High Speed Flexible Pairing Cryptoprocessor on FPGA Platform

Santosh Ghosh, Debdeep Mukhopadhyay, and Dipanwita Roychowdhury

Department of Computer Science and Engineering,
Indian Institute of Technology,
Kharagpur, India
{santosh,debdeep,drc}@cse.iitkgp.ernet.in

Abstract. This paper presents a Pairing Crypto Processor (PCP) over Barreto-Naehrig curves (BN curves). The proposed architecture is specifically designed for field programmable gate array (FPGA) platforms. The design of PCP utilizes the efficient implementation of the underlying finite field primitives. The techniques proposed maximize the utilization of in-built features of an FPGA device which significantly improves the performance of the primitives.

Extensive parallelism techniques have been proposed to realize a PCP which requires lesser clock cycles than the existing designs. The proposed design is the first reported result on an FPGA platform for 128-bit security. The PCP provides flexibility to choose the curve parameters for pairing computations.

The cryptoprocessor needs 1730 k, 1206 k, and 821 k cycles for the computation of Tate, ate, and R-ate pairings, respectively. On a Virtex-4 FPGA device it consumes 52 kSlices at 50MHz and computes the Tate, ate, and R-ate pairings in 34.6 ms, 24.2 ms, and 16.4 ms, respectively, which is comparable to known CMOS implementations.

Keywords: $\mathbb{F}_{p^k}$-arithmetic, FPGA, Barreto-Naehrig curves, elliptic-curve cryptography (ECC), pairing-based cryptography.

1 Introduction

CRYPTOGRAPHIC PAIRING [24] is a bilinear map $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are typically additive groups and $\mathbb{G}_3$ is a multiplicative group. Many cryptographic pairings such as the Tate pairing [27], ate pairing [20], and R-ate pairing [8] choose $\mathbb{G}_1$ and $\mathbb{G}_2$ to be specific cyclic subgroups of $E(\mathbb{F}_{p^k})$, and $\mathbb{G}_3$ to be a subgroup of $\mathbb{F}_{p^k}$. Selection of such groups as well as field types have a strong impact on the security and computation cost of pairing. Barreto-Naehrig curves (BN curves) [19] are a type of elliptic curves which support the computation of cryptographic pairings with a 128-bit security level. It is defined over a 256-bit prime field having embedding degree $k = 12$.

Related works. The software implementation results of pairings over BN curve have been shown in [1], [15], [13], and [16]. The highly optimized software

M. Joye, A. Miyaji, and A. Otsuka (Eds.): Pairing 2010, LNCS 6487, pp. 450–466, 2010.

codes run on a 64-bit core2 processor which computes a R-ate pairing in only $10,000,000$ cycles. The software implementation of [1] gives the speed record for the computation of Optimal-ate pairing on BN curves, which is computed by $4,470,408$ cycles on a Intel Core 2 Quad Q6600 processor.

An application specific instruction-set processor (ASIP) has been proposed in [5]. It is designed by extending a RISC core with additional scalable functional units. It requires a special programming environment in order to execute pairings. Therefore, the authors have developed a special C compiler. Implementation result shows that the ASIP can compute an Optimal-ate pairing in 15.8 ms over a 256-bit BN curve at 338 MHz with a 130 nm CMOS library.

A pairing processor specially for BN curves has been proposed in [6]. It exploits the characteristic of the field defined by BN curves and choose curve parameters such that the underlying $\mathbb{F}_p$ multiplication becomes more efficient. It shows a 5.4 times speedup of a pairing computation compared to the ASIP proposed in [5]. However, the main limitation of the pairing processor [6] is that it is useful only for computing pairings over a fixed BN curve.

Contribution. This paper proposes a flexible cryptoprocessor for the computation of pairings over BN curves. Field programmable gate array (FPGA) is one of the suitable platforms for implementing cryptographic algorithms. In this paper, we propose new implementation techniques of addition and multiplication on FPGAs. The in-built features available inside an FPGA device have been utilized to develop a high speed 256-bit adder circuit. We show that when utilizing such adder circuits and adopting a parallelism technique, the multiplication in $\mathbb{F}_p$ can be substantially improved. Based on such $\mathbb{F}_p$ arithmetic cores, we develop a parallel configurable hardware for computing addition, subtraction, and multiplication on $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. Existing techniques to speed up arithmetic in extension fields (see [16,21]) for fast computation in $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ are used on top of it. The major contributions of the paper are highlighted here.

- The paper implements underlying primitives for $\mathbb{F}_p$ arithmetic on FPGA platforms, which provides a significant speedup from existing platform-independent techniques.
- It proposes a pairing hardware that is flexible for curve parameters.
- Parallelism techniques are adopted in different levels including underlying finite field operations which drastically reduces the overall cycle count of pairing computation.
- The proposed FPGA design achieves a comparable speed with the existing CMOS design.

The proposed configurable $\mathbb{F}_{p^k}$ arithmetic cores and parallel computation result in a significant improvement on the performance of Tate, ate, and R-ate pairing over BN curves.

Organization of the paper. Section 2 of the paper gives a brief description of cryptographic pairings and BN curves. Efficient design of finite field primitives on FPGA platforms are described in section 3. Section 4 describes the pairing

cryptoprocessor. Section 5 shows the experimental results based on BN curves and provides comparative studies with existing contemporary designs. The paper is concluded in section 6.

2 Background of Pairings

The name bilinear pairing indicates that it takes a pair of vectors as input and returns a number, and it performs linear transformation on each of its variables. For example, the dot product of vectors is a bilinear pairing [11]. Similarly, for cryptographic applications the bilinear pairing (or pairing) operations are defined on elliptic or hyperelliptic Jacobian curves. Pairing is a mapping $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$, where $\mathbb{G}_1$ is a additive subgroup of $E(\mathbb{F}_q)$ on some elliptic or hyperelliptic Jacobian curve defined over a finite field $\mathbb{F}_q$, $\mathbb{G}_2$ is an another similar kind of subgroup of $E(\mathbb{F}_{q^k})$ over the lowest extension field $\mathbb{F}_{q^k}$, and $\mathbb{G}_3$ is a subgroup of the multiplicative group of $\mathbb{F}_{q^k}$. Here q is the characteristic field representative. It is normally 2^m, 3^m, or a large prime p. The parameter k corresponds to the embedding degree, often referred to as security multiplier in pairing computation, i.e. the smallest positive integer such that r divides $q^k - 1$ and r is a large odd prime which divides the order of the curve group ($\#E(\mathbb{F}_q)$). If the point P be a r-torsion point then the Tate pairing of order r is a map: $e_r : E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \rightarrow \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$, where $E(\mathbb{F}_q)[r]$ denote the subgroup of $E(\mathbb{F}_q)$ of all points of order dividing r, and similarly for $\mathbb{F}_{q^k}$. Tate pairing of order r satisfies the following properties:

- **Non-degeneracy :** For each $P \neq \mathcal{O}$ there exist $Q \in E(\mathbb{F}_{q^k})[r]$ such that $e_r(P, Q) \neq r$.
- **Bilinearity :** For any integer n, $e_r([n]P, Q) = e_r(P, [n]Q) = e_r(P, Q)^n$ for all $P \in E(\mathbb{F}_q)[r]$ and $Q \in E(\mathbb{F}_{q^k})[r]$.
- **Computability :** There exists an efficient algorithm to compute $e_r(P, Q)$ given P and Q.

The value e_r is representative of an element of the quotient group $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. However for cryptographic protocols it is essential to have a unique element so it is raised to the $((q^k - 1)/r)$-th power, obtaining an r^{th} root of unity. The resulting value is called reduced Tate pairing.

2.1 Choice of Elliptic Curve

The most important parameters for cryptographic pairings are the underlying finite field, the order of the curve, the embedding degree, and the order of $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_3$. These parameters should be chosen such that the best exponential time algorithms to solve the discrete logarithm problem (DLP) in $\mathbb{G}_1$ and $\mathbb{G}_2$ and the sub-exponential time algorithms to solve the DLP in $\mathbb{G}_3$ take longer than a chosen security level. This paper uses the 128-bit symmetric key security level. For the 128-bit security level, the National Institute of Standards and Technology (NIST) recommends a prime group order of 256 bits for $E(\mathbb{F}_p)$ and of 3072 bits for the finite field $\mathbb{F}_{p^k}$ [17].

Barreto-Naehrig curves, introduced in [19], are elliptic curves over fields of prime order p with embedding degree $k = 12$. The BN curve is represented as :

$$E_{\mathbb{F}_p} : Y^2 = X^3 + 3$$

with BN parameter $z = 6000000000001F2D$ (in hexadecimal) [16]. It forms the group $E(\mathbb{F}_p)$ with order $\#E(\mathbb{F}_p) = r = 36z^4 + 36z^3 + 18z^2 + 6z + 1$, which is a 256-bit prime of Hamming weight 91. The field characteristic $p = 36z^4 + 36z^3 + 24z^2 + 6z + 1$ is a 256-bit prime of Hamming weight 87, and $t - 1 = p - r = 6z^2 + 1$ is a 128-bit integer of Hamming weight 28. Here $t = p + 1 - r$ is the trace of $E_{\mathbb{F}_p}$. The prime $p \equiv 7 \pmod 8$ (so -2 is a quadratic non-residue, we represent it by β) and $p \equiv 1 \pmod 6$.

2.2 Pairing Computation

Pairing computation consists of two major steps : the computation of Miller function and the final exponentiation. Algorithm 1 shows computation of Tate pairing. The first part is computed by one of the optimized version of Miller algorithm [25]. Several optimizations of this algorithm have been presented in [27]. The resulting algorithm is called BKLS algorithm.

Algorithm 1. Computing the Tate pairing

Input: $P \in G_1$ and $Q \in G_2$.
Output: $e_r(P, Q)$.

Write r in binary : $r = \sum_{i=0}^{L-1} r_i 2^i$.
$T \leftarrow P$, $f \leftarrow 1$.
for *i from* $L - 2$ *downto* 0 **do**
  $T \leftarrow 2T$.
  $f \leftarrow f^2 \cdot l_{T,T}(Q)$.
  if $r_i = 1$ *and* $i \neq 0$ **then**
    $T \leftarrow T + P$.
    $f \leftarrow f \cdot l_{T,P}(Q)$.
  end
end
return $f^{(q^k-1)/r}$.

The BN curves also admits a sextic twist [15], which means that the point Q is mapped on a point Q' defined over $\mathbb{F}_{p^2}$. Thus the line functions $l_{T,T}(Q)$ and $l_{T,P}(Q)$ is computed over $\mathbb{F}_{p^2}$ instead of $\mathbb{F}_{p^{12}}$. Value of the line functions are represented as : $l_0 + l_1 W^2 + l_2 W^3$, with $l_0 \in \mathbb{F}_p$, $l_1, l_2 \in \mathbb{F}_{p^2}$, and a quadratic non-residue W over $\mathbb{F}_{p^2}$. The Miller function f is computed over $\mathbb{F}_{p^{12}}$, which is represented as : $f_0 + f_1 W + f_2 W^2 + f_3 W^3 + f_4 W^4 + f_5 W^5$, with $f_i \in \mathbb{F}_{p^2}$. So in the Tate pairing computation f^2, $f \cdot l_{T,T}(Q)$, and $f \cdot l_{T,P}(Q)$ are performed on $\mathbb{F}_{p^{12}}$. Whereas all other computations are performed on $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$.

The detailed procedure of pairing computation including the final exponentiation on BN curve is described in [15] and [16]. Another efficient way of computing final exponentiation is described in [7]. This paper follows the descriptions that are given in [15] for computing the Tate, ate, and R-ate pairings. We use Jacobian coordinate systems for performing elliptic curve operations, where a point (X, Y, Z) corresponds to the point (x, y) in affine coordinates with $x = X/Z^2$ and $y = Y/Z^3$. Let (m, s, i) denote the cost of multiplication, squaring, inversion in $\mathbb{F}_p$. Using Jacobian coordinate system the Miller function of Tate pairing on BN curve requires $27934m$ and the final exponentiation requires $7246m + i$ [15]. Thus the total cost for Tate pairing on BN curve is $35180m + i$. Similarly, the cost of ate pairing is $23047m + i$ and the cost of R-ate pairing is $15093m + 2i$.

3 Implementing $\mathbb{F}_p$ Primitives on FPGA

In 1983 Blakley introduced an interesting algorithm to perform modular multiplication of two integers A and B modulo an integer M [30,31]. It is an iterative binary double-and-add algorithm. The main idea of the algorithm is that it keeps the intermediate result after each iteration below the modulus value. Thus it avoids final division. Let A be the multiplicand, B be the multiplier, and $R = (A \cdot B) \bmod M$, where A, B, M, R are represented in two's complement number system. The binary representation of $B = \sum_{i=0}^{l-1} b_i 2^i$, and R is initialized by A. The algorithm first computes $R = 2R \bmod M$, and if $b_i = 1$ then it computes $R = (R + A) \bmod M$. The mod M operation are performed by single subtraction in both cases.

In the context of $\mathbb{F}_p$ multiplication the modulus M corresponds to p. All arithmetics in $\mathbb{F}_p$ are performed in two's complement number system. Therefore, all values are kept in two's complement number system throughout the whole pairing computation, which avoids input and output conversions like existing implementations [5,6].

3.1 Fast Carry Chains for $\mathbb{F}_p$ Primitives

The main difficulty of the Blakley algorithm is that the computation of addition on large operands. The modified Blakley algorithm for large operands are shown in [22,26]. The use of carry save adder (CSA) helps to speed up the repeated additions on large operands. However these modified versions require at least one final addition on large carry chain. Also they use some pre-computed values which require additional time and storage area.

This paper exploits the features available in an FPGA device for efficient computation of Blakley algorithm on large operands. The specific features that are available in an FPGA device are efficiently utilized for developing arithmetic primitives in $\mathbb{F}_{2^m}$ fields in [14]. However, to the best of the authors knowledge no work is existing in the literature for the same in case of $\mathbb{F}_p$ field. The modern FPGA consists of 16 slices (or 32 LUTs) within a single row which are connected through an in-built fast carry chain (FCC). The FCC can perform

addition on two 32-bit operands most efficiently compared to any other adder structures [3,29]. It is experimentally shown that on a Virtex-4 FPGA device the latency of a 32-bit addition using a fast carry chain takes only 5.8 ns; whereas, the same using a carry lookahead structure takes 8.7 ns. Hence, fast carry chain is 1.5 times faster than carry lookahead structure for computing addition of two 32-bit operands on an FPGA platform. In order to compute an addition of two operands longer than 32 bits, the FPGA will utilize more than one row which requires additional routing delay. For example, the addition $(A+B)$ of two 64-bit operands (A, B) using a single 64-bit carry chain is slower than the same using three 32-bit FCC and a 2:1 multiplexor [3].

We develop an efficient 256-bit adder using 32-bit fast carry chains. The repeated Karatsuba decomposition is applied on 256-bit operands. An operand is decomposed upto a depth of three for converting it into eight pieces of 32-bit operands. A 64-bit addition is performed by using three 32-bit fast carry chains with a carry select structure. Let, $A = A_1 2^{32} + A_0$, $B = B_1 2^{32} + B_0$, and $C = A + B$, where A_i, B_i are 32-bit integers. We compute $A_0 + B_0$, $A_1 + B_1 + 0$, and $A_1 + B_1 + 1$ in parallel on three FCC. Then the carry out of the least significant addition $(A_0 + B_0)$ is used to multiplex the results of the most significant additions. Thus the latency of a 64-bit adder is 1 FCC + 1 MUX, where MUX corresponds to a 2:1 multiplexer. Similarly, a 128-bit adder is developed by three 64-bit adders, and a 256-bit adder is developed by three 128-bit adders. Therefore, a 256-bit adder is developed hierarchically from 32-bit adder. At every level of hierarchy it adds one additional MUX in the critical path. Thus the latency of a 256-bit adder is 1 FCC + 3 MUX delay, which is $9.9ns$ on a Virtex-4 FPGA. Whereas, the latency of a 256-bit carry lookahead adder on the same platform is $16.7ns$, which is 1.7 times slower than the above technique.

3.2 Programmable $\mathbb{F}_p$ Primitive

The 256-bit high speed adder circuit that is designed by utilizing fast carry chains and Karatsuba decomposition technique is exploited to develop a programmable $\mathbb{F}_p$ primitive. Figure 1 shows the overall resulting structure of the $\mathbb{F}_p$ adder/subtractor/multiplier unit. We use the same structure for all three $\mathbb{F}_p$ operations for pairing computation. The configuration of the design to perform addition and subtraction can be easily formed. The input parameters (A, B) are added by the right most adder unit and then reduced (if necessary) by adding 2's complement of p. The select lines c_1 and c_2 of mx blocks (multiplexors) are generated from the carry out of 256-bit adders, which decide whether the resultant values are greater than the modulus p. The unit performs addition and subtraction in $\mathbb{F}_p$ in one clock cycle.

Proposed hardware follows the parallelism technique of Montgomery ladder [28] for computing Blakley multiplication algorithm in $\mathbb{F}_p$. The choice of this algorithm is due to its lower hardware cost. The algorithm also provides the scope of adaptability with Montgomery ladder for parallelism. The Montgomery ladder computes two intermediate results (R_0, R_1) in each iteration. It computes independent modular doubling and modular addition in each iteration, which are

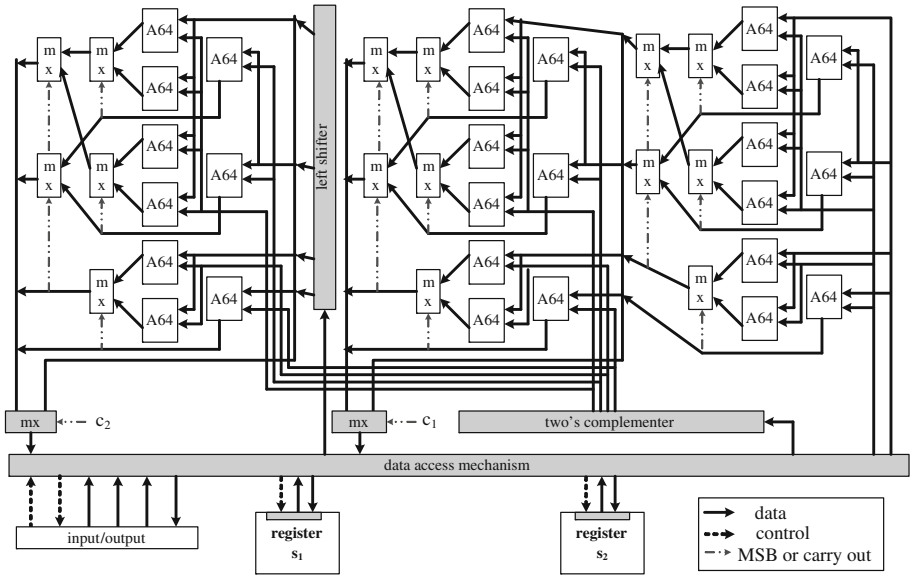

Fig. 1. The architecture of $\mathbb{F}_p$ adder/subtractor/multiplier unit

performed in parallel on the proposed unit. The data transfer in the registers for computing $(A \cdot B)$ mod p is as follows :

- The *register* s_1 is initialized by zero, and *register* s_2 is initialized by A, which are used to hold the intermediate result R_0 and R_1, respectively.
- Iterative execution starts from $i = 255$ and goes down to zero, where it is considered that the operands are the members of $\mathbb{F}_p$ with a 256-bit p.
- The right part of *left-shifter* block performs $(R_0 + R_1)$ mod p. This modular addition is performed by two 256-bit adder units. The first adder performs $R_0 + R_1$, whereas the second one makes the reduction by a 2's complement subtraction. The final result is multiplexed by the mx_1 block. The *left-shifter* performs $2R_{b_i}$ and its left part computes the reduction $(2R_{b_i})$ mod p. The final result is multiplexed by mx_2 block. In the architecture, an *A64* block corresponds to a 64-bit adder unit.
- The *data access mechanism* restores the intermediate results $R_{\bar{b_i}} \leftarrow (R_0+R_1)$ mod p and $R_{b_i} \leftarrow (2R_{b_i})$ mod p, where $\bar{b_i}$ represents the complement of b_i.
- After final iteration, the *register* s_1 holds the result of $(A \cdot B)$ mod p.

The procedures runs iteratively for 256 times. Thus, the above unit performs one 256-bit $\mathbb{F}_p$ multiplication in only 256 cycles.

4 The Pairing Cryptoprocessor (PCP)

This section describes the proposed architecture of the cryptoprocessor. The architecture is based on the efficient utilization of FPGA features. The parallelism

techniques are adopted in different level of computations for overall speed up. First, we explain the top level of the design followed by the internal parts.

4.1 The Datapath Design

The major operations for pairing computations are point doubling (PD), point addition (PA), line computation ($l(Q)$), f^2, and $f \cdot l(Q)$. In case of Tate pairing on BN curve, the PA and PD are performed on $E(\mathbb{F}_p)$. Hence, the underline operations are performed in $\mathbb{F}_p$. Similarly, the operation $l(Q)$ is performed in $\mathbb{F}_{p^2}$ while the other two operations are performed in $\mathbb{F}_{p^{12}}$. In case of ate and R-ate pairings, the PA, PD, $l(Q)$ are performed in $\mathbb{F}_{p^2}$, and f^2, $f \cdot l(Q)$ are performed in $\mathbb{F}_{p^{12}}$. However, each of the above computations are well defined and constitute a sequence of $\mathbb{F}_p$ operations which provides a scope of parallelism. The proposed datapath exploits such properties to speed up pairing computations on FPGA platforms.

Figure 2 shows the overall resulting structure of the datapath. Two Configurable $\mathbb{F}_{p^k}$ Arithmetic Units (CAU) are included which perform arithmetic in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ depending on their mode of configurations. The instructions that are decoded to configure the CAUs are stored into a small instruction memory segment. There is a special instruction fetch and decode (IFD) unit which reads the respective instructions and converts them to proper configuration signals for both the CAUs. The input data to the CAUs come in parallel from respective registers. The mechanism and regularity of data access for computing above operations are fairly simple. The distribution of access to the registers and resolution of access conflicts are handled efficiently at the runtime by a dedicated hardware block, the data access unit (DAU) that distributes the data access to the CAUs from the correct register and vice versa.

Each CAU performs atmost three $\mathbb{F}_p$ operations in parallel. Thus, overall twelve independent operands along with modulus p and six outputs are accessed in either directions between memory elements and the CAUs. This on-demand concurrent data requests result in multiple independent read or write connections between CAUs and DAU. The DAU takes care of granting accesses. Therefore, a simple multiplexing protocol is used between CAUs and registers, which is able to confirm a request within the same cycle in order not to cause any delay cycles when trying to access data in parallel. The data accesses and instruction sequences are hard coded into the *sequence control* of the architecture which avoids the additional software development costs.

The data access conflicts have been resolved prior to design the DAU protocol. The proposed one is a custom hardware for pairing computations which executes a fixed set of operations. The dependency of the instructions are predefined and thus the access conflicts are known. The priority of the data processing and the respective execution is rearranged accordingly which achieves maximum utilization of CAUs.

Figure 3 depicts the functionality of data access unit. The major connections between registers and CAUs are shown while the DAU performs as a mediator. Due to the demand of parallel access, the proposed pairing hardware stores all

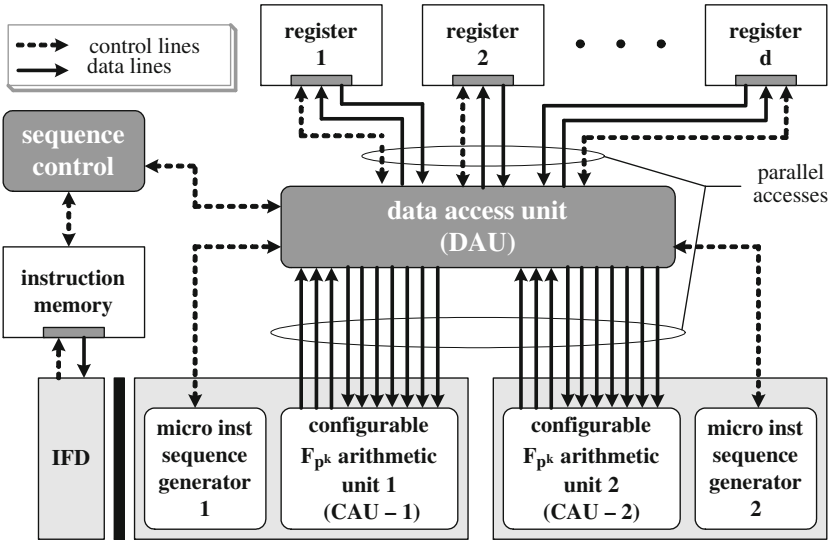

Fig. 2. The datapath of the pairing cryptoprocessor

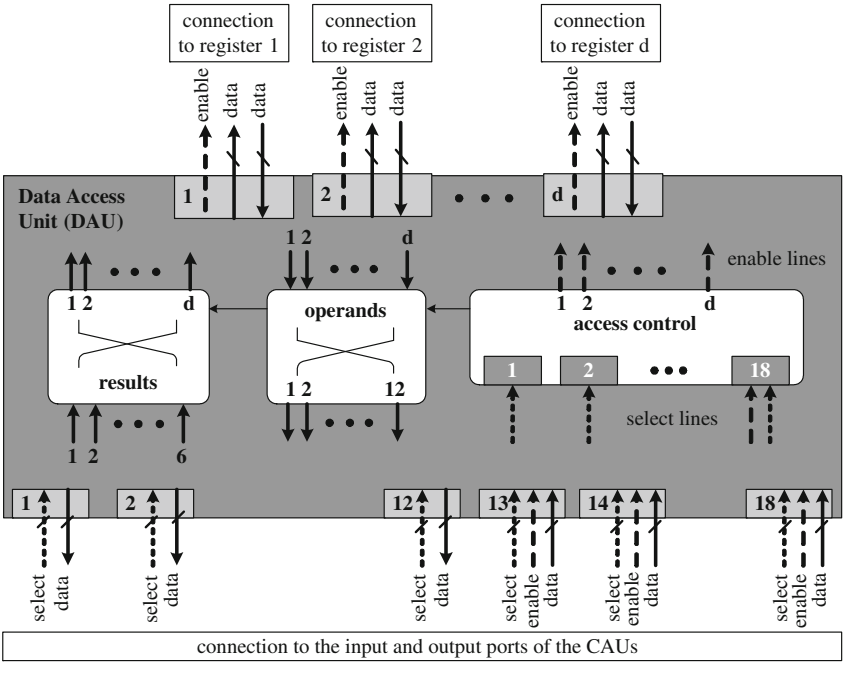

Fig. 3. The interconnect of data access unit (DAU)

intermediate results in its active registers. Fifty 256-bit registers ($d = 50$) are incorporated. Each of the register consists of data-in, data-out, and enable lines. It gets updated by data-in lines when the respective enable signal is invoked. The crossbar switch (*results*) redirects the outputs of each operation to registers. Similarly, the *operands* are redirected from registers to the input ports of the CAUs. The respective select signals are generated prior to the above two redirection procedures. The *access control* block synchronizes the select lines of the multiplexors for operands and results. It also synchronizes the enable signals of registers for restoring the intermediate results.

The *sequence control* runs synchronously with the IFD unit. It generates the select and enable signals with respect to each of the operand and result port which are going inside the DAU. In brief, the *sequence control* generates the signals for controlling input/output to the CAUs and registers while IFD is responsible to generate respective signals for configuring each of the CAUs.

A Configurable $\mathbb{F}_{p^k}$ Arithmetic Unit (CAU). It is observed that the major operations for pairing computations over BN curves are performed either on $\mathbb{F}_p$ or on $\mathbb{F}_{p^2}$. Thus we design a configurable architecture for performing arithmetics in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. Figure 4 shows the data processing inside the proposed CAU. It consists of three $\mathbb{F}_p$ arithmetic units, which are individually capable to perform addition, subtraction, and multiplication in $\mathbb{F}_p$. Thus a CAU performs atmost three parallel $\mathbb{F}_p$ operations, which demand six operands and a modulus p in parallel. The final outputs are stored in first three registers, respectively.

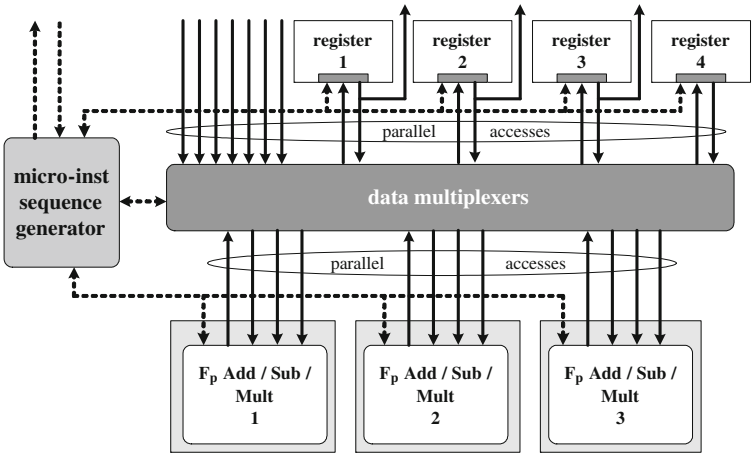

Fig. 4. The architecture of Configurable $\mathbb{F}_{p^k}$ Arithmetic Unit (CAU)

The major $\mathbb{F}_{p^2}$-operations are multiplication and squaring. Let an element $\alpha \in \mathbb{F}_{p^2}$ be represented as $\alpha_0 + \alpha_1 X$, where $\alpha_0, \alpha_1 \in \mathbb{F}_p$ and X is an indeterminate. The formula of Karatsuba multiplication $c = ab$ on $\mathbb{F}_{p^2}$ is :

$$v_0 = a_0 b_0,\ v_1 = a_1 b_1,$$
$$c_0 = v_0 + \beta v_1,$$
$$c_1 = (a_0 + a_1)(b_0 + b_1) - v_0 - v_1,$$

where $v_0, v_1, c_0, c_1, a_0, a_1, b_0, b_1 \in \mathbb{F}_p$. Here β is a quadratic non-residue in $\mathbb{F}_p$ which is -2 in case of BN curve. Similarly, the squaring $c = a^2$ on $\mathbb{F}_{p^2}$ using Complex method is computed by :

$$v_0 = a_0 a_1,$$
$$c_0 = (a_0 + a_1)((a_0 + \beta a_1)) - v_0 - \beta v_0$$
$$c_1 = 2v_0.$$

Hence the costs of multiplication and squaring in $\mathbb{F}_{p^2}$ are $3m$ and $2m$ [21]. However, due to the parallel independent structure, the cost of both operations on a CAU is only m.

The *micro inst sequence generator* finds the current operation type and generates the respective micro instructions. For example, let the current operation is a $\mathbb{F}_{p^2}$ multiplication. The sequence of operations for computing $c = ab$ in $\mathbb{F}_{p^2}$ on a CAU is as follows :

1. It computes $t_3 \leftarrow a_0 + a_1$ and $t_4 \leftarrow b_0 + b_1$ and stores them into *register-3* and *register-4*, respectively.
2. It performs $t_1 \leftarrow a_0 b_0$, $t_2 \leftarrow a_1 b_1$, and $t_3 \leftarrow t_3 t_4$ in parallel, and stores the results into *register-1*, *register-2*, and *register-3*, respectively.
3. It performs $t_1 \leftarrow t_1 - t_2$ and $t_4 \leftarrow t_1 + t_2$, and stores them into *register-1* and *register-4*.
4. It performs $t_1 \leftarrow t_1 - t_2$ and $t_2 \leftarrow t_3 - t_4$, and stores them into *register-1* and *register-2*.
5. It outputs the value of *register-1* and *register-2* as c_0 and c_1, respectively.

The *micro inst sequence generator* generates micro instruction like register enable, CAU reset, operand select, done, etc. It is constructed as a typical state machine which generates micro instructions at each state. Its deterministic state transition takes place at every cycle based on the current state and overall status of the CAU. In case of $\mathbb{F}_p$ multiplication, it remains in a same state for 256 cycles. Whereas, it remains only one cycle in a same state for computing $\mathbb{F}_p$ addition and subtraction. Thus the computation of $c = ab$ in $\mathbb{F}_{p^2}$ takes only 260 cycles which costs approximately m.

4.2 Computation of the Pairings on PCP

In this paper we follow the formula and algorithms for the computation of cryptographic pairings that are given in [15]. However, due to the multiple functional units the operations are performed in parallel by the proposed PCP. In Jacobian coordinates the formulae for doubling a point $T = (X, Y, Z)$ are $2T = (X_3, Y_3, Z_3)$ where $X_3 = 9X^4 - 8XY^2$, $Y_3 = (3X^2)(4XY^2 - X_3) - 8Y^4$

and $Z_3 = 2YZ$. The computation of $2T$, $l_{T,T}(Q)$, and f^2 are performed in parallel. The tangent line at T, after clearing denominators, is $l(x,y) = 3X^3 - 2Y^2 - 3X^2Z^2x + Z_3Z^2y$ [23].

In case of Tate pairing computation on BN curves $X, Y, Z, X_3, Y_3, Z_3 \in \mathbb{F}_p$ and $x, y \in \mathbb{F}_{p^2}$. The above operations are performed by one of the CAUs which costs $6m$. At the same time another CAU starts the computation of f^2. We represent the Miller function f as $(f_{0,0} + f_{0,1}v + f_{0,2}v^2) + (f_{0,0} + f_{0,1}v + f_{0,2}v^2)w$, where $f_{i,j} \in \mathbb{F}_{p^2}$. The equivalent representations of f are :

$$\begin{aligned} f &= f_0 + f_1 w, \text{ where} f_0, f_1 \in \mathbb{F}_{p^6} \\ &= (f_{0,0} + f_{0,1}v + f_{0,2}v^2) + (f_{1,0} + f_{1,1}v + f_{1,2}v^2)w, \text{ where} f_{i,j} \in \mathbb{F}_{p^2} \\ &= f_{0,0} + f_{1,0}W + f_{0,1}W^2 + f_{1,1}W^3 + f_{0,2}W^4 + f_{1,2}W^5 \end{aligned}$$

Thus, the squaring (f^2) is performed in $\mathbb{F}_{p^{12}}$ which requires two $\mathbb{F}_{p^6}$ multiplications using Complex method. Now, one $\mathbb{F}_{p^6}$ multiplication requires six multiplications in $\mathbb{F}_{p^2}$. The first $\mathbb{F}_{p^6}$ multiplication is computed in parallel with $2T$, $l_{T,T}(Q)$ by another CAU which costs $6m$. The second $\mathbb{F}_{p^6}$ multiplication is performed by both the CAUs, which costs only $3m$ in the PCP. Therefore, the total cost of computing $2T$, $l_{T,T}(Q)$, and f^2 by the PCP is $9m$.

The l_Q is represented as : $(l_0 + l_1 v) + (l_2 v)w$, where $l_0 \in \mathbb{F}_p$, $l_1, l_2 \in \mathbb{F}_{p^2}$, which is equivalent to $l_0 + l_1W^2 + l_2W^3$. The computation of $f \cdot l(Q)$ is done by three $\mathbb{F}_{p^6}$ multiplications using Karatsuba multiplication procedure. Due to the sparse representation of l_Q the costs are lesser than the actual costs. In total the computation of $f \cdot l(Q)$ requires 37 $\mathbb{F}_p$ multiplications, which costs only $7m$ in PCP.

The formulae for mixed Jacobian-affine addition are the following: if $P = (X_1, Y_1, Z_1)$ is in Jacobian coordinates and $Q = (X_2, Y_2)$ is in affine coordinates, then $P+Q = (X_3, Y_3, Z_3)$ where $X_3 = (Y_2Z_1^3 - Y_1)^2 - (X_2Z_1^2 - X_1)^2(X_1 + X_2Z_1^2)$, $Y_3 = (Y_2Z_1^3 - Y_1)[X_1(X_2Z_1^2 - X_1)^2 - X_3] - Y_1(X_2Z_1^2 - X_1)^3$, $Z_3 = (X_2Z_1^2 - X_1)Z_1$. The line through T and P is $l(x,y) = [(Y_2Z_1^3 - Y_1)X_2 - Y_2Z_3] - (Y_2Z_1^3 - Y_1)x + Z_3y$. The cost of computing $T + P$, $l_{T,P}(Q)$ is $5m$ in the PCP.

Therefore, the cost of computing doubling step of the Miller algorithm is $16m$ and the cost for evaluating addition step is $12m$ in the PCP. The total cost for evaluating iterative Miller function of the Tate pairing computation over BN curves is $5176m$. The final exponentiation is computed by the exact procedures that are described in [15]. It requires one inversion in $\mathbb{F}_p$ which we perform by a^{p-2}. The cost for computing the final exponentiation is $1477m$ in our PCP. Hence, the total cost for computing a Tate pairing over BN curves by our cryptoprocessor is $6653m$, which takes $1,729,780$ cycles.

The ate pairing interchanges the input points of Tate pairing and it runs a smaller number of iterations. It uses $t - 1$ (instead of r) to determine the number of iterations in the Miller operation [15]. In case of BN curve $t \approx \sqrt{r}$, which makes the number of iterations halved. The computation costs $3165m$, $1477m$, and $4642m$ for the Miller operation, the final exponentiation, and the ate pairing, respectively on our proposed hardware. Hence, the number of cycles required to compute an ate pairing by the PCP is $1,206,902$.

The R-ate pairing follows the same procedures of ate pairing but it uses $a = 6z + 2$ (instead of $t - 1$) to determine the number of iterations in the Miller operation. Since, $a \approx \sqrt{t}$ on BN curves, the Miller operation in R-ate pairing has half as many iterations as in ate pairing computation. In case of R-ate pairing the Miller operation consists an additional step which requires an inversion in $\mathbb{F}_p$. The costs $1681m$, $1477m$, and $3158m$ for the Miller operation, the final exponentiation, and the R-ate pairing, respectively. The total number of cycles required to compute an R-ate pairing is $821,080$ by the proposed hardware.

5 Results

The whole design has been done in Verilog (HDL). All results have been obtained from the place-and-route report of Xilinx ISE Design Suit [10] using a Virtex-4 xc4vlx200-12ff1531 FPGA device with a supply voltage of $1.2V$. The design can run at a maximum frequency of $50MHz$. The pairing hardware uses around $52k$ logic slices including controllers and data access unit. It uses $27k$ flip flops for registers. It finishes one Tate, ate, and R-ate pairing computations in $34.6ms$, $24.2ms$, and $16.4ms$, respectively. Table 1 shows the implementation results.

Table 1. Implementation results of proposed hardware on xc4vlx200 device

Operation	Slice	LUT	FF	Frequency [MHz]	Cycles	Security [bit]	Times [ms]
Tate	52 k	101 k	27 k	50	1730 k	128	34.6
ate					1207 k		24.2
R-ate					821 k		16.4

The critical path of the design goes through the data access mechanism, then through two 256-bit adders, the multiplexer mx_1, and back through data access mechanism. In § 3.1 it is shown that the latency of a 256-bit adder circuit is $9.9ns$. However, this addition latency consists of input buffer delay of $1.3ns$, addition logic delay of $6.2ns$, and output buffer delay of $2.4ns$. The individual delays of the addition logic includes input and output buffer delays. In our architecture the critical path is within two internal registers which includes neither the input buffer nor the output buffer. Therefore, the total latency of the critical path of the design is calculated as $3.8ns + 2 \times 6.2ns + 1.6ns + 2.2ns = 20ns$.

5.1 Comparison with Pairing Implementations

This section provides the performance comparison with related pairing implementations over BN curves. Performances are compared with actual implementations of cryptographic pairings on software and dedicated hardware achieving a 128-bit security level. Hardware implementations of η_T pairing over binary and cubic curves are shown in [9,18]. These designs are for lower security levels

Table 2. Hardware and software implementations of pairing over BN-curves

Reference designs	Platform	Pairing	Frequency $[MHz]$	Area	Cycles	Times $[ms]$
our design	Virtex-4	Tate	50	52 kSlices	1730 k	34.6
		ate			1206 k	24.2
		R-ate			821 k	16.4
Kammler *et al.* [5]	130 nm CMOS	Tate	338	97 kGates	11 627 k	34.4
		ate			7706 k	22.8
		R-ate			5340 k	15.8
Fan *et al.* [6] §	130 nm CMOS	ate	204	183 kGates	862 k	4.2
		R-ate			593 k	2.9
Naehrig *et al.* [1]	core2 Q6600	Opt.-ate	-	-	4 470 k	-
Beuchat *et al.* [4]	core i7 2:8GHz	Opt.-ate	-	-	2 630 k	-
Hankerson *et al.* [15]	Pentium-4 64-bit core2	ate	2400	-	81 000 k	-
		R-ate			10 000 k	-
Grabher *et al.* [13]	64-bit core2	ate	2400	-	14 640 k	-
Devegili *et al.* [16]	Pentium-4	Tate	3400	-	156 740 k	-
		ate			133 620 k	-

§ implementation specifically for BN-curves with fixed parameters.

(72-bit) and hence it shall be unfair to compare with the present work. Table 2 gives a performance comparison of related hardware and software implementations.

Due to the parallel structure our PCP computes six $\mathbb{F}_p$ multiplications in parallel which are completed in 256 cycles. The main features that strengthen the proposed PCP for pairing computations are as follows :

- The proposed cryptoprocessor is the first FPGA results for pairing computation with 128-bit security.
- Our adopted parallelism and efficient use of two $\mathbb{F}_{p^k}$ arithmetic cores reduce the total number of cycles drastically.
- Due to the inherent properties the frequency of a design in FPGA is much lower than that in ASIC (CMOS standard cell). However, the speed achieved of the PCP is comparable to the CMOS standard cell design.
- The PCP is flexible to configure for different curve parameters.

The underlying platform plays a crucial role in determining the performance of a design. Thus, existing designs on different platforms does not lead to a fair comparison. We try to find out the platform independent features of existing designs and compare them with our proposed one. The cycles required to compute pairings on different designs may be considered such a parameter.

Kammler *et al.* [5] reported the first hardware implementation of cryptographic pairings achieving a 128-bit security. In [5] the proposed hardware is not only a cryptoprocessor, but an actual ASIP : it is in fact a general purpose processor, augmented with finite field arithmetic units in order to compute pairings. It uses the same z that we have considered to generate a 256-bit BN curve. The Montgomery algorithm is used for $\mathbb{F}_p$ multiplication. The platform of the

design is 130 nm CMOS standard cell library, whereas our design is on Virtex-4 FPGA. The main feature of the design [5] is the fast modular multiplication in $\mathbb{F}_p$ which takes only 68 cycles. The average cycle count of our PCP for one $\mathbb{F}_p$ multiplication is only 43 which is 1.6 times faster than [5]. With respect to the Tate pairing computation, the design of [5] takes 11 627 k cycles, whereas our design takes only 1730 k cycles, which is much less (0.15 times) compared to [5].

Fan *et al.* [6] proposed a processor for cryptographic pairing over BN curves. They designed a fast modular multiplier in $\mathbb{F}_p$ only for BN parameters which takes only 23 cycles. The 130 nm ASIC design of [6] provides the best known performance which takes only $2.9ms$ for computing a R-ate pairing over BN-curve. This design also attain smaller area-latency product than that in [5]. But the main drawback of the design proposed in [6] is that it does not provide the flexibility to compute pairings on chosen parameters. Whereas, our design provides the above flexibility in all aspects which indeed requires more cycles.

The results of software implementations [15,13] are quite impressive. On an Intel 64-bit core2 processor, R-ate pairing requires only $10,000,000$ cycles. The advantages of Intel core2 is that it has a fast multiplier (two full 64-bit multiplications in 8 cycles) and relatively high clock frequency. It takes 13 times more clock cycles than our cryptoprocessor. In a very recent work by Naehrig et al. [1] shows that the Optimal-ate pairing on BN curves can be computed by $4,470,408$ cycles on an Intel Core 2 Quad Q6600 processor. The software implementation of same pairing on a different curve is described in [4]. It takes only 2.63 million clock cycles on a Intel Core i7 : 2.8 GHz processor. However, the exact time required to compute pairings by executing softwares on a Desktop or Server systems are not predictable. It depends on so many other factors like available cache memory, context switching, bus speed of the system, etc.

6 Conclusion

In this paper we explored the inherent FPGA features for designing efficient $\mathbb{F}_p$ primitives. The parallel scheduling has been applied to speed up multiplication in $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^{12}}$. The proposed pairing cryptoprocessor can be programmed for any curve parameters. It provides a comparable speed with the existing ASIC designs. The overall clock cycles required to compute pairings over BN curves are less than existing designs. To the best of our knowledge it is the first FPGA result for high security (128-bit) cryptographic pairings.

The recent work by Granger and Scott [2] shows that a 1-2-4-12 towering is to be preferred to a 1-2-6-12 towering as implemented in this work. Therefore, in future it could be considered for further speedup of pairing computations.

Acknowledgement

The authors would like to thank the anonymous reviewers for their critical suggestions that greatly improved the quality of this paper.

References

1. Naehrig, M., Niederhagen, R., Schwabe, P.: New software speed records for cryptographic pairings. Cryptology ePrint Archive, Report 2010/186, http://eprint.iacr.org/
2. Granger, R., Scott, M.: Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 209–223. Springer, Heidelberg (2010)
3. Ghosh, S., Mukhopadhyay, D., Roychowdhury, D.: High Speed F_p Multipliers and Adders on FPGA Platform. In: DASIP 2010, Scotland (2010)
4. Beuchat, J.L., Díaz, J.E.G., Mitsunari, S., Okamoto, E., Henríquez, F.R., Teruya, T.: High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves. Cryptology ePrint Archive, Report 2010/354, http://eprint.iacr.org/
5. Kammler, D., Zhang, D., Schwabe, P., Scharwaechter, H., Langenberg, M., Auras, D., Ascheid, G., Mathar, R.: Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 254–271. Springer, Heidelberg (2009)
6. Fan, J., Vercauteren, F., Verbauwhede, I.: Faster F_p-arithmetic for cryptographic pairings on Barreto-Naehrig curves. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 240–253. Springer, Heidelberg (2009)
7. Scott, M., Benger, N., Charlemagne, M., Perez, L.J.D., Kachisa, E.J.: On the Final Exponentiation for Calculating Pairings on Ordinary Elliptic Curves. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 78–88. Springer, Heidelberg (2009)
8. Lee, E., Lee, H.S., Park, C.M.: Efficient and generalized pairing computation on abelian varieties. Cryptology ePrint Archive, Report 2009/040, http://eprint.iacr.org/
9. Beuchat, J., Detrey, J., Estibals, N., Okamoto, E., Rodríguez-Henríguez, F.: Hardware accelerator for the Tate pairing in characteristic three based on Karatsuba-Ofman multipliers. Cryptology ePrint Archive, Report 2009/122 (2009)
10. Xilinx ISE Design Suit (2009), http://www.xilinx.com/tools/designtools.htm
11. Hoffstein, J., Pipher, J., Silverman, J.H.: An introduction to mathmatical cryptography. Springer, Heidelberg (2008)
12. Barenghi, A., Bertoni, G., Breveglieri, L., Pelosi, G.: A FPGA coprocessor for the cryptographic Tate pairing over F_p. In: Proc. Fifth Intl. Conf. Information Technology: New Generations - ITNG 2008, pp. 112–119 (2008)
13. Grabher, P., Großschädl, J., Page, D.: On software parallel implementation of cryptographic pairings. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 35–50. Springer, Heidelberg (2009)
14. Rebeiro, C., Mukhopadhyay, D.: High speed compact elliptic curve cryptoprocessor for FPGA platforms. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 376–388. Springer, Heidelberg (2008)
15. Hankerson, D., Menezes, A., Scott, M.: Software implementation of pairings. In: Joye, M., Neven, G. (eds.) Identity-Based Cryptography (2008)
16. Devegili, A.J., Scott, M., Dahab, R.: Implementing cryptographic pairings over Barreto-Naehrig curves. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 197–207. Springer, Heidelberg (2007)

17. Barke, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management - part 1: General (revised). National Institute of Standards and Technology, NIST Special Publication 800-57 (2007)
18. Shu, C., Kwon, S., Gaj, K.: FPGA accelerated Tate pairing based cryptosystems over binary fields. In: FPT 2006, pp. 173–180 (2006)
19. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
20. Hess, F., Smart, N.P., Vercauteren, F.: The Eta pairing revisited. IEEE Transactions on Information Theory 52(10), 4595–4602 (2006)
21. Devegili, A., ÓhÉigeartaigh, C., Scott, M., Dahab, R.: Multiplication and squaring on pairing-friendly fields. Cryptology ePrint Archive, Report 2006/471 (2006)
22. Amanor, D.N., Paar, C., Pelzl, J., Bunimov, V., Schimmler, M.: Efficient hardware architectures for modular multiplication on FPGAs. In: International Conference on Field Programmable Logic and Applications 2005, pp. 539–542 (2005)
23. Chatterjee, S., Sarkar, P., Barua, R.: Efficient computation of Tate pairing in projective coordinate over general characteristic fields. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 168–181. Springer, Heidelberg (2005)
24. Galbraith, S.: Pairings. In: Blake, I.F., Seroussi, G., Smart, N.P. (eds.) Advances in Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series, ch. IX, Cambridge University Press, Cambridge (2005)
25. Miller, V.S.: The Weil pairing, and its efficient calculation. Journal of Cryptology 17, 235–261 (2004)
26. Bunimov, V., Schimmler, M.: Area and time efficient modular multiplication of large integers. In: ASAP 2003. IEEE Computer Society, Los Alamitos (2003)
27. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (2002)
28. Joye, M., Yen, S.M.: The Montgomery powering ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
29. Hauck, S., Hosler, M.M., Fry, T.W.: High-performance carry chains for FPGAs. In: FPGA 1998, pp. 223–233 (1998)
30. Blakley, G.R.: A computer algorithm for calculating the product A*B modulo M. IEEE Transactions on Computers C-32(5), 497–500 (1983)
31. Sloan, K.R.: Comments on a computer algorithm for calculating the product A*B modulo M. IEEE Transactions on Computers C-34(3), 290–292 (1985)