Double Level Montgomery Cox-Rower Architecture, New Bounds

Jean-Claude Bajard, Nabil Merkiche

Double Level Montgomery Cox-Rower Architecture, New Bounds

Jean-Claude Bajard[1] and Nabil Merkiche[2]

[1] Sorbonnes Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France, CNRS, UMR 7606, LIP6, F-75005, Paris, France
[2] DGA/MI, Rennes
jean-claude.bajard@lip6.fr, nabil.merkiche@intradef.gouv.fr

Abstract. Recently, the Residue Number System and the Cox-Rower architecture have been used to compute efficiently Elliptic Curve Cryptography over FPGA. In this paper, we are rewriting the conditions of Kawamura's theorem for the base extension without error in order to define the maximal range of the set from which the moduli can be chosen to build a base. At the same time, we give a procedure to compute correctly the truncation function of the Cox module. We also present a modified ALU of the Rower architecture using a second level of Montgomery Representation. Such architecture allows us to select the moduli with the new upper bound defined with the condition. This modification makes the Cox-Rower architecture suitable to compute 521 bits ECC with radix downto 16 bits compared to 18 with the classical Cox-Rower architecture. We validate our results through FPGA implementation of a scalar multiplication at classical cryptography security levels (NIST curves). Our implementation uses 35% less LUTs compared to the state of the art generic implementation of ECC using RNS for the same performance [5]. We also slightly improve the computation time (latency) and our implementation shows best ratio throughput/area for RNS computation supporting any curve independently of the chosen base.

Keywords: Residue Number System, High Speed, Hardware Implementation, Elliptic Curve Cryptography, FPGA

1 Introduction

The Residue Number System (RNS) has shown interest for efficient implementation and high performances in large integer computations for public key cryptography and digital signature[6,5]. Due to the ability to compute any operation quickly ($O(n)$ complexity in RNS vs $O(n^{\log_2(3)})$ in multiprecision for multiplications when using Karatsuba) without carry propagation and with natural parallelism, RNS has gained interest in the literature[11,12,1]. Recently, it has also been demonstrated to be suitable for pairing computations[3,13]. Improvement has been made for efficient computation of the final exponentiation in [2]. All these implementations are based on the Cox-Rower architecture proposed by Kawamura for RSA [6] and improved by Guillermin for ECC computations [5].

In this paper, we reformulate the conditions for the base extension in order to build bases for the RNS Cox-Rower. Then, we present a new ALU that takes advantages of the new conditions for the base extension.

The paper is organised as follow: in Section 2, we will recall briefly mathematical background about RNS, Montgomery over RNS and approximations made in the base extension. Section 3 deals with the range of the moduli set induced by the approximation made during the base extension. The truncation function of the Cox is re-evaluated under those conditions. Section 4 presents a new Rower architecture, together with its base extension algorithm, to take advantage of the maximal range of the moduli set defined in Section 3. Section 5 gives results with scalar multiplication as well as area and performance comparisons with the classical Rower architecture. Section 6 concludes the paper.

2 Background Review

2.1 Residue Number System

RNS represents a number using a set of smaller integers. Let $\mathfrak{B} = \{m_1, \ldots, m_n\}$ be a set of coprime natural integers. $\mathfrak{B}$ is also called a base. Let $M = \prod_{i=1}^{n} m_i$. The RNS representation of $X \in \mathbb{Z}/M\mathbb{Z}$ is the unique set of positive integers $\{X\}_{\mathfrak{B}} = \{x_1, \ldots, x_n\}$ with $x_i = X \mod m_i$. The conversion from RNS representation to binary representation can be computed using the Chinese Remainder Theorem (other methodology as Mixed Radix is possible):

$$X = \left(\sum_{i=1}^{n} (x_i M_i^{-1} \mod m_i) M_i \right) \mod M \text{ with } M_i = \frac{M}{m_i} \tag{1}$$

Operations in RNS are computed as follows:
$\forall X, Y \in \mathbb{Z}/M\mathbb{Z}, \exists Z \in \mathbb{Z}/M\mathbb{Z}$ s.t.:

$$Z = X \odot Y \mod M \Leftrightarrow z_i = x_i \odot y_i \mod m_i \text{ with } \odot \in \{+, -, *, \div\}$$

and $\div$ only available when Y is coprime with M and a divisor of X.

Notation: In the rest of the paper, $\{X\}_{\mathfrak{B}}$ will refer to the representation of X in the RNS base $\mathfrak{B}$. We use braces to denote the fact that this is a set of integers.

2.2 RNS and Montgomery

RNS arithmetic has several drawbacks over multiprecision arithmetic. One of them is that reduction over p is complex. Reduction over p is still possible when using Montgomery Reduction since it computes exactly the value using a base extension[9,10]. Thereafter, we recall the algorithm to compute the Montgomery Reduction in RNS[6,5,10].

The main part of Montgomery Reduction relies on the Base Extension function (BE in the algorithm) that is described in the next section.

Algorithm 1: Montgomery Reduction in RNS

Input: $\{X\}_{\mathfrak{B}}, \{X\}_{\mathfrak{B}'}$
Output: $\{S\}_{\mathfrak{B}}, \{S\}_{\mathfrak{B}'}$
Precomputed: $\{-p^{-1}\}_{\mathfrak{B}}, \{p\}_{\mathfrak{B}'}, \{M^{-1}\}_{\mathfrak{B}'}$
$\{Q\}_{\mathfrak{B}} \leftarrow \{X\}_{\mathfrak{B}} * \{-p^{-1}\}_{\mathfrak{B}}$
$\{Q\}_{\mathfrak{B}'} \leftarrow BE(\{Q\}_{\mathfrak{B}}, \mathfrak{B}, \mathfrak{B}')$
$\{S\}_{\mathfrak{B}'} \leftarrow (\{X\}_{\mathfrak{B}'} + \{Q\}_{\mathfrak{B}'} * \{p\}_{\mathfrak{B}'}) * \{M^{-1}\}_{\mathfrak{B}'}$
$\{S\}_{\mathfrak{B}} \leftarrow BE(\{S\}_{\mathfrak{B}'}, \mathfrak{B}', \mathfrak{B})$

2.3 Base Extension

Let n be the cardinality of the base in RNS. In [9,10], Posch and Posch introduced a floating approach to compute the base extension function. In [6], Kawamura came to a similar result, but the base extension function introduced by Kawamura supposes that the moduli m_i are pseudo-Mersenne numbers of the form $m_i = 2^r - \mu_i$ with $0 \leq \mu_i \ll 2^r, \forall i \in [\![1, n]\!]$. The base extension function relies on the conversion from RNS representation to binary representation. From (1), we have:

$$x = \sum_{i=1}^{n} (x_i M_i^{-1} \mod m_i) M_i - kM,$$

for some k to be determined. Let $\xi_i(x_i) = x_i M_i^{-1} \mod m_i$ but we will use ξ_i to lighten notations. Then it follows:

$$\sum_{i=1}^{n} \xi_i / m_i = k + x/M$$

Since $0 \leq x/M < 1$, we have $k \leq \sum_{i=1}^{n} \xi_i/m_i < k+1$. Hence:

$$k = \left\lfloor \sum_{i=1}^{n} \xi_i / m_i \right\rfloor \tag{2}$$

Thanks to the special form of m_i and to the condition $0 \leq \mu_i \ll 2^r$, Kawamura has approximated m_i by 2^r to ease the computation. Let $\hat{k}$ be:

$$\hat{k} = \sum_{i=1}^{n} \frac{trunc_q(\xi_i)}{2^r} + \alpha \text{ where } trunc_q(\xi_i) = \left\lfloor \frac{\xi_i}{2^{(r-q)}} \right\rfloor 2^{(r-q)} \text{ and } 0 \leq \alpha < 1 \tag{3}$$

One can see that $0 \leq \xi_i - trunc_q(\xi_i) \leq 2^{(r-q)} - 1$. To evaluate the error due to the truncation approximation, Kawamura introduced some definitions that we recall here:

$$\epsilon_{m_i} = \frac{2^r - m_i}{2^r}, \quad \delta_{m_i} = \frac{\xi_i - trunc_q(\xi_i)}{m_i} \tag{4}$$

$$\epsilon = \max_{i \in [\![1,n]\!]} (\epsilon_{m_i}), \quad \delta = \max_{i \in [\![1,n]\!]} (\delta_{m_i}) \tag{5}$$

The denominator's approximations error is called ϵ_{m_i} whereas δ_{m_i} is due to the numerator's approximation. Then, Kawamura proved 2 theorems for the base extension function. The conditions of one of the theorems will help to find the μ_i's upper bound (called μ_{max}), which is the maximal range of the set from which we can select the moduli to build a base.

Theorem 1 (Kawamura [6]). *If* $0 \leq n(\epsilon + \delta) \leq \alpha < 1$ *and* $0 \leq x < (1-\alpha)M$*, then* $\hat{k} = k$ *and the base extension function extends the base without error.*

One can see from the proof of the Theorem 1 in [6] that the conditions can be relaxed in:

$$0 \leq n(\epsilon + \delta(1-\epsilon)) \leq \alpha \text{ and } 0 \leq x < (1-\alpha)M \text{ with } \alpha < 1 \tag{6}$$

This new condition will help us to estimate μ_{max}'s upper bound. To our knowledge, conditions on μ_{max} have not been clearly established. In order to ease the moduli selection, we define the conditions on μ_{max} in the next section.

3 New Bounds for the Cox-Rower Architecture

3.1 μ_i's Upper Bound for RNS Base

In the previous section, we have presented Kawamura's approximation of the factor k for the base extension. The only condition given by Kawamura is $0 \leq \mu_i \ll 2^r$. In this section, we will explore the different equations to evaluate the impact on μ_i's upper bound. From (4) and (5), we have:

$$\epsilon = \max\left(\frac{2^r - m_i}{2^r}\right) = \frac{2^r - \min(m_i)}{2^r} \text{ which leads to } \min(m_i) = 2^r(1-\epsilon)$$

On the other hand, $\forall x \in \mathbb{Z}/M\mathbb{Z}$ we have:

$$0 \leq \delta = \max\left(\frac{\xi_i - trunc_q(\xi_i)}{m_i}\right) \leq \frac{2^{(r-q)} - 1}{\min(m_i)} = \frac{2^{(r-q)} - 1}{2^r(1-\epsilon)}$$

From the new condition (6), it follows that:

$$0 \leq n\left(\epsilon + \frac{2^{(r-q)} - 1}{2^r(1-\epsilon)}(1-\epsilon)\right) \leq \alpha \text{ then } 0 \leq \epsilon \leq \frac{\alpha}{n} - \frac{2^{(r-q)} - 1}{2^r} \tag{7}$$

Now, we will evaluate Equation (7) in ϵ to find the condition on m_i since $\epsilon = \frac{2^r - \min(m_i)}{2^r} = \frac{\mu_{max}}{2^r}$. Let substitute ϵ in (7):

$$0 \leq \frac{\mu_{max}}{2^r} \leq \frac{\alpha}{n} - \frac{2^{r-q} - 1}{2^r} \Leftrightarrow 0 \leq \mu_{max} \leq 2^r\frac{\alpha}{n} - 2^{r-q} + 1 \tag{8}$$

If $q = r$, then μ_{max} is maximum and is in the range of:

$$0 \leq \mu_{max} \leq 2^r \frac{\alpha}{n} \tag{9}$$

Then, we can rewrite an equivalent condition of the Theorem 1 using only the parameters α, r, n, q and μ_{max}, which is more explicit for implementations:

Theorem 2. *If* $0 \leq \mu_{max} \leq 2^r \frac{\alpha}{n} - 2^{r-q} + 1$ *and* $0 \leq x < (1-\alpha)M$ *with* $\alpha < 1$*, then* $\hat{k} = k$ *and the base extension function extends the base without error.*

With this new formulation, we can easily build bases for the RNS Cox-Rower architecture.

3.2 Lower Bound for the parameter q of the Cox

In [6], Kawamura described a procedure to determine $n, \epsilon, \delta, \alpha$ and q for a given p. While n is easy to determine (same order of magnitude as $n \approx \log_2(p)/r$), q is determined using the approximations $\epsilon \ll 1$ and $2^{-(r-q)} \ll 1$ with Theorem 1's conditions. While those approximations are asymptotically correct, we want to determine q for any range of parameters. We give, here, a new procedure to determine correctly q from α, n, r and μ_{max}.

Once the bases are choosen using (9), from the Theorem 2's conditions, the following equation can be applied to find the parameter q:

$$q \geq \left\lceil -\log_2 \left(\frac{\alpha}{n} + 2^{-r} - \frac{\mu_{max}}{2^r} \right) \right\rceil \text{ with } \mu_{max} = \max_{\mu_i \in \{B, B'\}} (\mu_i) \tag{10}$$

This is a necessary and sufficient condition to get an exact computation. Unlike Kawamura's method [6], no assumption is made on ϵ (or equivalently on μ_{max}) and $2^{-(r-q)}$.

4 A New Cox-Rower Architecture

In the previous section, conditions on μ_{max} has been determined. In this section, we first present the algorithm and the classical ALU used to compute the reduction inside the Rower. To our knowledge, it is the only ALU used with the RNS Cox-Rower architecture[8,5,3,13,2].

Then, we introduce the new ALU proposed in this paper. This new ALU has been designed to fit on FPGAs, and we compare it with the classical ALU. Our comparison analysis uses 3 types of cells: DSP (Digital Signal Processing) blocks, LUTs (Look-Up Table) and registers (basic elements of FPGA) to compare the 2 ALUs. Multipliers are implemented inside DSP blocks on FPGA, with some additional features such as pre/post-adder/substracter. LUTs are the cell bases to implement any combinatorial logic.

Algorithm 2: Efficient Reduction Algorithm

Input: $a \leq 2^r, b \leq 2^r$ and $m_i = 2^r - \mu_i$ with $0 \leq \mu_i < \sqrt{2^r}$
Output: $z = (ab) \mod m_i$

1 $c \leftarrow ab = c_1 2^r + c_0$
2 $d \leftarrow c_1 \mu_i = d_1 2^r + d_0$
3 $e \leftarrow d_1 \mu_i$
4 $z \leftarrow (e + d_0 + c_0) \mod m_i$

4.1 Classical Rower Unit

The Cox-Rower architecture defined in [6,8,5,3,13] computes the reduction inside the Rower using Algorithm 2 when $0 \leq \mu_i < \sqrt{2^r}$.

The last addition (line 4 of Algorithm 2) gives a number up to $3 \cdot 2^r < 4m_i$. It is also possible to reduce the last addition during the computation of the multiplications, if the adder/reducer block are not the critical path of the design compared to the multipliers. Such implementation gives good results for efficient implementation and computation for $\mathbb{F}_p$/RSA and ECC [6,8,5,3,13,2]. Figure 1 presents the ALU of the Rower unit introduced by Guillermin [5].

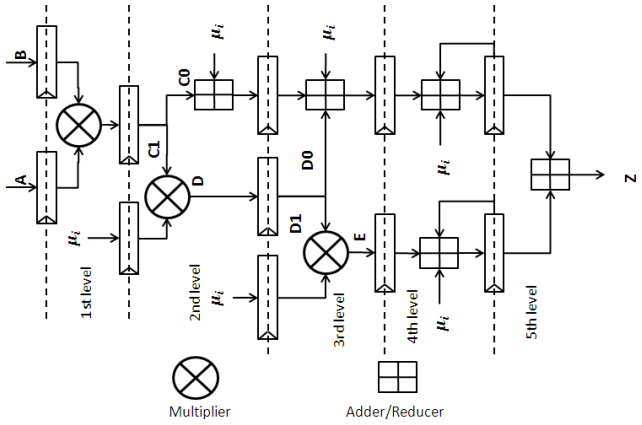

Fig. 1. Classical ALU's Rower

The first reduction stage (second level in Fig. 1) is not necessary because its output is reduced within the second stage (third level in Fig. 1) (in the design, we have $2^r + m_i < 3m_i$ but $2^r + 2^r < 3m_i$). The last part of the design is two accumulators before adding and reducing the 2 branches.

4.2 New Rower Unit

A drawback of the previous ALU is the condition $0 \leq \mu_i \leq \sqrt{2^r}$. This restriction on moduli is taken to allow efficient reduction. Notice that, on the contrary, the condition we derived in (9) has to be met to ensure a base extension without error.

Then the two following cases can be met:

(i) $0 \leq \mu_{max} \leq 2^r \frac{\alpha}{n} < \sqrt{2^r}$. In that case, choosing moduli in the range $[2^r \frac{\alpha}{n}; \sqrt{2^r}]$ may lead to erroneous computations.

(ii) $0 \leq \mu_{max} \leq \sqrt{2^r} < 2^r \frac{\alpha}{n}$. In this second case we observe that, using the classical ALU, we are restricted for the choice of moduli while our conditions (9) shows that taking more moduli without inducing errors is possible.

As an example, when $r \geq 14$ and $\log_2(p) = 521$, we are restricted by the condition $0 \leq \mu_{max} \leq \sqrt{2^r}$ to select the moduli. The condition given for efficient reduction, when r is large, is sufficient to be in (ii), which is the case in [6,8,5,3,13,2].

We propose here a new ALU for the Rower unit to exploit the upper bound $\mu_{max} \leq 2^r \frac{\alpha}{n}$ given by our condition (9). Using this upper bound, we will be able to use smaller radix than the classical ALU for computing equivalent size of p ($r = 16$ for computing $\log_2(p) = 521$ whereas we need $r = 18$ with the classical ALU). Our ALU is based on the Montgomery reduction[3] inside the Rower unit (called inner level of Montgomery). Our ALU computes the reduction using Algorithm 3 without any assumption on m_i excepted the one that m_i is coprime with 2^r to ease the computation in hardware[4].

Algorithm 3: Inner Montgomery Reduction algorithm

Input: $a \leq 2^r, b \leq m_i, m_i = 2^r - \mu_i$ with $\gcd(m_i, 2^r) = 1, m_i < 2^r$
Output: $z = (ab2^{-r}) \mod m_i$

1 $c \leftarrow ab = c_1 2^r + c_0$
2 $q \leftarrow (c_0(-m_i^{-1})) \mod 2^r = q_0$
3 $s \leftarrow (q_0 m_i) + c = s_1 2^r + s_0$
4 $z \leftarrow s_1 \mod m_i$

The most significant bits of the last addition (line 3 of Algorithm 3) gives a number up to $2m_i$ (compared to $4m_i$ with the classical ALU). Figure 2 presents the ALU of the Rower unit proposed in this paper.

Levels of multiplication and reduction are also well separated, which makes our design fully pipelinable inside DSP blocks of the FPGA. Our ALU has also one accumulator. Moreover, we can take advantage of the adder integrated in the DSP blocks to compute the last addition of the Montgomery reduction algorithm (Algorithm 3).

[3] Barrett reduction is also possible, but we would need larger multipliers for the same results.

[4] For $m_i = 2^r$ (only one even number can be selected), we use a classical multiplier and gather the r least significant bits of the multiplier

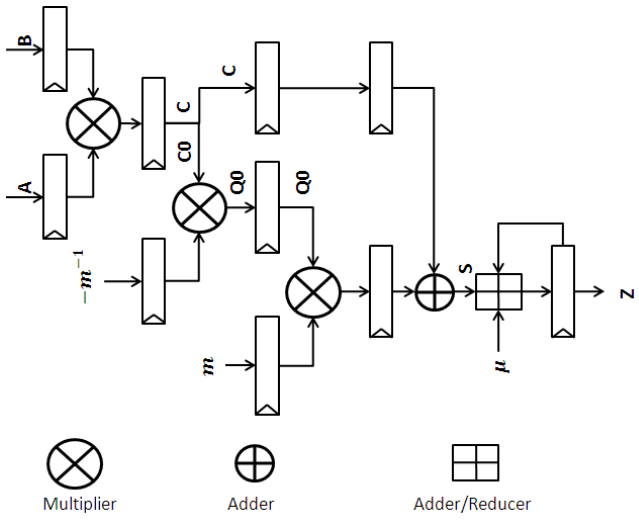

Fig. 2. New ALU's Rower

4.3 Computation Algorithm

The computation of the Montgomery reduction over RNS (called outer level of Montgomery), when using the classical ALU, is given in [5]. We recall this algorithm in the Appendix. It is based on precomputation of values depending on the parameters of the elliptic curve ($a4, a6, p$ with $y^2 = x^3 + a_4x + a_6$) and on the values of the bases ($m_i, M_i, M_i^{-1}, M, M^{-1}, m'_i, M'_i, M'^{-1}_i, M'$).

Our ALU uses the same algorithm as the one given in [5]. Differences reside in the precomputed values. Indeed, values that have to be computed are $\{X2^r\}_{\mathfrak{B}} = \{\tilde{x_i} = x_i 2^r \mod m_i\}$[5]. Mainly, we precompute the values using Montgomery representation inside the ALU (which is $\times 2^r \mod m_i$ in the inner level of Montgomery). When we use the base extension function, we need to compute the real value (inner level of Montgomery representation to normal representation $\mod m_i$) to extend it to the second base. The new ALU needs the same number of cycles in order to compute the outer Montgomery compared to the classical ALU (Algorithms for outer Montgomery computation, as well as precomputed values, for the classical ALU and our ALU are given in Appendix A).

4.4 Comparison Analysis

Despite the fact that our ALU was designed specifically to fit on FPGA, we give some comparisons for ASIC implementations.

[5] It is well known that the Montgomery representation is stable for addition and product using Algorithm 3

Area analysis. Size of the multipliers are not the same between the classical ALU and our ALU. When using the classical ALU, we need 3 multipliers of size $r \times r \to 2r, r \times r/2 \to 3r/2$ and $r/2 \times r/2 \to r$ (line 1, 2 and 3 of Algorithm 2). Our ALU costs the same number of multipliers, but the size will be $r \times r \to 2r, r \times r \to r$ and $r \times r \to 2r$. With our ALU, we fully used the full size of the DSP blocks on FPGA whereas quarter and half of the DSP blocks are lost with the classical ALU. When looking at LUTs used on FPGA, our ALU is less complex (in term of additions and reductions) than the classical ALU. This reduces the number of LUTs used within our ALU. The final adder in Montgomery reduction algorithm (Algorithm 3) can also be included inside the DSP blocks of the FPGA to help reducing the number of LUTs used, which is not the case with the classical ALU. Looking at Fig. 2, we can estimate that we would use 5 times less LUTs with our ALU than with the classical one. For ASIC, those considerations are no more true since the cost of the reduction level is far more important on FPGA than in ASIC (where multipliers are far more area consuming than adders).

Timing analysis. Timing path of a classical multiplier is an affine function on the size of its inputs. In the classical ALU, for each multiplications, we need the most significant bits of the previous multiplication (line 2 and 3 of Algorithm 2). In ASIC or FPGA, this is usually the critical path of the design if it is not well pipelined. On the other hand, our ALU only needs the least significant bits from one multiplier to the next (line 2 and 3 of Algorithm 3), which reduces the length of the critical timing path.

Others considerations. Stages of multiplications and reductions are well separated, which reduces the fanouts, placement and routing issues. Stages of multiplication are also fully pipelinable without any impact on the final reduction in our ALU.

Remarks. With the classical ALU, Kawamura's approximation on $\epsilon \ll 1$ and $2^{-(r-q)} \ll 1$ to determine q is correct when r is large enough to have $\sqrt{2^r} \ll 2^r \frac{\alpha}{n}$. With the new ALU, the procedure to determine q, defined in the previous section, is available.

5 Experiments and Comparison

5.1 Validation on FPGA

Target technology. We have implemented our ALU (and also the classical ALU [5] for the purpose of comparison) on a Xilinx Kintex-7 FPGA using the KC705 evaluation board available from Xilinx. This board includes the device xc7k325t which is a mid range FPGA on the 28nm process node.

Parameters design. We have implemented the classical cryptography security level from NIST but no restriction is given on the parameters of the elliptic curve but to be a valid curve. DSP blocks of the Xilinx 7 series family are signed multipliers of size $25 \times 18 \to 43$. Since we need only the unsigned part of the multiplier, and we want to be base-independent, we choose to take radix $r = 17$. The base has been chosen such that we can take $\alpha = 0.5$.

Implementation. For both design (classical ALU and our ALU) and each curve, Table 1 gives the area in terms of slices[6], maximum frequency after Place and Route, number of cycles for a whole computation (binary to RNS or INT2RNS, scalar multiplication or MULT, final inversion or INV, and RNS to binary transformation or RNS2INT), the computation time, q (size of the adder in the Cox module), $\log_2(\mu_{max})$ and the ratio bits.s^{-1}/slices. The slice count is independent on DSP slices or BRAM (Block RAM). Table 2, in Appendix A, gives the details account on LUTs, registers, DSP and BRAM, as well as the cycles for each command. Area implementation results take the datapath, the sequencer and the interface into accounts. Only the ALU has been modified as well as the precomputations.

Design	Curve	n	Cycles	Slices	Fmax	Latency	q	$\log_2(\mu_{max})$	Ratio
Classical ALU (C)	160	10	78892	1614	233,8	0,337 ms	5	7	293,7
Our ALU (O)				1011	285,7	0,276 ms	5	7	573,1
C	192	12	106205	1880	231,3	0,459 ms	5	7	222,4
O				1190	283,0	0,375 ms	5	7	429,8
C	224	14	137360	2249	232,5	0,590 ms	5	8	168,6
O				1358	285,0	0,481 ms	5	8	342,2
C	256	16	172520	2540	224,2	0,769 ms	5	8	130,9
O				1630	281,5	0,612 ms	5	8	256,2
O	384	23	339463	2163	281,0	1,208 ms	6	9	146,9
O	521	31	585926	2565	265,9	2,203 ms	7	10	92,2

Table 1. P&R performances and comparisons

Comparison of the 2 ALUs. Because of the condition given for an efficient reduction ($0 \leq \mu_i \leq \sqrt{2^r} = 362$) with the classical ALU, we were not able to build 2 bases with $r = 17$ for $\log_2(p) > 256$ which is a critical size for the DSP block for the Xilinx FPGA. On the other hand, using our ALU and the condition (9) ($0 \leq \mu_{max} \leq 2^r \frac{\alpha}{n} = 2114$), we were able to build 2 bases up to $\log_2(p) = 521$. To reach similar size of p, Guillermin took $r = 18$ with the classical ALU to overcome this issue [5], which it's not acceptable if we want to use 1 DSP block per multiplication and don't want to penalize the maximal frequency and latency.

As expected in the previous section, we use 35% less area, globally, with the Montgomery ALU than with the classical ALU. The area reduction given here takes into account the logic for the whole datapath, the sequencer and the interface. The area reduction inside the ALU is around 75%. The area of the 256 bits with the classical ALU is almost the same as with the 521 bits for our ALU.

The gap on the maximal frequency between the 2 ALUs is due to the placement and routing issues. Indeed, critical timing paths of the classical ALU are from multipliers to adder/reducers blocks (Fig. 1). The multiple interconnections make those paths really

[6] The slices is the cells counting system on Xilinx FPGA. A slice on a Kintex-7 includes 4 LUTs with 6 inputs and 8 registers.

difficult to place and route efficiently (essentially due to the fanouts). On the other hand, critical timing paths of our ALU is from one multiplier to the next multiplier. Thus, if we want to increase the frequency, we will have to increase the pipeline. For scalar multiplication in ECC, a pipeline of 5 registers is enough to have 95% of the pipeline used during the whole computation (Guillermin came to similar results [5]). For application to pairing computations, we can increased the pipeline to 10 registers thus expecting better frequency than for scalar multiplication[3,13].

5.2 Comparison

We compare our design with 3 others design RNS and non RNS. Our architecture supports any elliptic curve over $\mathbb{F}_p$ and implements the Montgomery Ladder algorithm to be SPA resistant. We used projective coordinates for computations. We considered the general elliptic curve in the Weierstrass form $y^2 = x^3 + a_4x + a_6$ with no assumption on the parameters. Our architecture does not make assumption on the form of the moduli except that they respect Theorem 2's conditions.

(i) First design is the one given in [5] and is based on RNS. The ALU used is the classical one. A larger size of radix has also been used in his implementation. This design shows really fast computation with any elliptic curve over $\mathbb{F}_p$. To our knowledge, it is the fastest implementation of elliptic curve scalar multiplication with generic curves independently of the choosen base on FPGA using RNS Cox-Rower architecture. For ratio comparison, a slice in recent Xilinx devices (virtex-5 and beyond) is equivalent to 3 ALMs[7] in Altera. To achieve high running frequency, all the precomputed values and the GPR are implemented into registers inside ALMs.

(ii) Second design is an implementation of a specific curve where p is a pseudo-Mersenne number [4]. Using the property of the pseudo-Mersenne value, this implementation can be specialized to run at high frequency and quickly computing the multiplication scalar.

(iii) Third design is based on fast quotient pipelining Montgomery multiplication algorithm in [7]. The scalar multiplication algorithm is based on window method algorithm. Jacobian coordinates is used and a_4 parameter is set to -3 (which is not a real restriction with Weierstrass form through an isogeny). To our knowledge, it is the fastest implementation of scalar multiplication over ECC and smallest design for such performance with generic curves.

Design	Curve	Device	Size (DSP)	Frequency	Latency	Ratio
Our work	256 any	Kintex-7	1630 slices (46)	281,5	0,612 ms	256,2
	521 any		2565 slices (91)	265,9	2,203 ms	92,2
[5]	256 any	Stratix-2	9177 ALM (96)	157,2	0,68 ms	123,1
	512 any		17017 ALM (244)	144,97	2,23 ms	40,47
[4]	256 NIST	Virtex-4	1715 slices (32)	490	0,49 ms	304,6
[7]	256 any	Virtex-4	4655 slices (37)	250	0,44 ms	250,0
		Virtex-5	1725 slices (37)	291	0,38 ms	390,5

[7] An ALM, in the Stratix-2 family, contains 2 LUTs with 5 inputs and 2 registers, and equivalent to the Xilinx Virtex-4 slice.

Design (i) is the one we compare during the paper. Our implementation is smaller and a slightly faster than the implementation in [5].

Design (ii) used the specific form of the parameter p to improve the overall performance. This design is faster than ours, but it is dependent on the pseudo-Mersenne form of the parameter p of the elliptic curve.

Design (iii) shows really fast computation of ECC scalar multiplication. Compared to our design, the gain in computation time comes from the use of Jacobian coordinates and the window method algorithm whereas we use Montgomery Ladder and projective coordinates. But when comparing the numbers of cycles to complete a multiplication and an addition/substraction, 35 cycles is needed to compute a multiplication whereas we need $2n + 3$ cycles (35 cycles for 256 bits), and 7 cycles is needed to compute an addition/substraction, whereas we need 1 cycle for an addition/substraction. Eventually, the gain in performance is not scalable to any size of elliptic curve as our work.

6 Conclusion and perspectives

In this paper, we established the link between moduli's properties and base extension for the Cox-Rower architecture. To our knowledge, that was not clearly defined yet. Now, the given bounds are more appropriate for designers. We also give a new procedure to determine q parameter which is used for truncation in the Cox module. We propose a new ALU design, based on an inner Montgomery reduction. This ALU is designed to fully use the bounds of the Cox-Rower architecture and to reduce the combinatorial area of the architecture on FPGA without penalizing performance. Moreover, using the same pipeline depth, we manage to increase the frequency of our ALU compare to the classical one.

In future works, we will increase the pipeline depth in DSP blocks for applications to pairing computations in order to improve computation time. Furthermore, we will take advantage of the pre-substracter of the DSP block to easily compute $(-AB) \mod p$ and reduce computation time. In the perspective of improving the algorithmic, we will study the use of different coordinates and implementations, such as Jacobian coordinates and window method. Although our ALU is designed for FPGA, we will also study the potential application of our ALU to ASIC.

References

1. Samuel Antão, Jean-Claude Bajard, and Leonel Sousa. Rns-based elliptic curve point multiplication for massive parallel architectures. *Comput. J.*, 55(5):629–647, May 2012.
2. Karim Bigou and Arnaud Tisserand. Improving Modular Inversion in RNS Using the Plus-Minus Method. In *CHES*, pages 233–249. Springer, 2013.
3. 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.

4. Tim Güneysu and Christof Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 2008.
5. 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.
6. 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.
7. Yuan Ma, Zongbin Liu, Wuqiong Pan, and Jiwu Jing. A high-speed elliptic curve cryptographic processor for generic curves over $\mathrm{GF}(p)$. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, Lecture Notes in Computer Science, pages 421–437. Springer Berlin Heidelberg, 2014.
8. Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shinichi Kawamura. Implementation of RSA Algorithm Based on RNS Montgomery Multiplication. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 364–376. Springer, 2001.
9. K.C. Posch and R. Posch. Base extension using a convolution sum in residue number systems. *Computing*, 50(2):93–104, 1993.
10. 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.
11. D.M. Schinianakis, A.P. Fournaris, H.E. Michail, A.P. Kakarountas, and T. Stouraitis. An rns implementation of an f_p elliptic curve point multiplier. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(6):1202–1213, June 2009.
12. Robert Szerwinski and Tim Güneysu. Exploiting the Power of GPUs for Asymmetric Cryptography. In *Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '08, pages 79–99, Berlin, Heidelberg, 2008. Springer-Verlag.
13. Gavin Xiaoxu Yao, Junfeng Fan, Ray C.C. Cheung, and Ingrid Verbauwhede. Faster Pairing Coprocessor Architecture. In *Proceedings of the 5th International Conference on Pairing-Based Cryptography*, Pairing'12, pages 160–176, Berlin, Heidelberg, 2013. Springer-Verlag.

A Algorithm to compute the Montgomery reduction over RNS and Implementation details

Let $\mathfrak{B}$ and $\mathfrak{B}'$ be 2 RNS bases such that $\mathfrak{B} = \{m_i\}$ and $\mathfrak{B}' = \{m'_i\}$ with $M = \prod_{i=1}^{n} m_i$, $M' = \prod_{i=1}^{n} m'_i$, $\gcd(p, M) = 1$ and $\gcd(M, M') = 1$. Algorithm 4 recalls the Montgomery reduction over RNS, when using the classical ALU. Precomputed values are in bald.

Algorithm 5 is the algorithm for the Montgomery reduction over RNS, when using our ALU. Operation $\otimes$ will denote the inner Montgomery multiplication and reduction (Algorithm 3) such that $a \otimes b \mod m = ab2^{-r} \mod m$.

Algorithm 4: Montgomery Reduction over RNS with classical ALU

Input: $\{X\}_{\mathfrak{B}} = \{x_i\}$ and $\{X\}_{\mathfrak{B}'} = \{x'_i\}$
Output: $\{S = (XM^{-1} \mod p) \mod M\}_{\mathfrak{B}} = \{s_i\}$ and $\{S = (XM^{-1} \mod p) \mod M'\}_{\mathfrak{B}'} = \{s'_i\}$

for $i = 1$ **to** n **do**
  $q_i \leftarrow x_i(-\mathbf{p^{-1}})\mathbf{M_i^{-1}} \mod m_i$
  $q'_i \leftarrow 0$
  $s_i \leftarrow 0$
end
$k \leftarrow 0$ // Initialization of the cox with $\alpha = 0$
for $i = 1$ **to** n **do**
  $k \leftarrow k + trunc_q(q_i)$ // Evaluating the factor k
  for $j = 1$ **to** n **do**
    $q'_j \leftarrow (q'_j + q_i\mathbf{M_i}\mathbf{p}\mathbf{M^{-1}}\mathbf{M'_j}^{-1}) \mod m'_j$
  end
end
for $i = 1$ **to** n **do**
  $q'_i \leftarrow (q'_i + \lfloor \frac{k}{2^r} \rfloor(-\mathbf{M})\mathbf{p}\mathbf{M^{-1}}\mathbf{M'_i}^{-1}) \mod m'_i$
end
for $i = 1$ **to** n **do**
  $s'_i \leftarrow (q'_i + x'_i\mathbf{M^{-1}}\mathbf{M'_i}^{-1}) \mod m'_i$
end
$k \leftarrow errinit$ // Initialization of the cox with $\alpha = errinit$
for $i = 1$ **to** n **do**
  $k \leftarrow k + trunc_q(s'_i)$ // Evaluating the factor k
  for $j = 1$ **to** n **do**
    $s_j \leftarrow (s_j + s'_i\mathbf{M'_i}) \mod m_j$
  end
end
for $i = 1$ **to** n **do**
  $s_i \leftarrow (s_i + \lfloor \frac{k}{2^r} \rfloor(-\mathbf{M'})) \mod m_i$
  $s'_i \leftarrow (s'_i\mathbf{M'_i}) \mod m'_i$
end

Algorithm 5: Montgomery Reduction over RNS with Montgomery ALU

Input: $\{\tilde{X}\}_{\mathfrak{B}} = \{\tilde{x_i} = x_i 2^r \mod m_i\}$ and $\{\tilde{X}\}_{\mathfrak{B}'} = \{\tilde{x'_i} = x'_i 2^r \mod m'_i\}$
Output: $\{\tilde{S} = (XM^{-1} \mod p)2^r \mod M\}_{\mathfrak{B}} = \{\tilde{s_i} = s_i 2^r \mod m_i\}$ and
$\{\tilde{S} = (XM^{-1} \mod p)2^r \mod M'\}_{\mathfrak{B}'} = \{\tilde{s'_i} = s'_i 2^r \mod m'_i\}$

for $i = 1$ **to** n **do**
  $q_i \leftarrow \tilde{x_i} \otimes (-\mathbf{p^{-1}})\mathbf{M_i^{-1}} \mod m_i$
  $q'_i \leftarrow 0$
  $s_i \leftarrow 0$
end
$k \leftarrow 0$ // Initialization of the cox with $\alpha = 0$
for $i = 1$ **to** n **do**
  $k \leftarrow k + trunc_q(q_i)$ // Evaluating the factor k
  for $j = 1$ **to** n **do**
    $q'_j \leftarrow (q'_j + q_i \otimes \mathbf{M_i p M^{-1} M'^{-1}_j 2^r}) \mod m'_j$
  end
end
for $i = 1$ **to** n **do**
  $q'_i \leftarrow (q'_i + \lfloor \frac{k}{2^r} \rfloor \otimes (-\mathbf{M}) \mathbf{p M^{-1} M'^{-1}_i 2^r}) \mod m'_i$
end
for $i = 1$ **to** n **do**
  $s'_i \leftarrow (q'_i + \tilde{x'_i} \otimes \mathbf{M^{-1} M'^{-1}_i}) \mod m'_i$
end
$k \leftarrow errinit$ // Initialization of the cox with $\alpha = errinit$
for $i = 1$ **to** n **do**
  $k \leftarrow k + trunc_q(s'_i)$ // Evaluating the factor k
  for $j = 1$ **to** n **do**
    $s_j \leftarrow (s_j + s'_i \otimes \mathbf{M'_i 2^{2r}}) \mod m_j$
  end
end
for $i = 1$ **to** n **do**
  $s_i \leftarrow (s_i + \lfloor \frac{k}{2^r} \rfloor \otimes (-\mathbf{M'}) \mathbf{2^{2r}}) \mod m_i$
  $s'_i \leftarrow (s'_i \otimes \mathbf{M'_i 2^{2r}}) \mod m'_i$
end

Design	Curve	LUTs	Regs	DSP	BRAM	INT2RNS	MULT	INV	RNS2INT
Classical	160	4864	2959	28	10	228	66406	11598	682
	192	5691	3497	34	12	262	89659	15446	862
	224	6688	4028	40	14	300	116227	19805	1058
	256	7482	4605	46	16	336	146144	24804	1270
Ours	160	2988	2023	28	10	228	66406	11598	682
	192	3446	2346	34	12	262	89659	15446	862
	224	3847	2696	40	14	300	116227	19805	1058
	256	4250	3532	46	16	336	146144	24804	1270
	384	5517	4962	67	23	462	289101	47810	2090
	521	7067	5882	91	31	606	500577	81437	3306

Table 2. Performances details