A High Speed Pairing Coprocessor Using RNS and Lazy Reduction

Gavin Xiaoxu Yao[1], Junfeng Fan[2],
Ray C.C. Cheung[1], and Ingrid Verbauwhede[2]

[1] Department of Electronic Engineering
City University of Hong Kong, Hong Kong SAR,
Email: Gavin.Yao@student.cityu.edu.hk, R.Cheung@cityu.edu.hk
[2] Katholieke Universiteit Leuven, ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
Email: {Junfeng.Fan, Ingrid.Verbauwhede}@esat.kuleuven.be

Abstract. In this paper, we present a high speed pairing coprocessor using Residue Number System (RNS) and lazy reduction. We show that combining RNS, which are naturally suitable for parallel architectures, and lazy reduction, which performs one reduction for more than one multiplication, the computational complexity of pairings can be largely reduced. The design is prototyped on a Xilinx Virtex-6 FPGA, which utilizes 7023 slices and 32 DSPs, and finishes one 254-bit optimal ate pairing computation in 0.664 ms.

Keywords: RNS, Moduli Selection, Hardware Implementation of Pairing, FPGA

1 Introduction

Pairing-Based Cryptography (PBC) has been utilised to provide efficient and elegant solutions to several long-standing problems in cryptography, such as three-way key exchange [23], identity-based encryption [9], identity-based signatures [10], and non-interactive zero-knowledge proof systems [19]. As cryptographic schemes based on pairings are introduced and investigated, the performance of pairing computation also receives increasing interests.

The complex pairing computation can be broken down to multiplications and additions in the underling fields. For example, one 256-bit optimal ate pairing consists of around ten thousand modular multiplications [1]. Thus, having an efficient multiplier is the key step to a high performance pairing processor. For pairings over ordinary curves defined over prime fields $\mathbb{F}_p$, Montgomery algorithm [31] is the most widely deployed reduction algorithm. Using Montgomery algorithm, the complexity of both multiplication and reduction is $O(s^2)$ using schoolbook method, or $O(s^{\log_2 3})$ using Karatsuba algorithm [25], where s is the number of machine-words to represent p.

Residue Number System (RNS) has a complexity of $O(s)$ for multiplication. In other words, multiplication is cheaper compared to the Montgomery algorithm. Besides, RNS distributes computation on a group of small integers, and are naturally suitable for parallel implementations. For this reason, RNS has been studied for cryptographic applications and Montgomery algorithm in RNS context was proposed [3, 26, 35, 38].

On the other hand, the Montgomery reduction in RNS has a higher complexity than ordinary Montgomery reduction. Fortunately, this overhead of slow reduction can be partially removed by reducing the number of reductions, also known as lazy reduction. Lazy reduction in pairing computation was introduced by Scott [36] and then generalised by Aranha *et al.* in [1]. In short, it performs one reduction for multiple multiplications. This is possible for expressions like $AB \pm CD \pm EF$ in $\mathbb{F}_p$. As such, lazy reduction brings a significant reduction of the complexity.

In this paper, we first illustrate how RNS moduli selection can reduce the complexity of modular reduction. Then, a pairing coprocessor is proposed using RNS multiplier. The major contributions of this work are as follows.

- We propose a set of base moduli to reduce the complexity of modular reduction in RNS.
- We present an efficient F_p multiplier using RNS Montgomery.
- We describe a high-speed pairing processor using the proposed multiplier.

This paper is organised as follows: Section 2 provides the backgrounds on RNS and pairing. Section 3 introduces a novel specification on RNS parameter selection to reduce the complexity of modular reduction, and optimised parameters are provided for pairing. In Section 4, hardware architecture of the pairing coprocessor is described. Section 5 illustrates how to perform pairing computation efficiently on the proposed architecture. Section 6 provides the experimental results and its analysis. In Section 7, we conclude the paper.

2 Backgrounds and Preliminaries

2.1 Bilinear Pairing

A bilinear pairing is a non-degenerate map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups and $\mathbb{G}_T$ is a subgroup of a multiplicative group. The core property of map e is linearity in both components, which allows the construction of novel cryptographic protocols. Popular pairings such as Tate pairing [4], ate pairing [22], R-ate pairing [28], optimal pairing [39] choose $\mathbb{G}_1$ and $\mathbb{G}_2$ to be specific cyclic subgroups of $E(\mathbb{F}_{p^k})$, and $\mathbb{G}_T$ to be a subgroup of $\mathbb{F}^*_{p^k}$.

The selection of parameters has an essential impact on the security and performance of a pairing computation. Not all the elliptic curves are suitable for pairings. Indeed, to achieve higher performance and security, a group of pairing-friendly curves are constructed. We refer to Freeman [16] for a summary of known pairing-friendly curves. Barreto and Naehrig [5] (BN) described a parameterised family of elliptic curves, and it is well-suited for computing asymmetric parings. BN-curves are defined with $E : y^2 = x^3 + b, b \neq 0$ over $\mathbb{F}_p$, where $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and n, the order of E, is $36u^4 + 36u^3 + 18u^2 + 6u + 1$.

Note that any $u \in \mathbb{Z}$ that generates prime p and n will suffice. BN-curves have embedding degree $k = 12$. Because of the limited space, we mainly focus on the discussion of optimal ate pairing on BN-curves.

Let $E' : y^2 = x^3 + b/\zeta$ be a sextic twist of E with ζ not a cube nor a square in $\mathbb{F}_{p^2}$, and $E[n]$ be the subgroup of n-torsion points of E, then the optimal ate pairing is defined as [1,32]:

$$a_{opt} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$$

$$(Q, P) \to (f_{r,Q}(P) \cdot l_{[r]Q,\pi_p(Q)}(P) \cdot l_{[r]Q+\pi_p(Q),-\pi_p^2(Q)}(P))^{\frac{p^{12}-1}{n}}$$

where $r = 6u + 2$. The group $\mathbb{G}_1 = E[n] \bigcap \mathrm{Ker}(\pi_p - [1]) = E(\mathbb{F}_p)[n]$ and $\mathbb{G}_2$ is the preimage $E'(\mathbb{F}_{p^2})[n]$ of $E[n] \bigcap \mathrm{Ker}(\pi_p - [p]) \subseteq E(\mathbb{F}_{p^{12}})[n]$ under the twisting isomorphism $\psi : E' \to E$. The group $\mathbb{G}_T$ is the subgroup of n-th roots of unity $\mu_n \subset \mathbb{F}^*_{p^{12}}$. The map $\pi_p : E \to E$ is the Frobenius endomorphism $\pi_p(x, y) = (x^p, y^p)$, and $f_{r,Q}(P)$ is a normalized function with divisor $(f_{r,Q}) = r(Q) - ([r]Q) - (r - 1)(\mathcal{O})$. The line function, $l_{Q_1,Q_2}(P)$, is the line arising in the addition of Q_1 and Q_2 evaluated at point P.

Miller [29] proposed an algorithm that constructs $f_{r,Q}$ in stages by suing double-and-add method. When u is selected as a negative integer, the corresponding Miller algorithm is given as Algorithm 1 [1].

Algorithm 1 Optimal ate pairing on BN curves for $u < 0$

```
Require: $P \in \mathbb{G}_1, Q \in \mathbb{G}_2, r = |6u + 2| = \sum_{i=0}^{\lfloor \log_2 r \rfloor} r_i 2^i$, where $u < 0$
Ensure: $a_{opt}(Q, P)$
 1: $T \leftarrow Q, f \leftarrow 1$
 2: for $i = \lfloor \log_2 r \rfloor - 1$ downto 0 do
 3:     $f \leftarrow f^2 \cdot l_{T,T}(P), T \leftarrow 2T$
 4:     if $r_i = 1$ then
 5:         $f \leftarrow f \cdot l_{T,Q}(P), T \leftarrow T + Q$
 6:     end if
 7: end for
 8: $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow \pi_p^2(Q)$
 9: $T \leftarrow -T, f \leftarrow f^{p^6}$
10: $f \leftarrow f \cdot l_{T,Q_1}(P), T \leftarrow T + Q_1$
11: $f \leftarrow f \cdot l_{T,-Q_2}(P), T \leftarrow T - Q_2$
12: $f \leftarrow f^{(p^6-1)(p^2+1)(p^4-p^2+1)/n}$
13: return $f$
```

Lazy reduction saves expensive modular reduction computation, especially in RNS context. Essentially, at least k reductions are required for a multiplication in $\mathbb{F}_{p^k}$, as the result has k coefficients. We deploy the same tower extension field as in [1], and only 12 reductions are required for a multiplication in $\mathbb{F}_{p^{12}}$:

– $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta)$, where $\beta = -1$

– $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \zeta)$, where $\zeta = 1 + i$
– $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$ or $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[W]/(W^6 - \zeta)$

Hence, an element $\alpha \in \mathbb{F}_{p^{12}}$ can be represented in any of the following three ways:

$$\begin{aligned}\alpha &= a_0 + a_1 w, \text{ where } a_0, a_1 \in \mathbb{F}_{p^6}\\ &= (a_{0,0} + a_{0,1}v + a_{0,2}v^2) + (a_{1,0} + a_{1,1}v + a_{1,2}v^2)w, \text{ where } a_{i,j} \in \mathbb{F}_{p^2}\\ &= a_{0,0} + a_{1,0}W + a_{0,1}W^2 + a_{1,1}W^3 + a_{0,2}W^4 + a_{1,2}W^5\end{aligned}$$

2.2 Residue Number System

A Residue Number System (RNS) represents a large integer with a set of smaller integers. Let $\mathfrak{B} = \{b_1, b_2, \ldots, b_s\}$ be a set of pairwise co-prime integers, and $M_{\mathfrak{B}} = \prod_{i=1}^{s} b_i$. The unique RNS representation of any integer X, $0 \leqslant X < M_{\mathfrak{B}}$, is $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_s\}$, where $x_i = |X|_{b_i}$, $1 \leqslant i \leqslant s$. Using RNS, arithmetic operations in $\mathbb{Z}/M_{\mathfrak{B}}\mathbb{Z}$ can be efficiently performed. Consider two integers X, Y and their RNS representations $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_s\}$ and $\{Y\}_{\mathfrak{B}} = \{y_1, y_2, \ldots, y_s\}$, then

$$\{|X \odot Y|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}} = \{|x_1 \odot y_1|_{b_1}, \ldots, |x_s \odot y_s|_{b_s}\}.$$

for $\odot \in \{+, -, \times, /\}$. The division is available only if Y is coprime with $M_{\mathfrak{B}}$, i.e. the multiplicative inverse of Y exists in $\mathfrak{B}$. The set $\mathfrak{B}$ is also known as a *base*, and the element b_i, $1 \leqslant i \leqslant s$, is called *RNS modulus* (pl. moduli). Each modulus forms a RNS *channel*, and the operation is carried out independently in each channel.

Also, the original value of X can be restored from $\{X\}_{\mathfrak{B}}$ using the Chinese Remainder Theorem (CRT):

$$X = \left| \sum_{i=1}^{s} \left| x_i \cdot B_i^{-1} \right|_{b_i} \cdot B_i \right|_{M_{\mathfrak{B}}} \tag{1}$$

where

$$B_i = \frac{M_{\mathfrak{B}}}{b_i} = \prod_{j=1, j \neq i}^{s} b_j, \ 1 \leqslant i \leqslant s. \tag{2}$$

What makes RNS extremely interesting is the natural ease for parallel implementations. For all the basic operations $(+, -, \times, /)$, computations between x_i and y_i have no dependency on other channels. Besides, the complexity of multiplication is $O(s)$, while the complexity of naive multiplication is $O(s^2)$, and that of Karatsuba multiplication is $O(s^{\log 3/\log 2})$ [25,30][3,4]. In order to simplify

[3] For the simplicity of the paper, we only use naive multiplication method to count the number of multiplications and complexity analysis in the remaining paper. Readers can easily derive the results using Karatsuba method.

[4] In practice, the number of modulus in a RNS base and the number of words in normal binary system to represent field size p are equivalent in the complexity analysis, and we use s to denote both these parameters.

channel reduction, pseudo-Mersenne numbers of the form $b_i = 2^w - \alpha$, where $\alpha < 2^{\frac{w}{2}}$ are chosen as moduli [37].

2.3 RNS Montgomery Reduction

Algorithm 2 shows the Montgomery modular reduction algorithm in RNS context, and for comparison reason, redundant Montgomery algorithm is also provided. RNS Montgomery algorithm is consistent with the redundant Montgomery algorithm without conditional subtraction in [31,40], with the parameter R setting to $M_{\mathfrak{B}}$. Since R^{-1} (i.e. $M_{\mathfrak{B}}^{-1}$) does not exist in base $\mathfrak{B}$, a new base, $\mathfrak{C} = \{c_1, c_2, \ldots, c_s\}$, where $M_{\mathfrak{C}}$ is co-prime with $M_{\mathfrak{B}}$, is introduced to perform the division (i.e. $(T + QN)/R$). Note that all the moduli from both $\mathfrak{B}$ and $\mathfrak{C}$ are pairwise coprime as $M_{\mathfrak{B}}$ and $M_{\mathfrak{C}}$ are coprime. The overhead is two base extensions required in Algorithm 2.

Algorithm 2 RNS Montgomery Reduction [26] and Redundant Montgomery Reduction [40]

```
Require: RNS bases $\mathfrak{B}$ and $\mathfrak{C}$ with $M_{\mathfrak{B}}, M_{\mathfrak{C}} > 2N$
Require: $N$, $M_{\mathfrak{B}}$, $M_{\mathfrak{C}}$ are pairwise coprime
Require: $\{X\}_{\mathfrak{B}}$, $\{X\}_{\mathfrak{C}}$, $\{Y\}_{\mathfrak{B}}$, $\{Y\}_{\mathfrak{C}}$ with $X, Y < 2N$
  Precompute: $\{N'\}_{\mathfrak{B}} \leftarrow \{|-N^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$
  Precompute: $\{M'\}_{\mathfrak{C}} \leftarrow \{|M_{\mathfrak{B}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$ and $\{N\}_{\mathfrak{C}}$
Ensure: $\{U\}_{\mathfrak{B}}$, $\{U\}_{\mathfrak{C}}$ s.t. $|U|_N = |XYM_{\mathfrak{B}}^{-1}|_N$, $U < 2N$

    RNS Montgomery                                                   Redundant Montgomery
  1 $\{Q\}_{\mathfrak{B}} \leftarrow \{T\}_{\mathfrak{B}} \times \{N'\}_{\mathfrak{B}}$                              $Q \leftarrow \left||T|_R \times N'\right|_R$
  2 $\{Q\}_{\mathfrak{B}} \xrightarrow{Base\ Extension\ 1} \{Q\}_{\mathfrak{C}}$
  3 $\{U\}_{\mathfrak{C}} \leftarrow \left(\{T\}_{\mathfrak{C}} + \{Q\}_{\mathfrak{C}} \times \{N\}_{\mathfrak{C}}\right) \times \{M'\}_{\mathfrak{C}}$      $U \leftarrow (T + QN)/R$
  4 $\{U\}_{\mathfrak{B}} \xleftarrow{Base\ Extension\ 2} \{U\}_{\mathfrak{C}}$
```

The operation to transform the representation in one RNS base to that in other base is called Base Extension (BE). To compute $\{X\}_{\mathfrak{C}} = \{x'_1, x'_2, \ldots, x'_s\}$ given $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_s\}$, one can use Posch-Posch Method [26, 34].

Given $\{X\}_{\mathfrak{B}}$, for Eqn(1), there must exist certain integer $\gamma < s$ such that:

$$X = \left| \sum_{i=1}^{s} \left| x_i \cdot B_i^{-1} \right|_{b_i} \cdot B_i \right|_{M_{\mathfrak{B}}} = \left| \sum_{i=1}^{s} \xi_i \cdot B_i \right|_{M_{\mathfrak{B}}} = \sum_{i=1}^{s} \xi_i \cdot B_i - \gamma \cdot M_{\mathfrak{B}} \qquad (3)$$

where $\xi_i = \left| x_i \cdot B_i^{-1} \right|_{b_i}$, $1 \leqslant i \leqslant s$. In Posch-Posch method, γ can be calculated by the following equation:

$$\gamma = \left\lfloor \sum_{i=1}^{s} \frac{\xi_i \cdot B_i}{M_{\mathfrak{B}}} \right\rfloor = \left\lfloor \sum_{i=1}^{s} \frac{\xi_i}{b_i} \right\rfloor \qquad (4)$$

In [26], ξ_i/b_i is further approximated by $\xi_i/2^w$ as b_i is chosen as a pseudo-Mersenne number near 2^w. Once γ is obtained, $\{X\}_{\mathfrak{C}} = \{x'_1, \ldots, x'_s\}$ can be computed as follows:

$$x'_j = \left| \sum_{i=1}^{s} \xi_i \cdot B_i - \gamma \cdot M_{\mathfrak{B}} \right|_{c_j} = \left| \sum_{i=1}^{s} \xi_i \cdot |B_i|_{c_j} - \gamma \cdot |M_{\mathfrak{B}}|_{c_j} \right|_{c_j} . \tag{5}$$

The $|B_i|_{c_j}$ and $|M_{\mathfrak{B}}|_{c_j}$, $1 \leqslant i, j \leqslant s$, can be precomputed once $\mathfrak{B}$ and $\mathfrak{C}$ are fixed.

Lazy reduction performs only one reduction for patten like $AB \pm CD \pm EF$, hence, it trades expensive reduction with multi-precision addition. In RNS context, as multiplication takes only $2s$ word operations, lazy reduction dramatically decreases the complexity. For instance, to compute $AB \pm CD \pm EF$, it takes $2s^2 + 11s$ word multiplications in RNS, while it takes $4s^2 + s$ using digit-serial Montgomery modular multiplication [27].

3 RNS and Pairing Parameter Selection

3.1 Specifications on RNS Base Selection

The selection of RNS base has a huge impact on the complexity of reduction. Clearly, one should choose pseudo-Mersenne numbers as the RNS moduli, since the reduction by each modulus will be largely simplified. We notice that the complexity of RNS modular reduction can be reduced when the moduli are near the same power of 2 and close to each other.

The BE step [Eqn(5)] is the most computational expensive operation in the RNS Montgomery algorithm. It requires around $s^2 + s$ word-size multiplications. Eqn(5) is essentially a matrix multiplication followed by reduction.

$$\begin{pmatrix} x''_1 \\ \vdots \\ x''_s \end{pmatrix} := \begin{pmatrix} |B_1|_{c_1} & \cdots & |B_s|_{c_1} \\ \vdots & \ddots & \vdots \\ |B_1|_{c_s} & \cdots & |B_s|_{c_s} \end{pmatrix} \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_s \end{pmatrix} - \gamma \begin{pmatrix} |M_{\mathfrak{B}}|_{c_1} \\ \vdots \\ |M_{\mathfrak{B}}|_{c_s} \end{pmatrix} \tag{6}$$

$$\{X\}_{\mathfrak{C}} \equiv \{x'_1, \ldots, x'_{s'}\} := \{|x''_1|_{c_1}, \ldots, |x''_{s'}|_{c_s}\} \tag{7}$$

The computational intensiveness comes from Eqn(6), which involves s^2 word-size multiplications. Note the element in the matrix $|B_i|_{c_j}$, $1 \leqslant i, j \leqslant s$, can be computed as follows:

$$|B_i|_{c_j} = \left| \prod_{k=1, k\neq i}^{s} b_k \right|_{c_j} = \left| \prod_{k=1, k\neq i}^{s} (b_k - c_j) \right|_{c_j} \tag{8}$$

Define $\tilde{B}_{i,j} := \prod_{k=1,k\neq i}^{s}(b_k - c_j)$. When the moduli b_k and c_j are close to each other, the difference $b_k - c_j$ is small. In practice, the absolute value of $\tilde{B}_{i,j}$ could be much less than the value of c_j ($\tilde{B}_{i,j}$ could be a negative number). In other words, if $\tilde{B}_{i,j}$ substitutes $|B_i|_{c_j}$ to perform the matrix multiplication, the bit-length of the operand is much less than the original word-size. The substitution of $|B_i|_{c_j}$ with $\tilde{B}_{i,j}$ will not affect the final result as the possible negative result is corrected by the following Eqn(7).

Furthermore, there could be at most one even number among all the moduli because all the moduli are pairwise co-prime. For an odd modulus c_j, $2^{s-1}|\tilde{B}_{i,j}$, because there is at most one $(b_k - c_j)$ term cannot be divided by 2. In practice, the number of zeros at the least significant bits are usually more than $s - 1$, because $b_k - c_j$ can be 2^i, where $i > 1$. Hence, we can further save the bit-length by truncation of the least significant zeros, and compensate the result by left-shifting. We denote $\tilde{B}_{i,j}$ after truncation of zeros as $\tilde{B}'_{i,j}$.

As the bit-length of $\tilde{B}'_{i,j}$ is much shorter than the standard word-size, one can employ asymmetric multiplier whose inputs are of different bit-length. The asymmetric multiplier takes less area and is faster than the symmetric standard word-size multiplier (cf. Sec. 4.1). We denote the bit-length of all $\tilde{B}_{i,j}$ in matrix format as $L_{\tilde{B}}$, and the bit-length matrix of all $\tilde{B}'$ as $L_{\tilde{B}'}$. The mechanism of denotation also works for base $\mathfrak{C}$, and one operand size of the asymmetric multiplier inputs is shown by $L_{\tilde{B}'}$ and $L_{\tilde{C}'}$.

3.2 RNS Parameter Selection

To achieve 128-bit security level, p should be around 256 bits, where $\mathbb{F}_{p^k}$ is the operating field. We set s, the number of moduli in each base to be 8 and the moduli are chosen near 2^{33} considering the following facts:

- The channel multipliers should be symmetric, because the inputs are of the same bit-length for all channel multiplications except the matrix multiplications in BEs.
- In modern FPGA device, DSP slices with 2's complement 25×18 multipliers are embedded [41]. To construct a symmetric multiplier using these slices, the standard word-size (i.e. the bit-length of a modulus) should be equal to or a little bit less than a multiple of 17 excluding the sign bit.
- $s = 8$ are the most suitable trade-off. For $s = 4$, the parallelism is undermined, and it needs 9 DSP slices to construct a 51×51 multiplier. For $s = 16$, it takes $s = 16$ loops to perform one matrix multiplication in cox-rower design [26], while for $s = 8$, it takes half less loops.
- We select moduli near 2^{33} because some of the moduli are a slightly greater than 2^{33}, and hence all the moduli fit in 34-bit multiplier. If one chooses all the moduli less than a power of 2, some moduli will have a large Hamming weight, which implies the channel reduction is expensive.

As the number of moduli and moduli range are decided, we choose the following bases.

$$\mathfrak{B} = \{2^w - 1, 2^w - 9, 2^w + 3, 2^w + 11, 2^w + 5, 2^w + 9, 2^w - 31, 2^w + 15\}$$
$$\mathfrak{C} = \{2^w, 2^w + 1, 2^w - 3, 2^w + 17, 2^w - 13, 2^w - 21, 2^w - 25, 2^w - 33\}$$

where $w = 33$. Consequently,

$$L_{\tilde{B}} = \begin{pmatrix} 23 & 20 & 21 & 20 & 21 & 20 & 18 & 19 \\ 22 & 20 & 22 & 20 & 21 & 20 & 18 & 19 \\ 25 & 23 & 23 & 22 & 23 & 22 & 21 & 22 \\ 25 & 24 & 25 & 26 & 25 & 26 & 23 & 28 \\ 29 & 30 & 28 & 28 & 28 & 28 & 28 & 27 \\ 32 & 33 & 32 & 31 & 31 & 31 & 33 & 31 \\ 32 & 33 & 32 & 32 & 32 & 32 & 34 & 32 \\ 33 & 33 & 33 & 32 & 33 & 33 & 37 & 32 \end{pmatrix}, \; L_{\tilde{C}} = \begin{pmatrix} 24 & 23 & 23 & 20 & 21 & 20 & 20 & 19 \\ 25 & 25 & 26 & 24 & 26 & 25 & 24 & 24 \\ 26 & 27 & 25 & 24 & 24 & 23 & 23 & 23 \\ 30 & 31 & 30 & 31 & 29 & 29 & 29 & 28 \\ 28 & 28 & 27 & 27 & 26 & 26 & 26 & 25 \\ 30 & 30 & 30 & 30 & 29 & 28 & 28 & 28 \\ 27 & 27 & 27 & 26 & 28 & 29 & 29 & 31 \\ 30 & 30 & 30 & 33 & 29 & 29 & 29 & 29 \end{pmatrix}$$

After truncation of least significant zeros of $\tilde{B}_{i,j}$, we get:

$$L_{\tilde{B}'} = \begin{pmatrix} 23 & 20 & 21 & 20 & 21 & 20 & 18 & 19 \\ 20 & 18 & 20 & 18 & 19 & 18 & 16 & 17 \\ 22 & 20 & 20 & 19 & 20 & 19 & 18 & 19 \\ 18 & 17 & 18 & 19 & 18 & 19 & 16 & 21 \\ 23 & 24 & 22 & 22 & 22 & 22 & 22 & 21 \\ 22 & 23 & 22 & 21 & 21 & 21 & 23 & 21 \\ 23 & 24 & 23 & 23 & 23 & 23 & 24 & 23 \\ 20 & 20 & 20 & 19 & 20 & 20 & 24 & 19 \end{pmatrix}, \; L_{\tilde{C}'} = \begin{pmatrix} 24 & 23 & 23 & 20 & 21 & 20 & 20 & 19 \\ 23 & 23 & 24 & 22 & 24 & 23 & 22 & 22 \\ 23 & 24 & 22 & 21 & 21 & 20 & 20 & 20 \\ 23 & 24 & 23 & 24 & 22 & 22 & 22 & 21 \\ 22 & 22 & 21 & 21 & 20 & 20 & 20 & 19 \\ 20 & 20 & 20 & 20 & 19 & 18 & 18 & 18 \\ 18 & 18 & 18 & 17 & 19 & 20 & 20 & 22 \\ 21 & 21 & 21 & 24 & 20 & 20 & 20 & 20 \end{pmatrix}$$

Notice that all of $\tilde{B}'_{i,j}$ and $\tilde{C}'_{i,j}$ are less than 25 bits, and they fit in one operand of FPGA DSP slice, while the standard 34-bit operands do not. For the ease of implementation, we do not truncate all the least significant zeros. Instead, all the $\tilde{B}_{i,j}$ in the same row are truncated with the same number of zeros, and we do not truncate more zeros when all elements in a row fit in 25 bits.

3.3 Pairing Parameter Selection

Lazy reduction reduces complexity in RNS (cf. Sec. 2.3). In order to perform lazy reduction, the allowed range for input to reduction should be big enough. For instance, to compute $AB \pm CD \pm EF$, the allowed input should be as big as $3p^2$ instead of p^2. In normal binary number system, one can perform conditional correction to keep the sum always in the operating range, and the redundancy is no need too much. However, in RNS, as the comparison of two number values are prohibitive computational intensive, one can hardly tell whether to perform the conditional correction. Hence, the actual operating range should have enough redundancy to perform lazy reduction.

According to our observation, the operating range should be greater than $22p^2$ (the computation of $f_{0,0}$ requests the largest operating range, c.f. Sec.5). Furthermore, the redundant Montgomery algorithm along with the approximation of ξ_i/b_i in Eqn(4) requires 9 times of redundancy [20, 26]. Thus, $M_{\mathfrak{B}} = \prod_{i=1}^{s} b_i$, where $b_i \in \mathfrak{B}$, should be greater than $9 \cdot 22 \cdot p$ [20].

For demonstration, we choose $u = -(2^{62} + 2^{55} + 1)$, and consequently, p is a 254-bit prime and $198p < M_{\mathfrak{B}}$ holds. We admit the security level is slightly weak than 128-bit. However, the little compromise will gain great performance improvement. Also as $p \equiv 3 \bmod 4$, multiplications by $\beta = -1$ can be computed as simple subtractions in $\mathbb{F}_p^2$ [33].

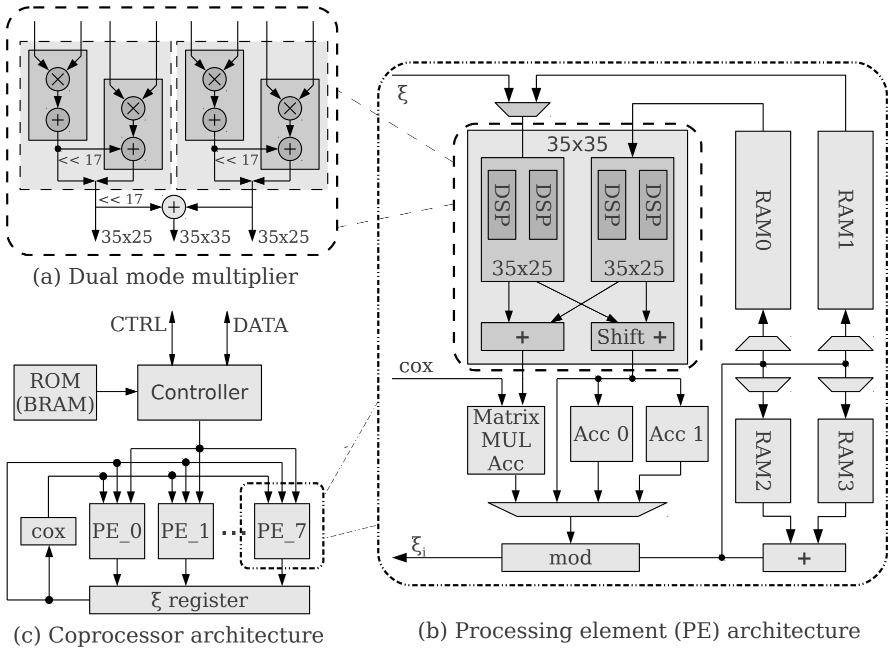

Fig. 1. Pairing coprocessor hardware architecture

4 Coprocessor Architecture Design

Fig. 1 shows the whole hardware architecture of the proposed pairing coprocessor, which contains 8 processing elements (PEs) and each PE employs one dual mode multiplier.

4.1 Dual Mode Multiplier

Using the DSP slices embedded in FPGA can improve the performance dramatically. The dominating DSP slice in advanced FPGAs is made of a signed 25×18 bit multiplier, a 17-bit left shifter, an accumulator, and other processing elements. Excluding the sign bit, it is actually a 24×17 multiplier. Two such DSPs can form a signed 35×25 multiplier, and four DSPs can form a signed 35×35 multiplier in a speed-optimised fashion (in fact, 4 DSPs can form an 35×42 multiplier without external logic, but it is useless in this work).

Fig. 1(a) illustrates the hardware architecture of the dual mode multiplier. One such multiplier is made of 4 DSP slices, and can work either as a 35×35 multiplier, or two 35×25 multipliers. The partial products from DSP slices are added by the accumulator inside of DSP slices, while the partial products after addition (i.e. the full products of 35×25) are added in an adder tree fashion. The adder tree balances the timing of the two 35×25 multiplier, and accelerates the partial product summation.

The advantages of the dual mode multiplier are:

– The same hardware for standard word-size multiplication can perform two matrix multiplication at the same time, which saves area.
– It saves half of the matrix multiplication time as the two multiplications are taken at the same time.
– The time to perform 35×25 multiplication is shorter than that of 35×35 multiplication, as there are less partial products to accumulate.

4.2 Processing Elements

Each Processing Element (PE) preforms operations in one channel of $\mathfrak{B}$ and one channel of $\mathfrak{C}$. Fig. 1(b) shows the internal design of a PE. There is a dual mode multiplier, an adder, 2 RAMs for multiplier inputs and 2 RAMs for adder inputs, 3 accumulators, and a channel reduction module. In order to avoid data fetch latency, two RAMs are attached to the dual mode multiplier and the two operands are supplied simultaneously. As addition is also an important channel operation, dedicated adder and RAMs are designed. The two 35×25 products are added directly as such mode is dedicated to the matrix multiplication step, and all the products should be added together finally in matrix multiplication.

One accumulator is dedicated for matrix multiplication of the base extension step, which can shift to compensate the truncated zeros, and handle cox value (cf. Alg. 3). The other two accumulators are to compute polynomial multiplication. As modular reduction will not takes simultaneously, only one modular reduction module is employed for each PE.

4.3 Coprocessor Architecture

We deploy the cox-rower architecture model [26]. The design of pairing coprocessor is showed in Fig. 1(c). There are $s = 8$ PEs, and each PE performs the operations in its channel. The module *cox*, named by the authors of [26], is an accumulator to provide result correction information for all PEs in the BE operation. The ξ register receives the ξ values from every PE and delivers two ξ values to all PEs at a time.

A ROM is deployed in the top level, which stores the assembly code for all PEs. The controller reads these instructions from the ROM, distributes the control signals to all PEs and ξ registers, and renews the instruction pointer. One can program his own code to control the data flow. This provides high flexibility to different applications. Indeed, one can perform almost all algorithms on $\mathbb{F}_p$ where p is around 256-bit, not only optimal ate pairing on BN curves, but also other pairing on other curves or ECC.

5 Pairing Computation on the Proposed Coprocessor

5.1 RNS Modular Reduction

For modular reduction, the cox-rower algorithm [26] is adopted. Using the dual mode multiplier, each PE performs two element multiplications at a time for matrix multiplication in the BE operation. Hence, the cox-rower algorithm is modified as Algorithm 3. In this algorithm, Step 4-6 distribute the computation of γ in Eqn(6) to each loop, and the operations are the same for all the channels. The *cox* module performs these accumulations and provides value l to each PE for result correction. As the number of loops is reduced from s to $s/2$, the modular reduction operation is essentially accelerated.

Algorithm 3 Cox-rower algorithm for k-th PE with two element multiplications at a time

```
Require: $|X|_{b_k}$
Ensure: $|X|_{c_k}$
    Precompute: $B_k^{-1}, \tilde{B}'_{i,k}, 1 \leqslant i \leqslant s, |M_{\mathfrak{B}}|_{c_k}, \text{bias}_{c_k}$
1: $\xi_k \leftarrow x_k \cdot B_k^{-1}$
2: $y = 0, z = 0$
3: for $i = 0$ to $(s/2 - 1)$ do
4:     $y \leftarrow y + \xi_i + \xi_{i+s/2}$
5:     $l \leftarrow \lfloor y/2^w \rfloor$, $\{l \in 0, 1, 2\}$
6:     $y \leftarrow |y|_{2^w}$
7:     $z \leftarrow z + (\xi_i \cdot \tilde{B}'_{i,k} + \xi_{i+s/2} \cdot \tilde{B}'_{(i+s/2),k}) \cdot 2^{\text{bias}_{c_k}} - l \cdot |M_{\mathfrak{B}}|_{c_k}$
8: end for
9: return $z$
```

5.2 Back to Schoolbook Method for Field Operations

Karatsuba-like method has been used intensively in field operations in literature to save the expensive multiplication [1, 6, 21, 36]. For a $\mathbb{F}_{p^2}$ multiplication, it is computed as follows:

$$\begin{aligned}(a_0 + a_1 i)(b_0 + b_1 i) &= (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0) i \\ &= (a_0 b_0 - a_1 b_1) + [(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - b_0 b_1] i \qquad (9)\end{aligned}$$

In Eqn(9), 3 multiplications are deployed instead of 4 using schoolbook or naive method, but the overhead is 3 more additions. In normal positional nubmer system, this method saves computation power, as multiplication, with complexity of $O(s^2)$, is much more expensive than addition, whose complexity is $O(s)$. However, in RNS, the complexities of multiplication and addition are the same. Both multiplication and addition take $2s$ operations. Hence, the schoolbook method involves less operations (counting both additions and multiplications) and is employed.

Furthermore, accumulator after multiplier can be deployed to hide the latency of products addition, reduce memory access, and simplify control. Also, Karatsuba method requires larger redundancy (for the $\mathbb{F}_{p^2}$ example, operating range is enlarged from $2p^2$ to $6p^2$), which will decrease the security level given fixed operating range.

For $\mathbb{F}_{p^{12}}$ multiplication, the schoolbook method also provides an elegant solution. Using lazy reduction, one multiplication in $\mathbb{F}_{p^{12}}$ needs only 12 reductions and it is computed as follows:

$$f \cdot g = \sum_{j+k<6} f_j \cdot g_k \cdot W^{j+k} + (\sum_{j+k \geqslant 6} f_j \cdot g_k) \cdot \zeta \cdot W^{j+k-6} \tag{10}$$

where $f, g \in \mathbb{F}_{p^2}[W]/(W^6 - \zeta)$, and $f_j, g_k \in \mathbb{F}_{p^2}, 1 \leqslant j, k \leqslant 6$, are the coefficients of f, g, respectively. The two coefficients of the intermediate results of $f_j \cdot g_k$ are less than $2p^2$, and the coefficients of $f_j \cdot g_k \cdot \zeta$ is less than $4p^2$.

According to Eqn(10), $f_j \cdot g_k \cdot \zeta$ is one fundamental operation (such kind of computation also exists in point doubling and addition formulas). Let $f_j = f_0 + f_1 \cdot i$ and $g_k = g_0 + g_1 \cdot i$, then

$$f_j \cdot g_k \cdot \zeta = (f_0 g_0 - f_1 g_1 - f_0 g_1 - f_1 g_0) + (f_0 g_0 - f_1 g_1 + f_0 g_1 + f_1 g_0) \cdot i$$

Note that all the four products, $f_0 g_0, f_1 g_1, f_0 g_1, f_1 g_0$, are used to compute both the results, and the two accumulator design can calculate the two results at the same time with the same product inputs.

5.3 High Level Operations in Pairing Algorithm

Miller Loop We adopt the homogeneous coordinates proposed in [11]. Again, we use schoolbook method to avoid more additions. Table 1 provides the formulas for point doubling, addition and line raising, along with the pipeline groups and operation counts. M, S, R represent square, multiplication and reduction in $\mathbb{F}_{p^2}$, respectively, and m, r represent multiplication and reduction in $\mathbb{F}_p$. The cost of square in $\mathbb{F}_p$ is also denoted as m as there is no dedicated hardware to optimize it. As we use schoolbook method, $M = 4m$, $S = 3m$, and $R = 2r$. Each pipeline group has $5R$ or $6R$, and this way of grouping makes the best effort to squeeze the pipeline bubbles.

Finial Steps The finial steps of an optimal pairing include Final Addition (FA, Line 10, 11 in Alg. 1) and Final Exponentiation (FE, Line 12 in Alg. 1). The operation count is given in Table 2. The formulas for FA are the same with the point addition formulas in the Miller loop, but for the second step, only line raising is kept and the computation for $T \leftarrow T - Q_2$ is skipped because the point value will not be used any more. The Frobenius endomorphism of Q_1, Q_2 computation is also included in FA count in Table 2.

For FE step, the power $(p^{12} - 1/n)$ is factored into three small exponents: $(p^6 - 1)$, $(p^2 + 1)$, and $(p^4 - p^2 + 1)/n$. To compute $(p^6 - 1)$, it requires an inversion. The formulas from [36] is utilized, and we find that when the computation of inversion and multiplication is integrated, the multiplication turns to a square. The explicit formula is given as Algorithm 4. Eventually, one needs to perform an inversion in $\mathbb{F}_p$. We employ Fermats little theorem, i.e. $d^{-1} = d^{p-2} \bmod p$

Table 1. Pipeline design and operation count of Miller loop

Condition	Step	Operations	Count
$r_i = 0$	1	$A = y_1^2$, $B = 3b'z_1^2$, $C = 2x_1y_1$ $D = 3x_1^2$, $E = 2y_1z_1$	$3S + 2M + 5R$
	2	$x_3 = (A - 3B)C$, $y_3 = A^2 + 6AB - 3B^2$, $z_3 = 4AE$ $l_0 = (B - A)\zeta$, $l_3 = \overline{y_P}E$, $l_4 = x_PD$	$2S + 3M + 4m + 5R$
	3	$f = f^2$	$6S + 15M + 6R$
	4	$f = f \cdot l$	$18M + 6R$
$r_i = 1$	1	$A = y_1^2$, $B = 3b'z_1^2$, $C = 2x_1y_1$ $D = 3x_1^2$, $E = 2y_1z_1$, $f_0 = (f^2)_0$	$5S + 4M + 6R$
	2	$x_3 = (A - 3B)C$, $y_3 = A^2 + 6AB - 3B^2$, $z_3 = 4AE$ $l_0 = (B - A)\zeta$, $l_3 = \overline{y_P}E$, $l_4 = x_PD$, $f_1 = (f^2)_1$	$2S + 6M + 4m + 6R$
	3	$A = y_3 - y_Qz_3$, $B = x_3 - x_Qz_3$ $f_{2,3,4,5} = (f^2)_{2,3,4,5}$	$4S + 12M + 4m + 6R$
	4	$f = f \cdot l$	$18M + 6R$
	5	$C = A^2$, $D = B^2$ $l_0 = (x_QA - y_QB)\zeta$, $l_3 = y_PB$, $l_4 = \overline{x_P}A$	$2S + 2M + 4m + 5R$
	6	$C = z_3C$, $E = BD$, $D = x_3D$ $f_{0,1,2} = (f \cdot l)_{0,1,2}$	$12M + 6R$
	7	$x_3 = B(E + C - 2D)$, $y_3 = A(3D - E - C) - y_3E$ $z_3 = z_3E$, $f_{3,4,5} = (f \cdot l)_{3,4,5}$	$13M + 6R$

if p is prime. This is because the proposed coprocessor has no ability to perform extended Euclidean algorithm or other algorithms which have less complexity, otherwise new module dedicated for these algorithms should be developed. The problem with this exponentiation in $\mathbb{F}_p$ is that the computation cannot be pipelined. Indeed, only one stage is taken out of all pipeline stages, which causes low pipeline occupation rate and a huge waste (c.f. Sec. 6).

The formula for the last exponentiation comes from [12]. In a recently paper [1], using cyclotomic subgroup structure to accelerate this *hard* exponentiation is proposed. However, it requires three more inversions, and as we stated, inversion is very expensive and inefficient on our platform, hence this method is not used.

6 Experimental Results and Discussion

6.1 Experimental Results

The prototype of the prposed pairing coprocessor design is implemented on Xilinx Virtex-6 XC6VLX240T-2 FPGA. The tool for synthesise, simulation and implementation is Xilinx ISE 12.4.

The logic utilisation is provided by Table 3. As there are 8 PEs and each PE contains 4 DSP slices, the total number of DSP is 32. The block RAM

Algorithm 4 Computation of f^{p^6-1} in $\mathbb{F}_{p^{12}}$

```
Require: $f \in \mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[W]/(W^6 - \zeta)$, and $f = \sum_{j=0}^{5} f_j \cdot W^j$, where $f_j \in \mathbb{F}_{p^2}$
Ensure: $f^{p^6-1}$
1: $f \leftarrow \sum_{j=0}^{5} (-1)^j \cdot f_j \cdot W^j$
2: $a \leftarrow f_0^2 + (-2f_1f_5 + 2f_2f_4 - f_3^2)\zeta$
   $b \leftarrow 2f_0f_2 - f_1^2 + (-2f_3f_5 + f_4^2)\zeta$
   $c \leftarrow 2f_0f_4 - 2f_1f_3 + f_2^2 - f_5^2\zeta$
3: $A \leftarrow a^2 - bc\zeta$, $B \leftarrow c\zeta - ab$, $C \leftarrow b^2 - ac$
4: $F = F_0 + F_1 i \leftarrow aA + (bC + cB)\zeta$, where $F_0, F_1 \in \mathbb{F}_p$
5: $d \leftarrow F_0^2 + F_1^2$
6: $d \leftarrow d^{-1}$
7: $F \leftarrow d \cdot F_0 - d \cdot F_1 \cdot i$
8: $f \leftarrow f^2 \cdot (A \cdot F + B \cdot F \cdot W^2 + C \cdot F \cdot W^4)$
9: return $f$
```

Table 2. Operation count of final steps

Step	Operations	Operation Count
FA1	$Q_1 \leftarrow \pi_p(Q)$	$M + 2m + R$
	$T \leftarrow T + Q_1$ and $l_{T,Q_1}(P)$	$2S + 11M + 8m + 13R$
	$f \leftarrow f \cdot l_{T,Q_1}(P)$	$18M + 6R$
FA2	$Q_2 \leftarrow \pi_p(Q_1)$	$M + 2m + R$
	$l_{T,-Q_2}(P)$	$4M + 8m + 5R$
	$f \leftarrow f \cdot l_{T,-Q_2}(P)$	$18M + 6R$
f^{p^6-1}	Before d^{-1}	$9S + 12M + 2m + 7R + r$
	$d^{-1} = d^{p-2}$	$294m + 294r$
	After d^{-1}	$6S + 36M + 16R$
f^{p^2+1}	f^{p^2+1}	$\tilde{M} + \tilde{R} + 8m + 4R$
$f^{p^4-p^2+1/n}$	$a \leftarrow f^{6\|u\|-5}$	$64\tilde{S} + 4\tilde{M} + 68\tilde{R}$
	$b \leftarrow a^{p+1}$	$\tilde{M} + \tilde{R} + 3M + 4m + 5R$
	f^p, f^{p^2}, f^{p^3}	$6M + 12m + 12R$
	$T \leftarrow b \cdot (f^p)^2 \cdot f^{p^2}$	$2\tilde{M} + \tilde{S} + 3\tilde{R}$
	$T \leftarrow T^{6u^2+1}$	$126\tilde{S} + 12\tilde{M} + 138\tilde{R}$
	$f^{p^3} \cdot T \cdot b \cdot (f^{p+1})^9 \cdot f^4$	$7\tilde{M} + 5\tilde{S} + 12\tilde{R}$

(BRAM) serves as the ROM and stores the assembly code. With fine tuned pipeline steps, the coprocessor can operate at 250MHz. The pipeline depth is 8 cycles (bypass the accumulator), or 9 cycles (with accumulation). With latency hidden by pipeline, a multiplication only takes 2 cycles and a reduction takes 13 cycles. Due to the addition at Step 3 in Alg. 2, there will be an idle cycle for each pipeline group. Note that it would take 21 cycles for reduction without dual mode multiplier, and our design saves 38% of the time for reduction with the same hardware.

Table 4 gives the number of multiplications and reductions in $\mathbb{F}_p$ for all the computations employed in Miller loop. The overhead of additions is hidden by the

Table 3. Logic Utilization

Logic Utilization	Used	Available	Utilization
# DSP48E1s	32	2,016	1%
# Slice Register	19,194	595,200	3%
# Slice LUTs	25,618	297,600	9%
# Occupied Slices	7,032	74,400	9%
# 18Kb BRAM	48	832	6%

Table 4. Number of operations and cycles per computation in Miller loop

	2T and $l_{T,T}(P)$	T+Q and $l_{T,Q}(P)$	f^2	$f \cdot l$	Miller's loop	
					Ate	Optimal
#Multiplication	39	54	78	72	-	-
#Reduction	20	26	12	12	-	-
#Cycles	340	456	313	301	128531	64084

Table 5. Number of operations and cycles per computation in final steps

Step	Operation	# Cycles	# Idle Cycles	Occupation Rate
Final Addition	FA1	799	7	99.1%
	FA2	566	50	91.2%
f^{p^6-1}	d^{p-2} mod p	25537	21127	17.3 %
	Others	1333	240	82.0%
f^{p^2+1}	f^{p^2+1}	573	9	98.4%
$f^{p^4-p^2+1/n}$	$f^{6\|u\|-5}$	21812	68	99.7%
	T^{6u^2+1}	44778	138	99.7%
	Others	6905	47	99.3%
Total	Ate	100578	21629	78.5%
	Optimal	101943	21686	78.7%

accumulators and carefully-designed pipeline input sequence. The cycle counting for each operation are also shown in Table 4. Out of the 64084 cycles, the multiplier is left idle for only 268 cycles. In other words, the pipeline occupation rate is 99.58%. Table 5 provides the timing information for the final exponentiation. The efficiency is very poor in the inversion operation, as most of the pipeline is left idle and the ξ delivery latency cannot be hidden in this stage. However, the occupation rate for other steps is high, especially for multiplication and square in $\mathbb{F}_{p^{12}}$.

6.2 Comparison and Discussion

Table 6 lists the performance of software and hardware implementation reported in recent literature. Compared with the other hardware implementations [15,17,24], our design achieves a speed-up of factor 3 at least. This design is the first hardware work which computes a pairing within 1 ms. On the other hand, the software implementations achieve very high performance, partially with the help

Table 6. Performance comparison of software and hardware implementations of pairings

Design	**Pairing**	**Security**	**Platform**	**Algorithm**	**Area**	**Freq.**	**Cycle**	**Delay**
		[bit]				[MHz]		[ms]
This design	ate	127	Xilinx FPGA (Virtex-6)	RNS (Parallel)	7032 Slices 32 DSP48E1s	250	229109	0.916
	optimal ate						166027	0.664
[13]	optimal ate	127	Altera FPGA (Stratix III)	RNS (Parallel)	4233 A‡	165	176111	1.07
[15]	ate	128	ASIC (130 nm)	HMM (Digit-serial)	183 kGates	204	861,724	4.22
	optimal ate						592,976	2.91
[17]	Tate	128	Xilinx FPGA (Virtex-4)	Blakley [8]	52k Slices	50	1,730,000	34.6
	ate						1,207,000	24.2
	optimal ate						821,000	16.4
[24]	Tate	128	ASIC (130 nm)	Montgomery	97 kGates	338	11,627,200*	34.4
	ate						7,706,400*	22.8
	optimal ate						5,340,400*	15.8
[14]	Tate over $\mathbb{F}_{3^{5\cdot 97}}$	128	Xilinx FPGA (Virtex-4)	-	4755 Slices 7 BRAMs	192	428,853	2.23
[21]	ate	128	64-bit Core2	Montgomery	-	2400	15,000,000	6.25
	optimal ate						10,000,000	4.17
[18]	ate	128	64-bit Core2	Montgomery		2400	14,429,439	6.01
[32]	optimal ate	128	Core2 Quad	Hybrid Mult.	-	2394	4,470,408	1.86
[6]	optimal ate	126	Core i7	Montgomery	-	2800	2,330,000	0.83
[1]	optimal ate	127	Phenom II	Montgomery	-	3000 †	1,562,000	0.52
[2]	η_T over $\mathbb{F}_{2^{1223}}$	128	Xeon (8 cores)	-	-	2000	3,020,000	1.51
[7]	η_T over $\mathbb{F}_{3^{509}}$	128	Core i7 (8 cores)	-	-	2900	5,423,000	1.87

* Estimated by the authors.

† Processor frequency is not mentioned in the original paper. We take 3.0 GHz (typical frequency) for delay estimation.

‡ It has 8 Rowers, each consisting of 2 36x36 DSP blocks and one 9x9 multiplier.

of the powerful 64-bit CPUs. Although it have not broken the software record, the speed of our work is already close to that of software.

An independent work using RNS and lazy reduction to compute cryptographic pairing is reported by Duquesne and Guillermin [13] [5]. Our design differs from this implementation mainly in two steps: the selection of the base for RNS and the architecture of the processing elements. By selecting the RNS bases (under certain specification), we reduce the size of multipliers for base extension and thus speed up the reduction. Furthermore, although both works use schoolbook multiplication, the accumulator used in our PE hides the lattency introduced by addition and subtraction. Our design is about 37% faster than the design in [13]. However, due to the difference in the platform, it is difficult to give a fair comparison on the efficiency of the architectures.

Both RNS-based pairing processors achieve higher performance than the ones using Montgomery (e.g. [24]), Hybrid Montgomery [15] and Blakley [17] algorithms. The results confirm the reduction of computational complexity by RNS. Indeed, multiplying two 256-bit integers (without considering the reduction) re-

[5] The first version of our paper was submitted to a conference before this paper was listed on Cryptology ePrint Archive.

quires only 16 times 32x32-bit multiplications, while Karatsuba multiplication requires 27 of them. The RNS reduction is indeed slower than traditional Montgomery reduction, but this overhead is largely reduced by lazy reduction. The results of [13] and the results of our implementation have demonstrated the efficiency of RNS in pairing implementations.

7 Conclusions

In this paper we present an efficient pairing processor that utilises RNS and lazy reduction. We describe a set of moduli that are suitable for RNS base for 254-bit multiplications. The proposed architecture is prototyped on Xilinx Virtex-6 FPGA, which utilizs 7032 slices and 32 DSPs and can run at 250 MHz. The coprocessor finishes one optimal pairing in 0.664 ms. To the best of our knowledge, this is by far the fastest hardware implementation for pairings that achieves 127-bit security.

References

1. Aranha, D., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster explicit formulas for computing pairings over ordinary curves. In: Eurocrypt 2011. To appear
2. Aranha, D., López, J., Hankerson, D.: High-speed parallel software implementation of the η_T pairing. In: Pieprzyk, J. (ed.) Topics in Cryptology - CT-RSA 2010, LNCS, vol. 5985, pp. 89–105. Springer Berlin / Heidelberg (2010)
3. Bajard, J.C., Kaihara, M., Plantard, T.: Selected RNS bases for modular multiplication. In: ARITH '09: Proceedings of the 2009 19th IEEE Symposium on Computer Arithmetic. pp. 25–32. IEEE Computer Society, Washington, DC, USA (2009)
4. Barreto, P., Kim, H., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) Advances in Cryptology CRYPTO 2002, LNCS, vol. 2442, pp. 354–369. Springer Berlin / Heidelberg (2002)
5. Barreto, P. and Naehrig, M. : Pairing-friendly elliptic curves of prime order. In: Proceedings of SAC 2005, volume 3897 of LNCS. pp. 319–331. Springer-Verlag (2006)
6. Beuchat, J.L., González-Díaz, J., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing-Based Cryptography - Pairing 2010, LNCS, vol. 6487, pp. 21–39. Springer Berlin / Heidelberg (2010)
7. Beuchat, J.L., López-Trejo, E., Martínez-Ramos, L., Mitsunari, S., Rodríguez-Henríquez, F.: Multi-core implementation of the Tate pairing over supersingular elliptic curves. In: Garay, J., Miyaji, A., Otsuka, A. (eds.) Cryptology and Network Security, LNCS, vol. 5888, pp. 413–432. Springer Berlin / Heidelberg (2009)
8. Blakely, G.: A computer algorithm for calculating the product AB modulo M. IEEE Transactions on Computers C-32(5), 497 –500 (1983)
9. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) Advances in Cryptology CRYPTO 2001, LNCS, vol. 2139, pp. 213–229. Springer Berlin / Heidelberg (2001)

10. Cha, J.C., Cheon, J.H.: An identity-based signature from gap Diffie-Hellman groups. In: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography: Public Key Cryptography. pp. 18–30. PKC '03, Springer-Verlag, London, UK, UK (2003)
11. Costello, C., Lange, T., Naehrig, M.: Faster pairing computations on curves with high-degree twists. In: Nguyen, P., Pointcheval, D. (eds.) Public Key Cryptography PKC 2010, LNCS, vol. 6056, pp. 224–242. Springer Berlin / Heidelberg (2010)
12. Devegili, A., Scott, M., Dahab, R.: Implementing cryptographic pairings over barreto-naehrig curves. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing-Based Cryptography Pairing 2007, Lecture Notes in Computer Science, vol. 4575, pp. 197–207. Springer Berlin / Heidelberg (2007)
13. Duquesne, S., Guillermin, N.: A FPGA pairing implementation using the residue number system. Cryptology ePrint Archive, Report 2011/176 (2011), http://eprint.iacr.org/
14. Estibals, N.: Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing-Based Cryptography - Pairing 2010, LNCS, vol. 6487, pp. 397–416. Springer Berlin / Heidelberg (2010)
15. Fan, J., Vercauteren, F., Verbauwhede, I.: Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on Barreto-Naehrig curves. In: Clavier, C., Gaj, K. (eds.) CHES 2009, LNCS, vol. 5747, pp. 240–253. Springer Berlin / Heidelberg (2009)
16. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. Journal of Cryptology 23, 224–280 (2010)
17. Ghosh, S., Mukhopadhyay, D., Roychowdhury, D.: High speed flexible pairing cryptoprocessor on FPGA platform. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing-Based Cryptography - Pairing 2010, LNCS, vol. 6487, pp. 450–466. Springer Berlin / Heidelberg (2010)
18. Grabher, P., Großschädl, J., Page, D.: On software parallel implementation of cryptographic pairings. In: Avanzi, R., Keliher, L., Sica, F. (eds.) Selected Areas in Cryptography, LNCS, vol. 5381, pp. 35–50. Springer Berlin / Heidelberg (2009)
19. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology. pp. 415–432. EUROCRYPT'08, Springer-Verlag, Berlin, Heidelberg (2008)
20. Guillermin, N.: A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$. In: Proceedings of the 12th international conference on Cryptographic hardware and embedded systems. pp. 48–64. CHES'10, Springer-Verlag, Berlin, Heidelberg (2010)
21. Hankerson, D., Menezes, A., Scott, M.: Software implementation of Pairings. In: Joye, M., Neven, G. (eds.) Identity-Based Cryptography (2008)
22. Hess, F., Smart, N., Vercauteren, F.: The Eta pairing revisited. IEEE Transactions on Information Theory 52(10), 4595 –4602 (2006)
23. Joux, A.: A one round protocol for tripartite Diffie-Hellman. Journal of Cryptology 17, 263–276 (2004)
24. 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.) Cryptographic Hardware and Embedded Systems - CHES 2009, LNCS, vol. 5747, pp. 254–271. Springer Berlin / Heidelberg (2009)
25. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. Soviet Phys. Doklady 7(7), 595–596 (1963)

26. Kawamura, S., Koike, M., Sano, F., Shimbo, A.: Cox-rower architecture for fast parallel Montgomery multiplication. In: Proceedings of EUROCRYPT 2000, vol. 1807, pp. 523–538. LNCS (2000)
27. Koç, Ç.K., Acar, T., Kaliski, B.S.: Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro 16, 26–33 (1996)
28. Lee, E., Lee, H.S., Park, C.M.: Efficient and generalized pairing computation on abelian varieties. IEEE Transactions on Information Theory 55(4), 1793 –1803 (2009)
29. Miller, V.S.: The Weil pairing, and its efficient calculation. Journal of Cryptology 17, 235–261 (2004), 10.1007/s00145-004-0315-8
30. Montgomery, P.L.: Five, six, and seven-term Karatsuba-like formulae. IEEE Transactions on Computer 54(3), 362 – 369 (2005)
31. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
32. Naehrig, M., Niederhagen, R., Schwabe, P.: New software speed records for cryptographic pairings. In: Abdalla, M., Barreto, P. (eds.) Progress in Cryptology LATINCRYPT 2010, LNCS, vol. 6212, pp. 109–123. Springer Berlin / Heidelberg (2010)
33. Pereira, G., Simplcio, M.J., Naehrig, M., Barreto, P.: A family of implementation-friendly BN elliptic curves. Cryptology ePrint Archive, Report 2010/429 (2010), `http://eprint.iacr.org/`
34. Posch, K., Posch, R.: Base extension using a convolution sum in residue number systems. Computing 50, 93–104 (1993)
35. Posch, K., Posch, R.: Modulo reduction in residue number systems. IEEE Transactions on Parallel and Distributed Systems 6(5), 449 –454 (May 1995)
36. Scott, M.: Implementing cryptographic pairings. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing-Based Cryptography - Pairing 2007, LNCS, vol. 4575, pp. 117–196. Springer (2007)
37. Solinas, J.: Generalized Mersenne numbers. Tech. rep., Combinatorics and Optimization Department, University of Waterloo (1999)
38. Szerwinski, R., Güneysu, T.: Exploiting the power of GPUs for asymmetric cryptography. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems CHES 2008, Lecture Notes in Computer Science, vol. 5154, pp. 79–99. Springer Berlin / Heidelberg (2008)
39. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory 56(1), 455 –461 (2010)
40. Walter, C.: Montgomery exponentiation needs no final subtractions. Electronics Letters 35(21), 1831 –1832 (1999)
41. Xlinx, Inc.: Virtex-6 FPGA DSP48E1 Slice: User Guide, `http://www.xilinx.com/support/documentation/user_guides/ug369.pdf`