A New Approach to Elliptic Curve Cryptography: an RNS Architecture

Abstract—**An Elliptic Curve Point Multiplier (ECPM) is the main part of all Elliptic Curve Cryptography (ECC) systems and its performance is decisive for the performance of the overall cryptosystem. A VLSI Residue Number System (RNS) architecture of an ECPM is presented in this paper. In the proposed approach, the necessary mathematical conditions that need to be satisfied, in order to replace typical finite field circuits with RNS ones, are investigated. It is shown that such an application is feasible and that it leads to a significant improvement in the execution time of a scalar point multiplication.**

I. Introduction

ECC was first proposed in 1985 independently by N. Koblitz [4] and V. Miller [5] and is gradually gathering both scientific and industry interest. The main reason for this is that, key bit lengths in ECC are much smaller than other public-key cryptosystems. Consequently, less hardware is required to implement such public-key cryptosystems. The result is higher speed, lower power consumption and smaller certificates, which is especially useful in constrained environments such as PDAs, smart cards, etc. [6].

However, the finite field operations that need to be implemented in ECC are time consuming and the operands are large integers. Thus, since ECC is based on finite field arithmetic, there is a need for fast and efficient finite field implementations.

RNS has been widely studied and used in many applications, from digital signal processing to multiple precision arithmetic. The main advantages of RNS are the decomposition of a given range into parallel paths of smaller dynamic ranges, carry-free operations among paths employing different moduli, reduced complexity of the arithmetic units when large word lengths are needed and reduced power consumption. On the other hand, RNS implementations require the extra cost of an input converter to translate numbers from the standard binary format into residues and an output converter to implement the translation from RNS to the binary representation [6].

The performance of the converters is possible to have a major impact on the complexity, latency and power consumption. Nevertheless, in the case of data intensive computations (e.g. filters or filter banks) these disadvantages can be easily counterbalanced by the savings in the internal RNS computations. For these reasons, recently, the RNS arithmetic has been proposed for use in public-key cryptography [12], which requires multiplications of very large numbers, and in transmission systems based on Code Division Multiple Access (CDMA) [6].

The contribution of this paper is, to the authors' knowledge, the first documented ECPM architecture, which exploits the RNS representation.

In the remainder of this article, the basic mathematics of RNS and a specific version of RNS useful for our purposes, called *extended RNS*, are described in section II. In section III the main concepts of elliptic curve arithmetic are briefly analyzed. In section IV a methodology to synthesize RNS and finite field operations is presented. In section IV the hardware architecture of the proposed ECPM is described. Finally, in section V, the main results derived from the synthesis tools, as well as comparisons with other published implementations are presented. Furthermore, the impact of the binary-to-residue and residue-to-binary conversion on the performance of the ECPM is investigated. It is shown that, because of the data intensive character of Elliptic Curve (EC) arithmetic, the cost of the conversions is negligible.

II. The Residue Number System

A. *Mathematical Background*

RNS is defined by a set of relative prime integers ($p_1, p_2, \dots, p_n$) with dynamic range $P = \prod p_i$. Any integer $x \in \{0, 1, \dots, P-1\}$ has a unique representation given by

$$x \xrightarrow{RNS} \left(\langle x \rangle_{p_1}, \langle x \rangle_{p_2}, \dots, \langle x \rangle_{p_n} \right)$$

where $\langle x \rangle_{p_i} = x \bmod p_i$. Assuming two integers a, b in RNS format i.e. $a = (a_1, a_2, \dots, a_n)$ and $b = (b_1, b_2, \dots, b_n)$ then one can perform the operations $\otimes = (+, -, *)$ in parallel by

$$a \otimes b = \left(\langle a_1 \otimes b_1 \rangle_{p_1}, \langle a_2 \otimes b_2 \rangle_{p_2}, \dots, \langle a_n \otimes b_n \rangle_{p_n} \right).$$

However, if $a \otimes b \geq P$, a reduction modulo is performed in the result. Hence, to achieve an exact result, overflow must be prevented [3].

B. *The Extended RNS*

One way to perform residue-to-two's-complement conversion is through the Chinese Remainder Theorem (CRT). Let $\tilde{P}_i = P/p_i$ and the notation $\tilde{P}_i^{-1}$ will be used for the multiplicative inverse of $\tilde{P}_i$ according to modulus p_i. It can be proved that the exact value x can be expressed by

$$x = \sum_{i=1}^{n} \tilde{P}_i \left\langle \tilde{P}_i^{-1} x_i \right\rangle_{p_i} - r_x P \qquad (1)$$

where r_x is an integer and $r_x \leq n-1$ [2]. This bound on the value of r_x can be used for choosing an adequate redundant modulus. Let p_r be a redundant modulus, such that

$$p_r \geq n+1 \text{ and } \gcd(P, p_r) = 1 \tag{2}$$

and x_r be the corresponding residue. This redundant residue is available from start to finish along with the residues $x_1, x_2, \ldots, x_n$. Reducing (1) to the redundant residue we get

$$x_r = \left\langle \sum_1^n \tilde{P}_i \left\langle \tilde{P}_i^{-1} x_i \right\rangle_{p_i} - \left\langle r_x P \right\rangle_{p_r} \right\rangle_{p_r} . \tag{3}$$

We arrange it to

$$\left\langle r_x \right\rangle_{p_r} = \left\langle P^{-1} \sum_1^n \tilde{P}_i \left\langle \tilde{P}_i^{-1} x_i \right\rangle_{p_i} - x_r \right\rangle_{p_r} . \tag{4}$$

Since r_x is strictly bound from above by n ($< p_r$), it follows that $r_x = \langle r_x \rangle_{p_r}$ can be calculated from (4). All operations performed for moduli $p_1, p_2, \ldots, p_n$ are performed as well on p_r, so our number will be represented as $(x_1, x_2, \ldots, x_n, x_r)$, which forms the *extended RNS* representation. It has been proved that any number $x \in [-P+1, \ldots, P+1]$ represented by $(x_1, x_2, \ldots, x_n, x_r)$ will be correctly calculated by formulas (1) and (4) [2]. The fact that negative numbers can now be calculated correctly by equations (1) and (4) will later prove to be extremely useful in order to embed RNS in the EC arithmetic.

III. Elliptic Curve Arithmetic

A. Elliptic Curves over $\mathbb{F}_p$

In the remainder of this article we will only focus on elliptic curves defined over $\mathbb{F}_p$, where p is a "large" prime number. Field elements will be naturally represented as integers in the range 0, 1,…, $p-1$, with the usual arithmetic modulo-p. An elliptic curve over $\mathbb{F}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b \tag{5}$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0 \pmod{p}$ together with a special point $\mathcal{O}$, called the *point at infinity*. The set $E(\mathbb{F}_p)$ consists of all points (x, y), $x, y \in \mathbb{F}_p$, which satisfy equation (5), together with $\mathcal{O}$. The group law defines the addition of two points on an elliptic curve. Together with this addition operation, the set of points $E(\mathbb{F}_p)$ forms a group, with $\mathcal{O}$ serving as its identity element. It is this group that is used in the construction of elliptic curve cryptosystems. The special case of adding a point to itself is called a point doubling.

It has been proved that in *affine* representation one needs to compute the inverse of an element in $\mathbb{F}_p$ to perform an addition or a doubling of a point [1]. Inversion can be very time consuming and thus, to avoid inversions, *projective coordinates* can be used. Given a point $\mathcal{P} = (x, y)$ in affine coordinates, the projective coordinates $\mathcal{P} = (X, Y, Z)$ are given by

$$X = x; \quad Y = y; \quad Z = 1 \tag{6}$$

There are various projective representations that lead to more efficient implementations than using the one in (6). Jacobian coordinates is an example of such a representation, employed in the implementation presented here. In this case, the affine representation is given by

$$x = \frac{X}{Z^2} \quad y = \frac{Y}{Z^3} . \tag{7}$$

B. The Group Law in Jacobian Representation

Using the representation in (6) and (7), (5) becomes

$$E\left(\mathbb{F}_p\right): \quad Y^2 = X^3 + aXZ^4 + bZ^6 . \tag{8}$$

Then, addition and doubling of points can be defined as follows. Let $\mathcal{P}_0 = (X_0, Y_0, Z_0)$, $\mathcal{P}_1 = (X_1, Y_1, Z_1) \in E$. The sum $\mathcal{P}_2 = (X_2, Y_2, Z_2) = \mathcal{P}_0 + \mathcal{P}_1 \in E$ can be computed as follows. If $\mathcal{P}_0 = \mathcal{P}_1$, then

$$\mathcal{P}_2 = 2\mathcal{P}_1 = \begin{cases} X_2 = M^2 - 2S \\ Y_2 = M(S - X_2) - T \\ Z_2 = 2Y_1 Z_1 \end{cases} \tag{9}$$

where $M = 3X_1^2 + aZ_1^4$, $S = 4X_1Y_1^2$, and $T = 8Y_1^4$. On the other hand, if $\mathcal{P}_0 \neq \mathcal{P}_1$,

$$\mathcal{P}_2 = \mathcal{P}_0 + \mathcal{P}_1 = \begin{cases} X_2 = R^2 - TW^2 \\ 2Y_2 = VR - MW^3 \\ Z_2 = Z_0 Z_1 W \end{cases} \tag{10}$$

$W = X_0Z_1^2 - X_1Z_0^2$, $R = Y_0Z_1^3 - Y_1Z_0^3$, $T = X_0Z_1^2 + X_1Z_0^2$,
$M = Y_0Z_1^3 + Y_1Z_0^3$ and $V = TW^2 - 2X_2$.

Based on (9) and (10), one can implement point doubling and point addition, respectively. With these algorithms available, the next step is to implement the scalar point multiplication, which is the most important operation in ECC. For our purposes, the binary method algorithm [1] was chosen because it is easy to implement and minimizes memory requirements. The binary method algorithm is based on the binary expansion of k, as follows:

Algorithm 1 Binary method for EC point multiplication

```
INPUT: A point $\mathcal{P}$, an $l$-bit integer $k$, $k = \sum k_j 2^j$, $k_j \in \{0, 1\}$, $j = 0, 1, \ldots, l-1$

OUTPUT: $\mathcal{Q} = [k]\mathcal{P}$.
1. $\mathcal{Q} \longleftarrow \mathcal{O}$
2. for $j = l-1$ to 0 by $-1$ do:
3.     $\mathcal{Q} \longleftarrow [2]\mathcal{Q}$
4.     if $k_j = 1$ then $\mathcal{Q} \longleftarrow \mathcal{Q} + \mathcal{P}$
5. return $\mathcal{Q}$
```

The binary method requires l–1 point doublings and W–1 point additions, where l is the length and W the Hamming weight of the binary expansion of k [1]. For any positive integer k, the notation $[k]$ is used to denote the *multiplication-by-k* map from the curve to itself. The notation $[k]$ is extended to $k \leq 0$ by defining $[0]\mathcal{P} = \mathcal{O}$, and $[-k]\mathcal{P} = -([k]\mathcal{P})$ [1].

IV. Embedding RNS in the Point Addition and Doubling Algorithms

All operations in equations (9) and (10) are modulo-p, where p is the characteristic of the field. Thus, if we want to implement a cryptographic scheme with an n–bit key, all operands and finite field circuitry will be n–bit long. Consequently, the main idea proposed here is to replace finite field circuits with RNS ones. In that way, the benefits of using smaller RNS operands are exploited to implement a faster ECPM architecture.

In order for this replacement to be valid, we need to choose an adequate RNS dynamic range. First of all, we assume that equations (9) and (10) are computed over the integers, i.e., without the modulo-p reduction. Let m be the absolute maximum value of these computations. In order for the RNS implementation to be valid, we need to choose an RNS dynamic range $P \geq m$. Then, we can perform the point multiplication using RNS circuits and data in RNS format. For example, if the field characteristic p is 160-bit long, then the equivalent RNS range needs to be 660-bit. In the proposed implementation the moduli set chosen, consists of twenty moduli, 33-bit long each. It is interesting though that for a field characteristic 192-bit long, each moduli needs to be just 42-bit long. If a point multiplication result needs to be transformed back to a finite field element, we only need to implement equation (1) to get a valid number over the integers and then perform a final modulo-p reduction to the result to get the finite field element. Note that the use of *extended RNS* allows the correct computation of the EC algorithms for point addition and doubling, by providing an adequate range, in case a negative result occurs.

V. The Hardware Implementation

The first step to the proposed architecture was to design modulo-p circuits in order to synthesize RNS adders, subtracters, and multipliers. Modulo adders and subtracters are typical architectures i.e., an addition or a subtraction is performed and then a reduction-by-p stage follows. The implemented algorithms for modular addition and subtraction are as follows [9].

Algorithm 2 Modular addition and subtraction

```
Require: $p$, $0 \leq X < p$, $0 \leq Y < p$
Ensure: $Z = A + B \bmod p$
1: $Z' = X + Y$
2: $Z'' = Z' - p$
3: if $Z'' < 0$ then
4:     $Z = Z'$
5: else
6:     $Z = Z''$
7: end if
```

```
Require: $p$, $0 \leq X < p$, $0 \leq Y < p$
Ensure: $Z = A - B \bmod p$
1: $Z' = X - Y$
2: $Z'' = Z' + p$
3: if $Z' < 0$ then
4:     $Z = Z''$
5: else
6:     $Z = Z'$
7: end if
```

Modulo multiplication is implemented using Horner's rule shown in (11) [7]

$$\langle XY \rangle_p = \langle (\ldots((x_{r-1}Y) \cdot 2 + x_{r-2}Y) \cdot 2 + \ldots) \cdot 2 + x_0 Y \rangle_p \qquad (11)$$

where r is the number of digits of X. The algorithm for modular multiplication has as follows.

Algorithm 3 Modular multiplication based on Horner's rule

```
Require: $0 \leq X, Y \in \mathbb{N}$ and $r, p \in \mathbb{N}^*$
Ensure: $Z[0] = \langle XY \rangle_p$
1: $Z[r] = \longleftarrow 0$;
2: for $i = 0$ to $r$ do
3:     $Z[r-i] \longleftarrow \langle 2Z[r-i+1] + x_{r-i}Y \rangle_p$
4: end for
```

The designed circuits are shown in Fig. 1, 2, and 3.

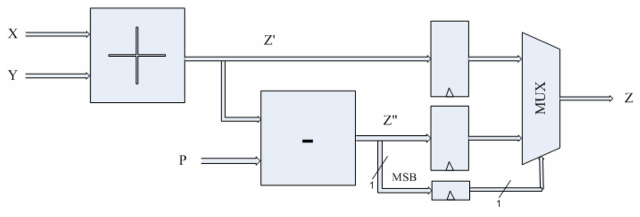

Figure 1. The modulo–p adder.

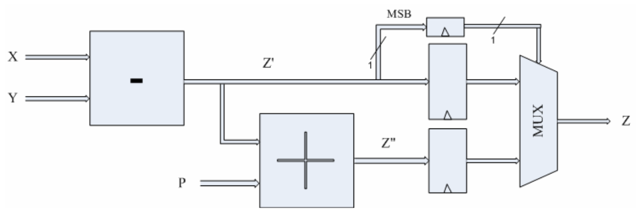

Figure 2. The modulo–p subtracter.

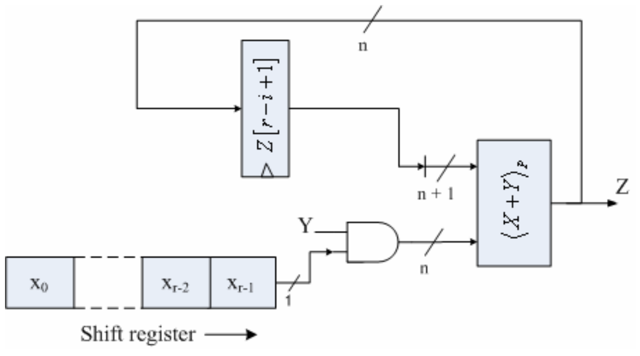

Figure 3. The modulo–p multiplier.

The RNS circuits are a parallel combination of the circuits of Fig.1, 2, 3. Their architecture is shown in Fig.4, where each block represents one of the circuits of Fig.1, 2, 3.

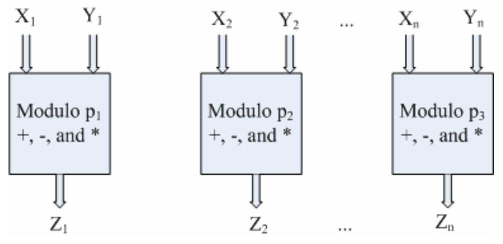

Figure 4. Architecture of the RNS structures.

With the use of the RNS structures presented, the *point addition* and *doubling* algorithms can now be implemented. These structures use 1 RNS adder, 1 RNS subtracter, 1 RNS multiplier, multiplexers and decoders. A multiplexer is placed at the inputs of each RNS structure, in order to choose the operands that will be processed. Correspondingly, a decoder is placed at the output of each RNS structure, which drives the result to an appropriate register. The multiplexers' and decoders' control signals are derived from a counter.

The proposed *Elliptic Curve Point Adder* (ECA) and *Elliptic Curve Point* "*Doubler*" (ECD) architectures are shown in Fig.5 and 6. The notation "to_X" means that data are driven to register "X" whereas "from_X" means that data are read from register "X". Note that the registers shown on Fig.5, 6 are RNS registers i.e., each block stores data in RNS format. For a point addition, 17 RNS multiplications are needed whereas for a point doubling, 15 RNS multiplications are performed.

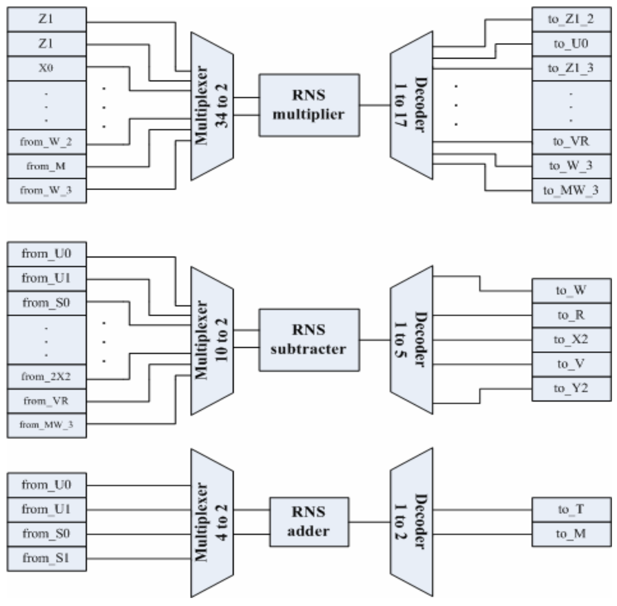

Figure 5. The proposed architecture of the ECA.

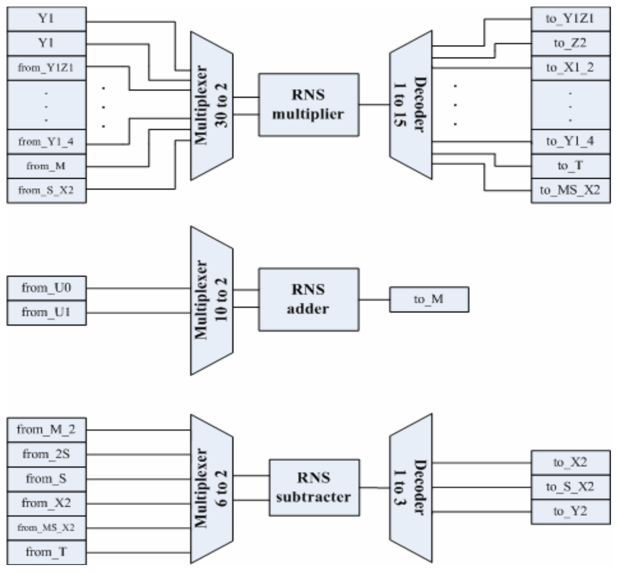

Figure 6. The proposed architecture of the ECD.

By exploiting the binary method we present the proposed architecture of the ECPM in Fig.7.

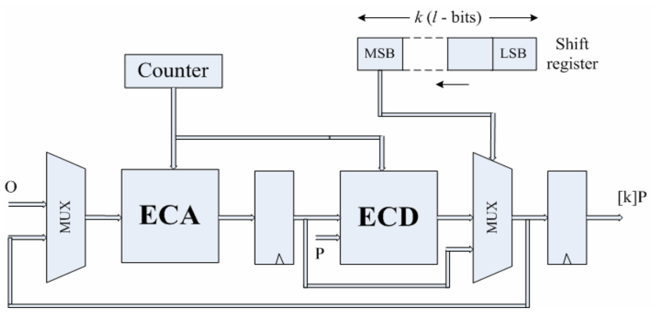

Figure 7. The proposed architecture of the ECPM.

In each clock cycle a point addition or a point doubling is performed, according to the bit value of *k*. The shift register also generates the control signal of the input multiplexer. When a complete shift of *k* has been detected, the multiplexer drives point $\mathcal{O}$ at the input of the ECD. This operation refers to the first line of the binary method algorithm and is necessary for the initialization of the point multiplication procedure.

VI. PERFORMANCE AND COMPARISONS

ECPM was synthesized in a Xilinx Virtex 2–Pro (XCVP125, FF1696) FPGA. Table 1 summarizes the main results derived from synthesis as well as comparisons with a well–known design in $\mathbb{F}_p$.

TABLE I. DELAY TIMES OF THE MAIN OPERATIONS IN ECPM

Operation	**Delay (ms)**	
	$\mathbb{F}_p$ [9]	$\mathbb{F}_p$ *with RNS representation*
Multiplication	0.0050	0.00055
EC addition	0.0740	0.0110
EC doubling	0.070	0.0096
General Point Multiplication	14.140	2.416

The above table's analysis shows the efficiency of the proposed implementation. An approximate 82% improvement in the execution time of a scalar point multiplication was achieved. Note that the implemented ECPM utilizes a 160-bit integer *k*, i.e., the underlying field characteristic *p*, is 160-bit long and the selected RNS base consists of twenty relative prime moduli, 33-bit long each. The implementation in [9] also features a 160-bit *k* and was synthesized in a Xilinx Virtex-E (V1000E-BG-560-8) FPGA. Furthermore, the reported frequency was 75 MHz, whereas in [9] the maximum clock frequency was 91.308 MHz. The only existing previous work done on a FPGA, apart from [9], is from Orlando and Paar [10]. They reported that their processor would compute a point multiplication in about 3msec but only if were possible to extract 100% throughput from their multiplier. The reported frequency in [10] was 40 MHz.

As far as the area is concerned, the proposed implementation utilizes approximately 50,000 4 input LUTs, whereas in [9] and [10] only 11,227 LUTs and 11,416 LUTs are used, respectively. This can be explained by the fact that, the RNS range that needs to be chosen is much larger than the field characteristic *p*, as proved in section IV. This is the main reason why twenty moduli were chosen to construct the adequate RNS range mentioned.

Furthermore, the impact of the binary-to-residue and residue-to-binary conversion was investigated. As far as the binary-to-residue conversion is concerned, it was found that the overall performance of the ECPM is barely affected, even under the worst-case assumption that modulo-p multipliers are used to implement the conversion, i.e. $x_{p_i} = \langle x \rangle_{p_i} = \langle x \cdot 1 \rangle_{p_i}$. In addition, following the conversion, input data participate in a large number of operations (17 RNS multiplications for every point addition and 15 RNS multiplications for every point doubling) and thus the cost of the conversion becomes negligible with respect to the overall processing.

Moreover, assuming a *Compressed Multiply Accumulate Converter* (CMAC) [8], it was estimated that at most 11 μs are needed to perform a residue-to-binary conversion, assuming a similar technology to [8]. Considering that approximately 5 ms are needed to perform a point multiplication, the cost of the conversion is negligible.

VII. Conclusion

In this work, an RNS implementation of an ECPM is proposed. Following the selection of an adequate RNS range the proposed architecture for point addition, point doubling and scalar point multiplication was described. We presented the main results derived from the synthesis tool, proving the efficiency of the proposed implementation towards other published works, in terms of delay.

Current work is being conducted on reducing the area of an ASIC version of the proposed implementation, which will be exploiting the binary-to-residue and residue-to-binary converters as well. For example, by exploiting *two's-complement* representation, modular addition and subtraction can be realized by using the same circuit [11]. Further mathematical work can also be performed, on combining RNS and finite field arithmetic more efficiently, in order to reduce the dynamic range of RNS and consequently the area of the implementation.

References

[1] I. Blake, G. Seroussi, N. Smart, "Elliptic curves in cryptography", Cambridge University Press, 2002.

[2] I. O. Aichholzer, H. Hassler, "A fast method for modulus reduction and scaling in residue number system", Workshop on Economic Parallel Processing (EPP '93), 40 – 56 .

[3] F. J Taylor, "Residue arithmetic: a tutorial with examples", IEEE COMPUTER 1984.

[4] N. Koblitz, "Elliptic curve cryptosystems", Math. Comp., 48, 203 – 209, 1987.

[5] V. Miller, "Use of elliptic curves in cryptography", In advances in Cryptology, CRYPTO – '85, Springer LNCS 218, 47 – 426, 1986.

[6] G.C. Cardarilli, A. Del Re, R. Lojacono, M. Re, "RNS Implementation of High Performance Filters for Satellite Demultiplexing", 2003 IEEE Aerospace Conference, Proceedings. 2003 IEEE, Vol. 3.

[7] J.-L Beuchat and J.-M Muller, "Modulo M multiplication-addition: algorithms and FPGA implementation", IEE ELECTRONIC LETTERS, vol. 40, No. 11.

[8] T. Srikanthan, M. Bhardwaj, C. T. Clarke, "Area-time-efficient VLSI residue-to-binary converters", IEE Proc.-Comput. Digit. Tech., vol. 145, No. 3, May 1998.

[9] Siddika Berna Örs, Lejla Batina, Bart Preneel, Joos Vandewalle, "Hardware Implementation of an Elliptic Curve Processor over *GF*(*p*)", Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP '03).

[10] G. Orlando and C. Paar, "A scalable GF(*p*) Elliptic Curve Processor Architecture for Programmable Hardware", Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001).

[11] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", Oxford University Press, Inc., New York, 2000.

[12] Bajard J.-C., Imbert L., "A Full RNS Implementation of RSA", IEEE Transactions on Computers, Vol. 53, Issue 6, June 2004, 769–774