FPGA Implementation of Pairings using Residue Number System and Lazy Reduction *

Ray C.C. Cheung[1], Sylvain Duquesne[2], Junfeng Fan[4], Nicolas Guillermin[2,3], Ingrid Verbauwhede[4], and Gavin Xiaoxu Yao[1]

[1] Department of Electronic Engineering
City University of Hong Kong, Hong Kong SAR,
r.cheung@cityu.edu.hk, gavin.yao@student.cityu.edu.hk
[2] IRMAR, UMR CNRS 6625, Université Rennes 1
Campus de Beaulieu, 35042 Rennes cedex, France
[3] DGA.IS, La Roche Marguerite - 35170 - Bruz, France
sylvain.duquesne@univ-rennes1.fr, nicolas.guillermin@m4x.org
[4] Katholieke Universiteit Leuven, COSIC & IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{junfeng.fan, ingrid.verbauwhede}@esat.kuleuven.be

Abstract. Recently, a lot of progress has been made in the implementation of pairings in both hardware and software. In this paper, we present two FPGA-based high speed pairing designs using the Residue Number System and lazy reduction. We show that by combining RNS, which is naturally suitable for parallel architectures, and lazy reduction, which performs one reduction for multiple multiplications, the speed of pairing computation in hardware can be largely increased. The results show that both designs achieve higher speed than previous designs. The fastest version computes an optimal ate pairing at 126-bit security level in 0.573 ms, which is 2 times faster than all previous hardware implementations at the same security level.

Keywords: Optimal Pairing, Residue Number System, Lazy Reduction, FPGA

1 Introduction and Motivation

Bilinear pairings on elliptic curves have been introduced in cryptography in the middle of 90's for cryptanalysis [18, 33]. In 2000, Joux introduced the first constructive use of pairings with a tripartite key exchange protocol [25]. In the last decade many pairing-based schemes such as identity-based encryption [10],

* This work was supported in part by the European Commission's ECRYPT II NoE (ICT-2007-216676), by the Belgian State's IAP program P6/26 BCRYPT, by the K.U. Leuven-BOF (OT/06/40), by the Research Council K.U. Leuven: GOA TENSE (GOA/11/007), by French ANR projects no. 07-BLAN-0248 "ALGOL" and 09-BLAN-0020-01 "CHIC", and by City University of Hong Kong Start-up Grant 7200179.

identity-based signatures [12] and short signatures [11] have been proposed and studied. Compared with other popular public key cryptosystems, e.g. Elliptic Curve Cryptography (ECC) [30,34] and RSA [41], pairing computation is much more complicated. For this reason, efficient implementation of cryptographic pairings has received increasing interests [3,8,16,20,23,27,36].

The computation of a pairing can be broken down into modular operations in the underlying fields. For example, one optimal ate pairing [44] defined on a 256-bit Barreto-Naehrig (BN) curve [7] requires around 10^4 modular multiplications [3]. Thus, having an efficient modular multiplier is the key step to a high performance pairing processor. In this work, we are interested in hardware implementation of pairings over large characteristic fields. In this case one possible optimization is to use lazy reduction. Lazy reduction in pairing computation was introduced by Scott [42] and then generalized by Aranha *et al.* in [3]. In short, it performs one reduction for expressions like $\sum A_i B_j$, where $A_i, B_j \in \mathbb{F}_p$. Aranha *et al.* have shown that lazy reduction can significantly speed up optimal ate pairings in software [3].

As suggested by Duquesne [14], lazy reduction can be combined with the Residue Number System (RNS) to further reduce the complexity. Besides, the RNS distributes computation over a group of small integers, and is naturally suitable for parallel implementations [22, 29]. In this paper, we propose two FPGA-based pairing processors that use RNS representation and lazy reduction. The first design, referred as Design I, is based on a general (but enhanced) RNS data-path. Design I is scalable in terms of security level and flexible in terms of target devices. On an Altera Stratix III FPGA, Design I computes a pairing at 126-bit security level in 1.07 ms. The second design, referred as Design II, has an optimized architecture for pairings over 254-bit curves. We use a set of parameters that leads to a reduced complexity and an optimized datapath that benefits from the reduction. Design II computes a pairing at 126-bit security level in 0.573 ms on a Xilinx Virtex-6 FPGA. To the best of our knowledge, these are the first hardware designs of pairing using RNS, and the designs outperform all previous hardware implementations at a similar security level [16,19,27].

The rest of the paper is organised as follows. Section 2 and Section 3 provide the background on optimal ate pairing and RNS, respectively. In Section 4 and Section 5 we describe the architecture of Design I and Design II. Section 6 describes the control of the data-path and the high level scheduling. We give a detailed analysis of the performance in Section 7. Finally, Section 8 concludes the paper.

2 Optimal Ate Pairings

2.1 Pairings on Barreto-Naehrig Curves

A bilinear pairing is a non-degenerate map from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$ which is linear in both components. Popular pairings such as Tate pairing [6], ate pairing [24], R-ate pairing [32], optimal pairing [44] 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 $\mathbb{F}^*_{p^k}$.

Let $\mathbb{F}_p$ be a finite field and let E be an elliptic curve defined over $\mathbb{F}_p$. Let ℓ be a large prime dividing $\#E(\mathbb{F}_p)$ and k the embedding degree with respect to ℓ, namely, the smallest positive integer k such that $\ell | p^k - 1$. Small embedding degrees can be easily obtained using supersingular curves. However, it is too small ($k \leq 2$) if large characteristic base fields are used. Thus, we use ordinary curves with prescribed embedding degrees constructed via the complex multiplication method as surveyed in [17]. We focus on the most popular one to date, namely the Barreto-Naehrig curves [7]. The reason for their popularity is that they are well-suited for 128-bit security level and they have degree 6 twist.

Let $u \in \mathbb{Z}$ such that $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and $\ell = 36u^4 + 36u^3 + 18u^2 + 6u + 1$ are prime. A BN curve is an elliptic curve defined over $\mathbb{F}_p$ by

$$E : y^2 = x^3 + b,$$

where $b \neq 0$ such that $\#E = \ell$, and it has 12 as an embedding degree.

In this paper, we mainly focus on the discussion of optimal ate pairing [36] because it is the most efficient to date for BN curves. Let $r = 6u + 2$, an optimal ate pairing on BN curves is defined as follows [2, 36]:

$$\begin{aligned} a_{opt} : E(\mathbb{F}_{p^{12}}) \cap \mathrm{Ker}(\pi_p - p) \times E(\mathbb{F}_p)[\ell] \ &\rightarrow \ \mathbb{F}^*_{p^{12}} / \left(\mathbb{F}^*_{p^{12}}\right)^{\ell} \\ (Q, P) \ &\mapsto \ \left(f_{(r,Q)}(P) \cdot g_{(rQ, \pi_p(Q))}(P) \cdot g_{(rQ + \pi_p(Q), -\pi_p^2(Q))}(P) \right)^{\frac{p^{12}-1}{\ell}} \end{aligned}$$

where π_p is the Frobenius map on the curve ($\pi_p(x, y) = (x^p, y^p)$), and $g_{(Q_1, Q_2)}$ is the line through Q_1 and Q_2.

2.2 Pairing Computation and Parameter Selection

Pairing computation The computation consists of two main functions, $f_{(r,Q)}$ and $f^{\frac{p^{12}-1}{\ell}}$. The function $f_{(r,Q)}$ has the following divisor:

$$\mathrm{div}(f_{(r,Q)}) = r(Q) - (rQ) - (r-1)(\mathcal{O}).$$

It is normally computed using a double-and-add method (also known as Miller's loop [35]). Concerning $f^{\frac{p^{12}-1}{\ell}}$, also known as the final exponentiation, Koblitz and Menezes show in [31] that it can be split in two steps due to the integer factorization

$$\frac{p^{12} - 1}{\ell} = \left(p^6 - 1\right)\left(p^2 + 1\right)\left(\frac{p^4 - p^2 + 1}{\ell}\right).$$

The first step is powering to $p^6 - 1$ and to $p^2 + 1$. This is easily obtained via cheap Frobenius computations and an inversion. The second step is powering to $\frac{p^4 - p^2 + 1}{\ell}$ which is called the hard part of the final exponentiation.

Parameter selection The selection of parameters has an essential impact on the security and the performance of a pairing computation. It is explained in [39] how to generate BN curves with nice properties. For this work we choose two curves both with $b = 2$. The first one, BN_{126}, is defined by $u = -(2^{62} + 2^{55} + 1)$ and has already been used in [2, 39]. It ensures only 126 bits of security, but is well suited to registers on a general-purpose CPU when lazy reduction is used. However, FPGA architectures have no such constraints, since multipliers of larger size can always be constructed with DSP slices. Hence, we also consider a curve BN_{128} defined by $u = -(2^{63}+2^{22}+2^{18}+2^7+1)$ which ensures 128 bits of security. Finally, we propose BN_{192}, the BN curve defined by $u = -(2^{160}+2^{74}+2^{12}+1)$. Optimal pairing defined on this curve provides 192 bits of security [38].

In the three cases, the extension fields are defined as follows:

- $\mathbb{F}_{p^2} = \mathbb{F}_p[\mathbf{i}]/(\mathbf{i}^2 + 1)$
- $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[\boldsymbol{\beta}]/(\boldsymbol{\beta}^3 - (1 + \mathbf{i}))$
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[\boldsymbol{\Gamma}]/(\boldsymbol{\Gamma}^2 - \boldsymbol{\beta}) = \mathbb{F}_{p^2}[\boldsymbol{\gamma}]/(\boldsymbol{\gamma}^6 - (1 + \mathbf{i}))$

This tower of extensions has many advantages, the most important being an efficient multiplication algorithm for the canonical polynomial base.

Note that BN curves always have degree 6 twists. This means E is isomorphic over $\mathbb{F}_{p^{12}}$ to a curve E' defined by $y^2 = x^3 + \frac{b}{\zeta}$, where ζ is neither a square nor a cube in $\mathbb{F}_{p^2}$. In our case we take $\zeta = 1 + \mathbf{i}$ so that it defines both the sextic extension of $\mathbb{F}_{p^2}$ and the twist. Then we can define twisted versions of pairings on $E'(\mathbb{F}_{p^2}) \times E(\mathbb{F}_p)[\ell]$. In other words, the coordinates of Q can be written as $(x_Q\zeta^{\frac{1}{3}}, y_Q\zeta^{\frac{1}{2}})$ where x_Q and y_Q are in $\mathbb{F}_{p^2}$. For u selected as a negative integer, the optimal ate pairing for BN curves is computed by Algorithm 1 [2] where `dbl`, `add` and `hard-part` are given in appendix.

3 Residue Number System

A Residue Number System (RNS) represents a large integer using a set of smaller integers. Let $\mathfrak{B} = \{b_1, b_2, \ldots, b_n\}$ be a set of pairwise co-prime integers, and $M_{\mathfrak{B}} = \prod_{i=1}^{n} b_i$. For any integer X, $0 \leq X < M_{\mathfrak{B}}$, there is a unique RNS representation on $\mathfrak{B}$: $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$, where $x_i = |X|_{b_i}$, $1 \leq i \leq n$. Throughout the paper we use $|a|_b$ to denote $a \bmod b$. Given $\{X\}_{\mathfrak{B}}$, one can recover X using the Chinese Remainder Theorem (CRT):

$$X = \left| \sum_{i=1}^{n} \left| x_i \cdot B_i^{-1} \right|_{b_i} \cdot B_i \right|_{M_{\mathfrak{B}}} \text{ where } B_i = \frac{M_{\mathfrak{B}}}{b_i}. \tag{1}$$

The set $\mathfrak{B}$ is also known as a *base*, and each element b_i, $1 \leq i \leq n$, is called an RNS modulus or an RNS channel.

RNS representation admits efficient parallel computations. Consider two integers X, Y and their RNS representations $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$ and $\{Y\}_{\mathfrak{B}} = \{y_1, y_2, \ldots, y_n\}$, then we have

$$\{|X \odot Y|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}} = \{|x_1 \odot y_1|_{b_1}, \ldots, |x_n \odot y_n|_{b_n}\}, \odot \in \{+, -, \times, /\}. \tag{2}$$

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

Require: $P \in E(\mathbb{F}_p)[\ell], Q = (x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3) \in E(\mathbb{F}_{p^{12}}) \cap \mathrm{Ker}(\pi_p - p)$ with x_Q and $y_Q \in \mathbb{F}_{p^2}, r = |6u+2| = \sum_{i=0}^{s-1} r_i 2^i$, where $u < 0$.
Ensure: $a_{opt}(Q,P) \in \mathbb{F}_{p^{12}}$
1: $T = (X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T) \leftarrow (x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3, 1), f \leftarrow 1$
2: **for** $i = s-2$ downto 0 **do**
3: $\quad T, g \leftarrow \mathtt{dbl}(T,P), f \leftarrow f^2 \cdot g$
4: $\quad$ **if** $r_i = 1$ **then**
5: $\quad\quad T, g \leftarrow \mathtt{add}(T,Q,P), f \leftarrow f \cdot g$
6: $\quad$ **end if**
7: **end for**
8: $T \leftarrow -T, f \leftarrow f^{p^6}$ (f^{p^6} is equivalent to f^{-1} as noticed in [3])
9: $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow -\pi_p(Q_1)$
10: $T, g \leftarrow \mathtt{add}(T, Q_1, P), f \leftarrow f \cdot g$
11: $T, g \leftarrow \mathtt{add}(T, Q_2, P), f \leftarrow f \cdot g$
12: $f \leftarrow \left(f^{p^6-1}\right)^{p^2+1}$
13: $f \leftarrow \mathtt{hard\text{-}part}(f, |u|)$
14: **return** f

Note that the division is available only if Y is co-prime with $M_{\mathfrak{B}}$. For all these operations, computations between x_i and y_i have no dependency on other channels, which makes RNS naturally suitable for parallel implementations.

The computation of $|X|_{b_i}$ is called a channel reduction. To accelerate this operation, pseudo-Mersenne numbers of the form $b_i = 2^{\mathbf{w}} - \varepsilon_i$, where $\varepsilon_i < 2^{\mathbf{w}/2}$, are typically selected as RNS moduli. Hence, the computation of $|X|_{b_i}$ is performed using 2 times of $X \leftarrow \lfloor X/2^{\mathbf{w}} \rfloor \cdot \varepsilon_i + (X \bmod 2^{\mathbf{w}})$, and a correction step in the end to bring the result back to the range $[0, b_i)$.

3.1 RNS Montgomery Reduction

Using RNS representation ensures efficient computation in $\mathbb{Z}/M_{\mathfrak{B}}\mathbb{Z}$. Unfortunately, it can't be applied directly in $\mathbb{F}_p$ since $M_{\mathfrak{B}}$ is not prime. One way to utilize RNS for field multiplication is to combine RNS and Montgomery reduction [22, 29]. This is shown in Algorithm 2.

RNS Montgomery reduction requires two bases, $\mathfrak{B}$ and $\mathfrak{C}$, with $M_{\mathfrak{C}}$ co-prime to $M_{\mathfrak{B}}$. The reason of including $\mathfrak{C}$ is that division by $M_{\mathfrak{B}}$ is not possible in $\mathfrak{B}$. Note that the size of $M_{\mathfrak{B}}$ and $M_{\mathfrak{C}}$, compared to p, determine the upper bound of input X. Guillermin found that if $X < \alpha p^2$, $M_{\mathfrak{B}} > \alpha p$ and $M_{\mathfrak{C}} > 2p$, then Algorithm 2 has output $S < 2p$ [22, Proposition 1]. This is an important principle for base selection.

3.2 Base Extension

The operation to transform the representation in one RNS base to another base is called Base Extension (BE). To compute $\{X\}_{\mathfrak{C}} = \{x'_1, x'_2, \ldots, x'_n\}$ from $\{X\}_{\mathfrak{B}} =$

Algorithm 2 RNS Montgomery reduction [5]

Require: RNS bases $\mathfrak{B}$ and $\mathfrak{C}$ with $M_{\mathfrak{B}} > \alpha p, M_{\mathfrak{C}} > 2p$, p coprime with $M_{\mathfrak{B}}M_{\mathfrak{C}}$, $\{X\}_{\mathfrak{B}}$ and $\{X\}_{\mathfrak{C}}$ being the RNS representations of $X < \alpha p^2$.
Precomputed: $\{|-p^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$, $\{|M_{\mathfrak{B}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$ and $\{p\}_{\mathfrak{C}}$.
Ensure: $\{S\}_{\mathfrak{B}}$, $\{S\}_{\mathfrak{C}}$ such that $|S|_p = |XM_{\mathfrak{B}}^{-1}|_p$ and $S < 2p$
1: $\{Q\}_{\mathfrak{B}} \leftarrow \{X\}_{\mathfrak{B}} \times \{-p^{-1}\}_{\mathfrak{B}}$
2: $\{Q\}_{\mathfrak{B}} \xrightarrow{Base\ Extension} \{Q\}_{\mathfrak{C}}$
3: $\{S\}_{\mathfrak{C}} \leftarrow \left(\{X\}_{\mathfrak{C}} + \{Q\}_{\mathfrak{C}} \times \{p\}_{\mathfrak{C}}\right) \times \{M_{\mathfrak{B}}^{-1}\}_{\mathfrak{C}}$
4: $\{S\}_{\mathfrak{B}} \xleftarrow{Base\ Extension} \{S\}_{\mathfrak{C}}$

$\{x_1, x_2, \ldots, x_n\}$, one can use the Posch-Posch method [40]. Given $\{X\}_{\mathfrak{B}}$, for (1), there must exist an integer $\lambda < n$ such that:

$$X = \left| \sum_{i=1}^{n} \left| x_i \cdot B_i^{-1} \right|_{b_i} \cdot B_i \right|_{M_{\mathfrak{B}}} = \left| \sum_{i=1}^{n} \xi_i \cdot B_i \right|_{M_{\mathfrak{B}}} = \sum_{i=1}^{n} \xi_i \cdot B_i - \lambda \cdot M_{\mathfrak{B}} \tag{3}$$

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

$$\lambda = \left\lfloor \sum_{i=1}^{n} \frac{\xi_i \cdot B_i}{M_{\mathfrak{B}}} \right\rfloor = \left\lfloor \sum_{i=1}^{n} \frac{\xi_i}{b_i} \right\rfloor \tag{4}$$

In [29], ξ_i/b_i is further approximated by $\xi_i/2^{\mathbf{w}}$ as b_i is chosen as a pseudo-Mersenne number near $2^{\mathbf{w}}$. Once λ is obtained, $\{X\}_{\mathfrak{C}} = \{x'_1, \ldots, x'_n\}$ can be computed as follows:

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

$|B_i|_{c_j}$ and $|M_{\mathfrak{B}}|_{c_j}$, $1 \leq i, j \leq n$, can be precomputed once $\mathfrak{B}$ and $\mathfrak{C}$ are fixed. The following algorithm describes the computation of $|X|_{c_k}$.

4 Design I: A Scalable Architecture

In this section we propose an enhancement of the Cox-Rower architecture first proposed in [29] such that it is suitable for pairing computation.

4.1 Cox-Rower Architecture

The Cox-Rower architecture was first proposed by Kawamura *et al.* in [29]. It was first implemented in a VLSI design of an RSA cryptosystem [37]. It was later enhanced by Guillermin [22] to support all arithmetic operations in $\mathbb{F}_p$ and fast Elliptic Curve Scalar Multiplications (ECSM).

Algorithm 3 Base extension algorithm for k-th element of X [29]

Require: $|X|_{b_i}$ for $i \in \{1, .., n\}$
Ensure: $|X|_{c_k}$
Precomputed: $|B_i^{-1}|_{b_i}, |B_i|_{c_k}, 1 \leq i \leq n, |M_{\mathfrak{B}}|_{c_k}$
1: $\xi_i \leftarrow |x_i \cdot B_i^{-1}|_{b_i}$ for $i \in \{1, .., n\}$
2: $\psi \leftarrow 0, z \leftarrow 0$
3: **for** $i = 0$ to $(n-1)$ **do**
4: $\quad \psi \leftarrow \psi + \xi_i$
5: $\quad \rho \leftarrow \lfloor \psi / 2^{\mathbf{w}} \rfloor \qquad //\rho \in \{0, 1\}$
6: $\quad \psi \leftarrow |\psi|_{2^{\mathbf{w}}}$
7: $\quad z \leftarrow z + (\xi_i \cdot |B_i|_{c_k}) - \rho \cdot |M_{\mathfrak{B}}|_{c_k}$
8: **end for**
9: **return** z

Fig. 1 shows our Cox-Rower implementation. The top-level structure is similar to the one proposed by [29]. It is consists of n similar Rower units performing in parallel operations in one RNS channel of $\mathfrak{B}$ or $\mathfrak{C}$. The Cox unit is only used during the base extension (Algorithm 3) to generate the ρ. The Rower unit normally consists of a multiplier and channel reduction logic, and serves as the workhorse of both multiplication (operation $\times$ in (2)) and reduction (step 1, 3 in Algorithm 2 and step 1, 7 in Algorithm 3). The Cox-Rower is driven by a microcoded sequencer, and the sequencer can be easily reprogrammed to provide support for different algorithms.

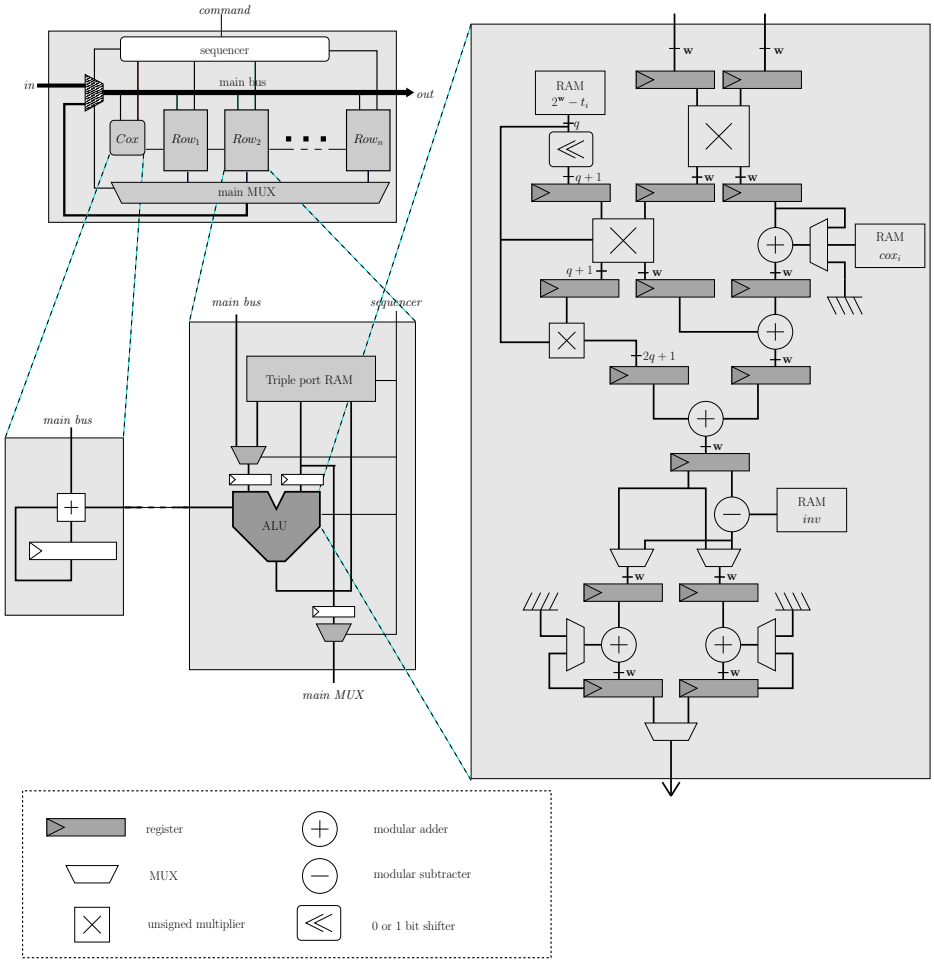
While the basic structure of our design resembles the Cox-Rower architecture in Guillermin's ECC processor [22], our architecture has an optimized pipeline structure and a more aggressive memory organization specifically designed for pairing support.

4.2 Cox-Rower Parametrization for Pairing

First we need to parametrize the Cox-Rower to support the pairings on the three curves defined in Section 2.2. Popular Altera and Xilinx FPGAs have embedded 18×18 multipliers or even 25×18 multipliers (Virtex-5 and higher). Moreover, a full 36×36 multiplier can be built by efficiently combining several such small multipliers. It is thus natural to select $\mathbf{w}$=18 or $\mathbf{w}$=36. For the implementation of BN_{126} or BN_{128}, $\mathbf{w}$=36 gives the best trade-off. Indeed the gain in frequency brought by smaller multipliers and adders does not compensate the necessary cycle surplus of reductions. The parameter n can then be set to 8 for BN_{126} and BN_{128}, and 19 for BN_{192}. Because of our chosen arithmetic (see Section 6), the value α defined in Section 3.1 reaches 198 for all the curves. The worst case is reached during the schoolbook multiplication in $\mathbb{F}_{p^{12}}$. One can verify that, in order to have $M_{\mathfrak{B}} > \alpha p$, $\mathbf{w}$=33 is enough to support BN_{126} and w=34 for both BN_{126} and BN_{128}. For BN_{192}, $\mathbf{w}$ is set to 35. In this architecture, we used the same method to select bases as in [22], with pseudo-Mersennes of the form $t_i = 2^{\mathbf{w}} - \varepsilon_i$, $t_i \in \mathfrak{B} \cup \mathfrak{C}$ and ε_i positive.

An adaptation of Guillermin's architecture is necessary to provide pairing support. As the number of local variables and precomputations is much larger for pairings than that of ECSM (in fact, the Miller's loop has a built-in ECSM), we use a single triple port RAM of 256 words instead of the ROM and a group of 16 registers to store precomputed values and temporary results. This is enough to support all curves listed in Section 2.2.

Fig. 1. Design I: architecture of the Cox-Rower and its pipeline structure



4.3 Pipeline Architecture

Our goal is to keep the maximal frequency already available in [22]. Additive hardware must be carefully introduced, to keep the critical path under control. On the other hand, we can easily raise the pipeline depth without a lot of cycle loss. Indeed, pairing computation has more parallelizable operations than classical elliptic curve scalar multiplication over $\mathbb{F}_p$. The architecture in [37] and [22] uses 2-stage and 5-stage pipelines, respectively. For the implementation of pairings, we found that a pipeline of up to 10 stages can still be efficiently filled during the whole pairing computation except for $\mathbb{F}_p$ inversion.

Based on Guillermin's architecture, two accumulators are included in the pipeline to efficiently support pairing computation. Fig. 1 shows the adapted pipeline architecture. The first 4 stages perform $|a\times b|_{t_i}$, where $t_i \in \mathfrak{B}\cup\mathfrak{C}$. We also introduce shift and MUX units in the first 4 stages to compute $|2\times a\times b|_{t_i}$, which are utilized to accelerate squarings in the extension fields. Three multipliers ($\mathbf{w}\times\mathbf{w}$, $\mathbf{w}\times(\mathbf{q}+1)$ and $(\mathbf{q}+1)\times\mathbf{q}$) are used, where $\mathbf{q}=\lceil\log_2 \varepsilon_i\rceil$.

In the 6^{th} stage we implemented two independent accumulators. They are preceded by a subtracter (which computes $|-a\times b|_{t_i}$) and a MUX in the 5^{th} stage. The output of the first 4 stages can then be independently added, subtracted or ignored by both accumulators. These two accumulators, together with the use of tower extensions, save a lot of cycles during the pairing computation. See Section 6 for details.

On this architecture, an RNS multiplication costs 2 cycles and the results can be accumulated immediately. An RNS reduction costs $2n+3$ cycles as previously described in [22].

5 Design II: Hardware/Algorithm Co-optimization

In this section, we propose an optimized design that achieves an even higher throughput. The improvement comes mainly from two tricks: a set of *good* bases that admits a less-expensive base extension, and a fine-tuned pipeline structure that allows a higher frequency.

5.1 Base Selection Revisited

The core observation here is that the complexity of base extension can be reduced if the moduli in the two bases are close to each other. The base extension is the most computational expensive operation in the RNS Montgomery algorithm. It requires n^2 times of $\mathbf{w}\times\mathbf{w}$ multiplications. Indeed, (5) can be written as a matrix multiplication below.

$$\begin{pmatrix} x'_1 \\ \vdots \\ x'_n \end{pmatrix} \equiv \begin{pmatrix} |B_1|_{c_1} & \cdots & |B_n|_{c_1} \\ \vdots & \ddots & \vdots \\ |B_1|_{c_n} & \cdots & |B_n|_{c_n} \end{pmatrix} \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_n \end{pmatrix} - \gamma \begin{pmatrix} |M_{\mathfrak{B}}|_{c_1} \\ \vdots \\ |M_{\mathfrak{B}}|_{c_n} \end{pmatrix} \tag{6}$$

Note that the elements in the matrix, $|B_i|_{c_j}$, $1 \le i, j \le n$, are constants and are generated as follows:

$$|B_i|_{c_j} = \left| \prod_{k=1,k\neq i}^{n} b_k \right|_{c_j} = \left| \prod_{k=1,k\neq i}^{n} (b_k - c_j) \right|_{c_j} \tag{7}$$

Define $\tilde{B}_{i,j} := \prod_{k=1,k\neq i}^{n}(b_k - c_j)$. When b_k and c_j are close to each other, the difference $b_k - c_j$ is small. In practice, $|\tilde{B}_{i,j}|$ could be much smaller than $|c_j|$ if n is relatively small. Note that using $|B_i|_{c_j}$ or $\tilde{B}_{i,j}$ makes no difference in the final results due to the channel reduction on the products.

Furthermore, $\tilde{B}_{i,j}$ for $1 \le i, j \le n$ will be predictably divisible by 2^{n-2} if c_j is odd. Consider the two bases $\mathfrak{B} = \{b_1, b_2, \cdots, b_n\}$ and $\mathfrak{C} = \{c_1, c_2, \cdots, c_n\}$. Since there is at most one even number in $\mathfrak{B} \cup \mathfrak{C}$, $(b_i - c_j)$ is divided by 2 unless b_i or c_j is even. In practice, $\tilde{B}_{i,j}$ can have more than $n-2$ zero bits in the least significant bits (LSBs). To shrink the operand size of the matrix multiplications, we can use truncate the least significant zeros of $\tilde{B}_{i,j}$, and restore the correct results by a simple left-shift. We denote $\tilde{B}'_{i,j}$ the $\tilde{B}_{i,j}$ after truncation.

Considering the size of p and the Cox-Rower architecture, we again select $n = 8$ and thus b_i (and c_j) close to 2^{33}. The selection of moduli also takes into account the size of $\tilde{B}'_{i,j}$ for $1 \le i, j \le n$. The following 16 moduli ($\mathbf{w} = 33$) were selected as the bases.

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

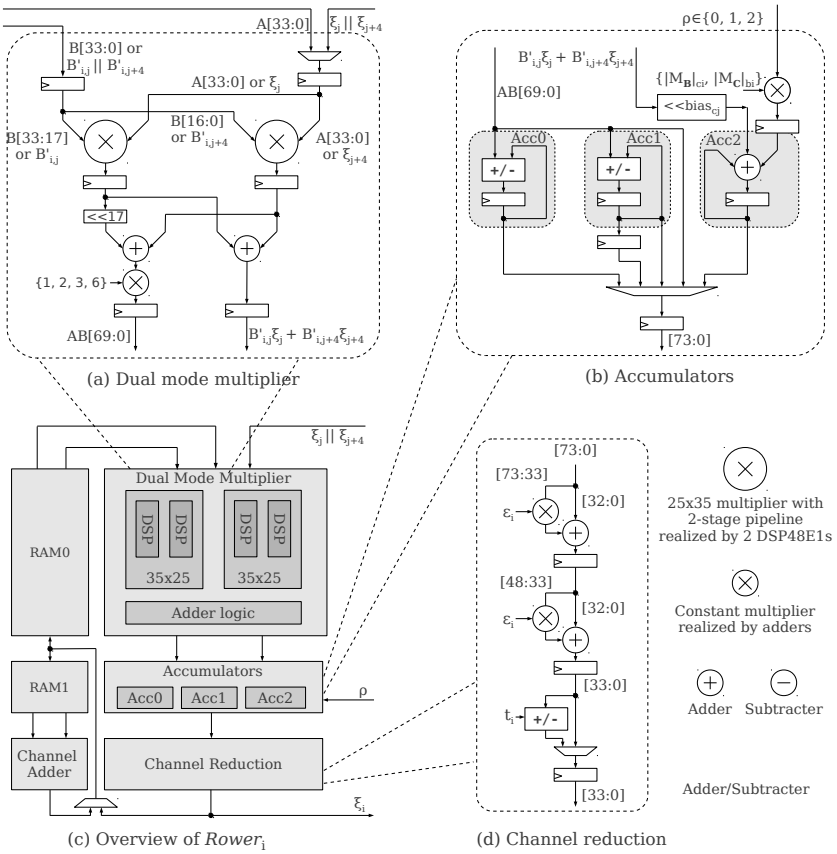
After applying the truncation of zero bits, we manage to reduce the bitlength of all $|B_{i,j}|$ (actually, $\tilde{B}'_{i,j}$) from standard 34 to 25. While this complexity reduction seems negligible, it admits notable savings in hardware design. A detailed analysis of the base selection is given in Appendix A.2.

5.2 A Fine-tuned Rower for Pairing Computation

Our refinement focuses solely on the design of Rowers. Fig. 2(c) shows the overview of a Rower. It consists of a dual mode multiplier, 3 accumulators, a channel reduction module, a channel adder, a triple port RAM for multiplier inputs and a RAM for adder inputs.

Dual mode multiplier The multiplier in the Rower is built to support the bases selected in Section 5.1. There are two types of multiplication executed in an RNS multiplication: 34×34 multiplication (step 1, 3 of Algorithm 2) and 25×35 multiplication (step 1, 7 of Algorithm 3). The DSP slices in recent Xilinx FPGAs are made of a signed 25×18 bit multiplier, a 17-bit left shifter, and an accumulator. The dual mode multiplier is built with four DSP slices, which supports either 34×34 unsigned multiplication or two signed 35×25 multiplications in parallel. Fig. 2(a) illustrates the structure of the dual mode multiplier.

Fig. 2. The fine-tuned Rower architecture



For pairing computation, multiplication by small constants (2, 3, 6) is widely employed. Therefore, a constant multiplier is also included in the pipeline.

Because two multiplications are executed simultaneously in the base extension, the number of cycles to perform the Cox-Rower algorithm (Algorithm 3) is reduced from n to $n/2$. This is a significant speedup of the base extension.

Other optimizations The architecture of channel reduction is shown in Fig. 2(d). As the Hamming weights of all the ε_i are either equal to or less than 3 in non-adjacent form, multiplication by ε_i can be realized by 4 adders instead of multipliers. In order to maintain a high operating frequency, we use a three-stage pipeline to realize the channel reduction.

To achieve the maximum usage of the multiplier, a separate adder together with a dedicated RAM is included. Because of the lazy reduction inside the Rower, the write port of RAM0 is not always occupied by the multiplier. Therefore, the addition and the multiplication can run in parallel. While most of the

additions in the pairing computation are performed by the accumulators, the separate adder is useful in point additions and doublings.

While this architecture requires less cycles for reduction than Design I, it is less scalable. Indeed, in order to support larger p, we need to choose larger n. The bitlength of $B_{i,j}$ increases quickly when n goes up, and two multiplications in parallel becomes impossible on the current Rower.

6 Scheduling the Pairing Algorithm

In this section we present the implementation choices for every step of the pairing computation.

6.1 Arithmetic in $\mathbb{F}_{p^2}$: Back to the Schoolbook Method

Karatsuba and derived interpolation methods have been used intensively in field operations in the literature to save the expensive multiplication [3, 8, 23, 42]. Karatsuba uses 3 instead of 4 multiplications at the cost of 3 extra additions. In a normal positional number system, this method saves computation power, as multiplications are much more expensive than additions. However, in RNS, the complexities of a multiplication and an addition are the same. Hence, the schoolbook method involves less operations (counting both additions and multiplications) and is preferred. On both architectures, a full $\mathbb{F}_{p^2}$ multiplication finishes in 8 cycles, and a squaring in only 6 cycles.

6.2 Arithmetic in $\mathbb{F}_{p^{12}}$: Interpolation with Parsimony

Let $X = \{x_1, \cdots, x_{12}\}$, $Y = \{y_1, \cdots, y_{12}\}$ and $Z = X \times Y \in \mathbb{F}_{p^{12}}$. To accelerate $\mathbb{F}_{p^{12}}$ arithmetic, we aim to execute only once $x_i \times y_j$, for all $\{i, j\} \in \{1, \cdots, 12\}^2$. As $\mathbf{i}^2 = -1$ and $\gamma^6 = (1 + \mathbf{i})$, the result of $x_i \times y_j$ is either added to or subtracted from at most two components of Z. This is the reason for the presence of two independent accumulators. Moreover, the result of the multiplication can be multiplied by 2 on the fly before it is accumulated, which speeds up the squaring in $\mathbb{F}_{p^{12}}$.

We estimate if the interpolation techniques at higher levels may save cycles or not. We exclude the reduction step, therefore the cycle count is equivalent on the flexible as well as the optimized design. We only consider Karatsuba in $\mathbb{F}_{p^{12}}$ over $\mathbb{F}_{p^6}$ which is the best case for these techniques (interpolation at other levels $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}/\mathbb{F}_{p^2}$ or $\mathbb{F}_{p^4}/\mathbb{F}_{p^2}$ would give worse results). Four different types of multiplications and squaring are needed during pairing, each one requiring a different algorithm:

- Squaring during the Miller loop: the operand of this squaring has no specific structure. We propose to use schoolbook squaring. Thanks to the multiplication by 2 included in the pipeline of both Design I and Design II, a squaring costs 156 cycles. The interpolation on $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}$ leads to the following

formula: $(X_H + \boldsymbol{\Gamma} X_L)^2 = \big((X_H + X_L)(X_H + \boldsymbol{\Gamma}^2 X_L) - (1 + \boldsymbol{\Gamma}^2) X_H X_L\big) + \boldsymbol{\Gamma}(2 X_H X_L)$ (with $X_H, X_L \in \mathbb{F}_{p^6}$). We found no method to go below 156 cycles on both designs.
- Multiplication by the line in the Miller loop: half the values of B are equal to 0, therefore the schoolbook multiplication costs 144 cycles. Interpolation techniques do not manage to take the advantage of the half size operand.
- Squaring during final exponentiation: during the hard part of FE, the operands to be squared are in $G_{\Phi_6}(\mathbb{F}_{p^2})$. We use the formulae given in [21] (Algorithm A.1). On Design I, we need 84 cycles to implement it. This approach is much more efficient than interpolation techniques. [5]
- Multiplication during final exponentiation: Without counting pipeline bubbles, Karatsuba requires 278 and 254 cycles on Design I and Design II, respectively, while the schoolbook method requires 288 on both. Note that there are only 20 such multiplications in the whole pairing for BN_{126}, and the savings are really limited. Because of the moderate gain and the complication brought to the sequencer, we decided not to implement it.

6.3 $\mathbb{F}_p$ Inversion

In the final exponentiation of pairing, an $\mathbb{F}_{p^{12}}$ inversion is required. It can be done with only one inversion, 35 reductions and additional multiplications/additions in $\mathbb{F}_p$ [14, 23], but the remaining inversion in $\mathbb{F}_p$ is very expensive. Since comparison in RNS is difficult, inversion through exponentiation ($X^{-1} \equiv X^{p-2} \bmod p$) is used. For this operation, we use a simple square and multiply algorithm with least significant bit first. It is more memory consuming, but it allows to perform multiplications in parallel. On a pipelined datapath, LSB-first exponentiation is more efficient than the MSB-first method. In total, an inversion needs $\lfloor log(p) - 1 \rfloor$ squarings and many multiplications, but the cost of multiplications is hidden in the pipeline.

6.4 Higher Level Scheduling

Based on the operation over $\mathbb{F}_p$ and its extension fields, we are ready to implement the whole pairing computation. The following two points are important to efficiently utilize the datapath and to have limited memory usage:

- control of the number of local variables to limit the size of RAMs;
- control of the dependency between operations, to avoid pipeline bubbles.

For the first point, the step which requires the most live local variables is in the `hard-part` of the final exponentiation. We use the algorithm described by Scott et al. [43], which is to date the fastest way to compute the `hard-part` (Algorithm A.1). To keep its full power while limiting the number of local variables

[5] More recently, Karabina introduced a compressed form for elements in $\mathbb{F}_{p^{12}}$ which require less operations to be squared [3, 28]. Unfortunately, this method involves extra inversions so that it is not suitable for our designs.

(with a classical register allocation technique) we slightly rearranged their original formulae. For the second point, excluding the $\mathbb{F}_p$ inversion where idle states cannot be avoided, the most constrained part is the $\mathbb{F}_{p^2}$ arithmetic in the Miller loop. Therefore we rearranged the projective coordinates addition and doubling formulae to emphasize the inherent parallelism. Formulas can be found in Algorithm 4 and 5 in appendix. On both architectures, we managed to eliminate almost all pipeline bubbles in the Miller loop. Idle states on the multiplier remain less than %1 of the time on both architectures in the pairing computation, excluding the $\mathbb{F}_p$ inversion.

7 Implementation Results and Analysis

7.1 Area

The prototype of the proposed pairing coprocessors were implemented on commercial FPGAs. Table 1 gives the logic utilization of both designs.

Design I We synthesized the first design for $n = 8$ on three different FPGAs :

- EP2C35: A low cost (less than $100) 65 nm node Altera FPGA. It is designed for industrial series production;
- EP2S30: A 65 nm node high end series of Altera. It is picked for the sake of comparison with the Xilinx Virtex-4 used in [15].
- EP3SE50: A 40 nm node high end FPGA. Note that it is the smallest FPGA of the series. BN_{192} ($n = 19$) is also implemented on this device.

The area consumption is given in ALM, the equivalence of the Xilinx Virtex-4 slice, and in LE for the Cyclone, which can be considered as a simple 4×4 LUT with carry chain and registers. We refer the reader to [1] for details.

We let the synthesizer decide how to implement multiplications (with speed constraints). Note that this choice lead to the maximal use of DSP blocks. We used also embedded RAMs: the RAM of each Rower is built the M4k or M9k (depending on the FPGA). We gave no instructions to the design suite for the implementation of the sequencer (containing 20 kB microcode), therefore they were placed in the big available memories. The same design provides support for the two curves defined in 2.2, with the only difference being the content of the RAM blocks (precomputed values and the sequencer).

Design II Design II is implemented on a Xilinx Virtex-6 XC6VLX240T-2 FPGA, which embeds 25×18 DSP slices. As there are 8 Rowers and each Rower contains 4 DSP slices, the total number of DSPs is 32. Data RAMs used in each Rower are implemented with distributed memory blocks. The block RAMs (BRAMs) serve as the microcode sequencer. Thanks to the fine-tuned pipeline, the coprocessor can operate at 250MHz.

Table 1. Logic Utilization

	n	Device	Freq.	Multipliers	Logic Elements	Data Memory	Sequencer
Design I	8	Cyclone II	91 MHz	35 18-bit Mult.	14274 LE	32 M4k	35 M4k
		Stratix II	165 MHz	72 DSP18el	4227 ALMs	32 M4k	1 M512
		Stratix III	165 MHz	72 DSP18el	4233 ALMs	16 M9k	1 M144 +2 M9k
	19	Stratix III	131 MHz	171 DSP18el	9910 ALMs	38 M9k	1 M144 +2 M9k
Design II	8	Virtex-6	250 MHz	32 DSP48E1s	7032 Slices	-	45 18Kb BRAMs

7.2 Performance

Table 2 gives number of cycles used by the sub-functions used in optimal ate pairing on both Design I and Design II. Due to the carefully selected bases and dual mode multiplier, Design II achieved a lower cycle count. The improvement mainly comes from the speedup of RNS reduction: 19 cycles on Design I compared to 12 cycles on Design II when $n = 8$.

7.3 Comparison and Discussion

Table 3 lists the performance of software and hardware implementations reported in recent literature. Both Design I and II achieve a better performance than previous hardware implementations [16, 19, 27]. Due to the use of different platforms, a fair comparison is difficult. Nevertheless, compared with the design of [16] which also uses Virtex-6, Design II achieves a speedup of factor 2. The software implementations achieve very high performance. Although we did not break the software record, the speed of our design is already close to that of software.

The speedup comes mainly from three improvements. First, RNS multiplication has lower complexity than traditional integer multiplications. For example, an 256-bit multiplication on RNS involves 16 33×33 multiplications, while a 256-bit integer multiplication requires 27 32×32 multiplications using Karatsuba's method. Second, the use of lazy reduction reduces the cost of multiplications in extension fields. Third, RNS representation is very parallelization-friendly. Indeed, it only involves relatively small numbers (no long carry propagation) and always uses data in the local memory (no inter-core communication overhead). The performance of both Design I and II has demonstrated the efficiency of RNS in pairing implementations, and confirms with actual implementations on FPGA the analysis of [14].

Table 2. Cycle count in one optimal pairing

	Curve	Mul./Red. in $\mathbb{F}_p$	2T and $g_{(T,T)}(P)$	T+Q and $g_{(T,Q)}(P)$	f^2	$f \cdot g$	Miller's Loop	Final Exp.	Total
Design I	BN_{126}	2 / 19	507	581	384	372	86,530	89,581	176,111
	BN_{128}						92,480	94,101	192,502
	BN_{192}	2 / 41	947	1153	648	636	401,565	388,284	789,849
Design II	BN_{126}	2 / 12	320	430	301	289	61,116	81,995	143,111

Table 3. Performance comparison of software and hardware implementations of pairings

Design	Pairing	Security [bit]	Platform	Algorithm	Area	Freq. [MHz]	Cycle [$\times 10^3$]	Delay [ms]
Design I	optimal ate	126	Altera FPGA (Cyclone II)	RNS Montgomery	14274 LE 35 mult.	91	176	1.93
			Altera FPGA (Stratix III)		4233 A 72 DSPs	165	176	1.07
		192	Altera FPGA (Stratix III)		9910 A 171 DSPs	131	790	6.03
Design II	optimal ate	126	Xilinx FPGA (Virtex-6)	RNS Montgomery	7032 slices 32 DSPs	250	143	0.573
[16]	ate	128	Xilinx FPGA (Virtex-6)	Hybrid Montgomery	4014 slices 42 DSPs	210	336	1.60
	optimal ate						245	1.17
[19]	Tate	128	Xilinx FPGA (Virtex-4)	Blakley	52k Slices	50	1,730	34.6
	ate						1,207	24.2
	optimal ate						821	16.4
[27]	Tate	128	ASIC (130 nm)	Montgomery	97 kGates	338	11,627*	34.4
	ate						7,706*	22.8
	optimal ate						5,340*	15.8
[15]	Tate over $\mathbb{F}_{3^{5\cdot 97}}$	128	Xilinx FPGA (Virtex-4)	-	4755 Slices 7 BRAMs	192	429	2.23
[2]	optimal Eta over $\mathbb{F}_{2^{367}}$	128	Xilinx Virtex-4	-	4518 Slices	220	774*	3.52
[23]	ate	128	64-bit Core2	Montgomery	-	2400	15,000	6.25
	optimal ate						10,000	4.17
[20]	ate	128	64-bit Core2	Montgomery		2400	14,429	6.01
[36]	optimal ate	128	Core2 Quad	Hybrid Mult.	-	2394	4,470	1.86
[8]	optimal ate	126	Core i7	Montgomery	-	2800	2,330	0.83
[3]	optimal ate	126	Phenom II	Montgomery	-	3000	1,562	0.52
[4]	η_T over $\mathbb{F}_{21223}$	128	Xeon	-	-	2000	3,020	1.51
[9]	η_T over $\mathbb{F}_{3^{509}}$	128	Core i7	-	-	2900	5,423	1.87
[2]	opt. Eta $\mathbb{F}_{2^{367}}$	128	Core i5	-	-	2530	2,440	0.96

* Estimated by the authors.

8 Conclusions

In this paper, we demonstrated that RNS together with lazy reduction is a really competitive approach for pairing computation in hardware. Thanks to the use of RNS, both datapath and memory can be nicely parallelized. The results show that both designs we proposed achieve higher speed than all previous designs in hardware. The fastest version computes an optimal ate pairing at 126-bit security level in 0.573 ms, which is 2 times faster than all previous hardware implementations. Moreover, we also reported the first hardware pairing implementation at 192-bit security level.

For future work, we would like to further optimize the pipeline structure to achieve higher speed, particularly for 192-bit optimal pairing. We would like implement using RNS a pairing processor that provides 256-bit security. For these levels, BN curves may become less competitive, and have to be compared to other approaches such like KSS curves [26]. Also, as shown in Section 5, carefully selected RNS bases can help to reduce the complexity of RNS base extension. However, it is not clear how much we can benefit from it when the

number of channels is relatively large, and this is definitely part of the design exploration for the implementation of pairings at higher security levels.

References

1. Altera web site. http://www.altera.com.
2. D. Aranha, J.-L. Beuchat, J. Detrey, and N. Estibals. Optimal eta pairing on supersingular genus-2 binary hyperelliptic curves. Cryptology ePrint Archive, Report 2010/559, 2010. http://eprint.iacr.org/.
3. D. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 48–68. Springer, Heidelberg, 2011.
4. D. Aranha, J. López, and D. Hankerson. High-speed parallel software implementation of the η_T pairing. In *Topics in Cryptology - CT-RSA 2010*, volume 5985 of *LNCS*, pages 89–105. Springer, Heidelberg, 2010.
5. J.-C. Bajard, L.-S. Didier, and P. Kornerup. An RNS Montgomery modular multiplication algorithm. *Computers, IEEE Transactions on*, 47(7):766–776, 1998.
6. P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology CRYPTO 2002*, volume 2442 of *LNCS*, pages 354–369. Springer, Heidelberg, 2002.
7. P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected areas in cryptography–SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, 2006.
8. J.-L. Beuchat, J. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *LNCS*, pages 21–39. Springer, Heidelberg, 2010.
9. J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core implementation of the Tate pairing over supersingular elliptic curves. In *Cryptology and Network Security*, volume 5888 of *LNCS*, pages 413–432. Springer, Heidelberg, 2009.
10. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology–CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, 2001.
11. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
12. J. C. Cha and J. H. Cheon. An identity-based signature from gap Diffie-Hellman groups. In *International Workshop on Theory and Practice in Public Key Cryptography – PKC 2003*, volume 2567, pages 18–30, Springer, Heidelberg, 2003.
13. C. Costello, T. Lange, and M. Naehrig. Faster pairing computations on curves with high-degree twists. In *Public Key Cryptography PKC 2010*, volume 6056 of *LNCS*, pages 224–242. Springer, Heidelberg, 2010.
14. S. Duquesne. RNS arithmetic in $\mathbb{F}_{p^k}$ and application to fast pairing computation. Cryptology ePrint Archive, Report 2010/555, 2010. http://eprint.iacr.org/, to appear in Journal of Mathematical Cryptology.
15. N. Estibals. Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves. In *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *LNCS*, pages 397–416. Springer, Heidelberg, 2010.

16. J. Fan, F. Vercauteren, and I. Verbauwhede. Efficient hardware implementation of $\mathbb{F}_p$-arithmetic for pairing-friendly curves. *Computers, IEEE Transactions on*, PP(99):1, 2011.
17. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23:224–280, 2010.
18. G. Frey and H.G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of computation*, 62(206):865–874, 1994.
19. S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury. High speed flexible pairing cryptoprocessor on FPGA platform. In *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *LNCS*, pages 450–466. Springer, Heidelberg, 2010.
20. P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 35–50. Springer, Heidelberg, 2009.
21. R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. *Public Key Cryptography–PKC 2010*, 6056:209–223, 2010.
22. N. Guillermin. A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over $\mathbb{F}_p$. In *Cryptographic Hardware and Embedded Systems-CHES 2010*, volume 6225 of *LNCS*, pages 48–64, Springer, Heidelberg, 2010.
23. D. Hankerson, A. Menezes, and M. Scott. *Software Implementation of Pairings*, volume 2 of *Cryptology and Information Security Series*, pages 188–206. IOS Press, M. Joye and G. Neven edition, 2009.
24. F. Hess, N.P. Smart, and F. Vercauteren. The Eta pairing revisited. *Information Theory, IEEE Transactions on*, 52(10):4595 –4602, 2006.
25. A. Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17:263–276, 2004.
26. E. J. Kachisa, E. F. Schaefer, and M. Scott. Constructing brezing-weng pairing-friendly elliptic curves using elements in the cyclotomic field. In *Pairing-Based Cryptography - Pairing 2008*, volume 5209 of *LNCS*, pages 126–135, Springer, Heidelberg, 2008.
27. D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar. Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, volume 5747 of *LNCS*, pages 254–271. Springer, Heidelberg, 2009.
28. K. Karabina. Squaring in cyclotomic subgroups. Cryptology ePrint Archive, Report 2010/542, 2010. http://eprint.iacr.org/.
29. S. Kawamura, M. Koike, F. Sano, and A. Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In *Advances in Cryptology EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 523–538. Springer, Heidelberg, 2000.
30. N. Koblitz. Elliptic Curve Cryptosystem. *Math. Comp.*, 48:203–209, 1987.
31. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. *Cryptography and coding*, 3796:13–36, 2005.
32. E. Lee, H.-S. Lee, and C.-M. Park. Efficient and generalized pairing computation on abelian varieties. *Information Theory, IEEE Transactions on*, 55(4):1793 –1803, 2009.
33. A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *Information Theory, IEEE Transactions on*, 39(5):1639–1646, 1993.
34. V. Miller. Uses of Elliptic Curves in Cryptography. In *Advances in Cryptology: Proceedings of CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, Heidelberg, 1985.

35. V. S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17:235–261, 2004. 10.1007/s00145-004-0315-8.
36. M. Naehrig, R. Niederhagen, and P. Schwabe. New software speed records for cryptographic pairings. In *Progress in Cryptology LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 109–123. Springer, Heidelberg, 2010.
37. H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura. Implementation of RSA algorithm based on RNS montgomery multiplication. In *Cryptographic Hardware and Embedded Systems-CHES 2001*, volume 2162 of *LNCS*, pages 364–376, Springer, Heidelberg, 2001.
38. National Institute of Standard and technology. Key management, 2007. http://csrc.nist.gov/groups/ST/toolkit/key_management.html.
39. G.C.C.F. Pereira, M.A. Simplício, M. Naehrig, and P.S.L.M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 2011.
40. K. Posch and R. Posch. Base extension using a convolution sum in residue number systems. *Computing*, 50:93–104, 1993.
41. R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
42. M. Scott. Implementing cryptographic pairings. In *Pairing-Based Cryptography - Pairing 2007*, volume 4575 of *LNCS*, pages 117–196. Springer, Heidelberg, 2007.
43. M. Scott, N. Benger, M. Charlemagne, L. Dominguez Perez, and E. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. *Pairing-Based Cryptography–Pairing 2009*, volume 5671 of *LNCS*, pages 78–88, Springer, Heidelberg, 2009.
44. F. Vercauteren. Optimal pairings. *Information Theory, IEEE Transactions on*, 56(1):455 –461, 2010.

A Appendix

A.1 Sub-routines for Optimal Ate Pairing

Algorithm 4 `dbl`, doubling step

Require: $T = (X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T) \in E(\mathbb{F}_{p^{12}})$ with X_T, Y_T and $Z_T \in \mathbb{F}_{p^2}$, $P = (x_P, y_P) \in E(\mathbb{F}_p)$.

Ensure: The point $2T$ and the evaluation in P of the equation of the tangent line in T to the curve up to multiplicative factors in $\mathbb{F}_{p^2}$.

1: $B \leftarrow Y_T^2, C \leftarrow 3Z_T^2, D \leftarrow 2X_TY_T$
2: $F \leftarrow B + 3\mathbf{i}C, G \leftarrow B - 3\mathbf{i}C, H \leftarrow 3C, t_3 \leftarrow B + \mathbf{i}C, A \leftarrow X_T^2, E \leftarrow 2Y_TZ_Y$
3: $X_{2T} \leftarrow DF, Y_{2T} \leftarrow G^2 + 4HC, Z_{2T} \leftarrow 4BE, t_0 \leftarrow Ey_P, t_1 \leftarrow -3Ax_P$
4: **return** $(X_{2T}\boldsymbol{\gamma}^2, Y_{2T}\boldsymbol{\gamma}^3, Z_{2T}), t_0 + t_1\boldsymbol{\gamma} + t_3\boldsymbol{\gamma}^3$

These algorithms are specific to the BN curve $y^2 = x^3 + 2$ and the tower of extensions given in Subsection 2.1. Jacobian coordinates are usually used for pairing computations [8, 23, 36] but projective coordinates are more interesting

in our case [3, 13]. Using the degree 6 twist on the curve, the point Q can be written as $(x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3)$ with x_Q and $y_Q \in \mathbb{F}_{p^2}$. The "doubling" step of the Miller loop consists of two stages: the doubling of a temporary projective point $T = (X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T)$ with X_T, Y_T and $Z_T \in \mathbb{F}_{p^2}$ and the evaluation in P of the tangent line in T to the curve. This is given in Algorithm 4 where the classical formulae are rearranged in a way to highlight the reductions (every temporary result needs a reduction except F, G, H and t_3), and the inherent parallelism in the local variables (each line can be implemented in random order). This is important to avoid idle states in Cox-Rower. The total cost of this step is 4 multiplications, 5 squarings, 8 reductions in $\mathbb{F}_{p^2}$ and 2 modular multiplications of an element of $\mathbb{F}_{p^2}$ by an element of $\mathbb{F}_p$. Note that multiplications like $2X_TY_T$ can be transformed into squaring of $(X_T + Y_T)$ at the cost of some extra additions [3, 13], thus it is not interesting for our design. In the same way, the cost of the addition step given by Algorithm 5 is 11 multiplications, 2 squarings, 11 reductions in $\mathbb{F}_{p^2}$ and 2 modular multiplications of an element of $\mathbb{F}_{p^2}$ by an element of $\mathbb{F}_p$.

Algorithm 5 `add`, addition step

Require: $T = (X_T\boldsymbol{\gamma}^2, Y_T\boldsymbol{\gamma}^3, Z_T) \in E(\mathbb{F}_{p^{12}})$ with X_T, Y_T and $Z_T \in \mathbb{F}_{p^2}$, $Q = (x_Q\boldsymbol{\gamma}^2, y_Q\boldsymbol{\gamma}^3) \in E(\mathbb{F}_{p^{12}})$, $P = (x_P, y_P) \in E(\mathbb{F}_p)$.
Ensure: The point $T + Q$ and the evaluation in P of the equation of the line passing through T and Q up to multiplicative factors in $\mathbb{F}_{p^2}$.
1: $E \leftarrow x_QZ_T - X_T, F \leftarrow y_QZ_T - Y_T$
2: $E_2 \leftarrow E^2, F_2 \leftarrow F^2$
3: $A \leftarrow F_2Z_T - 2X_TE_2 - EE_2, B \leftarrow X_TE_2, E_3 \leftarrow EE_2$
4: $X_{T+Q} \leftarrow AE, Z_{T+Q} \leftarrow Z_TE_3, t_3 \leftarrow Fx_Q - Ey_Q$
5: $Y_{T+Q} \leftarrow F(B - A) - y_QE_3, t_0 \leftarrow Ey_P, t_1 \leftarrow -Fx_P$
6: **return** $(X_{T+Q}\boldsymbol{\gamma}^2, Y_{T+Q}\boldsymbol{\gamma}^3, Z_{T+Q}), t_0 + t_1\boldsymbol{\gamma} + t_3\boldsymbol{\gamma}^3$

Algorithm 6 `hard-part`, hard part of the final exponentiation according [43]

Require: $f \in \mathbb{F}_{p^{12}}$ of order $p^4 - p^2 + 1$, $x = |u|$.
Ensure: $f^{(p^4-p^2+1)/\ell}$ with p and ℓ as in 2.1.
{Computation of the y_i}
1: $y_0 \leftarrow f^p f^{p^2} f^{p^3}$, $y_1 \leftarrow f^x$, $y_3 \leftarrow y_1^x$, $y_5 \leftarrow y_3^x$, $y_4 \leftarrow y_5^p$, $y_6 \leftarrow y_4y_5 \left(= f^{x^3}\left(f^{x^3}\right)^p\right)$
2: $y_5 \leftarrow y_3^p$, $y_2 \leftarrow y_5^{-1}$, $y_4 \leftarrow y_1y_2 \left(= f^x/\left(f^{x^2}\right)^p\right)$
3: $y_2 \leftarrow y_5^p \left(= \left(f^{x^2}\right)^{p^2}\right)$, $y_5 \leftarrow y_3^{-1} \left(= 1/f^{x^2}\right)$, $y_3 \leftarrow y_1^p\,((m^x)^p)$, $y_1 \leftarrow f^{-1}$
{Multi-addition chain for computing $y_0.y_1^2.y_2^6.y_3^{12}.y_4^{18}.y_5^{30}.y_6^{36}$}
4: $t_0 \leftarrow y_6^2$, $t_0 \leftarrow t_0y_4$, $t_0 \leftarrow t_0y_5$, $t_1 \leftarrow y_3y_5$, $t_1 \leftarrow t_1t_0$, $t_0 \leftarrow t_0y_2$, $t_1 \leftarrow t_1^2$
5: $t_1 \leftarrow t_1t_0$, $t_1 \leftarrow t_1^2$, $t_0 \leftarrow t_1y_1$, $t_1 \leftarrow t_1y_0$, $t_0 \leftarrow t_0^2$, $t_0 \leftarrow t_0t_1$
6: **return** t_0

Algorithm 7 Squaring during final exponentiation (hard-part) [21]

Require: $A = \sum_{i=0}^{5} a_i \boldsymbol{\gamma}^i \in \mathbb{F}_{p^{12}}$ with $a_i \in \mathbb{F}_{p^2}, 0 \leq i \leq 5$
Ensure: A^2
$A_0 \leftarrow 3a_0^2 + 3(1+\mathbf{i})a_3^2 - 2a_0, A_1 \leftarrow 6(1+\mathbf{i})a_2a_5 + 2a_1$
$A_2 \leftarrow 3a_1^2 + 3(1+\mathbf{i})a_4^2 - 2a_2, A_3 \leftarrow 6a_0a_3 + 2a_3$
$A_4 \leftarrow 3a_2^2 + 3(1+\mathbf{i})a_5^2 - 2a_4, A_5 \leftarrow 6a_1a_4 + 2a_5$
return $\sum_{i=0}^{5} A_i \boldsymbol{\gamma}^i$

A.2 RNS Parameter Selection

Since p should be around 254 bits, we set n to be 8 and the moduli are chosen near 2^{33}. We consider for $i, j \in [1,n]$ the following issues: (1) bitlength of $|B_i|_{c_j}$ and $|C_i|_{b_j}$; (2) Hamming weight of b_i and c_j. A simple (bounded) exhaustive search program returns the bases shown in Section 5.

We denote the bitlength of all $\tilde{B}_{i,j}$ in a length matrix, $L_{\tilde{B}}$, and the bitlength matrix of all $\tilde{C}_{i,j}$ as $L_{\tilde{C}}$. For the selected bases, we have the following $L_{\tilde{B}}$ and $L_{\tilde{C}}$.

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

After truncating the least significant zeros, we get the following bitlength, denoted as $L_{\tilde{B}'}$ and $L_{\tilde{C}'}$. Note that we applied the same truncation parameter for all the numbers in the same row. The number of zeros truncated is chosen as

$$min\{z(B_{1,j}), z(B_{2,j}), \cdots, z(B_{n,j})\}$$

where $z(B_{i,j})$ gives the number of zeros at the LSBs of $B_{i,j}$.

$$L_{\tilde{B}'} = \begin{pmatrix} 23 & 20 & 21 & 20 & 21 & 20 & 18 & 19 \\ 12 & 10 & 12 & 10 & 11 & 10 & 8 & 9 \\ 16 & 14 & 14 & 13 & 14 & 13 & 12 & 13 \\ 15 & 14 & 15 & 16 & 15 & 16 & 13 & 18 \\ 17 & 18 & 16 & 16 & 16 & 16 & 16 & 15 \\ 20 & 21 & 20 & 19 & 19 & 19 & 21 & 19 \\ 19 & 20 & 19 & 19 & 19 & 19 & 21 & 19 \\ 19 & 19 & 19 & 18 & 19 & 19 & 23 & 18 \end{pmatrix}, \; L_{\tilde{C}'} = \begin{pmatrix} 14 & 13 & 13 & 10 & 11 & 10 & 10 & 9 \\ 15 & 15 & 16 & 14 & 16 & 15 & 14 & 14 \\ 16 & 17 & 15 & 14 & 14 & 13 & 13 & 13 \\ 20 & 21 & 20 & 21 & 19 & 19 & 19 & 18 \\ 20 & 20 & 19 & 19 & 18 & 18 & 18 & 17 \\ 21 & 21 & 21 & 21 & 20 & 19 & 19 & 19 \\ 17 & 17 & 17 & 16 & 18 & 19 & 19 & 21 \\ 20 & 20 & 20 & 23 & 19 & 19 & 19 & 19 \end{pmatrix}$$

Now all of $\tilde{B}'_{i,j}$ and $\tilde{C}'_{i,j}$ are less than 25 bits, and they fit in one operand of an FPGA DSP slice, while the standard 34-bit operands do not. In fact, in the implementation we do not truncate more zeros as far as all elements in that row fit in 25 bits.