Hardware Architectures for MSP430-based Wireless Sensor Nodes Performing Elliptic Curve Cryptography

Erich Wenger

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8010 Graz, Austria
erich.wenger@iaik.tugraz.at

Abstract. Maximizing the battery lifetime of wireless sensor nodes and equipping them with elliptic curve cryptography is a challenge that requires new energy-saving architectures. In this paper, we present an architecture that drops a hardware accelerator between CPU and RAM. Thus neither the CPU nor the data memory need to be modified. In a detailed comparison with a software-only and a dedicated hardware architecture, we show that the drop-in concept is smaller than the dedicated hardware module, while achieving similarly fast runtimes. Most interesting for micro-chip manufacturers is that only 4 kGE of chip area need to be committed for the dedicated drop-in accelerator.

Keywords: MSP430, ASIC, Hardware, Software, Elliptic Curve Cryptography, Wireless Sensor Nodes.

1 Introduction

Privacy, authenticity, and confidentiality pose three of the most challenging current demands on wireless sensor networks. To solve those requirements the use of cryptography is essential. Unfortunately, it is hardly possible to solve this challenge using only symmetric cyphers. The most promising solutions are based on asymmetric cryptography, in particular Elliptic Curve Cryptography (ECC).

Efficiently implementing ECC is a complex task, especially when a designer also needs to be aware of the capabilities of the entities of a sensor network: A sensor node usually comes with a microprocessor, a sensor (e.g., for humidity), a wireless communication interface (e.g., IEEE 802.15.4 [16], ZigBee [31]), and a battery, which should keep the sensor-node alive for a lifetime (some years) within a hostile environment. This means that a solution to the initial requirements should be light-weight and efficient. For maximizing the battery live and keeping the price of a sensor node at a minimum, ECC has to be implemented with care. To realize the scope of the difficulty, be aware that within the time

required for a single elliptic-curve point multiplication, several hundreds of symmetric encryptions and decryptions can be preformed. Thus ECC has a major impact on both communication latency and energy consumption.

A lot of research has been focused on efficiently and securely implementing ECC. The research is performed based on three different approaches: one is based on efficiently implementing ECC in software, one is based on adding dedicated hardware, and one is a combination of the two preceding approaches. Several papers discuss the use of assembly optimizations [12], instruction-set extensions [6, 10], and dedicated ECC hardware designs [19, 20]. The drawback of those techniques are the relatively low performance, the requirement to change the microprocessor, and the potential waste of precious chip area, respectively. As CPU vendors usually do not give away the source code of microprocessors, but obfuscated code instead, adding new instructions is a troublesome task. Dedicated hardware modules provide locally optimized solutions, but ignore the existence of already available hardware modules. Our paper fills this gap.

Our contribution. In this paper, we perform a fair comparison (common algorithms, technologies, tools) of three different hardware architectures, all capable of performing ECC. Using an openMSP430 at the core, we present (i) an area and speed-optimized software solution, (ii) a dedicated hardware module, and most importantly (iii) a novel ECC 'drop-in' architecture. For the drop-in architecture, a lightweight ECC accelerator is placed right between the CPU and its data memory. It requires less chip area than a dedicated hardware module, while being similarly fast. Compared to the optimized software solution, the energy consumption is reduced by a factor of 28, which certainly will make a major impact on the lifetime of a wireless sensor node. The drop-in concept is also most interesting for micro-chip manufacturers as only 4 kGE of dedicated chip area need to be committed for the drop-in accelerator.

The paper is structured as follows. Section 2 gives a short introduction on how to securely implement ECC and Section 3 discusses different architectures for ECC. The most promising architectures are then implemented within Sections 4–6 and compared within Section 7. Conclusions are drawn within Section 8.

2 A Short Introduction to ECC

Elliptic curves, used for cryptography, are built on top of finite fields. As finite field, one can either choose a prime field or a binary extension field. Prime fields are fast in software as they are based on integers and integer multipliers are available in nearly all (embedded) microprocessors. Binary-extension fields on the other hand are built on polynomials, which when implemented in hardware do not have the drawback of carry propagation. However, in software a multiplication of two polynomials has to be realized using branches, which are vulnerable to side-channel attacks.

The for us most interesting standardized elliptic curves [1, 2, 23] are all based on the Weierstrass equation: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$. Depending on whether prime or binary-extension fields are used, this equation is simplified

Table 1. ECC formulas used within this paper.

			Finite-field operations per key bit		
Formula	**Field**	**Registers**	**Add/Subtract**	**Square**	**Multiply**
Hutter *et al.* [15]	$\mathbb{F}_p$	$7 + 3 = 10$	17	4	12
López and Dahab [22]	$\mathbb{F}_{2^m}$	$5 + 3 = 8$	3	5	6

to $y^2 = x^3 + ax + b$ or $y^2 + xy = x^3 + ax^2 + b$, respectively. Also the formulas used to perform point additions and doublings depend on the used finite field. For further information, the reader is referred to standard literature on elliptic curves [3, 13].

For the following comparison, it is important that all implementations are based on a common methodology. For the constant-runtime software implementations, the integer and polynomial arithmetic has been separated from the reduction operation. The reduction is performed using only simple shift and addition operations. Thereby advantage was taken of the used prime and irreducible polynomial. To perform an inversion in constant time, an exponentiation, based on Fermat's little theorem ($a^{q-2} \equiv a^{-1} \bmod (q)$) is used. For binary-extension fields an optimized inversion algorithm based on Itoh and Tsujii [17] is used.

More important than the used finite field is that ECC implementations are vulnerable to side-channel attacks [7]. Attackers can use runtime information, power consumption profiles, or induce faults to recover the secret key. This is a significant problem for the easily accessible wireless sensor nodes that usually are deployed within unsafe environments. Thus, a methodology must be utilized that minimizes the potential threats.

In this paper we take advantage of differential addition formulas optimized for Montgomery ladders. Table 1 gives a short summary of the used formulas. By using a Montgomery ladder, the underlying finite-field operations are independently performed from the used private scalar. Thus a key-independent constant runtime is achievable under the assumption that all finite-field operations are performed in constant time (which they are). The formulas are also lightweight. Only 7/5 registers are required during the point double-and-add operations. For the recovery of the y-coordinate another two registers are needed which store the original base point. Another register that stores the private scalar is also included in all comparisons within this paper.

To further increase the resistance against power-analysis attacks one would use Randomized Projective Coordinates [5] and to resist fault attacks, perform point verifications before and after each point multiplication. In practice the resistance against those attacks is verified by performing real-world evaluations. As those evaluations would go beyond the scope of this paper, they have not (yet) been done. However, the algorithms and methodologies used for our implementations are applicable to build real-world secure hardware.

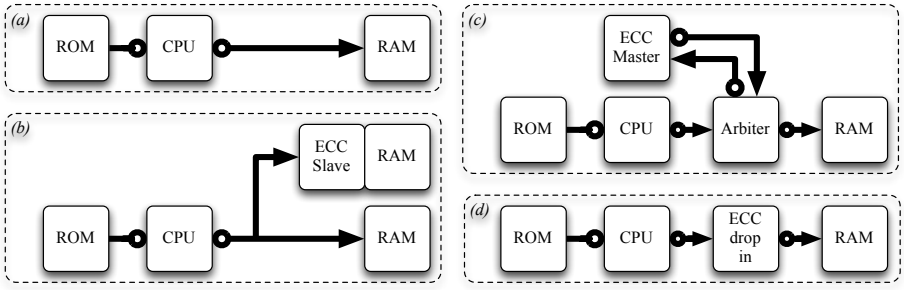

Fig. 1. Microprocessor-based architectures.

3 Architectures

The decision regarding the best architecture is most important for a final design as it greatly influences area, runtime, power, and energy characteristics. Only by considering all requirements and the system as a whole, a *global optimum* can be found. By optimizing a single (e.g., ECC) component it is probable to reach a *local optimum* only. Figure 1 shows four different architectures which are based on microprocessors, as microprocessors are the central component in all currently available sensor nodes. The ECC-independent components, such as the wireless interface and the actual sensor, are considered to be constants and therefore independent from the used architecture.

(a) The most straight-forward solution is to perform and optimize ECC in software[1]. The hardware designer only has to make sure that the data memory is sufficiently large and the assembly-optimized ECC code is placed within the program memory. The microprocessor (CPU) is then used to execute the code. In Figure 1, the program memory is simplified as ROM and the data memory as RAM. As ECC is very resource demanding and a software-only solution is in most cases insufficiently slow, one could add a memory-mapped ECC co-processor.

(b) Co-processors have already been extensively studied and optimized in related work [19, 20]. However, comparing area and power results of designs that use different technologies and tools is inaccurate. Therefore, Section 5 presents an ECC co-processor on-par with related work.

The drawback of so-called ECC slaves is that they waste chip area by having their own memory. A solution in which the global RAM is reused is preferable. Even when area-efficient RAM macros are used, practical evaluations show that one RAM macro with more entries is smaller than two RAM macros with fewer entries (c.f. 128×8-bit: 2,073 GE vs. 256×8-bit: 2,897 GE). An ECC accelerator without RAM, which only performs finite-field operations would be a solution. Unfortunately, for this solution the CPU has to manually move operands from the RAM to the ECC slave and vice versa, thus wasting potential performance.

[1] Section 4 discusses this solution.

Table 2. HW synthesis of openMSP430 [24].

Functional Blocks	**Chip-Area** [GE]
openMSP430	7,801
Execution unit	5,536
Register file	2,709
ALU	693
Multiplier	1,826
openMSP430 w/o Multiplier	5,958

Algorithm 1 Accessing the 16-bit memory-mapped multiplier.

```
1: MOV R4, &MPY
2: MOV R5, &OP2
3: NOP
4: MOV @RESLO, R6
5: MOV @RESHI, R7
```

(c) An ECC circuit which, like the CPU, is capable of accessing the global data memory by itself solves that problem: an ECC bus master. This assumes that the used microprocessor must support a multi-master scenario, which embedded light-weight microprocessors usually do not. Also, the required arbiter can have a significant impact on the total chip area.

(d) A more sophisticated concept is to unite the ECC master with the arbiter. This within the context of ECC novel concept "drops" an ECC accelerator right between the CPU and the data memory. From the viewpoint of the CPU it behaves as simple ECC slave and does not hinder any access to the data memory. From the viewpoint of the drop-in module, direct access to the data memory is possible. Advantageous is also that neither the CPU (compared to instruction-set extensions) nor the data-memory need to be modified. Section 6 discusses this solution in more detail.

Tools. For this paper we use the 130 nm low-leakage ASIC technology by UMC with the Faraday design libraries in combination with area-efficient single-port register-based RAM macros. For hardware synthesis Cadence RTL Compiler v08.10, for place-and-route and power simulation Cadence First Encounter v08.10, and for simulation Cadence NCSim v08.20 are used. In this technology, one gate equivalent is equal to 5.12 μm^2. All evaluations are performed at 1 MHz and can easily be synthesized to exceed an operating frequency of 50–100 MHz.

4 ECC on openMSP430

At the core of all previously discussed hardware designs is a microprocessor. The selection of an appropriate microprocessor crucially influences the final runtime, chip area, power, and energy results. The MSP430 [27] developed by Texas Instruments, is considered to be a role model when it comes to low-cost and low-power applications. It is currently already used for the sensor-node platforms BEAN, COOKIES, EPIC mode, PowWow, Shimmer, TelosB, T-Mote Sky, and XM1000, just to name a few. The MSP430 is a 16-bit RISC processor with a Von Neumann architecture. This is important for saving data memory, as constants do not have to be loaded to the expensive RAM before they are used. The

Algorithm 2 $ACC \leftarrow ACC + (A[0] \times B[2]) + (A[1] \times B[1]) + (A[2] \times B[0])$.

```
 1: ADD   #4   ,  OPB                 9: MOV  @OPA   ,  &MAC
 2: MOV   @OPA+, &MPY                10: MOV  @OPB   ,  &OP2
 3: MOV   @OPB , &OP2                11: SUB  #4     ,   OPA
 4: DECD   OPB                       12: ADD  @RESLO ,   ACC0
 5: MOV   @OPA+, &MAC                13: ADDC @RESHI ,   ACC1
 6: MOV   @OPB , &OP2                14: ADDC @SUMEXT,   ACC2
 7: DECD   OPB                       15: MOV   ACC0  , 4(DEST)
 8: ADD @SUMEXT,  ACC2               16: CLR   ACC0
```

MSP430 comes with 16 16-bit registers, where `R0` is the program counter, `R1` is the stack pointer, `R2` is the status register, and `R3` is the constant-generator register. So only 12 registers (`R4`–`R15`) are useable as general-purpose registers. The MSP430 comes with only 27 instructions, from which none is a multiplication instruction. To perform a 16-bit integer multiplication, the MSP430 optionally has a memory-mapped multiplier. This will be discussed in detail later.

4.1 openMSP430

As our desired goal is a microprocessor-based hardware design, we need a hardware model of the MSP430. Olivier Girard programmed a synthesizable Verilog clone of the MSP430, called openMSP430 [24]. This clone fully supports the instruction set of the original MSP430 (with nearly identical timings), interrupts, and power-saving modes. It optionally comes with a 16×16-bit hardware multiplier, watchdog, timer, and GPIOs. A first evaluation of this core is depicted in Table 2. An openMSP430 without data or program memory (which will be chosen appropriately) requires 7,801 GE. Most of this chip area is spent on the execution unit (71 %), and the hardware multiplier (23 %). Without the multiplier, which is not necessary for binary-field based ECC, the openMSP430 only requires 5,958 GE.

4.2 Integer Arithmetic

In order to perform a 16-bit integer multiplication, four memory accesses are necessary. Algorithm 1 shows the assembly code necessary to multiply `R4` with `R5` and to store the product in `R6` and `R7`. The code shown in Algorithm 1 needs $4 + 4 + 1 + 2 + 2 = 13$ cycles to complete.

As multiple words are needed to represent integers within the used finite field, the multi-precision product-scanning multiplication technique of Comba [4] is used. Algorithm 2 sketches the used methodology. Three registers are used to hold pointers to the operands (`OPA` and `OPB`) and the result (`DEST`), three registers for the accumulator (`ACC0-2`) and three registers to hold addresses of the memory-mapped multiplier (`RESLO`, `RESHI`, and `SUMEXT`). In order to avoid loading the product after each multiplication, multiply-accumulate operations

Algorithm 3 64 × 1-bit polynomial multiplication.

```
1: RLA B0          8: XOR #0, C0
2: JNC +10         9: XOR #0, C1
3: XOR A0, C0     10: XOR #0, C2
4: XOR A1, C1     11: XOR #0, C3
5: XOR A2, C2     12: NOP
6: XOR A3, C3     13: NOP
7: JMP +12
```

Table 3. Comparison with related work.

Curve	Type	ROM [Bytes]	RAM [Bytes]	Runtime [kCycles]
Gouvêa *et al.* [9] and Szczechowiak *et al.* [26]				
`secp160r1` [9]	$\mathbb{F}_p$	23,300	2,800	2,528
`sect163k1` [9]	$\mathbb{F}_{2^m}$	27,800	3,600	2,032
Custom [26]	$\mathbb{F}_p$	31,300	2,900	5,898
`sect163k1` [26]	$\mathbb{F}_{2^m}$	32,100	2,800	8,519
ours on MSP430				
`secp160r1`	$\mathbb{F}_p$	4,230	282	5,721
`sect163r2`	$\mathbb{F}_{2^m}$	4,126	294	7,447

are performed directly within the memory-mapped multiplier. The overflowing bit stored within the `SUMEXT` register needs to be loaded within line 8. After line 14, the accumulated product resides within the registers `ACC0-2`. This technique has already been presented by Gouvêa and López [8].

4.3 Polynomial Arithmetic

As the MSP430 lacks a carry-less multiplier, a polynomial multiplication has been implemented using branch operations. Algorithm 3 shows a 64×1-bit polynomial multiplication which was used to build a 64×32-bit multiplication. The 64×32-bit multiplication can be performed without the use of a single, costly memory load or store operation. Using the methodology of Karatsuba and Ofman a three-way split of a single 192-bit multiplication to 6 64-bit multiplications has been performed. On the MSP430 a 64×32-bit polynomial multiplication takes 383 cycles and a 192-bit polynomial multiplication takes 6,089 cycles. For constant runtime, lines 8-13 in Algorithm 3 perform dummy operations. Without the dummy operations, a speedup of 23 % is possible on average. For comparison, a 192-bit integer multiplication takes 2,254 cycles and therefore is 2.7 times faster. Gouvêa *et al.* [9] report an assembly optimized implementation for `sect163k1` which only needs 3,907 cycles, but their implementation is not safe from timing attacks.

4.4 Software Results

Four standardized elliptic curves providing security-levels of 80–96 bits have been implemented. `secp192r1` and `sect163r2` are chosen because they are the smallest elliptic curves within the NIST standard [2, 23], still providing a sufficient level of security. `secp160r1` has been chosen because it is popularly used within related work. As `c2tnb191v1` [1] provides a similar security level as `secp192r1` (95 vs 96 bits) it can be used for comparison.

Note that Table 3 shows the runtimes of our software implementation, simulated on a cycle-accurate model of the MSP430, while Table 4 shows the slightly

Table 4. Synthesized software implementations of ECC on the openMSP430.

Curve	Type	Security	ROM	RAM	ROM	RAM	Area	Runtime	Power	Energy
		[Bits]	[Bytes]	[Bytes]	[GE]	[GE]	[GE]	[kCycles]	[µW]	[µJ]
`secp160r1`	$\mathbb{F}_p$	80	4,230	282	5,907	3,175	16,638	5,445	55.9	304.3
`secp192r1`	$\mathbb{F}_p$	96	4,846	322	6,173	3,400	17,128	8,650	53.9	466.7
`sect163r2`	$\mathbb{F}_{2^m}$	81	4,126	294	5,737	3,275	14,167	7,217	49.1	354.3
`c2tnb191v1`	$\mathbb{F}_{2^m}$	95	3,994	310	5,735	3,375	14,014	8,376	55.4	463.8

better runtimes for an openMSP430. In Appendix A a detailed comparison of all software implementations is depicted.

In literature many speed-optimized ECC implementations for the MSP430 have been reported [9, 21, 26, 28] (cf. Table 3). Because of the extentsively performed assembler optimization, our software implementation outperforms the related work of Szczechowiak *et al.* [26] that also requires larger memories. The fastest (ECDSA) implementation was done by Gouvêa *et al.* [9] in 2012. Compared to our implementations, they report twofold faster runtimes at the expense of 7 times larger program and 12 times larger data memories. As we synthesize the program memory and choose appropriately large RAM macros, their implementation would result in a significantly larger hardware design, compared to ours.

Table 4 shows the measured chip area, runtime, power, and energy results for the four implemented elliptic curves. The biggest impact of up to 60 % on the total chip area is due to the size of the program memory and data memory. For the elliptic curves over $\mathbb{F}_{2^m}$ the integer multiplier has been removed. The binary-field-based ECC implementations are about 16 % smaller and similarly fast, compared to the prime-field-based ECC implementations. For `sect163r2` the used 176-bit polynomial multiplier which is based on the 192-bit multiplication algorithm discussed before, renders the runtime results inferior compared to `secp160r1`.

The elliptic curve requiring the least amount of energy is `secp160r1` (303.3 µJ). However, the biggest potential for hardware optimizations (cf. [29]) lies within binary-field based elliptic curves (354.3 µJ). Therefore `sect163r2` alias NIST B-163 has been selected for the following hardware implementations.

5 Stand-alone ECC Hardware

The dedicated hardware design used for this paper (cf. Figure 2) is strongly related to the works of Kumar and Paar [19] and Lee *et al.* [20], but uses a different memory architecture. As register-based memory is most expensive, it is replaced by latches, which are 27 % smaller in the used 130 nm technology. As latches are not synchronous, the depicted circuit only works because a common `Work` register is placed before the latches. At the positive clock level, activated via the clock gate (CG), the latch inherits the contents stored within the `Work` register. A single multiplexer is used to select the content of a latch which is

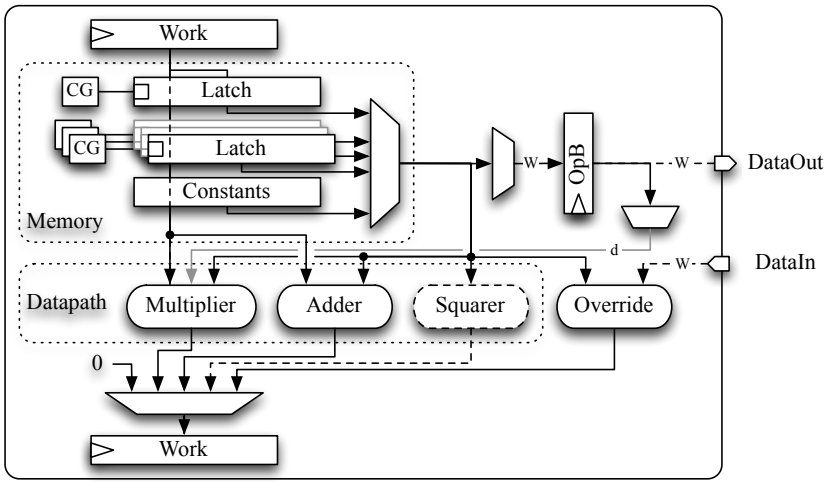

Fig. 2. Dedicated ECC hardware.

then used as operand `OpA` for the datapath. The datapath consists of an MSB-first digit-serial multiplier, an adder, and optionally a squaring unit. For the multiplication, an operand is split into W-bit sized parts which are stored in `OpB`. d of the W bits are then concurrently handled within the multiplication circuit. Dependent on the desired speed grade, it is possible to increase the size of d, or to use a dedicated squaring circuit. For interfacing the module with an external W-bit wide bus, the existing multiplexers are reused. For memory storing operations, W of the N-bit wide bus are overridden by the externally driven bus signal.

5.1 Stand-alone ECC Hardware Results

A complete ECC coprocessor including datapath, controlpath, memory, private scalar, modifiable base point, and resulting point with recovered y-coordinate needs at least 11,778 GE and up to 341,835 cycles. Table 5 summarizes our results for different d parameters. Adding a dedicated 1-cycle squaring unit only costs 884 GE (7.5 %) of additional hardware, but improves the runtime by a factor of approximately two. The most energy-efficient circuit is using $d = 2$. The circuit with the best scaled area-runtime product (SARP) is using $d = 4$.

Compared to related work [14, 19, 20, 25, 30], our designs are smaller or faster and therefore provide a better area-time product. In terms of power and energy, which are highly dependent on the used technology, our results are similar to related work.

The chip area shown in Table 5 does not include the area needed by the MSP430 (5,958 GE), its data memory (8×16-bit RAM – 1,443 GE), and its program memory (354 bytes – 801 GE). So all our dedicated ECC hardware designs

Table 5. Synthesis results of the dedicated ECC hardware design without MSP430.

Design	**Technology** [nm]	**Area** [GE]	**Runtime** [kCycles]	**Power** [µW]	**Energy** [µJ]	**SARP**
$d = 1$ w/o squ.	130	11,778	341,835	63.3	21.6	5.2
$d = 1$ w/ squ.	130	12,662	174,025	71.5	12.4	2.8
$d = 2$ w/ squ.	130	13,307	93,997	78.4	7.4	1.6
$d = 4$ w/ squ.	130	14,552	53,489	140.1	7.5	1.0
Kumar and Paar [19] $d = 1$	350	15,094	376,864	788.0	297.0	7.3
Hein *et al.* [14]	180	11,904	296,299	101.9	30.2	4.5
Lee *et al.* [20] $d = 1$	130	12,506	302,457	32.4	9.8	4.9
Lee *et al.* [20] $d = 5$	130	20,316	83,375	48.9	4.1	2.2

need additional 8,202 GE of hardware in order to provide the full functionality of an MSP430.

The major drawback of the ECC hardware module is the inefficient data memory. Unfortunately, there are no efficient RAM macros with a 163-bit interface. Even though latches are used, the memory requires 6,924 GE, or 59 % of the total hardware area. A comparable register-based RAM macro with $8 \times 163 = 1,467$ bits requires only 2,600 GE. That is 62 % less. For the drop-in concept discussed in the next section, such an area-efficient RAM macro is used.

6 Drop-in Concept

The drop-in concept has some similarities with instruction-set extensions. The drawback of ISE is that the HW designer needs to be able to modify both the controlpath and the datapath of the used processor, as well as the corresponding software toolchain. A different solution, based on a memory mapped carry-less multiply-accumulate unit has similarly large access times as the already existing integer multiply-accumulate unit of the MSP430. Therefore, it would only make a minor impact on the ECC runtime.

The drop-in concept provides full advantage even when the hardware designer is not able to modify the used microprocessor. Performance similar to dedicated ECC hardware is achievable and the verification and validation process regarding the used microprocessor does not have to be redone. The drop-in concept is also flexible: A hardware designer can shift control logic between the program memory and the dedicated hardware module. In this paper the drop-in module is designed to efficiently perform finite-field arithmetic (addition, squaring, and multiplication) only. The finite-field inverse as well as the point-multiplication algorithm are implemented in software.

As interface, the drop-in module provides three address, a command, and a status register. Before each operation, the address registers are written with two source and a destination memory address, and the operation is started by writing the command register. The status register is then polled to check whether the operation has been finished. Actually, experiments showed that waiting at the

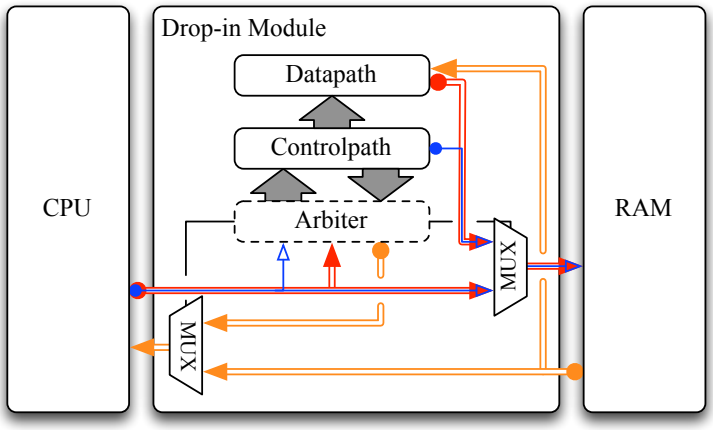

Fig. 3. Drop-in module for Elliptic Curve Cryptography.

beginning of the finite-field operations for the previous operations to finish is more performant. In this way the CPU and the drop-in module can partly work in parallel.

6.1 Drop-in Architecture

Figure 3 shows the architecture of the drop-in module. The data-bus is depicted in red and orange, the address bus in blue. The drop-in module consists of a lightweight arbiter, controlpath, and datapath. If both the CPU and the drop-in module want to access the data memory, the currently pending operation within the drop-in module is put on hold and the CPU is given access to the data memory. Therefore the drop-in module needs to be specially prepared for the case in which it is put on hold. For our ECC design, only 7 1-bit registers are necessary to provide this functionality. As a side note, the openMSP430 does not support to have delayed memory access.

The datapath within the drop-in module is very similar to the datapath of the dedicated ECC hardware module. Figure 4 shows that only two N-bit registers and a W-bit register are necessary for an MSB-first digit-serial multiplier. In each cycle, the N bits of `OpA` are multiplied with d bits of `OpB`, which are added to a d-bit shifted intermediate product, stored within the N-bit `Work` register. The $(N+d)$-bit sum is then reduced and used to update the `Work` register. At the beginning of the algorithm, `Work` is initialized with zero and `OpA` is initialized with the value stored within the data memory. The W-bit chunks of `OpB` are loaded on-demand, as it is shown within Figure 5 (d). When the multiplication is finished, the result within `Work` is stored back to the data memory.

Optionally, a dedicated squaring unit can be used. In our implementation (Figure 5 (a)), `OpA` is loaded from the data memory, the squaring is performed within a single cycle, and the result is stored back to the data memory. The

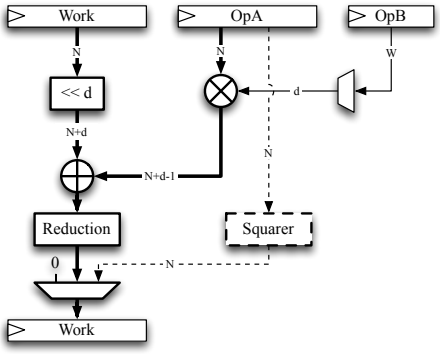

Fig. 4. Datapath of the ECC drop-in.

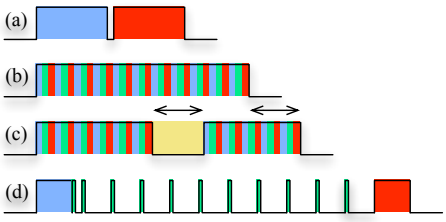

Fig. 5. Bus access during squaring (a), addition (b), addition with stall (c), and multiplication (d) operations. Memory operations regarding `OpA`, `OpB`, and `Dest` are colored in blue, green, and red, respectively.

datapath of the addition is not shown in Figure 4 as it only is a simple XOR-gate. For the finite-field addition (Figure 5 (b)) three times $\lceil N/W \rceil$ memory operations are necessary.

If at any moment, the CPU needs to do some (real-time) interrupt handling and needs access to the data memory, the operation in progress within the drop-in module is simply halted and continued when the data memory bus is free to use (Figure 5 (c)).

6.2 Drop-in Concept Hardware Results

Similar to before, the drop-in module was evaluated for different configurations (cf. Table 6). Independent of the size of the digit-serial multiplier and the availability of a squaring unit, the size of the CPU (5,715 GE), the program memory (1,426 bytes – 2,635 GE), and the data memory (222 bytes – 2,875 GE) are constant. The drop-in module only needs between 4,114 GE and 6,760 GE in chip area.

This is the most interesting number for microchip manufacturers. As not every customer actually needs ECC, they want to leave out unnecessary components, as they produce unnecessary costs. On the other hand, customers that require performant ECC can take advantage of the drop-in ECC module. Compared to a dedicated ECC hardware module, which requires 12–15 kGE, the drop-in module requires only a fraction of it: 35 %.

6.3 Related Work

In 2009, Guo and Schaumont [11] identified the data bus as potential bottleneck for ECC designs. Cause of that, they add the necessary data memory to the dedicated ECC accelerator to keep the number of necessary bus accesses at a

Table 6. Synthesis results of all ECC hardware architectures at 1 MHz for `sect163r2`.

Design	**Module** [GE]	**Chiparea** [GE]	**Runtime** [Cycles]	**Power** [µW]	**Energy** [µJ]
Architecture *(a)* – Software-only implementation					
openMSP430 w/o mult.	-	14,167	7,216,905	49.1	354.3
Architecture *(b)* – Dedicated ECC Hardware Accelerator					
$d = 1$ w/o squ.	11,778	19,980	342,724	93.8	32.1
$d = 1$ w/ squ.	12,662	20,864	174,910	112.9	19.7
$d = 2$ w/ squ.	13,307	21,509	94,882	152.4	14.5
$d = 4$ w/ squ.	14,552	22,754	54,376	181.7	9.9
Architecture *(d)* – Drop-in Module Based					
$d = 1$ w/o squ.	4,114	15,282	467,370	66.1	30.9
$d = 1$ w/ squ.	4,895	16,121	303,202	77.6	23.5
$d = 2$ w/ squ.	5,512	16,738	224,222	73.6	16.5
$d = 4$ w/ squ.	6,760	17,986	182,130	70.0	12.8

minimum. Thus their ECC accelerator becomes more like a dedicated hardware module. As it is an FPGA design, a comparison with our work is impracticable.

Most comparable to our drop-in concept is the work of Koschuch *et al.* [18]. They implemented a memory-less ECC accelerator and used a DMA controller for efficiently accessing the data memory. Their architecture is best comparable with the previously discussed architecture *(c)*. Their DMA controller is 1,029 GE large, their ECC accelerator is 11,618 GE large, and their total design for $\mathbb{F}_{2^{191}}$ requires 29,491 GE. For a scalar multiplication, they require 1,416 kCycles. Thus their design is slower and larger than our drop-in designs.

7 Comparison of Implemented Architectures

In the previous sections, architectures *(a)* - a plain software implementation, *(b)* - a dedicated ECC hardware module, and *(d)* - a drop-in module - have been presented and discussed in connection with the appropriate related work. Thereby all implementations are on-par with related work or outperform related work. Most important however is the comparison of the three implemented architectures *(a,b,d)* with each other.

Table 6 shows the area, runtime, power, and energy values of all architectures. The column 'Module' gives the area for the dedicated ECC hardware blocks, while 'Chiparea' accumulates the program memory, the data memory, the microprocessor, and the special hardware module. The runtimes of architecture *(b)* now include the calling overhead needed to trigger and poll the dedicated hardware module. In comparison to Table 5, the area and power values now also include the RAM, ROM, and CPU.

The smallest of all implementations is the plain software implementation *(a)* needing only 14,167 GE. Both the drop-in solution *(d)* (15,282 GE) and the dedicated hardware solution *(b)* (19,980 GE) are larger. However, those solutions

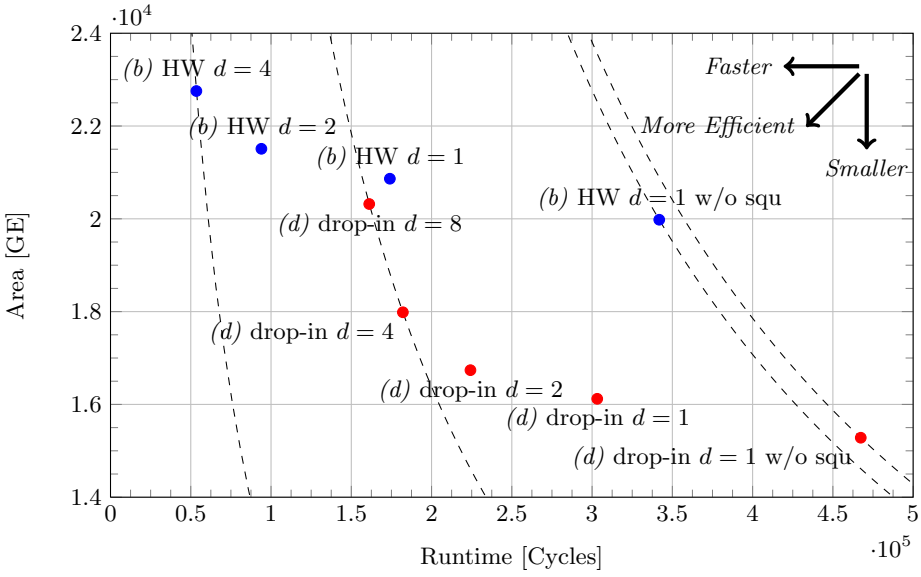

Fig. 6. Area-runtime-characteristics of the various ECC architectures.

are up to 132 times faster and up to 36 times more energy efficient. Thus architecture *(a)* can be considered as fall-back solution, but is practically to slow for most relevant applications. The runtime is nearly one second at a common sensors-node frequency of 8 Mhz.

Thus the question is whether architecture *(b)* or *(d)* is better. The drop-in concept *(d)* is 22 % smaller and requires 50 % less power. On the other hand, architecture *(b)* is faster. The comparison is visualized in Figure 6, which prints the chiparea values versus the runtimes. The dashed lines indicate constant area-runtime products. After investigating the results in detail, our conclusion is that both architectures *(b)* and *(d)* have the very right of existence. However, if the application requires that a point multiplication is finished within, e.g., 30 ms (@ 8 MHz), architecture *(d)* based on the drop-in concept with $d = 2$ is the smallest and therefore best solution.

8 Conclusion

This work proofs that the drop-in concept is a viable alternative to previously existing plain software and dedicated hardware solutions. Both the presented software-only and the presented dedicated hardware solution enable a fair comparison using a common side-channel aware methodology and identical tools. The software implementation is (supposed to be) side-channel secure and needs 7–12 times less memory compared to latest related work. The hardware implementation is more area-efficient compared to related work, because a specially

designed data memory is used. However, a plain hardware implementation is not aware of the versatile MSP430, which is usually available in wireless sensor nodes. Hereby the drop-in concept provides a novel solution which actually is smaller than the hardware module based architecture, while being similarly fast, and requiring 36 times less energy than the dedicated software solution. This makes the newly presented drop-in concept a great solution for microchip and sensor-node manufacturers.

Acknowledgments

The research described in this paper has been supported, in part, by the European Commission through the ICT Program under contract ICT-SEC-2009-5-258754 TAMPRES.

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. Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, September 2000.
3. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, Boca Raton, FL, 2006.
4. P. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, pages 526–538, 1990.
5. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *CHES 1999, Proceedings*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.
6. H. Eberle, A. Wander, N. Gura, S. Chang-Shantz, and V. Gupta. Architectural Extensions for Elliptic Curve Cryptography over GF(2^m) on 8-bit Microprocessors. In *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 343–349. IEEE Computer Society, 2005.
7. J. Fan and I. Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In *Cryptography and Security: From Theory to Applications*, LNCS, pages 265–282. Springer, 2012.
8. C. P. L. Gouvêa and J. López. Software Implementation of Pairing-Based Cryptography on Sensor Networks Using the MSP430 Microcontroller. In *INDOCRYPT*, LNCS, pages 248–262, 2009.
9. C. P. L. Gouvêa, L. Oliveira, and J. López. Efficient Software Implementation of Public-Key Cryptography on Sensor Networks Using the MSP430X Microcontroller. *Journal of Cryptographic Engineering*, 2:19–29, 2012.
10. J. Großschädl and E. Savaş. Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m). In *CHES*, pages 133–147, 2004.
11. X. Guo and P. Schaumont. Optimizing the HW/SW boundary of an ECC SoC design using control hierarchy and distributed storage. In *DATE*, pages 454–459, 2009.

12. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs. In *CHES*, pages 119–132, 2004.
13. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.
14. D. Hein, J. Wolkerstorfer, and N. Felber. ECC is Ready for RFID - A Proof in Silicon. In *SAC*, LNCS, pages 401–413, 2008.
15. M. Hutter, M. Joye, and Y. Sierra. Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In *AFRICACRYPT*, pages 170–187, 2011.
16. IEEE. IEEE Standard 802.15.4-2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), May 2003.
17. T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronic Letters*, pages 334–335, 1988.
18. M. Koschuch, J. Großschädl, D. Page, P. Grabher, M. Hudler, and M. Krüger. Hardware/Software Co-Design of Public-Key Cryptography for SSL Protocol Execution in Embedded Systems. In *Workshop on Embedded Systems Security*, pages 63–79, 2009.
19. S. S. Kumar and C. Paar. Are standards compliant Elliptic Curve Cryptosystems feasible on RFID? In *Workshop on RFID Security – RFIDSec 2006*, 2006.
20. 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.
21. 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*, pages 245–256, 2008.
22. J. López and R. Dahab. Fast Multiplication on Elliptic Curves over GF(2^{m}) without Precomputation. In Ç. K. Koç and C. Paar, editors, *CHES 1999, Proceedings*, volume 1717 of *LNCS*, pages 316–327. Springer, 1999.
23. National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), 2009.
24. Olivier Girard. openMSP430, 2013. Available online at `http://opencores.org/project,openmsp430`.
25. E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In M. Joye and J.-J. Quisquater, editors, *CHES 2004, Proceedings*, volume 3156 of *LNCS*, pages 92–106. Springer, August 2004.
26. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Wireless Sensor Networks 5th European Conference*, LNCS, pages 305–320, 2008.
27. Texas Instruments. MSP430C11x1 - Mixed Signal Microcontroller. Available online at `http://focus.ti.com`, 2008.
28. H. Wang, B. Sheng, and Q. Li. Elliptic Curve Cryptography-based Access Control in Sensor Networks. *International Journal of Security and Networks*, pages 127–137, 2006.
29. E. Wenger and M. Hutter. Exploring the design space of prime field vs. binary field ecc-hardware implementations. In *Information Security Technology for Applications*, LNCS, pages 256–271. Springer, 2012.
30. J. Wolkerstorfer. Is Elliptic-Curve Cryptography Suitable for Small Devices? In *Workshop on RFID and Lightweight Crypto*, pages 78–91, 2005.
31. ZigBee Alliance. The ZigBee Alliance Website. `http://www.zigbee.org/`.

A Implementation Runtimes

Table 7 lists the constant key-independent runtimes of all implementations done for this paper. Architectures *(b)* and *(d)* implemented the elliptic curve `sect163r2`.

We distinguish between runtimes for the original MSP430 and the openMSP430. The runtimes of the openMSP430 are better, because several instructions of the openMSP430 perform the same operation in less cycles than the original MSP430. In average, the openMSP430 is 5 % faster for the prime field based elliptic curves (`secp160r1`, `secp192r1`) and 3 % faster for the binary field based elliptic curves (`sect163r2`, `c2tnb191v1`).

Table 7. Runtimes for finite-field addition/subtraction (ADD), squaring (SQU), multiplication (MUL), inversion (INV), and point-multiplication (P-MUL) operations.

Implementation	ADD [Cycles]	SQU [Cycles]	MUL [Cycles]	INV [Cycles]	P-MUL [Cycles]
(a) MSP430 `secp160r1`	163	1,905	1,905	327,366	5,721,420
(a) MSP430 `secp192r1`	191	2,559	2,559	526568	9,100,128
(a) MSP430 `sect163r2`	109	852	6,604	199,815	7,446,677
(a) MSP430 `c2tnb191v1`	118	778	6,566	229,297	8,610,906
(a) openMSP430 `secp160r1`	161	1,808	1,808	310,812	5,445,010
(a) openMSP430 `secp192r1`	189	2,426	2,426	499,331	8,650,455
(a) openMSP430 `sect163r2`	107	781	6,446	186,653	7,216,905
(a) openMSP430 `c2tnb191v1`	116	725	6,420	217,209	8,376,138
(b) HW $d = 1$ w/o squ	2	174	174	29,754	341,835
(b) HW $d = 1$	2	1	174	1,728	174,025
(b) HW $d = 2$	2	1	93	999	93,997
(b) HW $d = 4$	2	1	52	630	53,489
(d) drop-in $d = 1$ w/o squ	40	208	208	36,419	467,370
(d) drop-in $d = 1$	40	38	208	9,963	303,202
(d) drop-in $d = 2$	40	38	128	9,227	224,222
(d) drop-in $d = 4$	40	38	80	8,843	182,130