INTEGRATION, the VLSI journal 44 (2011) 270–279

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Optimized FPGA-based elliptic curve cryptography processor for high-speed applications

Kimmo Järvinen [1]

Aalto University, School of Science and Technology, Department of Information and Computer Science, P.O. Box 15400, FIN-00076 Aalto, Finland

ARTICLE INFO

Available online 1 September 2010

Keywords:
Elliptic curve cryptography
Field-programmable gate arrays
Koblitz curve
Parallelism

ABSTRACT

In this paper, we introduce an FPGA-based processor for elliptic curve cryptography on Koblitz curves. The processor targets specifically to applications requiring very high speed. The processor is optimized for performing scalar multiplications, which are the basic operations of every elliptic curve cryptosystem, only on one specific Koblitz curve; the support for other curves is achieved by reconfiguring the FPGA. We combine efficient methods from various recent papers into a very efficient processor architecture. The processor includes carefully designed processing units dedicated for different parts of the scalar multiplication in order to increase performance. The computation is pipelined providing simultaneous processing of up to three scalar multiplications. We provide experimental results on an Altera Stratix II FPGA demonstrating that the processor computes a single scalar multiplication on average in 11.71 μs and achieves a throughput of 235,550 scalar multiplications per second on NIST K-163.

1. Introduction

Neal Koblitz and Victor Miller independently proposed the use of elliptic curves for public-key cryptography in 1985 [1,2]. Since then, elliptic curve cryptography (ECC) has been intensively studied because it offers both shorter keys [3] and faster performance [4,5] compared to more traditional public-key cryptosystems, such as RSA [6]. Hardware implementation of ECC has also gained considerable interest and, as a consequence, many descriptions of hardware implementations exist in the literature; see [7] for a comprehensive review.

Field-programmable gate arrays (FPGAs) have proven to be highly feasible platforms for implementing cryptographic algorithms because of the combination of programmability and high speed. Several advantages of FPGAs in cryptographic applications were listed in [8]. One of the advantages was entitled "Architecture efficiency", of which they stated,

> In certain cases a hardware architecture can be much more efficient if it is designed for a specific set of parameters. Parameters for cryptographic algorithms can be, for example, the key, the underlying finite field, the coefficient used (e.g., the specific curve of an ECC system), and so on. Generally speaking, the more specific an algorithm is implemented the more efficient it can become.

In an FPGA, optimizations for specific parameters can be done without major restrictions in the generality of a system because, if other parameters are needed, the FPGA can be reprogrammed to implement the new parameters [8]. This fact has been exploited in numerous papers describing FPGA-based implementations of ECC [7]. However, a vast majority of papers optimize only field arithmetic units for one specific field while the higher abstraction levels of ECC still remain unoptimized and use more generic architectures. This approach seems rather pointless because fixing the underlying field already restricts the number of usable elliptic curves to very few. For instance, fixing the field to $\mathbb{F}_{2^{163}}$ means that from the total of 15 curves recommended by the U.S. National Institute of Standards and Technology (NIST) in [9] only two curves, namely B-163 and K-163, could be used. Hence, if the field is fixed in order to increase performance, one should optimize the architecture also on higher levels for a specific curve. In this paper, we describe an FPGA-based processor that is optimized specifically for Koblitz curves [10].

1.1. Related work

The first FPGA-based implementation using Koblitz curves was presented in [11], where one scalar multiplication was shown to require 45.6 ms on the NIST K-163 curve with an Altera Flex 10K FPGA. They concluded that Koblitz curves are approximately twice as fast as general curves. Ref. [12] presented an implementation which computes scalar multiplication in 75 μs on

E-mail address: kimmo.jarvinen@tkk.fi

[1] The author was supported by EU FP7 project CACE.

0167-9260/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2010.08.001

NIST K-163 in a Xilinx Virtex-E FPGA. Neither of the two designs includes a circuitry for conversions that are mandatory for Koblitz curves (see Section 2.3). Refs. [13,14] proposed a multiple-base expansion which can be used for increasing the speed of Koblitz curve computations and presented FPGA implementations for both elliptic curve scalar multiplication and conversion. Scalar multiplication was shown to require 35.75 μs on NIST K-163 with a Xilinx Virtex-II whereas the conversion requires 3.81 μs in [13]. Ref. [14] presented a parallelized version of the processor of [13] achieving computation delay of 17.15 μs on Stratix II including the conversion. Ref. [15] presented a high-speed processor using parallel formulations of scalar multiplication on Koblitz curves. Their processor achieves a very fast computation delay of 7.22 μs on NIST K-233 with Virtex-II, but it also neglects the mandatory conversions.

Our recent work considering scalar multiplication on Koblitz curves in FPGAs consists of [16–18]. It was shown in [18] that up to 166,000 signature verifications can be computed using a single Stratix II FPGA with parallel processing. More general parallelization studies were presented in [17] and they resulted in an implementation that computes scalar multiplication in 25.81 μs. Recently, we showed that even shorter computation delay of only 4.91 μs (without the conversion) can be achieved on NIST K-163 with interleaved operations [16].

1.2. Contributions of the paper

In this article, we present a processor for ECC that is optimized for a specific elliptic curve on all levels and, hence, it fully exploits the architecture efficiency advantage provided by FPGAs. This work builds on our previous works [16,19–21] and combines them into a highly efficient processor for computing scalar multiplications on Koblitz curves. Especially, this paper is based on [19] which presented a preliminary version of the processor. This contribution extends it by providing more details and further optimizations, as well as an adaptation to another scalar multiplication algorithm. We provide implementation results on an Altera Stratix II FPGA for NIST K-163 curve [9]. They show that the processor outperforms other implementations currently available in the literature.

1.3. Structure of the paper

The remaining of the paper is structured as follows. Section 2 presents the preliminaries of finite fields, elliptic curves, and Koblitz curves. Section 3 introduces algorithms that are used in the proposed processor. The processor architecture is described in detail in Section 4. Section 5 discusses certain properties of the processor and the implementation results are collected in Section 6. Finally, we conclude the paper in Section 7.

2. Preliminaries

2.1. Finite fields

Elliptic curves defined over finite fields $\mathbb{F}_q$ are used in cryptography and only curves over binary fields, where $q=2^m$, with polynomial basis are considered in this paper. Polynomial bases are commonly used in elliptic curve cryptosystems because they provide fast performance on both software and hardware. Another commonly used basis, normal basis, provides very efficient squaring but multiplication is more complicated.

Elements of $\mathbb{F}_{2^m}$ with polynomial basis are represented as binary polynomials with degrees less than m as $a(x)=\sum_{i=0}^{m-1} a_i x^i$. Arithmetic operations in $\mathbb{F}_{2^m}$ are computed modulo an irreducible polynomial[2] with a degree m. Because sparse polynomials offer considerable computational advantages, trinomials (three nonzero terms) or pentanomials (five nonzero terms) are used in practice. The curve, NIST K-163, considered in this paper is defined over $\mathbb{F}_{2^{163}}$ with the pentanomial $p(x)=x^{163}+x^7+x^6+x^3+1$ [9].

Addition, $a(x)+b(x)$, in $\mathbb{F}_{2^m}$ is a bitwise exclusive-or (XOR). Multiplication, $a(x)b(x)$, is more involved and it consists of two steps: ordinary multiplication of polynomials and reduction modulo $p(x)$. If both multiplicands are the same, the operation is called squaring, $a^2(x)$. Squaring is cheaper than multiplication because the multiplication of polynomials is performed simply by adding zeros to the bit vector. Reduction modulo $p(x)$ can be performed with a small number of XORs if $p(x)$ is sparse and fixed, i.e., the same $p(x)$ is always used, which is the case in this paper. Repeated squaring denotes several successive squarings, i.e., exponentiation $a^{2^e}(x)$. Inversion, $a^{-1}(x)$, is an operation which finds $b(x)$ such that $a(x)b(x)=1$ for a given $a(x)$. Inversion is the most complex operation and it can be computed either with the extended Euclidean algorithm or Fermat's little theorem (e.g., as suggested in [22]) that gives $a^{-1}(x)=a^{2^m-2}(x)$.

Multiplication has the most crucial effect on performance of an elliptic curve cryptosystem. A digit-serial multiplier computes D bits of the output in one cycle resulting in a total latency of $\lceil m/D \rceil$ cycles. We use hardware modifications of the multiplier described in [23]. Instead of using precomputed look-up tables as in [23], our multiplier computes everything on-the-fly similarly as in [12]. Repeated squarings can be computed efficiently with the repeated squarers presented in [21] which are components that compute $a^{2^e}(x)$ directly in one clock cycle.

2.2. Elliptic curve scalar multiplication

Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Points on E form an additive Abelian group, $E(\mathbb{F}_q)$, together with a point called the point at infinity, $\mathcal{O}$, acting as the zero element. The group operation is called point addition. Let P_1 and P_2 be two points in $E(\mathbb{F}_q)$. Point addition P_1+P_2 where $P_1=P_2$ is called point doubling. In order to avoid confusion, point addition henceforth refers solely to the case $P_1 \neq \pm P_2$.

The principal operation of elliptic curve cryptosystems is scalar multiplication kP where k is an integer and $P \in E(\mathbb{F}_q)$ is called the base point. The most straightforward practical algorithm for scalar multiplication is the double-and-add algorithm (binary algorithm) where k is represented as a binary expansion $\sum_{i=0}^{\ell-1} k_i 2^i$ with $k_i \in \{0,1\}$. Each bit in the representation results in a point doubling and an additional point addition is computed if $k_i=1$. Let w denote the Hamming weight of k, i.e., the number of nonzeros in the expansion. Depending on whether the algorithm starts from the most significant ($k_{\ell-1}$) or the least significant (k_0) bit of the expansion, the algorithm is called either left-to-right or right-to-left double-and-add algorithm. They are shown in Algorithms 1 and 2, respectively. They both have the same costs: $\ell-1$ point doublings and $w-1$ point additions (the first operations are simply substitutions).

[2] A polynomial, $f(x) \in \mathbb{F}[x]$, with a positive degree is irreducible over $\mathbb{F}$ if it cannot be presented as a product of two polynomials in $\mathbb{F}[x]$ with positive degrees.