An Efficient Hardware Architecture for Factoring Integers with the Elliptic Curve Method

Jens Franke[1], Thorsten Kleinjung[1], Christof Paar[2], Jan Pelzl[2],
Christine Priplata[3], Martin Šimka[4], Colin Stahlke[3]

[1] University of Bonn, Department of Mathematics, Beringstraße 1, D-53115 Bonn, Germany
{franke,thor}@math.uni-bonn.de

[2] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{cpaar,pelzl}@crypto.rub.de

[3] EDIZONE GmbH, Siegfried–Leopold–Straße 58, D-53225 Bonn, Germany
{priplata,stahlke}@edizone.de

[4] Department of Electronics and Multimedia Communications, Technical University of Košice,
Park Komenského 13, 04120 Košice, Slovak Republic
Martin.Simka@tuke.sk

Abstract

The security of the most popular asymmetric cryptographic scheme RSA depends on the hardness of factoring large numbers. The best known method for this integer factorization is the General Number Field Sieve (GNFS). One important step within the GNFS is the factorization of mid-size numbers without small prime divisors. This can be done efficiently by the Elliptic Curve Method (ECM), e.g. in special hardware.

In this work, we present an efficient hardware implementation of ECM to factor numbers up to 200 bit, which is also scalable to other bit lengths. For proof-of-concept purposes, ECM is realized as a software-hardware co-design on an FPGA and an embedded microcontroller. This appears to be the first publication of a realized hardware implementation of ECM. We adapted ECM for the requirements of efficient special hardware and provide estimates for a state-of-the-art CMOS implementation of the design and for the application of massive parallel ECM engines to the GNFS. The factorization of large integers such as RSA moduli can be improved considerably by using the ECM hardware presented.

Keywords: integer factorization, elliptic curve method, software-hardware codesign, field programmable gate array.

1 Introduction

There are several good algorithms for factoring integers, each one appropriate for a different situation. The Generalized Number Field Sieve (GNFS) is best for factoring numbers with large factors, e.g., for attacking RSA. Within GNFS, a method is needed to factor lots of smaller numbers efficiently (factorization of the cofactors, smoothness test). A good choice for this task is MPQS (Multiple Polynomial Quadratic Sieve) or ECM (Elliptic Curve Method, see [1]).

The current world record in factoring a random RSA modulus is 576 bit and was achieved with a complete software implementation of the GNFS in 2003 [2], using MPQS for the factorization of the cofactors. For larger moduli it will become crucial to use special hardware for factoring.

We propose an efficient smoothness test architecture in special hardware to support GNFS. Our aim is to minimize its AT product (area time product) such that massive parallel devices for smoothness tests become as cheap as possible. It appears that the use of ECM rather than MQPS is the better choice for this task, since MQPS requires a larger silicon area and irregular operations.

A new hardware design for the sieving step in GNFS is called SHARK [3] and has been proposed very recently. SHARK distinguishes itself from other proposed architectures for RSA factorization (see, e.g. [4]) by relying on conventional technologies and a very high modularity. However, all these architectures including software implementations on conventional PCs can be improved considerably by the massive use of highly efficient units for smoothness testing, e.g. using the proposed ECM architecture. Note that in all these variants we have to factor mid-size numbers without small prime divisors. In other variants (see [5]) with little or no sieving one has to factor many large arbitrary numbers. In this case using our ECM units gives an upper bound for the overall costs, but many improvements are possible.

ECM is an almost ideal algorithm for dramatically improving the AT product through special purpose hardware. First, it performs a very high number of operations on a very small set of input data, and is, thus, not very I/O intensive. Second, it requires relatively little memory. Lastly, it should be noted that the nature of the smoothness testing in the GNFS allows for a very high degree of parallelization. The key for efficient ECM hardware lies in fast arithmetic units. Such units for modular addition and multiplication have been studied thoroughly in the last few years, e.g., for the use in cryptographic devices using ECC.

Our hardware implementation of ECM can factor numbers up to 200 bit. The design is scalable to larger and smaller bit lengths (by changing a parameter in the VHDL code). In some range, 100 to 300 bit, say, both the time and the silicon area depend linearly on the bit length. Such a design for 125 bit perfectly fits the needs of SHARK [3]. Choosing other parameters in SHARK, massive use of our ECM units can reduce the overall costs considerably.

There are many possible improvements of the original ECM. Based on these improvements, we adapted the method to the very restrictive memory requirements of efficient hardware, thus, minimizing the AT product. The parameters used in our implementation are best suited to find factors of up to about 40 bit. For proof-of-concept purposes, ECM is realized as a software-hardware co-design on an FPGA and an embedded microcontroller. A highly efficient modular multiplication architecture described by Tenca and Koç (see [6]) is used and allows for reliable estimates for a future ASIC implementation. We propose to group about 1000 ECM units together on an ASIC and describe a controlling unit that synchronously feeds the units with programming steps, such that the ECM algorithm does not need to be stored in every single unit. In this way we keep the overall ASIC area small and still do not need much bandwidth for communication. These estimates allow us to assess the costs of massive parallel ECM engines used in a GNFS machine.

Section 2 introduces ECM and some optimizations relevant for our implementation. The choice of parameters and arithmetic algorithms, the design of the ECM unit and a possible parallelization are described in Section 3. The next two sections present our FPGA implementation, some estimates for a realization as ASIC and a case study of GNFS support with ECM hardware. The last section collects results and conclusions.

2 Elliptic Curve Method

The principles of ECM are based on Pollard's $(p-1)$-method [7]. We now describe H. W. Lenstra's Elliptic Curve Method (ECM) [1] (see also [8]).

2.1 The Algorithm

Let N be an integer without small prime factors which is divisible by at least two different primes, one of them q. Such numbers appear after trial division and a quick prime power test.

Let $E/\mathbb{Q}$ be an elliptic curve with good reduction at all prime divisors of N (this can be checked by calculating the gcd of N and the discriminant of E which very rarely yields a prime factor of N) and a point $P \in E(\mathbb{Q})$. Let $E(\mathbb{Q}) \rightarrow E(\mathbb{F}_q), Q \mapsto \overline{Q}$ be the reduction modulo q. If the order o of $\overline{P} \in E(\mathbb{F}_q)$ satisfies certain smoothness conditions described below, we can discover the factor q of N as follows:

Phase 1 Calculate $Q = kP$ where $k = \prod_{p \leq B_1 \text{ prime}} p^{e_p}$ and $e_p = \left[\frac{\log B_1}{\log p}\right]$.

Phase 2 Check for each prime $B_1 < p \leq B_2$ whether pQ reduces to the neutral element in $E(\mathbb{F}_q)$. This can be done, using the Weierstraß form and projective coordinates $pQ = (x_{pQ} : y_{pQ} : z_{pQ})$, by testing whether $\gcd(z_{pQ}, N)$ is bigger than 1.

All calculations are done modulo N. If an inversion is not possible (e.g., using affine coordinates in the Weierstraß form) a divisor is found. The parameters B_1 and B_2 control the probability of finding a divisor q. More precisely, if o factors into a product of coprime prime powers (each $\leq B_1$) and at most one additional prime between B_1 and B_2, the prime factor q is discovered.

The procedure will be repeated for other elliptic curves and starting points. To generate them, one commences with the starting point P and constructs an elliptic curve such that P lies on it.

It is possible that more than one or even all prime divisors of N are discovered simultaneously. This happens rarely for reasonable parameter choices and can be ignored by proceeding to the next elliptic curve.

Note that we can avoid all gcd computations but one at the expense of one modular multiplication per gcd by accumulating the numbers to be checked in a product modulo N and doing one final gcd.

2.2 The Elliptic Curves

Apart from the Weierstraß form there are various other forms for the elliptic curves. We use Montgomery's form (1) and compute in the set $S = E(\mathbb{Z}/N\mathbb{Z})/\{\pm 1\}$ only using the x- and z-coordinates (see [9]).

$$By^2z = x^3 + Ax^2z + xz^2 \tag{1}$$

The residue class of $\overline{P+Q}$ in this set can be computed from $\overline{P}$, $\overline{Q}$ and $\overline{P-Q}$ using 6 multiplications (2). A duplication, i.e. $\overline{2P}$, can be computed from $\overline{P}$ using 5 multiplications (3). Since we are only interested in checking whether we obtain the point at infinity for some prime divisor of N, computing in S is no restriction. In the following we will not distinguish between $E(\mathbb{Z}/N\mathbb{Z})$ and S, and pretend to do all computations in $E(\mathbb{Z}/N\mathbb{Z})$.

Addition: (2)

$$\begin{aligned} x_{P+Q} &= z_{P-Q}[(x_P - z_P)(x_Q + z_Q) + (x_P + z_P)(x_Q - z_Q)]^2 \\ z_{P+Q} &= x_{P-Q}[(x_P - z_P)(x_Q + z_Q) - (x_P + z_P)(x_Q - z_Q)]^2 \end{aligned}$$

Duplication: (3)

$$\begin{aligned} 4x_P z_P &= (x_P + z_P)^2 - (x_P - z_P)^2 \\ x_{2P} &= (x_P + z_P)^2 (x_P - z_P)^2 \\ z_{2P} &= 4x_P z_P[(x_P - z_P)^2 + 4x_P z_P (A + 2)/4] \end{aligned}$$

2.3 The First Phase

If the triple $(P, nP, (n+1)P)$ is given we can compute either $(P, 2nP, (2n+1)P)$ or $(P, (2n+1)P, (2n+2)P)$ by one addition and one duplication in Montgomery's form. Thus $Q = kP$ can be calculated using $[\log_2 k]$ additions and duplications, amounting to $11[\log_2 k]$ multiplications. In the case $z_P = 1$ we can reduce this to $10[\log_2 k]$ multiplications.

By handling each prime factor of k separately and using optimal addition chains the number of multiplications can be decreased to roughly $9.3[\log_2 k]$ (see [9]). The addition chains can be precalculated.

2.4 The Second Phase

The standard way to calculate the points pQ for all primes $B_1 < p \leq B_2$ is to precompute a (small) table of multiples kQ where k runs through the differences of consecutive primes in the interval $]B_1, B_2]$. Then, $p_0 Q$ is computed with p_0 being the smallest prime in that interval and the corresponding table entries are added successively to obtain pQ for the next prime p. There is a faster variant which additionally allows the use of Montgomery's form ([10, 9]).

The improved standard continuation uses a parameter $2 < D < B_1$. First, a table T of multiples kQ of Q for all $1 \leq k < \frac{D}{2}$, $\gcd(k, D) = 1$, is calculated. Each prime $B_1 < p \leq B_2$ can be written as $mD \pm k$ with $kQ \in T$. Now, $\gcd(z_{pQ}, N) > 1$ if and only if $\gcd(x_{mDQ} z_{kQ} - x_{kQ} z_{mDQ}, N) > 1$. Thus, we calculate the sequence mDQ (which can easily be done in Montgomery's form) and accumulate the product of all $x_{mDQ} z_{kQ} - x_{kQ} z_{mDQ}$ for which $mD - k$ or $mD + k$ is prime.

The memory requirements for the improved standard continuation are $\frac{\varphi(D)}{2}$ points for the table T and the points DQ, $(m-1)DQ$, mDQ for computing the sequence, altogether $\varphi(D) + 6$ numbers. The computational costs consist of the generation of T and the calculation of mDQ which amounts to at most $\frac{D}{4} + \frac{B_2}{D} + 7$ elliptic curve operations (mostly additions) and at most $3(\pi(B_2) - \pi(B_1))$ modular multiplications, $\pi(x)$ being the number of primes up to x. The last term can be lowered if D contains many small prime factors since this will increase the number of pairs (m, k) for which both $mD - k$ and $mD + k$ are prime. Neglecting space considerations a good choice for D is a number around $\sqrt{B_2}$ which is divisible by many small primes.

3 Methodology

This section discusses the parametrization and design of our ECM unit.

3.1 Parametrization of ECM

Our implementation focuses on the factorization of numbers up to 200 bit with factors of up to around 40 bit. Thus, “good” parameters for B_1, B_2, and D have to be found, yielding a high probability of success and a relatively small running time and area consumption. With the running time depending on the size of the (unknown) factors to be found, optimal parameters cannot be known beforehand. Hence, good parameters can be found by experiments with different prime bounds.

Deduced from software experiments, we choose $B_1 = 960$ and $B_2 = 57\,000$ as prime bounds. The value of k has 1375 bits and, assuming the simple binary method, 1374 point additions and point duplications for the execution of phase 1 are required. Due to the use of Montgomery coordinates, the coordinate z_P of the starting point P can be set to 1, thus, addition takes only 5 multiplications instead of 6. The improved phase 1 (with optimal addition chains) has to use the general case, where $z_P \neq 1$. For the sake of simplicity and a preferably simple control logic, we choose the simple method for the time being. For the chosen parameters, the computational complexity of phase 1 is 13740 modular multiplications and squarings[1].

According to Equation (3), duplicating a point $2P_A = P_C$ involves the input values x_A, z_A, A_{24} and N, where $A_{24} = (A+2)/4$ is computed from the curve parameter A (see Equation (1)) in advance and should be stored in a fixed register. A point addition $P_C = P_A + P_B$ handles the input values $x_A, z_A, x_B, z_B, x_{A-B}, z_{A-B}$ and N (see Equation (2)). Notice that during phase 1 the values N, A_{24}, x_{A-B} and z_{A-B} do not change. Furthermore, $z_{A-B} = z_1$ can be chosen to be 1. Thus, no register is required for z_{A-B}. The output values x_C and z_C can be written to certain input registers to save memory. If we assume that the ECM unit does not execute addition and duplication in parallel, at most 7 registers for the values in $\mathbb{Z}/N\mathbb{Z}$ are required for phase 1. Additionally, we will require 4 temporary registers for intermediate values. Thus, a total of 11 registers is required for phase 1.

For phase 2, the value $D = 30$ is chosen to keep the size of the table low at the cost of an increased running time. Since $\varphi(D) = 8$ is small, only 8 additional registers are required to store all coordinates in the table. Unlike in phase 1, we have to consider the general case where $z_{A-B} \neq 1$. Hence, an additional register for this quantity is needed. For the product Π of all $x_A \cdot z_B - z_A \cdot x_B$, one more register is neccessary. The temporary registers from phase 1 suffice to store the intermediate results $x_A \cdot z_B$, $z_A \cdot x_B$ and $x_A \cdot z_B - z_A \cdot x_B$. Hence, a total of 21 registers is necessary for both phases. The total computational complexity of phase 2 is 1881 point additions and 10 point duplications. Together with the 13590 modular multiplications for computing the product, 24926 modular multiplications and squarings are required.

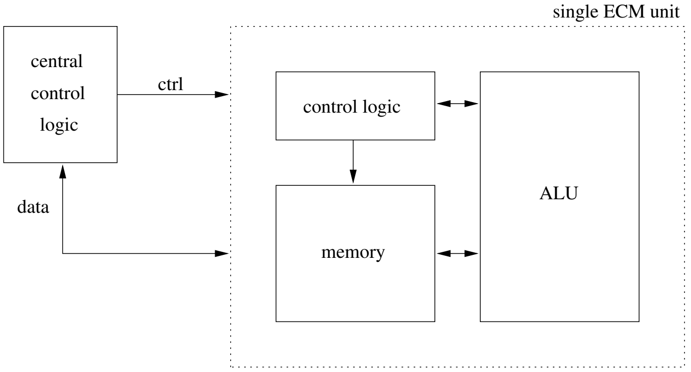
For a high probability of success ($> 80\%$), software experiments showed for the parameters given that we need to run ECM on approximately 20 different curves.

[1] In this contribution, squarings and multiplications are considered to have an identical complexity since the hardware will compute a squaring with the multiplication circuit.

3.2 Design of the ECM Unit

The ECM unit mainly consists of three parts: the ALU (arithmetic logic unit), the memory part (register) and an internal control logic (see Figure 1). Each unit has a very low communication overhead since all results are stored *inside* the unit during computation. Before the computation starts, all required initial values are assigned to memory cells of the unit. Only at the very end when the product has been computed, the result is read from the unit's memory. Commands are generated and timed by a central control logic outside the ECM unit(s).

Figure 1: Overview of one ECM Unit



3.2.1 Central Control Logic

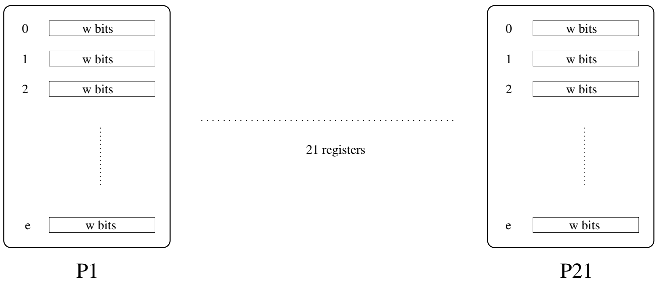
The central control logic is connected to each ECM unit via a control bus (*ctrl*). It coordinates the data exchange with the unit before and after the computation and starts each computation in the unit by a special set of commands. The commands contain an instruction for the next computation to be performed (i.e., add, subtract, multiply), including the in- and output registers to be used (P1-P21). The control bus has to offer the possibility to specify which input register(s) and which output register is active. Only certain combinations of in- and output registers occur, offering the possibility to reduce the complexity of the logic and the width of the control bus by compressing the necessary information.

If several ECM units work in parallel, only one central control logic is needed. All commands are sent in parallel to all units. Only in the beginning and in the end, the unit's memory cells have to be written and read out separately. Centralizing the control saves a lot of space, since the ECM program is rather complex.

3.2.2 Memory

Inside each unit, a register has a unique hardware address and can be addressed from outside the unit. This is mandatory since the central control logic writes data to these registers before phase 1 starts and it reads data from one of the registers after phase 2 has been finished. Each register can contain n bits and is organized in $e = \lceil \frac{n+1}{w} \rceil$ words of size w (see Figure 2). Memory access is performed word wise. Reasonable values for w are $w = 4, 8, 16, 32$ but are, though, not mandatory.

Figure 2: Memory Management



3.2.3 Arithmetic Logic Unit (ALU)

The ALU performs the arithmetic in $\mathbb{Z}/N\mathbb{Z}$, i.e., modular multiplication, modular squaring, modular addition and subtraction. Objectives for a choice of implemented algorithms are mentioned in the following subsection.

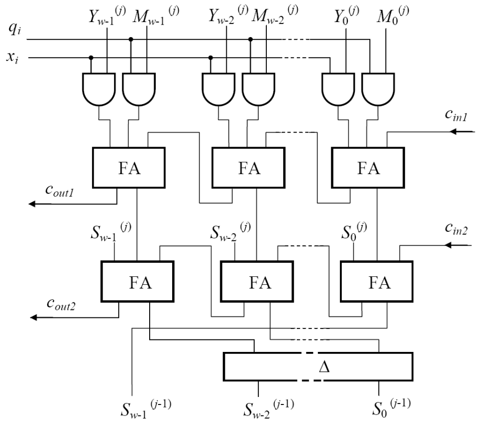
3.3 Choice of the Arithmetic Algorithms

The main purpose of the design is to synthesize an area-time optimal implementation of ECM. Low area consumption can be achieved by structures, which do not require much memory and which allow for a certain degree of serialization. In the following, we briefly describe the algorithms for modular addition, subtraction and multiplication to be implemented for the ALU. Squaring is done with the multiplication circuit, since a separate hardware circuit for squaring would increase the overall AT product. A similar approach is applied for subtraction that can be computed with an only slightly changed adder circuit.

Modular Montgomery multiplication is achieved with the very efficient architecture proposed in [6] and is scalable in terms of the bit length and word length of the operands. The multiplier architecture relies on a fast word multiplication which can be pipelined. The internal word additions are performed by simple adders. Figure 3 shows the architectures of one stage. While in [6] a structure with carry-save adders and redundant representation of operands has been implemented, we have chosen a configuration with carry-propagate adders and non-redundant representation that makes a more efficient implementation possible, especially when the target platform supports fast carry chain logic. A detailed comparison of both structures can be found in [11]. A minimal AT product of the multiplier can be achieved with a word width of 8 bit and a pipelining depth of 1 ($w = 8, p = 1$, see [6]). However, for our ECM architecture, the AT product does not only depend on the AT product of the multiplier. In fact, the multiplier only takes a comparably small part of the overall area. On the other hand, the overall speed relies primarily on the speed of the multiplier. Thus, we choose a pipelining depth of $p = 8$ for word width $w = 8$ bit, in order to achieve a higher speed. The additional area increases only marginally, hence, yielding a lower AT product.

Addition and subtraction is implemented in one circuit. As with the multiplication circuit, the operations are done word wise and the word size and number of words can be chosen arbitrary. Since the same memory is used for input and output operands, we choose the same word size as for the multiplier. The subtraction relies on the same hardware, only one control bit has to be set in order to compute a subtraction rather than an addition.

Figure 3: Multiplier Stage



Note that a modular addition needs a comparison with the modulus, whereas a modular subtraction only needs to check for a negative result, which is less expensive.

3.4 Parallel ECM

ECM can be perfectly parallelized by using different curves in parallel since the computations of each unit are completely independent. For the control of more than one ECM unit, it is essential to know that both phases, phase 1 and phase 2, are controlled completely identically, independent of the composite to be factored. Solely the curve parameter and possibly the modulus of the units and, hence, the coordinates of the initial point differ. Thus, all units have to be initialized differently which is done by simply writing the values into the corresponding memory locations sequentially[2]. During the execution of both phases, exactly the same commands can be sent to all units in parallel. Since running time of multiplication/squaring is constant (does not rely on input values) and for addition/subtraction differs at most in $2(e+1)$ clock cycles, all units can execute the same command at exactly the same time. After phase 2, the results are read from the units one after another. The required time for this data IO is negligible for one ECM unit since the computation time of both phases dominates by far. In this way even several thousand ECM units can be connected to a single control unit, initializing the units sequentially.

4 Implementation

4.1 Hardware Platform

The ECM implementation at hand is a hybrid design. It consists of an ECM unit implemented on an FPGA (Xilinx Virtex2000E-6) and a control logic implemented in software on an embedded microcontroller (ARM7TDMI, 25MHz, see [12]). The ECM unit is coded in VHDL and was synthesized for a Xilinx FPGA (Virtex2000E-6, see [13]). For the actual VHDL implementation, memory cells have been realized with the FPGA's internal block RAM.

The unit, as implemented, listens for commands which are written to a control register accessible by the FPGA. Required point coordinates and curve parameters are loaded into

[2] At a later stage, we might think of some modifications to improve the data IO of the registers.

the unit before the first command is decoded. For this purpose, these memory cells are accessible from the outside by a unique address. Internal registers, which are only used as temporary registers during the computation are not accessible from the outside.
The control of the unit(s) is done by a microcontroller present on the board and controls the data transfer from and to the units and issues the commands for all steps in phase 1 and phase 2.
Remark, that the design is done for $n = 199$ bit numbers N. Scaling the design to bit lengths from 100 to 300 bit can be easily accomplished. In this case, the AT product will de-/increase according to the size of n^2.

4.2 Results

After the synthesis and place and route, the binary image was loaded onto the FPGA. Table 1 shows the timing estimates for 40 MHz of the implementation. Phase 1 has been completely implemented. Since the control logic of phase 1 has not been optimized yet, we provide estimates for expected timings for phase 1 (and phase 2) based on the timings of the elementary operations.

Table 1: Average Running Time Estimates of the ECM Implementation (195 bit modulus), $p = 8$, $w = 8$ (Xilinx Virtex2000E-6, 40 MHz)

Operation	Time
modular addition	2 μs
modular subtraction	1.3 μs
modular multiplication	16.7 μs
modular squaring	16.7 μs
point addition	93.6 μs
point duplication	89.8 μs
Phase 1	252 ms
Phase 2	438 ms

The time for the initialization is not taken into account, since it only delays the computation at the very beginning and the very end.

5 A Case Study: Supporting GNFS with ECM Hardware

Building an efficient and cheap ECM hardware can influence the overall performance of the GNFS since ECM can be perfectly used for smoothness testing within the GNFS. In this section, we briefly estimate the costs, space requirements and power consumption of a special ECM hardware as ASIC (*Application Specific Integrated Circuit*). This special hardware could be produced as single ICs (such as common CPUs), ready for the use in larger circuits.

5.1 Estimation of the Running Time

We can determine the running time of both phases on basis of the underlying ring arithmetic. The upper bounds for the number of clock cycles of a modular addition and a modular subtraction for a setting with $n = 199$, $w = 8$ and $e = 25$ are determined by $T_{add} = 3(e + 1) = 78$ and $T_{sub} = 2(e + 1) = 52$ cycles. According to [6], the implemented multiplier requires

$$T_{mul} = \left\lceil \frac{n}{p} \right\rceil (e + 1) + 2p$$

clock cycles per multiplication. Hence, a 195 bit multiplication requires $T_{mul} = 666$ cycles. For each operation we should include $T_{init} = 2$ clock cycles for the initialization of the ALU at the beginning of each computation.

For the operations on the elliptic curve for phase 1 we obtain

$$\begin{aligned} T_{Padd} &= 5T_{mul} + 3T_{add} + 3T_{sub} + 11T_{init} = 3742 \quad \text{and} \\ T_{Pdpl} &= 5T_{mul} + 2T_{add} + 2T_{sub} + 9T_{init} = 3608 \end{aligned}$$

clock cycles. For phase 2, T_{Padd} changes to $T'_{Padd} = 4410$ since $z_{A-B} \neq 1$ in most cases, hence, we have to take the multiplication with z_{A-B} into account.
The total cycle count for both phases is

$$\begin{aligned} T_{Phase\ 1} &= 1374(T_{Padd} + T_{Pdpl}) = 10098900 \quad \text{and} \\ T_{Phase\ 2} &= 1881T'_{Padd} + 50T_{Pdpl} + 13590T_{mul} = 17553730 \end{aligned}$$

clock cycles. Excluding the time for pre- and postprocessing, a unit needs approximately $27.7 \cdot 10^6$ clock cycles for both phases on one curve. If we assume a frequency of 500 MHz, such a complex computation can be performed in approximately 55 ms.

5.2 Estimation of Area Requirements

According to [6][3], the multiplier with $w = 8, p = 8$ requires 21 400 transistors in standard CMOS technology (assuming 4 transistors per NAND gate). We assume that the circuit for addition and subtraction can be achieved with at most 1000 transistors. For the memory, we assume (area expensive) statical RAM which requires 25 200 transistors for 21 registers. For the control inside the unit and the control of the whole circuit, we assume additional 8000 transistors per ECM unit. Hence, one unit requires approximately 55 600 transistors. Assuming the CMOS technology of a standard Pentium 4 processor (0.13 μm, approximately 55 million transistors), we could fit 990 ECM units into the area of one standard processor. One ECM unit needs an area of approximately 0.1475 mm^2 and has a power dissipation of approximately 40 mW.

5.3 Application to the GNFS

Considering the architecture for a special GNFS hardware of [3], we have to test approximately $1.7 \cdot 10^{14}$ cofactors up to 125 bit for smoothness within a year. Since both the running time as well as the area requirement scales linearly with the bit size, we can multiply the results from the subsections above with a factor of $125/199 \approx 0.628$. If we distribute the computation over a whole year, we have to check 5.4 million cofactors per second. For a probability of success of $> 80\%$, we test 20 curves per cofactor, thus, we need

[3] Remark: The numbers provided in that contribution refer to a multiplier built with carry-save adders. Since we implemented the architecture with carry-propagate adders, the given number of 21 400 transistors is larger (approximately 20%) than what would be achieved with our design.

approximately 3.7 million ECM units which would yield a total chip area of 350 000 mm^2 (= 2400 ICs of the size of a Pentium 4) and a power consumption of approximately 100 kW. If we assume a cost of US$ 5000 per 300 mm wafer, the ECM units would cost less than US$ 25 000 for the whole GNFS architecture which is negligible towards the US$ 160 million for the rest of the machine. A different choice of parameters uses much more ECM and can reduce the costs of SHARK considerably.

6 Conclusions

The work at hand presents the first ECM hardware implementation. The implementation is a hardware-software codesign and has been implemented on an FPGA and an ARM microcontroller for factoring integers of a size of up to 199 bit. We implemented a variant of ECM which allows for a very low area time product and, hence, is very cost effective. Our implementation impressively shows that due to very low area requirements and low data IO, ECM is predestinated for the use in hardware. A single unit for factoring composites of up to 199 bit requires 519 flip-flops, 857 lookup-tables and 22 BlockRAMs (less than 2% of logic and 13% of memory resources of the Xilinx Virtex2000E device). Regarding a possible ASIC implementation, around 990 ECM units could be placed on a single Pentium4-sized chip.
As demonstrated, ECM can be perfectly parallelized and, thus, an implementation at a larger scale can be used to assist the GNFS factoring algorithm by carrying out the factorization of the cofactors. A low cost ASIC implementation of ECM can decrease the overall costs of the GNFS architecture SHARK, as shown in [3].

As future steps, the control logic for the second phase will be implemented. Variants of phase 2 can be examined in order to achieve the lowest possible AT product. To achieve a higher maximal clock frequency of the ECM unit, we need to focus on a better structure of the control logic inside the unit.
Furthermore, most of the computation time is spent for modular multiplications, and an improvement of the implementation of the multiplication will directly affect the overall running time. With the VHDL source code at hand, the next logical step is the design and simulation of a full custom ASIC containing the logic, which currently is implemented in the FPGA. For an ASIC implementation, a parallel design of many ECM units is preferable. The control can still be handled outside the ASIC by a small microcontroller, as it is the case with the work at hand. Alternatively, a soft core of a small microcontroller can be adopted to the specific needs of ECM and be implemented within the ASIC. With a running ECM ASIC, exact cost estimates for the support of algorithms such as the GNFS can be obtained.

Acknowledgement: We would like to thank Miloš Drutarovský and Viktor Fischer for providing assistance with programming the arithmetic logic unit.

References

[1] H. W. Lenstra, "Factoring Integers with Elliptic Curves", *Annals of Mathematics*, vol. 126, no. 2, pp. 649–673, 1987.

[2] J. Franke, T. Kleinjung et al., "RSA-576", Email announcement, `http://www.crypto-world.com/announcements/rsa576.txt`, 2003.

[3] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, C. Priplata and C. Stahlke, "SHARK — A Realizable Special Hardware Sieving Device for Factoring 1024-bit Integers", in *Special-Purpose Hardware for Attacking Cryptographic Systems — SHARCS 2005*, Paris, 2005.

[4] A. Shamir and E. Tromer, "Factoring Large Numbers with the TWIRL Device", in *Advances in Cryptology — Crypto 2003*, vol. 2729 of *LNCS*, pp. 1–26, Springer, 2003.

[5] D. Bernstein, "Circuits for Integer Factorization: A Proposal", Manuscript, `http://cr.yp.to/papers.html#nfscircuit`, 2001.

[6] A. Tenca and Ç.K. Koç, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm", *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1215–1221, 2003.

[7] J. Pollard, "A Monte Carlo Method for Factorization", *Nordisk Tidskrift for Informationsbehandlung (BIT)*, vol. 15, pp. 331–334, 1975.

[8] H. Cohen, "A Course in Computational Algebraic Number Theory", Graduate Texts in Mathematics, Springer, 1993.

[9] P. Montgomery, "Speeding up the Pollard and elliptic curve methods of factorization", *Mathematics of Computation*, vol. 48, pp. 243–264, 1987.

[10] R. P. Brent, "Some Integer Factorization Algorithms Using Elliptic Curves", in *Australian Computer Science Communications 8*, pp. 149–163, 1986.

[11] M. Drutarovský, V. Fischer and M. Šimka, "Comparison of two implementations of scalable Montgomery coprocessor embedded in reconfigurable hardware", in *Proceedings of the XIX Conference on Design of Circuits and Integrated Systems — DCIS 2004* (Bordeaux, France), pp. 240–245, Nov. 24–26, 2004.

[12] ARM Limited, "ARM7TDMI (Rev 3) — Technical Reference Manual", `http://www.arm.com/pdfs/DDI0029G_7TDMI_R3_trm.pdf`, 2001.

[13] Xilinx, "Virtex-E 1.8V Field Programmable Gate Arrays — Production Product Specification", `http://www.xilinx.com/bvdocs/publications/ds022.pdf`, June 2004.