Hardware/Software Co-Design of Elliptic Curve Cryptography on an 8051 Microcontroller

Manuel Koschuch, Joachim Lechner, Andreas Weitzer, Johann Großschädl, Alexander Szekely, Stefan Tillich, and Johannes Wolkerstorfer

Institute for Applied Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
{manuel.koschuch,joachim.lechner,andreas.weitzer}@student.tugraz.at
{jgrosz,aszekely,stillich,jwolkers}@iaik.tugraz.at

Abstract. 8-bit microcontrollers like the 8051 still hold a considerable share of the embedded systems market and dominate in the smart card industry. The performance of 8-bit microcontrollers is often too poor for the implementation of public-key cryptography in software. In this paper we present a minimalist hardware accelerator for enabling elliptic curve cryptography (ECC) on an 8051 microcontroller. We demonstrate the importance of removing system-level performance bottlenecks caused by the transfer of operands between hardware accelerator and external RAM. The integration of a small direct memory access (DMA) unit proves vital to exploit the full potential of the hardware accelerator. Our design allows to perform a scalar multiplication over the binary extension field $GF(2^{191})$ in 118 msec at a clock frequency of 12 MHz. Considering performance and hardware cost, our system compares favorably with previous work on similar 8-bit platforms.

1 Introduction

Embedded systems made up of hardware and software components constitute the fastest growing segment of the semiconductor industry with products ranging from mobile phones over MP3 players to automotive braking systems. The traditional design techniques (i.e. separate treatment of hardware and software) do not cope with the complexity of today's embedded systems and the steadily increasing time-to-market pressure. Sloppily speaking, "building a machine and seeing whether it works" is not feasible due to unpredictable design times when heterogeneous applications are getting integrated to create a complex system [23]. A promising approach to deal with the complexity of modern embedded systems is *hardware/software co-design*, i.e. the concurrent (or simultaneous) design of hardware and software components with the goal to meet system-level objectives [5]. This includes the analysis of different boundaries and interfaces between hardware and software and the evaluation of design alternatives in a reasonable amount of time [8].

Hardware/software co-design is gaining in importance since the boundary between hardware and software becomes more and more blurred. One factor behind this trend is the advent of flexible architectures that combine general-purpose processors with custom, customizable, or reconfigurable logic. In recent years, major FPGA vendors started to offer special devices consisting of a processor core, on-chip memories, peripherals, and large amounts of reconfigurable logic for the implementation of application-specific hardware. In addition, some of these devices contain application-specific building blocks like fast multipliers for digital signal processing (e.g. Altera Stratix). Therefore, devices consisting of a processor core, application-specific parts, and reconfigurable logic are an ideal co-design platform for heterogenous embedded systems that may comprise several applications domains such as signal processing, networking, and security. Recently, the security domain has attracted particular interest since more and more embedded devices store or transmit sensitive data. This makes a strong case for applying hardware/software co-design techniques to the implementation of cryptographic primitives [20, 21].

In this paper we investigate the co-design of *elliptic curve cryptography* on embedded 8-bit platforms, in particular on the 8051 microcontroller. Elliptic curve cryptography (ECC) is highly computation-intensive as it involves arithmetic operations in finite fields of large order (typically about 160 bits) [3]. The results from previous work [12, 16, 24] show that a "pure" software implementation of ECC does not allow to reach sub-second performance on a standard 8051 clocked at 12 MHz. Therefore, some kind of *hardware acceleration* of the performance-critical operations carried out in ECC is necessary. Elliptic curve cryptography offers a multitude of implementation options for both field and curve (group) arithmetic, respectively [13]. In addition, a number of different boundaries between hardware and software are possible, which allows a system designer to find the proper trade-off between performance and silicon area. One could, for instance, implement the point addition/doubling in hardware and the rest in software [15]. An alternative approach is to implement the field arithmetic in hardware and the curve/point arithmetic in software [1, 2, 7, 14]. Furthermore, hardware acceleration at the granularity of instruction set extensions for the finite field multiplication has also been investigated [6, 11, 17]. Besides the hardware/software boundary, the *interface* between hardware accelerator and host processor is essential for the system performance, especially for "low-cost" accelerators without local storage since they require a high number of data transfers.

We have co-designed an elliptic curve cryptosystem over binary extension fields using the Dalton 8051 [22] as host controller which executes the software part of our design. The hardware part consists of an *elliptic curve acceleration unit (ECAU)* and an *interface with direct memory access (DMA)* to enable fast data transfer between the ECAU and the external RAM (XRAM) attached to the 8051 microcontroller. Our design goal is to achieve a maximum of performance with a "minimalist" hardware accelerator—the ECAU—composed of a bit-serial multiplier for binary extension fields of order ≤ 192 bits and a supporting register infrastructure. The ECAU allows to perform a full scalar multiplication over the field $GF(2^{191})$ in about 118 msec, assuming that the Dalton 8051 is clocked with 12 MHz. A scalar multiplication over the field GF(2^{163}) takes less than 100 msec, which is more than 25 times faster than the co-design for hyperelliptic curve cryptography (HECC)[1] presented by Batina et al. at CHES 2005 [2]. The hardware cost of the ECAU and the DMA controller is 12,65k gates altogether when synthesized with a 0.35 μm standard cell library.

1.1 Improvements over Previous Work

During the past five years, numerous papers dealing with the hardware/software co-design of (hyper)elliptic curve cryptography on 8-bit platforms (e.g. AVR or 8051) have been published [1, 2, 6, 7, 14, 15, 17]. However, the co-design approach for ECC presented in this paper differs from previous work in two important aspects. First, we pay special attention to the efficient implementation of the data transfer between the hardware accelerator and the external RAM attached to the 8051. Second, our approach uses a (limited) scalable hardware accelerator able to perform field arithmetic in all binary fields GF(2^m) with $m \leq 192$ and not just in a single field.

The efficiency of the *data transfer* between ECAU and XRAM impacts the overall performance since the ECAU is a low-cost hardware accelerator, which means that it does not contain local storage for intermediate results. Consequently, all intermediate results occurring during a scalar multiplication have to be transferred between the ECAU and the XRAM[2]. Unfortunately, a standard 8051 only provides 8-bit I/O ports and a serial interface for the communication with the "world outside," both of which are rather slow. There are two principal options to alleviate the communication bottleneck. One possibility is to equip the ECAU with local storage for the intermediate results. The second option is to design a dedicated interface with direct memory access (DMA). We opted for the latter since the former would entail a considerable increase in silicon area. In addition, we have also integrated an I/O register into the ECAU which allows to overlap data transfer and computation phases.

A second point in which our co-design approach differs from previous work is *scalability*, i.e. the ability to process operands of any size without the need to modify or re-design a given implementation [19]. The ECAU contains a 192-bit multiplier that can be used for any binary extension field GF(2^m) of degree up to 192, e.g. for the field GF(2^{191}) or GF(2^{163}). This means that our system is limited scalable similar to the cryptographic processor described in [9], but does not provide the high scalability of the ECC hardware from [19]. We emphasize that attaining scalability in hardware/software co-design affects all abstraction levels and layers between hardware and software (including the operand transfers), and is not a "pure" hardware design issue as in [19]. For instance, when

[1] Batina et al. presented a hyperelliptic curve cryptosystem of genus 2 over the field GF(2^{83}). The security level of this HECC system is approximately 166 bits, and thus it is comparable to the ECC system over the field GF(2^{163}) that we have used.

[2] We store the intermediate values in the XRAM since the internal RAM of a standard 8051 microcontroller has a size of only 128 bytes (see Appendix A).

using a "small" field like $\mathrm{GF}(2^{163})$, only 21 bytes per operand need to be transferred between ECAU and XRAM. Furthermore, all software routines have an additional parameter specifying the degree m of the field. To the best of our knowledge, this paper presents the first hardware/software co-design approach for elliptic curve cryptography providing a certain level of scalability.

2 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) has a number of advantages over the traditional public-key cryptosystems based on the integer factorization problem or the discrete logarithm problem in finite fields. The most important advantage is the absence of a subexponential-time algorithm that could solve the discrete logarithm problem in a properly selected EC group. As a consequence, elliptic curve cryptosystems can use much shorter keys, which results in faster implementations and lower memory and bandwidth requirements [3].

Formally, an elliptic curve cryptosystem operates in a group of points on an elliptic curve defined over a finite field. Most practical ECC implementations use special types of finite fields to improve performance; among these special field types are binary extension fields $\mathrm{GF}(2^m)$, prime fields $\mathrm{GF}(p)$, and optimal extension fields (OEFs), i.e. extension fields $\mathrm{GF}(p^m)$ whose characteristic p and extension degree m are specifically selected [13]. The latter two field types allow for efficient software implementation, especially on processors equipped with a fast integer multiplier. For hardware implementation, on the other hand, binary extension fields $\mathrm{GF}(2^m)$ are generally the better choice. Therefore, we shall only consider binary extension fields in the rest of this paper.

An elliptic curve over a binary field $\mathrm{GF}(2^m)$ can be defined as the set of all solutions $(x, y) \in \mathrm{GF}(2^m) \times \mathrm{GF}(2^m)$ to the (affine) Weierstraß equation

$$y^2 + xy \;=\; x^3 + ax^2 + b \quad \text{with} \quad a, b \in \mathrm{GF}(2^m) \tag{1}$$

A tuple $(x, y) \in \mathrm{GF}(2^m) \times \mathrm{GF}(2^m)$ satisfying Equation 1 is called a *point* on the curve. The set of all points, together with a special point $\mathcal{O}$ (referred to as the "point at infinity"), allows to form an *Abelian group* with $\mathcal{O}$ acting as identity element. The group operation is the addition of points, which can be performed through arithmetic operations (addition, multiplication, squaring, and inversion) in the underlying field $\mathrm{GF}(2^m)$ according to well-defined formulae [13].

A basic building block of all elliptic curve cryptosystems is *scalar multiplication*, an operation of the form $k \cdot P$ where k is an integer and P is a point on the curve. In its simplest form, a scalar multiplication can be realized through a sequence of point additions and doublings, respectively. There exist a number of advanced algorithms for point multiplication; one of the most efficient was proposed by López and Dahab in [18]. Their algorithm requires to carry out $4\lfloor \log_2 k \rfloor + 6$ additions, $2\lfloor \log_2 k \rfloor + 4$ multiplications, $2\lfloor \log_2 k \rfloor + 2$ squarings and $2\lfloor \log_2 k \rfloor + 1$ inversions in the underlying finite field $\mathrm{GF}(2^m)$ to obtain the result of $k \cdot P$ [18, Lemma 4].

Table 1. Overview of arithmetic operations when using LD projective coordinates

Operation	# Field Add	# Field Mul	# Field Sqr	# Field Inv
Point addition (`Madd`)	2	4	1	0
Point doubling (`Mdouble`)	1	2	4	0
Conv. affine to proj. coord.	1	0	2	0
Conv. proj. to affine coord.	6	10	1	1
Scalar multiplication $k \cdot P$	$3\lfloor\log_2 k\rfloor+7$	$6\lfloor\log_2 k\rfloor+10$	$5\lfloor\log_2 k\rfloor+3$	1

Inversion is generally the most demanding—and hence slowest—arithmetic operation in $GF(2^m)$. Therefore, it is prudent to use an algorithm for scalar multiplication that minimizes the number of inversions. If the points on the curve are represented in *projective coordinates* [3], then the inversion operation can be almost completely avoided at the cost of additional field multiplications and some extra storage for auxiliary variables. Only one inversion is needed for the re-conversion from projective to affine coordinates. The point addition and doubling in López-Dahab (LD) projective coordinates can be calculated as shown in Algorithm `Madd` and `Mdouble` in [18, Appendix A], respectively. Table 1 specifies the overall number of field arithmetic operations for point addition, point doubling, re-conversion from projective to affine coordinates, and a full scalar multiplication. A special property of the LD scalar multiplication algorithm is the fact that it performs exactly one `Madd` and one `Mdouble` operation for each bit of the scalar k. Consequently, the total number of `Madd`/`Mdouble` operations depends only on the bitlength of k, but not on its Hamming weight, i.e. the number of "0" and "1" bits in the binary representation of k. This property helps to prevent certain *side-channel attacks* like simple power analysis (SPA) attacks and timing attacks [13].

The elements of a binary extension field $GF(2^m)$ can be represented by binary polynomials of degree up to $m-1$. Addition in $GF(2^m)$ is simply a logical XOR operation, while the multiplication of two field elements is performed modulo an *irreducible polynomial* p of degree m. Hardware multipliers for $GF(2^m)$ produce the product of two field elements by generation and addition of partial products as well as generation and addition of multiples of p. Squaring in $GF(2^m)$ is a special case of multiplication and can be implemented in hardware in one clock cycle when p is fixed and has a low weight. Finally, the inversion can be realized either by using the extended Euclidean algorithm (EEA) or with help of Fermat's theorem by calculating $a^{-1} = a^{2^m-2} \bmod p$, which results in a sequence of field multiplications and squarings, respectively. Therefore, a "minimalist" hardware accelerator should be able to perform addition and multiplication in $GF(2^m)$.

3 Hardware/Software Boundaries and Trade-Offs

Efficient software implementation of ECC on 8-bit platforms is a challenging task, in particular if the order of the underlying field is beyond 160 bits. Recent research has shown that highly-optimized software implementations can reach sub-second performance on the ATmega128 [12], but not on a standard 8051 microcontroller, at least not if the order of the finite field is 160 bits or more [12, 16, 24]. The main reason is the rather poor performance of a standard 8051 in relation to the ATmega128 (see Appendix A). Hardware/software co-design offers numerous possibilities for speeding up ECC at the cost of a moderate increase in silicon area. A survey of the recent literature allows to identify three basic co-design approaches for enabling ECC on 8-bit processors. In this section we discuss the different hardware/software boundaries and analyze the pros and cons of these approaches.

One way to partition between hardware and software is to assign a full point addition/doubling operation to the hardware part and the rest to the software part. A concrete implementation following this approach was reported by Janssens et al. in [15]. They implemented the field arithmetic operations in hardware, together with local RAM for storing intermediate results and dedicated state machines to control the point addition/doubling operations. While this approach is very fast (there are no operand transfers during a point addition/doubling), it suffers from high hardware cost. Furthermore, implementing the point addition/doubling in hardware does not allow to respond to progress in ECC, e.g. when more efficient addition/doubling formulae are developed.

A second way to draw the line between hardware and software is to offload the field arithmetic operations from the host processor and execute them in a dedicated hardware accelerator like a co-processor. All other operations, i.e. point addition/doubling and scalar multiplication, are implemented in software and executed on the host processor. In general, this approach offers high flexibility, including the ability to integrate the latest countermeasures against side-channel attacks into the algorithm for scalar multiplication. On the other hand, this approach may entail a significant communication overhead, especially when the accelerator hardware does not provide local storage for auxiliary variables. The fastest implementations following this approach have been reported by Ernst et al. [7] and Aigner et al. [1]. The latter implements all field arithmetic operations in hardware (including squaring and inversion) and uses affine coordinates. Other implementations are described in [17] and in [2, 14], whereby the latter two are based on hyperelliptic curve cryptography (HECC). Detailed performance figures of all these works can be found in Table 5 in Section 5. Our co-design for ECC presented in the next section also follows this approach.

Finally, the boundary between hardware and software can also be defined at the level of custom instructions that are specifically designed to accelerate the field arithmetic, most notably the field multiplication [11]. Hardware/software co-design at the granularity of instruction set extensions provides the highest flexibility and requires the least amount of extra hardware of all approaches discussed in this section. It was demonstrated in [6] that instruction set extensions enable an ATmega128 to execute a scalar multiplication over $GF(2^{163})$ in 290 msec (at a clock frequency of 8 MHz). However, it is highly questionable whether similar performance can be reached on a standard 8051 microcontroller where one instruction cycle takes 12 clock cycles (see Appendix A).

4 Implementation Details

In the following, we describe the hardware accelerator that we implemented to enable fast ECC on the Dalton 8051 microcontroller [22]. We begin with a system overview. Then, the Elliptic Curve Acceleration Unit (ECAU), the interface to the external RAM (XRAM), and the system software are presented.

4.1 System Overview

The overall system structure is depicted in Figure 1. It consists of four major parts: The Dalton 8051 microcontroller core, the Elliptic Curve Acceleration Unit (ECAU) with a separate datapath and control unit, and the DMA interface to the XRAM.

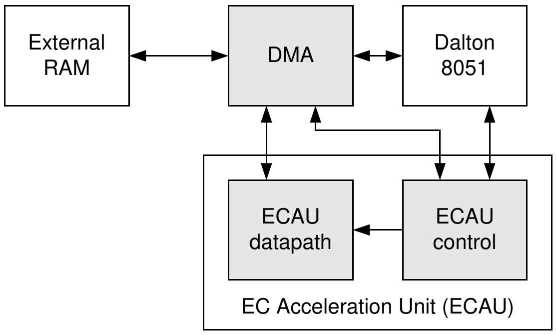

Fig. 1. System block diagram

The control unit inside the ECAU is responsible for generating appropriate control signals for the ECC datapath and provides busy signals to the DMA interface. The ECAU and the DMA interface support operand lengths of up to 192 bits, but can be configured at runtime for smaller operands.

4.2 Elliptic Curve Acceleration Unit (ECAU)

Figure 2 shows the internal architecture and the I/O interface of the ECAU. The heart of our EC accelerator is the GF(2^m) arithmetic unit, which consists of a bit-serial polynomial multiplier with interleaved reduction and several registers for operands and the result. Furthermore, the GF(2^m) arithmetic unit can also be used for the addition (i.e. XOR) of two field elements.

The I/O register decouples the ECAU from the DMA interface, which makes it possible to transfer data while the unit is performing a multiplication. The DMA interface shifts data in and out of the I/O register in blocks of 8 bits each. All other registers are accessed via the I/O register and support parallel data transfer through an internal 192-bit bus.

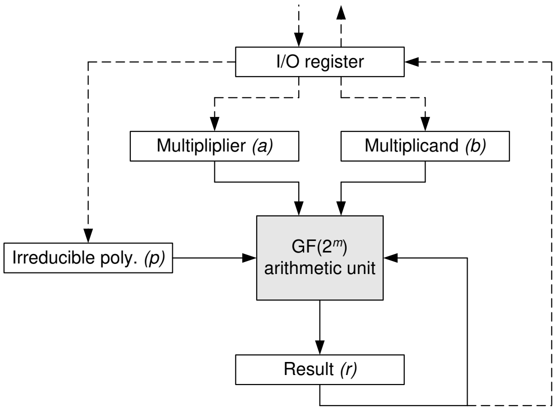

Fig. 2. Elliptic Curve Acceleration Unit

The datapath of the GF(2^m) arithmetic unit, shown in Figure 3, is based on the structure proposed in [10]. It supports addition and multiplication in binary extension fields GF(2^m) with $m \leq 192$ and puts no restriction on the form of the irreducible polynomial, i.e. it works with any irreducible polynomial. The control signal $\overline{add}/mul$ allows to switch between addition and multiplication mode.

In order to perform an addition, the first operand must be present inside the result register. This is almost always the case during a scalar multiplication since one of the two operands is the result of the previous arithmetic operation (addition or multiplication). The second operand needs to be stored in the multiplicand register b. The GF(2^m) arithmetic unit operates in addition mode if the $\overline{add}/mul$ signal is set to 0 and the *multiplier bit* input is 1. This selects the upper inputs of the multiplexers and disables the reduction modulo the irreducible polynomial p. The addition in GF(2^m) is nothing else than a simple bit-wise XOR of the coefficients and the sum is written back to the result register.

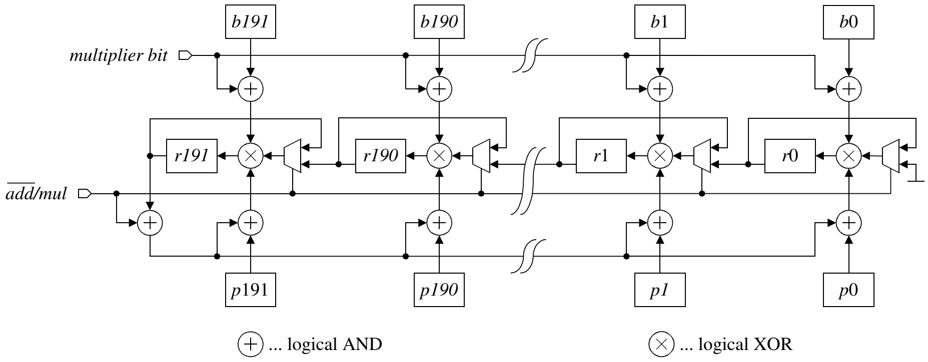

Fig. 3. Datapath of the GF(2^m) arithmetic unit for operands up to 192 bits

The multiplication in $GF(2^m)$ is realized through an MSB-first bit-serial multiplier with interleaved reduction modulo the irreducible polynomial. Before a multiplication can be started, the result register must be cleared and the two operands need to be present in the multiplicand and multiplier register, respectively. To enable the multiplication mode, the $\overline{add}/mul$ signal must be set to 1. The control logic then causes the multiplier register to perform a 1-bit left-shift operation in each cycle, which delivers one bit of the multiplier polynomial a to the *multiplier bit* input, starting with the most significant bit a_{m-1}.

The partial products are calculated by a bit-wise logical AND operation between the *multiplier bit* and each bit of the multiplicand polynomial b. To get the final result of the multiplication, a total of m partial products need to be summed up. Together with the generation and addition of partial products, the bit-serial multiplier performs 1-bit left-shift operations and reduces the intermediate result modulo the irreducible polynomial p in each cycle. The reduction modulo p is performed by adding p to the intermediate value stored in the result register whenever its most significant bit (MSB) is 1. An MSB of 1 means that the intermediate result would have a degree of m after the next 1-bit left-shift operation, and therefore the irreducible polynomial p must be added to reduce the intermediate result to a degree of at most $m - 1$ (see [10] for details). Note that the addition of the partial product, the 1-bit left-shift operation of the intermediate result, and reduction step (i.e. the conditional addition of p) are taking place simultaneously in each clock cycle. After the final coefficient of the multiplier polynomial has been processed, the result of the multiplication resides within the result register (after m clock cycles).

Because a required reduction is detected by checking if $r191 = 1$, all arguments in the registers need to be left aligned. For example, if the field $GF(2^{191})$ is used, then all operands need to be shifted left by one bit, and for $GF(2^{163})$ by 29 bits. However, these shift operations have to be carried out only once at the beginning of the scalar multiplication. All field arithmetic operations during a scalar multiplication are performed with the shifted operands.

4.3 Interface with Direct Memory Access (DMA)

The 8051 itself has too little internal RAM to hold the operands and auxiliary variables needed during a scalar multiplication. Since the Dalton 8051 needs 17 clock cycles for each instruction cycle, it would require at least $17 \cdot 192/8 = 408$ cycles to transfer one 192-bit operand from XRAM to the ECAU and another 408 cycles to transfer the result back into XRAM (assuming only one instruction cycle for XRAM access). This is unreasonably slow compared to the m clock cycles needed for a multiplication in $GF(2^m)$ and the single-cycle execution of a field addition. Therefore, we propose to use a DMA controller to facilitate fast data transfers between the ECAU and XRAM, bypassing the slow 8051.

In order to load a whole 192-bit operand, one just needs to provide the start address of the argument in the XRAM and then start the DMA controller. The controller transfers the whole operand byte by byte in 85 clock cycles to the I/O register. If operands shorter than 192 bits are used (e.g. 163 bits), then the

Table 2. ECAU instructions and their execution times

Command	Cycles	Description
MUL	$m+4$	Result $\leftarrow$ Multiplicand $\times$ Multiplier $\bmod p$
ADD	4	Result $\leftarrow$ Result + Multiplicand
LOAD_IOR	4	I/O register $\leftarrow$ Result register
CLEAR_RR	4	Result $\leftarrow 0$
LOAD_MDR	4	Multiplicand $\leftarrow$ I/O register
LOAD_MR	4	Multiplier $\leftarrow$ I/O register
LOAD_IPR	4	Irreducible Polynomial (p) $\leftarrow$ I/O register

interface automatically clears all unused bits in the I/O register and aligns the operand to the most significant byte. However, bit-wise alignments have to be done in software.

4.4 Software

To control the ECAU, three special function registers (SFRs) are used. The degree m of the binary field is set via the bitlength register. The status register provides feedback about the current operation status of the ECAU and the DMA interface. A third SFR is used to send instructions to the ECAU. Table 2 shows all implemented instructions and their respective timings.

In order to take advantage of the additional hardware, the software must be adapted accordingly. We have developed assembler-optimized functions that use our hardware extensions. Wherever possible, data transfers to the I/O register are interleaved with ECAU operations. By careful examination of the dataflow during a scalar multiplication, transfer delays can be reduced to a minimum.

5 Implementation Results

In order to determine the size of the new hardware units, we have synthesized the extended Dalton 8051 with a 0.35 μm standard-cell library from Austria Microsystems. The targeted delay for the critical path was set to 83 nsec (12 MHz). The minimal possible critical path delay is about 13 nsec (77 MHz), whereby the critical path is located within the ALU of the Dalton 8051. Our additional units (DMA, ECAU) could be clocked at significantly higher frequencies than the microcontroller.

Table 3 lists the size of the original Dalton 8051 microcontroller, the DMA unit, and the components of the ECAU (control logic, datapath logic, as well as datapath flip-flops). The size is given in absolute values in μm^2 as well as in gate equivalents (GE). The GE count has been derived from the absolute size of the component divided by the size of a NAND gate with the lowest driving strength from the used library.

Note that the internal RAM (IRAM) of the 8051 has been implemented as flip-flops for our synthesis. In practice a part of the IRAM could be implemented

Table 3. Hardware size and maximal clock frequency of the extended system

Component	**Size** μm^2	**Size** GE	**Max. Freq.** MHz
8051 core (excl. IRAM, ROM)	272,145	4,984	77
8051 IRAM (as flip-flops)	647,574	11,860	
DMA	56,202	1,029	333
ECAU control	39,203	718	
ECAU datapath logic	228,246	4,180	
ECAU datapath FFs	366,912	6,720	
Total	1,610,282	29,491	77

with SRAM macros in order to save silicon area. The enhanced 8051 microcontroller has a size of about 30,000 GEs. The additional components for ECC are about 75% of the original Dalton 8051's size and account for about 43% of the extended system.

Table 4. Execution times of operations for scalar multiplication over GF(2^{191})

Operation	**Cycles**
Transfer of one 191-bit operand	85
Addition in GF(2^{191}) excluding operand transfers	4
Multiplication in GF(2^{191}) excl. operand transfers	195
Point addition (`Madd`) including operand transfers	2,623
Point doubling (`Mdouble`) incl. operand transfers	2,623
Full scalar multiplication over GF(2^{191})	1,416,000

Table 4 shows the execution times for diverse arithmetic operations with 191-bit operands. Table 5 compares the performance of ECC multiplication with related work. Systems built around an AVR microcontroller are faster than systems using an 8051, which is caused by the generally better performance of AVR devices (see Appendix A). The work by Aigner et al. uses affine coordinates and has an additional squaring and inverter unit which can perform a squaring operation in one clock cycle and an inversion in $2m$ clock cycles.

Batina at al. use in their work a HECC system of genus two over the field GF(2^{83}), which provides roughly the same level of security as 163-bit ECC. Our work reaches significantly better performance compared to Batina et al. mainly due to efficient operand transfer between ECAU and XRAM thanks to direct memory access. Hodjat et al. try to circumvent the performance bottleneck by using a local storage unit of 256 bytes, which needs additional silicon area.

Table 6 compares our implementation with related work in terms of hardware and code size. Our work needs more silicon area than the design by Batina et al., but achieves a 16-fold better area-delay product. Also our code size is considerable smaller, which directly translates into savings in silicon area when the program code is stored in on-chip ROM.

Table 5. Performance comparison with ECC/HECC scalar multiplication of related work. The first six table entries refer to "pure" software implementations and the rest to hardware/software co-designs.

Reference	**Target Platform**	**Security Level**	**Field Type**	**Freq** MHz	**Performance** msec	cycles
Woodb. [24]	8051 (SLE44C24S)	ECC 134 bit	GF(p^m)	12.00	1,830.0	21.96M
Kumar [16]	8051 (CC1010)	ECC 134 bit	GF(p^m)	3.69	2,999.8	11.06M
Gura [12]	8051 (CC1010)	ECC 160 bit	GF(p)	14.74	4,580.0	67.53M
Gura [12]	8051 (CC1010)	ECC 192 bit	GF(p)	14.74	7,560.0	111.48M
Gura [12]	AVR (ATmega128)	ECC 160 bit	GF(p)	8.00	810.0	6.48M
Gura [12]	AVR (ATmega128)	ECC 192 bit	GF(p)	8.00	1,240.0	9.92M
Ernst [7] [a]	AVR (AT94K)	ECC 113 bit	GF(2^m)	12.00	1.2	14.40k
Kumar [17]	AVR (AT94K)	ECC 163 bit	GF(2^m)	4.00	113.0	452.00k
Janssens [15]	AVR (AT94K)	ECC 192 bit	GF(2^m)	10.00	45.0	450.00k
Eberle [6]	AVR (ATMega128)	ECC 163 bit	GF(2^m)	8.00	290.0	2.32M
Aigner [1] [a]	8051 (SLE66CX)	ECC 191 bit	GF(2^m)	10.00	44.3	443.86k
Batina [2]	8051 (Dalton)	HECC 166 bit	GF(2^m)	12.00	2,488.0	29.86M
Hodjat [14]	8051 (Dalton)	HECC 166 bit	GF(2^m)	12.00	656.0	7.87M
This work	8051 (Dalton)	ECC 163 bit	GF(2^m)	12.00	99.2	1.19M
This work	8051 (Dalton)	ECC 191 bit	GF(2^m)	12.00	118.0	1.42M

[a] Estimated performance figures.

Table 6. Comparison of hardware cost, code size, and XRAM requirements

Component	**Size** (norm.)	**Area-delay product** Size (norm.) × msec	**Code size** Bytes	**XRAM** Bytes
Dalton 8051	1.00			
Batina [2]	1.15	2,861.2	11,524	936
This work (163 bit)	1.75	173.6	2,568	384
This work (192 bit)	1.75	206.5	2,568	336

6 Conclusions

In this paper we have presented a hardware/software co-design approach for enabling ECC on 8-bit platforms using a minimalist hardware accelerator. We have demonstrated the importance of a thorough analysis of the overall system performance to remove bottlenecks. Communication overhead due to operand transfers has been minimized by integration of a small DMA unit and through the inclusion of an additional I/O register into the hardware accelerator. With the help of our simple and fast finite field arithmetic unit, we can support scalar multiplication over binary fields of degree up to 192. At the cost of about 12.65k gates in hardware, ECC scalar multiplication requires 118 msec over GF(2^{191}) and 99.2 msec over GF(2^{163}) on our enhanced 8051 system when clocked with 12 MHz. Considering performance gain in relation with hardware overhead, our solution relates very well to previous work on comparable 8-bit platforms.

Acknowledgements. The research described in this paper was supported by the Austrian Science Fund under grant P16952-NO4 "Instruction Set Extensions for Public-Key Cryptography" and in part by the European Commission through the IST Programme under contract IST-2002-507932 ECRYPT. The information in this paper reflects only the authors' views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. H. Aigner, H. Bock, M. Hütter, and J. Wolkerstorfer. A low-cost ECC coprocessor for smartcards. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, LNCS 3156, pp. 107–118. Springer Verlag, 2004.
2. L. Batina, D. Hwang, A. Hodjat, B. Preneel, and I. Verbauwhede. Hardware/software co-design for hyperelliptic curve cryptography (HECC) on the 8051 μP. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, LNCS 3659, pp. 106–118. Springer Verlag, 2005.
3. I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.
4. J. Catsoulis. *Designing Embedded Hardware.* O'Reilly Media, 2002.
5. G. De Micheli and R. K. Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, Mar. 1997.
6. H. Eberle et al. Architectural extensions for elliptic curve cryptography over $\mathrm{GF}(2^m)$ on 8-bit microprocessors. In *Proceedings of the 16th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 2005)*, pp. 343–349. IEEE Computer Society Press, 2005.
7. M. Ernst et al. A reconfigurable system on chip implementation for elliptic curve cryptography over $\mathrm{GF}(2^n)$. In *Cryptographic Hardware and Embedded Systems — CHES 2002*, LNCS 2523, pp. 381–399. Springer Verlag, 2002.
8. R. Ernst. Codesign of embedded systems: Status and trends. *IEEE Design & Test of Computers*, 15(2):45–54, April-June 1998.
9. J. R. Goodman. *Energy Scalable Reconfigurable Cryptographic Hardware for Portable Applications.* Ph.D. Thesis, Massachusetts Institute of Technology, 2000.
10. J. Großschädl. A low-power bit-serial multiplier for finite fields $\mathrm{GF}(2^m)$. In *Proceedings of the 34th IEEE International Symposium on Circuits and Systems (ISCAS 2001)*, vol. IV, pp. 37–40. IEEE, 2001.
11. J. Großschädl and G.-A. Kamendje. Instruction set extension for fast elliptic curve cryptography over binary finite fields $\mathrm{GF}(2^m)$. In *Proceedings of the 14th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2003)*, pp. 455–468. IEEE Computer Society Press, 2003.
12. N. Gura et al. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, LNCS 3156, pp. 119–132. Springer Verlag, 2004.
13. D. R. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Verlag, 2004.
14. A. Hodjat, D. Hwang, L. Batina, and I. Verbauwhede. A hyperelliptic curve crypto coprocessor for an 8051 microcontroller. In *Proceedings of the 19th IEEE Workshop on Signal Processing Systems (SIPS 2005)*, pp. 93–98. IEEE, 2005.

15. S. Janssens et al. Hardware/software co-design of an elliptic curve public-key cryptosystem. In *Proceedings of 15th IEEE Workshop on Signal Processing Systems (SIPS 2001)*, pp. 209–216. IEEE, 2001.
16. S. S. Kumar et al. Embedded end-to-end wireless security with ECDH key exchange. In *Proceedings of the 46th IEEE Midwest Symposium on Circuits and Systems (MWSCAS 2003)*, vol. 2, pp. 786–789. IEEE, 2003.
17. S. S. Kumar and C. Paar. Reconfigurable instruction set extension for enabling ECC on an 8-bit processor. In *Field Programmable Logic and Application — FPL 2004*, LNCS 3203, pp. 586–595. Springer Verlag, 2004.
18. J. López and R. Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In *Cryptographic Hardware and Embedded Systems*, LNCS 1717, pp. 316–327. Springer Verlag, 1999.
19. E. Savaş, A. F. Tenca, and Ç. K. Koç. A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m). In *Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, pp. 277–292. Springer Verlag, 2000.
20. P. Schaumont and I. Verbauwhede. Domain specific tools and methods for application in security processor design. *Design Automation for Embedded Systems*, 7(4):365–383, Nov. 2002.
21. P. Schaumont and I. Verbauwhede. Domain-specific codesign for embedded security. *Computer*, 36(4):68–74, Apr. 2003.
22. University of California at Riverside. Synthesizable VHDL Model of 8051. Available for download at `http://www.cs.ucr.edu/~dalton/i8051/i8051syn/`.
23. W. H. Wolf. Hardware-software co-design of embedded systems. *Proceedings of the IEEE*, 28(7):967–989, July 1994.
24. A. D. Woodbury, D. V. Bailey, and C. Paar. Elliptic curve cryptography on smart cards without coprocessors. In *Smart Card Research and Advanced Applications*, pp. 71–92. Kluwer Academic Publishers, 2000.

A 8-bit Architectures for Embedded Systems

Most previous work dealing with co-design of ECC for embedded systems used either an 8051-compatible microcontroller or an AVR-based processor to execute the software. Both the 8051 and the AVR platform possess a significant share of the worldwide smart card market and other security-critical segments of the embedded systems industry, e.g. sensor nodes.

A.1 The 8051 Microcontroller

The 8051 is an 8-bit microcontroller originally developed by Intel for use in embedded systems. After its launch in 1980, the 8051 has quickly gained popularity in the 1980s and early 1990s, and is today generally considered as the most widely used microcontroller of all times. There exist more than 20 independent manufacturers of 8051-compatible microcontroller cores; among these are leading semiconductor vendors like Atmel, Infineon, and Philips.

A typical 8051-compatible microcontroller includes 128 bytes of internal data RAM (IRAM), 4 kB of internal program memory (ROM), four 8-bit I/O ports and a serial port, two 16-bit timers/counters, and optionally an extended data

RAM (XRAM). Numerous enhanced variants of the "original" 8051 have been developed during the past 25 years. For instance, the 8052 features 256 bytes of internal RAM instead of 128 bytes, 8 kB of ROM instead of 4 kB, and a third 16-bit timer. Other 8051 derivatives, such as the Infineon SLE44/SLE66 families of smart card controllers, have additional 16-bit instructions and extended addressing modes for smart card applications. Both the SLE44 and the SLE66 are referred to as 16-bit smart card controllers in the data sheets, but they are fully opcode-compatible to the original 8051.

The 8051 has probably the widest range of derivatives of any embedded microcontroller on the market today, and, as a consequence, the performance of the different 8051 devices varies significantly, even when running at the same clock frequency. Each instruction executed on an original 8051 microcontroller takes either 1, 2, or 4 instruction cycles to complete, whereby a single instruction cycle corresponds to 12 clock cycles. Therefore, the original 8051 is rather slow as it can execute at most 1 million instructions per second when clocked with 12 MHz. Newer variants of the 8051 run at six, four, two, or even one clock cycle per instruction cycle, and are able to operate at clock frequencies of 100 MHz or even more. For example, the Infineon SLE66 executes instructions at a rate of two clock cycles per instruction cycle, and thus it is up to six times faster than a standard 8051 at the same clock frequency. On the other hand, the Dalton 8051 [22] requires 17 clock cycles per instruction cycle, which means that the Dalton is even slower than the original 8051 developed some 25 years ago.

A.2 The ATmega128 Microprocessor

The AVR is an 8-bit RISC architecture with 32 general-purpose registers and separate memories for program and data (Harvard architecture). All instructions have a fixed length of 16 bits. The AVR instruction set is more regular than that of the 8051, but not completely orthogonal. Arithmetic/logical instructions have a two-operand format and allow to carry out operations between two registers or between a register and an immediate (constant) value.

The AVR implementations by Atmel, such as the ATmega128, feature a two-stage pipeline and execute most instructions in a single clock cycle. Multiply instructions need a second cycle to complete. Any access to RAM requires two cycles, while reading from program memory takes three cycles. The ATmega128 has 4 kB SRAM, 128 kB Flash memory, and 4 kB EEPROM. It can be clocked with frequencies of up to 16 MHz and achieves throughputs approaching 1 MIPS per MHz. Thus, the ATmega128 outperforms the original 8051 by more than an order of magnitude at the same clock frequency. It was stated in [4] that, for certain applications, an AVR core can be a whopping 28 times faster than an 8051 running at the same clock frequency. This must be taken into account when comparing the execution times of elliptic curve cryptosystems on these two platforms. Furthermore, the ATmega128 has certain architectural features (e.g. large number of general-purpose registers, two-cycle multiply instruction) which facilitate the efficient software implementation of long integer arithmetic operations used in ECC.