Multicore Curve-Based Cryptoprocessor with Reconfigurable Modular Arithmetic Logic Units over $\mathrm{GF}(2^n)$

Kazuo Sakiyama, *Student Member*, *IEEE*, Lejla Batina, *Member*, *IEEE*, Bart Preneel, *Member*, *IEEE*, and Ingrid Verbauwhede, *Senior Member*, *IEEE*

Abstract—This paper presents a reconfigurable curve-based cryptoprocessor that accelerates scalar multiplication of Elliptic Curve Cryptography (ECC) and HyperElliptic Curve Cryptography (HECC) of genus 2 over $\mathrm{GF}(2^n)$. By allocating α copies of processing cores that embed reconfigurable Modular Arithmetic Logic Units (MALUs) over $\mathrm{GF}(2^n)$, the scalar multiplication of ECC/HECC can be accelerated by exploiting Instruction-Level Parallelism (ILP). The supported field size can be arbitrary up to $\alpha(n+1)-1$. The superscaling feature is facilitated by defining a single instruction that can be used for all field operations and point/divisor operations. In addition, the cryptoprocessor is fully programmable and it can handle various curve parameters and arbitrary irreducible polynomials. The cost, performance, and security trade-offs are thoroughly discussed for different hardware configurations and software programs. The synthesis results with a 0.13-μm CMOS technology show that the proposed reconfigurable cryptoprocessor runs at 292 MHz, whereas the field sizes can be supported up to 587 bits. The compact and fastest configuration of our design is also synthesized with a fixed field size and irreducible polynomial. The results show that the scalar multiplication of ECC over $\mathrm{GF}(2^{163})$ and HECC over $\mathrm{GF}(2^{83})$ can be performed in 29 and 63 μs, respectively.

Index Terms—Multiprocessor systems, processor architectures, reconfigurable hardware, arithmetic and logic units, public key cryptosystems.

1 Introduction

Since Diffie and Hellman introduced the idea of Public-Key Cryptography (PKC) [1] in the mid-1970s, public-key cryptosystems have been an essential building block for digital communication. PKC allows for secure communications over insecure channels without prior exchange of a secret key. It can offer both key exchange and digital signature. The most popular and most widely used PKCs are RSA [2] and Elliptic Curve Cryptography (ECC) [3], [4]. In embedded systems, ECC is considered a more suitable choice than RSA because ECC obtains higher performance, lower power consumption, and smaller area on most platforms. Another appealing candidate for PKC is Hyper-Elliptic Curve Cryptography (HECC). Recently, many software and hardware implementations of HECC have been described, whereas more theoretical work has shown that HECC is also secure for curves with a small genus [5]. Nevertheless, the performance is still much slower than one for private-key cryptography, such as AES [6].

Implementing a fast PKC is a challenge for most application platforms, varying from software to hardware. For the choice of the implementation platform, several factors have to be taken into account. Application-Specific Integrated Circuit (ASIC) solutions provide the speed and more physical security, but their flexibility is limited. For that property, software solutions are needed; however, a pure software solution is not a feasible option because of low performance. Application-Specific Instruction set Processor (ASIP) architectures based on hardware/software codesign potentially allow an efficient design platform that offers a trade-off between cost, performance, and security.

A considerable amount of work has been reported on improving the performance of Elliptic Curve (EC) scalar multiplication. The work can be classified into the following categories: First, mathematical investigations have been reported for various types of ECs, for example, Koblitz curves [7]. Second, various algorithms for scalar multiplication have been proposed and criteria for improvements include performance and side-channel security. One of the best-known examples that meets both requirements is Montgomery's powering ladder [8]. Various types of coordinates have been prepared as well as a number of approaches to speed up finite-field arithmetic. Last, architecture-level improvements can be considered from a hardware implementation point of view.

The first contribution of this paper is the acceleration of curve-based cryptosystems by deploying a superscalar architecture. The solution is algorithm independent and can be applied to any scalar multiplication algorithm. We discuss the improvement of the performances for ECC and HECC over binary fields. Some previous work reported parallel use of modular arithmetic units to accelerate scalar multiplication [9], [10], [11], [12], [13], [14]. In the papers, point/divisor doubling and addition operations are reformulated so that they can take advantage of the parallel processing. On the

• *The authors are with the Department of Electrical Engineering, Katholieke Universiteit Leuven, ESAT/SCD-COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium.*
E-mail: {ksakiyam, lbatina, preneel, iverbauw}@esat.kuleuven.be.

Manuscript received 26 Dec. 2006; revised 31 Mar. 2007; accepted 24 Apr. 2007; published online 3 May 2007.
Recommended for acceptance by S. Nikoletseas.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0485-1206.
Digital Object Identifier no. 10.1109/TC.2007.1071.

other hand, our proposed architecture embeds an instruction scheduler that explores the highest level of parallelism and assigns tasks for the processing units in an optimal way. This way, the parallelism within the operations can be found *on the fly* by *dynamically* checking the data dependencies in the instructions.

The second contribution of this research is the design of a reconfigurable data path over binary fields. In order to support multiple curve-based cryptosystems and various field sizes for them, it is necessary to provide different field-length modular operations, for example, 193 bits for ECC and 97 bits for HECC. In [15], Satoh and Takano solve this problem by using an r-bit $\times$ r-bit multiplier and applying an algorithm originally used for software implementations. In other words, the advantage of their solution is that the operand size can be freely chosen, limited only by the size of the memory for storing intermediate variables. The area and time complexities of modular multiplications are considered, respectively, as $O(r^2)$ and $O(m^2)$, where $n = m \cdot r$ is the field size. For high-speed modular multiplications, the parameter r needs to be large and the data path delay of the multiplier becomes longer. In contrast, the critical path delay is independent of the field size in our solution, where an n-bit $\times$ d-bit digit-parallel multiplier is used. Furthermore, the organization of the multiplier can be reconfigured by changing the interconnections between processing cores that have Modular Arithmetic Logic Unit (MALU) and memory. The so-called coarse-grained reconfigurable data path offers high-speed modular multiplications, efficient use of hardware resource for various field sizes, and support for arbitrary field sizes by providing enough MALU cores.

The third contribution of this paper is a fair comparison between ECC and HECC. For HECC of genus 2, the field size is two times smaller than the one for ECC for the same level of security [16]. Our programmable architecture enables one to use the same hardware design for the two curve-based cryptosystems. Moreover, we also explore different arithmetic operations in the MALU and examine the effects on the level of the parallelism in ECC and HECC. As a result, we discuss the trade-offs between cost, performance, and security.

The remainder of this paper is organized as follows: Section 2 gives a survey of relevant previous work and some background information for implementations of curve-based cryptography. In Section 3, the architecture of our proposed cryptoprocessor is explained. The performance is evaluated with a system-level simulation and the results are reported in Section 4. The details of our implementation are introduced in Section 5 and the results are shown for various implementation options in Section 6. Section 7 concludes the paper.

2 Curve-Based Cryptography

Here, we consider some background information for curve-based cryptography over binary fields: For hyperelliptic curves, we are interested in genus 2 curves only. We mention the basic algorithms and the structure of the operations. Good references for the mathematical background are [17], [16], [18].

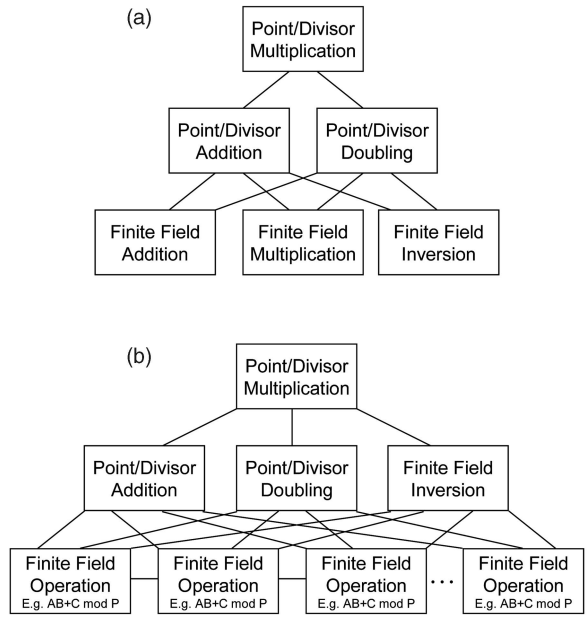


Fig. 1. Scheme of the hierarchy for ECC/HECC operations. (a) Conventional hierarchy. (b) Proposed hierarchy using a single finite-field operation at the lowest level.

The main operation in any curve-based primitive is point/divisor multiplication (aka scalar multiplication). The general hierarchical structure for operations required for implementations of curve-based cryptography is given in Fig. 1a. Point/Divisor multiplication is at the top level. At the next (lower) level are the point/divisor group operations. The lowest level consists of finite-field operations, such as finite-field addition, multiplication, and inversion, required to perform the group operations. The only difference between ECC and HECC is the sequence of operations in the middle level. The sequence for HECC is more complex when compared with the ECC point operations; however, HECC uses shorter operands. One can also perform inversion with a chain of multiplications [19] and only provide hardware for finite-field addition and multiplication. The corresponding hierarchy is illustrated in Fig. 1b. The hierarchy uses several copies of operation units at the lowest level to accelerate point/divisor group operations and inversions. We use this structure for our cryptoprocessor.

2.1 ECC over a Binary Field

ECC relies on a group structure induced on an EC. A set of points on an EC (with one special point added, that is, the so-called point at infinity $\mathcal{O}$), together with a point addition as a binary operation, has the structure of an abelian group. As we consider a finite field of characteristic 2, that is, $\mathrm{GF}(2^n)$, a nonsupersingular EC E over $\mathrm{GF}(2^n)$ is defined as the set of solutions $(x, y) \in \mathrm{GF}(2^n) \times \mathrm{GF}(2^n)$ of the equation

$$y^2 + xy = x^3 + ax^2 + b, \tag{1}$$

where $a, b, x, y \in \mathrm{GF}(2^n)$ and $b \neq 0$, together with $\mathcal{O}$. The inverse of the point $P = (x_1, y_1)$ is $-P = (x_1, -y_1)$. The sum $P + Q$ of the points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ ($P, Q \neq \mathcal{O}$ and $P \neq \pm Q$) is the point $R = (x_3, y_3)$. Here,

$$\begin{aligned}\lambda &= \frac{y_1 + y_2}{x_1 + x_2},\\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a,\\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1.\end{aligned} \tag{2}$$

This operation is called point addition. For $P = Q$, the point doubling formulas are

$$\begin{aligned}\lambda &= \frac{y_1}{x_1} + x_1,\\ x_3 &= \lambda^2 + \lambda + a,\\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1.\end{aligned} \tag{3}$$

The point at infinity $\mathcal{O}$ is the neutral element, similar to the number 0 in ordinary addition. Thus, $P + \mathcal{O} = P$ and $P + (-P) = \mathcal{O}$ for all points P.

The scalar multiplication, that is, the multiplication of a point P on the curve with a scalar k is the main operation for ECC. The scalar multiplication kP can be computed as a combination of sequential point doublings and point additions. There are several computation sequences to compute point doubling and addition, including the recommendation in IEEE P1363 [20]. The selection of the sequences also has a great impact on the performance and cost of curve-based cryptosystems. The sequences are generally implemented as a controller block (for example, Finite-State Machine (FSM)) in hardware design. The number of modular multiplications and the required memory size vary according to the sequence. Moreover, some sequences imply parallelism in point doubling/addition. In this sense, hardware/software codesign, where the arithmetic is executed on hardware acceleration units while the sequences run in software, is an attractive choice for curve-based cryptoprocessors because it offers the equivalent performance of an ASIC while maintaining the flexibility to support a wide range of curve options.

2.2 HECC of Genus 2

Let $\overline{\mathrm{GF}}(2^n)$ be an algebraic closure of the field $\mathrm{GF}(2^n)$. Here, we consider a hyperelliptic curve C of genus $g = 2$ over $\mathrm{GF}(2^n)$, which is given in the form

$$C : y^2 + h(x)y = f(x) \quad \text{in} \quad \mathrm{GF}(2^n)[x, y], \tag{4}$$

where $h(x) \in \mathrm{GF}(2^n)[x]$ is a polynomial of degree $deg(h) \leq g$ and $f(x)$ is a monic polynomial of degree $deg(f) = 2g + 1$. Also, there are no solutions $(x, y) \in \overline{\mathrm{GF}}(2^n) \times \overline{\mathrm{GF}}(2^n)$ that simultaneously satisfy (4) and the equations $2v + h(u) = 0$ and $h'(u)v - f'(u) = 0$. These points are called singular points. For the genus 2, in the general case, the following equation is used:

$$\begin{aligned}&y^2 + (h_2x^2 + h_1x + h_0)y =\\ &x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0.\end{aligned}$$

A divisor D is a formal sum of points on the hyperelliptic curve C, that is, $D = \sum m_P P$, and its degree is $deg(D) = \sum m_P$. Let Div denote the group of all divisors on C and Div_0 the subgroup of Div of all divisors with degree zero. The Jacobian J of the curve C is defined as the quotient group $J = Div_0/P$. Here, P is the set of all principal divisors, where a divisor D is called principal if $D = div(f)$ for some element f of the function field of C ($div(f) = \sum_{P \in C} ord_P(f)P$). The discrete logarithm problem in the Jacobian is the basis of security for HECC. We use the Mumford representation, according to which each divisor is represented as a pair of polynomials $[u, v]$, where u is monic of degree 2, $deg(v) < deg(u)$, and $u | f - hv - v^2$ (the so-called reduced divisors). For implementations of HECC, we need to implement the multiplication of elements of the Jacobian, that is, divisors with some scalar.

2.3 Algorithms for Our Implementations

In our implementations, the scalar multiplication of ECC is achieved by two different computation sequences: The first is from the recommendation of the IEEE P1363 and the second is based on the idea of the Montgomery's powering ladder of López and Dahab (denoted as ECC_M in this paper) [21]. With regard to the scalar multiplication of HECC, we use the formulas of Byramjee and Duquesne [22]. All of the sequences use projective coordinates and we apply the binary nonadjacent form (NAF) or the windowed NAF method for scalar multiplication [16], [23], except for ECC_M. This way, the scalar is decomposed as an NAF and scalar multiplication is performed with a lower cost than the binary method. Modular inversion is performed with a chain of modular multiplications repeatedly [19]. The total number of modular multiplications required for the modular inverse is $\{\lfloor log_2(n-1) \rfloor + w(n-1) - 1 + \ (n-1)\}$. Since we need only one modular inversion for each scalar multiplication of ECC and HECC, the inversion cost is not a serious bottleneck. The details will be discussed in Section 6.

As our data path performs one basic operation, $AB + C$ or $A(B + D) + C$, over a binary field, we have rewritten the sequences of point/divisor doubling and addition to obtain an optimal usage of our new data path. For example, the formulas for the mixed addition of HECC includes 48 operations of $A(B + D) + C$ instead of six squarings, 34 multiplications, and a lot of additions. Note that our strategy for refining the sequences also minimizes the number of intermediate variables to save hardware resource. As a result, scalar multiplication can be performed with at most 16 and 32 registers, respectively, for ECC (including ECC_M) and HECC.

3 Architecture of the Curve-Based Cryptoprocessor

3.1 System Architecture

The proposed architecture of the curve-based cryptoprocessor is composed of the main controller, several MALU cores, and the Register Files (RFs) that store intermediate variables and share them with the MALU cores. The block diagram of the cryptoprocessor is illustrated in Fig. 2. The hardware configuration of the cryptoprocessor is flexible to provide from the smallest to the fastest implementation, depending on the target application. Some components can be added or removed, as will be explained in the next sections.

The main CPU communicates with the cryptoprocessor through memory-mapped I/O (for example, a Static RAM (SRAM) interface) and has three types of 32-bit inputs and outputs: One of them is a signal that tells the controller to

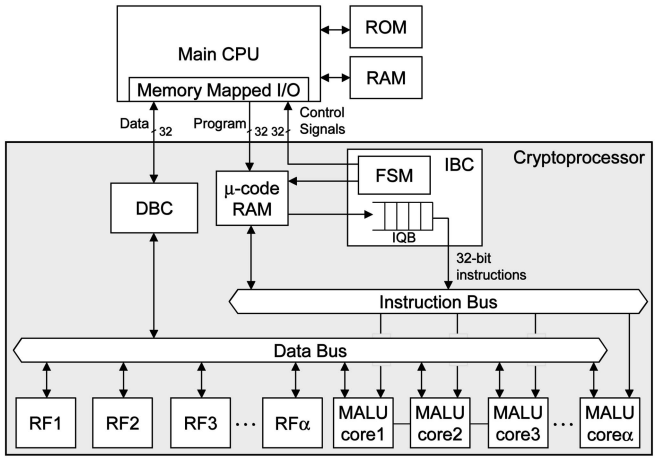


Fig. 2. Block diagram for the system architecture with the curve-based cryptoprocessor.

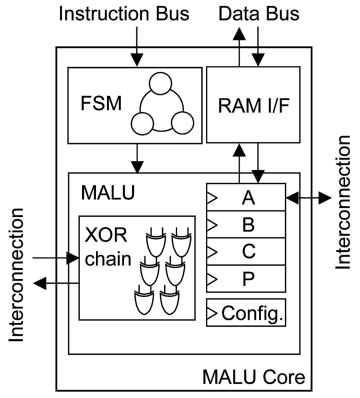


Fig. 3. Block diagram for the architecture of the MALU core for $\mathrm{GF}(2^n)$ operations. Interconnections are determined by the configuration register.

stop sending instructions when the instruction buffer is full. A 32-bit I/O passes data backward and forward between the main CPU and the cryptoprocessor and a 32-bit output is used to send instructions. The data transfer between the main CPU and the cryptoprocessor is controlled by a Data Bus Controller (DBC). If the intermediate variables for ECC/HECC operations are stored in the SRAM attached to the main CPU, then the cryptoprocessor can be constructed without use of the RFs. However, the I/O transfer overhead becomes the bottleneck of the performance. Hence, the RFs have to be embedded in the cryptoprocessor for the purpose of reducing the data transfer overhead. This way, the path through the DBC is only activated when an initial point and the curve parameters are sent to the RFs or when the result of a scalar multiplication is retrieved.

Instructions are sent to the MALU cores either from the main CPU or from preset microcodes in the μ-code RAM. When the main CPU is in charge of dispatching instructions, the Instruction Bus Controller (IBC) block can be detached from the cryptoprocessor. In this case, typically, the throughput of issuing instructions is not high enough for the MALU cores to be utilized effectively. However, if the μ-code RAM is used for assisting the main CPU, then the IBC can handle one instruction per cycle. For instance, the sequence of point doubling is stored in the μ-code RAM and the main CPU calls it a single instruction. Thus, multiple MALU cores can be activated in parallel without any instruction stalls. During point/divisor multiplications, the IBC keeps on reading instructions from the μ-code RAM and stores them in an Instruction Queue Buffer (IQB), unless the IQB is full. The IBC checks if there is Instruction-Level Parallelism (ILP) by checking the data dependency of instructions in the IQB and forwards them to the MALU(s) (see Section 3.6).

3.2 Architecture of the MALU Core

The data path plays an important role in accelerating scalar multiplication. One way to implement an efficient data path is to use a specific irreducible polynomial, for example, a trinomial such as $P(x) = x^{193} + x^{15} + 1$. In this case, modular multiplications can be implemented efficiently and the squaring operation only needs several modular adders if it is implemented separately from the multiplier. The critical path delay of the squarer is low enough to use it as a one-cycle operation. Likewise, the modular inversion can be efficiently implemented [24]. Therefore, three different modular operations can be used for the data path. However, this dedicated approach makes the data path inflexible in the size of the operand (that is, the field size).

In contrast, our proposed MALU core is a flexible processor that executes a single operation on a finite field over $\mathrm{GF}(2^n)$, for example, $A(x)B(x) + C(x) \pmod{P(x)}$. Also, the irreducible polynomial $P(x)$ can be chosen arbitrarily. The core, as illustrated in Fig. 3, decodes an incoming instruction in the FSM, loads operands from the RF, executes the finite-field operation by the data path (the MALU), and writes back the result into the RF. All operations necessary for scalar multiplication of the curve-based cryptography can be processed by using the single core iteratively, including modular inversions.

In order to exploit parallelism in scalar multiplications, multiple MALU cores can be instantiated in the cryptoprocessor. The intermediate variables are then shared with the MALU cores through the data bus. The RF architecture is discussed in detail in Section 3.4, since it is one of the most critical blocks in multicore systems. Another advantage of our multicore system is that wider field sizes can be supported by reconfiguring the data path. That is, our proposed core has additional ports that are used for interconnecting MALUs in neighboring cores by setting the configuration register. Thus, we can construct a new data path that can handle larger operands. The details are explained in Section 3.3.

3.3 Reconfigurable Data Path

In this section, the architecture for the MALU is explained. The MALU is a data path that is based on an MSB-first bit-serial polynomial-basis $\mathrm{GF}(2^n)$ multiplier, as illustrated in Fig. 4a. This is a hardware implementation that computes $A(x)B(x) \pmod{P(x)}$, where $A(x) = \sum_{i=0}^{n-1} a_i x^i$, $B(x) = \sum_{i=0}^{n-1} b_i x^i$, and $P(x) = x^n + \sum_{i=0}^{n-1} p_i x^i$. The intermediate result $T(x) = \sum_{i=0}^{n} t_i x^i$ is stored in a register. The case for the digit-serial multiplier is shown in Algorithm 1.

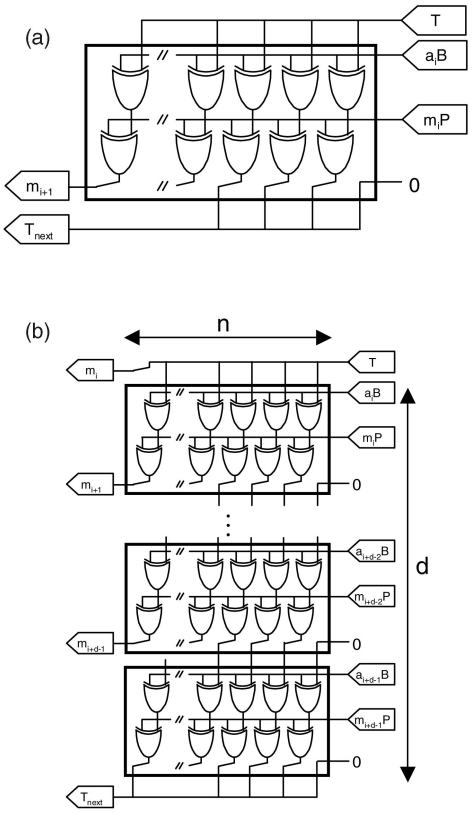


Fig. 4. Block diagram for one MALU—extension of the digit size. (a) The building block corresponding to Algorithm 1. (b) $\mathrm{GF}(2^n)$ multiplier with the digit size d.

Algorithm 1. Bit-serial MSB-first modular Multiplication over $\mathrm{GF}(2^n)$.
INPUT: $A(x) = a_{n-1}x^{n-1} + \cdots + a_1x + a_0$,
$B(x) = b_{n-1}x^{n-1} + \cdots + b_1x + b_0$,
$P(x) = x^n + p_{n-1}x^{n-1} + \cdots + p_1x + p_0$
OUTPUT: $A(x)B(x) \bmod P(x)$
1. $T(x) = 0$;
2. For $(i = n-1; i \geq 0; i = i-1)$ then
3. $\quad m_i = t_n$;
4. $\quad T(x) = (T(x) + a_iB(x) + m_iP(x))x$;
5. Return $T(x)/x$;

The MALU XORs three inputs, which are $a_iB(x)$, $m_iP(x)$, and $T(x)$, and then outputs the next intermediate result $T(x)$ by computing

$$T(x) = (T(x) + a_iB(x) + m_iP(x))x, \tag{5}$$

where $m_i = t_n$. By providing $T(x)$ as the next input and repeating the same computation n times, one can obtain the result $A(x)B(x)$ (see [25]). Moreover, by providing $B(x) + D(x)$ in place of $B(x)$ and XORing the result with $C(x)$, the operation form $A(x)(B(x) + D(x)) + C(x) \pmod{P(x)}$ can also be supported.

The proposed data path is scalable in the digit size d (vertical direction in Fig. 4b). The corresponding algorithm can be obtained by loop unrolling Algorithm 1, as shown in Algorithm 2. In this case, one operation finishes in $\lceil n/d \rceil$ cycles. Thus, the appropriate digit size can be parameterized in the data path design and can be determined by exploring the best combination of cost and performance.

Algorithm 2. Digit-serial MSB-first modular Multiplication over $\mathrm{GF}(2^n)$.
INPUT: $A(x) = a_{n-1}x^{n-1} + \cdots + a_1x + a_0$,
$B(x) = b_{n-1}x^{n-1} + \cdots + b_1x + b_0$,
$P(x) = x^n + p_{n-1}x^{n-1} + \cdots + p_1x + p_0$
OUTPUT: $A(x)B(x) \bmod P(x)$
1. $T(x) = 0$;
2. For $(i = d\lceil \frac{n}{d} \rceil - 1; i \geq 0; i = i-d)$ then
3-1. $\quad m_i = t_n$; $q_i = t_{n+1}$;
4-1. $\quad T(x) = (T(x) + a_iB(x) + m_iP(x))x$;
3-2. $\quad m_{i+1} = t_n$; $q_{i+1} = t_{n+1}$;
4-2. $\quad T(x) = (T(x) + a_{i+1}B(x) + m_{i+1}P(x))x$;
$\quad \vdots$
3-d. $\quad m_{i+d-1} = t_n$; $q_{i+d-1} = t_{n+1}$;
4-d. $\quad T(x) = (T(x) + a_{i+d-1}B(x) + m_{i+d-1}P(x))x$;
5. Return $T(x)/x$;

The field size n is determined by the key length. A larger field size can also be obtained by interconnecting several MALUs in the horizontal direction. Hence, various implementation options can be chosen with the MALU. For instance, the cryptoprocessor can support arbitrary field sizes up to 587 bits when using six copies of the MALU cores, each of which supports a field size of 97 bits.

The schematic circuit diagram illustrated in Fig. 5 describes how the MALUs are reconfigured for supporting different field sizes. When each of the MALUs is used independently without interconnections for the purpose of parallel processing, $cfg1$ is set to zero so that a d-bit vector $\mathbf{m} = (m_{i+d-1}, \cdots, m_{i+1}, m_i)$ in Algorithm 2 can be used for the modular reduction in its own core. More precisely, the vector $\mathbf{m}$ determines whether the irreducible polynomial should be XORed with the intermediate result so that the degree of $T(x)$ can be at most n or $deg(T) \leq n$. This way, the LSBs of $T(x)$ can always be 0 because they are provided by another d-bit vector $\mathbf{q} = (q_{i+d-1}, \cdots, q_{i+1}, q_i)$ (see Algorithm 2) of the neighboring core. This corresponds to the 1-bit left-shift operation.

On the other hand, when supporting a wider field-sized data path by interconnecting several MALU cores, each vector $\mathbf{m}$ is exchanged with one from the neighboring core. For instance, α copies of the MALU over $\mathrm{GF}(2^n)$ with digit size d, that is, $\mathrm{MALU}_{n\times d}$, can be reconfigured as one $\mathrm{MALU}_{(\alpha(n+1)-1)\times d}$. Suppose that $\alpha = 3$ in Fig. 5 and the MALU1, MALU2, and MALU3 are reconfigured to make a data path for the triple field size (more precisely $3n+2$). The configuration signals in the MALU1, MALU2, and MALU3 should be set to 0, 1, and 1, respectively.

3.4 Architecture of the RF

If the MALU supports the operation $A(x)(B(x) + D(x)) + C(x) \pmod{P(x)}$, then four different operands need to be read from the RF and the result is written back to the RF after completing the execution. When using three MALU cores, for instance, 12 read and three write operations occur for three parallel executions. This heavy memory access was one of the bottlenecks in our previous multicore cryptoprocessor [26]. In order to reduce the memory-access cycles,

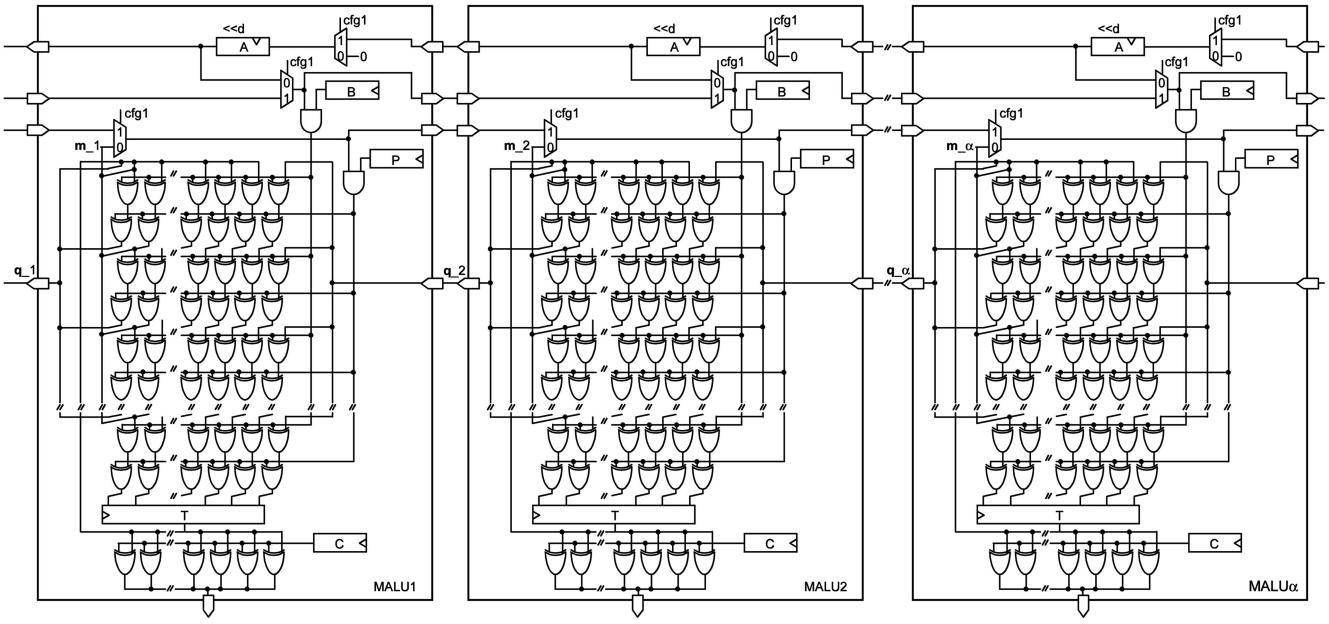
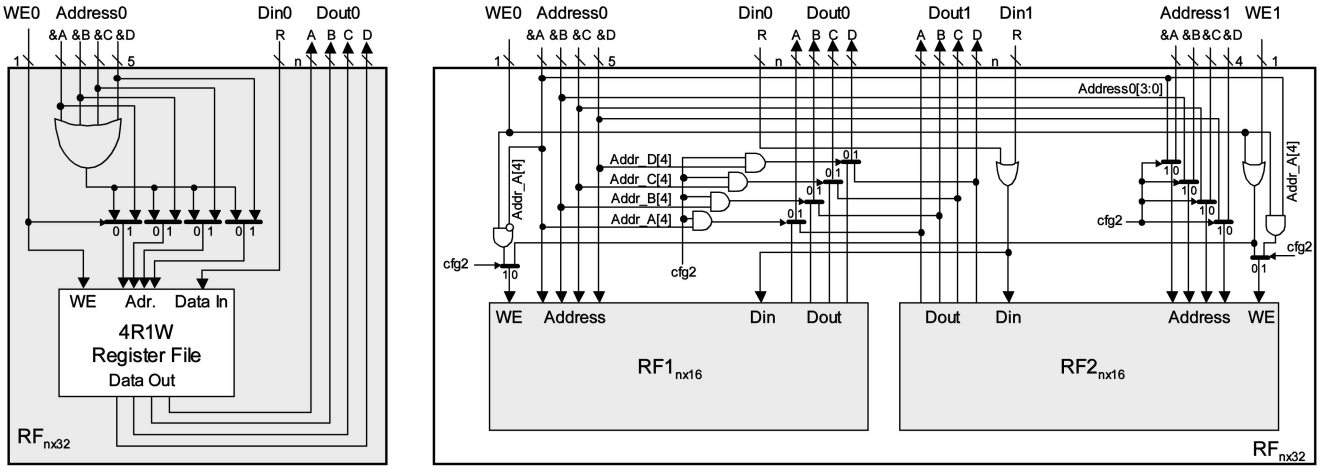


Fig. 5. Block diagram for the reconfigurable data path—extension of parallelism and the field size. Depending on the setting of the interconnections, various data paths can be reconfigured, for example, α copies of the MALU over $\mathrm{GF}(2^n)$ with digit size d or one MALU over $\mathrm{GF}(2^{\alpha(n+1)-1})$ with digit size d.



(a) (b)

Fig. 6. Hardware architecture of the RFs. (a) Four-ported RF supporting four reads and one write (4R1W) to support simultaneous reading of four operands. A 32-entry n-bit RF $(\mathrm{RF}_{n\times 32})$. (b) A pair of $\mathrm{RF}_{n\times 16}$ can be configured as $\mathrm{RF}_{n\times 32}$ by setting the signal $cfg2 = 1$.

especially in read operations, a multiport RF is implemented, as illustrated in Fig. 6a.

The multiport RF supports four simultaneous read operations at four different addresses per cycle (4R). This allows one to read all of the necessary operands for the operation form $A(x)(B(x) + D(x)) + C(x)$ in a single cycle. This way, the number of the read-access cycles can be reduced by 3/4 or 75 percent. The read cycle is reduced to only three cycles for three parallel executions. The write operation can be done unless those read operations are executed (1W).

Note that one RF can be shared with multiple MALU cores. Only when supporting a wider field size should multiple RFs be allocated in the cryptoprocessor. In addition, the required number of entries in the RF differs from ECC to HECC in that ECC needs 16 registers, whereas HECC uses 32 registers, as mentioned previously. This difference can be a problem when the cryptoprocessor needs to support both cryptosystems. A simple solution is to prepare 32 entries in each RF (denoted as $\mathrm{RF}_{n\times 32}$ in Fig. 6a). Another solution is to make one 32-entry RF from two 16-entry RFs, as shown in Fig. 6b. The figure illustrates how $\mathrm{RF}_{n\times 32}$ can be configured with $\mathrm{RF1}_{n\times 16}$ and $\mathrm{RF2}_{n\times 16}$. As will be investigated in detail in Section 5, both solutions have drawbacks and advantages.

3.5 The MALU Instruction

We now design a new instruction called `MALU()`. It is worth mentioning again that this is the only instruction that operates on the data path:

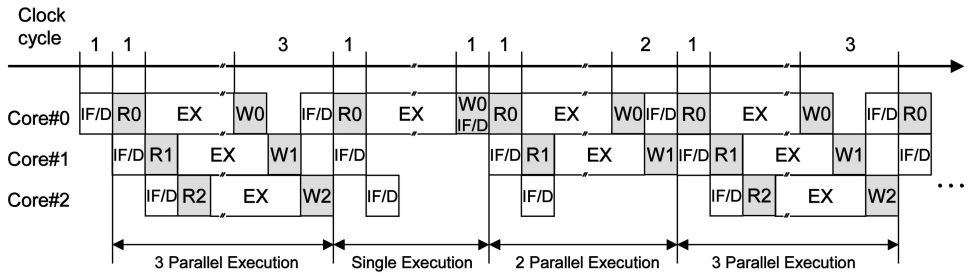


Fig. 7. Example of parallel issue of instructions for three MALUs (IF/D: instruction fetch/decode, EX: execution of MALU, and R/W: read/write from/to the RF). The consecutive write cycles depend on the number of instructions issued in parallel. The execution cycle is determined by the MALU configuration.

$$\begin{aligned}&\texttt{MALU}(\texttt{\&R}, \texttt{\&A}, \texttt{\&B}, \texttt{\&C}, \texttt{\&D}) : \\ &R(x) = A(x)(B(x) + D(x)) + C(x) \pmod{P(x)}.\end{aligned} \tag{6}$$

Here, &A, &B, &C, &D denote the addresses for four inputs of the instruction and &R denotes the address where the result is stored. As illustrated in Fig. 7, the whole procedure to execute MALU() starts from an instruction fetch and decode (IF/D). Then, variables for $A(x)$, $B(x)$, $C(x)$, and $D(x)$ are loaded via the RF (R) for the succeeding execution stage. The result is stored to the RF (W) in the last step. When performing parallel processing, the write operations from every MALU core should be sequential in order to escape memory-write conflicts. More precisely, in order to keep data integrity between the RFs, only one data item can be written to the RFs within a cycle: This is a consequence of the way in which the 4R1W RF is embedded in our cryptoprocessor. From another viewpoint, the operands for different MALU cores can also be read sequentially, which means that one RF can be shared with multiple MALU cores.

The number of instructions that can be issued in parallel decides consecutive write cycles. In total, an l-way parallel execution takes $l+1$ cycles, in addition to the execution cycles that depend on the MALU configuration.

3.6 Dynamic Scheduling for Multicore Architecture

ILP is exploited for all MALU() instructions as long as two or more instructions are buffered in the IQB. Here, we introduce our strategy to find ILP. The instruction has four source operands and outputs the result to the RF; that is, MALU(&R, &A, &B, &C, &D) deals with five types of addresses for the operation $A(x)(B(x) + D(x)) + C(x) \pmod{P(x)}$. They are expressed as

$$\texttt{MALU} \ : \ \texttt{\&R} = \texttt{\&A}, \texttt{\&B}, \texttt{\&C}, \texttt{\&D}. \tag{7}$$

The MALU instruction also refers to the $P(x)$ that is stored in the RF. To include out-of-order executions, two types of dependencies need to be checked between two instructions: $\texttt{MALU}^i$ and $\texttt{MALU}^j$ (i and j are labels indicating the order of instruction in the IQB). For all i and j that satisfy $0 \le i < j < ILP_D$, where ILP_D is the size of the instruction window to exploit ILP, one can determine the number of instructions to be issued in parallel by checking the following dependencies:

Read-After-Write (RAW) dependency check for in-order execution ($\texttt{\&R}^i = \texttt{\&A}^j$ or $\texttt{\&R}^i = \texttt{\&B}^j$ or $\texttt{\&R}^i = \texttt{\&C}^j$ or $\texttt{\&R}^i = \texttt{\&D}^j$): If the result of the instruction $\texttt{MALU}^i$, $\texttt{R}^i$ is used as the input of the instruction $\texttt{MALU}^j$, then $\texttt{MALU}^j$ cannot be issued until or before $\texttt{MALU}^i$ completes its operation. In other words, if the condition in parentheses above is false, then $\texttt{MALU}^j$ can be issued with $\texttt{MALU}^i$.

However, when the parallel issue of $\texttt{MALU}^i$ and $\texttt{MALU}^j$ includes an out-of-order execution, the next condition has to be verified as well.

RAW dependency check for out-of-order execution ($\texttt{\&R}^j = \texttt{\&A}^i$ or $\texttt{\&R}^j = \texttt{\&B}^i$ or $\texttt{\&R}^j = \texttt{\&C}^i$ or $\texttt{\&R}^j = \texttt{\&D}^i$): As a result of checking the conditions for an in-order execution, it is possible that the instruction $\texttt{MALU}^j$ can be issued, whereas some preceding instructions cannot. In this case, we need to check if the result of the instruction $\texttt{MALU}^j$, $\texttt{R}^j$ is used for the input of the preceding instructions that cannot be issued. The corresponding condition is described above in parentheses.

The proposed architecture needs no check for Write-After-Read and Write-After-Write dependencies, in contrast to a general superscalar machine. Indeed, the instruction MALU() is a fixed-length multicycle instruction and, hence, we can skip those dependencies in checking the sequence of point/divisor operations.

As the zeroth instruction $\texttt{MALU}^0$ is issued unconditionally, the number of conditions to check ILP becomes $4(ILP_D - 1)^2$. This fact indicates that the hardware complexity for ILP grows quadratically in a large ILP_D, but, in exchange, further parallelism can be exploited. The choice of the ILP_D is discussed in Section 4.4.

4 Performance Evaluation

4.1 Design Platform

The proposed design is constructed on the GEZEL hardware/software codesign platform with the ARM Instruction Set Simulator (ISS) [27]. The cryptoprocessor is described in an FSM with Data Path (FSMD) manner. The platform provides cycle-accurate simulations for various hardware/software system configurations. As mentioned in Section 3, the cryptoprocessor is attached to the memory-mapped interface of the ARM. Thus, various types of system configurations are examined to verify the functionality and estimate the system-level performance quickly. The GEZEL codes are automatically translated into very high density logic (VHDL) codes that can be used to prototype the proposed cryptoprocessor on an FPGA.

4.2 Instruction Set for the Cryptoprocessor

Table 1 shows some of the primary instructions for the cryptoprocessor. For a 32-bit CPU such as the ARM, storing

TABLE 1
Primary Instructions for the Proposed Cryptoprocessor and the Computation Costs in the Case of the Operation Form $A(B+D)+C$

INSTRUCTION	OPERATION	COMPUTATION COST
STORE(data,num,@dst)	Storing data in the RF#num	-
LOAD(num,@src)	Loading data from the RF#num	-
CFG(data,num)	Setting cfg$\star$ for the MALU core#num	-
MALU(&R,&A,&B,&C,&D)	MALU operation: $R = A(B+D)+C$	$\lceil n/d \rceil$ cycles
ECC_PA()	Point Addition for ECC	15 MALU() instructions
ECC_PD()	Point Doubling for ECC	10 MALU() instructions
ECC_M_PA()	Point Addition for ECC_M	6 MALU() instructions
ECC_M_PD()	Point Doubling for ECC_M	7 MALU() instructions
HECC_PA()	Divisor Addition for HECC	61 MALU() instructions
HECC_PD()	Divisor Doubling for HECC	39 MALU() instructions
INV()	Modular Inverse (Itoh-Tsujii algorithm [19])	$\{\lfloor log_2(n-1) \rfloor + w(n-1) - 1 + (n-1)\}$ MALU() instructions

Here, $w(k)$ denotes the Hamming weight of the positive integer k.

data to the address dst requires four STORE() instructions for HECC over $\mathrm{GF}(2^{97})$. After all operands are set at the corresponding addresses of the RF, the main CPU sends the instructions MALU(). By using the μ-code RAM in the cryptoprocessor, it is possible to define an instruction that consists of a series of MALU() instructions. In this paper, all necessary point/divisor operations are preprogrammed in the μ-code RAM and the main CPU uses these instructions (for example, ECC_PA() and ECC_PD()) for scalar multiplication.

4.3 Configuration of the MALU Cores

The system performance is heavily dependent on the number of MALU cores and the data path configuration in each MALU core. They also determine the supported range of field sizes. The field sizes of interest in this paper are 163 and 193 bits for ECC because they offer a security level greater than or equal to a 1,024-bit RSA [16]. The corresponding field sizes for HECC are 83 and 97 bits. Therefore, it is reasonable to use the MALU cores with a data path of length $n = 97$. As for the digit size, $d = 12$ is chosen as an example case. This data path is denoted as $\mathrm{MALU}_{97\times12}$ and one modular multiplication over $\mathrm{GF}(2^{97})$ can be computed in $\lceil n/d \rceil$ or nine cycles.

In [28], the EC Digital Signature Algorithm (ECDSA) standard is designed and the recommended curve parameters and irreducible polynomials are listed for several field sizes of up to 571 bits. Therefore, we also investigate the performance of our cryptoprocessor for a 571-bit ECC.

Suppose that six cores with the data path $\mathrm{MALU}_{97\times12}$ are allocated in the cryptoprocessor. Various data path configurations can be supported. Table 2 summarizes selected hardware configurations and the number of clock cycles for one MALU instruction ($\lceil n/d \rceil$) over different field sizes. The throughput of the MALU instruction can be estimated with

$$\frac{l \cdot n}{\lceil n/d \rceil + l + 1} \quad [\text{bits/clock}] \quad (l = 1, 2, 3, 4), \tag{8}$$

where l is the degree of parallelism (that is, the execution of l-way parallelism). Although the number of data paths used in parallel varies from 1 to 6, depending on the field lengths, we exploit at most four-way parallelism in this paper in order to reduce the logic complexity in the IBC. Fig. 8 illustrates the minimal and maximal throughput of the MALU operation as a function of the field size. As can be seen in the figure, this configuration can offer a high throughput for a field size of around 97, 195, and 293 bits because we use $\mathrm{MALU}_{97\times12}$ as a building block of the data path. The maximum throughput can be obtained only if all of the data paths are used in parallel. When no parallelism can be found in the MALU instructions (that is, in the case of single execution of the MALU instruction), our cryptoprocessor performs at the minimum throughput. In other words, by exploiting parallelism for $n \leq 293$ in the MALU instructions, the throughput can be improved, depending

TABLE 2
Possible Configurations of the Data Path with Six Cores of $n = 97$ and $d = 12$ ($\mathrm{MALU}_{97\times12}$) and the Execution Cycles for One MALU Instruction over a Binary Field

Configuration	Field Size	Execution Cycles
6· $\mathrm{MALU}_{97\times12}$	$\mathrm{GF}(2^{97})$	9 cycles
3· $\mathrm{MALU}_{195\times12}$	$\mathrm{GF}(2^{193})$	17 cycles
2· $\mathrm{MALU}_{293\times12}$	$\mathrm{GF}(2^{283})$	24 cycles
1· $\mathrm{MALU}_{587\times12}$	$\mathrm{GF}(2^{571})$	48 cycles

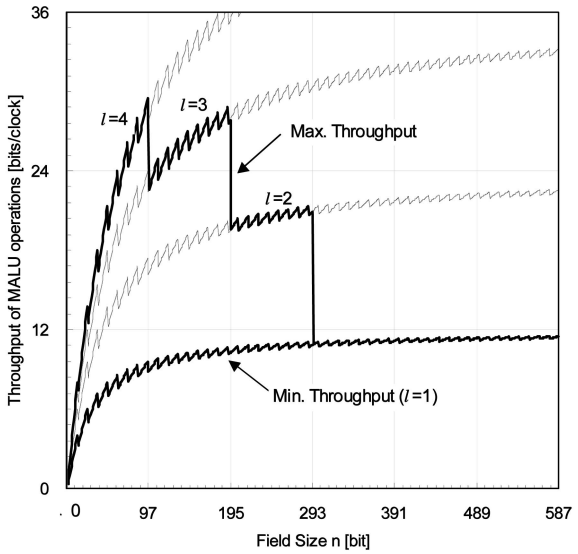


Fig. 8. Throughput for different configurations of the data path with six $\mathrm{MALU}_{97\times12}$.

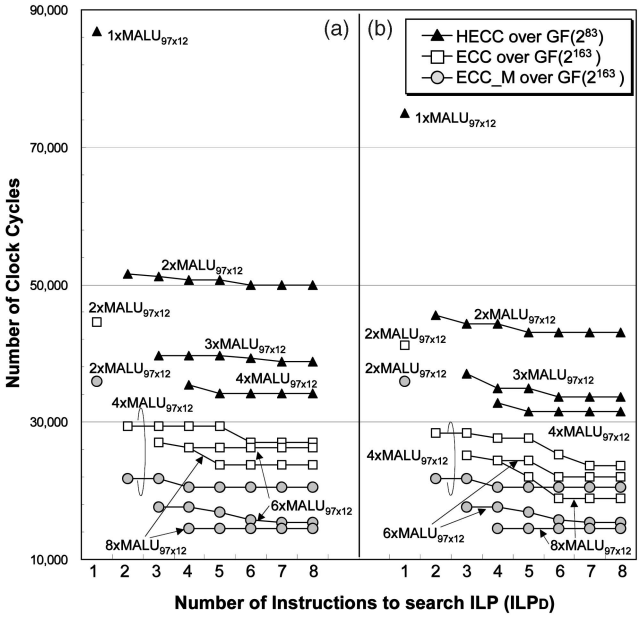


Fig. 9. The number of clock cycles for scalar multiplication of ECC163, ECC_M163, and HECC83 for different ILP_Ds when allocating one to eight $\mathrm{MALU}_{97\times12}$. (a) Operation form is $AB+C$. (b) Operation form is $A(B+D)+C$.

on the degree of parallelism. However, the throughput is almost constant for $n > 293$.

4.4 Degree of Parallelism in ECC and HECC

As the performance of the superscalar architecture is dependent on the degree of parallelism, it is also important to determine ILP_D, which is an appropriate number of instructions to search for ILP, as well as the number of MALU cores. Fig. 9 shows the number of clock cycles for a scalar multiplication when setting ILP_D from 1 to 8.

Up to eight copies of the MALU cores with $\mathrm{MALU}_{97\times12}$ are instantiated in the cryptoprocessor to evaluate the performance improvement by the superscaling feature for ECC163, ECC_M163, and HECC83. Here, we assume that enough RFs are allocated in the cryptoprocessor, that is, $\mathrm{RF}_{97\times32}$ is assigned for each MALU core with $\mathrm{MALU}_{97\times12}$ so that the cryptosystem has no limitations on the supported field sizes and the type of cryptosystem. As a result of the GEZEL system-level simulation, we observe that, for both operation forms, the overall performance improves as the number of MALU_{97} increases. Also, a large ILP_D helps exploit more parallelism and leads to higher performance. We can also see the effectiveness of an operation if the form is $A(B+D)+C$. The results of using this operation with $ILP_D = 6$ are also summarized in Table 3.

In order to investigate the performance bottleneck of ECC and HECC, the number of clock cycles for a scalar multiplication is split by the degree of parallelism. Fig. 10 shows the results for ECC163, ECC_M163, and HECC83 by changing the number of MALU cores and the type of the operation. Note that the same performance is obtained for ECC_M163, regardless of the type of the operation.

We consider the utilization of the MALU cores in order to know if the data paths are effectively used in parallel. If a parallel execution utilizes all data paths, then the utilization is defined as 100 percent during execution of the parallel operations. On the other hand, the utilization is 50 percent when half of the data paths are used: For example, a two-way computation is executed on a configuration with four data paths. Note that memory accesses are included as a part of a parallel execution. The figures in percentage marked on the bars indicate the utilization of the MALU cores defined as

$$\frac{\sum_{i=1}^{l_{max}} i \cdot R_i}{l_{max} \sum_{i=1}^{l_{max}} R_i} \quad (l_{max} = 1, 2, 3, 4), \tag{9}$$

where l_{max} is the maximum degree of parallelism under the given hardware configuration and R_i is the number of clock cycles required for i-way computations. As can be seen in Fig. 10, the proposed superscalar feature can reduce the overall number of clock cycles. However, utilization of the MALU cores decreases as the value of l_{max} increases. This fact indicates that area and performance trade-offs are getting worse for a larger l_{max} and it can be exploited by inherent data dependencies in ECC and HECC. From this observation, we decide to employ $l_{max} \leq 3$ for ECC and $l_{max} \leq 4$ for HECC to maintain high utilization of the MALU cores.

5 Implementation

The cryptoprocessor discussed in Section 3 has been synthesized with a 0.13-μm CMOS technology by using the Synopsys Design Vision. From the performance evaluation discussed in Section 4, we allocate up to eight instantiations of the MALU cores with $\mathrm{MALU}_{97\times12}$ with an

TABLE 3
The Number of Clock Cycles for a Scalar Multiplication with $d = 12$, $ILP_D = 6$, and the Operation Form $A(B+D)+C$

Cryptoprocessor Configuration	HECC 83	ECC 163	ECC_M 163	HECC 97	ECC 193	ECC_M 193	HECC 139	ECC 283	ECC_M 283	HECC 283	ECC 571	ECC_M 571
1·$\mathrm{MALU}_{97\times12}$	74,970 (1.00)	-	-	107,086 (1.00)	-	-	-	-	-	-	-	-
2·$\mathrm{MALU}_{97\times12}$	43,036 (1.74)	41,136 (1.00)	35,936 (1.00)	60,413 (1.77)	57,840 (1.00)	50,528 (1.00)	195,303 (1.00)	-	-	-	-	-
4·$\mathrm{MALU}_{97\times12}$	31,548 (2.38)	25,225 (1.63)	20,530 (1.75)	43,630 (2.45)	35,196 (1.64)	28,600 (1.77)	108,321 (1.80)	106,134 (1.00)	85,624 (1.00)	733,460 (1.00)	-	-
6·$\mathrm{MALU}_{97\times12}$	31,548 (2.38)	22,030 (1.87)	15,730 (2.28)	43,630 (2.45)	30,656 (1.89)	21,779 (2.32)	82,827 (2.36)	58,980 (1.80)	47,815 (1.79)	397,663 (1.84)	450,319 (1.00)	393,394 (1.00)
8·$\mathrm{MALU}_{97\times12}$	31,548 (2.38)	18,850 (2.18)	14,530 (2.47)	43,630 (2.45)	26,138 (2.21)	20,074 (2.52)	82,827 (2.36)	58,980 (1.80)	47,815 (1.79)	397,663 (1.84)	450,319 (1.00)	393,394 (1.00)

The numbers in parentheses are the speedup ratio compared to the single-scalar configuration.

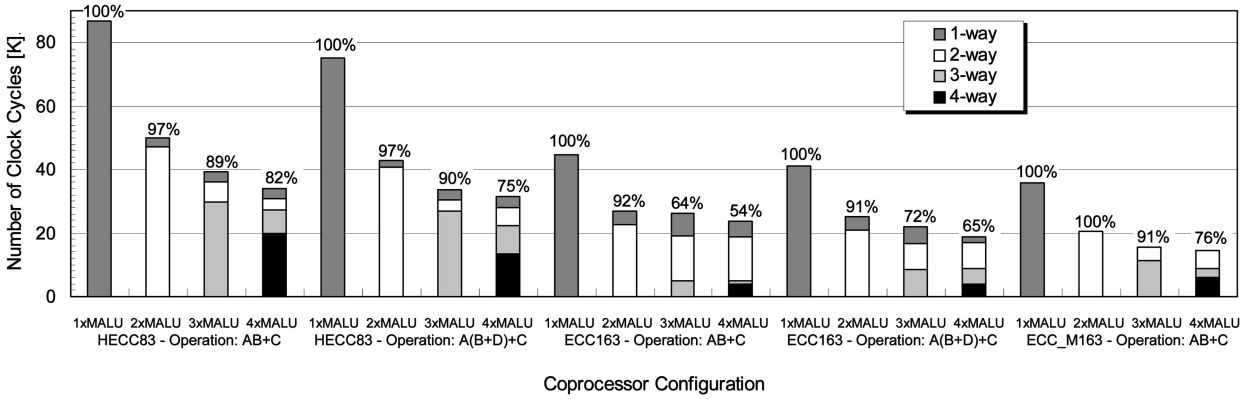


Fig. 10. The profile graphs of the number of clock cycles in ECC/HECC scalar multiplication for different hardware settings of the cryptoprocessor $(d = 12)$.

TABLE 4
Configurations of the Data Path with Six Cores, Each of which Has $\mathrm{MALU}_{97\times12}$

Configuration	HECC over GF(2^n)				ECC over GF(2^n)		
	$n \le 97$	$97 < n \le 195$	$195 < n \le 293$	$293 < n \le 587$	$n \le 195$	$195 < n \le 293$	$293 < n \le 587$
$6\times\mathrm{MALU}_{97\times12}$ $+6\times\mathrm{RF}_{97\times32}$	Supported (4-way)	Supported (3-way)	Supported (2-way)	Supported (1-way)	Supported (3-way)	Supported (2-way)	Supported (1-way)
$6\times\mathrm{MALU}_{97\times12}$ $+6\times\mathrm{RF}_{97\times16}$	Supported (3-way)	Supported (2-way)	Supported (1-way)	Not Supported	Supported (3-way)	Supported (2-way)	Supported (1-way)

operation $A(B+D)+C$. For the size of the instruction window, $ILP_D = 6$ is selected. The trade-offs between cost and performance are discussed for ECC, ECC_M, and HECC with different RF configurations. The synthesis results show that all designs meet with the timing constraint of 292 MHz.

A pair of $\mathrm{MALU}_{97\times12}$ can be reconfigured as one $\mathrm{MALU}_{195\times12}$ by changing the interconnection between the MALU cores. This way, both HECC97 and ECC193 can be supported by allocating $2\cdot\mathrm{MALU}_{97\times12}$ in the cryptoprocessor. As for the RF, we need to prepare either a pair of $\mathrm{RF}_{97\times32}$ or a pair of $\mathrm{RF}_{97\times16}$ to support the field lengths.

As explained previously, HECC requires $\mathrm{RF}_{n\times32}$, whereas ECC can be computed with $\mathrm{RF}_{n\times16}$, where n is the field size of ECC and HECC. Therefore, depending on the configuration of the RF, the supported field lengths and the degree of parallelism are differently determined, as shown in Table 4. In other words, the configuration using $6\cdot\mathrm{RF}_{97\times32}$ offers enough registers for both ECC and HECC and, hence, a better degree of parallelism can be expected. Moreover, we can apply the NAF_4 (NAF with a width-4 window) for ECC by utilizing the redundant 16 entries in the RF. In contrast, $6\cdot\mathrm{RF}_{97\times16}$ can be considered as an ECC-centric configuration because it can save redundant registers when ECC is performed. The drawback of this configuration is that the degree of parallelism in HECC is restricted by the number of $\mathrm{RF}_{97\times16}$, that is, the cryptoprocessor can exploit at most a three-way parallelism for HECC in this case.

Figs. 11 and 12 show the average time for an ECC and an HECC scalar multiplication for the configuration of $\alpha\cdot\mathrm{MALU}_{97\times12}+\alpha\cdot\mathrm{RF}_{97\times32}$, where $\alpha = 1,2,3,4,5,6,8$ (CONFIG-I). Figs. 13 and 14 show the results for the configuration of $\alpha\cdot\mathrm{MALU}_{97\times12}+\alpha\cdot\mathrm{RF}_{97\times16}$, where $\alpha = 2,3,4,5,6,8$ (CONFIG-II). Here, we use a 1-Kbyte μ-code RAM in order to support ECC and HECC.

As can be seen in the figures, the cost of supporting ECC571 with CONFIG-I is 393 Kgates, which is more expensive than that with CONFIG-II, whose gate size is 244 Kgates. However, CONFIG-II shows a slightly lower performance for HECC. This performance degradation is more apparent for a larger field size. For example, in the case of $\alpha = 6$, the performance of HECC139 decreases from 284 to 670 μs when changing the configuration.

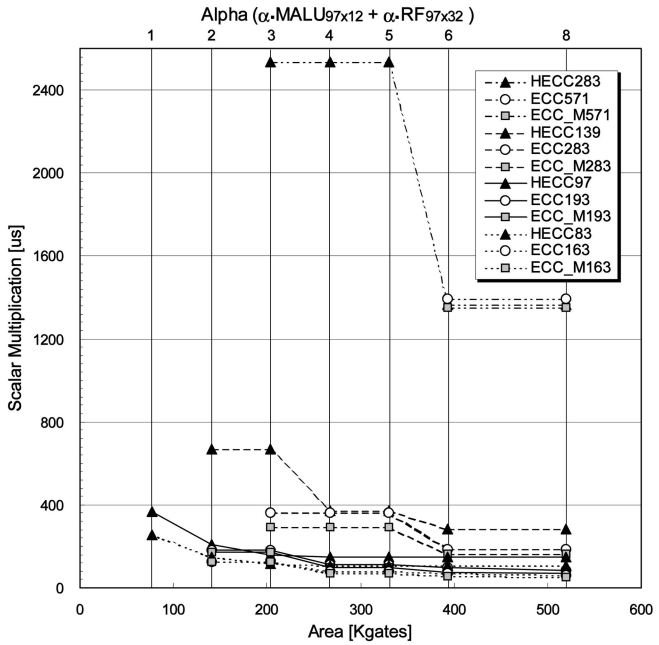


Fig. 11. The number of clock cycles required for different configurations of the MALU and the RF, that is, $\alpha\cdot\mathrm{MALU}_{97\times12}+\alpha\cdot\mathrm{RF}_{97\times32}$.

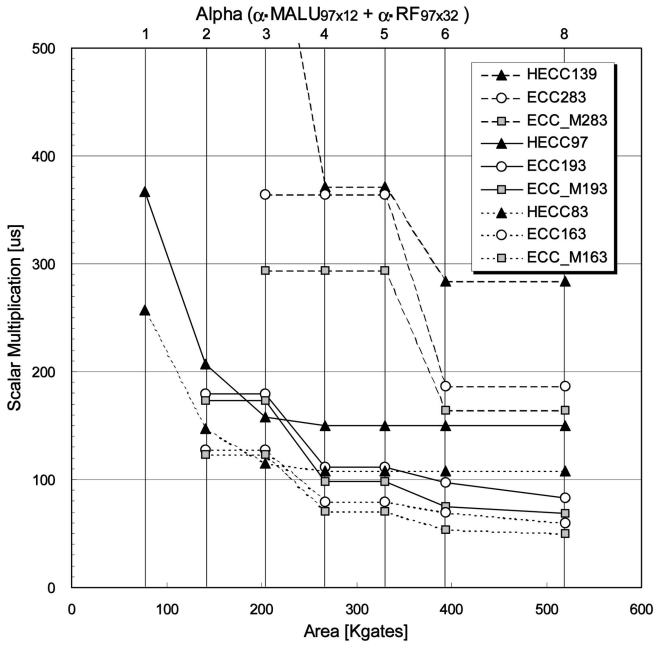


Fig. 12. Magnified image of Fig. 11.

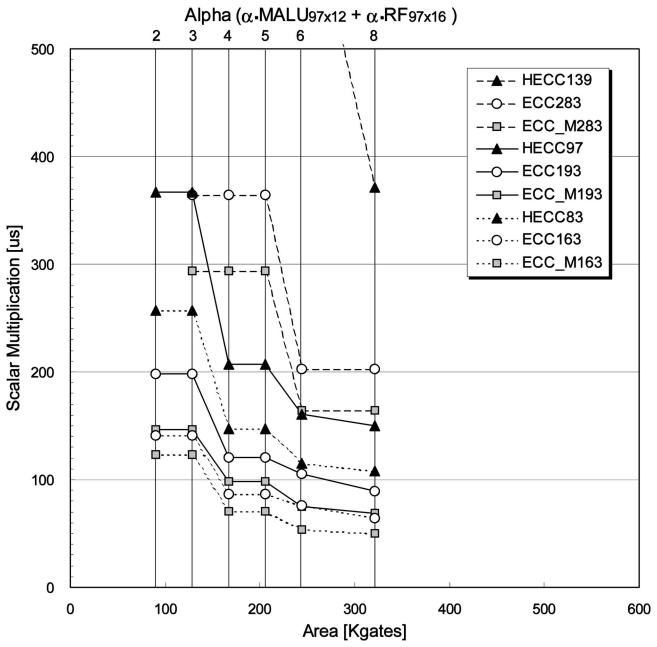


Fig. 14. Magnified image of Fig. 13.

The computation cost for modular inversion is summarized in Table 5 for CONFIG-I. The ratio of the inversion cost to the computation cost for the scalar multiplication varies from 7 percent to 18 percent in ECC and ECC_M. This is due to the fact that we use the MALU instructions for modular squarings in the Itoh-Tsujii algorithm. However, considering the flexibility of the proposed hardware architecture, the inversion cost can be regarded as low enough. In the case of HECC, the cost for modular inversion is negligible.

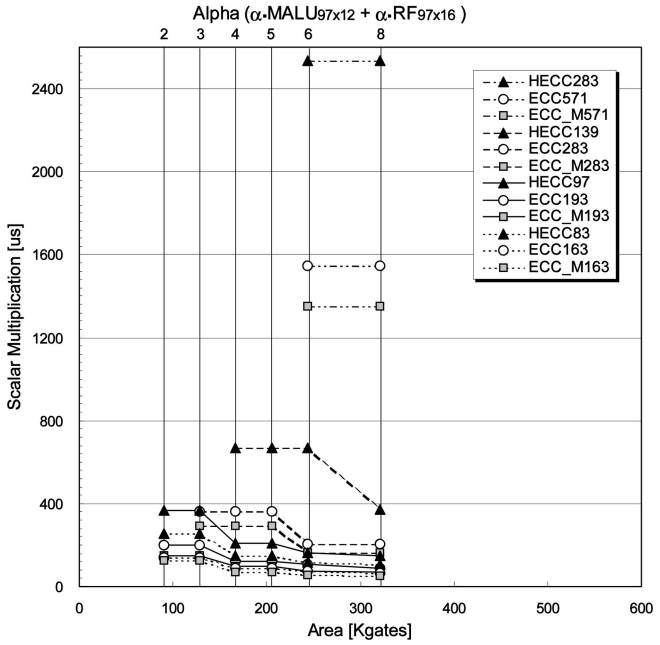


Fig. 13. The number of clock cycles required for different configurations of the MALU and the RF, that is, $\alpha \cdot \mathrm{MALU}_{97\times12} + \alpha \cdot \mathrm{RF}_{97\times16}$.

For achieving faster performance, other configurations are also considered by fixing the field length and the irreducible polynomial and supporting either ECC or HECC only. In these configurations, one fixed-size RF can be shared with the MALU cores. For instance, we can consider the configuration of $4 \cdot \mathrm{MALU}_{83\times12} + \mathrm{RF}_{83\times32}$ for HECC83. In addition, we use ROM for storing the μ-code program. These configurations offer a higher performance with lower cost compared to CONFIG-I and CONFIG-II at the cost of reduced flexibility and programmability. The results of this configuration are also discussed in Section 6.

6 Results

Table 6 summarizes the performance of ECC and HECC scalar multiplication for selected field sizes and different hardware configurations. Our proposed cryptoprocessor can provide various choices of area and performance. The observed performance maintains high overall for all supported field sizes.

When implementing the cryptoprocessor based on CONFIG-I, the highest flexibility and performance can be obtained for both ECC and HECC, as shown in Table 6, with a gate size of 393 Kgates. On the other hand, for CONFIG-II, the gate size becomes 244 Kgates, with some performance penalty for ECC and HECC. In both configurations, the performance of HECC is lower than the

TABLE 5
Computation Cost of Modular Inversion with CONFIG-I
(at the Clock Frequency of 292 MHz)

Field size	83	97	139	163	193	283	571
Time[μs]	2.8	3.9	7.1	9.4	13.2	26.1	99.8
Ratio in ECC	-	-	-	13%	15%	14%	7%
Ratio in ECC_M	-	-	-	17%	18%	16%	7%
Ratio in HECC	3%	3%	2%	-	-	2%	-

TABLE 6
Performance Comparison of HECC/ECC Hardware Implementations over a Binary Field

Ref. Design	Technology / FPGA	f_{max} [MHz]	Area [Slices/Gates]	Galois Field	Irreducible Polynomial	Performance† [μsec]	Comments
ECC							
This work	0.13-μm CMOS	292	393 Kgates	$GF(2^{163})$ $GF(2^{193})$ $GF(2^{283})$ $GF(2^{571})$	Arbitrary	70 / 54 90 / 75 187 / 164 1,394 / 1,349	CONFIG-I ($\alpha = 6$) $6{\cdot}\mathrm{MALU}_{97\times12}$ + $6{\cdot}\mathrm{RF}_{97\times32}$.
			244 Kgates	$GF(2^{163})$ $GF(2^{193})$ $GF(2^{283})$ $GF(2^{571})$	Arbitrary	76 / 54 105 / 75 202 / 164 1,545 / 1,349	CONFIG-II ($\alpha = 6$) $6{\cdot}\mathrm{MALU}_{97\times12}$ + $6{\cdot}\mathrm{RF}_{97\times16}$.
		500	115 Kgates	$GF(2^{163})$	$x^{163}+x^7+x^6+x^3+1$	38 / 29	$4{\cdot}\mathrm{MALU}_{163\times12}$ + $\mathrm{RF}_{163\times16}$.
			135 Kgates	$GF(2^{193})$	$x^{193}+x^{15}+1$	52 / 40	$4{\cdot}\mathrm{MALU}_{193\times12}$ + $\mathrm{RF}_{193\times16}$.
[29]	0.13-μm CMOS	416.7	90 Kgates♯ 113 Kgates♯	$GF(2^{163})$	Arbitrary	209 / - - / 30	Multiplier, divider and squarer. Two multipliers, divider and squarer.
[15]	0.13-μm CMOS	510.2	117.5 Kgates	$GF(2^{163})$	Arbitrary	190 / -	Support of GF(p). 64-bit × 64-bit multiplier.
[30]	Virtex-E (xcv2000E-7)	66.4	10,034 Slices†	$GF(2^{163})$ $GF(2^{193})$ $GF(2^{233})$	Arbitrary	- / 300 - / 420 - / 510	MSD first multiplier, divider and squarer for field sizes up to 255.
				$GF(2^{163})$ $GF(2^{193})$ $GF(2^{233})$	Fixed	- / 140 - / 190 - / 230	
[31]	Virtex-II (xc2v6000)	54	-	$GF(2^{162})$	Fixed	- / 60	Optimal normal basis multiplier for a fixed field size.
		35	-	$GF(2^{270})$	Fixed	- / 170	
[32]	Virtex-E (xcv2000E)	66	5,009 Slices†	$GF(2^{163})$	$x^{163}+x^7+x^6+x^3+1$	233 / - 75 / -	Multiplier, divider and squarer; Generic curve. Multiplier, divider and squarer; Koblitz curve.
[33]	Virtex-E (xcv400E)	76.7	3,002 Slices + 10 BRAMs	$GF(2^{167})$	$x^{167}+x^6+1$	- / 210	167-bit×16-bit multiplier and 167-bit×167-bit squarer.
[24]	Virtex-E (xcv3200E)	9.99	19,626 Slices + 26 BRAMs	$GF(2^{191})$	$x^{191}+x^9+1$	- / 59.26	Two binary Karatsuba multipliers, squarer and inverter.
HECC							
This work	0.13-μm CMOS	292	393 Kgates	$GF(2^{83})$ $GF(2^{97})$ $GF(2^{139})$ $GF(2^{283})$	Arbitrary	108 150 284 1,364	CONFIG-I (α = 6) $6{\cdot}\mathrm{MALU}_{97\times12}$ + $6{\cdot}\mathrm{RF}_{97\times32}$.
			244 Kgates	$GF(2^{83})$ $GF(2^{97})$ $GF(2^{139})$ $GF(2^{283})$	Arbitrary	115 160 670 2,533	CONFIG-II (α = 6) $6{\cdot}\mathrm{MALU}_{97\times12}$ + $6{\cdot}\mathrm{RF}_{97\times16}$.
		500	65 Kgates	$GF(2^{83})$	$x^{83}+x^7+x^4+x^2+1$	63	$4{\cdot}\mathrm{MALU}_{83\times12}$ + $\mathrm{RF}_{83\times32}$.
[13]	Virtex-II Pro (xc2vp20-7)	57.0 60.7	4,039 Slices 7,737 Slices	$GF(2^{81})$	Fixed	787 387	Two multipliers and inverter. Three multipliers and two inverters.

†: *The ECC performance is denoted as ECC/ECC_M.* ‡: *Estimate based on their result of the number of LUTs.*
♯: *Estimate based on their results of 0.47* mm^2 *and 0.59* mm^2.

performance of ECC and ECC_M overall when comparing the field sizes that have the same security strength for ECC and HECC.

By fixing the field size and the irreducible polynomial, the gate size of the cryptoprocessor for ECC163, ECC193, and HECC83 requires only 115, 135, and 65 Kgates and performs a scalar multiplication in 29, 40, and 63 μsec, respectively, with the configuration of $4 \cdot \mathrm{MALU}_{163\times12} + \mathrm{RF}_{163\times16}$, $4 \cdot \mathrm{MALU}_{193\times12} + \mathrm{RF}_{193\times16}$, and $4 \cdot \mathrm{MALU}_{83\times12} + \mathrm{RF}_{83\times32}$.

Comparing with previous work, our HECC implementation results are faster than the implementation reported by Wollinger [13], which was one of the fastest HECC implementations. Furthermore, our implementation can support both ECC and HECC. Our ECC implementation results also show better performance than other previous work, except an ECC_M implementation of Sozzani et al. [29]. This is because their ASIC design uses the 163-bit fixed field size and a hardwired controller, which offers less scalability and flexibility than our reconfigurable design. In fact, our design with a fixed irreducible polynomial shows better performance than their result while supporting both ECC and ECC_M.

7 Conclusions

This paper presented a multicore cryptoprocessor for ECC and HECC to support a wide range of field sizes and to accelerate the scalar multiplication of ECC and HECC of genus 2 over $\mathrm{GF}(2^n)$ by exploiting ILP on the fly. The superscaling feature is facilitated by defining a single instruction that is flexibly defined as $AB + C$ or $A(B + D) + C$ and can be used for all field operations such as modular multiplications, modular additions, and point/divisor operations. We conclude that the operation $A(B + D) + C$ is effective to decrease the number of clock cycles for scalar multiplication.

The fully programmable cryptoprocessor can handle various curve parameters and an arbitrary irreducible polynomial. In addition, a wide range of the field size of modular operations can be supported by reconfiguring the data path in the MALU cores. Thus, the trade-off between performance and security can be obtained simply by

changing the program and reconfiguring the MALUs. The synthesis results show that scalar multiplication can be performed at 292 MHz, with a gate size of 244 Kgates, while supporting ECC over $GF(2^{571})$ and HECC over $GF(2^{283})$. In our design, ECC offers better cost and performance trade-offs than HECC.

The compact and fastest configuration of our design shows that scalar multiplication of ECC over $GF(2^{163})$ and HECC over $GF(2^{83})$ can be performed in 29 and 63 μs, respectively. This speedup is achieved by exploiting parallelism in ECC and HECC.

ACKNOWLEDGMENTS

Kazuo Sakiyama and Lejla Batina are funded by a research grant from the Katholieke Universiteit (KU) Leuven and Fund for Scientific Research-Flanders (FWO) projects G.0450.04 and G.0475.05. This work was supported in part by the Interuniversity Attraction Pole (IAP) program P6/26 Belgian Fundamental Research on Cryptology and Information Security (BCRYPT) of the Belgian State (Belgian Science Policy), by the European Union Information Society Technologies (EU IST) FP6 projects Security for embedded systems on chip (SESOC) and European Network of Excellence for Cryptology (ECRYPT), by KU Leuven, and by the Interdisciplinary Institute for Broadband Technology Quality of Experience (IBBT-QoE) project of the IBBT.

REFERENCES

[1] W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory,* vol. 22, pp. 644-654, 1976.

[2] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM,* vol. 21, no. 2, pp. 120-126, 1978.

[3] N. Koblitz, "Elliptic Curve Cryptosystem," *Math. Computation,* vol. 48, pp. 203-209, 1987.

[4] V. Miller, "Uses of Elliptic Curves in Cryptography," *Advances in Cryptology: Proc. Int'l Cryptology Conf. (CRYPTO '85),* H.C. Williams, ed., pp. 417-426, 1985.

[5] N. Thériault, "Index Calculus Attack for Hyperelliptic Curves of Small Genus," *Advances in Cryptology—Proc. Ninth Int'l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT '03),* C.S. Laih, ed., pp. 75-92, 2003.

[6] A. Hodjat and I. Verbauwhede, "Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Trans. Computers,* vol. 55, no. 4, pp. 366-372, Apr. 2006.

[7] N. Koblitz, "CM-Curves with Good Cryptographic Properties," *Advances in Cryptology: Proc. Int'l Cryptology Conf. (CRYPTO '91),* J. Feigenbaum, ed., pp. 279-287, 1991.

[8] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Math. Computation,* vol. 48, no. 177, pp. 243-264, 1987.

[9] N.P. Smart, "The Hessian Form of an Elliptic Curve," *Proc. Third Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01),* Ç.K. Koç, D. Naccache, and C. Paar, eds., pp. 121-128, May 2001.

[10] M. Joye and S.-M. Yen, "The Montgomery Powering Ladder," *Proc. Fourth Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '02),* B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, eds., pp. 291-302, 2002.

[11] T. Izu and T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks," *Proc. Fifth Int'l Workshop Practice and Theory in Public Key Cryptosystems (PKC '02),* D. Naccache and P. Paillier, eds., pp. 280-296, 2002.

[12] P.K. Mishra and P. Sarkar, "Parallelizing Explicit Formula for Arithmetic in the Jacobian of Hyperelliptic Curves," *Proc. Int'l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT '03),* J. Hartmanis, G. Goos, and J. van Leeuwen, eds., pp. 93-110, 2003.

[13] T. Wollinger, "Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems," PhD dissertation, Ruhr-Univ. Bochum, Germany, 2004.

[14] A. Hodjat, L. Batina, D. Hwang, and I. Verbauwhede, "HW/SW Co-Design of a Hyperelliptic Curve Cryptosystem Using a Microcode Instruction Set Coprocessor," *Elsevier Integration, the VLSI J.,* special issue on embedded cryptographic hardware, vol. 40, no. 1, pp. 45-51, 2006.

[15] A. Satoh and K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," *IEEE Trans. Computers,* special issue on cryptographic hardware and embedded systems, vol. 52, no. 4, pp. 449-460, Apr. 2003.

[16] I. Blake, G. Seroussi, and N.P. Smart, *Elliptic Curves in Cryptography.* Cambridge Univ. Press, 1999.

[17] N. Koblitz, *Algebraic Aspects of Cryptography,* first ed. Springer, 1998.

[18] A. Menezes, Y.-H. Wu, and R. Zuccherato, *An Elementary Introduction to Hyperelliptic Curves—Appendix,* pp. 155-178. Springer, 1998.

[19] T. Itoh and S. Tsujii, "Effective Recursive Algorithm for Computing Multiplicative Inverses in $GF(2^m)$," *Electronics Letters,* vol. 24, no. 6, pp. 334-335, 1988.

[20] IEEE P1363/D13 (Draft Version 13), Standard Specifications for Public Key Cryptography, Nov. 1999.

[21] J. López and R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$," *Proc. First Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '99),* Ç.K. Koç and C. Paar, eds., pp. 316-327, 1999.

[22] B. Byramjee and S. Duquesne, *Classification of Genus 2 Curves over F_2^n and Optimization of Their Arithmetic,* Cryptology ePrint Archive: Report 2004/107, 2004.

[23] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curves Cryptography.* Springer, 2004.

[24] N.A. Saqib, F. Rodríguez-Henruez, and A. Díaz-Pérez, "A Reconfigurable Processor for High Speed Point Multiplication in Elliptic Curves," *Int'l J. Embedded Systems,* vol. 1, nos. 3/4, pp. 237-249, 2005.

[25] K. Sakiyama, B. Preneel, and I. Verbauwhede, "A Fast Dual-Field Modular Arithmetic Logic Unit and Its Hardware Implementation," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '06),* pp. 787-790, 2006.

[26] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Superscalar Coprocessor for High-speed Curve-Based Cryptography," *Proc. Eighth Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '06),* L. Goublin and M. Matsui, eds., pp. 415-429, 2006.

[27] P. Schaumont and I. Verbauwhede, "Interactive Cosimulation with Partial Evaluation," *Proc. Design, Automation and Test in Europe Conf. (DATE '04),* pp. 642-647, 2004.

[28] US Dept. of Commerce and Nat'l Inst. of Standards and Technology, Digital Signature Standard (DSS) FIPS PUB 186-2, Jan. 2000.

[29] F. Sozzani, G. Bertoni, S. Turcato, and L. Breveglieri, "A Parallelized Design for an Elliptic Curve Cryptosystem Coprocessor," *Proc. Int'l Symp. Information Technology: Coding and Computing (ITCC '05),* pp. 626-630, 2005.

[30] H. Eberle, N. Gura, and S.C. Shantz, "Cryptographic Processor for Arbitrary Elliptic Curves over $GF(2^m)$," *Proc. IEEE Int'l Conf. Application-Specific Systems, Architectures, and Processors (ASAP '03),* M. Schulte, S. Bhattacharyya, N. Burgess, and R. Schreiber, eds., pp. 444-454, June 2003.

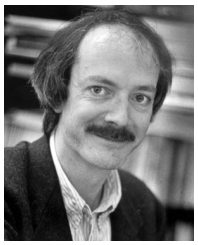
[31] R.C.C. Cheung, N.J. Telle, W. Luk, and P.Y.K. Cheung, "Customizable Elliptic Curve Cryptosystems," *IEEE Trans. Very Large Scale Integration Systems,* vol. 13, no. 9, pp. 1048-1059, 2005.

[32] J. Lutz and A. Hasan, "High Performance FPGA Based Elliptic Curve Cryptographic Co-Processor," *Proc. Int'l Conf. Information Technology: Coding and Computing,* vol. 02, p. 486, 2004.

[33] G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$," *Proc. Second Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '00),* Ç.K. Koç and C. Paar, eds., pp. 41-56, 2000.

Kazuo Sakiyama received the BEng and MEng degrees in electrical engineering from Osaka University, Japan, in 1994 and 1996, respectively, and the MS degree in electrical engineering from the University of California, Los Angeles (UCLA). From 1996 to 2004, he was with the Semiconductor and IC Division of Hitachi, Ltd. (now Renesas Technology Corp.). He is currently working toward the PhD degree at the Katholieke Universiteit Leuven (KU Leuven), Belgium. His main research interests include efficient and secure embedded system architectures and design methodologies. He is a student member of the IEEE.

Lejla Batina received the MSc degree in mathematics from the University of Zagreb, Croatia, in 1995 and the PhD degree in engineering from the Katholieke Universiteit Leuven (KU Leuven), Belgium, in 2005. She is with the Computer Security and Industrial Cryptography (COSIC) Research Group, Electrical Engineering Department, KU Leuven, as a postdoctoral researcher. She also studied and worked as a research assistant at the Technical University of Eindhoven, The Netherlands, from 1999 to 2001. Her research interests include efficient arithmetic for cryptographic algorithms, secure implementations of cryptographic algorithms, side-channel security, lightweight cryptography, for example, cryptography for radio frequency identifications (RFIDs), sensor networks, and so forth. She is a member of the IEEE.



Bart Preneel received the degree in electrical engineering and the doctorate in applied sciences from the Katholieke Universiteit Leuven, Belgium, where he is currently a full professor. He was a visiting professor at several universities in Europe. He is the vice president of the International Association for Cryptologic Research (IACR) and a member of the editorial board of the *Journal of Cryptology* and the *IEEE Transactions on Forensics and Information Security*. He has participated in 20 research projects sponsored by the European Commission, acting as the project manager for four of these. He has been the program chair of nine international conferences (including the 19th Annual Eurocrypt Conference (Eurocrypt '00), 20th Annual ACM Symposium on Applied Computing (SAC '05), and Ninth Information Security Conference (ISC '06)), and he has been an invited speaker at 25 conferences. He is the president of Leuven Security Excellence Consortium (L-SEC) vzw., an association of 45 companies and research institutions in the area of e-security. His main research interests are cryptography and information security. He has authored or coauthored more than 200 scientific publications. In 2003, he received the European Information Security Award in the area of academic research and he received an honorary Certified Information Security Manager (CISM) designation from the Information Systems Audit and Control Association (ISACA). He is a member of the IEEE. His Web site is http://homes.esat.kuleuven.be/~preneel.

Ingrid Verbauwhede received the degree in electrical engineering and the PhD degree from the Katholieke Universiteit Leuven (KU Leuven), Belgium. She was a lecturer and a visiting research engineer at the University of California, Berkeley, from 1992 to 1994. From 1994 to 1998, she was a principal engineer first at TCSI and then at Atmel in Berkeley, California. She joined the University of California, Los Angeles (UCLA) as an associate professor in 1998 and KU Leuven in 2003. She is currently a professor at KU Leuven and an adjunct associate professor at UCLA. At KU Leuven, she is the codirector of the Computer Security and Industrial Cryptography (COSIC) Laboratory. She is active in many conferences. She was the program chair in 2002 and the general chair in 2003 of the International Symposium on Low Power Electronics and Design (ISLPED). She was a member of the executive committee of the 42nd and 43rd Design Automation Conference (DAC '05 and '06) as the design community chair. She is the program chair of the 2007 Workshop on Cryptographic Hardware and Embedded Systems (CHES '07). Her research interests include circuits, processor architectures, and design methodologies for real-time embedded systems for security, cryptography, digital signal processing, and wireless communications. This includes the influence of new technologies and new circuit solutions on the design of next-generation systems on chip. She is a senior member of the IEEE. More information on her research can be found at www.emsec.ee.ucla.edu or www.esat.kuleuven.be/cosic.