algorithm with one of the current SHA-3 candidates [19] would be easily possible. For prime-field multiplications and inversion, we decided to implemented Montgomery-arithmetic operations. We implemented the Finely Integrated Product Scanning Form (FIPS) according to Koç et al. [24]. The algorithm is used only four times, so we optimized the code for low area (no

Table 5. Comparison with related work.

Related Work	Area [GE]	Cycles [kCycles]	Power [μW@1MHz]	Energy [μJ]	CMOS Technology
Kumar06 d=1 [25]	15,094	430	-	-	AMI C35
Batina06[a] d=4 [2]	14,816	95	27.00	2.57	130 nm
Batina06[a] d=3 [2]	14,258	125	27.00	3.38	130 nm
Batina06[a] d=2 [2]	13,681	182	27.00	4.91	130 nm
Batina06[a] d=1 [2]	13,104	354	27.00	9.56	130 nm
Bock08 d=8 [4]	16,247	47	148.76	6.99	INF SRF55V01P
Bock08 d=4 [4]	12,876	80	93.27	7.46	INF SRF55V01P
Bock08 d=1 [4]	10,392	280	54.31	15.21	INF SRF55V01P
Lee08 d=4 [26]	15,356	79	37.39	2.95	UMC L130
Lee08 d=3 [26]	14,729	101	38.32	3.87	UMC L130
Lee08 d=2 [26]	14,064	145	36.52	5.30	UMC L130
Lee08 d=1 [26]	12,506	276	32.42	8.95	UMC L130
Hein08 16-bit [17]	11,904	296	101.87	30.15	UMC L180
This work 16-bit	**8,958**	**286**	**32.34**	**9.25**	**UMC L130**

[a] For a fair comparison a RAM approximated with 4,890 GE was added. The power values lack the power consumption of this RAM.

Table 6. Area and power estimations of our processor supporting ECDSA.

Program	**Area** [GE]	**Cycles** [kCycles]	**Lines of Code**	**Power** [μW@1MHz]	**Energy** [μJ]
ECC Only	8,958	294	637	32.09	9.43
ECC Protected[a]	9,728	298	828	32.48	9.68
ECDSA Sign[a,b]	15,387	378	1771	41.11	15.54
ECDSA Verify[b]	16,005	605	1784	40.76	24.66

[a] The numbers include y-recovery, randomized projective coordinates (RPC) side-channel countermeasure [6], and ECC point-validity check [8].

[b] Includes the SHA-1 hash function [29], Random Number Generation (RNG) [30], and prime-field arithmetics.

loop unrolling etc.). Furthermore, we implemented the Montgomery-inversion algorithm according to Kalinski et al. [22].

For signature verification, we applied Shamir's trick [7, 10] to improve the performance of multiple-point multiplication. All described operations for ECDSA have been implemented as Assembler functions for our processor and have been stored in program memory. Table 6 shows the results after synthesizing the processor. For ECDSA signature generation, our processor needs 15,387 GEs which outperforms existing solutions in terms of area, power, and speed [13, 18, 34, 35]. Signature verification can be realized using a chip area of 16,005 GEs.

7 Conclusions

In this paper, we presented a low-resource implementation of an ECC hardware processor. The processor needs 8,958 GEs and performs a point multiplication within 285 kcycles. The power consumption is about 3.2 μW at 100 kHz. We met the low-resource constraints of embedded systems by applying a very modular microcontroller architecture that allows the execution of higher-level protocols like ECDSA. The elliptic-curve operations have been performed over the NIST $\mathbb{F}_{2^{163}}$ elliptic curve using multi-precision arithmetic. The outcome improves the state of the art in low area ECC hardware designs and provides even a smaller area footprint than most of the proposed SHA-3 candidates [19].

Acknowledgements

We would like to thank Mario Kirschbaum and Martin Feldhofer for several fruitful discussions.

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. R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2005.
2. L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede. Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks. In L. Buttyán, V. Gligor, and D. Westhoff, editors, *Security and Privacy in Ad-Hoc and Sensor Networks – ESAS 2006, Third European Workshop, Hamburg, Germany, September 20-21, 2006, Revised Selected Papers*, volume 4357, pages 6–17, Berlin Heidelberg, 2006. Springer-Verlag.
3. I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, Cambridge, UK, 1999.
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. Cadence Design Systems. The Cadence Design Systems Website. `http://www.cadence.com/`.
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. P. de Rooij. Efficient Exponentiation using Procomputation and Vector Addition Chains. In A. D. Santis, editor, *Advances in Cryptology EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 389–399. Springer Berlin / Heidelberg, 1994.
8. 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.
9. H. Eberle, N. Gura, S. C. Shantz, V. Gupta, and L. Rarick. A Public-key Cryptographic Processor for RSA and ECC. In *Proceedings of the 15th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2004)*, pages 98–110. IEEE Computer Society, September 2004.
10. T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology - CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.
11. 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.
12. Faraday Technology Corporation. Faraday FSA0A_C 0.18 μm ASIC Standard Cell Library, 2004. Details available online at `http://www.faraday-tech.com`.
13. 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.

14. J. Großschädl. Full-Custom VLSI Design of a Unified Multiplier for Elliptic Curve Cryptography on RFID Tags. In F. Bao, M. Yung, D. Lin, and J. Jing, editors, *Information Security and Cryptology - 5th Internationa Conference, Inscrypt 2009, Beijing, China, December 12-15, 2009. Revised Selected Papers*, volume 6151 of *Lecture Notes in Computer Science*, pages 366–382. Springer, 2011.
15. J. Großschädl and G.-A. Kamendje. Optimized RISC Architecture for Multiple-Precision Modular Arithmetic. In D. Hutte, G. Müller, W. Stephan, and M. Ullmann, editors, *Security in Pervasive Computing - SPC 2003*, volume 2802 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2003.
16. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004.
17. D. Hein, J. Wolkerstorfer, and N. Felber. ECC is Ready for RFID - A Proof in Silicon. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, Canada, August 14-15, 2008, Revised Selected Papers*, Lecture Notes in Computer Science (LNCS), September 2008.
18. 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.
19. IAIK. Hash Function Zoo. `http://ehash.iaik.tugraz.at/index.php/HashFunctionZoo`.
20. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronic Letters*, 24(6):334–335, March 1988.
21. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2003.
22. B. Kaliski. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.
23. N. Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994. ISBN 0-387-94293-9.
24. Ç. K. Koç, T. Acar, and B. S. K. Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3):26–33, June 1996.
25. 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.
26. 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.
27. J. López and R. Dahab. Fast Multiplication on Elliptic Curves over $\mathrm{GF}(2^{\mathrm{m}})$ without Precomputation. 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 316–327. Springer, 1999.
28. P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, January 1987. ISSN 0025-5718.

29. National Institute of Standards and Technology (NIST). FIPS-180-3: Secure Hash Standard, October 2008. Available online at `http://www.itl.nist.gov/fipspubs/`.
30. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009. Available online at `http://www.itl.nist.gov/fipspubs/`.
31. E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. 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 92–106. Springer, August 2004.
32. K. Sakiyama, L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede. Small-footprint ALU for public-key processors for pervasive security. In *Workshop on RFID Security 2006 (RFIDSec06), July 12-14, Graz, Austria*, 2006.
33. S. Tillich and J. Großschädl. VLSI Implementation of a Functional Unit to Accelerate ECC and AES on 32-bit Processors. In C. Carlet and B. Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 40–54. Springer, June 2007.
34. 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.
35. J. Wolkerstorfer. Is Elliptic-Curve Cryptography Suitable for Small Devices? In *Workshop on RFID and Lightweight Crypto, July 13-15, 2005, Graz, Austria*, pages 78–91, 2005.

A Statistics for ECC multiplication

During the development of the ECC and ECDSA functions we used a statistics feature of our tool-chain to investigate the code-line and cycle consumption of each function. Table 7 shows the number of times each function is called, the size of each function in code lines and the total runtime of each function. Even though the multiplication algorithm is optimized down to 222 cycles it still covers 74 % of the total runtime.

Table 7. Functions used during ECC point multiplication with y-recovery and point-validity check.

Function	Calls	Code Lines	Cycles
B163.Multiplication	990	222	219,780
B163.Square	969	41	39,729
B163.Add	490	35	17,150
PointOperation.Multiplication	1	148	16,636
B163.FermatInverseHelp	7	31	2,041
Utilities.Copy	16	24	384
PointOperation.yRecovery	1	90	90
B163.FermatInverse	1	88	88
PointOperation.isValidPoint	1	44	44
Utilities.CMP	1	35	35
Utilities.Clear	2	13	26
TOTAL	**2,479**	**771**	**296,003**
TOTAL including test functions	**2,480**	**828**	**296,547**

Table 8 shows how often each and every type of instruction is used. The parallelized commands are a combination of other commands. They cover 71 % of the total runtime. Note that only 4.4 % of the total runtime is used for program-flow instructions such as RET, CALL, BRA, and JMP. This overhead would not exist if a dedicated state machine instead of a CPU with instruction set would be used.

Table 8. Instructions used during an ECC point multiplication with y-recovery and point-validity check.

Mnemonic	Description	CPI	Cycles	Used
PAR: BMULACC | LD		1	109,869	111
PAR: MOVNF | LD		1	65,707	83
LD	Load from memory	1	35,410	65
PAR: BREDUCE_ADD_ST | BRSACC		1	13,818	14
PAR: BMULACC | ST | BRSACC		1	11,859	12
PAR: BREDUCE_ADDBYTE_ST | BRSACC		1	9,690	10
CALL	Call a function	3	7,440	70
LDI	Load Immediate	1	6,900	105
AND	Logic AND	1	6,038	7
PAR: XOR | ST		1	5,390	11
MOVNF	Copy register to register without flag update	1	5,269	47
RET	Return from function	2	4,960	13
BMULACC	Binary multiply-accumulate	1	3,876	4
STR	Store a register to memory	1	2,725	32
XOR	Logic XOR	1	2,120	3
LDR	Load from memory and store to register	2	1,942	10
ADDI	Add with carry	1	573	11
BRA	Branch if flag is set/cleared	1	488	6
PUSH	Push a value to the stack	2	338	5
POP	Pop a value from the stack	2	336	4
LSI	Left shift by immediate	1	326	4
SUBI	Subtract with carry	1	324	6
ADD	Add	1	163	2
RS	Right shift	1	163	2
RSI	Right shift immediate	1	163	2
SUB	Subtract	1	163	2
ASRI	Arithmetic shift right	1	161	1
JMP	Jump to address	1	160	1
CMP	Compare	1	84	2
MOV	Copy register to register	1	82	1
CMPC	Compare with carry	1	10	10
TOTAL			**296,547**	**656**