International Journal of Applied Mathematics

**Volume 26 No. 2 2013, 241-262**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)
**doi:** http://dx.doi.org/10.12732/ijam.v26i2.10

# BIT-PARALLEL COPROCESSOR FOR STANDARD ECC-$GF(2^m)$ ON FPGA

Maurício Araújo Dias[1] [§], Márcio R.A. Gouveia[2]
José Raimundo de Oliveira[3], Ignacio Bravo Muñoz[4]

[1]Department of Mathematics and Computation
School of Science and Technology
São Paulo State University (UNESP)
Roberto Simonsen street, 305
Presidente Prudente, SP, 19060-900, BRAZIL
[2]Department of Mathematics
Institute of Biosciences, Letters and Science
São Paulo State University (UNESP)
Cristvão Colombo street, 2265, Jardim Nazareth
São José do Rio Preto, SP, 15054-000, BRAZIL
[3]Department of Computer Engineering and Industrial Automation
School of Electrical and Computer Engineering
State University of Campinas (UNICAMP)
Av. Albert Einstein, 400, Cid. Universitária Zeferino
Vaz Distrito Barão Geraldo, Campinas, SP, 13083-852, BRAZIL
[4]Electronics Department
University Alcala
DO-217. Polytechnic School
Ctra. Madrid - Barcelona km. 33.6
Alcalá de Henares, Madrid, 28871, SPAIN

**Abstract:** This paper presents the design of a high-speed coprocessor for Elliptic Curve Cryptography over binary Galois Field (ECC-$GF(2^m)$). The purpose of our coprocessor is to accelerate the scalar multiplication performed over elliptic curve points represented by affine coordinates in polynomial basis. Our method consists of using elliptic curve parameters over $GF(2^{163})$ in accordance

Received: April 7, 2013

[§]Correspondence author

with international security requirements to implement a bit-parallel coprocessor on field-programmable gate-array (FPGA). Our coprocessor performs modular inversion by using a process based on the Stein's algorithm. Results are presented and compared to results of other related works. We conclude that our coprocessor is suitable for comparing with any other ECC-hardware proposal, since its speed is comparable to projective coordinate designs.

**AMS Subject Classification:** 14H52, 94A60, 97P60
**Key Words:** elliptic curves; $GF(2^m)$, cryptography, bit-parallel, coprocessor, FPGA

## 1. Introduction

Elliptic Curve Cryptography (ECC) is a well-known branch of the studies in cryptography that still remains incompletely explored, despite ECC has been studied since 1985, see [38], [31]. The earliest researches in this branch presented several ECC-software designs [26], [35], [24], [54], [55]. ECC-software designs are easier to develop than ECC-hardware designs. However, ECC-hardware designs are often faster than ECC-software designs. Thus, ECC-hardware designs came later to supply speed requirements, [15]. Nowadays, although the literature provides descriptions of a significant variety of ECC-hardware designs, the researches often consider the following issues: elliptic curve points are either represented by affine coordinates in polynomial basis or converted to projective coordinates in other bases, [36]. On one hand, affine coordinates in polynomial basis are suitable for hardware implementation and storage. Nevertheless, they require a modular inversion (or modular division), which is the most complex and, consequently, slower operation among all important operations used to perform ECC algorithms, [31]. On the other hand, projective coordinates in other bases allow replacing the slow modular inversion (or division) by a number of fast multiplications. Nevertheless, they need more temporary storage space. Therefore, accurate comparisons among ECC-hardware designs depend on finding descriptions of cryptosystems for these two elliptic curve point representations.

Our priority was to investigate ECC over binary finite fields (binary Galois Field - $GF(2^m)$ [37]). Anyway, we found a wide range of works describing ECC-hardware designs, for which elliptic curve points are represented by projective coordinates in a variety of bases, such as normal basis [40], [53], [43], [2], [3], [11], [41], optimal normal basis [40], [1], Gaussian normal basis [48], [3], [44], [13], reordered normal basis [42], redundant representation [58], [21],

[42], type II optimal normal basis [22], [56]. We also have verified that, while papers continuously and widely describe projective coordinate designs, affine coordinate designs still require more descriptions to allow performing more accurate comparisons among ECC-hardware designs. Although we found several works describing ECC-hardware designs, for which elliptic curve points are represented by affine coordinates in polynomial basis, there is still plenty of space for research in this area. For example, we did not find any work describing a coprocessor for $ECC - GF(2^m)$ based on affine coordinates in polynomial basis that presents a speed comparable to other ECC-hardware designs based on projective coordinates in other bases. In other words, our research allowed finding advantages in using affine coordinate designs and recognizing absences.

Recently published papers have showed some advantages in using affine coordinate designs. For example, researchers showed that affine coordinates provide more security than projective coordinates against side channel attacks and simple power attacks [20]. Moreover, other researchers showed that the usage of affine coordinates no longer offers significant disadvantages in comparison with projective coordinates, when the design uses an efficient modular inversion algorithm [50]. By studding other recently published papers, we recognized absences in ECC-hardware designs. For example, we found papers describing crypto-processors, such as [28], [32], [23], but none of these papers present bit-parallel designs. A bit-parallel design often allows accelerating a cryptosystem. Bit-parallel designs have been published, such as in the papers [49], [46], but both papers only show designs of finite field multipliers, instead of processors or coprocessors. We found other descriptions of processors in [7], [12]. The processor in [7] uses finite field multiplications instead of either modular inversion or division, despite the published papers describing efficient division/inversion algorithms [57], [14], [59], [50], [12]. The processor in [12] was implemented on no-reconfigurable technology instead of field-programmable gate-array (FPGA), but ECC-hardware designs are more flexible when implemented on FPGAs than on no-reconfigurable technologies.

Since in the literature the majority of previous works describes ECC-hardware designs based on elliptic curve points represented by projective coordinates in a variety of bases, many researchers use to suppose that these designs are much faster than ECC-hardware designs based on elliptic curve points represented by affine coordinates in polynomial basis. Then, these researchers discourage developing ECC-hardware designs, such as a bit-parallel coprocessor for standard ECC-$GF(2^m)$ on FPGA, for which elliptic curve points are represented by affine coordinates in polynomial basis. However, this paper proposes the design of a high-speed coprocessor for ECC-$GF(2^m)$ based on affine coordinates in polynomial basis to show that this type of ECC-hardware design presents a speed comparable to projective coordinate designs. We chose elliptic curve parameters defined in standards, such as *NIST, IEEE P1363, IPSec, WAP, eCheck, ANSI X9.62* and *ANSI X9.63* [9], since these standards are in accordance with international security requirements. Our coprocessor speeds up modular inversion by using an efficient algorithm based on the Stein's algorithm [51]. It is a bit-parallel coprocessor, for which the speed is comparable to projective coordinate designs.

The remainder of this paper is organized as follows. A background about ECC is presented in Section 2. The design and the implementation of our bit-parallel coprocessor are described in Section 3. The behavior of our coprocessor is commented in Subsection 3.1. Our results are presented in Section 4. Finally, discussion is presented in Section 5.

## 2. Elliptic Curve Cryptography Background

A binary finite field, also called binary Galois Field ($GF(2^m)$), is a set of $2^m$ elements, each one represented by $m + 1$ bits, [34]. Therefore, the finite field arithmetic operates over these elements. The method used to perform finite field operations depends on the manner that these elements will be interpreted, i.e., the method depends on the basis representation [29].

The usual representation is the polynomial basis. For the polynomial basis $\{1, t, t^2, ..., t^{m-1}\}$ of $GF(2^m)$, an element $(a_{m-1}...a_2a_1a_0)$ represents the polynomial $a_{m-1}t^{m-1} + ... + a_2t^2 + a_1t + a_0 \in GF(2^m)$, where $a_0, a_1, a_2, ..., a_{m-1} \in GF(2)$. For this basis, the finite field operations are followed by mod $p(t)$, where $p(t)$ is an irreducible polynomial.

Now, let us consider $x$ and $y$ as any pair of elements in $GF(2^m)$. When $x$ and $y$ are presented as a pair of coordinates in the form $(x, y)$, they are representing any point by affine coordinates. In other words, we say that $P$ is represented by affine coordinates whenever $P$ is represented by a pair of coordinates in the form $P = (P_x, P_y)$.

For ECC-$GF(2^m)$, the elliptic curve $E$ is the set of all solutions $(x, y)$ to the equation:

$$y^2 + xy = x^3 + ax^2 + b, \tag{1}$$

where $x, y, a$ and $b$ are elements of $GF(2^m)$, and $b$ must be nonzero, see [27], [8].

ECC is based on the difficulties in solving the Elliptic Curve Discrete Logarithm Problem (ECDLP), [52]. In other words, finding $k$, given $P$ and $Q$, where

$Q = kP$, is a computationally intractable problem for large values of $k$ in ECC, because the scalar multiplication ($Q = kP$) has an one-way solution [10].

Therefore, all algorithms based on ECC-$GF(2^m)$ compute the point $Q = kP$ on the elliptic curve $E$, where $k$ is an integer and $P$ and $Q$ are points on $E$ [39].

For example, we can compute $Q = kP$ by using the Double-and-Add algorithm, that sweeps the binary decomposition of $k$, doubling on each digit (bit) and adding on digits equal to $'1'$ (skipping the most significant digit $'1'$) [25]:

$$Q = 17P \Rightarrow Q = (10001)P \Rightarrow Q = (((2P)2)2)2 + P. \tag{2}$$

The result of each doubling or adding is a new point on $E$ that, here, will be named either $P'$ or $Q$.

Point doublings and point additions are based on the finite field arithmetic. They are composed by modular operations such as addition, multiplication, squaring and inversion, [34]. We perform these operations on coordinates of elliptic curve points. To double $P' = (P_x^{'}, P_y^{'})$ or add the points $P = (P_x, P_y)$ and $P' = (P_x^{'}, P_y^{'})$ of an elliptic curve to obtain a new point $Q = (Q_x, Q_y)$, we use a single set of equations [16], as follows:

$$S = F + ((G + P_y^{'}) * (H + P_x^{'})^{-1}) \bmod p, \tag{3}$$

$$Q_x = (S^2 + S + P_x + P_x^{'} + a) \bmod p, \tag{4}$$

$$Q_y = (S * (P_x + Q_x) + P_y + Q_x) \bmod p, \tag{5}$$

where: $a$ defines an elliptic curve and $p$ represents the irreducible polynomial; $P_x^{'}$ and $P_y^{'}$ represent the coordinates of the point that will be doubled or added; $P_x$ and $P_y$ represent the coordinates of a standardized point $P$, defined by [9]; $Q_x$ and $Q_y$ represent the coordinates of a new point $Q$; $F = P_x$, $G = 0$ and $H = 0$, for point doublings, while for point additions, we have $F = 0$, $G = P_y$ and $H = P_x$ [16].

For the Eq. (3)-(5), we also consider: if $P_x = 0$ and $P_y = 0$, then $Q = P'$; if $P_x^{'} = 0$ and $P_y^{'} = 0$, then $Q = P$; if $P = P' = 0$, then $Q = 0$, i.e., $Q$ is a point in the infinity. The point $Q$ will be in the infinity in two other conditions: if $P_x = 0$, for point doublings; if $P_y = P_y^{'}$, for point additions.

In our coprocessor, we implemented each operation of the finite field arithmetic used in Eqs. (3)-(5): addition, square, multiplication, module and modular inversion.

Addition is performed by an ordinary *xor* logic operation and is represented by the operator " + ". We implemented addition as follows: if $A = A(t), B = B(t) \in GF(2^m)$, then

$$C \equiv (A + B) \bmod p(t). \tag{6}$$

Square is represented by $A^2$ and uses a straightforward algorithm. We perform this operation by inserting a bit $'0'$ between each bit of $A$. We implemented square as follows: if

$$A(t) = a_{m-1}t^{m-1} + a_{m-2}t^{m-2} + \ldots + a_1 t + a_0, \tag{7}$$

then

$$[A(t)]^2 = a_{m-1}t^{2m-2} + a_{m-2}t^{2m-4} + \ldots + a_1 t^2 + a_0. \tag{8}$$

Multiplication is represented by " $*$ " and uses a simple algorithm based on a loop, for which each iteration performs a shift left followed by a *xor*. We implemented multiplication as follows: if $A = A(t), B = B(t) \in GF(2^m)$ is given by

$$A(t) = \sum_{i=0}^{m-1} a_i t^i = a_{m-1}t^{m-1} + a_{m-2}t^{m-2} + \ldots + a_1 t + a_0 \tag{9}$$

and

$$B(t) = \sum_{i=0}^{m-1} b_i t^i = b_{m-1}t^{m-1} + b_{m-2}t^{m-2} + \ldots + b_1 t + b_0, \tag{10}$$

then the multiplication $A * B = A(t) * B(t)$ is given by

$$C(t) = \sum_{i=0}^{m-1} c_i t^i = [A(t) \cdot B(t)] \mod p(t). \tag{11}$$

Therefore, we perform the multiplication as a sequence of additions over $GF(2)$, as follows:

Consequently, considering $A(t) = (a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$ and defining $D_i = (d^i_{m-1}, d^i_{m-2}, \ldots, d^i_1, d^i_0)$, for $i = 0, 1, \ldots, m-1$, by

$$D_i = \begin{cases} (0, 0, \ldots, 0) & \text{if} \quad b_i = 0, \\ (a_{m-1}, a_{m-2}, \ldots, a_1, a_0) & \text{if} \quad b_i = 1, \end{cases}$$

we obtain

$$A(t) * B(t) = (r_{2m-1}, r_{2m-2}, \ldots, r_j, \ldots, r_1, r_0) \tag{12}$$

with $r_j = \left( \sum_{i=0}^{j} d^i_{j-i} \right) \bmod 2$, for $j = 0, 1, \ldots, m-1$, and $r_j = \left( \sum_{i=j-(m-1)}^{m-1} d^i_{j-i} \right) \bmod 2$, for $j = m, m+1, \ldots, 2m-1$.

We used VHDL [47] to describe the aforementioned operations of the Eq. (3)-(5) for our coprocessor.

Modular inversion is the most complex and, consequently, slower operation among all operations of the finite field arithmetic, [31]. Our coprocessor performs modular inversion by using Algorithm 1. Algorithm 1 is based on the Stein's algorithm [51] and is similar to the modular division algorithm described by Wu et al. in [57]. We have chosen Algorithm 1 for modular inversion, since this algorithm presents high performance and a straightforward implementation in hardware.

**Algorithm 1.** Modular Inversion (MI)

```
Input : $A = P'x,\ B = p,\ U = 1,\ V = 0,\ DCC = 2,\ Flag = 1,$
        $slice = 2m - 1$
Output : $(P'_x)^{-1}$
1 :     while slice $> 0$
2 :        if $A_0 = 1$
3 :           if $Flag = 1$ and $DCC_0 = 0$
4 :              $(A = A + B,\ B = A,\ U = U + V,\ V = U,$
                  $Flag = 0)$
5 :           else
6 :              $(A = A + B,\ B = U + V)$
7 :              endif
8 :           endif
9 :           $(A = A/2,\ U = (U/2)\ mod\ p)$
10 :           if Flag $= 0$ and $DCC_0 = 0$
11 :              $DCC = DCC/2$
12 :           else
13 :              $(DCC = (DCC * 2),\ Flag = 1)$
14 :           endif
15 :           slice = slice - 1
16 :     endwhile
```

For Algorithm 1, $P'_x$ and $p$ are, respectively, the value that must be inverted and the irreducible polynomial. The three operations $+$, $/2$ and $*2$ represent, respectively, a *xor*, a shift right and a shift left. More details about Algorithm 1 and about the origin of the values attributed to the variables in the first line are found in [57].

Fig. 1 shows Algorithm 1 implemented as a bit-sliced circuit. In other words, the circuit for modular inversion is composed by smaller bit-width circuits, arranged side by side, to form a longer word-length circuit. It processes one bit-field or bit-slice at time. The chained circuits are able to process the full word-length required by the ECC-based software. In this work, a bit-slice is exactly equivalent to a single iteration of Algorithm 1. Whereas Algorithm 1 requires at most $2m-1$ iterations to perform a modular inversion, we need to link, serially, $2m-1$ bit-slices in the case of developing a combinatorial circuit to perform modular inversion. Therefore, the bit-sliced circuit for modular inversion is formed by $2m-1$ bit-width circuits (bit-slices), grouped side by side, to compose a circuit able to invert a $P_x^{'}$ of $2m-1$ bits. The outputs of the first bit-slice are connected to the inputs of the second bit-slice; the outputs of the second bit-slice are connected to the inputs of the third bit-slice and so on. The inputs of the bit-slice 1 start as follows: $Ain = P'x$, $Bin = p$, $Uin = 1$, $Vin = 0$, $DCCin = 2$, $Flagin = 1$, $slice = 2m-1$ [57]. The bit-slice $2m-1$ presents the modular inverse of $P_x^{'}$ in *Vout*. We implemented Algorithm 1 as a bit-sliced circuit, since it is the easiest way to implement large circuits. The circuit for modular inversion is large, since we designed this circuit to prioritize the speed to the detriment of the area, attending to the proposed high-speed requirement.

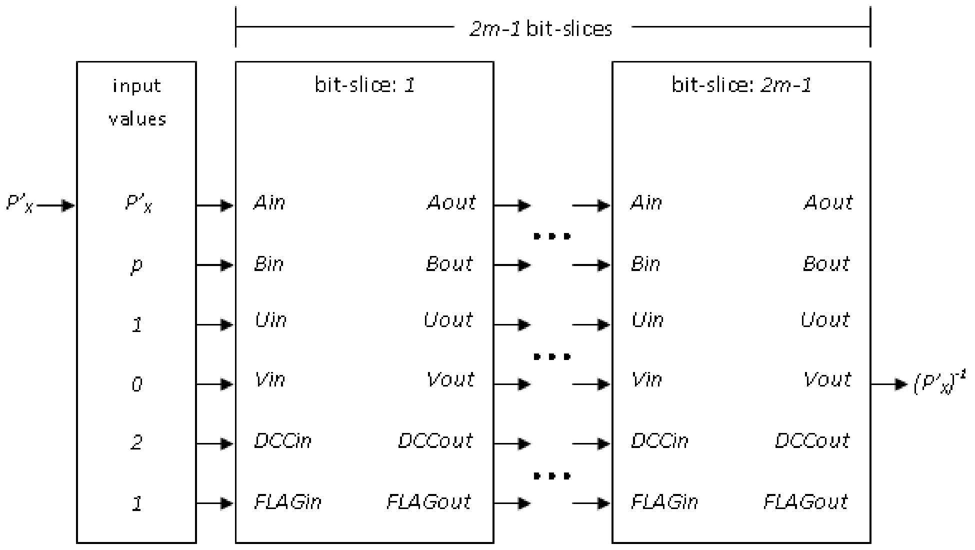


Figure 1: Bit-sliced circuit for modular inversion.

Fig. 2 shows the schematic of each bit-slice of the circuit for modular inversion. In Fig. 2, (a) represents that each bit-slice has six inputs and six outputs [57]: *Ain, Bin, Uin, Vin, DCCin, Aout, Bout, Uout, Vout* and *DCCout* (all of them are $m+1$ bits wide); $FLAGin$ and $FLAGout$ (they are both one bit wide). Moreover, in Fig. 2, (b), (c), (d), (e), (f) and (g) represent the logic correspondent to a single iteration of Algorithm 1.

## 3. Design and Implementation of the Proposed Coprocessor

Our coprocessor is a hardware unit that helps any ECC based software speeding up the computation of $Q = kP$. It computes $Q = kP$ in high speed, because we implemented the aforementioned finite field operations as digital circuits. We designed our coprocessor to be implemented on FPGAs and to be used on a PC-board adapter.

Fig. 3 presents the basic diagram of the PC-board adapter containing our coprocessor. Fig. 3 shows the on-board elements and the PC's components that communicate with the adapter. Our coprocessor is composed by independent circuits working together. These circuits were implemented on Altera's $EP2S180F1020C4$ and $EP2S90F1508C3$ FPGAs, due to high speed and density requirements [5], [6]. The former implements the modular inversion showed by Eq. (3); the latter implements the remainder of operations showed by Eq. (3)-(5), i.e., multiplication, module, square and addition. Moreover, the latter implements the Double-and-Add algorithm, a random number generator ($RNG$) [33], general purpose registers and the logic of the bus interface. Inputs $P^{'}_x$ and $P^{'}_y$, receive $P' = (P^{'}_x, P^{'}_y)$. Outputs $Q_x$ and $Q_y$ inform $Q = (Q_x, Q_y)$.

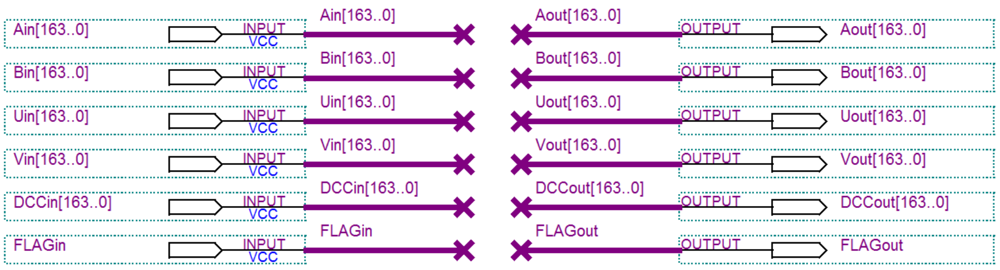
The PC-board adapter receives data through the PC data bus (PCI) that is $w$ bits wide ($w = 32$ or $w = 64$). The data is sent to the PC-board adapter, for example, by a software executing the Diffie-Hellman key-exchange model in the CPU [27].

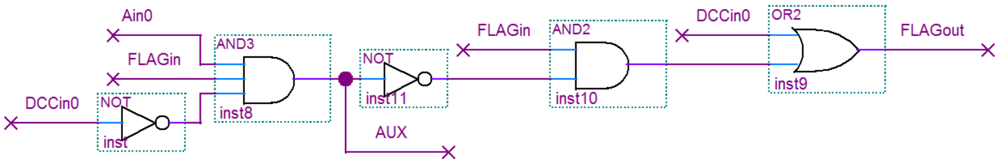
The data received by the PC-board adapter is a point in the form $P' = (P^{'}_x, P^{'}_y)$. Each data is $2(m+1)$ bits wide, i.e., $m+1$ bits by coordinate of the point $P'$, where $m$ represents the finite field.

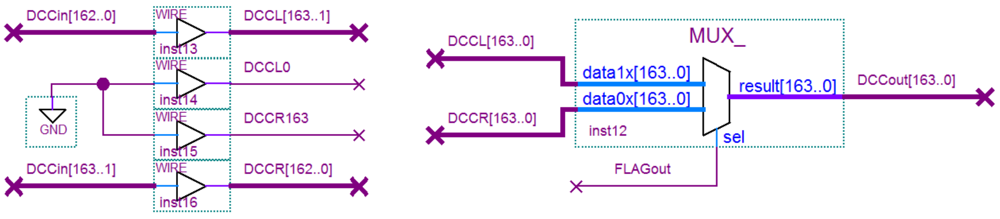
Thus, the point $P' = (P^{'}_x, P^{'}_y)$ arrives at the PC-board adapter fragmented in $2(m+1)/w$ parts, where $w$ represents the width (in bits) of the PC data bus. The point $P' = (P^{'}_x, P^{'}_y)$ is rebuilt and stored in the input register, from the less significant $w$ bits to the more significant ones.

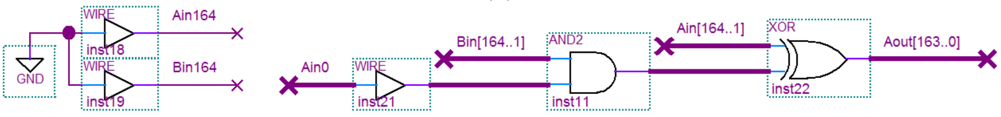
### 3.1. Behavior of the Proposed Coprocessor

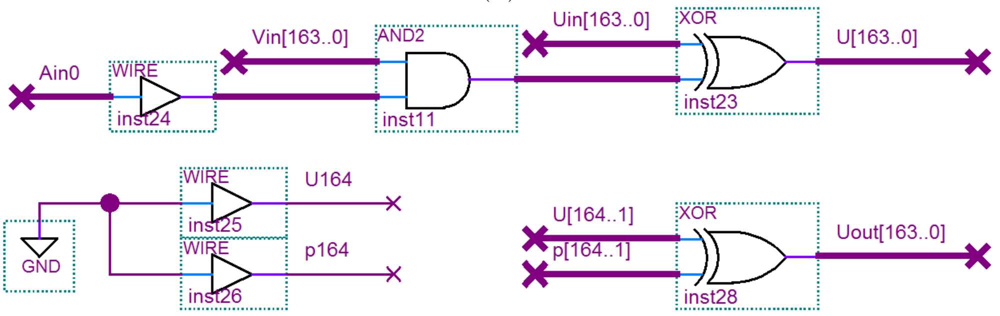
The behavior of the PC-board adapter and the flow of data through its components is presented by Fig. 4. Fig. 4 shows that the cryptographic algorithm

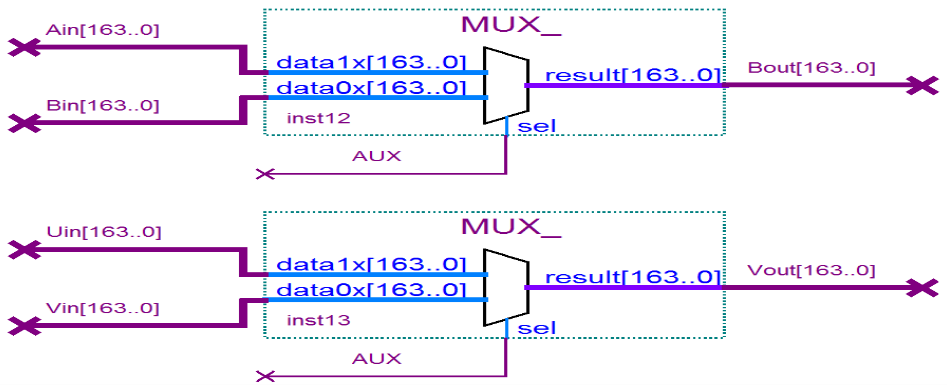

Ain[163..0]
INPUT VCC
Bin[163..0]
Uin[163..0]
Vin[163..0]
DCCin[163..0]
FLAGin
Aout[163..0]
OUTPUT
Bout[163..0]
Uout[163..0]
Vout[163..0]
DCCout[163..0]
FLAGout
(a)


Ain0
FLAGin
DCCin0
NOT
inst
AND3
inst8
NOT
inst11
AUX
FLAGin
AND2
inst10
DCCin0
OR2
inst9
FLAGout
(b)


DCCin[162..0]
WIRE
inst13
DCCL[163..1]
inst14
DCCL0
GND
inst15
DCCR163
DCCin[163..1]
inst16
DCCR[162..0]
MUX_
DCCL[163..0]
data1x[163..0]
data0x[163..0]
result[163..0]
DCCout[163..0]
DCCR[163..0]
inst12
sel
FLAGout
(c)


GND
WIRE
inst18
Ain164
inst19
Bin164
Ain0
inst21
Bin[164..1]
AND2
inst11
Ain[164..1]
XOR
inst22
Aout[163..0]
(d)


Ain0
WIRE
inst24
Vin[163..0]
AND2
inst11
Uin[163..0]
XOR
inst23
U[163..0]
GND
inst25
U164
inst26
p164
U[164..1]
p[164..1]
inst28
Uout[163..0]
(e)



(f)

Figure 2: Schematic of each bit-slice of the circuit for modular inversion.

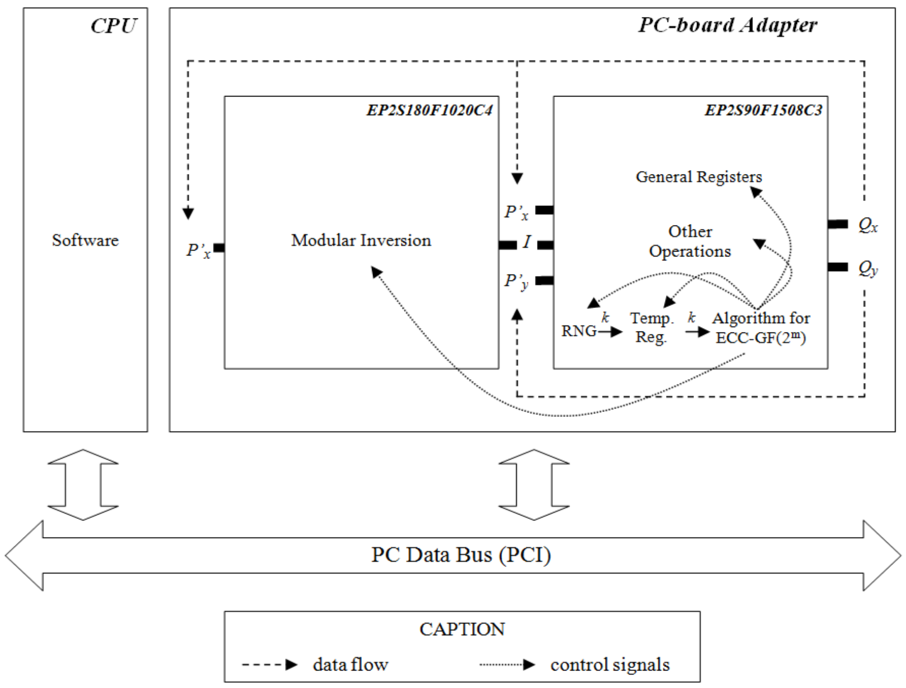


Figure 3: Basic diagram of the PC-board adapter.

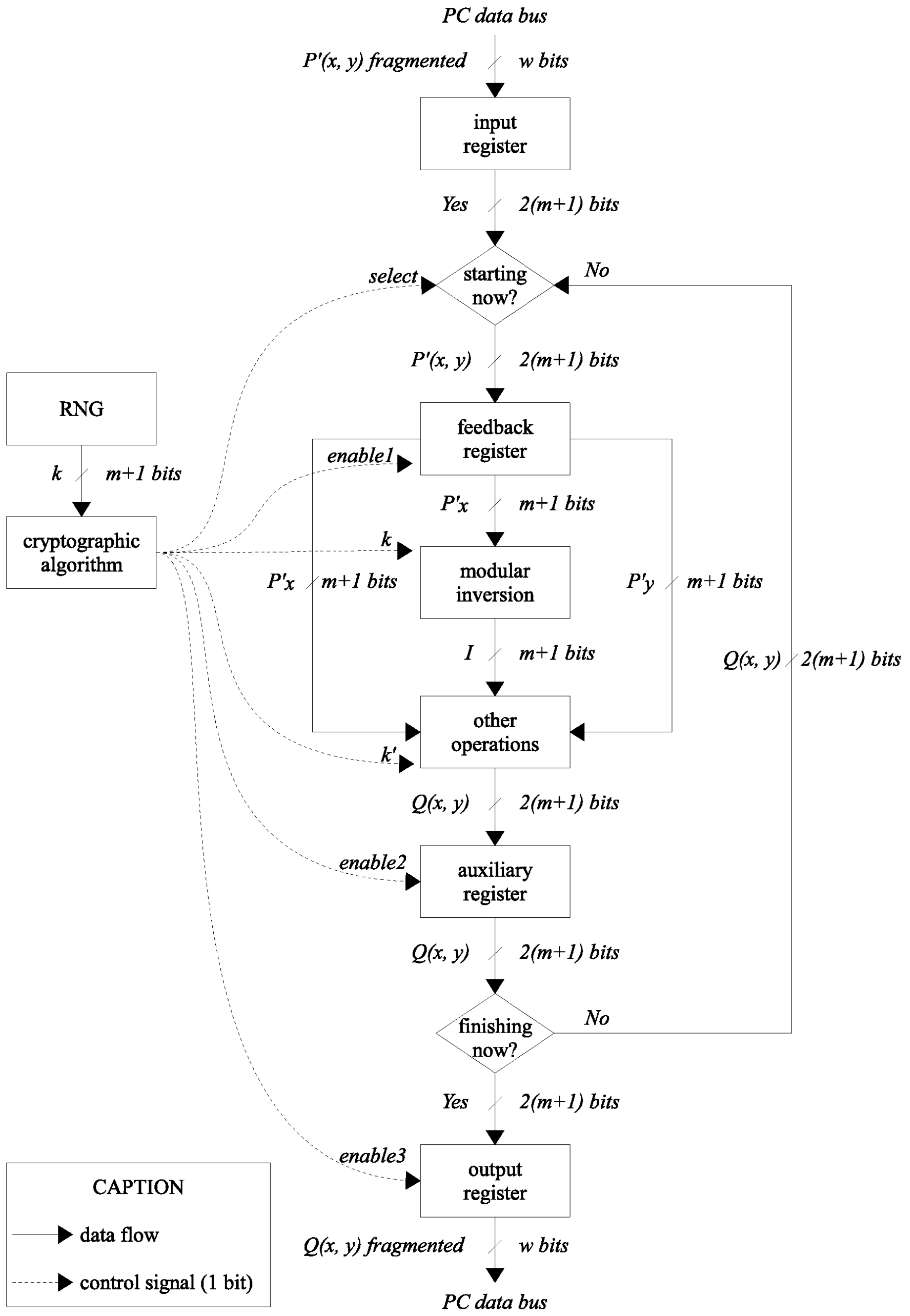


Figure 4: Data flow of the PC-board adapter presented by Fig. 3.

(Double-and-Add) generates signals to control point doublings, point additions and the flow of data. We have chosen the Double-and-Add as the cryptographic algorithm, since the implementation of the Double-and-Add in hardware is straightforward. The Double-and-Add receives the $m+1$ bits wide value $k$ from a random number generator (*RNG*). It analyzes each bit of $k$ to generate the control signals $k'$, *select* and *enable*. The Double-and-Add performs a point doubling for $k' = '0'$ and a point addition for $k' = '1'$. For *select* $= '1'$, $P' = (P^{'}_x, P^{'}_y)$ goes from the input register to the feedback register; for *select* $=$ $'0'$, the PC-board adapter performs a process of feedback. The signals *enable* allow enabling or disabling the input of data into the registers.

At the moment when the Double-and-Add starts working, the point $P' = (P^{'}_x, P^{'}_y)$ goes from the input register to the feedback register, passing through a multiplexer between these two registers. Otherwise, a partial point $Q = (Q_x, Q_y)$, goes from the auxiliary register to the feedback register, as explained later in this section. Anyway, any point in the feedback register is named $P' = (P^{'}_x, P^{'}_y)$. After stored in the feedback register, the coordinates $P^{'}_x$ and $P^{'}_y$ go to different ways.

First, the coordinate $P^{'}_x$ goes to a combinatorial circuit to be inverted. In other words, this circuit uses the coordinate $x$ of $P'$ to perform $I$, where $I = (P^{'}_x)^{-1} \bmod p$. The modular inversion is part of Eq. (3).

Next, the coordinates $x$ and $y$ of $P'$ and the modular inverse of $P^{'}_x$ (the value $I$) go to the circuit responsible to perform all the other operations present in the Eq. (3)-(5).

By passing $P' = (P^{'}_x, P^{'}_y)$ through these two circuits, the PC-board adapter calculates a point $Q = (Q_x, Q_y)$ that will be stored in an auxiliary register. This point $Q = (Q_x, Q_y)$ represents either a final or a partial result, depending on the step reached while processing the cryptographic algorithm.

When $Q = (Q_x, Q_y)$ is a partial result, it goes to the feedback register, through the multiplexer "start?". This process of feedback is repeated several times, following the logic of the Double-and-Add, while the final point $Q = (Q_x, Q_y)$ is not found. In other words, the Double-and-Add uses the value $k$ to determine the number of doublings and additions required to find $Q = kP$.

At the end of the process, the final point $Q = (Q_x, Q_y)$ goes to the output register. From this register, $Q = (Q_x, Q_y)$ is fragmented in $2(m+1)/w$ parts and goes to the PC data bus. Thus, the software executing in the CPU, finally, receives the point $Q = (Q_x, Q_y)$.

Summarizing, some software requiring to perform a scalar multiplication is executed by the CPU of a PC. This software needs a point $Q$ that is found by calculating the $Q = kP$ equation. To achieve a better performance, the software calls our coprocessor to perform the $Q = kP$ operation. When the adapter starts working, the software sends the point $P'$ to the adapter through the PC data bus. In the adapter, the integer $k$ is generated by a random number generator ($RNG$). The circuits use the integer $k$ and the point $P'$ to calculate $Q = kP'$. At the end of the process, the adapter sends the point $Q$ back to the software through the PC data bus. Aided by our coprocessor, this software obtains the point $Q = kP$ significantly fast.

## 4. Results

Since we have not found any previous work describing a bit-parallel coprocessor for standard ECC-$GF(2^m)$ on FPGA, for which elliptic curve points are represented by affine coordinates in polynomial basis, comparisons among this specific type of design with other ECC-hardware designs were unknown until this moment. Therefore, to compare the speed of our coprocessor with other hardware designs: first, we implemented a prototype of our coprocessor on Altera's $EP2S180F1020C4$ and $EP2S90F1508C3$ FPGAs that run at 250 MHz [5], [6]; second, we found the number of clock cycles required to calculate different operations in our circuit; next, we searched for the computation time required to perform $Q = kP$ for different hardware designs. Our coprocessor used the following parameters recommended by [9] to define the elliptic curve points over $GF(2^{163})$:

$$p : X^{163} + X^7 + X^6 + X^3 + 1$$

$$a : 000000000000000000000000000000000000000001 \text{ (hexa.)}$$

$$X : 3F0EBA16286A2D57EA0991168D4994637E8343E36 \text{ (hexa.)}$$

$$Y : 0D51FBC6C71A0094FA2CDD545B11C5C0C797324F1 \text{ (hexa.)}$$

where $a$ is used to define an elliptic curve, $p$ represents the irreducible polynomial, $x$ and $y$ represent the coordinates of the point $P$. These parameters represent a single set among all sets of parameters recommended by [9] for $GF(2^{163})$, which was randomly chosen for this work.

The platform used to develop and simulate the circuits of our coprocessor was the Quartus II v5.0 of Altera [4].

Table 1 presents the number of clock cycles required by our coprocessor to calculate, respectively: the modular inversion operation of Eq. (3); all the other operations of Eq. (3)-(5); either a point doubling or a point addition; the $Q = kP$ operation (to calculate $Q = kP$, our coprocessor needs to perform $m$ operations (point doublings or point additions) in average, where $m$ represents the finite field).

| **Finite Field** | **Clock Cycles for Modular Inversion** | **Clock Cycles for Other Operat. of Eq. (3)-(5)** | **Clock Cycles for Either Point Doubling or Point Addition** | **Clock Cycles for Scalar Multiplic. $Q = kP$ (in Average)** |
|---|---|---|---|---|
| $GF(2^{163})$ | 123 | 31 | 154 | $25,013$ |

Table 1: Number of clock cycles required by our coprocessor

The number of clock cycles required by our coprocessor to calculate either a point doubling or a point addition is equivalent to the sum of the values presented by Column 2 and Column 3. The $Q = kP$ is calculated multiplying the number of clock cycles required to calculate either a point doubling or a point addition by the average number of operations used to calculate $Q = kP$. For achieve this results, our coprocessor uses 329 pins and occupies an area of 216,288 ALUTs (adaptive look-up tables), 270,360 LEs (equivalent logic elements) of two FPGAs that run at 250 MHz [5], [6].

Table 2 shows the computation time required to perform $Q = kP$ for different hardware designs. These designs use elliptic curve points represented either by projective or affine coordinates.

| **Design** | **Finite Field** | **Technology** | $Q = kP$ **(ms)** |
|---|---|---|---|
| [53] | $GF(2^{155})$ | FPGA | 18.40 |
| [45] | $GF(2^{167})$ | FPGA | 0.21 |
| [17] | $GF(2^{163})$ | FPGA | 0.14 |
| [7] | $GF(2^{163})$ | FPGA | 0.80 |
| [12] | $GF(2^{163})$ | FPGA | 0.44 |
| [41] | $GF(2^{163})$ | FPGA | 0.25 |
| Our Coprocessor | $GF(2^{163})$ | FPGA | 0.10 |

Table 2: Computation time required to perform $Q = kP$

Note that our coprocessor presents the lowest computation time required to perform $Q = kP$ among all ECC-hardware designs presented in Table 2. Hence, the results show that it is also possible to develop a high speed coprocessor without using the conversion to projective coordinates in normal basis or other basis. Moreover, the results show that projective coordinate designs are no longer faster than affine coordinate designs.

## 5. Discussion

In this paper, we have presented a high-speed coprocessor for ECC-$GF(2^m)$, for which we represent elliptic curve points by affine coordinates in polynomial basis. When our coprocessor is compared to other hardware designs [53], [45], [17], [7], [12], [41], including those for which elliptic curve points are represented by projective coordinates in other bases, the results show that our coprocessor performs the scalar multiplication ($Q = kP$) significantly fast. By using our coprocessor to accelerate the scalar multiplication performed over elliptic curve points represented by affine coordinates in polynomial basis, we show that it is also possible to develop a high-speed coprocessor without using the conversion to projective coordinates in normal basis or other basis. Therefore, this paper shows that projective coordinate designs are no longer better than affine coordinate designs. Some recently published papers present analogous opinion [19], [12]. Since we have not found any other paper describing a bit-parallel coprocessor for standard ECC-$GF(2^m)$ on FPGA, for which elliptic curve points are represented by affine coordinates in polynomial basis, our main contribution is to offer a paper that allows comparing this specific type of design with other ECC-hardware designs.

### 5.1. Conclusions

We presented the design and implementation of a coprocessor that reaches a speed comparable to projective coordinate designs [53], [45], [41]. Our coprocessor significantly accelerates the scalar multiplication performed over elliptic curve points represented by affine coordinates in polynomial basis. Since the majority of previous works describing ECC-hardware designs are based on elliptic curve points represented by projective coordinates in a variety of bases, researchers use to suppose that these designs are much faster than ECC-hardware designs based on elliptic curve points represented by affine coordinates in polynomial basis. However, our results show that projective coordinate designs are no longer faster than affine coordinate designs. Therefore, our coprocessor is suitable for comparing with any other ECC-hardware design. The obvious drawback of our design is the large area of our circuits. Anyway, this large area no longer offers significant limitations for our design, since mobile computing was apart from the goal of this project. For mobile computing, we certainly will prefer a projective coordinate design.

## Acknowledgments

Special thanks for collaborations between Spain and Brazil (UAH, Spain; UNICAMP, UNESP, CAPES and FUNDUNESP, Brazil).

## References

[1] G.B. Agnew, R.C. Mullin and S.A. Vanstone, An implementation of elliptic curve cryptosystems over $F(2^{155})$, *IEEE Journal on Selected Areas in Communications*, **11**, No 5 (1993), 804-813.

[2] O. Al-Khaleel, C. Papachristou, F. Wolff and K. Pekmestzi, An elliptic curve cryptosystem design based on FPGA pipeline folding, In: *13th IEEE International On-Line Testing Symposium* (IOLTS'07), IEEE, Hersonissos-Heraklion (2007), 71-78.

[3] O. Al-Khaleel, C. Papachristou, F. Wolff and K. Pekmestzi, FPGA-based design of a large moduli multiplier for public-key cryptographic systems, In: *24th IEEE International Conference on Computer Design* (ICCD-2006), IEEE, San Jose (2006), 314-319.

[4] Altera, *Quartus II, Programmable Logic Development System and Software*, Data Sheet, Altera Corporation, San Jose (1999).

[5] Altera, *Stratix II Device Handbook, Volume 1*, Data Sheet, ver. 4.5, Altera Corporation, San Jose (2011).

[6] Altera, *Stratix II Device Handbook, Volume 2*, Data Sheet, ver. 4.5, Altera Corporation, San Jose (2009).

[7] M. Benaissa and W.M. Lim, Design of flexible $GF(2^m)$ elliptic curve cryptography processors, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **14**, No 6 (2006), 659-662.

[8] I. Blake, G. Seroussi and S. Nigel, *Elliptic Curves in Cryptography*, Cambridge University Press, Cambridge (1999).

[9] Certicom Research, *SEC 2: Recommended Elliptic Curve Domain Parameters*, Standards for Efficient Cryptography (SEC), Version 1.0, Certicom Corporation, Mississauga (2000).

[10] Certicom Research, *SEC 1: Elliptic Curve Cryptography*, Standards for Efficient Cryptography (SEC), Version 1.0, Certicom Corporation, Mississauga (2000).

[11] W.N. Chelton and M. Benaissa, Fast elliptic curve cryptography on FPGA, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **16**, No. 2 (2008), 198-205.

[12] J.H. Chen, M.D. Shieh, W.C. Lin, A high-performance unified-field reconfigurable cryptographic processor, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **18**, No 8 (2010), 1145-1158.

[13] H.M. Choi, C.P. Hong and C.H. Kim, High performance elliptic curve cryptographic processor over $GF(2^{163})$, In: *4th IEEE International Symposium on Electronic Design, Test and Applications* (DELTA 2008), IEEE, Kowloon (2008), 290-295.

[14] A.K. Daneshbeh and M.A. Hasan, A class of unidirectional bit serial systolic architectures for multiplicative inversion and division over, *IEEE Transactions on Computers*, **54**, No 3 (2005), 370-380.

[15] G.M. de Dormale and J.J. Quisquater, High-speed hardware implementations of elliptic curve cryptography: a survey. *Journal of Systems Architecture*, **53**, No 2-3 (2007), 72-84.

[16] M.A. Dias and J.R. Oliveira, An inverter architecture for ECC-$GF(2^m)$ based on the Stein's algorithm, In: *3rd International Workshop on Boolean Functions Cryptography and Applications* (BFCA'07), University Paris, Paris (2007), 119-133.

[17] H. Eberle, N. Gura and S.C. Shantz, A cryptographic processor for arbitrary elliptic curves over $GF(2^m)$, In: *14th IEEE International Conference on Application-Specific Systems, Architectures and Processors* (ASAP 2003), IEEE, The Hague (2003), 444-454.

[18] M.A. Fayed, M.W. El-Kharashi and F. Gebali, A high-speed, high-radix, processor array architecture for real-time elliptic curve cryptography over $GF(2^m)$, In: *7th IEEE International Symposium on Signal Processing and Information Technology* (ISSPIT 2007), IEEE, Cairo (2007), 56-61.

[19] K. Fong, D. Hankerson, J. Lpez, and A. Menezes, Field inversion and point halving revisited, *IEEE Transactions on Computers*, **53**, No 8 (2004), 1047-1059.

[20] A.P. Fournaris and O. Koufopavlou, Low area elliptic curve arithmetic unit, In: *2009 IEEE International Symposium on Circuits and Systems* (ISCAS 2009), Taipei (2009), 1397-1400.

[21] S. Gao, J. von zur Gathen and D. Panario, Gauss periods: orders and cryptographical applications, *Mathematics of Computation*, **67** (1998), 343-352.

[22] S. Gao and S.A. Vanstone, On orders of optimal normal basis generators, *Mathematics of Computation*, **64**, (1995), 1227-1233.

[23] J. Goodman and A.P. Chandrakasan, An energy-efficient reconfigurable public-key cryptography processor, *IEEE Journal of Solid-State Circuits*, **36**, No 11 (2001), 1808-1820.

[24] D.M. Gordon, A survey of fast exponentiation methods, *Journal of Algorithms*, **27** (1998), 129-146.

[25] P. Guillot and O. Orcire, *Speeding up Elliptic Curve Computations Using Addition-Subtraction Chains and Horner Rule*, Thomson-CSF Communications, Paris (1998).

[26] D. Hankerson, J.L. Hernandez and A. Menezes, Software implementation of elliptic curve cryptography over binary fields, *Lecture Notes in Computer Science*, **1965** (2000), 243-267.

[27] D. Hankerson, A. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Professional Computing, Springer-Verlag, New York (2004).

[28] C. Huang, J. Lai, J. Ren and Q. Zhang, Scalable elliptic curve encryption processor for portable application, In: *5th IEEE International Conference on ASIC* (ASICON 2003), IEEE, Beijing (2003), 1312-1316.

[29] IEEE P1363/D13, *Standard Specification for Public Key Cryptography*, IEEE, Inc., New Jersey (1999).

[30] K. Jrvinen and J. Skytt, On parallelization of high-speed processors for elliptic curve cryptography, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **16**, No 9 (2008), 1162-1175.

[31] N. Koblitz, Elliptic curve cryptosystems, *Mathematics of Computation*, **48** (1987), 203-209.

[32] J. Lee, W. Kwon, S. Lee and C. Lee, Design of a programmable cryptoprocessor for multiple cryptosystems, In: *17th IEEE International SOC Conference* (SOCC 2004), IEEE, Santa Clara (2004), 157-158.

[33] L. Lee and K. Wong, A random number generator based on elliptic curve operations, *Computers and Mathematics with Applications*, **47**, No 2-3 (2004), 217-226.

[34] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their Applications*, revised edition, Cambridge University Press, Cambridge (1994).

[35] J. Lpez andR. Dahab, *An Overview of Elliptic Curve*, technical report, UNICAMP, Campinas (2000).

[36] J. Lpez and R. Dahab, Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation, *Lecture Notes in Computer Science*, **1717** (1999), 316-327.

[37] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, Norwell (1993).

[38] V.S. Miller, Use of elliptic curves in cryptography, *Lecture Notes in Computer Science*, **218** (1986), 417-426.

[39] F. Morain and J. Olivos, Speeding up the Computations on an Elliptic Curve Using Addition-Subtraction Chains, *Theoretical Informatics and Applications*, **24**, No 6 (1990), 531-544.

[40] R. Mullin, I. Onyszchuk, S. Vanstone and R. Wilson, Optimal normal bases in GF($p^n$), *Discrete Applied Mathematics*, **22** (1988), 149-161.

[41] B. MuthuKumar and S. Jeevananthan, High speed hardware implementation of an elliptic curve cryptography (ECC) co-processor, In: *2nd IEEE International Conference on Trends in Information Sciences and Computing* (TISC-2010), IEEE, Chennai-Tamil Nadu (2010), 176-180.

[42] A.H. Namin, H. Wu and M. Ahmadi, Comb architectures for finite field multiplication in $F2^m$, *IEEE Transactions on Computers*, **56**, No 7 (2007), 909-916.

[43] A.H. Namin, H. Wu and M. Ahmadi, A word-level finite field multiplier using normal basis, *IEEE Transactions on Computers*, **60**, No 6 (2011), 890-895.

[44] V. Trujillo-Olaya, J. Velasco-Medina and J.C. Lopez-Hernandez, Efficient hardware implementations for the gaussian normal basis multiplication over $GF(2^{163})$, In: *3rd Southern Conference on Programmable Logic* (SPL2007), IEEE, Mar del Plata (2007), 45-50.

[45] G. Orlando and C. Paar, A high-performance reconfigurable elliptic curve processor for $GF(2^m)$, *Lecture Notes in Computer Science*, **1965** (2000), 41-56.

[46] S. Park, K. Chang and D. Hong, Efficient bit-parallel multiplier for irreducible pentanomials using a shifted polynomial basis, *IEEE Transactions on Computers*, **55**, No 9 (2006), 1211-1215.

[47] D.L. Perry, *VHDL: Programming by Example*, 4th ed., McGraw-Hill, New York (2002).

[48] A.K. Rahuman and G. Athisha, Reconfigurable architecture for elliptic curve cryptography, In: *2010 International Conference on Communication and Computational Intelligence* (INCOCCI 2010), IEEE, Erode (2010), 461-466.

[49] A. Reyhani-Masoleh and M.A. Hasan, Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$, *IEEE Transactions on Computers*, **53**, No 8 (2004), 945-959.

[50] E. Savas, M. Naseer, A.A.A. Gutub and Ç.K. Koç, Efficient unified Montgomery inversion with multibit shifting, *IEEE Proceedings - Computers and Digital Techniques*, **152**, No 4 (2005), 489-498.

[51] J. Stein, Computational problems associated with Racah algebra, *J. Computational Physics*, **1** (1967), 397-405.

[52] P.-C. Su, H.K.-C. Chang and E.-H. Lu, ID-based threshold digital signature schemes on the elliptic curve discrete logarithm problem, *Applied Mathematics and Computation*, **164**, No 3 (2005), 757-772.

[53] S. Sutikno, R. Effendi and A. Surya, Design and implementation of arithmetic processor $F(2^{155})$ for elliptic curve cryptosystems, In: *1998 IEEE Asia-Pacific Conference on Circuits and Systems* (APCCAS 1998), IEEE, Chiangmai (1998), 647-650.

[54] W.-J. Tsaur, Several security schemes constructed using ECC-based self-certified public key cryptosystems, *Applied Mathematics and Computation*, **168**, No 1 (2005), 447-464.

[55] W.-J. Tsaur and C.-H. Chou, Efficient algorithms for speeding up the computations of elliptic curve cryptosystems, *Applied Mathematics and Computation*, **168**, No 2 (2005), 1045-1064.

[56] Y.-B. Wang, X.-J. Dong and Z.-G. Tian, FPGA based design of elliptic curve cryptography coprocessor, In: *The 3rd International Conference on Natural Computation* (ICNC'07), IEEE, Haikou (2007), 185-189.

[57] C.H. Wu, C.M. Wu, M.D. Shieh and Y.T. Hwang, High-speed, low-complexity systolic designs of novel iterative division algorithms in $GF(2^m)$, *IEEE Transactions on Computers*, **53**, No 3 (2004), 375-380.

[58] H. Wu, A. Hasan and I.F. Blake, Highly regular architectures for finite field computation using redundant basis, *Lecture Notes in Computer Science*, **1717** (1999), 269-279.

[59] Z. Yan and D.V. Sarwate, New systolic architectures for inversion and division in $GF(2^m)$, *IEEE Transactions on Computers*, **52**, No 11 (2003), 1514-1519.