Kimmo Järvinen and Jorma Skyttä, On Parallelization of High-Speed Processors for Elliptic Curve Cryptography, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 9, Sep. 2008, pp. 1162-1175.

On Parallelization of High-Speed Processors for Elliptic Curve Cryptography

Kimmo Järvinen, *Student Member, IEEE*, and Jorma Skyttä

***Abstract*—This paper discusses parallelization of elliptic curve cryptography hardware accelerators using elliptic curves over binary fields $\mathbb{F}_{2m}$. Elliptic curve point multiplication, which is the operation used in every elliptic curve cryptosystem, is hierarchical in nature, and parallelism can be utilized in different hierarchy levels as shown in many publications. However, a comprehensive analysis on the effects of parallelization has not been previously presented. This paper provides tools for evaluating the use of parallelism and shows where it should be used in order to maximize efficiency. Special attention is given for a family of curves called Koblitz curves because they offer very efficient point multiplication. A new method where the latency of point multiplication is reduced with parallel field arithmetic processors is introduced. It is shown to outperform the previously presented multiple field multiplier techniques in the cases of Koblitz curves and generic curves with fixed base points. A highly efficient general elliptic curve cryptography processor architecture is presented and analyzed. Based on this architecture and analysis on the effects of parallelization, a few designs are implemented on an Altera Stratix II field-programmable gate array (FPGA).**

***Index Terms*—Elliptic curve cryptography (ECC), field-programmable gate arrays (FPGAs), Koblitz curves, parallel processing, public key cryptography.**

I. Introduction

THE USE OF elliptic curves in public-key cryptography was independently proposed by Koblitz [1] and Miller [2] in 1985 and, since then, an enormous amount of work has been done on elliptic curve cryptography (ECC). The attractiveness of using elliptic curves arises from the fact that similar level of security can be achieved with considerably shorter keys than in methods based on the difficulties of solving discrete logarithms over integers or integer factorizations.

Public-key cryptography is computationally intensive, and hardware acceleration is frequently required in practical applications. Thus, many publications have considered hardware acceleration of ECC. Some application-specific integrated circuit (ASIC) implementations have been published such as [3]–[6], but the majority of designs including [7]–[23] have been implemented on field-programmable gate arrays (FPGAs). A comprehensive survey of hardware acceleration of ECC is given in [24].

The research on hardware acceleration has concentrated on efficient implementation of elliptic curve point multiplication, the fundamental operation of all elliptic curve cryptosystems. The elliptic curve point multiplication is computed with point operations which, further, are computed using finite field arithmetic. The sequential nature of the point multiplication makes efficient use of parallelization challenging. However, although the point multiplication itself is hard to parallelize, it is possible to efficiently use parallelism in lower hierarchy levels, namely in point operations [9], [11], [16] and field arithmetic [14], [15], [25], [26].

Many published articles use parallel computing in both point operations, e.g., multiple field multipliers, and field arithmetic operations, e.g., digit-serial multipliers, without making any analysis of their efficiency. This paper provides tools for evaluating the use of parallelism and points out where parallelism should be used in order to maximize efficiency.

Koblitz curves [27] are a family of curves on which point multiplication is considerably faster than on generic curves. Thus, Koblitz curves are included in many standards, e.g., [28], [29]. Despite their efficiency, only few publications on hardware implementation have considered Koblitz curves. To the authors' knowledge, they have been discussed only in [12], [17], [19], [20]. Koblitz curves were shown to be fast and easy to implement in software in [30]. It is shown in this paper that point multiplication on Koblitz curves can be computed very efficiently also in hardware. In addition to faster point multiplication, Koblitz curves also provide interesting possibilities for further use of parallelism compared to generic curves as will be shown in this paper.

The main contributions of this work include the following (in order of appearance):

- highly efficient general ECC processor architecture is described for FPGAs (see Section IV);
- analysis on existing parallelization techniques is presented (see Section V);
- fair comparison between existing techniques is given which is possible because different techniques are evaluated on the same architecture (see Section V);
- method for reducing latency by using parallel processors is presented and analyzed (see Section VI);
- very efficient high-speed FPGA-based implementations are described (see Section VII).

Emphasis of this work is on studying effects of parallelization on performance, area, and their tradeoff in high-speed accelerators. Such aspects as side-channel attacks are not considered in order to keep the work focused.

The remainder of this paper is organized as follows. Section II presents preliminaries of ECC. Parallelization of point multiplication is discussed and previous work on the subject is reviewed in Section III. Section IV introduces the processor architecture

Manuscript received February 9, 2007; revised September 22, 2007. Published August 20, 2008 (projected). This work was supported in part by TEKES under Contract 40508/05.

The authors are with the Signal Processing Laboratory, Helsinki University of Technology, FIN-02015 TKK, Finland (e-mail: kimmo.jarvinen@tkk.fi; jorma.skytta@tkk.fi).

Digital Object Identifier 10.1109/TVLSI.2008.2000728

that is used in the analysis. Parallelization techniques are described and their effects on the architecture are discussed in Section V. A new method for reducing latency with parallel field arithmetic processors is suggested and analyzed in Section VI. Finally, results on an Altera Stratix II FPGA are presented in Section VII and conclusions are drawn in Section VIII.

II. Elliptic Curve Cryptography (ECC)

This paper considers elliptic curves defined over finite binary fields $\mathbb{F}_{2^m}$. The curves are so-called ordinary curves (see [31], for example), i.e., they are defined as

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

with $a, b \in \mathbb{F}_{2^m}$ so that $b \neq 0$. Let $E(\mathbb{F}_{2^m})$ denote the set of all points on E. A pair (x, y), where $x, y \in \mathbb{F}_{2^m}$, is a point in $E(\mathbb{F}_{2^m})$ if it satisfies (1). The point at infinity, denoted as $\mathcal{O}$, is also a point in $E(\mathbb{F}_{2^m})$.

A binary field $\mathbb{F}_{2^m}$ with polynomial basis (PB) is constructed by representing field elements as polynomials of degree at most $m - 1$. Addition is performed as an addition of polynomials modulo 2, i.e., a bitwise exclusive-or (XOR), and multiplication is performed modulo an irreducible polynomial. In normal basis (NB), the elements are represented with a basis of the form $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$ with the property that $\beta^{2^m} = \beta$. Obviously, squaring is a cyclic shift and thus very inexpensive but multiplications cost more than in PB. Inversion of $x \in \mathbb{F}_{2^m}$, i.e., computing $x^{-1} \in \mathbb{F}_{2^m}$ such that $x^{-1}x = 1$, is the most expensive field operation regardless of the basis, see, e.g., [32]. A method based on Fermat's Little Theorem suggested by Itoh and Tsujii in [33] is used in this paper.

Every elliptic curve cryptosystem requires computation of an operation called elliptic curve point multiplication. Given a point $P = (x, y) \in E(\mathbb{F}_{2^m})$, called the base point, and an integer k, the point multiplication is defined as

$$Q = kP = \underbrace{P + P + \cdots + P}_{k \text{ times}} \tag{2}$$

where $Q \in E(\mathbb{F}_{2^m})$ is called the result point. Point addition, $P_1 + P_2$ with $P_{1,2} \in E(\mathbb{F}_{2^m})$, and point doubling, $2P_1$, are basic operations in computing (2).

Binary method (or double-and-add method) is probably the most common way to compute (2). In the binary method, k is represented with binary expansion as $\sum_{i=0}^{\ell-1} k_i 2^i$ and a point doubling is performed for each bit k_i and a point addition if $k_i = 1$. Thus, because $\ell \approx m$, m point doublings and $m/2$ point additions are required on average.

Traditionally points are represented by using two coordinates as (x, y). This representation is henceforth referred to as affine coordinates, or $\mathcal{A}$ for short. If points are presented in $\mathcal{A}$, point addition and point doubling both require one inversion in $\mathbb{F}_{2^m}$. Because inversions are expensive, it is commonly preferred to represent points by using three coordinates as (X, Y, Z) because then the number of inversions in computation of (2) can be reduced to one. Such coordinate systems considered in this paper are projective and López-Dahab coordinates, $\mathcal{P}$ and $\mathcal{LD}$ for short. A point (X, Y, Z) in $\mathcal{P}$ and $\mathcal{LD}$ represents the affine points $(X/Z, Y/Z)$ and $(X/Z, Y/Z^2)$, respectively. The $\mathcal{A} \mapsto \mathcal{P}$ and $\mathcal{A} \mapsto \mathcal{LD}$ mappings are simply $(x, y, 1)$ and do not require any operations.

The costs of addition, squaring, multiplication, and inversion in $\mathbb{F}_{2^m}$ are denoted as A, S, M, and I, respectively. Point operation costs vary depending on coordinate systems. Table I lists point operation costs that are relevant in this paper. Considerable savings can be achieved by computing point addition $P_1 + P_2$ in mixed coordinates where P_1 and P_2 are in different coordinates, e.g., P_1 in $\mathcal{LD}$ and P_2 in $\mathcal{A}$.

The Hamming weight of k, denoted as $H(k)$, is the number of nonzero terms in the representation of k. It is of interest to reduce $H(k)$, because point additions are required only when $k_i \neq 0$. When a signed-bit representation in non-adjacent form (NAF), i.e., $k_i \in \{0, \pm 1\}$ so that $k_i k_{i+1} = 0$, for all i, is used, $H(k) = m/3$ on average. $H(k)$ can be further reduced with windowing methods, but then certain points need to be precomputed (see [32], for example).

An efficient method for computing (2) was presented in [34] by López and Dahab. The method is called the Montgomery point multiplication and it computes (2) in $\mathcal{P}$ with only the x-coordinate (X and Z) and the y-coordinate is recovered in the end. Both point doubling and point addition are computed for every k_i, but they are very efficient to compute because only the x-coordinate is considered. The combined recovery of the y-coordinate and $\mathcal{P} \mapsto \mathcal{A}$ mapping requires certain additional field operations as shown in Table I.

Curves for which $a \in \{0, 1\}$ and $b = 1$ in (1) are called Koblitz curves [27]. They have special attractiveness among elliptic curves, because point doublings can be replaced by efficiently computable Frobenius endomorphism [27]. Let K_a be a Koblitz curve. The Frobenius map $\tau : K_a(\mathbb{F}_{2^m}) \mapsto K_a(\mathbb{F}_{2^m})$ is defined by

$$\tau(x, y) = (x^2, y^2) \quad \tau(\mathcal{O}) = \mathcal{O}. \tag{3}$$

Frobenius maps cost $2\mathsf{S}$ or $3\mathsf{S}$ depending on the coordinate system. Notice that squaring is cheap. Actually, squaring in $\mathbb{F}_{2^m}$ with NB is only a cyclic shift of the bit vector. Thus, the cost of (2) is only $H(k) - 1$ point additions[1] with the binary method. Before the fast Frobenius maps can be utilized, the integer k needs to be converted in τ-adic expansion as $\sum_{i=0}^{\ell-1} k_i \tau^i$, where $\tau = ((-1)^{1-a} + \sqrt{-7})/2$. Algorithms for converting integers into τ-adic non-adjacent form (τNAF) were presented in [38].

III. Parallelism in Point Multiplication

As presented in [9], for example, the point multiplication decomposes into three hierarchical levels as shown in Fig. 1(a). Similar hierarchical levels can be found from the ways of how parallelism is used inside an accelerator. The hierarchy of parallelism is depicted in Fig. 1(b). This hierarchy is studied next from the bottom to the top.

[1]The first point addition is considered free as it is simply a substitution.

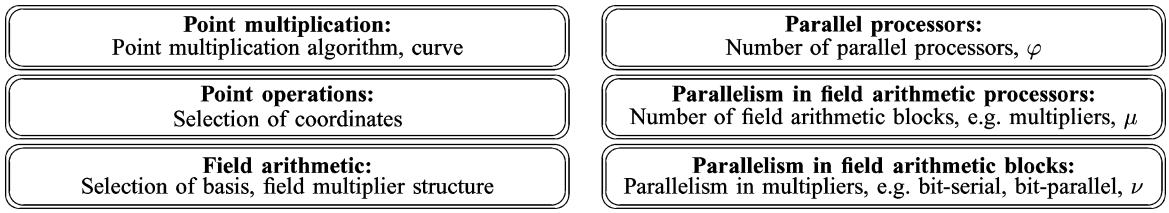

Fig. 1. Hierarchical levels of (a) computation of elliptic curve point multiplication [9] and (b) parallelization of hardware implementations. Decisions made in (a) define which techniques can be used in hierarchical levels of (b). For example, selection of the basis, i.e., PB or NB, on the lowest level of (a) define which multiplier architectures can be used in the lowest level of (b), etc.

TABLE I
COSTS OF POINT OPERATIONS (SEE [31], FOR EXAMPLE)

Operation	Coordinates	Cost
Addition	$\mathcal{A}$	I + 2M + S + 8A
Doubling	$\mathcal{A}$	I + 2M + S + 6A
Negation	$\mathcal{A}$	A
Addition	$\mathcal{P}$	16M + 2S + 8A
Doubling	$\mathcal{P}$	8M + 4S + 5A
Montgomery[a] [34]	$\mathcal{P}$	6M + 4S + 3A
Negation	$\mathcal{P}$	A
Addition [35]	$\mathcal{LD}$	13M + 4S + 9A
Doubling [36]	$\mathcal{LD}$	5M + 4S + 5A
Negation	$\mathcal{LD}$	M + A
Addition[b] [37]	$\mathcal{LD}/\mathcal{A}$	9M + 5S + 9A
Montgomery[c] [34]	$\mathcal{P} \mapsto \mathcal{A}$	I + 10M + S + 6A
Mapping	$\mathcal{P} \mapsto \mathcal{A}$	I + 2M
Mapping	$\mathcal{LD} \mapsto \mathcal{A}$	I + 2M + S

[a]Point addition and doubling; only the x-coordinate.
[b]One multiplication saved if $a \in \{0, 1\}$ and one addition is saved if $a = 0$.
[c]Recovery of the y-coordinate and $\mathcal{P} \mapsto \mathcal{A}$ mapping.

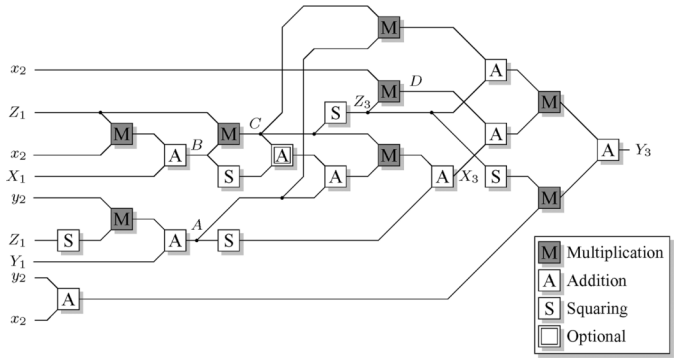

Fig. 2. Data dependency graph of computing (4). The critical path comprises 8, 5, or 4 multiplications if one, two, or three multipliers are available, respectively. The optional addition is performed if $a = 1$ and omitted if $a = 0$. If $a \notin \{0, 1\}$, one additional multiplication, which is not depicted, is required (aC).

A. Parallelism in Field Arithmetic Blocks

Parallelism in field arithmetic blocks, mostly in multipliers,[2] has been studied in numerous publications. Multiplication is the operation which has the most crucial effect on the performance and area of an accelerator. Work on parallelization in field multipliers includes, e.g., [14], [15], [25], [26] for PB and [39] for NB. A bit-serial multiplier computes one bit of the output per cycle with a single processing block resulting in latency of m. In bit-parallel multipliers, all m bits of the output are computed in one cycle. A digit-serial multiplier is a tradeoff where ν bits of the output are computed in parallel, thus, resulting in latency of $\lceil m/\nu \rceil$.

B. Parallelism in Field Arithmetic Processors

Parallelism in point operations is also an efficient way to reduce latency of point multiplication as shown, e.g., in [9], [11], [16], [22], and [40]. Certain field operations can be computed in parallel depending on the coordinate system. Parallelism that can be utilized in Montgomery point multiplication [34] and point addition in mixed coordinates [37] is considered next. Only multiplications are examined as adding parallel adders or squarers has only a negligible effect on performance. Let μ denote the number of parallel field multipliers.

Montgomery point multiplication [34] is used for generic curves because it is the most efficient method which does not involve precomputations [31]. Point addition and doubling together require six multiplications and parallel multipliers can be utilized efficiently as shown in [11], [22]. With $\mu = 2$, the critical path reduces to three multiplications and, with $\mu = 4$, to only two multiplications [11], [22]. Three or more than four multipliers do not give any further improvements.

For Koblitz curves point additions are computed with the mixed coordinate point addition as presented in [37]. The formula for computing $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2)$ are as follows [37]:

$$\begin{aligned} A &= Y_1 + y_2 Z_1^2 \\ B &= X_1 + x_2 Z_1 \\ C &= B Z_1 \\ Z_3 &= C^2 \\ D &= x_2 Z_3 \\ X_3 &= A^2 + C(A + B^2 + aC) \\ Y_3 &= (D + X_3)(AC + Z_3) + (y_2 + x_2) Z_3^2. \end{aligned} \tag{4}$$

The data dependency graph of (4) is presented in Fig. 2 which shows that (4) requires eight multiplications but the critical path can be reduced to five or four multiplications with $\mu = 2$ or $\mu = 3$, respectively. Data dependencies restrict from achieving further reductions with more than three multipliers.

C. Parallel Processors

Parallel processors can be used for increasing throughput of an accelerator by simply computing several point multiplica-

[2]The term "multiplier" refers exclusively to field multipliers in this paper. Similarly, "multiplication" always refers to field multiplication and point multiplication is consistently referred to with its full name.

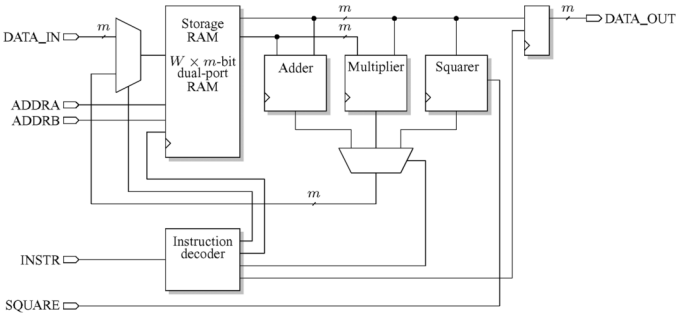

Fig. 3. Block diagram of the FAP.

tions in parallel. Optimizations in parallel processor cases were recently studied by the authors in [17]. Using parallel processors for reducing computation latency is hard because of the sequential nature of the point multiplication and, at least to the authors' knowledge, the method discussed in Section VI is the first such method presented in the literature.

IV. Architecture of the Accelerator

This section presents an accelerator computing elliptic curve point multiplication which is used for studying the effects of parallelization. The accelerator comprises field arithmetic processor (FAP), FAP control logic and interface logic. A τNAF converter is also required for Koblitz curves. The architecture itself is generic but the implementations are optimized for Altera Stratix II FPGAs [41].

Montgomery point multiplication is used for generic curves and the binary method where point additions are computed in mixed coordinates with k in τNAF is used for Koblitz curves. These methods were selected because they are the fastest methods which do not require precomputations [31].

For simplicity, the discussion in the remainder of this paper is restricted to the smallest field size $m = 163$ specified in the NIST recommended elliptic curves for federal government use [28]; namely the curves NIST B-163 and NIST K-163 are used. This does not sacrifice the generality of the architecture, the methods, or the analysis. Again, to keep discussion clear and simple, only NB $\mathbb{F}_{2^m}$ are considered similarly as in [11], for example. The methods and analysis tools are valid for PB too, but the results would be different, of course.

A. Field Arithmetic Processor

The FAP consists of adder, squarer, multiplier(s), storage RAM, and instruction decoder. A block diagram is presented in Fig. 3.

1) Adder and Squarer: The adder computes an m-bit bitwise XOR in one clock cycle, i.e., $\mathsf{A} = 1$. The squarer is a shifter which can compute d successive squarings x^{2^d}, where $x \in \mathbb{F}_{2^m}$ and $d \in [0, 31]$. Computation requires one clock cycle, i.e., $\mathsf{S} = 1$.

2) Multiplier: Multiplication is critical for the overall performance. Multiplication in NB is computed with a Massey-Omura multiplier [39]. One bit z_i of the result $z = x \times y$, where $x, y, z \in \mathbb{F}_{2^m}$ is computed from x and y by using a logic function called the F-function. Formulae for constructing the F-function are publicly available in the appendices of [28], for example. The F-function is field specific, and the same F is used for all output bits z_i as follows: $z_i = F(x_{\lll i}, y_{\lll i})$, where $\lll i$ denotes cyclical left shift by i bits. Hence, a bit-serial implementation of the Massey-Omura multiplier requires three m-bit shift registers and one F-function block. A bit-parallel multiplier requires m F-function blocks and an m-bit register for storing the result [28], [39].

In practice, the bit-serial multiplier requiring at least $m + 1$ clock cycles is too slow and the bit-parallel multiplier requires too much area. A good tradeoff is a digit-serial multiplier, where ν bits are computed in parallel with ν F-function blocks. The F-function blocks can be pipelined in order to increase the maximum clock frequency. The latency of a digit-serial multiplier is

$$\mathsf{M} = \left\lceil \frac{m}{\nu} \right\rceil + c + 1 \tag{5}$$

where c is the number of pipeline stages, i.e., $c \geq 0$. In this paper, $c = 1$. One clock cycle is also required in loading the operands into the shift registers.

FAPs can include several multipliers and the number of multipliers is denoted with μ.

3) Others: The storage RAM is used for storing elements of $\mathbb{F}_{2^m}$. It is implemented as a dual-port RAM by using embedded memory, e.g., M4K in Stratix II [41]. The storage RAM stores up to W elements. When Stratix II is used, a logical choice is $W = 256$ because, while in true dual-port mode, the widest mode that an M4K block can be configured to is 256 $\times$ 18-bits. The width of the storage RAM was selected to be 163 bits in order to minimize writing and reading delays. In memory constrained environments narrower bus widths could be used in order to reduce memory requirements at the expense of longer delays. The storage RAM requires $\lceil 163/18 \rceil = 10$ M4Ks resulting in a storage capacity of 256 $\times$ 163-bits. This much storage space is rarely needed, but it can be used for example for storing precomputed points. Furthermore, selecting a smaller depth would not reduce M4Ks. Both writing and reading require one clock cycle. However, the dual-port RAM can be configured into the read-during-write mode [41] which saves certain clock cycles, see Section IV-B.

The instruction decoder simply decodes instructions to signals controlling the FAP blocks.

B. Control Logic

The FAP control logic consists of finite-state machine (FSM) and ROM containing instruction sequences.

The instruction sequences are carefully hand-optimized in order to minimize latencies of point operations. As mentioned in Section IV-A3, the read-during-write mode can be used for reducing latencies. Operations are ordered so that the result of the previous operation is used as the operand of the next operation whenever possible. One clock cycle is saved every time this can be used, because the operands of the next operation can be read simultaneously with the writing of the result of the previous operation.

Inversions are computed with successive multiplications and squarings as suggested by Itoh and Tsujii in [33]. An Itoh-Tsujii inversion has the constant cost of

$$\mathsf{I} = (\lfloor \log_2(m-1) \rfloor + H(m-1) - 1)\,\mathsf{M} + (m-1)\mathsf{S} \tag{6}$$

TABLE II
INSTRUCTION SEQUENCES AND THEIR LATENCIES

Operation	Latency
Interfacing	11
1st point addition (Generic)	14
Montgomery add and double, $\mu = 1$	6M+29
Montgomery add and double, $\mu = 2$	3M+27
Montgomery add and double, $\mu = 4$	2M+26
Montgomery $\mathcal{P} \mapsto \mathcal{A}$ map, $\mu = 1$	19M+75
Montgomery $\mathcal{P} \mapsto \mathcal{A}$ map, $\mu = 2$	15M+74
Montgomery $\mathcal{P} \mapsto \mathcal{A}$ map, $\mu = 4$	13M+72
1st point addition (Koblitz)	5
Point addition $\mathcal{LD}/\mathcal{A}$, $\mu = 1$	8M+40
Point addition $\mathcal{LD}/\mathcal{A}$, $\mu = 2$	5M+43
Point addition $\mathcal{LD}/\mathcal{A}$, $\mu = 3$	4M+42
Frobenius maps τ^d	7
$\mathcal{LD} \mapsto \mathcal{A}$ map, , $\mu = 1$	11M+48
$\mathcal{LD} \mapsto \mathcal{A}$ map, , $\mu \geq 2$	10M+48
Point addition $\mathcal{A}$, $\mu = 1$	11M+71
kP (Generic), $\mu = 1$	$(\ell - 1)(6\mathsf{M} + 29) + 19\mathsf{M} + 100$
kP (Generic), $\mu = 2$	$(\ell - 1)(3\mathsf{M} + 27) + 15\mathsf{M} + 99$
kP (Generic), $\mu = 4$	$(\ell - 1)(2\mathsf{M} + 26) + 13\mathsf{M} + 97$
kP (Koblitz), $\mu = 1$	$(H(k) - 1)(8\mathsf{M} + 47) + 11\mathsf{M} + 71$
kP (Koblitz), $\mu = 2$	$(H(k) - 1)(5\mathsf{M} + 50) + 10\mathsf{M} + 71$
kP (Koblitz), $\mu = 3$	$(H(k) - 1)(4\mathsf{M} + 49) + 10\mathsf{M} + 71$

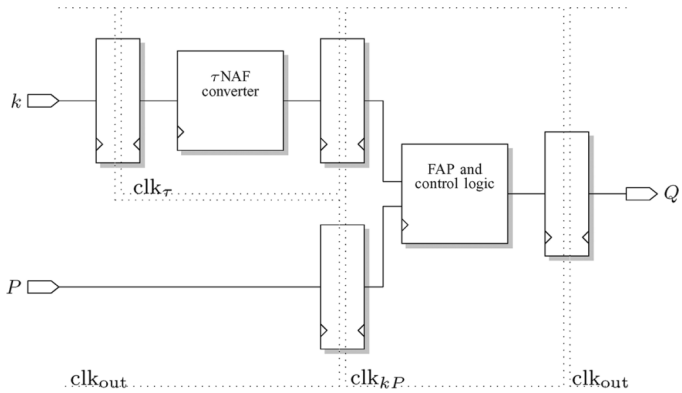

Fig. 4. Top-level view of the accelerator. The accelerator is divided into three clock domains: all communication with other components is clocked with $\mathrm{clk}_{\mathrm{out}}$. The FAP and its control logic operate at clk_{kP} and the τNAF converter uses clk_{τ}. Because the converter is needed only for Koblitz curves, generic curve implementations only have two clocks, $\mathrm{clk}_{\mathrm{out}}$ and clk_{kP}.

which results in $9\mathsf{M} + 162\mathsf{S}$ when $m = 163$ [33]. Although the number of squarings is high, the successive squaring feature of the squarer (see Section IV-A1) ensures that the cost remains reasonable.

As mentioned in Sections III and IV-A2, the latencies of point operations can be reduced with parallel multipliers, i.e., $\mu \geq 2$. Multiplications in the Itoh-Tsujii inversion cannot be computed in parallel but Montgomery point addition and doubling and point addition in mixed coordinates benefit from parallel multipliers as shown in Section III. Instruction sequences were optimized for different μ. Table II lists the latencies of instruction sequences used in this paper and presents the latencies of computing (2) with different setups.

C. Top-Level and Clocking

When (2) is computed on a Koblitz curve, a converter is required for converting k into τ-adic expansion. The τNAF converter presented by the authors in [42] is used in the designs. The latency of a conversion is $3m + 2$ clock cycles on average [42].

TABLE III
AREAS OF THE BLOCKS OF THE ACCELERATOR

Symbol	Description	Area (ALMs)
A_F	Area of an F- function block	177
A_c	Constant area of a multiplier	407
A_p	Constant area of an FAP	730
A_i	Area of the interface logic	233
A_τ	Area of a τNAF converter	928

Because the converter has a lower maximum clock frequency than the rest of the circuitry, it is separated into its own clock domain as shown in Fig. 4. The accelerator has an interface clock $\mathrm{clk}_{\mathrm{out}}$, the FAP, and its control logic operate with the clock clk_{kP}, and the τNAF converter operates with the clock clk_{τ}. The clock domains are separated by first-in, first-out (FIFO) buffers implemented in embedded memory, i.e., in M512 and M4K in Stratix II.

Table III presents the areas of different blocks in the architecture which are used in Section V for analyzing the effects of parallelization. The areas are averages from the values received from the synthesis for Stratix II FPGA because the exact values varied slightly after the place&route. The areas are given as the number of occupied adaptive logic modules (ALMs). It is assumed that the total area depends linearly on the number of blocks. Hence, an estimate for the area of an implementation on Stratix II is given by

$$A(\varphi, \psi, \mu, \nu) = \varphi\left(\mu(\nu A_F + A_c) + A_p\right) + \psi A_\tau + A_i \quad (7)$$

where φ, ψ, μ, and ν are the numbers of parallel FAPs, τNAF converters, field multipliers, and F-function blocks, respectively, and the areas are as in Table III.

V. PARALLELISM IN ELLIPTIC CURVE ACCELERATORS

This section discusses effects of parallelization in different parts of the ECC processor presented in Section IV. It is assumed in the following analysis that the complexity of the design does not have an effect on the quality of place&route results, i.e., on area or timings. Thus, the area of an accelerator is assumed to be given by (7). The clock frequency is assumed to be constant for all implementations because the same F-function block determines the critical path, see Section IV-A2. These assumptions are necessary in order to provide an analytic approach to parallelization. However, in reality the more difficult place&route becomes the more area it usually has to consume in order to meet the given timing constraints and the more probable it becomes that these constraints are not met at all. Hence, it is probable that the estimates given in the analysis are too optimistic for the most complex designs. Inaccuracies caused by the assumptions are analyzed in Section VII.

First, metrics for evaluating designs are defined. Let $\mathbf{p}$ denote the parameters which define the degrees of parallelism used in the accelerator, i.e., $\mathbf{p} = (\varphi, \psi, \mu, \nu)$.

Performance is rated by three metrics, namely latency, point multiplication time, and throughput. Latency is the average number of clock cycles required to compute point multiplication. Point multiplication time, hereafter referred to as pm-time, is the average time in seconds required in point multiplication. Throughput is the maximum number of point multiplications

computed in a given time frame on average. Throughput is measured with operations per second (ops).

Let $L(\mathbf{p})$, $t(\mathbf{p})$, and $T(\mathbf{p})$ denote latency, pm-time, and throughput with parallelism parameters $\mathbf{p}$, respectively. Parallelism may also have an impact on the maximum clock frequency and, therefore, the frequency, $f(\mathbf{p})$, must be considered as well. Latency, pm-time, and throughput are related through the following formula:

$$t(\mathbf{p}) = L(\mathbf{p})/f(\mathbf{p}) \tag{8}$$

$$T(\mathbf{p}) = \hat{\varphi}/t(\mathbf{p}) \tag{9}$$

where $\hat{\varphi}$ is the maximum number of point multiplications that can be computed in parallel. Because in this section each FAP computes a single point multiplication at a time, $\hat{\varphi} = \varphi$.

Let $A(\mathbf{p})$ denote the area of the accelerator with $\mathbf{p}$. In the following analysis, $A(\mathbf{p})$ is given by (7). Important evaluation metrics are the latency-area $R_L(\mathbf{p})$, time-area $R_t(\mathbf{p})$, and throughput-area $R_T(\mathbf{p})$ ratios defined as

$$R_L(\mathbf{p}) = 1/\left(L(\mathbf{p})A(\mathbf{p})\right) \tag{10}$$

$$R_t(\mathbf{p}) = 1/\left(t(\mathbf{p})A(\mathbf{p})\right) \tag{11}$$

$$R_T(\mathbf{p}) = T(\mathbf{p})/A(\mathbf{p}). \tag{12}$$

The higher the ratios are the better the implementation can be considered by that metric. Notice that, if the accelerator computes only one point multiplication at a time, $R_T(\mathbf{p}) = R_t(\mathbf{p}) = R_L(\mathbf{p})f(\mathbf{p})$. However, if an accelerator is capable of computing several point multiplications simultaneously, i.e., $\hat{\varphi} > 1$, $R_T(\mathbf{p}) > R_t(\mathbf{p}) = R_L(\mathbf{p})f(\mathbf{p})$.

Two designs with parallelism parameters $\mathbf{p}_1$ and $\mathbf{p}_2$ can be compared with speedup ratios as follows:

$$S_L(\mathbf{p}_1, \mathbf{p}_2) = L(\mathbf{p}_1)/L(\mathbf{p}_2) \tag{13}$$

$$S_t(\mathbf{p}_1, \mathbf{p}_2) = t(\mathbf{p}_1)/t(\mathbf{p}_2) \tag{14}$$

$$S_T(\mathbf{p}_1, \mathbf{p}_2) = T(\mathbf{p}_2)/T(\mathbf{p}_1) \tag{15}$$

for latencies, pm-times, and throughputs, respectively. All ratios describe how much faster $\mathbf{p}_2$ is compared to $\mathbf{p}_1$.

A. Parallelism in Field Multipliers

This section studies parallelization of the digit-serial Massey-Omura multiplier, see Section IV-A-2. The free parallelism parameter is the number of F-function blocks, ν.

Because the F-function blocks are the same for all bits of the result, the critical path of the multiplier is constant regardless of ν. Thus, it is assumed that the maximum clock frequency does not depend on ν and a normalized frequency $f(\nu) = 1$ can be used in the analysis. It suffices to consider only latency and area of the multiplier. The area of the multiplier consists of the area of F-function blocks and constant area of the shift registers. Thus, latency and area are given by the following formula:

$$L(\nu) = \left\lceil \frac{m}{\nu} \right\rceil + c + 1 = \left\lceil \frac{m}{\nu} \right\rceil + 2 \tag{16}$$

$$A(\nu) = \nu A_F + A_c \tag{17}$$

where A_F and A_c are as given in Table III.

Because of the round up in (16), only certain values of ν are feasible. Because $m = 163$, ν should be chosen from the set

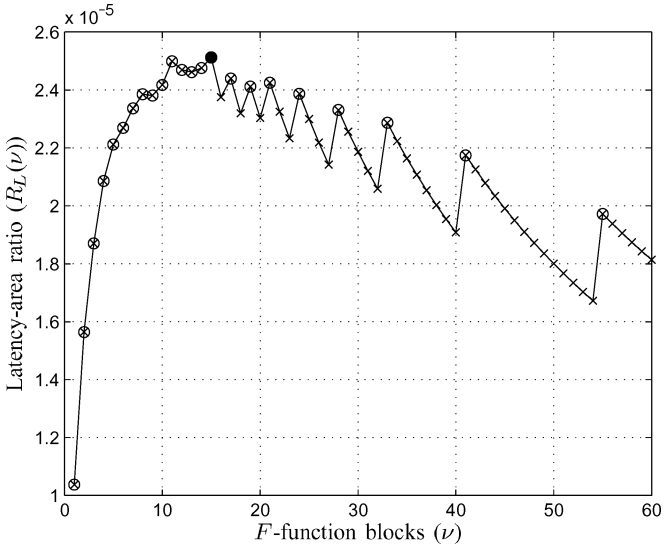

Fig. 5. Latency-area ratio $R_L(\nu)$ of the digit-serial Massey-Omura multiplier when $\mathbb{F}_2^m$ with $c = 1$, $A_c = 407$, and $A_F = 177$ on a Stratix II FPGA. The possible values of ν are denoted with $\times$, the feasible values are circled and the filled circle denotes the optimal value $\nu_{\text{opt}} = 15$.

$\{1\text{–}15, 17, 19, 21, 24, 28, 33, 41, 55, 82, 163\}$. These values are henceforth referred to as the feasible values of ν. Other values only add area but do not reduce latency. Furthermore, latency reduces by each step when ν increases if $\nu \leq 15$, but several additional F-function blocks are needed to decrease latency if $\nu > 15$. Hence, reductions in latency become more expensive when ν grows. It is not obvious which value of ν optimizes latency-area ratio $R_L(\nu)$. Fig. 5 depicts $R_L(\nu)$ and shows that the optimal value is $\nu_{\text{opt}} = 15$.

B. Parallelism in FAPs

This section studies parallelism in FAPs. Multipliers dominate in performance and area cost. Because parallel adders and squarers do not give any major performance benefits, the analysis is restricted to the number of multipliers, μ, and it is assumed that there is only one adder and squarer. The area of an FAP obtained from (7), is given by

$$A(\mu, \nu) = \mu(\nu A_F + A_c) + A_p \tag{18}$$

where A_p includes adder, squarer, and control logic.

Two questions are studied. First, what setup (μ, ν) gives the best latency-area ratio R_L and, second, how to determine whether one should use one fast multiplier or multiple slower ones? The first question is relevant when throughput is being maximized with parallel FAPs, because then one should use FAPs that give the best area efficiency. The second question has importance when one targets to certain latency and wants to achieve it with minimal area.

Fig. 6(a) and (b) plot R_L for generic and Koblitz curves, respectively. The best R_L is received when $\mu = 1$ and $\nu = 11$ for both generic and Koblitz curves. The similarity is not surprising considering that the ratio of multiplications and other operations is almost the same in both cases, see Table II, and $\nu = 11$ receives a high R_L also in the analysis of Section V-A, see Fig. 5. For Koblitz curves, $\mu = 1$ has a significantly higher R_L than

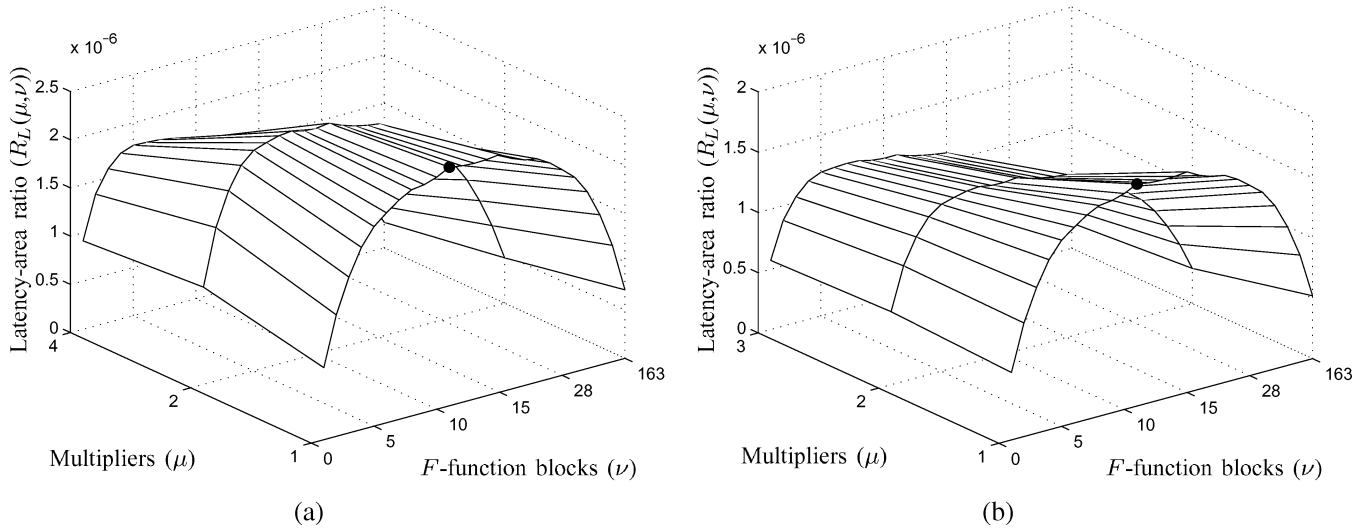

Fig. 6. Latency-area ratios $R_L(\mu,\nu)$ of the FAPs. In (a), the combined point addition and doubling [11] of the Montgomery point multiplication [34] is used and, in (b), $L(\mu,\nu)$ the mixed coordinate point addition algorithm [37] is used.

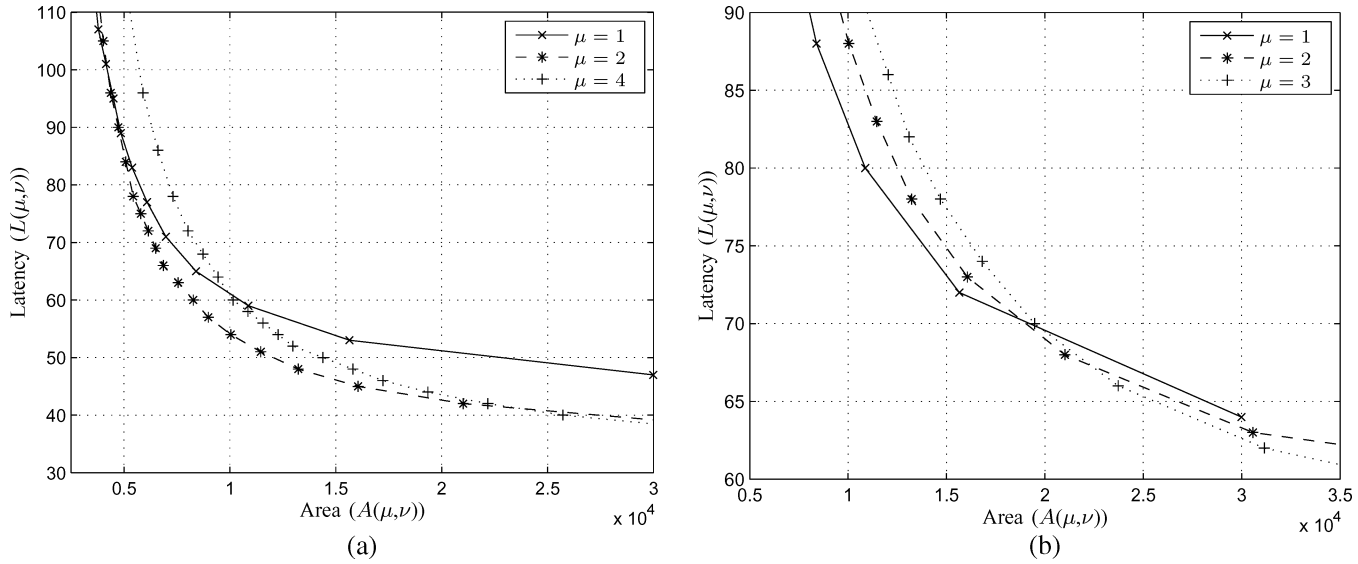

Fig. 7. Latency-area plots of the FAPs. In (a), $L(\mu,\nu)$ is the latency of the combined point addition and doubling [34] and, in (b), it is the latency of mixed coordinate point addition [37]. In (a), $\nu \in \{15,17,19,21,24,28,33,41,55,82,163\}$ when $\mu = 1$, $\nu \in \{7,8,9,10,11,12,13,14,15,17,19,21,24,28,33,41,55\}$ when $\mu = 2$ and $\nu \in \{5,6,7,8,9,10,11,12,13,14,15,17,19,21,24,28,33\}$ when $\mu = 4$. In (b), $\nu \in \{41,55,82,163\}$ when $\mu = 1$, $\nu \in \{24,28,33,41,55,82\}$ when $\mu = 2$ and $\nu \in \{19,21,24,28,33,41,55\}$ when $\mu = 3$ so that the smallest ν is on the left.

$\mu = 2,3$. Both $\mu = 1$ and $\mu = 2$ receive high R_L for generic curves, but $\mu = 4$ reduces R_L considerably.

When an implementation targets for low latency, the smaller number of multiplications on the critical path offered by parallel multipliers seems attractive. However, it is not obvious which one is the most efficient solution: several slow parallel multipliers or one fast multiplier with large ν. This question is studied in Fig. 7(a) and (b). As shown in Fig. 7(a), parallel multipliers offer a large benefit on generic curves. Fig. 7(a) shows that with loose latency constraints, one multiplier with $\nu \leq 15$ should be used. If lower latency is needed, one should select two multipliers with $7 \leq \nu \leq 55$. If even they are too slow, then one should switch to four multipliers with $\nu \geq 33$. Fig. 7(b) shows that, for Koblitz curves, one should use one multiplier up to the point where $\nu = 82$. If even lower latency is needed, then one should use either two or three parallel multipliers. However, it will be shown in Section VI that even lower latency can be achieved with smaller area by using parallel FAPs with $\mu = 1$ and, therefore, one multiplier is the only feasible solution for Koblitz curves.

C. Parallel FAPs

Let φ be the number of parallel processors implemented as presented in Section IV each of which is computing different point multiplications independently of each other, i.e., $\hat{\varphi} = \varphi$.

One FAP has latency $L(\varphi = 1,\mu,\nu)$ and throughput $T(\varphi = 1,\mu,\nu)$. When parallel FAPs compute different point multiplications simultaneously, average latency remains the same, i.e., $L(\varphi > 1,\mu,\nu) = L(\varphi = 1,\mu,\nu)$, but the throughput increases, i.e., $T(\varphi > 1,\mu,\nu) = \varphi, T(\varphi = 1,\mu,\nu)$. In order to maximize

INPUT: k, P
OUTPUT: $Q = kP$
$(\kappa, e) \leftarrow \text{split}(k)$
parallel $j = 0$ to $\varphi - 1$ **do**
  $P_j \leftarrow 2^{e_j} P$ **or** $P_j \leftarrow \tau^{e_j} P$
  $Q_j \leftarrow \kappa_j P_j$
end parallel
$Q \leftarrow \sum_{j=0}^{\varphi-1} Q_j$

Fig. 8. Algorithm for computing kP with φ parallel FAPs.

throughput-area ratio $R_T(\varphi > 1, \mu, \nu)$ of a multi-FAP design, one should replicate FAPs with maximum $R_T(\varphi = 1, \mu, \nu)$, i.e., based on the analysis of Section V-B, one should use FAPs with parameters $\mu = 1$ and $\nu = 11$.

When (2) is computed on Koblitz curves, a converter is required as discussed in Section IV-C and it must be considered in throughput calculations. Instead of attaching a converter to each FAP, it is preferable to let one converter serve several FAPs because the conversion time is much shorter than the point multiplication time, i.e., $t_\tau < t_{kP}$. However, it should be guaranteed that the converter(s) do not become a bottleneck and the number of τNAF converters, ψ, must satisfy

$$\psi \geq \varphi \frac{T_{kP}}{T_\tau} = \varphi \frac{t_\tau}{t_{kP}} = \varphi \frac{L_\tau f_{\text{clk}_{kP}}}{L_{kP} f_{\text{clk}_\tau}} \tag{19}$$

where T_{kP}, T_τ, t_{kP}, t_τ, L_{kP}, L_τ, $f_{\text{clk}_{kP}}$ and f_{clk_τ} denote throughput, pm-time, latency, and clock frequency of the FAP(s) and the converter(s), respectively.

VI. Reducing Latency With Parallel Processors

This section presents how the latency of point multiplication can be reduced with parallel FAPs. In other words, parallel FAPs compute a single point multiplication, i.e. $\hat{\varphi} = 1$ but $\varphi > 1$. It is assumed that the FAPs can exchange data with each others.

In [43] Okeya *et al.* presented a method for reducing memory requirements of windowing methods on Koblitz curves by exploiting the inexpensiveness of the Frobenius maps. The same feature can be exploited for reducing the computation latency as will be shown in this section. The method is not restricted to Koblitz curves but the base point P needs to be fixed before the method operates efficiently on a generic curve because of precomputations involving P. The new method can be combined with other techniques such as parallel multipliers or windowing methods.

Obviously, (2) can be expressed in the following manner by using the binary expansion of k

$$kP = \sum_{i=0}^{\ell-1} k_i 2^i P = k_{\ell-1} 2^{\ell-1} P + \cdots + k_1 2P + k_0 P. \tag{20}$$

Assume that φ parallel FAPs are available. Then (20) can be divided for these FAPs as follows:

$$kP = \sum_{j=0}^{\varphi-1} \sum_{i=0}^{\ell-1} \kappa_{j,i} 2^i P \tag{21}$$

where $\kappa_{j,i} = 1$ for some $j = j'$ and $\kappa_{j,i} = 0$, for all $j \neq j'$, if $k_i = 1$, and $\kappa_{j,i} = 0$, for all j, if $k_i = 0$, i.e., terms $2^i P$ can be divided for the FAPs arbitrarily as long as each term is processed in one and only in one FAP. Obviously, there are $\varphi H(k)$ different ways to choose κ. Point multiplication defined by (21) is computed so that each FAP computes $Q_{\varphi'} = \kappa_{\varphi'} P = \sum_{i=0}^{\ell-1} \kappa_{\varphi',i} 2^i P$, where φ' is the index of that particular FAP and, in the end, the results, Q_j, are combined by computing $Q = \sum_{j=0}^{\varphi-1} Q_j = kP$.

The number of point doublings can be reduced if P is fixed. Let $i_{j,\min}$ and $i_{j,\max}$ be the smallest and the largest i for which $\kappa_{j,i} \neq 0$. Then, one can precompute $P_j = 2^{i_{j,\min}} P$, and compute (21) as follows:

$$kP = \sum_{j=0}^{\varphi-1} \sum_{i=i_{j,\min}}^{i_{j,\max}} \kappa_{j,i} 2^{i - i_{j,\min}} P_j. \tag{22}$$

The number of point doublings in the FAP φ' has now reduced by $i_{\varphi',\min}$.

In order to minimize the number of Montgomery point additions and doublings on the critical path, one should choose κ which minimizes $\max(i_{j,\max} - i_{j,\min})$. The problem is, however, that one would require *a priori* information about k in order to precompute $P_j = 2^{i_{j,\min}} P$. As this information is not available in practice, one must split k into φ words by using fixed values, i.e., $i_{j,\min}$ and $i_{j,\max}$ are fixed. This method is used in Section VI-A1.

On Koblitz curves zeros in κ lose their significance because Frobenius maps are almost free. Thus, it is assumed that the complexity is defined solely by the number of ones in κ, and one should find κ which minimizes $\max(H\kappa_j)$, i.e., the maximum number of nonzeros processed in any FAP. Precomputations are also almost free and they can be computed on-the-fly. Thus, base points need not to be fixed.

An algorithm for computing kP by using φ parallel FAPs is shown in Fig. 8. Derivation of κ from k is referred to as splitting. It is performed with one of the two splitting algorithms discussed in Section VI-A. Both splitting algorithms, $\text{split}(k)$, return κ and exponents e_j for computing base points $P_j = 2^{e_j} P$ or $P_j = \tau^{e_j} P$. The parallel computations can be performed independently of each other by using any point multiplication method, e.g., windowing methods can be used. Combining parallel computations, i.e., $Q = \sum_{j=0}^{\varphi-1} Q_j$, requires $\varphi - 1$ point additions, but the critical path consists of only $\lceil \log_2 \varphi \rceil$ point additions because parallelism can be utilized.

A. Splitting Algorithms

In order to achieve the best possible performance with parallel FAPs, the computational load must be divided for the FAPs as evenly as possible. The problem is, however, that the computational cost depends on k. Moreover, the way in which k determines the computational cost depends on various parameters such as the curve, the coordinate system, etc.

Because of the reasons mentioned before, finding a splitting algorithm resulting in the optimal splitting result every time proved to be a difficult task. Thus, two different splitting algorithms are suggested, both having advantages and disadvan-

TABLE IV
SPLITTING EXAMPLES ($\varphi = 4$)

	κ_j	P_j	$H(\kappa)$	ℓ
k	1000010010110111101101111011101011010010101100101101101000110010	P	35	64
Fixed	1101101000110010	P	8	16
window	1101001010110010	$2^{16}P$	8	16
($w = 16$)	1011011110111010	$2^{32}P$	11	16
	1000010010110111	$2^{48}P$	8	16
Cyclic	1000001000010000100000100000001000000010000001000000000010	P	9	56
	1000000010000100001000010000000100000001000000010000000010000	P	9	59
	1000000000010000100000100001000001000000010000001000000000100000	P	9	64
	1000010000010000100000100000001000000010000001000000000000	P	8	54

tages. The splitting algorithms considered in the following are called fixed window and cyclic splitting algorithms.

1) Fixed Window: The integer k is split into φ words using predefined windows with a size of w. A logical choice for w is $w = \lceil m/\varphi \rceil$. Now, κ is constructed so that κ_0 consists of the w least significant bits (LSBs) of k, κ_1 contains the next w bits, etc. The base points P_j can be precomputed because the window sizes are fixed, and they are given by

$$P_j = 2^{jw}P \quad P_j = \tau^{jw}P \tag{23}$$

for generic and Koblitz curves, respectively, i.e., $e_j = jw$. The longest precomputation requires $(\varphi - 1)w$ point doublings or Frobenius maps.

2) Cyclic: Starting either from the LSB or MSB of k, each nonzero bit of k is split cyclically so that the first nonzero is processed in the first FAP, the second nonzero in the second FAP, etc. The $(\varphi + 1)$th nonzero is again processed in the first FAP, the $(\varphi + 2)$th in the second FAP, etc. Each bit results in either a zero or nonzero bit in κ and, therefore, the length of the longest κ_j is ℓ. The base points are simply given by

$$P_j = P \tag{24}$$

i.e., $e_j = 0$, for all j and there are no precomputations. The cyclic splitting algorithm always results in the minimum number of nonzeros, i.e., $\max(H(\kappa_j)) = \lceil H(k)/\varphi \rceil$.

B. Examples and Comparison of the Splitting Algorithms

Splitting examples are given in Table IV with four FAPs ($\varphi = 4$). The fixed window splitting results in the shortest κ_j but the number of nonzeros is suboptimal. The critical path of computation of a Montgomery point multiplication is 15 Montgomery point additions and doublings and $\lceil \log_2 4 \rceil = 2$ point additions with $\varphi = 4$ instead of 63 point additions and doublings if $\varphi = 1$. The longest precomputation requires 48 point doublings. The cyclic splitting results in κ with minimal $\max(H(\kappa_j))$, i.e., only 9 instead of the original 35. Thus, the critical path for Koblitz curves consists of only $9 - 1 + \lceil \log_2 4 \rceil = 10$ point additions instead of 34 with 1 FAP.

Performance of the splitting algorithms was tested by selecting 10 000 random 163-bit integers and evaluating them for both Koblitz and generic curves. When Koblitz curves were considered, the 163-bit k was first converted to τNAF. Frobenius maps were ignored. The Montgomery point multiplication was used for generic curves and the computational cost of precomputations required in the fixed window algorithm was neglected because a fixed P was assumed. Both one and zero bits have the same cost and, therefore, only the length of κ_j has significance. The results are presented in Fig. 9 which depicts speedups versus the one FAP case. Fig. 9 shows that the cyclic splitting results in the best speedups for Koblitz curves. On the other hand, the fixed window splitting algorithm is, expectedly, the only one performing well for generic curves. Notice that, although the speedups versus the one FAP case are smaller for Koblitz curves than for generic curves, the actual point multiplication is still considerably faster on Koblitz curves. Furthermore, Koblitz curves do not require any precomputations or they are very cheap.

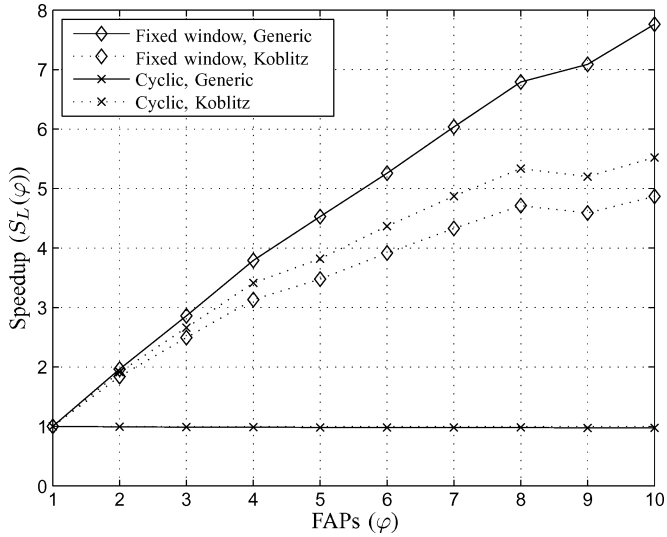

Fig. 9. Expected speedups of the splittings for generic and Koblitz curves.

C. Multiple FAPs Versus Multiple Multipliers

This section presents comparisons of implementations having multiple FAPs, i.e., $\varphi > 1$, and implementations having a single FAP with multiple multipliers, i.e., $\varphi = 1$ and $\mu > 1$.

As presented in Section V-B, $\nu = 11$ optimizes latency-area ratio R_L of an FAP for both generic and Koblitz curves. In that case, (18) gives the area of 3084 ALMs for the FAP of which the multiplier occupies 2354 ALMs (76.3%). This percentage is in line with other designs reported in the literature, see, e.g., [9], [13], and [14]. Let A denote the area of an FAP with $\mu = 1$. Because the area of a multiplier is 0.763 A, FAPs with $\mu = 2$,

TABLE V
COMPARISON OF MULTIPLE FAP AND MULTIPLE MULTIPLIER METHODS ON $K_1(\mathbb{F}_{2^{163}})$

φ	μ	Multiplications in $\mathrm{F}_{2^{163}}$	A	S_L	S_L/A
1	1	$8 \cdot (163/3 - 1)$ $+11 = 438$	1.000	1.000	1.000
1	2	$5 \cdot (163/3 - 1)$ $+10 = 277$	1.763	1.581	0.897
1	3	$4 \cdot (163/3 - 1)$ $+10 = 223$	2.526	1.964	0.778
2	1	$8 \cdot \lceil 163/(3 \cdot 2) - 1 \rceil + 11 \cdot \lceil \log_2 2 \rceil + 11 = 238$	2.000	1.840	0.920
3	1	$8 \cdot \lceil 163/(3 \cdot 3) - 1 \rceil + 11 \cdot \lceil \log_2 3 \rceil + 11 = 177$	3.000	2.475	0.825
4	1	$8 \cdot \lceil 163/(3 \cdot 4) - 1 \rceil + 11 \cdot \lceil \log_2 4 \rceil + 11 = 137$	4.000	3.197	0.799
5	1	$8 \cdot \lceil 163/(3 \cdot 5) - 1 \rceil + 11 \cdot \lceil \log_2 5 \rceil + 11 = 124$	5.000	3.532	0.706
6	1	$8 \cdot \lceil 163/(3 \cdot 6) - 1 \rceil + 11 \cdot \lceil \log_2 6 \rceil + 11 = 116$	6.000	3.776	0.629

TABLE VI
COMPARISON OF MULTIPLE FAP AND MULTIPLE MULTIPLIER METHODS ON $E(\mathbb{F}_{2^{163}})$

φ	μ	Multiplications in $\mathrm{F}_{2^{163}}$	Precomp.	A	S_L	S_L/A
1	1	$6 \cdot (163 - 1)$ $+19 = 991$	0	1.000	1.000	1.000
1	2	$3 \cdot (163 - 1)$ $+15 = 501$	0	1.763	1.978	1.122
1	4	$2 \cdot (163 - 1)$ $+13 = 337$	0	3.289	2.941	0.894
2	1	$6 \cdot \lceil 163/2 - 1 \rceil + 11 \cdot \lceil \log_2 2 \rceil + 19 = 516$	421	2.000	1.921	0.960
3	1	$6 \cdot \lceil 163/3 - 1 \rceil + 11 \cdot \lceil \log_2 3 \rceil + 19 = 365$	561	3.000	2.715	0.905
4	1	$6 \cdot \lceil 163/4 - 1 \rceil + 11 \cdot \lceil \log_2 4 \rceil + 19 = 281$	626	4.000	3.527	0.882
5	1	$6 \cdot \lceil 163/5 - 1 \rceil + 11 \cdot \lceil \log_2 5 \rceil + 19 = 244$	671	5.000	4.061	0.812
6	1	$6 \cdot \lceil 163/6 - 1 \rceil + 11 \cdot \lceil \log_2 6 \rceil + 19 = 214$	711	6.000	4.631	0.772

$\mu = 3$, and $\mu = 4$ have areas 1.763 A, 2.526 A, and 3.289 A, respectively. On the other hand, implementations having φ FAPs with $\mu = 1$ have the areas of φA.

Only the number of multiplications on the critical path is considered in the following comparison in order to keep comparison simple. This does not skew the results considerably because multiplications dominate in the overall cost. Koblitz curves are considered first with the cyclic splitting algorithm. A point addition is required when $k_i = \pm 1$ and $H(k) = m/3$, see Section II. A point addition has a critical path of 8, 5, or 4 multiplications with $\mu = 1$, $\mu = 2$, or $\mu = 3$, respectively, see Section III-B. Combination of Q_j is computed with point additions in $\mathcal{A}$ requiring 11 multiplications (including Itoh-Tsujii inversion). The $\mathcal{LD} \mapsto \mathcal{A}$ mapping also requires 11 multiplications (including Itoh-Tsujii inversion), but the critical path reduces to 10 multiplications if $\mu > 1$. Based on the previously mentioned facts, estimates of area, speedup, and speedup per area ratio were derived as presented in Table V.

The fixed window splitting was selected for generic curves because it was the only one of the two algorithms that offers significant speedups, see Section VI-B. The critical path consists of 6, 3, or 2 multiplications when $\mu = 1$, $\mu = 2$, or $\mu = 4$, respectively, see Section III-B. The $\mathcal{P} \mapsto \mathcal{A}$ mapping and the recovery of the y-coordinate have the critical path of 19, 15, or 13 multiplications (including Itoh-Tsujii inversion) when $\mu = 1$, $\mu = 2$, or $\mu = 4$, respectively. Combining Q_j is performed similarly as for Koblitz curves. The precomputations require multiplications on generic curves, because one needs to compute $2^{jw}P$. Because point doubling in $\mathcal{A}$ is expensive, it is faster to perform doublings in $\mathcal{LD}$ and then map the result point to $\mathcal{A}$. Thus, the longest precomputation requires $5(\varphi - 1)\lceil m/\varphi \rceil + 11$ multiplications (including Itoh-Tsujii inversion). Estimates for area, speedup, and speedup per area ratio are presented in Table VI.

Tables V and VI show that the multiple FAP method can be efficiently used for speeding up computation on Koblitz curves and, if P is fixed, also on generic curves. The method allows speedups beyond the limitations of the multiple multiplier methods and, moreover, even outperforms multiple multiplier methods in achieved speedup per area on Koblitz curves.

VII. IMPLEMENTATIONS

Several designs were implemented on an FPGA with different parameters $\mathbf{p}$ in order to investigate the validity of the analysis and methods presented in Sections V and VI. The designs were written in VHDL and synthesized for Altera Stratix II EP2S180F1020C3 FPGA, henceforth referred to as S180C3, by using Altera Quartus II 6.0 SP1 design software. Functionality of the designs was verified with ModelSim SE 6.1b. Stratix II S180C3 has 71 760 ALMs, 930 M512s, and 768 M4Ks [41]. Modular design style was used in VHDL and field multipliers were generated with automated designs tools written specifically for this purpose. Hence, implementing multiple designs could be done with moderate amount of work, but all designs required some hand optimization.

Parameters $\mu = 1$ and $\nu = 11$ were selected for the FAPs in parallel FAP implementations presented in Sections V-C and VI because they offer the best latency-area ratio R_L based on the analysis in Section V-B. The performance of the method presented in Section VI was demonstrated only on Koblitz curves because the base point would need to be fixed on generic curves. The cyclic splitting was used, because it performs better than the fixed window method as shown in Section VI-B.

A. Results

The results are shown in Tables VII and VIII for generic and Koblitz curves, respectively. The parameters of the designs are given on the left. The number of τNAF converters is always $\psi = 1$ in Table VIII. The number of point multiplications that can be computed simultaneously is denoted with $\hat{\varphi}$ in Table VIII. The results obtained from Quartus II are given in the middle so that the area of the design is given in the ALMs column followed by the maximum clock frequencies for the τNAF converter, clk_{tau}, and for the FAP, clk_{kP}. Multiplication latency, M, is given by (5) and the average point multiplication

TABLE VII
RESULTS ON STRATIX II S180C3 ON GENERIC CURVE $E(\mathbb{F}_{2^{163}})$

φ	μ	ν	ALMs	clk_{kP}	M	L_{kP}	t	T
1	1	8	2659	184.37	23	27591	149.65	6682
1	1	9	2837	181.26	21	25609	141.28	7078
1	1	10	3010	184.81	19	23627	127.84	7822
1	1	11	3222	181.03	17	21645	119.57	8364
1	1	12	3325	183.76	16	20654	112.40	8897
1	1	13	3455	180.15	15	19663	109.15	9162
1	1	14	3603	182.98	14	18672	102.04	9800
1	1	15	4269	174.28	13	17681	101.45	9857
1	1	17	4597	173.10	12	16690	96.42	10371
1	1	19	4902	166.44	11	15699	94.32	10602
1	1	21	5250	170.50	10	14708	86.26	11592
1	1	24	5808	169.18	9	13717	81.08	12334
1	1	28	6291	172.35	8	12726	73.84	13543
1	1	33	7113	162.55	7	11735	72.19	13852
1	1	41	8440	158.08	6	10744	67.97	14713
1	2	8	4344	183.69	23	15996	87.08	11483
1	2	9	4665	183.18	21	14994	81.85	12217
1	2	10	5035	185.56	19	13992	75.40	13262
1	2	11	5267	185.05	17	12990	70.20	14246
1	2	12	5619	179.31	16	12489	69.65	14357
1	2	13	5915	181.92	15	11988	65.90	15175
1	2	14	6080	181.32	14	11487	63.35	15785
1	2	15	7461	171.76	13	10986	63.96	15634
1	2	17	8114	171.59	12	10485	61.10	16365
1	2	19	8788	174.46	11	9984	57.23	17474
1	2	21	10468	161.58	10	9483	58.69	17039
1	2	24	12005	155.84	9	8982	57.64	17350
1	2	28	13678	145.71	8	8481	58.20	17181
1	2	33	15740	153.14	7	7980	52.11	19190
1	2	41	18489	144.74	6	7479	51.67	19353
1	4	8	7400	171.32	23	12060	70.39	14206
1	4	9	8212	180.21	21	11386	63.18	15827
1	4	10	8830	181.82	19	10712	58.92	16973
1	4	11	9325	183.28	17	10038	54.77	18259
1	4	12	10388	174.86	16	9701	55.48	18025
1	4	13	11177	186.95	15	9364	50.09	19965
1	4	14	11800	184.67	14	9027	**48.88**	20458
1	4	15	15594	170.01	13	8690	51.11	19564
1	4	17	17236	150.90	12	8353	55.35	18065
1	4	19	18782	149.25	11	8016	53.71	18619
1	4	21	20219	149.12	10	7679	51.50	19419
1	4	24	22672	142.80	9	7342	51.41	19450
2	1	11	6376	184.26	17	21645	117.47	17026
4	1	11	13807	175.72	17	21645	123.18	32473
6	1	11	21154	179.31	17	21645	120.71	49705
				MHz			μs	ops

latency, L_{kP}, is computed as in Table II. τNAF conversions require $3m + 2 = 491$ clock cycles on average [42] and the latency was omitted from Table VIII. The computation times in microseconds are computed by using the maximum clock frequencies. The average pm-time is denoted by t_{kP} and the average τNAF conversion time by t_τ. The end-to-end time is simply $t = t_{kP} + t_\tau$. For generic curves, it only consists of t_{kP} and Table VII, therefore, includes only t. Throughput, T, is defined solely by t_{kP} also for Koblitz curves because k for the next point multiplication can be converted simultaneously while the FAP computes a point multiplication.

The results presented in Tables VII and VIII were obtained by synthesizing each design once. Different constraints were used for generic and Koblitz curves. However, the same constraints were used for all designs with the same curve. Fig. 10 plots the pm-times of the designs with $\hat{\varphi} = 1$ presented in Tables VII and VIII as functions of areas.

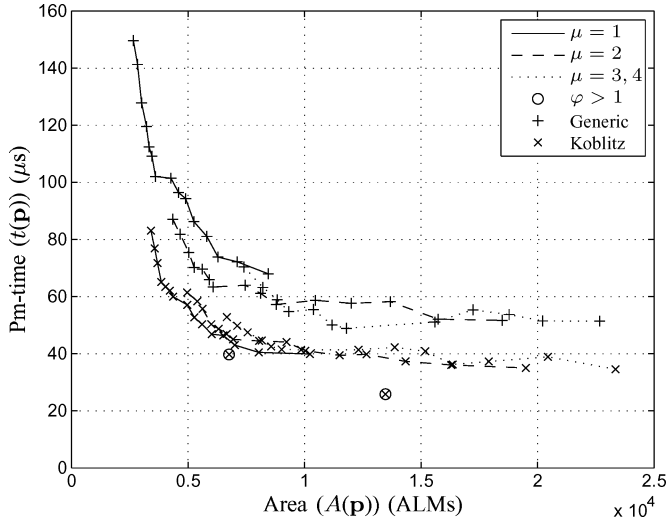

Fig. 10. Results of the implementations on Stratix II EP2S180F1020C3 FPGA presented in Tables VII and VIII. The plot includes designs for which $\hat{\varphi} = 1$, i.e., they compute a single point multiplication at a time.

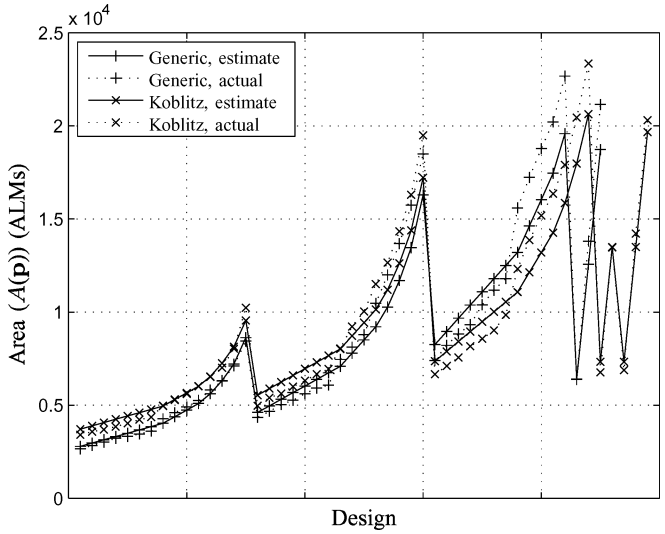

Fig. 11. Comparison of estimated and actual areas of the implementations presented in Tables VII and VIII. The designs follow the same order as in the tables and a design that is on the top in the table is on the left-hand side of this figure.

B. Discussion and Comparisons

Fig. 10 shows that $\mu = 1$ is always the best choice for Koblitz curves as estimated in Section V-B. For generic curves, however, multiple multipliers are feasible in practice, too. Actually, multiple multipliers perform even better than expected, because when ν grows, clock frequency decreases thus resulting in slower performance. This favors the use of multiple multipliers because multiple multipliers with small ν operate on a higher clock frequency than one with large ν.

The superiority of Koblitz curves is obvious in Fig. 10. Although k needs to be converted to τNAF, they are clearly faster and more area efficient than generic curves. Pm-time on Koblitz curves can be reduced with minor additional area to approximately 40 μs. Further reductions in pm-time result in considerable increase in area with traditional parallelization methods.

TABLE VIII
RESULTS ON STRATIX II S180C3 ON KOBLITZ CURVE $K_1(\mathbb{F}_{2^{163}})$. FOR ALL DESIGNS $\psi = 1$

φ	μ	ν	$\hat{\varphi}$	ALMs	clk_τ	clk_{kP}	M	L_{kP}	t_{kP}	t_τ	t	T
1	1	8	1	3411	88.94	163.03	23	12644	77.56	5.52	83.08	12894
1	1	9	1	3570	88.21	165.07	21	11769	71.30	5.57	76.86	14026
1	1	10	1	3690	87.78	164.69	19	10893	66.14	5.59	71.74	15118
1	1	11	1	3852	87.98	168.18	17	10018	59.57	5.58	65.15	16788
1	1	12	1	4024	88.80	165.45	16	9580	57.90	5.53	63.43	17270
1	1	13	1	4226	88.25	161.84	15	9143	56.49	5.56	62.06	17702
1	1	14	1	4358	89.33	159.80	14	8705	54.47	5.50	59.97	18357
1	1	15	1	4972	89.60	160.18	13	8267	51.61	5.48	57.09	19375
1	1	17	1	5274	88.46	165.92	12	7830	47.19	5.55	52.74	21191
1	1	19	1	5602	89.87	164.96	11	7392	44.81	5.46	50.27	22316
1	1	21	1	6019	88.75	168.35	10	6954	41.31	5.53	46.84	24208
1	1	24	1	6521	89.53	159.41	9	6517	40.88	5.48	46.36	24462
1	1	28	1	7011	89.27	162.15	8	6079	37.49	5.50	42.99	26674
1	1	33	1	8033	87.84	161.79	7	5641	34.87	5.59	40.46	28679
1	1	41	1	10234	89.38	151.24	6	5204	34.41	5.49	39.90	29064
1	2	8	1	4967	88.07	162.95	23	9101	55.85	5.58	61.43	17905
1	2	9	1	5400	89.02	161.66	21	8548	52.87	5.52	58.39	18913
1	2	10	1	5622	88.64	159.31	19	7994	50.18	5.54	55.72	19928
1	2	11	1	6002	88.11	166.20	17	7441	44.77	5.57	50.34	22336
1	2	12	1	6316	88.27	165.92	16	7164	43.18	5.56	48.74	23159
1	2	13	1	6662	90.02	166.09	15	6888	41.47	5.45	46.92	24114
1	2	14	1	6937	89.16	167.20	14	6611	39.54	5.51	45.05	25291
1	2	15	1	8053	89.57	162.52	13	6334	38.98	5.48	44.46	25657
1	2	17	1	9231	87.44	157.36	12	6058	38.50	5.62	44.11	25977
1	2	19	1	10032	88.71	162.81	11	5781	35.51	5.53	41.04	28163
1	2	21	1	11503	87.46	162.68	10	5504	33.84	5.61	39.45	29555
1	2	24	1	12661	88.06	152.77	9	5228	34.22	5.58	39.79	29223
1	2	28	1	14338	84.11	157.36	8	4951	31.46	5.84	37.30	31783
1	2	33	1	16301	87.25	153.66	7	4674	30.42	5.63	36.05	32873
1	2	41	1	19498	89.53	148.85	6	4398	29.54	5.48	35.03	33847
1	3	8	1	6667	89.67	165.26	23	7821	47.33	5.48	52.80	21130
1	3	9	1	7112	89.57	166.64	21	7374	44.25	5.48	49.73	22597
1	3	10	1	7562	87.97	165.26	19	6928	41.92	5.58	47.50	23855
1	3	11	1	8168	88.12	165.98	17	6481	39.05	5.57	44.62	25610
1	3	12	1	8573	88.46	169.12	16	6258	37.00	5.55	42.55	27026
1	3	13	1	9019	89.39	167.64	15	6034	36.00	5.49	41.49	27781
1	3	14	1	9852	87.63	162.58	14	5811	35.74	5.60	41.35	27978
1	3	15	1	12315	83.76	157.18	13	5588	35.55	5.86	41.41	28130
1	3	17	1	13870	89.84	145.88	12	5364	36.77	5.47	42.24	27194
1	3	19	1	15181	89.39	145.33	11	5141	35.37	5.49	40.87	28269
1	3	21	1	16358	88.61	160.26	10	4918	30.69	5.54	36.23	32589
1	3	24	1	17909	87.80	147.93	9	4694	31.73	5.59	37.33	31512
1	3	28	1	20443	89.38	133.83	8	4471	33.41	5.49	38.90	29933
1	3	33	1	23346	87.37	146.71	7	4248	28.95	5.62	34.57	34539
2	1	11	1	6763	87.47	159.95	17	5457	34.12	5.61	39.73	29311
4	1	11	1	13472	88.68	155.50	17	3153	20.28	5.54	**25.81**	49318
2	1	11	2	6884	88.14	160.26	17	10018	62.51	5.57	68.08	31994
4	1	11	4	14215	88.05	161.08	17	10018	62.19	5.58	67.77	64316
6	1	11	6	20308	87.80	159.72	17	10018	62.72	5.59	68.31	95660
					MHz	MHz			μs	μs	μs	ops

However, the new method presented in Section VI offers faster pm-times with smaller area (circled points in Fig. 10).

The synthesization results vary slightly from run to run which is one reason for the variation of maximum clock frequencies in Tables VII and VIII. On the other hand, when the size of the design grows, maximum clock frequencies start to decrease dramatically because the place&route becomes harder. Hence, the assumptions that the area grows linearly and the clock frequencies are constant are not valid for large designs as was conjectured in Section V. However, the assumptions hold well for smaller designs.

The size of the multiplier, ν, has a considerably larger effect on clock frequencies than μ or φ which is not surprising considering that the critical path is in the multiplier. Differences of estimated and actual areas are investigated in Fig. 11 which shows that estimates hold well if $\mu = 1$. However, when several multipliers are used, i.e. $\mu > 1$, area estimates are too optimistic with large ν. Again, this was expected because the place&route becomes hard when the size of an FAP grows.

A large number of FPGA-based implementations have been published in the literature. Fair comparison of these implementations is difficult—if not impossible—because of the variety of different FPGAs, elliptic curves, fields, coordinate systems, etc. Arguably, the largest problem for fair comparison is the variety of FPGAs, because it is hard to map area requirements and timings between different FPGA architectures without synthesizing the design for all them. A valuable effort for evaluating designs on different families of Xilinx FPGAs was made in [11], where estimates of the effect of the FPGA families were given by synthesizing the designs for different families. However, as

TABLE IX
PUBLISHED FPGA-BASED IMPLEMENTATIONS

Ref.	Year	Device	Curve	Pm-time (μs)
[7]	2006	Virtex-II	$E(\mathrm{F}_{2^{163}})$	41
[8]	2004	Virtex-II	$E(\mathrm{F}_{2^{163}})$	201
[9]	2002	Virtex	$E(\mathrm{F}_{2^{163}})$	270
[10]	2006	Virtex-E	$E(\mathrm{F}_{2^{163}})$	804
[11]	2005	Virtex-II	$E(\mathrm{F}_{2^{162}})$	100[a]
[12]	2006	Virtex-II	$K_1(\mathrm{F}_{2^{163}})$	36
[13]–[15]	2002	Virtex-II	$E(\mathrm{F}_{2^{163}})$	143
[16]	2004	Virtex-II	$E(\mathrm{F}_{2^{163}})$	106
[18]	2002	Virtex	$E(\mathrm{F}_{2^{155}})$	8300
[19]	2004	Virtex-E	$E(\mathrm{F}_{2^{163}})$	233
[19]	2004	Virtex-E	$K_1(\mathrm{F}_{2^{163}})$	75
[20]	2000	Flex 10K	$E(\mathrm{F}_{2^{163}})$	80300
[20]	2000	Flex 10K	$K_1(\mathrm{F}_{2^{163}})$	45600
[21]	2000	Virtex-E	$E(\mathrm{F}_{2^{167}})$	210
[22]	2004	Virtex-E	$E(\mathrm{F}_{2^{191}})$	63
[23]	2005	Virtex-E	$E(\mathrm{F}_{2^{163}})$	48
This work	2007	Stratix II	$E(\mathrm{F}_{2^{163}})$	49
This work	2007	Stratix II	$K_1(\mathrm{F}_{2^{163}})$	26

the VHDL describing the architecture of Section IV was written specifically for Stratix II FPGAs, this approach could not be used.

Table IX summaries FPGA implementations presented in the literature. When a publication presents many implementations, Table IX presents the one which is the most comparable with the designs presented in this article. The implementations presented in this paper are clearly among the fastest ones. However, as mentioned before, it is impossible to say which portion of the differences is caused by the different implementation platform.

FPGA implementations of Koblitz curves have been presented in [12], [17], [19], and [20]. The fastest implementation in this paper computes point multiplication in 25.81 μs including the conversion and it outperforms all previous implementations. The designs presented in [19] and [20] do not include a converter. The significant difference in their pm-times is caused by different FPGAs and design architectures. Fast performance presented in [12] was achieved by representing k with a double-base expansion. The implementation in [17] computes a multiple point multiplication and targets for maximum throughput which makes it incomparable with other implementations.

VIII. CONCLUSION

Parallelization of high-speed ECC accelerators was studied. A generic accelerator architecture was presented in Section IV and it was used in studying the effects of parallelization. The analysis concerned both generic and Koblitz curves.

Analytic tools were provided for estimating efficiency of different parallelism parameters. The accuracy of the tools was studied by implementing several designs on Stratix II S180C3. These implementations are among the fastest ones published in the literature. The tools were shown to provide accurate estimates although accuracy decreases when designs become large because the place&route is harder. Anyhow, the tools provide valuable information on how and where parallelism should be used in ECC implementations.

When parallel multipliers in an FAP are used for reducing latency, the optimal setup depends on the curve. Only one multiplier should be always used for Koblitz curves, but multiple multipliers offer considerable improvements for generic curves. For them, the optimal setup depends on various aspects, such as available area and pm-time constraints, as discussed in Section V-B.

Koblitz curves were shown to offer considerably faster point multiplication than generic curves with equal amount of area even when the τNAF converter was included. Furthermore, the new method utilizing parallel FAPs presented in Section VI can be used efficiently for Koblitz curves. If base points are fixed or changed infrequently, the method is useful also on generic curves but precomputations prevent its use if base point flexibility is essential. The method can be combined with existing techniques such as windowing methods. The implementations of the method have very high latency-area efficiencies which prove the usability of the new method.

REFERENCES

[1] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, pp. 203–209, 1987.

[2] V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology (CRYPTO)*, ser. Lecture Notes in Computer Science. New York: Springer, 1986, vol. 218, pp. 417–426.

[3] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$," *IEEE J. Sel. Areas Commun.*, vol. 11, no. 5, pp. 804–813, Jun. 1993.

[4] J. Goodman and A. Chandrakasan, "An energy effcient reconfigurable public-key cryptography processor architecture," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 2000, vol. 1965, pp. 175–190.

[5] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, Apr. 2003.

[6] F. Sozzani, G. Bertoni, S. Turcato, and L. Breveglieri, "A parallelized design for an elliptic curve cryptosystem coprocessor," in *Proc. Int. Conf. Inf. Technol.: Coding Comput. (ITCC)*, Las Vegas, NV, Apr. 2005, vol. 1, pp. 626–630.

[7] B. Ansari and M. A. Hasan, "High performance architecture of elliptic curve scalar multiplication," Centre for Applied Cryptographic Research, Univ. Waterloo, Waterloo, ON, Canada, Tech. Rep. CACR 2006-1, 2006.

[8] S. Bajracharya, C. Shu, K. Gaj, and T. El-Ghazawi, "Implementation of Elliptic Curve Cryptosystems Over $GF(2^n)$ in Optimal Normal Basis on a Reconfigurable Computer," in *Proc. Int. Conf. Field Programmable Logic and Application (FPL)*, ser. Lecture Notes in Computer Science. New York: Springer, 2004, vol. 3203, pp. 1001–1005.

[9] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich, "Reconfigurable implementation of elliptic curve crypto algorithms," in *Proc. Int. Parallel Distrib. Process. Symp., (IPDPS, RAW)*, Ft. Lauderdale, FL, Apr. 2002, pp. 157–164.

[10] M. Benaissa and W. M. Lim, "Design of flexible $GF(2^m)$ elliptic curve cryptography processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 6, pp. 659–662, Jun. 2006.

[11] R. C. C. Cheung, N. J. Telle, W. Luk, and P. Y. K. Cheung, "Customizable elliptic curve cryptosystem," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 9, pp. 1048–1059, Sep. 2005.

[12] V. S. Dimitrov, K. U. Järvinen, M. J. Jacobson, W. F. Chan, and Z. Huang, "FPGA implementation of point multiplication on Koblitz curves using Kleinian integers," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 2006, vol. 4249, pp. 445–459.

[13] H. Eberle, N. Gura, and S. Chang-Shantz, "A cryptographic processor for arbitrary elliptic curves over $GF(2^m)$," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Arch., Process., (ASAP)*, The Hague, The Netherlands, Jun. 2003, pp. 444–454.

[14] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, "An end-to-end systems approach to elliptic curve cryptography," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 2002, vol. 2523, pp. 349–365.

[15] N. Gura, H. Eberle, and S. C. Shantz, "Generic implementations of elliptic curve cryptography using partial reduction," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, Washington, DC, Nov. 2002, vol. 1, pp. 108–116.

[16] K. Järvinen, M. Tommiska, and J. Skyttä, "A scalable architecture for elliptic curve point multiplication," in *Proc. IEEE Int. Conf. Field-Program. Technol. (FPT)*, Brisbane, Australia, Dec. 2004, pp. 303–306.

[17] K. Järvinen, J. Forsten, and J. Skyttä, "FPGA design of self-certified signature verification on Koblitz curves," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 2007, vol. 4727, pp. 256–271.

[18] P. H. W. Leong and K. H. Leung, "A microcoded elliptic curve processor using FPGA Technology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 5, pp. 550–559, Oct. 2002.

[19] J. Lutz and A. Hasan, "High performance FPGA based elliptic curve cryptographic co-processor," in *Proc. Int. Conf. Inf. Technol.: Coding Comput. (ITCC)*, Las Vegas, NV, Apr. 2004, vol. 2, pp. 486–492.

[20] S. Okada, N. Torii, K. Itoh, and M. Takenaka, "Implementation of elliptic curve cryptographic coprocessor over $GF(2^m)$ on an FPGA," in *Cryptographic Hardware and Embedded Systems, CHES 2000*, ser. Lecture Notes in Computer Science. New York: Springer, 2000, vol. 1965, pp. 25–40.

[21] G. Orlando and C. Paar, "A high-performance reconfigurable elliptic curve processor for $GF(2^m)$," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 2000, vol. 1965, pp. 41–56.

[22] F. Rodríguez-Henríquez, N. A. Saqib, and A. Díaz-Pérez, "A fast parallel implementation of elliptic curve point multiplication over $GF(2^m)$," *Microprocess. Microsyst.*, vol. 28, no. 5–6, pp. 329–339, Aug. 2004.

[23] C. Shu, K. Gaj, and T. El-Ghazawi, "Low latency elliptic curve cryptography accelerators for NIST curves over binary fields," in *Proc. IEEE Int. Conf. Field-Program. Technol. (FPT)*, Singapore, Dec. 2005, pp. 309–310.

[24] G. Meurice de Dormale and J.-J. Quisquater, "High-speed hardware implementations of elliptic curve cryptography: A survey," *J. Syst. Architect.*, vol. 53, no. 2–3, pp. 72–84, Feb./Mar. 2007.

[25] G. Orlando and C. Paar, "A super-serial Galois fields multiplier for FPGAs and its application to public-key algorithms," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*, Napa Valley, CA, Apr. 1999, pp. 232–239.

[26] L. Song and K. K. Parhi, "Low-energy digit-serial/parallel finite field multipliers," *J. VLSI Signal Process.*, vol. 19, no. 2, pp. 149–166, Jul. 1998.

[27] N. Koblitz, "CM-curves with good cryptographic properties," in *Adv. Cryptology (CRYPTO)*, ser. Lecture Notes in Computer Science. New York: Springer, 1991, vol. 576, pp. 279–287.

[28] *Digital Signature Standard (DSS)*, FIPS PUB 186-2, Federal Information Processing Standard, National Institute of Standards and Technology (NIST), Computer Security, Jan. 27, 2000.

[29] *SEC 2: Recommended Elliptic Curve Domain Parameters*, , Standards for Efficient Cryptography, Certicom Research Std., Sep. 20, 2000 [Online]. Available: http://www.secg.org/download/aid-386/sec2_final.pdf

[30] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 2000, vol. 1965, pp. 1–24.

[31] C. Doche and T. Lange, "Arithmetic of elliptic curves," in *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, H. Cohen and G. Frey, Eds. Boca Raton, FL: Chapman & Hall/CRC, 2006, ch. 13, pp. 267–302.

[32] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. New York: Springer, 2004.

[33] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases," *Inform. Comput.*, vol. 78, no. 3, pp. 171–177, Sep. 1988.

[34] J. López and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 1999, vol. 1717, pp. 316–317.

[35] A. Higuchi and N. Takagi, "A fast addition algorithm for elliptic curve arithmetic in $GF(2^n)$ using projective coordinates," *Inf. Process. Lett.*, vol. 76, no. 3, pp. 101–103, Dec. 15, 2000.

[36] T. Lange, "A note on López-Dahab coordinates," Cryptology ePrint Archive, Tech. Rep. 2004/323, 2004 [Online]. Available: http://eprint.iacr.org/

[37] E. Al-Daoud, R. Mahmod, M. Rushdan, and A. Kilicman, "A new addition formula for elliptic curves over $GF(2^n)$," *IEEE Trans. Comput.*, vol. 51, no. 8, pp. 972–975, Aug. 2002.

[38] J. A. Solinas, "Efficient arithmetic on Koblitz curves," *Des. Codes Cryptography*, vol. 19, no. 2–3, pp. 195–249, 2000.

[39] C. C. Wang, T. K. Troung, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in $GF(2^m)$," *IEEE Trans. Comput.*, vol. 34, no. 8, pp. 709–717, Aug. 1985.

[40] T. Izu and T. Takagi, "Fast elliptic curve multiplications with SIMD operations," in *Proc. Int. Conf. Inf. Commun. Sec. (ICICS)*, ser. Lecture Notes in Computer Science. New York: Springer, 2002, vol. 2513, pp. 217–230.

[41] *"Stratix II Device Handbook"* ver. 4.1, Altera, San Jose, CA, Apr. 2006 [Online]. Available: http://www.altera.com/literature/hb/stx2/stratix2_handbook.pdf, vol. 1–2

[42] K. Järvinen, J. Forsten, and J. Skyttä, "Efficient circuitry for computing τ-adic non-adjacent form," in *Proc. IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nice, France, Dec. 2006, pp. 232–235.

[43] K. Okeya, T. Takagi, and C. Vuillaume, "Short memory scalar multiplication on Koblitz curves," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science. New York: Springer, 2005, vol. 3659, pp. 91–105.

Kimmo Järvinen (S'06) was born in Turku, Finland, in 1979. He received the M.Sc. (Tech.) degree in electrical engineering from the Helsinki University of Technology (TKK), Espoo, Finland, in 2003, where he is currently pursuing the D.Sc. (Tech.) degree in electrical engineering.

He has been with the Signal Processing Laboratory, TKK, since 2002 and in the Graduate School in Electronics, Telecommunications, and Automation (GETA) since 2004. His research interests include hardware realization of cryptographic algorithms, secret-key and public-key cryptographic algorithms, especially elliptic curve cryptography, and FPGAs.

Jorma Skyttä was born in Helsinki, Finland, in 1957. He received the M.Sc. (Tech.), Lic. Tech., and D.Sc. (Tech.) degrees from Helsinki University of Technology (TKK), Espoo, Finland, in 1981, 1984, and 1985, respectively.

He has been with the Signal Processing Laboratory, TKK, since 1989 as an Associate Professor and Professor since 1998. His research interests include realization of digital computation, especially digital signal processing applications and implementation of cryptographic systems utilizing FPGA, and FPGA-based computational architectures.