An Algorithmic and Architectural Study on Montgomery Exponentiation in RNS

Filippo Gandino, *Member*, *IEEE*, Fabrizio Lamberti, *Member*, *IEEE*, Gianluca Paravati, Jean-Claude Bajard, and Paolo Montuschi, *Senior Member*, *IEEE*

Abstract—The modular exponentiation on large numbers is computationally intensive. An effective way for performing this operation consists in using Montgomery exponentiation in the Residue Number System (RNS). This paper presents an algorithmic and architectural study of such exponentiation approach. From the algorithmic point of view, new and state-of-the-art opportunities that come from the reorganization of operations and precomputations are considered. From the architectural perspective, the design opportunities offered by well-known computer arithmetic techniques are studied, with the aim of developing an efficient arithmetic cell architecture. Furthermore, since the use of efficient RNS bases with a low Hamming weight are being considered with ever more interest, four additional cell architectures specifically tailored to these bases are developed and the tradeoff between benefits and drawbacks is carefully explored. An overall comparison among all the considered algorithmic approaches and cell architectures is presented, with the aim of providing the reader with an extensive overview of the Montgomery exponentiation opportunities in RNS.

Index Terms—RNS, montgomery reduction, modular exponentiation, modular multiplication.

1 Introduction

The hardware implementation of modular exponentiation for very large integers plays an important role in various fields, being security among the most notable ones. In recent years, many research efforts specifically focused on the hardware implementation of Montgomery exponentiation (ME) in the residue number system (RNS). In RNS, very long integer multiplications and additions can be split in independent short integer operations. However, other operations such as division and modular reduction may be difficult to execute [1]. ME, which is based on Montgomery multiplication (MM) [2], can be performed by avoiding the standard modular reduction approach.

A number of versions of the RNS Montgomery Exponentiation (RNS ME) algorithm have been proposed, aimed at reducing the delay of exponentiation (e.g., [3], [4], [5]). Most of the above approaches deal with the base extension (BE) part of the algorithms, since this operation, which calculates a number on a different RNS base, contributes in large part to the overall computational effort of RNS ME.

An RNS ME technique in the context of RSA was proposed in [4] by Kawamura et al. This technique uses a new BE algorithm, characterized by a summation that provides a result modulo a small multiple of the base, which is corrected after the sum of each element. In [6], further details and an architecture were presented. The architecture was compared with non-RNS approaches, showing better performance.

In [5], Bajard and Imbert described another ME approach based on [7] and exploiting two BE techniques: a new approximated BE and the BE algorithm proposed in [8], where the result is approximated and corrected by using an extra modulo.

In this paper, a comprehensive algorithmic and architectural study on RNS ME is presented. The paper is based on an earlier work presented in [9], where new and state-of-the-art approaches for reorganizing operations and for exploiting precomputation were combined and analyzed, and two RNS ME algorithms suitable to the BE approaches used in [4] and [5] were discussed. A previous study limited to the internal reorganization of the RNS Montgomery reduction was presented in [10]. Moreover, the idea of rearranging internal operations and precomputed values was simultaneously exploited by Guillermin in [11], [12], where a reorganized RNS Montgomery reduction algorithm with a BE approach based on [4] was applied to elliptic curve cryptography. The algorithmic analysis in the current work extends previous studies, and considers operations both internal and external to the RNS Montgomery Multiplication (RNS MM) with the aim of limiting the number of computations required during the Montgomery reduction.

The work in [9] also included a study on the cell architecture presented in [6], and proposed a novel cell architecture suitable to the approach used in [5] (which was evaluated by means of a theoretical analysis based on equivalent gates delay and area cost). In the current work, the architectural perspective is significantly extended, by exploiting the outcomes of the former analysis as well as the synergies between algorithmic and architectural aspects. In particular, on the one hand, the use of techniques well assessed in computer architecture design is explored,

- *F. Gandino, F. Lamberti, G. Paravati, and P. Montuschi are with the Dipartimento di Automatica e Informatica (DAUIN), Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy. E-mail: {filippo.gandino, fabrizio.lamberti, gianluca.paravati, paolo.montuschi}@polito.it.*
- *J.-C. Bajard is with LIP6, CNRS, Université Pierre et Marie Curie, Boite courrier 169, Couloir 26-00, etage 3, Bureau 315, 4 place Jussieu, 75252 Paris cedex 05. E-mail: jean-claude.bajard@lip6.fr.*

Manuscript received 22 Sept. 2011; revised 26 Jan. 2012; accepted 25 Mar. 2012; published online 11 Apr. 2012.
Recommended for acceptance by E. Antelo, D. Hough, and P. Ienne.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2011-09-0678.
Digital Object Identifier no. 10.1109/TC.2012.84.

and a new cell architecture exploiting pipeline and redundancy is proposed. On the other hand, efficient RNS bases studied in [13], [14] are considered, and four additional cell architectures are developed where a totally different design based on additions rather than on modular reductions is exploited. Furthermore, a comprehensive comparison encompassing reorganized algorithms and newly designed cells is carried out, based on algorithmic analysis and logic synthesis. Results show that, without additional constraints on the representable numbers, it is possible to achieve a 41.1 percent reduction in delay, with a 17.6 percent increase in area, with respect to [6]. Moreover, further improvements (up to a 41.8 percent reduction in delay with a 2.8 percent reduction in area) can be reached at the cost of stricter constraints on the size of the representable numbers.

The remaining of the paper is organized as follows: In Section 2, the impact of reorganization of operations and preprocessing is analyzed. In Section 3, the architectural study is discussed, by analyzing in detail the design of the novel cell architectures. In Section 4, an overall comparison is presented, considering all the algorithms and architectures described in the paper. Finally, in Section 5, conclusions are drawn.

2 Algorithmic Study

In this section, a suitable reorganization of the operations and of precomputations capable of reducing the number of multiplications in RNS ME algorithms is described. The RNS ME corresponds to a classic square and multiply algorithm, where RNS MMs are iterated in a loop. The RNS MM follows the normal Montgomery multiplication approach, but it is adapted to RNS. The main difference is the presence of two BEs, which are required by RNS MM in order to execute the modular reduction. The BE corresponds to an algorithm that calculates a value represented on an RNS base on a different base. Since the RNS ME uses the RNS MM that, in turn, requires the BE, after a brief background on RNS the three algorithms will be discussed separately by following a top-down presentation approach.

The study first identifies in an analytical way the reduction in the number of multiplications needed by the reorganized algorithms. Then, the multiplications are classified and their weight is analyzed in detail (by making reference to the concept of *multiplication step*, i.e., to a set of k multiplications distributed on k cells), with the aim of precisely determining the impact of such reorganization in the perspective of pipelining and parallelization opportunities offered by RNS. The algorithms will be discussed referring to symbols reported in Table 1.

2.1 Mathematical Background

RNS allows long integers to be represented as sets of short integers. Considering the base $\mathcal{A} = (a_1, a_2, \dots, a_k)$, composed of k relative prime numbers (where k is the base size), any number x with $0 \leq x < A = \prod_{i=1}^{k} a_i$ is uniquely represented by a sequence of positive integers $(x_1, x_2, \dots, x_k)$, where $x_i = |x|_{a_i}$, $\forall i : 1 \leq i \leq k$.

In RNS, multiplications, additions, and subtractions can be carried out independently and in parallel for each base element, limiting both the operations and the carry propagation to the bits of each independent element.

TABLE 1
List of Symbols

Context	Symbol	Meaning
RNS bases	$\mathcal{A}$	RNS base
	$\mathcal{B}$	RNS base
	k	Number of base elements
	r	Number of bits of each base element
	a_j	j^{th} element of the base $\mathcal{A}; \forall j; 1 \leq j \leq k$
	b_i	i^{th} element of the base $\mathcal{B}; \forall i; 1 \leq i \leq k$
	A	$\prod_{j=1}^{k} a_j$
	B	$\prod_{i=1}^{k} b_i$
	A_j	$\frac{A}{a_j}; \forall j; 1 \leq j \leq k$
	B_i	$\frac{B}{b_i}; \forall i; 1 \leq i \leq k$
	A_j^{-1}	Multiplicative inverse of A_j on a_j
	B_i^{-1}	Multiplicative inverse of B_i on b_i
	B_A^{-1}	Multiplicative inverse of B on A
	B_N^{-1}	Multiplicative inverse of B on N
	c_i	$2^r - a_i$
	h	Maximum number of bits of c_i
ME	g	Number of bits of the exponent
MM	R	Montgomery number
	N	Modulo of the operation
BE Values	λ	Approximation $\frac{\tilde{x}-x}{B}$
Architecture	p	Stages of pipeline
	M	Parallel multipliers
Kawamura et al. BE [4]	c	Approximated value of $\frac{q}{A}$
	f	Floor of c; $(\lfloor c \rfloor)$
	α	Initial value of c; $\{0; 0.5\}$
	ϱ	Number of accumulated bits of q_i
Bajard et al. BE [5]	a_r	Redundant base element
	A_r^{-1}	Multiplicative inverse of A on a_r
Accents	$\widetilde{tilde}$	Approximated values
	$\widehat{hat}$	Values multiplied by A_j^{-1} in $\mathcal{A}$
Operations and Relations	$\lvert x \rvert_y$	$x \bmod y$
	$<<$	Left shift
	$\equiv$	Equivalent

However, in RNS, other operations like overflow detection, division, and modular reduction are more computationally intensive than in other representations. For instance, an exact division $\frac{x}{y}$ can be executed by means of the multiplication of x by the multiplicative inverse of y, modulo A, but this operation can be performed without affecting the size of the representable values only if the greatest common divisor $\gcd(y, A) = 1$. If this condition is not valid, a BE has to be operated, requiring a significant computational effort.

An efficient BE technique is based on the Chinese Remainder Theorem (CRT) [15]. CRT can be used to convert a value x from an RNS base to a radix system. The conversion expression is

$$x = \left| \sum_{i=1}^{k} |x_i A_i^{-1}|_{a_i} A_i \right|_A , \qquad (1)$$

where $A_i = \frac{A}{a_i}$, A_i^{-1} is the multiplicative inverse of A_i on a_i and $\sum_{i=1}^{k} |x_i A_i^{-1}|_{a_i} A_i$ is equal to $x + \lambda A$ (with $\lambda < k$). In order to complete a BE, the modular reductions of x by the elements of the new base must be performed.

In this study, A_i and A_i^{-1} are always used on the base element a_i, or on a different base. Since they are never used on a base element a_j of $\mathcal{A}$ with $j \neq i$, in order to simplify the notation in the following A_i in $\mathcal{A}$ and A_i^{-1} in $\mathcal{A}$ will be used to indicate $(|A_0|_{a_0}, |A_1|_{a_1}, \dots, |A_k|_{a_k})$ and $(|A_0^{-1}|_{a_0}, |A_1^{-1}|_{a_1}, \dots, |A_k^{-1}|_{a_k})$, respectively.

2.2 Reorganization of the Operations

The algorithmic study reported in this work moves from considering that, in the two state-of-the-art RNS ME algorithms [4], [5], a common feature can be identified, i.e.,

```
Algorithm 1: State-of-the-art RNS ME
Input: $x$ in $\mathcal{A}\cup\mathcal{B}\cup a_r$ and $e=(e_{g-1}...e_1e_0)_b$, such that† $A=\prod_{j=1}^{k}a_j$, $B=\prod_{i=1}^{k}b_i$, and $\gcd(A,B)=1$, $\gcd(N,B)=1$, $0\le xy<NB$
Output: $z=|x^e|_N$ in $\mathcal{A}\cup\mathcal{B}$
Precomputation: $|B^2|_N$, $|B|_N$ in $\mathcal{A}\cup\mathcal{B}\cup a_r$
-
1: $\bar{x}\leftarrow$ MM $(x,|B^2|_N)$
2: $z=|B|_N$ in $\mathcal{A}\cup\mathcal{B}\cup a_r$
3: for $i$ from $g-1$ down to 0 do
4:     $z\leftarrow$ MM $(z,z)$
5:     if $e_i=1$ then
6:         $z\leftarrow$ MM$(z,\bar{x})$
7:     end if
8: end for
9: $z\leftarrow$ MM$(z,1)$
-
```

```
Algorithm 2: Reorganized RNS ME
Input: $x$ in $\mathcal{A}\cup\mathcal{B}\cup a_r$ and $e=(e_{g-1}...e_1e_0)_b$, such that† $A=\prod_{j=1}^{k}a_j$, $B=\prod_{i=1}^{k}b_i$, and $\gcd(A,B)=1$, $\gcd(N,B)=1$, $0\le xy<NB$
Output: $z=|x^e|_N$ in $\mathcal{A}\cup\mathcal{B}$
Precomputation: $\widehat{|B^2|_N}$, $\widehat{|B|_N}$ in $\mathcal{A}\cup\mathcal{B}\cup a_r$, $|A_j|_{a_j}$ and $|A_j^{-1}|_{a_j}$ for $j=1...k$
1: $\hat{x}_{a_j}=|x_{a_j}\cdot A_j^{-1}|_{a_j}$ for $j=1...k$
2: $\hat{\bar{x}}\leftarrow$ MM $(\hat{x},\widehat{|B^2|_N})$
3: $\hat{z}=\widehat{|B|_N}$ in $\mathcal{A}\cup\mathcal{B}\cup a_r$
4: for $i$ from $g-1$ down to 0 do
5:     $\hat{z}\leftarrow$ MM $(\hat{z},\hat{z})$
6:     if $e_i=1$ then
7:         $\hat{z}\leftarrow$ MM$(\hat{z},\hat{\bar{x}})$
8:     end if
9: end for
10: $\hat{z}\leftarrow$ MM$(\hat{z},\hat{1})$
11: $z_{a_j}=|\hat{z}_{a_j}\cdot A_j|_{a_j}$ for $j=1...k$
```

† With Kawamura et al. BE approach [4], $B>5N$, $A>4N$; with Bajard et al. BE approach [5], $B>(k+1)^2N$, $A>(k+1)N$

Fig. 1. The state-of-the-art and the reorganized RNS Montgomery exponentiation algorithms.

the presence of a significant number of multiplications of a partial result by a precomputed value.

Taking into account the above characteristic, it is observed that, by exploiting the commutative property, a sequence of two multiplications of a partial result by a precomputed value, e.g., $(x\times k_1)\times k_2$, could be substituted by one multiplication of the original value by the product of the two precomputed values, i.e., $x\times k_3$, with $k_3=k_1\times k_2$. Therefore, by precomputing a different value, it is possible to decrease the number of multiplications and reach the same result. By considering other arithmetic properties, e.g., the distributive and associative ones, operations involved in the considered algorithms could be arranged in a more effective way.

The reorganization of operations and precomputed values internal to the RNS MM has been already exploited in various studies [10], [11], [12]. This section analyzes reorganized algorithms for the RNS ME (where internal reorganizations are combined with external ones by following the approach proposed in [9]), and compares them to the state-of-the-art algorithms in [4], [5].

2.3 RNS Montgomery Exponentiation

ME is carried out by iterating MM, where MM ($|x\times y|_N$) gives $w\equiv xyR^{-1}(\text{mod}N)$, with R the Montgomery constant. ME computes $|x^e|_N$ at the maximum cost of $2\log_2 e+2$ MMs. Considering $\bar{x}$ and $\bar{y}$ such that $\bar{x}=|xR|_N$ and $\bar{y}=|yR|_N$, then $z=|\bar{x}\bar{y}R^{-1}|_N=|xyR|_N$. Therefore, the exponentiation can be executed by iterating MM on $\bar{x}$. ME provides the exact result, or the results plus N; hence, a final comparison and a possible subtraction are required.

The state-of-the-art and reorganized RNS ME algorithms are shown in Fig. 1 ([4] and [5] are discussed together since, although two different RNS ME approaches are exploited, the main part of the algorithm is common). RNS ME is performed on two RNS bases, $\mathcal{A}=(a_1,\dots,a_k)$ and $\mathcal{B}=(b_1,\dots,b_k)$, such that $A=\prod_{j=1}^{k}a_j$, $B=\prod_{i=1}^{k}b_i$, and $\gcd(A,B)=1$. B is used as the Montgomery constant. Since $R=B$, $|B|_N$ and $|B^2|_N$ must be precomputed. Step 1 in Algorithm 1 [4], [5] calculates $\bar{x}$. Step 2 initializes the exponentiation process, which is executed in the loop from Steps 3 to 8.

In the reorganized algorithm (Algorithm 2), two multiplication steps are added outside the loop. Before entering the loop, the values on base $\mathcal{A}$ are multiplied by A_j^{-1} (Step 1). This multiplication is performed on base $\mathcal{A}$ in the state-of-the-art BEs at the end of RNS MM, in order to extend the final result from $\mathcal{A}$ to $\mathcal{B}$. With the aim of saving a multiplication step, in the reorganized algorithm the correct result on $\mathcal{A}$ is not provided, directly calculating it multiplied by A_j^{-1}. All the input values in $\mathcal{A}$ of the RNS MM are represented in a new notation where all the values are premultiplied by A_j^{-1} (and represented with a $\widehat{hat}$ accent). This new notation is stable for the addition and our MM. After the loop, a multiplication by A_j is executed to reach the final result of the exponentiation (that can be either the exact result, or the results plus a multiple of N).

2.4 RNS Montgomery Multiplication

In the state-of-the-art algorithms [4], [7], RNS MM is performed on two RNS bases, $\mathcal{A}$ and $\mathcal{B}$. Since B is used as the Montgomery constant, B_A^{-1} must be precomputed on the base $\mathcal{A}$, where B_A^{-1} is the multiplicative inverse on A of B.

In Fig. 2, the state-of-the-art and reorganized RNS MM algorithms are presented (and, as before, [4] and [7] are treated together). B is used as the Montgomery constant; thus, by executing the multiplication of Step 3 in $\mathcal{B}$, the modular reduction by B is immediate. Since a division by B is required, which must be executed in a base composed by elements relatively prime to B, the BE to $\mathcal{A}$ of Step 4 is performed. The multiplication in Step 5 and the addition in Step 6 are only executed on $\mathcal{A}$, since the result of Step 6 on $\mathcal{B}$ would be equal to 0. The multiplication by B^{-1} in Step 7, which corresponds to the division, is only performed on $\mathcal{A}$, as previously described. A final BE to $\mathcal{B}$ is required, in order to reach a valid result that could be passed in input to other MMs.

The main difference between [4] and [7] is in the BE technique used. Both the techniques are based on (1) and avoid the modular reduction by A, which is computationally intensive. Without the modular reduction, which would give x as a final result, the partial result is equal to

```
Algorithm 3: State-of-the-art RNS MM
Input: $x$, $y$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$
Output: $w \equiv xyB_N^{-1} \pmod N$,
    $w < 2N$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$
Precomputation: $B_A^{-1}$, $N$ in $\mathcal{A} \cup a_r$; $N^{-1}$ in $\mathcal{B}$
 1: $s = x \cdot y$ in $\mathcal{B}$
 2: $s = x \cdot y$ in $\mathcal{A} \cup a_r$
 3: $u = s \cdot (-N^{-1})$ in $\mathcal{B}$
 4: $u$ in $\mathcal{A} \cup a_r \Leftarrow$ BE1($u$ in $\mathcal{B}$)
 5: $t = u \cdot N$ in $\mathcal{A} \cup a_r$
 6: $v = s + t$ in $\mathcal{A} \cup a_r$
 7: $w = v \cdot B_A^{-1}$ in $\mathcal{A} \cup a_r$
 8: $w$ in $\mathcal{B} \Leftarrow$ BE2($w$ in $\mathcal{A} \cup a_r$)
```

```
Algorithm 4: Reorganized RNS MM
Input: $\hat{x}$, $\hat{y}$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$
Output: $w \equiv xyB_N^{-1} \pmod N), w < 2N$ in $\mathcal{B} \cup a_r$,
    $\hat{w} \equiv xyB_N^{-1}A_j^{-1} \pmod N$ in $\mathcal{A}$
Precomputation:
 1: $s = x \cdot y$ in $\mathcal{B} \cup a_r$
 2: $\hat{\hat{s}} = \hat{x} \cdot \hat{y}$ in $\mathcal{A}$
 -
 3: $\hat{w}$ in $\mathcal{A} \cup a_r \Leftarrow$ BE1($s$ in $\mathcal{B} \cup a_r, \hat{\hat{s}}$ in $\mathcal{A}$)
 -
 -
 -
 4: $w$ in $\mathcal{B} \Leftarrow$ BE2($\hat{w}$ in $\mathcal{A}, w$ in $a_r$)
```

Fig. 2. The state-of-the-art and the reorganized RNS Montgomery multiplication algorithms.

$x + \lambda A$ (or $x + \lambda B$, according to the base of origin). In order to perform the correction, Kawamura et al. [4] evaluate λ by checking the most significant bits of the results of the previous multiplication step. In [7], two BE techniques are used: in the first one, there is no correction; in the second one (first proposed in [8]), a redundant base element is used to evaluate λ and to correct the result.

In the reorganized algorithm, the precomputed values and the order of operations are modified as illustrated in Table 2 (without considering the contribution of error evaluation). The differences are

1. The precomputed values $-N^{-1}$ and B_i^{-1} in $\mathcal{B}$ are substituted by $-N^{-1}B_i^{-1}$; hence, the multiplication by $-N^{-1}$ in RNS MM is merged with the multiplication by B_i^{-1} in the subsequent BE, thus saving a multiplication step.
2. The precomputed values N and A_j are substituted by A_jN; the multiplication by N in RNS MM is merged with the summation of multiplications by A_j in the previous BE and another multiplication step is avoided.
3. The multiplication by B_A^{-1} is split in two parts. One part is merged with the summation of multiplications by A_jN in $\mathcal{A}$ in the first BE. The precomputed value A_jN is substituted by $A_jNB_A^{-1}$. Also, s in $\mathcal{A}$ is multiplied by B_A^{-1} in the first BE; thus, the number of multiplication steps remains unchanged.
4. The multiplication by A_j^{-1} is not required, since all the inputs in $\mathcal{A}$ are premultiplied by A_j^{-1}. However, a multiplication by A_j is required to reach the correct value. This correction can be merged to the multiplication by B_A^{-1} and no additional multiplication step is required. Hence, the precomputed value B_A^{-1} is substituted by $B_A^{-1}A_j$.

In the following, two versions of the reorganized algorithm will be analyzed, by making reference to the different BE approaches used by Kawamura et al. and Bajard et al. in [4] and [7], respectively. The algorithms are presented by including the redundant base element a_r, even though it is involved only in the latter approach.

By analyzing the RNS MM algorithms in Fig. 2, it can be observed that Algorithm 3, Step 1 corresponds to Step 1 of the reorganized algorithm (Algorithm 4). In Step 2 of Algorithm 4, the two inputs of the multiplication are already premultiplied by A_j^{-1}; hence, the result is equal to Step 2 in Algorithm 3, multiplied by A_j^{-2}. In Algorithm 4, Steps 3, 5, 6, and 7 of Algorithm 3 are moved into the first BE, together with the multiplication by A_j that is required to correct the input.

2.5 BE Approach by Kawamura et al.

The first BE proposed by Kawamura et al. in [4] (Algorithm 5) and KBE1, i.e., the reorganized BE based on the same approach (Algorithm 6) are shown in Fig. 3. KBE1 includes all the operations of Algorithm 5 as well as the operations of

TABLE 2
Operations in the State-of-the-Art and the Reorganized MM and BE Algorithms (without Error Evaluation)

	[4], [7]		Reorganized	
	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A}$	$\mathcal{B}$
MM	$s = x \cdot y$	$s = x \cdot y$ $u = s \cdot \left(-N^{-1}\right)$	$\hat{\hat{s}} = \hat{x} \cdot \hat{y}$	$s = x \cdot y$
BE1	$\tilde{u} = \sum_{i=1}^{k} q_{b_i} \cdot B_i$ $u = \tilde{u} - \lambda \cdot B$	$q = u \cdot B_i^{-1}$	$\hat{w} = \hat{\hat{s}} \cdot \left(B_A^{-1}A_j\right) + \sum_{i=1}^{k} q_i \cdot \left(B_i N B_A^{-1} A_j^{-1}\right)$ $\hat{w} = \hat{w} + \lambda \cdot \left(-NA_j^{-1}\right)$	$q = u \cdot \left(-N^{-1}B_i^{-1}\right)$
MM	$t = u \cdot N$ $v = s + t$ $w = v \cdot B_A^{-1}$			
BE2	$q = w \cdot A_j^{-1}$	$\tilde{w} = \sum_{j=1}^{k} \hat{w}_{a_j} \cdot A_j$ $w = \tilde{w} - \lambda \cdot A$		$\tilde{w} = \sum_{j=1}^{k} \hat{w}_{a_j} \cdot A_j$ $w = \tilde{w} - \lambda \cdot A$

† A_j in $\mathcal{A}$ and A_j^{-1} in $\mathcal{A}$ are used to indicate $\left(|A_0|_{a_0}, |A_1|_{a_1}, \ldots, |A_k|_{a_k}\right)$ and $\left(\left|A_0^{-1}\right|_{a_0}, \left|A_1^{-1}\right|_{a_1}, \ldots, \left|A_k^{-1}\right|_{a_k}\right)$, respectively.

```
Algorithm 5: First Base Extension proposed by Kawamura et al. [4]
Input: $u$ in $\mathcal{B}$, $\alpha = 0$
Output: $u$ in $\mathcal{A}$
Precomputation: $\left|B_i^{-1}\right|_{b_i}$ for $i = 1...k$,
    $B_i$ in $\mathcal{A}$ for $i = 1...k$, $-B$ in $\mathcal{A}$.
1: $q_i = \left|u_{b_i} \cdot B_i^{-1}\right|_{b_i}$ for $i = 1...k$
2: $c = \alpha$
3: $u_{a_j} = 0$ for $j = 1...k$
4: for $i = 1$ to $k$ do
5:     $c = c + \frac{\mathrm{trunc}(q_i)}{2^r}$
6:     $f^{\dagger} = \lfloor c \rfloor$
7:     $c = c - f$
8:     $u_{a_j} = \left|u_{a_j} + q_i \cdot B_i + f(-B)\right|_{a_j}$
       for $j = 1...k$
9: end for
$^{\dagger}$ $f \in \{0, 1\}$
```

```
Algorithm 6: First Reorganized Base Extension based on the algorithm by Kawamura et al. (KBE1)
Input: $s$ in $\mathcal{B}$, $\alpha = 0$, $\hat{\hat{s}}$ in $\mathcal{A}$
Output: $\hat{w}$ in $\mathcal{A}$
Precomputation: $\left|-N^{-1}B_i^{-1}\right|_{b_i}$ for $i = 1...k$, $B_i N B_A^{-1} A_j^{-1}$
    in $\mathcal{A}$ for $i = 1...k$, $-NA_j^{-1}$, $B_A^{-1}A_j$ in $\mathcal{A}$.
1: $q_i = \left|s_{b_i} \cdot \left(-N^{-1}B_i^{-1}\right)\right|_{b_i}$ for $i = 1...k$
2: $c = \alpha$
3: $\hat{w}_{a_j} = \left|\hat{\hat{s}}_{a_j} \cdot \left(B_A^{-1}A_j\right)\right|_{a_j}$ for $j = 1...k$
4: for $i = 1$ to $k$ do
5:     $c = c + \frac{\mathrm{trunc}(q_i)}{2^r}$
6:     $f^{\dagger} = \lfloor c \rfloor$
7:     $c = c - f$
8:     $\hat{w}_{a_j} = \left|\hat{w}_{a_j} + q_i \cdot \left(B_i N B_A^{-1} A_j^{-1}\right) + f\left(-NA_j^{-1}\right)\right|_{a_j}$
       for $j = 1...k$
9: end for
```

Fig. 3. The first BE presented in [4] and the reorganized BE (KBE1).

```
Algorithm 7: Second Base Extension proposed by Kawamura et al. [4]
Input: $w$ in $\mathcal{A}$, $\alpha = 0.5$
Output: $w$ in $\mathcal{B}$
Precomputation: $\left|A_j^{-1}\right|_{a_j}$ for $j = 1...k$,
    $A_j$ in $\mathcal{B}$ for $j = 1...k$, $-A$ in $\mathcal{B}$.
1: $\hat{w}_j = \left|w_{a_j} \cdot A_j^{-1}\right|_{a_j}$ for $i = 1...k$
2: $c = \alpha$
3: $w_{b_i} = 0$ for $i = 1...k$
4: for $j = 1$ to $k$ do
5:     $c = c + \frac{\mathrm{trunc}(\hat{w}_j)}{2^r}$
6:     $f^{\dagger} = \lfloor c \rfloor$
7:     $c = c - f$
8:     $w_{b_i} = \left|w_{b_i} + \hat{w}_j \cdot A_j + f(-A)\right|_{b_i}$ for $i = 1...k$
9: end for
$^{\dagger}$ $f \in \{0, 1\}$
```

```
Algorithm 8: Second reorganized Base Extension based on Kawamura et al. approach (KBE2)
Input: $\hat{w}$ in $\mathcal{A}$, $\alpha = 0.5$
Output: $w$ in $\mathcal{B}$
Precomputation: $A_j$ in $\mathcal{B}$ for $j = 1...k$,
    $-A$ in $\mathcal{B}$.
-
1: $c = \alpha$
2: $w_{b_i} = 0$ for $i = 1...k$
3: for $j = 1$ to $k$ do
4:     $c = c + \frac{\mathrm{trunc}(\hat{w}_j)}{2^r}$
5:     $f^{\dagger} = \lfloor c \rfloor$
6:     $c = c - f$
7:     $w_{b_i} = \left|w_{b_i} + \hat{w}_j \cdot A_j + f(-A)\right|_{b_i}$ for $j = 1...k$
8: end for
```

Fig. 4. The second BE presented in [4] and the reorganized BE (KBE2).

Algorithm 3, Steps 3, 5, 6, and 7. While the first BE proposed by Kawamura et al. only extends the value u from $\mathcal{B}$ to $\mathcal{A}$, KBE1 directly calculates $\hat{w}$ of Algorithm 4, Step 3 from s of Step 1 and $\hat{\hat{s}}$ of Step 2.

The result of the summation in the BE of x from $\mathcal{A}$ to $\mathcal{B}$ is $\tilde{x} = x + \lambda A$, where λ is an integer number with $0 \leq \lambda < k$. The algorithm proposed by Kawamura et al. calculates the approximate value of λ as $\tilde{\lambda} = \lfloor \alpha + \sum_0^k \mathrm{trunc}(q_i)/2^r \rfloor \simeq \lfloor \sum_0^k q_i/b_i \rfloor$; $\tilde{\lambda}$ is estimated accumulating the ϱth most significant bits of q_i, cut by trunc() and divided by 2^r, where $\mathrm{trunc}(q_i) = q_i \wedge (1_{(r)} \ldots 1_{(r-\varrho+1)} 0_{(r-\varrho)} \ldots 0_{(1)})_{(2)}$ and $\wedge$ denotes a bitwise AND operation. The variable α represents the starting value of the parameter used to calculate λ. In [4], two theorems prove that with proper values of α, with a low ϱ and by selecting the base elements so that 2^r is close to b_i, the approximation does not introduce errors. In the first BE, $\alpha = 0$ and the input is lower than B; therefore, according to Theorem 2 in [4], the result of the BE is $\tilde{x} < 2B$. In the second BE, $\alpha = 0.5$ and the input is lower than $2N$; hence, according to Theorem 1 in [4], the result of the BE is correct. With $r = 32$, $k = 33$, and $\max(2^r - a_i; 2^r - b_i) < 2^{16}$, $\forall i$, it is possible to choose $\varrho \geq 7$. The algorithm is the same for both the BEs, with exchanged bases. This approach requires that $A \geq 4N$ and $B \geq 5N$.

Algorithm 5 starts with the multiplication of each partial result u_{b_i} by the precomputed value B_i^{-1}, modulo b_i. In KBE1, the partial result used in the corresponding multiplication is s_{b_i} and the precomputed value is $-N^{-1}B_i^{-1}$. The result of KBE1 Step 1 is the same of Algorithm 5, Step 1, for the associative property. In Algorithm 5, Step 3 corresponds to an initialization. In KBE1, Step 3 executes in $\mathcal{A}$ the multiplication of $\hat{\hat{s}}$ by the precomputed value $B_A^{-1}A_j$, in order to reach a result that can be added to the summation by providing a value ready for the BE. In both the algorithms, Steps from 5 to 7 evaluate the contribution of each partial result q_i to λ. In Algorithm 5, Step 8 requires k^2 multiplications, corresponding to the summation of the results of the multiplication of each partial result q_i by the corresponding precomputed value $|B_i|_{a_j}$, per base element a_j, and to the sum of the precomputed value $-B$, when the floor f, used to evaluate λ, is equal to 1. In the corresponding step of KBE1, $B_i N B_A^{-1} A_j^{-1}$ and $-NA_j^{-1} \equiv -BNB_A^{-1}A_j^{-1}$ are used instead of B_i and $-B$, respectively. For the associative and distributive properties, KBE1 provides the same result of Algorithm 7, Step 1.

Fig. 4 shows the second BE algorithm proposed in [4] and the related reorganized algorithm. Algorithm 7 is the same as Algorithm 5, but with the bases switched. KBE2 is the same as Algorithm 7, but without the first step, since the input is already multiplied by A_j^{-1}.

```
Algorithm 9: First Base Extension employed by Bajard et al. [5], [7]
Input: $u$ in $\mathcal{B}$
Output: $u$ in $\mathcal{A} \cup a_r$
Precomputation: $\left|B_i^{-1}\right|_{b_i}$ for $i = 1...k$,
    $B_i$ in $\mathcal{A} \cup a_r$ for $i = 1...k$.

1: $q_i = \left|u_{b_i} \cdot B_i^{-1}\right|_{b_i}$ for $i = 1...k$
2: $u_{a_j} = \left|\sum_{i=1}^{k} q_i \cdot B_i\right|_{a_j}$
   for $j = 1...k$ and $j = r$
   -
```

```
Algorithm 10: First reorganized BE based on the algorithm by Bajard et al. (BBE1)
Input: $s$ in $\mathcal{B} \cup a_r$, $\hat{\hat{s}}$ in $\mathcal{A}$
Output: $\hat{w}$ in $\mathcal{A}$, $w$ in $a_r$
Precomputation: $\left|-N^{-1}B_i^{-1}\right|_{b_i}$ for $i = 1...k$, $B_i N B_A^{-1} A_j^{-1}$
    in $\mathcal{A}$ for $i = 1...k$, $B_A^{-1}A_j$ in $\mathcal{A}$, $\left|B_A^{-1}\right|_{a_r}$, $\left|B_i N B_A^{-1}\right|_{a_r}$
    for $i = 1...k$.
1: $q_i = \left|s_{b_i} \cdot (-N^{-1}B_i^{-1})\right|_{b_i}$ for $i = 1...k$
2: $\hat{w}_{a_j} = \left|\hat{\hat{s}}_{a_j} \cdot B_A^{-1}A_j + \sum_{i=1}^{k} q_i \cdot (B_i N B_A^{-1} A_j^{-1})\right|_{a_j}$
   for $j = 1...k$
3: $w_r = \left|s_r \cdot B_A^{-1} + \sum_{i=1}^{k} q_i \cdot (B_i N B_A^{-1})\right|_{a_r}$
```

Fig. 5. The first BE used in [7] and the reorganized BE (BBE1).

```
Algorithm 11: Second Base Extension employed by Bajard et al. [5], [7], [8]
Input: $w$ in $\mathcal{A} \cup a_r$
Output: $w$ in $\mathcal{B}$
Precomputation: $\left|A_j^{-1}\right|_{a_j}$ for $j = 1...k$, $\left|A_r^{-1}\right|_{a_r}$, $-A$ in $\mathcal{B}$,
    $A_j$ in $\mathcal{B}$ for $j = 1...k$, $|A_j|_{a_r}$ for $j = 1...k$.
1: $\hat{w}_j = \left|w_{a_j} \cdot A_j^{-1}\right|_{a_j}$ for $j = 1...k$
2: $w'_r = \left|\sum_{j=1}^{k} \hat{w}_j A_j\right|_{a_r}$
3: $\lambda = \left|(w'_r - w_r)A_r^{-1}\right|_{a_r}$
4: $w_{b_i} = \left|\sum_{j=1}^{k} \hat{w}_j \cdot A_j\right|_{b_i}$ for $i = 1...k$
5: $w_{b_i} = |w_{b_i} - A \cdot \lambda|_{b_i}$ for $i = 1...k$
```

```
Algorithm 12: Second reorganized BE based on the algorithm by Bajard et al. (BBE2)
Input: $\hat{w}$ in $\mathcal{A}$, $|w|_{a_r}$
Output: $w$ in $\mathcal{B}$
Precomputation: $\left|A_r^{-1}\right|_{a_r}$, $-A$ in $\mathcal{B}$,
    $A_j$ in $\mathcal{B}$ for $j = 1...k$, $|A_j|_{a_r}$ for $j = 1...k$.
    -
1: $w'_r = \left|\sum_{j=1}^{k} \hat{w}_j A_j\right|_{a_r}$
2: $\lambda = \left|(w'_r - w_r)A_r^{-1}\right|_{a_r}$
3: $w_{b_i} = \left|\sum_{j=1}^{k} \hat{w}_j \cdot A_j\right|_{b_i}$ for $i = 1...k$
4: $w_{b_i} = |w_{b_i} - A \cdot \lambda|_{b_i}$ for $i = 1...k$
```

Fig. 6. The second BE used in [7] and the reorganized BE (BBE2).

2.6 BE Approach by Bajard et al.

The MM algorithm proposed by Bajard et al. in [7] (and applied in the context of RNS ME in [5]) uses two BE algorithms; Algorithm 9 in Fig. 5 shows the first one, which does not perform the BE correction in order to save time. Algorithm 11 in Fig. 6, originally proposed by Shenoy and Kumaresan [8], calculates the correct result. The error in the result of the first BE does not affect the final result of the MM, but larger bases are required, i.e., $A, B > N(k+2)^2$.

Algorithm 10 in Fig. 5 corresponds to the first reorganized BE based on the approach by Bajard et al. (BBE1) and includes the operations of Algorithm 9 and the operations of Algorithm 3, Steps 3, 5, 6, and 7.

Algorithm 9 starts with the multiplication of each partial result u_{b_i} by the precomputed value B_i^{-1}, modulo b_i. In BBE1, the partial result used in the corresponding multiplication is s_{b_i} and the precomputed value is $-N^{-1}B_i^{-1}$. The result of BBE1 Step 1 is the same as the result of Algorithm 9, Step 1, for the associative property. In Algorithm 9, Step 2 requires k^2 multiplications, corresponding to the summation of the results of the multiplications of each partial result q_i by the corresponding precomputed value $|B_i|_{a_j}$, per base element a_j. In the corresponding step of BBE1, $B_i N B_A^{-1} A_j^{-1}$ is used instead of B_i. The result of the summation of the results of the multiplications of these precomputed values by the corresponding partial results are added to the multiplication of $\hat{\hat{s}}$ by the precomputed value $B_A^{-1}A_j$. BBE1 provides the same result of the first step of the second BE used by Bajard et al., for the associative and distributive properties.

The second BE used by Bajard et al. (Algorithm 11), and the corresponding second reorganized BE (BBE2) (Algorithm 12) are shown in Fig. 6. The value of λ is calculated by evaluating the difference between the correct result and the extended result on a redundant base element a_r, such that $\gcd(a_r, A) = 1$ and $\gcd(a_r, B) = 1$, according to [8]. Algorithm 11, Step 2 calculates the difference between the correct value of x in a_r and the result of the BE on a_r, which correspond to

$$\lambda = \frac{|x|_{a_r} - |x + \lambda A|_{a_r}}{A}. \tag{2}$$

Algorithm 12 is the same as Algorithm 11 without Step 1, since the input of the BE is already multiplied by A_j^{-1}.

2.7 Analysis

This section presents an analysis focused on MM, which corresponds to the most computationally intensive part in the overall ME algorithm.

2.7.1 Number of Modular Multiplications

In [4] and [5], the metric used to evaluate the performance of the proposed approach was based on the number of modular multiplications needed (as shown in Table 3). The reorganized RNS MM algorithms requires $3k$ modular multiplications less than the state-of-the-art algorithms with the same BE technique. The reorganized RNS ME requires $2k$ additional modular multiplications. Considering an RSA scenario with an exponent of size $g = 1{,}024$ and $k = 33$, the maximum number of modular multiplications would be reduced from 4,938,450 to 4,735,566 (95.8 percent). Considering a smaller base where $g = 640$ and $k = 21$, the maximum number of modular multiplications would be reduced from 1,319,178 to 1,238,454 (93.9 percent). The percentage reduction is slight better with a smaller base size, since the relative weight of $3k$ (i.e., the reduction) over k^2 (i.e., main contribution to the total cost) is greater.

TABLE 3
Number of Modular Multiplications Required by the Considered RNS MM Algorithms

	Kawamura et al. [4]	Bajard et al. [7]	Reorganized (with KBE1/KBE2)	Reorganized (with BBE1/BBE2)
Step 1, 3, and 4 of MM	$5k$	$5k$	$2k$	$2k$
First BE without correction	k^2+k	k^2+k	k^2+2k	k^2+2k
First BE correction	k	0	k	0
Second BE without correction	k^2+k	k^2+k	k^2	k^2
Second BE correction	k	k	k	k
Total without BE correction	$2k^2+7k$	$2k^2+7k$	$2k^2+4k$	$2k^2+4k$
Total	$2k^2+9k$	$2k^2+8k$	$2k^2+6k$	$2k^2+5k$

2.7.2 Characterization of the Modular Multiplications

Tables 4 and 5 provide a characterization of the multiplications involved in the state-of-the-art and the reorganized RNS MM and BE algorithms (without correction) in terms of multiplication steps.

Since each operation is performed at least on k base elements, up to k cells can work in parallel, requiring $2k+7$ multiplication steps for the multiplications shown in Table 4, and only $2k+4$ for Table 5.

The multiplication steps can be classified in the perspective of pipelining and parallelization possibilities, in order to calculate the number of cycles required by each step. Multiplication step types are

- *Type 1*, multiplication steps that cannot be parallelized or executed in pipeline, since they use as an input the result of the previous operation, and their output is used as input by the subsequent one; the number of required cycles is p, where p corresponds to the number of stages of the pipeline.
- *Type 2*, a group of multiplication steps that can be executed in parallel or pipelined; they require p cycles for the last multiplication step and one cycle for the others; by considering M as the number of parallel multipliers per cell, a group of x multiplication steps requires $\lfloor \frac{x}{M} \rfloor + p - 1$ cycles.
- *Type 3*, multiplication steps that can be executed in parallel to others; they require zero cycles in parallel or pipelined architectures, 1 cycle otherwise ($\lfloor \frac{1}{p+M-1} \rfloor$).

Without considering the BE correction, the state-of-the-art RNS MM involves: six Type 1 multiplication steps, two groups of k Type 2 multiplication steps, and one Type 3 multiplication step (ID 2). Therefore, the total number of cycles per RNS MM is $2\lceil \frac{k}{M} \rceil - 2 + \lfloor \frac{1}{p+M-1} \rfloor + 8p$.

Comparing Table 5 to Table 4, it can be observed that the reorganized algorithm requires $4p$ cycles less than the state-of-the-art one, but it needs $\lfloor \frac{1}{p+M-1} \rfloor$ additional cycles.

Therefore, the improvement is directly linked to the number of pipeline stages and of parallel multipliers. As a matter of example, by considering $p=3$, $M=1$, and $k=33$ as in [4] (without BE correction), the reorganized RNS MM algorithm obtains a delay reduction of 13.63 percent. With a higher degree of pipelining, a larger reduction is achieved (e.g., 16.66 percent with $p=4$ and $M=1$, 19.23 percent with $p=5$ and $M=1$, etc.).

The reorganized RNS ME algorithm requires $2p$ additional multiplication steps (Algorithm 2, Steps 1 and 11). By considering an RSA scenario, with $g=1{,}024$, $p=3$, $M=1$, and $k=33$, the maximum number of cycles would be reduced from 180,400 to 155,806 (86.3 percent). Considering a smaller base where $g=640$ and $k=21$, the maximum number of cycles would be reduced from 82,048 to 66,670 (81.2 percent). The percentage reduction is better with a smaller base size, since the relative weight of $4p$ (i.e., the reduction) over k (i.e., main contribution to the total cost) is greater.

2.8 Extension to Other Cryptosystems and Remarks

The reorganization of RNS ME provides greater benefits when the RNS bases are small. However, in cryptography, the exponentiation is typically used for RSA, which requires very large bases. The reorganization of the operations internal to the RNS Montgomery reduction has been also exploited in cryptosystems that require single executions or sequences of Montgomery reductions, e.g., in elliptic curve cryptography [11], [12]. The RNS Montgomery reduction has been recently exploited also in the context of pairing [16], [17]. In these domains, the internal reorganization can produce the same reduction in the number of cycles, which, nonetheless, have a higher relative weight (as described in Sections 2.7.1 and 2.7.2). Moreover, although the new representation decreases the duration of each modular reduction, it introduces a fixed delay proportional to the number of input and output values. Furthermore, this representation requires that the input of the RNS Montgomery reduction is double hat (i.e., the

TABLE 4
Multiplication Steps of the State-of-the-Art MM

Multiplication step ID	Operation	Number of multiplications	Type
1	$s = xy$ in $\mathcal{B}$	k	1
2	$s = xy$ in $\mathcal{A}$	k	3
3	$u = s(-N^{-1})$	k	1
4	$q = uB_i^{-1}$	k	1
5… k+4	$u = q_i B_i$	k^2	2
k+5	$t = uN$	k	1
k+6	$w = vB_A^{-1}$	k	1
k+7	$\hat{w} = wA_j^{-1}$	k	1
k+8… 2k+7	$w = \hat{w}_j A_j$	k^2	2

TABLE 5
Multiplication Steps of the Reorganized MM

Multiplication step ID	Operation	Number of multiplications	Type
1	$s = xy$ in $\mathcal{B}$	k	1
2	$\hat{\hat{s}} = \hat{x}\hat{y}$ in $\mathcal{A}$	k	3
3	$q = s(-N^{-1}B_i^{-1})$	k	1
4	$\hat{w} = \hat{s}B_A^{-1}A_j$	k	3
5… k+4	$\hat{w}_i = B_i N B_A^{-1} A_j^{-1}$	k^2	2
k+5… 2k+4	$w = \hat{w}_j A_j$	k^2	2

product of two *hat* values). A different input would require a different Montgomery reduction algorithm or an additional multiplication step to adapt it. Based on such considerations, the application of reorganized algorithms to other cryptosystems seems appealing, though an extensive analysis would be required to precisely characterize advantages and drawbacks.

3 Architectural Study

In [6], Nozaki et al. proposed a cell architecture suitable to the algorithm presented in [4]. The cell is matched to a single element per RNS base; hence, a set of k cells is required to execute the exponentiation. A cell can be matched to more than one element per base, in order to reduce the area (though increasing the delay). The cell is composed by a three-stage pipelined Modular Multiplier and Accumulator Unit (MMAU), a Cox unit (that evaluates and corrects the BE error) and some memory elements. The MMAU used by Nozaki et al. is also suitable for the approach proposed by Bajard et al.; nevertheless, in this case, an additional redundant cell matched to the redundant base element is used, instead of the Cox unit.

The architectural study starts with the analysis of the cell architecture proposed in [6] (then extended to the approach proposed by Bajard and Imbert). Then, the possible base element types are analyzed, and five new cell architectures are presented. The study moves from an extensive theoretical investigation based on equivalent gates delay and area cost reported in [18], and produces an exhaustive analysis based on synthesis results.

3.1 Base Elements

RNS offers the opportunity to select the base elements, which strongly affect the computational effort required by the reduction. In [19], the authors showed that with a modulus $a_i = 2^r - 2^h - 1$ and $h < \frac{r+1}{2}$, the modular reduction of $|x|_{a_i}$, with $x < a_i^2$, can be reduced to

$$x \equiv x_1 + x_2 + x_4 + 2^h(x_3 + x_4) \pmod{a_i}, \qquad (3)$$

where $x_1 = x \bmod 2^r$, $x_2 = \lfloor x \div 2^r \rfloor$, $x_3 = x_2 \bmod 2^{r-h}$, and $x_4 = \lfloor x_2 \div 2^{r-h} \rfloor$.

This formula provides efficient base elements, which nonetheless must be relatively prime and must satisfy the algorithm size limitations. Formula $a_i = 2^r - 2^h \pm 1$ requires the same computational effort as $a_i = 2^r - 2_h - 1$, but it can provide more base elements.

A larger number of base elements can be reached by $a_i = 2^r - c_i$, with $c_i < 2^h$ and $h < \frac{r-1}{2}$. In [13], the authors showed that the modular reduction of $x < 2^{2r}$ requires two multiplications and three additions. The partial modular reduction $y \equiv x (\bmod\ a_i)$ can be calculated by

$$y = |x|_{2^r} + (x \gg r) \cdot c_i. \qquad (4)$$

With $x < 2^z$, $z > r$, and $c_i < 2^h$, it is $y < \max(2^{r+1}, 2^{z-r+h+1})$. Therefore, each iteration of this method can reach a reduction of $r - h - 1$ bits. In order to reach a larger reduction per step with input $x > 2^{2r}$, it is possible to calculate

$$y = |x|_{2^r} + |x \gg r|_{2^r} \cdot c_i + (x \gg 2r) \cdot c_i^2 \ . \qquad (5)$$

With $x < 2^z$, $z > 2r$, and $c_i < 2^h$, it is

$$y < \max(2^{r+h+2}, 2^{z-2r+2h+1}).$$

Therefore, each iteration can reach a reduction of $2r - 2h - 1$ bits. This approach, which is also used in the architecture proposed in [6], is more computationally expensive than the previous one, but provides a larger number of possible base elements.

An intermediate approach requires base elements compliant with $a_i = 2^r - c_i$, with $c_i < 2^h$, $h < \frac{r+1}{2}$, and the Hamming weight of c_i, denoted as $\omega(c_i)$, selected so that $\omega(c_i) < t$. The limit to the Hamming weight allows to perform the modular reduction as a sequence of additions. According to [14], a modular reduction of $x < 2^{2r}$ requires $2\omega(c_i) + 2$ additions.

3.2 Three-Stage MMAU Architecture

In this section, the MMAU architecture proposed in [6] is analyzed. The MMAU corresponds to a cell without the error correction, as shown in Fig. 7. This cell is characterized by three-stage pipelining ($p = 3$) and includes one multiplier ($M = 1$). The MMAU is divided into the units listed below:

- *Multiplier Adder Unit (MAU)*, which performs unsigned multiplications and additions. It provides an output on $2r + \log_2 k$ bits.
- *First Modular Reduction Unit (FMRU)*, which performs the partial modular reduction (5). It provides an output on $r + h + 1$ bits.
- *Second Modular Reduction Unit (SMRU)*, which calculates the final result of the modular reduction performing (4) and an addition. The final reduction requires to check the $(r + 1)$th least significant bit. When it is equal to "1," the value is greater than a_i; hence, $-a_i$ is added in order to reach the right modulo. The output is represented on r bits. Although the final result is smaller than 2^r, it can still be larger than a_i; nonetheless, according to [6], it does not affect the overall results.

The Type 1 multiplication steps are executed sequentially by each unit, requiring three cycles. The Type 3 multiplication steps are executed in parallel to the Type 1 ones, exploiting the unused units. Therefore, they do not require additional cycles. The groups of Type 2 multiplication steps are executed by the MAU, one multiplication and accumulation per cycle. After the last multiplication step, two cycles are required to reduce the result.

3.3 Proposed Four-Stage MMAU Architecture

An efficient cell can be obtained by introducing redundancy, increasing the level of pipeline, and moving the accumulation from the MAU to the FMRU. By using a carry-save representation, the final adder of the MAU and of the FMRU can be removed. Based on theoretical observations, the SMRU can be split in two units, thus augmenting the degree of pipelining. The area and the delay of the four-stage architecture can be reduced by moving the addition operation in the FMRU, thus limiting the number of input lines of the MAU and of the FMRU. Fig. 7 shows the corresponding architecture, which is divided into

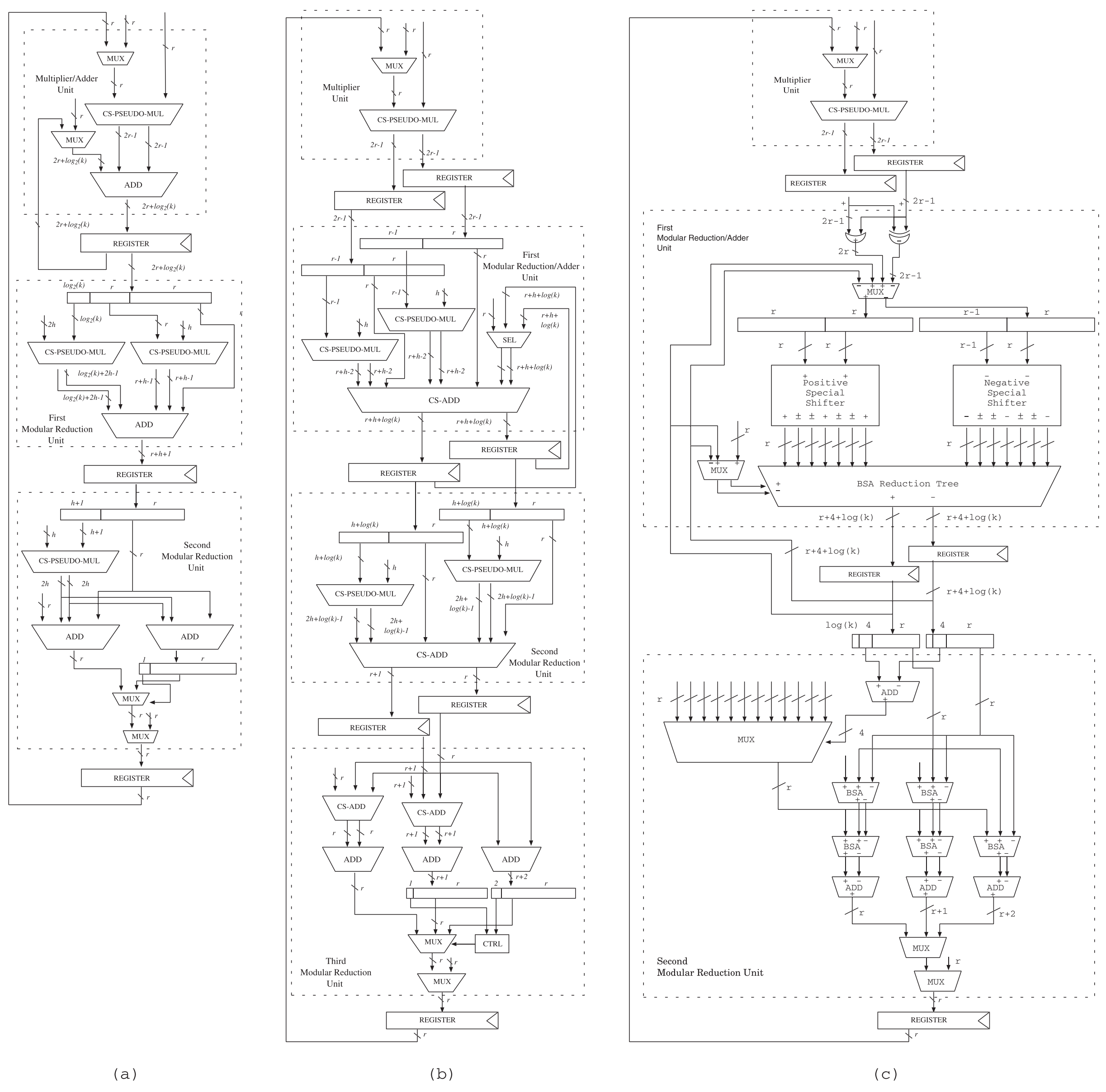

Fig. 7. (a) Three-stage, (b) Four-stage, and (c) $\omega(c_i) \leq 3$ MMAU architecture.

- The Multiplier Unit (MU), which performs unsigned multiplications. It provides a carry-save output on $2r - 1$ bits.
- The First Modular Reduction and Adder Unit (FMRAU), which executes (4) and can accumulate the results or add other values; (5) is not applicable, since the input is too short. It provides a carry-save output on $r + h + \log_2 k$ bits.
- The SMRU, which computes (4). It provides a carry-save output on $r + 1$ bits.
- The TMRU, which performs the final addition and reduction. It provides an output on r bits.

The area and the delay of the FMRAU are larger than in the previous cell architecture, since the result of the MU is redundant. However, the reduced number of bits attenuates the area increase. In order to execute the final reduction, three possible results are simultaneously calculated. One is the partial result of the reduction, one is added to $-a_i$, and the last is added to $-2a_i$. The most significant bits of the partial results are checked, in order to select the correct one.

In this MMAU, the Type 1 multiplication steps are executed sequentially by each unit, requiring four cycles. The Type 3 multiplication steps are executed in parallel to the Type 1 multiplications, without requiring additional cycles. The groups of Type 2 multiplication steps are executed by the MU and accumulated by the FMRAU, one per cycle. Lastly, three further cycles are required to complete the groups of Type 2 multiplication steps.

The four-stage MMAU architecture is characterized by a delay that is smaller than with the three-stage one, but requires a larger area.

3.4 Proposed $\omega(c_i) \leq 3$ MMAU Architecture

With a base compliant with $\omega(c_i) \leq 3$, it is possible to substitute the multiplications used for the reduction with additions. However, this strategy requires the management of negative numbers. In order to avoid additional delays, in the reduction units, the carry-save representation is replaced with the borrow-save one, where the values are expressed by two binary unsigned numbers (and the standard representation can be obtained by subtracting the second from the first). The $\omega(c_i) \leq 3$ MMAU architecture is divided into

- The MU, which performs unsigned multiplications. It provides a carry-save output on $2r - 1$ bits.
- The FMRAU, which can transform a carry-save input in borrow-save representation, convert each $2r$-bit input in $7r$-bit numbers, and accumulate the results or add other values. It provides a borrow-save output on $r + 4 + \log_2 k$ bits.
- The SMRU, which performs the final reduction. It produces a result on r bits.

The MMAU architecture is shown in Fig. 7. The two r-bit numbers that represent the inputs of the FMRAU are transformed in borrow-save representation and, then, in fourteen r-bit numbers by two special shifters. By considering a design without a knowledge of base elements (i.e., each cell can work with any base element respecting the constraints), the special shifter consists in some wired links and in six barrel shifters (which could be also substituted by multiplexers). The negative special shifter is identical to the positive one, but the sign of the numbers is switched. After the special shifters, a borrow-save adder reduction tree is used to reduce the number of partial results and to accumulate the results or add other values. The result is normally represented in borrow-save representation on $r + 4$ bits. However, at the end of the accumulation, it requires $r + \log_2 k + 4$ bits. The SMRU is only able to reduce numbers on $r + 4$ bits; thus, when the result is larger, it is used again as input in the FMRU, in order to reach the required length. The SMRU evaluates 4 bits from r to $r + 3$ of the two outputs of FMRAU, and adds/subtracts a proportional multiple of the base element. The partial result is equivalent to that of the four-stage architecture; hence, the same final reduction strategy is required.

This MMAU architecture can also be used when there is a knowledge of base elements; in this case, the barrel shifters are substituted by 2-input multiplexers. According to the delay reduction in FMRAU, the addition used to evaluate the 4 bits in the SMRU can be directly executed in the FMRAU.

3.5 Proposed $\omega(c_i) \leq 2$ MMAU Architecture

With a base compliant with $\omega(c_i) \leq 2$, it is possible to reduce the area and the delay of the modular reduction units. As with $\omega(c_i) \leq 3$, negative numbers have to be managed. The MMAU architecture is divided into the three units below:

- The MU, which performs unsigned multiplications. It provides a carry-save output on $2r - 1$ bits.
- The FMRU, which can transform a carry-save input in borrow-save representation, convert each $2r$-bit input in $4r$-bit numbers, and accumulate the results or add other values. It provides a borrow-save output on $r + 2 + \log_2 k$ bits.
- The SMRU, which performs the final reduction. It produces a result on r bits.

The $\omega(c_i) \leq 2$ MMAU architecture is close to the $\omega(c_i) \leq 3$ one, but smaller and faster. Only 3 bits of the result of the borrow-save reduction tree are used to select the value to add in the SMRU. This control is directly executed by the FMRAU, in order to reduce the delay of the SMRU. When there is no knowledge of base elements, two barrel shifters are required. Otherwise, it is possible to substitute them with 2-input multiplexers.

3.6 Higher Level of Pipelining

The proposed cell architectures are composed by macro-pipeline stages. Each macrostage has a clear mathematical function, and the stages of each cell architecture have been balanced by considering the equivalent gates delay. Therefore, they represent a technology independent and functionally atomic solution. It could be possible to further improve the time performance with a higher pipelining. In the literature, there are examples of RNS arithmetic cells designed for FPGA with higher level of pipeline [11], [12], [16], [17]. However, a higher pipelining would increase the area. Moreover, the identification of the best number of pipeline stages is an optimization problem that is out of the topic of this work, and its result would be strongly dependent on technological choices.

In order to provide the reader with a rough idea about the effects of a higher pipelined cell, an experimental analysis has been carried out, by splitting each macrostage of the Four-stage MMAU Architecture in two substages. The corresponding eight-stage MMAU would give a reduction in delay equal to 10.3 percent and an increase of 16.6 percent in area with respect to the original one.

3.7 BE Correction (Approach by Kawamura et al.)

The approach proposed in [4] requires a Cox unit, in order to perform the modular reduction. The Cox unit is composed by an adder, a register, and a set of AND gates. Fig. 8 shows the Cox unit connected to the MAU of the three-stage MMAU on the left, and to the FMRU of the fours-stage MMAU on the right. In the $\omega(c_i) \leq 3$ and $\omega(c_i) \leq 2$ MMAU architectures, it can be connected to the FMRU reduction tree. The size of the adder (ϱ) can be set to nine, with $r = 32$, $h = 11$, and $k = 33$.

3.8 Error Correction (Approach by Bajard et al.)

The approach used in [5] requires a redundant cell matched to a_r. This cell calculates λ, which is multiplied by A on the other cells.

Fig. 9 shows the redundant cell, which is composed by a Multiplier Unit and an Adder Unit (AU). Since this cell works on a number of bits smaller than the other cells, two multiplications can be processed in parallel and added in one step. Moreover, as proposed in [5], a_r can be a power of two; hence, the modular reduction is immediate. The area overhead is similar to the one proposed in [4]. The BE correction requires an additional step. However, as suggested in [5], it is possible to avoid a multiplication by using tables, but the result should be summed by an additional input line.

3.9 Analysis of Cell Architectures

All the described cell architectures have been synthesized using the Nangate 45 nm Open Cell Library with Synopsys Design Compiler. Table 6 shows the delay and area cost of the cells by considering $r = 32$, $k = 33$, and $h = 11$, as in [5] and [6]. The delay refers to the combinatory net of each unit involved in the considered cell architectures. The area of each cell includes the MMAU, its registers, and the RAM. The area is obtained by considering the RAM required for the state-of-the-art algorithms; when the reorganized

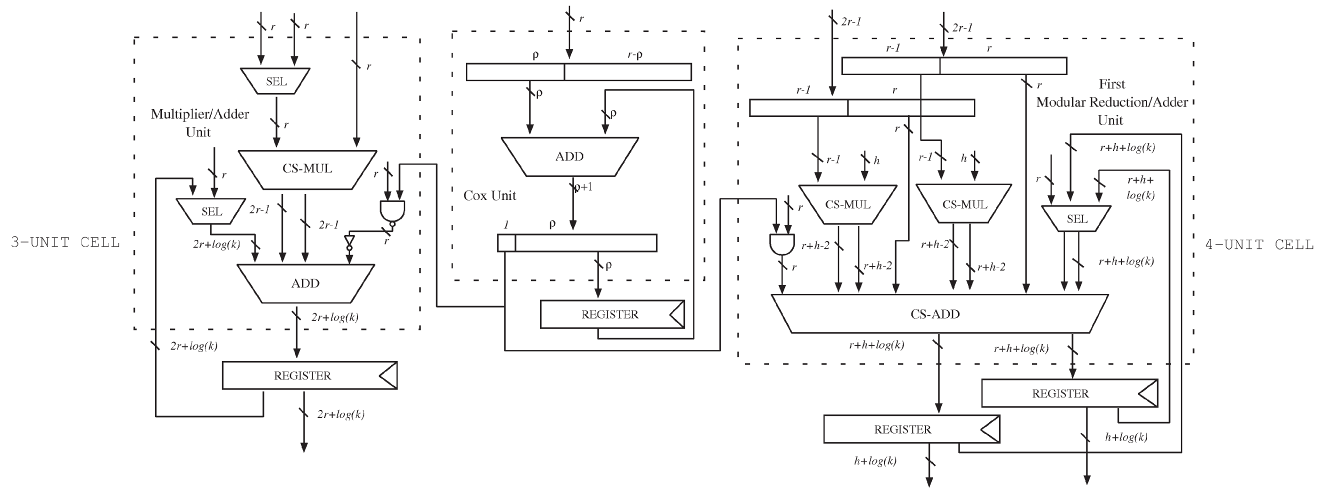

Fig. 8. Cox unit.

algorithms are used, $230\,\mu\mathrm{m}^2$ can be saved. All the precomputed values are considered as updatable. All the cells that use the Kawamura et al. BE approach include an additional input for the correction. However, only the three-stage cells directly include the Cox unit, while for the other architectures the Cox unit is a separate cell. In the latter case, the area of the Cox unit is $168\ \mu\mathrm{m}^2$ and its delay is 0.18 ns. The a_r cell is always separate. Its area is $17{,}043\ \mu\mathrm{m}^2$, while its delay is 0.49 ns and 0.39 ns for the MU and the AU, respectively.

In the three-stage cell, the MAU is the slowest unit. The version with the Kawamura et al. BE approach is slower, since the output of the Cox cell is calculated and added in the MAU during the same cycle. In the other cell architectures, the Cox output is added in the FMRAU; hence, it is calculated by the Cox unit during a cycle, and added during the subsequent one. In this way, the effects of the Cox unit on the delay are reduced. The architectural improvements applied in the four-stage architecture reduce the delay of all the units, but increase the area. The FMRAU is the slowest unit of the $\omega(c_i) \leq 3$ MMAU without knowledge of the base elements. This is mainly due to the presence of the barrel shifters, which also contribute to the large area. In the $\omega(c_i) \leq 3$ MMAU with knowledge of base elements, the slowest unit is the MU, as in the four-stage MMAU. This is due to the removal of the barrel shifters from the FMRAU. Even in the $\omega(c_i) \leq 2$ MMAU, the slowest unit is the MU, thanks to the smaller borrow-save adder reduction tree.

4 Overall Comparison

The comparison of all the algorithms and cell architectures is summarized in Table 7. The area cost corresponds to the k cells and the BE correction cell. The configuration used is $k = 33$, $r = 32$, $h = 11$, and $M = 1$.

The reorganized algorithm provides a reduction in the number of cycles, and a small reduction in the RAM size, thanks to the lower number of precomputed values. When compared to the state-of-the-art algorithm, with the Kawamura et al. BE and the three-stage cell, it provides a 13.6 percent reduction in delay, and a 0.6 percent in area.

The impact of the BE technique depends on the architecture considered. The technique proposed by Bajard et al. requires always one cycle more than the one by Kawamura et al.; however, the Kawamura et al. approach often increases the delay, especially in the three-stage cell. With the exception of the three-stage cell, where the Bajard et al. approach is faster, the delay is always similar. Even the area is not strongly affected by the BE approach.

The four-stage cell provides a reduction in delay, but involves a moderate increase in area. With the state-of-the-art algorithm and the Kawamura et al. BE, it reaches a 29.3 percent reduction in delay, and a 18.3 percent increase in area. The eight-stage version of this cell reaches a 36.5 percent reduction in delay, and a 37.9 percent increase in area.

The $\omega(c_i) \leq 3$ cell without knowledge of base elements performs worse than the four-stage cell both with respect to delay and area cost, whereas with knowledge of base elements, it achieves better results. Therefore, this approach can be effectively applied only if each cell is specifically designed for two specific base elements. With respect to the three-stage cell, the area is slightly larger.

The $\omega(c_i) \leq 2$ cell without knowledge of base elements provides area and delay performance close to those of the $\omega(c_i) \leq 3$ cell with knowledge of base elements. With knowledge of base elements, the area is still reduced, and it is even smaller than with the three-stage cell. However, the constraint on the base size ($k \leq 10$) limits the width of the representable numbers to 319 bits. Therefore, this approach cannot be applied to a 1,024-bit modulus.

Based on the above analysis, the following observations can be made. The reorganized algorithm is more efficient than the state-of-the-art one. By using efficient cells, the

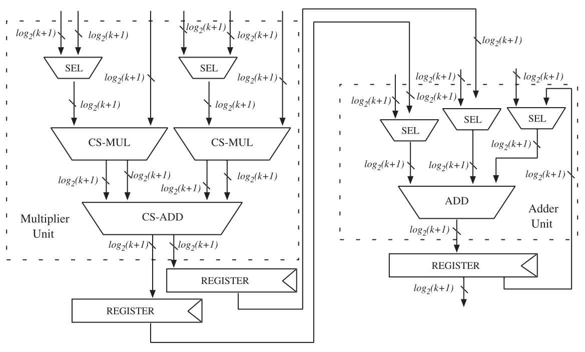

Fig. 9. Redundant cell matched to a_r.

TABLE 6
Area and Delay of the Units in the Designed Cells with $k = 33$, $r = 32$, and $h = 11$

Architecture	BE	Unit	Area (μm^2)		Delay
			Unit	Total‡	(ns)
Three-stage	[4]	MAU	8381†	33142†	**1.51**
		FMRU	4477		1.15
		SMRU	2543		1.15
	[7]	MAU	8004	32535	**1.42**
		FMRU	4477		1.15
		SMRU	2543		1.15
Four-stage	[4]	MU	7285	39372	**0.90**
		FMRAU	8124		0.86
		SMRU	4121		0.88
		TMRU	2101		0.76
	[7]	MU	7285	38398	**0.90**
		FMRAU	7380		0.88
		SMRU	4121		0.88
		TMRU	2101		0.76
$\omega(c_i) \leq 3$ without base knowledge	[4]	MU	7285	40616	0.90
		FMRAU	12002		**1.06**
		SMRU	3588		1.05
	[7]	MAU	7285	39755	0.90
		FMRAU	11371		**1.08**
		SMRU	3588		1.05
$\omega(c_i) \leq 3$ with base knowledge	[4]	MU	7285	34885	**0.90**
		FMRAU	6573		0.85
		SMRU	3286		0.87
	[7]	MAU	7285	34963	**0.90**
		FMRAU	6881		0.89
		SMRU	3286		0.87
$\omega(c_i) \leq 2$ without base knowledge	[4]	MU	7285	35037	**0.90**
		FMRAU	6510		0.83
		SMRU	3501		0.87
	[7]	MAU	7285	34395	**0.90**
		FMRAU	6098		0.87
		SMRU	3501		0.87
$\omega(c_i) \leq 2$ with base knowledge	[4]	MU	7285	33050	**0.90**
		FMRAU	4523		0.75
		SMRU	3501		0.87
	[7]	MAU	7285	32071	**0.90**
		FMRAU	3774		0.72
		SMRU	3501		0.87

†The Cox cell ($163\ \mu m^2$) is included. ‡Including $17{,}741\ \mu m^2$ and $17{,}511\ \mu m^2$ for the memory required by the state-of-the-art approaches, with [4] and [5] BEs, respectively. The reorganized approach saves $230\ \mu m^2$.

BE approach does not strongly affect performance. For inputs larger than 1,343 or, when the cells are designed independently of the base, larger than 319 bits, the four-stage cell provides the best results. When numbers between 1,343 and 320 bits have to be managed and the knowledge of base elements is available, the $\omega(c_i) \leq 3$ architecture should be considered. When numbers to be represented require less than 320 bits, the best cells are the $\omega(c_i) \leq 2$ ones.

5 Conclusion

In this paper, an algorithmic and architectural study on RNS ME has been presented. The opportunities that come from the reorganization of the operations and of precomputations have been investigated. In the architectural study, five new cell architectures have been presented, by exploiting well-known design techniques emerging from computer arithmetic and by considering efficient RNS bases. All the algorithms and the cell architectures have been compared. Results have shown that, by rearranging operations and preprocessing, the delay can be significantly reduced. Moreover, the analysis carried out on the cells for efficient bases has shown that their performance are particularly interesting only when their constraints are strict, hence limiting their applicability. The overall analysis should provide the reader with a comprehensive overview of algorithmic and architectural approaches that could be considered for performing RNE ME, by suggesting solutions to be possibly considered depending on the particular constraints set by the application scenario.

Acknowledgments

The authors wish to thank the anonymous reviewers for providing insightful feedback and giving valuable suggestions, which definitely helped to improve the overall quality of this work.

TABLE 7
Area and Delay of the Cells, with $k = 33$, $r = 32$, $h = 11$, and $M = 1$

Algorithm	BE	Architecture	Base knowledge	Max k	Cycles	Cycle delay	MM delay (ns)	Max ME delay (ns)	Cells area (μm^2)
[4], [7]	[4]	Three-stage	No	> 42	88	1.73	153	312103(100.0%)	1099266(100.0%)
Reorganized	[4]	Three-stage	No	> 42	76	1.73	132	269545(86.4%)	1091676(99.4%)
[4], [7]	[7]	Three-stage	No	> 42	89	1.64	146	299228(95.9%)	1113303(101.3%)
Reorganized	[7]	Three-stage	No	> 42	77	1.64	127	258884(83.0%)	1105713(100.6%)
[4], [7]	[4]	Four-stage	No	> 42	96	1.12	108	220425(70.7%)	1299444(118.3%)
Reorganized	[4]	Four-stage	No	> 42	80	1.12	90	183689(58.9%)	1291854(117.6%)
[4], [7]	[7]	Four-stage	No	> 42	97	1.12	109	222721(71.4%)	1284177(116.9%)
Reorganized	[7]	Four-stage	No	> 42	81	1.12	91	185985(59.6%)	1276587(116.2%)
[4], [7]	[4]	$\omega(c_i) \leq 3$	No	42	90	1.28	116	236160(75.6%)	1340496(122.0%)
Reorganized	[4]	$\omega(c_i) \leq 3$	No	42	78	1.28	100	204680(65.6%)	1332906(121.3%)
[4], [7]	[7]	$\omega(c_i) \leq 3$	No	42	91	1.30	119	242515(77.8%)	1328958(120.9%)
Reorganized	[7]	$\omega(c_i) \leq 3$	No	42	79	1.30	103	210543(67.5%)	1321368(120.3%)
[4], [7]	[4]	$\omega(c_i) \leq 3$	Yes	42	90	1.12	101	206640(66.3%)	1151373(104.8%)
Reorganized	[4]	$\omega(c_i) \leq 3$	Yes	42	78	1.12	88	179095(57.4%)	1143783(104.1%)
[4], [7]	[7]	$\omega(c_i) \leq 3$	Yes	42	91	1.12	102	208936(67.0%)	1170822(106.6%)
Reorganized	[7]	$\omega(c_i) \leq 3$	Yes	42	79	1.12	89	181391(58.2%)	1163232(105.9%)
[4], [7]	[4]	$\omega(c_i) \leq 2$	No	20†	90	1.12	101	206640(66.3%)	1156389(105.2%)
Reorganized	[4]	$\omega(c_i) \leq 2$	No	20†	78	1.12	88	179095(57.4%)	1148799(104.6%)
[4], [7]	[7]	$\omega(c_i) \leq 2$	No	20†	91	1.12	102	208936(67.0%)	1152078(104.9%)
Reorganized	[7]	$\omega(c_i) \leq 2$	No	20†	79	1.12	89	181391(58.1%)	1144488(104.2%)
[4], [7]	[4]	$\omega(c_i) \leq 2$	Yes	20†	90	1.12	101	206640(66.3%)	1090818(99.3%)
Reorganized	[4]	$\omega(c_i) \leq 2$	Yes	20†	78	1.12	88	179095(57.4%)	1083228(98.5%)
[4], [7]	[7]	$\omega(c_i) \leq 2$	Yes	20†	91	1.12	102	208936(67.0%)	1075386(97.9%)
Reorganized	[7]	$\omega(c_i) \leq 2$	Yes	20†	79	1.12	89	181391(58.2%)	1067796(97.2%)

† $\omega(ci) \leq 2$ is not compliant with $k = 33$, but the results are anyway presented with $k = 33$ in order to ensure a fair comparison.

REFERENCES

[1] N. Szabo and R. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology.* McGraw-Hill, 1967.

[2] P. Montgomery, "Modular Multiplication without Trial Division," *Math. of Computation,* vol. 44, no. 170, pp. 519-521, 1985.

[3] K. Posch and R. Posch, "Modulo Reduction in Residue Number Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 5, pp. 449-454, May 1995.

[4] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower Architecture for Fast Parallel Montgomery Multiplication," *Proc. Int'l Conf. Theory and Application of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT '00),* pp. 523-538, 2000.

[5] J.-C. Bajard and L. Imbert, "A Full RNS Implementation of RSA," *IEEE Trans. Computers,* vol. 53, no. 6, pp. 769-774, June 2004.

[6] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of RSA Algorithm Based on RNS Montgomery Multiplication," *Proc. Third Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01),* pp. 364-376, 2001.

[7] J.-C. Bajard, L.S. Didier, and P. Kornerup, "Modular Multiplication and Base Extensions in Residue Number Systems," *Proc. 15th IEEE Symp. Computer Arithmetic,* pp. 59-65, 2001.

[8] A. Shenoy and R. Kumaresan, "Fast Base Extension Using a Redundant Modulus in RNS," *IEEE Trans. Computers,* vol. 38, no. 2, pp. 292-297, Feb. 1989.

[9] F. Gandino, F. Lamberti, J.-C. Bajard, and P. Montuschi, "A General Approach for Improving RNS Montgomery Exponentiation Using Pre-Processing," *ARITH '11: Proc. 20th IEEE Symp. Computer Arithmetic,* July 2011.

[10] F. Gandino, F. Lamberti, J.-C. Bajard, and P. Montuschi, "Pre-Processing in RNS Montgomery Multiplication," technical report, 2010.

[11] N. Guillermin, "A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over F_p," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '10),* pp. 48-64, 2010.

[12] N. Guillermin, "A Coprocessor for Secure and High Speed Modular Arithmetic," Report 2011/354, Cryptology ePrint Archive, 2011.

[13] J. Bajard, N. Meloni, and T. Plantard, "Efficient RNS Bases for Cryptography," *IMACS '05: Proc. World Congress Scientific Computation, Applied Math. and Simulation,* July 2005.

[14] J.C. Bajard, M. Kaihara, and T. Plantard, "Selected RNS Bases for Modular Multiplication," *ARITH '09: Proc. 19th IEEE Symp. Computer Arithmetic,* pp. 25-32, 2009.

[15] *Algorithmic Algebraic Number Theory,* M. Pohst and H. Zassenhaus, eds., ch. 2.2.5. Cambridge Univ. Press, 1989.

[16] S. Duquesne and N. Guillermin, "A FPGA Pairing Implementation Using the Residue Number System," *Cryptology ePrint Archive,* Report 2011/176, 2011.

[17] R. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G. Yao, "FPGA Implementation of Pairings Using Residue Number System and Lazy Reduction," *Proc. Int'l Conf. Cryptographic Hardware and Embedded Systems (CHES '11),* B. Preneel and T. Takagi, eds., pp. 421-441, 2011.

[18] F. Gandino, F. Lamberti, G. Paravati, J.-C. Bajard, and P. Montuschi, "Investigation on Cell Architectures for RNS Montgomery Exponentiation," technical report, 2011.

[19] H. Wu, "On Modular Reduction," technical report, CACR, Univ. of Waterloo, 2000.

Filippo Gandino received the MS and PhD degrees in computer engineering from Politecnico di Torino, Italy, in 2005 and 2010, respectively. He is currently a research fellow with the Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interests include ubiquitous computing, RFID, WSNs, security and privacy, network modeling and digital arithmetic. He is a member of the IEEE.

Fabrizio Lamberti graduated with a degree in computer engineering in 2000 and received the PhD degree in computer engineering in 2005 from Politecnico di Torino, Italy. Since 2006, he has been an assistant professor at Politecnico di Torino. He has published a number of papers in the areas of digital arithmetic, mobile and distributed computing, image processing, and visualization. He has served as a program or organization committee member for several conferences. He is a member of Editorial Advisory Board of international journals. He is member of the IEEE and the IEEE Computer Society.

Gianluca Paravati received the MS degree in electronic engineering and the PhD degree in computer engineering from Politecnico di Torino, Italy, in 2007 and 2011, respectively. He is currently a research fellow with the Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interests include image processing, computer graphics and distributed architectures.

Jean-Claude Bajard received the PhD degree in computer science from the École Normale Supérieure de Lyon (ENS), France, in 1993. He taught mathematics in high school from 1979 to 1990 and served as a research and teaching assistant at the ENS in 1993. From 1994 to 1999, he was an assistant professor at the Université de Provence, Marseille, France. From 1999 to 2009, he was a professor at the University Montpellier 2, and was member of the ARITH Team of the LIRMM. He is currently a professor at the Université Paris 6, France, and a member of the LIP 6. His research interests include computer arithmetic and cryptography.

Paolo Montuschi received the PhD degree in computer engineering in 1989. He is a professor of computer engineering at Politecnico di Torino, chair of the Board for Financial External Affairs, member of the Board of Governors, and deputy chair of the Control and Computer Engineering Department. Previously, he served as chair of Department from 2003 to 2011, and as chair or member of several Boards. He is currently serving as an associate editor of the *IEEE Transactions on Computers*, as a member of the Board of Governors of the IEEE Computer Society, chair of the Electronic Products and Services Committee, and as a member of the Advisory Board of Computing Now. For more than 20 years, he has been a member of the IEEE Computer Society, serving as: chair and member of Digital Library Operations Committe, member-at-large of the Computer Society's Publications Board, member of an IEEE ad-hoc Committee for Quality of Conference Articles in IEEE-Xplore, and member of Conference Publications Operations Committee. He served as a guest and associate editor of the *IEEE Transactions on Computers* from 2000 to 2004. He was on the program committees for the 13th through 19th IEEE Symposia on Computer Arithmetic and program cochair of the 17th IEEE Symposium on Computer Arithmetic. In 2009, he served as a coguest editor for a special section on computer arithmetic in the *IEEE Transactions on Computers*. His research interests are in computer arithmetic, computer graphics, computer architectures, and electronic publications. Within the Computer Society, he is actively involved in opening the door to new publication frameworks geared toward e-reading and mobile devices. He is a senior member of the IEEE.