Contributions to the Design of Residue Number System Architectures

Benoît Gérard *†, Jean-Gabriel Kammerer * ‡, Nabil Merkiche * §
* DGA/MI, Rennes
† IRISA, Rennes benoit.gerard@irisa.fr
‡ IRMAR, Université de Rennes 1 jean-gabriel.kammerer@m4x.org
§ Sorbonnes Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France nabil.merkiche@lip6.fr

***Abstract*—Residue Number System (RNS) is nowadays considered as a real alternative to other hardware architectures for handling large-number computations. In this paper we propose algorithmic answers to some of the questions that may face a designer when implementing such solution. More precisely, we investigated the following three problems. First, we propose an efficient method for constructing maximal bases noticing that this problem can be seen as a max-clique problem. Second we consider the logical gates count reduction when two different bases share the same hardware modules. Again it is linked to graph theory since it corresponds to finding a maximum weighted matching. Eventually we detail how the presence of DSP blocks in FPGAs can be leveraged to reach higher design frequencies by implementing full computation units inside.**

***Keywords*—RNS,Graph Theory,Cox-Rower Architecture**

I. Introduction

During the past years the use of the Residue Number System (RNS) has been widely investigated. Applications range from signal-processing to many fields of cryptography (number-theoretic and elliptic curves based and lattice based). The point is to represent integers by their remainder modulo a set of small coprime integers called the *moduli basis* and reconstructing them using the Chinese Remainder Theorem [12] or Mixed Radix System [13]. Using RNS thus requires finding a large enough set of coprime moduli to dispatch computations. For all known applications it is not required that the product of the moduli equals a particular value but rather that it is larger than the manipulated numbers (plus a potential margin due to implementation choices).

In this paper we investigate algorithmic answers to some questions that may face anyone willing to implement RNS. Our work brings contributions that answer to the following three problems.

i) Constructing a basis producing the largest product modulo for a fixed size of computation units.
ii) Reducing logical gates count by sharing resources between two different basis.
iii) Leveraging on the presence of DSP blocks in a device to fully implement computation units into them.

Problem i) is a generic problem that is faced for any application of RNS. On the contrary, problem ii) and iii) correspond to applications that use Montgomery reduction over RNS (which essentially are number-theoretic and elliptic-curve based cryptography). Such architecture requires two RNS basis for performing Montgomery reduction. Each basis is used alternatively which allow a designer to share computation units between bases leading to problem ii) (which is specially relevant for ASIC implementations). Finally, problem iii) is of particular interest for implementations targeting FPGA. Indeed, being able to put the whole computation unit into DSP blocks will enable higher frequencies pushing the critical path to overrounding control.

We now provide a short description of our contributions to these problems.

a) Basis construction: We tackle the problem of finding a basis of coprime moduli whose product is large enough to enable targeted computations for a given moduli bitsize and a given architecture. More precisely for a given set of integers among which the moduli are to be chosen, we would like to find the subset that forms the largest RNS basis in terms of number of moduli. This problem is not mentioned in the literature: most of the time either the basis is considered given or it is constructed using a naive first come/first selected strategy.

We show here that it is possible to efficiently obtain optimal bases for typical cryptographic sizes of parameters. Our contribution is to express the problem as a max-clique problem that may be solved using off-the-shelf algorithms for this NP problem. We moreover propose a pre-processing algorithm leveraging the algebraic particularities of the problem in order to efficiently build a subset of such optimal basis. This allows decreasing the problem dimensions for it to become solvable using the aforementioned algorithms.

Being able to construct such maximal bases is crucial to determine which sizes of integer can be manipulated by a given RNS architecture. Indeed, our experiments have shown both positive and negative results that are of interest. A first example is the fact that 256-bits elliptic curves can actually be implemented with Kawamura *et al.* architecture [9] using 15-bits multipliers while this was not possible when using a naive basis construction method. As a second example, we can prove that 384-bits elliptic curves cannot be implemented using Kawamura *et al.* architecture and 16-bit multipliers.

b) Moduli matching: Second, we discuss the moduli matching problem that is most relevant for ASIC implementations of Montgomery over RNS. The main goal is to share logical gates between the two bases used for Montgomery

1063-6889/15 $31.00 © 2015 IEEE
DOI 10.1109/ARITH.2015.25

reduction. Concretely, the aim is to pair moduli from the two bases minimizing the sum of the couples Hamming weight. This problem can be transformed into a maximum matching problem, which is a graph theory problem that is solved with efficient (polynomial) algorithms. We show how to pair the moduli and discuss the possibility of taking this constraint into account during the basis choice.

c) Final subtraction removal: Our last contribution is related to the recently improved cox-rower architecture by Bajard *et al.* [2]. We suggest trading some freedom degrees on the moduli choice for removing the final subtraction allowing the implementation of the cox-rower to be fully made in DSP blocks. More precisely, reducing the range from which moduli can be chosen by one bit allows to remove this final reduction step.

Following this introduction, the paper is organized in four sections. First, we recall basics of RNS and the use of Montgomery representation over RNS. Then we present our first contribution regarding RNS basis construction. Further we propose a strategy to chose and to pair moduli from two different bases sharing the same hardware units. Before concluding we propose an adaptation of the Cox-Rower architecture presented in [2] to avoid the final reduction step at the cost of reducing the range of available moduli.

II. RNS AND MONTGOMERY OVER RNS

In this section we recall basics and introduce notations related to RNS and Montgomery representations.

A. Residue Number System.

The basic motivation for using RNS is because the cost of arithmetic operations is quadratic in the operand length[1]. To skirt this, RNS representation leverage on the Chinese Remainder Theorem by decomposing manipulated integers in smaller residues. This also enables simple parallelisation of operations and thus is of particular interest for hardware implementations of arithmetic on large numbers.

Let us consider the ring $\mathbb{Z}/M\mathbb{Z}$ where M is the product of small co-prime numbers:

$$M = \prod_{1 \leqslant i \leqslant n} m_i \quad \text{s.t.} \quad \forall 1 \leq i < j \leq n, \gcd(m_i, m_j) = 1.$$

Then, CRT states that ring $\mathbb{Z}/M\mathbb{Z}$ is isomorphic to $\mathbb{Z}/m_1\mathbb{Z} \times \mathbb{Z}/m_2\mathbb{Z} \times \cdots \times \mathbb{Z}/m_n\mathbb{Z}$. Thus we can make computations in the first ring using operations in the smaller ones. The set $\mathcal{B} = \{m_1, m_2, \ldots, m_n\}$ is referred as an *RNS basis*. Any element X belonging to $\mathbb{Z}/M\mathbb{Z}$ has an RNS representation in base $\mathcal{B}$ defined as $\{X\}_\mathcal{B} = (x_1, x_2, \ldots, x_n)$ with $x_i = X \mod m_i$.

Let X and Y be elements of $\mathbb{Z}/M\mathbb{Z}$ with corresponding RNS representations $\{X\}_\mathcal{B}$ and $\{Y\}_\mathcal{B}$, let Z be the result of the operation $X \odot Y$ where $\odot \in \{+, -, \times, \div\}$. Then, the RNS representation of Z is computed as

$$\{Z\}_\mathcal{B} = (z_1, z_2, \ldots, z_n) \text{ with } z_i = x_i \odot y_i \quad \forall 1 \leq i \leq n.$$

Notice that $X \div Y$ is only defined for values of Y for which $\gcd(M, Y) = 1$.

[1]Complexity can be lowered to $O(n^{log_2(3)})$ when using Karatsuba.

The last point to consider is how to convert a base-$\mathcal{B}$ RNS representation to a number in $\mathbb{Z}/M\mathbb{Z}$. Different techniques may be considered to do so but the most used one is the formula given in the constructive proof of the Chinese Remainder Theorem that is given below.

$$X = \left(\sum_{i=1}^{n} (x_i M_i^{-1} \mod m_i) M_i \right) \mod M, \quad (1)$$

where values $M_i = \frac{M}{m_i}$ and $M_i^{-1} \mod m_i$ are precomputed.

B. Montgomery Representation.

Montgomery representation has been introduced to decrease the cost of modular reductions. The naive way of reducing a number modulo another is to use Euclidean division which has a non negligible complexity and has input-dependent execution time. The Montgomery trick relies on the fact that computations are made on a hardware component that has a native base alphabet (up to now a binary alphabet) and that division by (*resp.* reduction modulo) 2^x thus corresponds to a right shift (*resp.* a bit-wise `and`) which are very simple operations. More generally, it benefits from being able to compute $\bmod R$ and $\div R$ operations at a very low cost (for a given R) to compute reduction modulo numbers that are coprime to R.

Numbers have first to be turned into Montgomery representation in order to use Montgomery reduction. Let X be an integer and p the integer we want to compute modulo (let us recall that p has to be co-prime with R). The Montgomery representation $\widetilde{X}$ of X is equal to $X \cdot R \mod p$. Correspondences between operations on integers and operations in Montgomery representation are

$$\begin{aligned} \widetilde{X \pm Y} &= \widetilde{X} \pm \widetilde{Y}, \\ \widetilde{X \times Y} &= \widetilde{X} \times \widetilde{Y} \times (R^{-1} \mod p), \\ \widetilde{X \div Y} &= \widetilde{X} \div \widetilde{Y} \times R. \end{aligned}$$

The Montgomery formula allows to compute $X \cdot R^{-1} \bmod p$ at low cost

$$\begin{aligned} \mathrm{Mred}(X) &= \frac{X + \left(X \cdot (-p^{-1}) \mod R\right) \cdot p}{R} \\ &= X \cdot R^{-1} \mod p. \end{aligned} \quad (2)$$

By construction this formula yields a result that is correct modulo p. Nevertheless, since X is larger than p, the result may also be larger than p. Actually, for a large enough R the output of Mred will be smaller than $2p$ (see [7] for instance for more details).

Notice that Mred can be used for computing Montgomery representation of a number since $\widetilde{X} = \mathrm{Mred}(X \times R^2)$ and for recovering the original value from the Montgomery representation since $X = \mathrm{Mred}(\widetilde{X})$. Moreover, addition/subtraction can be performed without making reduction modulo p since this will have a very small impact on the size of manipulated numbers[2]. Only the output of the multiplication should be

[2]More precisely, a designer should either choose parameters such that output of additions/subtractions will never grow beyond the maximum size or alternatively test for overflow and reduce modulo p if necessary [7].

reduced modulo p. In that case again the reduction can be used since $\widetilde{X \times Y} = \mathrm{Mred}\left(\widetilde{X} \times \widetilde{Y}\right)$.

Remark 1: As for the RNS representation some values have to be pre-computed namely $-p^{-1} \mod R$ and $R^2 \mod p$. Notice also that the cost of converting numbers from/to Montgomery representation has a cost thus using Montgomery is only relevant when it is amortized by the gain on the number of reductions, such as operations chains arising from elliptic curve point exponentiation.

Remark 2: In the particular case[3] where computations are of the form $\sum_i \widetilde{A}_i \times \widetilde{B}_i$, the reduction should only be made at the end of the accumulation. Again, designers should take care that such accumulation will not lead to overflows by carefully choosing parameters.

C. Montgomery Over RNS.

We saw that, on the one hand RNS allows efficient computations modulo a smooth number M but is not suited for prime moduli and, on the other hand, Montgomery representation takes profit of cheap computations modulo a number R to operate modulo a prime p. This suggests to use both techniques setting $R = M$. This idea has been explored but unfortunately the technical details are far less simple than it sounds. Nevertheless, implementations made up to now show that this idea is not meaningless and can be exploited to obtain efficient parallel implementations. We describe here how the use of Montgomery representation over an RNS decomposition has been achieved in previous works.

1) Main Principle: The main problem arising when combining both representations is the fact that the operation $\div R$ in Equation 2 is not computable modulo M since we took $R = M$. This problem is thwarted using a second RNS basis $\mathcal{B}'$ (with a modulo M' being co-prime with M) for computing this part of the formula then coming back to the original basis $\mathcal{B}$. We only describe here the proposed solution: for theoretical justifications the interested reader should refer to [11].

Algorithm 1 Montgomery Reduction in RNS

Inputs: $\{X\}_{\mathcal{B}}, \{X\}_{\mathcal{B}'}$ with $\prod_{m_i \in \mathcal{B}} m_i = M$
Outputs: $\{S\}_{\mathcal{B}}, \{S\}_{\mathcal{B}'}$
Precomputations: $\{-p^{-1}\}_{\mathcal{B}}, \{p\}_{\mathcal{B}'}, \{M^{-1}\}_{\mathcal{B}'}$

$\{Q\}_{\mathcal{B}} \leftarrow \{X\}_{\mathcal{B}} * \{-p^{-1}\}_{\mathcal{B}}$
$\{Q\}_{\mathcal{B}'} \leftarrow BE(\{Q\}_{\mathcal{B}}, \mathcal{B}, \mathcal{B}')$
$\{S\}_{\mathcal{B}'} \leftarrow (\{X\}_{\mathcal{B}'} + \{Q\}_{\mathcal{B}'} * \{p\}_{\mathcal{B}'}) * \{M^{-1}\}_{\mathcal{B}'}$
$\{S\}_{\mathcal{B}} \leftarrow BE(\{S\}_{\mathcal{B}'}, \mathcal{B}', \mathcal{B})$

Algorithm 1 describes the way Kawamura *et al.* performs Montgomery reduction over an RNS representation. The Base Extension function BE consists in reconstructing the first argument expressed in the first basis then decomposing the obtained integer into the second basis.

2) Base Extension: Base Extension is computed using the constructive proof of CRT. Indeed, the integer represented by $\{X\}_{\mathcal{B}} = \{x_1, x_2, \ldots, x_n\}$ is equal to $X = \sum_{i=1}^{n} \xi_i(x_i) M_i \mod M$ where $\xi_i(x_i) \stackrel{def}{=} x_i M_i^{-1} \mod m_i$ and $M_i = M/m_i$. Since this sum is generally larger than M it is equal to X plus a term $k(\xi_1(x_1), \ldots, \xi_n(x_n))M$. Unfortunately, the next step is decomposing the reconstructed number in basis $\mathcal{B}'$, that is computing $(X \mod M) \mod M'$. Obviously a nonzero value of $k(\xi_1(x_1), \ldots, \xi_n(x_n))$ (that we will denote by k) is a problem here and must be balanced by subtracting kM to the reconstructed X.

A floating point approach was proposed in [11] to compute the value of k. This approach was later improved by Kawamura *et al.* in [9] by approximating the value of

$$k = \left\lfloor \sum_{i=1}^{n} \frac{\xi_i(x_i)}{m_i} \right\rfloor \quad \text{by} \quad \hat{k} \stackrel{def}{=} \left\lfloor \alpha + \sum_{i=1}^{n} \frac{\mathrm{trunc_q}(\xi_i(\mathrm{x_i}))}{2^r} \right\rfloor,$$

where, r is the maximum bitsize of moduli, the truncation function is defined as $\mathrm{trunc_q}(\xi_i(\mathrm{x_i})) \stackrel{\mathrm{def}}{=} \left\lfloor \frac{\xi_i(\mathrm{x_i})}{2^{\mathrm{r-q}}} \right\rfloor 2^{\mathrm{r-q}}$ (for $0 \leq q < r$) and $\alpha \in (0, 1)$. The role of α is to reduce the range $[0, M)$ of possible values for X to a smaller one in order to ensure that the estimate $\hat{k}$ **does always** correspond to k as stated in the following theorem from [9].

Theorem 1 (Kawamura *et al.* [9])**.** *Let*

$$\epsilon \stackrel{def}{=} \max_{1 \leq i \leq n} \frac{2^r - m_i}{2^r} \quad \textit{and}$$

$$\delta \stackrel{def}{=} \max_{1 \leq i \leq n} \max_{x_i} \frac{\mathrm{trunc_q}(\xi_i(\mathrm{x_i})) - \xi_i(\mathrm{x_i})}{m_i}.$$

Then, if $0 \leq n(\epsilon + \delta) \leq \alpha < 1$ *and* $0 \leq X < (1 - \alpha)M$*, then* $\hat{k} = k$ *and thus the base extension function extends the base without error.*

Note that the conditions for this theorem to hold are not tight. Refined conditions can be found in [2].

D. Cox-Rower architecture

In order to compute Algorithm 1, an architecture was proposed in [9] called the Cox-Rower architecture. This architecture is composed of 3 parts: a sequencer unit, a set of Rower unit and a Cox unit. The sequencer unit is the control unit in charge of the global coherency of the computation (execution of the main algorithm such as binary method or window method). The set of Rower unit and the Cox unit together are the datapath of this architecture. A Rower unit computes the product $z_i = a_i b_i \mod m_i$ and accumulates it. The Cox unit is in charge of evaluating the factor $\hat{k}$ during the BE function implemented as in [9]. Figure 1 gives an overview of the Cox-Rower architecture.

III. Graph Theory Approach to Basis Construction

Several moduli selection strategies have been proposed depending on the targeted architecture. For instance, Bajard *et al* [1] exhaustively searched for bases with very sparse moduli. Guillermin [8] chose to minimize the gap between the moduli and 2^r. His reduction with 3 multiplications required the lower bound on moduli to be larger than $2^r - 2^{r/2}$. He used a greedy "first come, first selected" strategy: first add 2^r and $2^r - 1$ (which are coprime) then run through other integers of the

[3] This situation corresponds to the Base Extension presented in the next Section II-C.

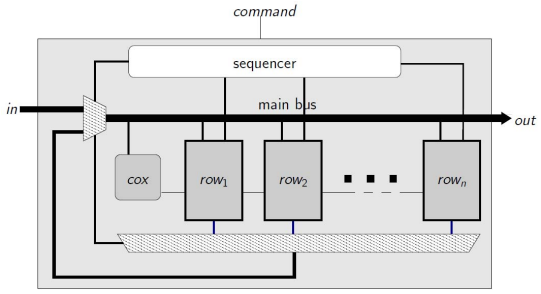

Fig. 1. Cox-Rower Architecture

range in decreasing order and add them to the basis if they are coprime with every already selected modulus.

The question we answer in this section is the one of forming the largest basis (in terms of number of moduli) for a given bitsize. For cryptographic applications this upper bound can be translated to the maximum reachable security level. In the case of signal processing it is directly linked to the maximum precision for computations.

A. Moduli Selection as a Max-Clique Problem

We propose a strategy to build an RNS basis with maximal cardinality, given a moduli range from which moduli can be taken. Such ranges correspond to intervals of the form $[\![2^r - \mu, 2^r]\!]$ in the typically used architectures (see [9], [2] for instance). The problem is then to find the largest set of coprime integers within such interval.

This problem can actually be viewed as a graph problem. Indeed consider the graph with vertices labeled by the integers in the aforementioned interval and with edges joining two vertices if and only if the corresponding integers are coprime. The problem of finding the larger basis of coprime integers boils down to finding a maximum clique (namely one of the complete subgraphs containing the maximum possible number of vertices).

Finding a maximum clique is NP-complete in general, but we can simplify the graph before computing it. There is indeed a strong algebraic structure beside the graph we consider here. The strategy we propose is to first perform a pass to select (*resp.* discard) integers for which we already know that they will belong (*resp.* not belong) to a max-clique. Then, the pruned graph will either contain no vertices at all (in which case it is already finished) or be small enough for the max-clique problem to be solved in reasonable time. Albeit exponential in general, efficient algorithms and heuristics have been devised for solving the clique problem, which work very well in practice with these (quite intensively) pruned graphs. Available off-the-shelf maximum clique solvers include Cliquer [10] or Magma [3].

B. Reducing the Graph Size

Let E_0 contains all possible moduli that is $E_0 = [\![a, b]\!]$ for some a and b. Let $\mathcal{G}(E_0)$ be the corresponding graph (with edges between coprime vertices). Note that a and b are small since they may correspond to RNS operands, ranging typically from 16 to 32 bits[4]. Since the operands are small, we suppose the prime number factorization of the elements in E_0 are known.

The intuition of the proposed pruning algorithm is that selecting a modulo into the basis will disable the choice for all moduli having a common prime factor. The goal is thus to select first the moduli that will disable the smallest number of integers.

Lemma 1. *Let $B_0 \subset E_0$ be the set of prime numbers in E_0. Then, there exists a maximal basis $\mathcal{B}_m$ such that $B_0 \subset \mathcal{B}_m$.*

Proof: Let $\mathcal{B}_m$ be a maximal basis. For each element e in B_0, a multiple of e can be found in $\mathcal{B}_m$ (otherwise e could be added contradicting the maximal property of $\mathcal{B}_m$). Replacing this multiple with e in the maximal basis does not alter the validity of the basis. We end up with a maximal basis containing B_0. ■

We can thus select numbers in B_0 as moduli and thus remove from E_0 all integers not being coprime with elements in B_0. We denote by E_1 the set obtained by this removal. A corollary of Lemma 1 is that the union of any maximal basis of E_1 and B_0 yields a maximal basis of $[\![a, b]\!]$.

Next, we observe that some prime factors may appear only once in the interval, such as the primes Λ such that $\Lambda > (b-a)$.

Lemma 2. *Let $E_1' \subset E_1$ be the set of elements having exactly one small prime factor:*

$$E_1' = \{\pi^e \Lambda \in E_1 | \pi \text{ is prime}, \pi \leqslant (b-a), \\ \Lambda = 1 \text{ or } \Lambda \text{ is prime and } \Lambda > (b-a)\}.$$

Let $B_1 \subset E_1'$ be a maximal basis[5] of E_1' . Then, there exists a maximal basis $\mathcal{B}_m$ of E_1 such that $B_1 \subset \mathcal{B}_m$.

Proof: See appendix. ■

Again we select numbers in B_1 as moduli and thus remove from E_1 all integers not being coprime with elements in B_1. We denote by E_2 the set obtained by this removal. Together Lemma 1 Lemma 2 implies that the union of B_0, B_1 and any maximal basis of E_2 yields a maximal basis of $[\![a, b]\!]$.

Unfortunately we cannot go further without loosing generality. As an illustration, assume that there remains three integers $3 \cdot 5 \cdot \Lambda$, $3 \cdot 7 \cdot 11 \cdot \Lambda'$ and $5 \cdot 13 \cdot 17 \cdot \Lambda''$ in the set. Then, we should not pick $3 \cdot 5 \cdot \Lambda$ but the two others that have more small factors.

Before sending E_2 to the max-clique solver, we can filter out elements of E_2 being coprime with any other element and add them to the current basis.

Remark 3: When computing B_1, once an even number has been added to the current basis (*e.g.* 2^r) the bound $\Lambda > b - a$ in Lemma 2 can be relaxed to $\Lambda > (b-a)/2$.

[4] Thus asymptotic estimations on the density of prime and smooth numbers are too coarse and are not of any help, as would be complexity estimates on the algorithms.

[5] B_1 can be obtained by removing, for each π, all but one multiples of π in E_1'.

Remark 4: One may also go slightly further with pruning by considering integers in E_2 which have a large prime factor $\Lambda > b - a$ that appears only once (that is, it only appears as a factor of this specific integer). Such element can be added to the basis if and only if none except at most one of the small primes in its decomposition appear in any other decomposition of elements in E_2.

C. Applications / Examples

We implemented the pruning algorithm in a straightforward manner in C++. The maximum-clique was found using Cliquer [10].

With 16 bits moduli, on the Cox-Rower architecture proposed by Guillermin [8] (that is $\mu = \sqrt{2^{16}} = 2^8$), 43 moduli are found using the first come first selected approach, which in turn gives only 42 usable moduli since we need two bases. Our approach finds 49 moduli, 48 of which are given in appendix A. Eliminating 6 bits due to the ALU implementation [8], our algorithm chooses a basis that allows to compute on 381-bits elliptic curves, short of the classical 384 cryptography standard which provably cannot be attained with 16-bits moduli and this ALU. Notice that the first come first selected approach reaches only 333 bits that is 48 bits less than ours. In this particular settings, the graph is reduced to only one element after the first stage pruning. One could always increase the size of μ beyond $\sqrt{2^{16}}$ but it will lose the generic architecture of the Rower and would require more subtraction to complete the reduction (which adds more complexity and testability to the hardware).

Our experiments with r ranging from 16 to 24 bits and moduli ranging from $2^r - 4\sqrt{2^r}$ to 2^r showed we missed at most 1 modulus compared to clique solving after pruning E_1 when pruning also E_2, then E_3 and so on until exhaustion instead of computing a maximum-clique on the reduced graph. For example, with the $2^{24} - 4\sqrt{2^{24}}, 2^{24}$ interval, the initial graph has size 16384 while after rigorous pruning (primes and only one small prime) it has only 326 nodes. In this case, the maximum clique on the pruned graph has 240 elements while pruning the graph until exhaustion will find 239 elements in the E_2 set and none in E_i with $i \geqslant 3$, missing only 1 element compared to the optimal case. All in all, the max-clique method finds 1395 elements compared to 1375 with the first come first selected one.

Our experiments showed that pruning is indeed necessary with large intervals since computing a maximum clique directly on the whole set of integers may become quite difficult. This effect seems to depend on the graph topology rather than its size. As an example, on a standard laptop, computing a max-clique using Cliquer on the interval $[2^{16} - 4 \cdot \sqrt{2^{16}}, 2^{16}]$ without pruning (the graph having then 1024 nodes) takes about a minute. Surprisingly, it only takes less than 2 seconds to compute a clique of the pruned graph arising from the interval $[2^{32} - 3 \cdot \sqrt{2^{32}}, 2^{32}]$ which contains as much as 1786 nodes.

IV. Pairing Moduli for Hardware Pooling

A. Introduction.

After the selection of an admissible set among which we can choose enough moduli, we need to build the base needed by RNS ALUs. Building the two sub-bases $\mathcal{B}$ and $\mathcal{B}'$ amounts to selecting a (partial) *matching* of these moduli. Several matchings may be better than others in a performance or resource consumption perspective.

Previous section gives a rather large room for selecting optimal bases depending on the target architecture. If we target FPGA with fixed size DSP blocks, where multipliers are essentially free, we would look for a gain on the reduction step and try to mutualize resources by matching sub-bases elements. If we target ASIC or FPGA without DSP and do the full computation in the logic, it would be nice to reduce the area needed by the multipliers (which will by far dominate the total area). In this case, we will want to choose a base that minimize multipliers cost (thus with quite sparse base elements) and maximize the hardware reuse (thus matching elements that are closest to one another). These idea are not new [1], [4] but rather than heuristics, our formulation of these problems as a graph one makes it possible to provably find the optimal solution.

B. Optimization modeling.

Let $\mathcal{B}_{\max} = (m_i)_{1 \leqslant i < N}$ be a set of moduli of r bits given by our moduli selection algorithm. We have to build two sub-bases $\mathcal{B}$ and $\mathcal{B}'$ among this set, each one of cardinality n (thus $2n \leqslant N$). Build the complete graph with N vertices where each vertex is labelled by a modulus. Then the selection of two sub-bases and the association of moduli by pairs amounts to choosing a matching in the graph. The matching is perfect if $N = 2n$, that is when we have to put every modulus into one sub-basis. In this case, the matching covers all vertices of the graph.

We then add a cost function to the edges. Let each edge have weight corresponding to the hardware cost $c(m_i, m_j)$ of a computation element able to be used alternatively by these two moduli if we would match them. This hardware cost corresponds to the area complexity of hardware elements.

For example, a multiplier always having the same modulus as input needs as many shift and add steps as the Hamming weight of the modulus. Thus the area cost of a multiplier can be modeled by $rHW(m_i \text{ OR } m_j)$ when it's alternatively used by m_i and m_j. Note that reducing the number of shift and add steps of a multiplier also reduces its latency, which may be beneficial if it turns out it is on the critical path. However, while this modeling is very good for minimizing the area, it might be refined. It certainly improves area but does not care as much as could be possible about improving latency. For example, minimizing the output register bitsize will slightly improve area, but will certainly improve the latency since the carries in the adders will be propagated far less. Minimizing both the Hamming weight and the result bitsize could be combined into the cost function, or we could want to improve the area first and then improve the latency of the slowest multiplier among the different results. The exact compromise between area and latency depends too much on the application to be explored here further.

C. Optimization algorithm.

Once modeled, the area cost of a subbasis selection and matching corresponds to the total weight of the edges of

the corresponding matching. Minimizing hardware cost of associated to a sub-basis results in finding a minimal weight matching (or perfect matching when $N = 2n$). Fortunately, efficient algorithms have been devised for this problem, initiated by the blossom algorithm proposed by Edmonds [6], [5]. It finds minimum weight perfect matching in time $O(N^4)$ on a complete graph. Variants of this algorithms find minimum weight (partial) matching of prescribed cardinality, which is precisely our problem: given a security parameter, we know exactly how many moduli pairs we will need.

D. Refinements.

When computing a maximal basis using the idea of Section III, there's a new degree of freedom. Indeed when selecting the maximal sets E_i, several choices may be equivalent from a basis size perspective that may change things a lot from a Hamming weight point of view. For example, suppose there exists two elements 3Λ and $3\Lambda'$. Both cannot be in the basis simultaneously since they are not coprime, but any of them may belong to a maximal cardinality basis. It could be useful to defer the choice between them to the matching phase and choose the best one for the implementation.

Consider two elements μ and μ' in our moduli set. Let (μ_k) and (μ'_l) be the families of elements that can be used indistinctively in place of μ and μ'. That is, for each k, μ_k is coprime with any element in $\mathcal{B}_{\max}$ except μ. Then label the vertices with the families μ_k and μ_l instead of (solely) μ and μ'. Define the cost of the edge joining these two nodes as $\min_{k,l} c(\mu_k, \mu'_l)$. If this edge is selected in the minimum weight matching, select the indices k and l corresponding to that cost.

E. Experimental results.

Since we mainly targeted FPGA with DSP blocks where multiplier optimization was not necessary, we tried to find a cost model for the ALU hardware elements computing the final reduction step, which primarily involves adders. Since synthesis tools already do a very good job at optimizing these, especially with constants, we did not find a very significant gain (less than one percent on the total area needed by the ALU on the logic part of the FPGA). Most of the benefits of our formulation of the optimization problem as a matching one would be for multipliers. Unfortunately, we did not have ASIC synthesis tools at hand. Nonetheless, the optimization algorithm does better global choices than local heuristics such as [4], [1] which already obtained significant results. Moreover, it is amenable to any cost function: several different ones may be tried and fine-tuned to find the best optimization combination between moduli selection and synthesis.

V. Removing the Final Reduction from Cox-Rower

A. Introduction

In the Cox-Rower architecture, when setting radix size to fit in a single DSP block on FPGA, a way to improve the overall latency is to increase the frequency. Finding the critical paths of such design and reducing them can be done by increasing the pipeline if the critical paths reside in the datapath. A comparison of the two existing ALUs have been done on the same FPGA in [2]. It shows that the critical paths of the classical ALU resides in its addition/reduction part whereas the critical paths of the inner Montgomery ALU resides in interconnection between DSP blocks when using a pipeline depth of 5. It also shows an improvement of 20% of the frequency over the classical ALU (285Mhz *vs.* 235Mhz). In order to further increase the frequency, we propose an improvement of Kawamura *et al.* conditions in order to delete the final reduction stage in the inner Montgomery ALU doing the full computation in the DSP blocks on FPGA and thus, moving the critical path from the datapath to the sequencer unit.

B. Correction using k

Conditions for base extension to output the correct value are recalled in Theorem 1. These conditions have been refined by Bajard *et al.* in [2]. In both works, the base extension is defined for $\xi_i(x_i) < m_i$ and thus conditions on the parameters are derived using this assumption.

The point is that the computation of $x_i\, M_i^{-1} \mod m_i$ using both the classical ALU or the inner Montgomery one yields numbers larger than m_i (up to $4m_i$ for the classical one and $2m_i$ for the inner Montgomery one). Thus they have to perform test(s) and potential subtraction(s) before using the result as input of the base extension.

Nevertheless, one may want to investigate the applicability of Kawamura *et al.* approach in the case $\xi_i(x_i) < l\, m_i$ for a value l equal to 2 or 4 for instance. Indeed, this would remove the final subtractions, leading to a high frequency gain for FPGA implementations (as detailed later on). We propose here to derive conditions on the different parameters for the base extension to return the correct value when allowing $\xi_i(x_i)$'s being a bit larger than expected.

First, let us denote by $\xi'_i(x_i)$ the value obtained when computing $x_i M_i^{-1}$ in a rower. This value is correct modulo m_i (that is $\xi'_i(x_i) = \xi_i(x_i) \mod m_i$) but may be larger than m_i. More precisely, let us say that $\xi'_i(x_i) = \xi_i(x_i) + \beta_i\, m_i$ with $\beta_i \in \{0, 1, \ldots, l-1\}$. Then, the computation of $\sum_{i=1}^n \xi'_i(x_i) M_i$ will be equal to $\sum_{i=1}^n \xi_i(x_i) M_i$ plus a new error $M \sum_i \beta_i$. Restricting the range from which moduli can be selected, this additional term will be balanced by an other additional term appearing in $\hat{k}(\xi'_1(x_1), \xi'_2(x_2), \ldots, \xi'_n(x_n))$. More precisely, dividing by l the range will enable the correctness of base extension for values of β_i smaller than l. The following proposition summarizes this statement.

Proposition 1. *If* $0 \leqslant 2^r - m_i \leqslant \frac{\alpha}{l \cdot n} 2^r$, $0 \leqslant \alpha < 1$, $0 \leqslant x < (1-\alpha)M$, *and* $\forall i \in [\![1, n]\!], 0 \leqslant \xi_i < l \cdot m_i$ *then* $\hat{k} = k$ *and the base extension function extends without error.*

The proof of Proposition 1 is given in Appendix. The notation of the conditions given in Proposition 1 are the one rewritten in [2].

C. Hardware Architecture of the ALU

The inner Montgomery ALU has been redesigned in order to implement all the features in the DSP blocks. Figure 2 gives an overview of the new ALU architecture. Hereafter

TABLE I. P&R PERFORMANCES AND COMPARISONS

Curve	n	Cycles	Slices (DSP/BRAM)	Fmax	Latency
160	11	88536	1168 (45/13)	400	0,221 ms
192	13	117732	1316 (53/15)	400	0,297 ms
224	15	150795	1542 (61/17)	400	0,377 ms
256	17	187828	1658 (69/19)	400	0,472 ms
384	25	373946	2630 (101/27)	400	0,935 ms
512	33	622053	3405 (133/35)	400	1,555 ms
521	33	632588	3435 (133/35)	400	1,581 ms

are summarized the features that have been added or modified from the original proposition in [2]:

1) The size of the radix has to be diminished by 1 (from $r = 17$ to $r = 16$ in Xilinx FPGA) in order to not reduce the results.
2) The final accumulator/reducer has been moved to the adder of the first multiplication as proposed in [7]. In order to keep the results of the accumulated reduction $< 2m_i$, we need to use $R = 2^{r+l}$ as the Montgomery representation in the ALU. The value l can be evaluated as the base-2 logarithm of the maximum reachable value of the sum during the whole scalar multiplication (including the BE function). For instance, the value for $r = 16$ and $\log_2(p) = 521$ is $l = 7$.
3) The pipeline depth has been increased from 5 to 10. The algorithm used to compute the scalar multiplication (which is Montgomery Ladder) does not penalize the pipeline occupancy.

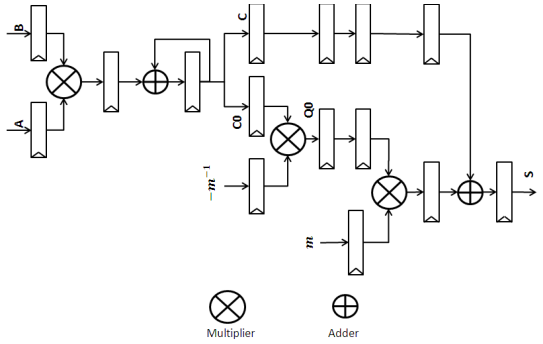

Fig. 2. Inner Montgomery ALU Architecture without final reduction

D. Implementation and Results

When increasing the pipeline depth from 5 to 8, the critical paths move from the interconnection between DSP blocks to the accumulator/reducer parts of the inner Montgomery ALU. The gain in frequency is only of 10%. The accumulator/reducer part of the design is implemented in LUTs. By using the ALU proposed here and by increasing the pipeline depth to 10, we manage to reach a frequency of 400Mhz and to move the critical paths from the ALU to the sequencer unit. Table I gives details of implementations results with radix size $r = 16$ and $\alpha = 0.5$.

We now compare the attainable integer sizes obtained using the max-clique approach for the three ALUs (classical, inner Montgomery and the one proposed in this section). These ALUs may be used for two kind of applications, namely modular arithmetic and computation over $\mathbb{Z}$. We summarize in Table II these attainable integer sizes for modular arithmetic applications (*e.g.* elliptic curves or RSA). These sizes have to be doubled when computing only with integers, since in this case there's no need to have two moduli bases.

In particular, the moduli range is large enough with our inner Montgomery ALU without reduction to compute on *e.g.* the NIST 521-bits elliptic curve even with 16-bits multiplier sizes, whereas Kawamura *et al.* [9] ALU needs at least 18-bits moduli.

TABLE II. ATTAINABLE INTEGER SIZES FOR ALUS

	With reduction		Without reduction
Multiplier bitsize	Kawamura *et al.*	Inner Montgomery	Inner Montgomery
15	266	536	375
16	380	782	536
17	506	1118	782

REFERENCES

[1] Jean-Claude Bajard, Marcelo E. Kaihara, and Thomas Plantard. Selected RNS Bases for Modular Multiplication. In Javier D. Bruguera, Marius Cornea, Debjit Das Sarma, and John Harrison, editors, *IEEE Symposium on Computer Arithmetic*, pages 25–32. IEEE Computer Society, 2009.

[2] Jean-Claude Bajard and Nabil Merkiche. Double level montgomery cox-rower architecture. In *CARDIS 2014*, LNCS. Springer, 2014.

[3] W. Bosma, J. Cannon, and C. Playoust. The Magma Algebra System I: The user language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.

[4] Ray C. C. Cheung, Sylvain Duquesne, Junfeng Fan, Nicolas Guillermin, Ingrid Verbauwhede, and Gavin Xiaoxu Yao. FPGA Implementation of Pairings Using Residue Number System and Lazy Reduction. In *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'11, pages 421–441, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965.

[6] Jack Edmonds. Paths, Trees, and Flowers. *Canad. J. Math.*, 17:449–467, 1965.

[7] Shay Gueron. Enhanced Montgomery Multiplication. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 46–56. Springer, 2002.

[8] Nicolas Guillermin. A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over $\mathbb{F}_p$. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop*, volume 6225 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2010.

[9] Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-Rower Architecture for Fast Parallel Montgomery Multiplication. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 523–538, Berlin, Heidelberg, 2000. Springer-Verlag.

[10] S. Niskanen and P.R.J. Östergård. *Cliquer User's Guide: Version 1.0*. Helsinki University of Technology, 2003.

[11] K.C. Posch and R. Posch. Modulo reduction in residue number systems. *Parallel and Distributed Systems, IEEE Transactions on*, 6(5):449–454, May 1995.

[12] P. P. Shenoy and R. Kumaresan. Fast Base Extension Using a Redundant Modulus in RNS. *IEEE Trans. Comput.*, 38(2):292–297, February 1989.

[13] Nicholas S Szabó and Richard I Tanaka. *Residue arithmetic and its applications to computer technology*. New York : McGraw-Hill, 1967. Based on the authors' Report on residue (modular) arithmetic survey.

APPENDIX

PROOF OF LEMMA 2

Let $\mathcal{B}_m$ be a maximal basis of set E_1 and B_1 be a maximal basis of E_1'. The aim of this proof is to show that we can construct a maximal basis containing B_1 from $\mathcal{B}_m$. We will show that each element in B_1 that does not belong to $\mathcal{B}_m$ can be swapped with an element of $\mathcal{B}_m$ without altering the maximal property of this last one.

Let $p = \pi^e \cdot \Lambda$ be an element of B_1 that does not belong to $\mathcal{B}_m$. Then, p cannot be added to $\mathcal{B}_m$ otherwise it would contradict the fact that $\mathcal{B}_m$ is maximal.

Thus, we deduce that there exists an element $m \in \mathcal{B}_m$ such that $\gcd(p, m) \neq 1$. This implies that either $\gcd(\pi, m) = \pi$ or $\gcd(\Lambda, m) = \Lambda$.

Since Λ is prime and larger than $b - a$, Λ appears at most once in the decomposition of integers in $[\![a, b]\!]$. Since $m \neq p$ we conclude that $\gcd(\Lambda, m) = 1$ and $\gcd(\pi, m) = \pi$.

Thus, the basis $\mathcal{B}_m \backslash m \cup p$ still is a valid basis (that is containing pairwise co-prime integers) and also have the maximal cardinality. It is hence a maximal basis containing p.

Iterating the reasoning on all elements in B_1 concludes the proof.

BASIS COMPUTATION EXAMPLE

Basis elements between $2^{16} - 2^8$ *and* 2^{16}

Primes: 65521 65519 65497 65479 65449 65447 65437 65423 65419 65413 65407 65393 65381 65371 65357 65353 65327 65323 65309 65293 65287

Numbers with exactly one small prime factor: 65536 ($= 2^{16}$) 65531 65529 65525 65509 65507 65503 65501 65491 65489 65483 65477 65473 65459 65443 65431 65411 65389 65383 65369 65363 65347 65339 65321 65311 65281

Number with exactly 3 small prime factors: 65453

PROOF OF THE PROPOSITION 1

We want to derive conditions for the base extension to return the correct value in the case where we obtain values $\xi_i'(x_i) = \xi_i(x_i) + \beta_i\, m_i$ with integer β_i smaller than l.

The value X that has to be computed is

$$X = \sum_{i=1}^{n} \xi_i(x_i)\, M_i - \left\lfloor \sum_{i=1}^{n} \frac{\xi_i(x_i)}{m_i} \right\rfloor M.$$

Assuming now that we only get the larger values $\xi_i'(x_i)$, we want to compute

$$\begin{aligned} X &= \sum_{i=1}^{n} \xi_i'(x_i)\, M_i - \left\lfloor \sum_{i=1}^{n} \frac{\xi_i'(x_i)}{m_i} \right\rfloor M \\ &= \sum_{i=1}^{n} \xi_i(x_i)\, M_i + M \sum_{i=1}^{n} \beta_i - \left\lfloor \sum_{i=1}^{n} \frac{\xi_i(x_i) + \beta_i\, m_i}{m_i} \right\rfloor M \\ &= \sum_{i=1}^{n} \xi_i(x_i)\, M_i - \left\lfloor \sum_{i=1}^{n} \frac{\xi_i(x_i)}{m_i} \right\rfloor \end{aligned}$$

We thus want to derive conditions for $\hat{k}(\xi_1'(x_1), \xi_2'(x_2), \ldots, \xi_n'(x_n))$ to correspond to $k(\xi_1'(x_1), \xi_2'(x_2), \ldots, \xi_n'(x_n))$. To do so we use a similar approach as in previous works [9], [2].

$$\begin{aligned} \sum_{i=1}^{n} \frac{\mathrm{trunc_q}(\xi'(\mathrm{x_i}))}{2^r} &= \sum_{i=1}^{n} \frac{\mathrm{trunc_q}(\xi'(\mathrm{x_i})) - \xi'(\mathrm{x_i})}{2^r} \\ &+ \sum_{i=1}^{n} \frac{m_i}{2^r} \cdot \frac{\xi'(x_i)}{m_i} \\ &= -\sum_{i=1}^{n} \frac{\xi'(x_i) - \mathrm{trunc_q}(\xi'(\mathrm{x_i}))}{2^r} \\ &+ \sum_{i=1}^{n} \frac{m_i - 2^r}{2^r} \cdot \frac{\xi'(x_i)}{m_i} + \sum_{i=1}^{n} \frac{\xi'(x_i)}{m_i}. \end{aligned}$$

Since $\xi'(x_i) - \mathrm{trunc_q}(\xi'(x_i)) \leqslant 2^{r-q} - 1$ and $\xi_i'(x_i)/m_i < l$ and $m_i - 2^r \leqslant 0$, we obtain

$$\begin{aligned} \sum_{i=1}^{n} \frac{\mathrm{trunc_q}(\xi'(x_i))}{2^r} &> -\sum_{i=1}^{n} \frac{2^{r-q} - 1}{2^r} \\ &+ l \cdot \frac{\sum_{i=1}^{n} m_i - 2^r}{2^r} + \sum_{i=1}^{n} \frac{\xi'(x_i)}{m_i} \\ &> \sum_{i=1}^{n} \frac{\xi'(x_i)}{m_i} \\ &- \frac{n\left(2^{r-q} - 1\right) + l \sum_{i=1}^{n} 2^r - m_i}{2^r} \end{aligned}$$

On the other hand, $\sum_{i=1}^{n} \frac{\mathrm{trunc_q}(\xi'(x_i))}{2^r} \leqslant \sum_{i=1}^{n} \frac{\xi'(x_i)}{m_i}$. Thus, for $\alpha \geqslant \frac{n\left(2^{r-q}-1\right) + l \sum_{i=1}^{n} 2^r - m_i}{2^r}$, the value

$$\hat{k}(\xi_1'(x_1), \xi_2'(x_2), \ldots, \xi_n'(x_n)) = \left\lfloor \alpha + \sum_{i=1}^{n} \frac{\mathrm{trunc_q}(\xi_i'(x_i))}{2^r} \right\rfloor$$

will be equal to $\sum_{i=1}^{n} \frac{\xi'(x_i)}{m_i}$.

This yields the following condition for $\sum_{i=1}^{n} 2^r - m_i$ (denoted as $\sum_{i=1}^{n} \mu_i$ in [9], [2]):

$$\sum_{i=1}^{n} 2^r - m_i \leqslant \frac{\alpha\, 2^r - n\left(2^{r-q} - 1\right)}{l}$$

and thus the stronger condition

$$\sum_{i=1}^{n} 2^r - m_i \leqslant \frac{\alpha\, 2^r}{l}$$

is sufficient which proves Proposition 1.