$Area - Time$ Efficient Hardware Implementation of Elliptic Curve Cryptosystem.

Anissa Sghaier[1], Medien Zeghid[1,2], Belgacem Bouallegue[1], Adel Baganne[3], and Mohsen Machhout[1]

[1] Laboratory of Electronics and Microelectronics, Faculty of Sciences Monastir, University of Monastir , Monastir 5019, Tunisia
sghaieranissa@yahoo.com, belgacem.bouallegue@fsm.rnu.tn, machhout@yahoo.fr
[2] Higher Institute of Applied Sciences and Technology, Taffala city 4003 Sousse, Tunisia, medien.zeghid@fsm.rnu.tn
[3] Information and Communication Science and Technology Laboratory (Lab STICC), CNRS: FRE2734 University of South Brittany, Lorient, France, adel.baganne@univubs.fr

Abstract. The strength of ECC lies in the hardness of elliptic curve discrete logarithm problem (ECDLP) and the hight level security with significantly smaller keys. Thus, using smaller key sizes is a gain in term of speed, power, bandwidth, and storage. Point multiplication is the most common operation in ECC and the most used method to compute it is Montgomery Algorithm. This paper describes an area-efficient hardware implementation of Elliptic Curve Cryptography (ECC) over $GF(2^m)$. We used the Montgomery modular multiplication method for low cost implementation. Then, to accelerate the elliptic curve point multiplication, we firstly adopted projective coordinates, and then we reduced the number of multiplication block used, so we have a gain at area occupation and execution time. We detailed our optimized hardware architecture and we prove that it outperform existing ones regarding area, power, and energy consumption. Our hardware implementation, on a Xilinx virtex 5 ML 50 FPGA, used only 9670 Slices achieving maximum frequency of 221 MHz, it computed scalar multiplication in only 2.58 μs. FPGA implementations represent generally the first step to obtain faster ASIC implementations.Further, we implemented our design on an ASIC CMOS 45 nm technology, it uses 0.121 mm^2 of area cell, it runs at a frequency of 990 MHz and consumes 39(mW).
Keywords: Elliptic Curves Cryptosystems(ECC), RSA, ASIC, Discrete Logarithm (DL), Elliptic Curves Discrete Logarithm Problems (ECDLP), memory resources.

1 Introduction

Since it's apparition in 1976 [1], ECC did a revolution in public-key cryptography because of its relatively short operand length compared to RSA which make it the stronger public key cryptosystem. In addition, ECC have many other advantages compared with other cryptosystems, firstly their key exchange protocol

is based on the difficulty of solving the discrete logarithm (DL) problem over a finite field. Secondly, there are a big number of available curves and ECC runs in exponential time which deal with current attacks. Thus, ECC can be used in various security services such as key exchange, authentification, digital signature etc. For example,the cryptographical use of ECC is in Public Key Identity (PKI) is to generate asymmetric(public/private) keys and encrypt/decrypt data. For these reasons, elliptic curve cryptosystems (ECC) have been extensively studied by the research community, Standards (like IEEE, ANSI, ISO) and also in industry.

ECC is particularly beneficial for application where computational power is limited, integrated circuit space is limited, high speed is required, intensive use of signing, verifying or authenticating is required, signed messages are required to be stored or transmitted and bandwidth is limited such as wireless communications devices, smart cards, PC cards etc.

Many Software and Hardware implementations are developed to give an efficient and practical elliptic curve cryptosystem design. ECC cryptosysem is based on computation of point addition and point doubling. The most crucial operation in ECC is the computation of point multiplication, i.e., computation of $kP = \underbrace{P + P \ldots + P}_{k \text{ times}}$ for given integer k and point P on elliptic curve. There are many available algorithms for the point multiplication, but the most used is Montgomery Algorithm.

To implement an efficient Point Multiplication Cryptosystem we need a hard study of modular arithmetic operation. We note that addition can be implemented by bitwise Xoring, but the most costly operation is inversion then multiplication. However, to solve a problem of inversion cost we can use projective coordinates. The LÓpez-Dahab algorithm [14] is the most used algorithms if we speak about binary field $GF(2^m)$ because it is a natural extension to binary case of so called Montgomery Ladder Algorithm, which is especially suitable for hardware implementation, so that point addition and point doubling data are independent.

To implement elliptic curve cryptography, we can use prime fields $GF(Z_p)$, where integers are defined between 1 and a prime p, or binary fields $GF(2^m)$, where polynomials have a set number of bits. In [13],Erich Wenger and Michael Hutter design both a binary and prime-field based ECC processor in order to compare them in a fair environment, and they find that the $GF(2^m)$ based processor outperforms the $GF(p)$ based processor in area, runtime, power and energy.

ECC Hardware implementation presents better performance than software implementation. Hardware implementations studies uses $GF(2^m)$, $GF(p)$, the key lengths varies from 163 bits to 233 bits, and different platforms was used FPGA, ASIC, sensor, smart card etc. Let's review some of the FPGA implementations of ECC over $GF(2^m)$. In 2008, Chang Hoon Kim and all.[2] give an elliptic curve cryptographic processor over $GF(2^{163})$ based on the López-Dahab point multiplication algorithm, and they parallelized their proposed architecture which is greedy in term of used slices. In 2010, Yu Zhang and all.[15] propose an ECC

cryptographic processor over $GF(2^{163})$, and they studied also parallelism, regarding both data dependency and critical path. They found best performance comparing to [2]. In 2012, Sutter G.D. and all [16] used either field-programmable gate array or application-specified integrated circuit technology to seed up point multiplication for elliptic curve cryptography and they implement $GF(2^m)$ scalar multiplication for m = 163, 233, 409, and 571. In 2013, Mahdizadeh H. and all. [10] proposed an architecture for elliptic curve scalar point multiplication and presented two implementations, the first suitable for speed-critical cryptographic applications and the second is suitable for applications that may require speed-area tradeoff. In 2014, Shuai Liu and all. [11] proposed hardware implementation architecture of elliptic curve scalar multiplication over binary fields, based on the Montgomery ladder method and uses polynomial basis for finite field arithmetics and using Karatsuba multiplier. In 2015, Zia U.A.Khan and M.Benaissa [12] proposed hardware design which used pipelined operations and cascaded operations. In this paper, we will implement Point Multiplication Processor using the minimum of components needed to compute the entire algorithm at time. The proposed design was synthesized using Xilinx ISE 14.5 and simulated with ModelSim XE III 6.4b.
The remainder of the paper is organized as follows.Section 2 gives a brief overview of the mathematical background related to ECC. Section 3 summarizes contributions dealing with previous implementations and comparisons of ECC and RSA. Section 4 present our architecture design. Section 5 introduces the implementation of ECC on FPGA Platform. Finally, we end this contribution with a discussion of our results and some conclusions.

2 Elliptic Curve Group Operation

In this section, we will give the definition of elliptic curve and we will present the strength of it based on the Discrete Logarithm Problem (DLP). Then we will give the elliptic curve encryption scheme.

2.1 Definition of an elliptic curve

An elliptic curve is the set of points (x, y) which are solutions of a bivariate cubic equation over a field K (see [17]). An equation of the form:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

Where $a_i \in K$, defines an elliptic curve over K. If char $K \neq 2$ and char $K \neq 3$, equation 1 can be transformed to: $y^2 = x^3 + ax + b$ with $a, b \in K$. In the field $GF(2^n)$ of characteristic 2, equation (1) can be reduced to the form: $y^2 + xy = x^3 + ax^2 + b$ with $a, b \in K$.
The set of points on an elliptic curve, together with a special point O called the point at infinity can be equipped with an Abelian group structure by the following addition operation:

Addition formula [6] for char $K \neq 2, 3$:
Let $P = (x_1, y_1) \neq O$ be a point, the inverse of P is $-P = (x_1, -y_1)$. Let $Q = (x_2, y_2) \neq O$ be a second point with $Q \neq -P$, the sum $P + Q = (x_3, y_3)$ can be calculated as:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

With:

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \text{ if } P \neq Q; \\ \frac{3x_1^2 + a}{2y_1} \text{ if } P = Q. \end{cases}$$

To subtract the point $P = (x, y)$, one adds the point $-P$.
Addition formula for char $K = 2$:
Let $P = (x_1, y_1) \neq O$ be a point, the inverse of P is $-P = (x_1, x_1 + y_1)$. Let $Q = (x_2, y_2) \neq O$ be a second point with $Q \neq -P$, the sum $P + Q = (x_3, y_3)$ can be calculated as:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$
$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

if $P \neq Q$ and:

$$x_3 = \lambda^2 + \lambda + a$$
$$y_3 = x_1^2 + (\lambda + 1)x_3$$
$$\lambda = x_1 + \frac{y_1}{x_1}$$

if $P = Q$.
Point addition and doubling each require 1 inversion and 2 multiplications. We neglect the costs of squaring and addition. Montgomery noticed that the x-coordinate of $2P$ does not depend on the y-coordinate of P.

2.2 Elliptic Curve Discrete Logarithm Problem (ECDLP)

The strength of ECC is how to solve a discrete logarithm problem In modern cryptography, one of the difficult problems to solve is ECDLP, when speaking about public key cryptography.
We can resume the problem as follows, if we take a point P on the elliptic curve, and Q a point defined as: $Q = kP$ which is in the same elliptic curve. In this way, ECDLP involves a scalar multiplication, so having k and P; it is very easy to find Q. But, having point P and point Q, it's very hard to find the scalar k. Furthermore, if scalar k becomes larger, it is computationally infeasible to obtain it [19].

2.3 Elliptic Curve Encryption Scheme

Elliptic Curve Encryption Scheme is one of the Public-key encryption. Here, message is encrypted with a public key and can't be decrypted without possessing private key. In this section, we will present EC encryption scheme which analogous to El-Gamal encryption [8].
System parameters:

- An elliptic curve E over $GF(p)$ or $GF(2^n)$.
- The order of E denoted E must be divisible by a large prime q.
- $G \in E$ of order q.

Key generation:

- Secret key: $d \in R[1, q-1]$.
- Public key: $Q = dP$.

Encryption of a message m:

- Pick $k \in R[1, q-1]$.
- Compute the points $kP = (x_1, y_1)$ and $kQ = (x_2, y_2)$, and $c = x_2 + m$.
- The ciphertext is (x_1, y_1, c).

Decryption:

- Compute $(x_2', y_2') = d(x_1, y_1)$ and $m = c - x_2'$.

2.4 Projective coordinates

Two-dimensional projective space P_k^2over K is given by the equivalence classes of triples (x, y, z) with x, y, z in K and at least one of x, y, z nonzero.
Two triples (x_1, y_1, z_1) and (x_2, y_2, z_2) are said to be equivalent if there exists a non-zero element λ in K:

- $(x_1, y_1, z_1) = (\lambda x_2, \lambda y_2, \lambda z_2)$
- The equivalence class depends only the ratios and hence is denoted by $(x : y : z)$

If $z \neq 0$, $(x : y : z) = (x/z : y/z : 1)$, and if $z = 0$, we obtain the point at infinity. The two dimensional affine plane over K:

$$A_k^2 = \{(x, y) \in K \times K\}, henceusing, (x, y) \rightarrow (X : Y : 1)$$
$$A_k^2 = P_k^2$$

The number of inversions and multiplications for a group operation on EC heavily depends on the chosen coordinate system. Assuming that the cost of one field inversion is equivalent to m field multiplications, so from the implementation point of view, there are advantages with projective coordinates. We used Lopez and Dahab Projective Transformation to reduce Inverters: (X, Y, Z), $Z \neq 0$, maps to $(X/Z, Y/Z^2)$

In Projective Coordinates:

$$X_3 = \begin{cases} xZ_3 + (X_1Z_2 + X_2Z_1) \text{ if } P \neq Q; \\ X_1^4 + bZ_14 \qquad\qquad\quad \text{if } P = Q. \end{cases}$$

$$Z_3 = \begin{cases} (X_1Z_2 + X_2Z_1)^2 \text{ if } P \neq Q; \\ Z_1^2X_1^2 \qquad\qquad\quad \text{if } P = Q. \end{cases}$$

Let $P(x, y)$ be a point of the curve, and B a polynomial, change from affine coordinates to projective coordinates is:

$$\begin{cases} X_1 = x \\ Z_1 = 1 \\ X_2 = X^4 + B \\ Z_2 = X^2 \end{cases}$$

Then to return from projective coordinates to affine coordinates:

$$x_3 = X_1/Z_1$$

$$y_3 = (x + X_1/Z_1)[(X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1Z_2)](xZ_1Z_2)^{-1} + y$$

This conversion requires 10 multiplications and one inverse operation.

3 Montgomery Point Multiplication Architecture

There are different algorithms to compute scalar multiplication. One of the most used is Montgomery's Algorithm which compute kP over non-supersingular Elliptic Curve. In this section we will present the Montgomery Algorithm and sub-algorithms used to compute it.

3.1 Montgomery Algorithm and Hierarchy of kP

The Montgomery method, as it's shown in algorithm 1, is based on two operations: Point Addition (PA) and Point Doubling (PD). PA and PD perform point mathematics operations of elliptic curve cryptosystem. Montgomery's algorithm, over a binary field, is based on parallel point multiplication. PA ad PD operations are calculated independently, for this reason they can be paralleled and computed at the same time. Furthermore, complexity to compute PD is simpler as that PA. The point addition is computed using arithmetic operations, thus it need one squaring, four multiplications, and two additions. Similarly, point doubling need one addition, two multiplications, and four squarings.

Algorithm 1 : Montgomery Point Multiplication

Input: $k = (k_{n-1}, k_{n-2}, ..., k_1, k_0)_2$ with $k_{n-1} = 1$, $P(x, y) \in E(F_2^m)$
Output: $Q = kP$
Set $X_1 = x; Z_1 = 1; X_2 = x^4 + b; Z_2 = x^2$
for i from n-2 down to 0 **do**
  if $k_i = 1$ **then**
    Madd$(X_1; Z_1; X_2; Z_2)$;
    Mdouble$(X_2; Z_2)$;
  else
    Madd$(X_2; Z_2; X_1; Z_1)$;
    Mdouble$(X_1; Z_1)$
  end if
end for
return $(Q = Mxy(X_1; Z_1; X_2; Z_2)$

Figure 1 present the hierarchy of scalar multiplication. As we can see the main implementation of scalar multiplication relies on: firstly basic arithmetic operations which are square, multiplication and inversion, secondly, point doubling and point addition based on arithmetic operations. So, the scalar multiplication consists of basic arithmetic operations and point doubling and point addition.

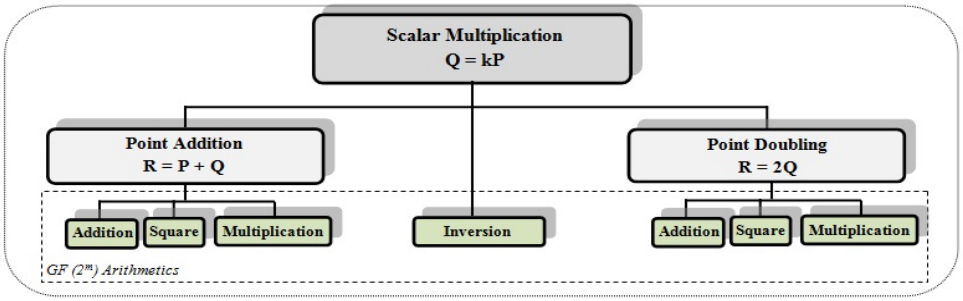

Fig. 1. Hierarchy of scalar multiplication

Modular Inversion
An inversion is the most costly operation, is defined as follows: given $a(x) \in GF(2^m)$, find $a(x)^{-1}$ such that $a(x) \times a(x)^{-1} = 1 mod I(x)$, where $I(x)$ is the irreducible polynomial. There are two known methods which are Fermat Algorithm and Extended Euclidean Algorithm. But the most popular and most used method is the Extended Euclidean Algorithm.

Modular multiplication
Performance of Public Key Cryptography, such as HECC, ECC and RSA is strongly related to an important operation to compute them, which is modu-

lar multiplication, for this reason different implementation methods were presented in the literature. In ECC over binary field, polynomial multiplication is simply a shift and Xoring operation. For this reason, we choose to implement Montgomery's algorithm to obtain an efficient modular multiplication. When presenting the Montgomery multiplication algorithm, the main idea is define different coordinate systems in order to have an efficient design with the best performance related to area and execution time.
For implementing Montgomery modular multiplication, we proceed as follows: let's $x = a \times bmodn$, the first procedure is converting all the operands to Montgomery representations; then, performing Montgomery's algorithm for each operations; finally, converting all Montgomery representation of operands back to their original representations [15]. R is represented as a power of two, $R = 2^k$, where k is large enough so that $R > N$, so modulo R is a division by 2^k which is a k bits shift, it is very easy to determine its result.

3.2 Projective system choice

Let's take a point $P = (x, y)$, to transform (x, y) from affine to projective coordinates (X, Y, Z) with $Z \neq 0$. Thus, $x = X/Z^\alpha$, $y = Y/Z^\beta$, the elliptic curve equation be:

$$(\frac{Y}{Z^\beta})^2 + (\frac{X}{Z^\alpha}.\frac{Y}{Z^\beta}) = (\frac{X}{Z^\alpha})^3 + a.(\frac{X}{Z^\alpha})^2 + b$$

α and β should be well chosen, in a way that scalar multiplication necessitate only multiplication and addition in the finite field GF(2^m).

Table 1. Number of operations needed in projective system

Projective System		**Multiplier Number**	**Square Number**	**Multiplexer Number**
Homogenous	Addition	16	1	16
	Doubling	7	5	7
Jacobian	Addition	11	4	11
	Doubling	5	5	5
Standard	Addition	14	5	14
	Doubling	5	5	5
Montgomery	Addition	3	1	3
	Doubling	3	5	3

So to find the appropriate projective system and a good choice of α and β, we have implemented four methods of projective system which are: standard ($\alpha = 1$ et $\beta = 1$), Jacobian ($\alpha = 2$ et $\beta = 3$), Lopez-Dahab ($\alpha = 1$ et $\beta = 2$) et Montgomery ($\alpha = 1$ et $\beta = 1$).
Table 1 present the number of needed operations to compute arithmetic calculus in elliptic curve, using the different projective systems in term of multiplication,

squaring and multiplexing.

An optimized implementation to arithmetic operations using different projective systems was done, then we drew table 2 which represents the number of needed operations, based in different projective system, to compute arithmetic calculus in elliptic curve after reduction.

Table 2. Number of operations needed in proposed projective system

Projective System		**Multiplier Number**	**Square Number**	**Multiplexer Number**
Homogenous	Addition	5	3	10
	Doubling	2	4	4
Jacobian	Addition	3	4	6
	Doubling	4	2	6
Standard	Addition	3	4	6
	Doubling	1	5	2
Montgomery	Addition	1	1	2
	Doubling	1	5	2

From results of Table 2, we remark that the most optimized projective system is Montgomery, because it use the minimum number of operation to compute Addition and Doubling operation in elliptic curve over GF(2^m).
We will present implementation results of scalar multiplication over $F_{2^{163}}$ using Certicom recommendation [31], of different projective systems mentioned earlier. Performance results, shown in figure 2, 3, 4, are given in term of memory occupation, execution time and power consumption. Scalar multiplication using different projective systems was implemented on a FPGA VirtexE XCV2600-8fg1156.

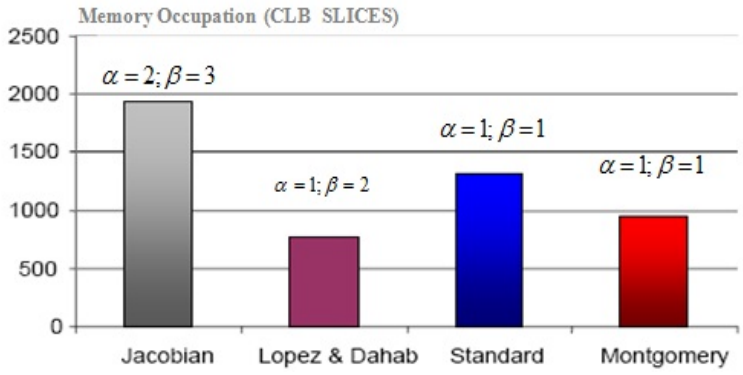

Fig. 2. Memory occupation histogram of projective systems

Implementation results in figure 2 show that scalar multiplication using Lopez-Dahab projective system is the most optimal in term of memory occupation, which is 7579 Slices.

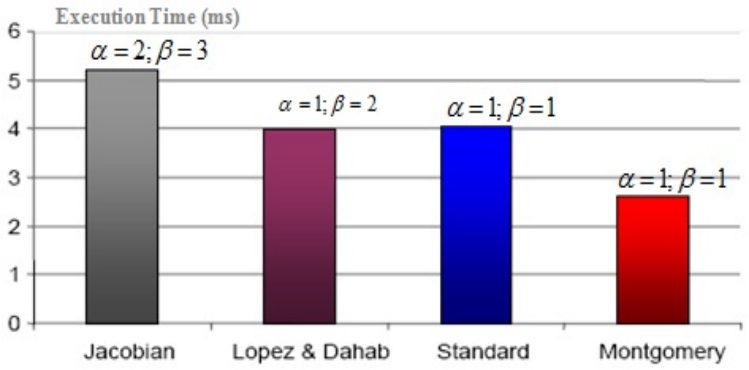

Fig. 3. Execution time histogram of projective systems

Then, from figure 3, we note that scalar multiplication based on Montgomery projective system is the most optimal in term of execution time, 2,618 ms and power consumption 65.83 mW.

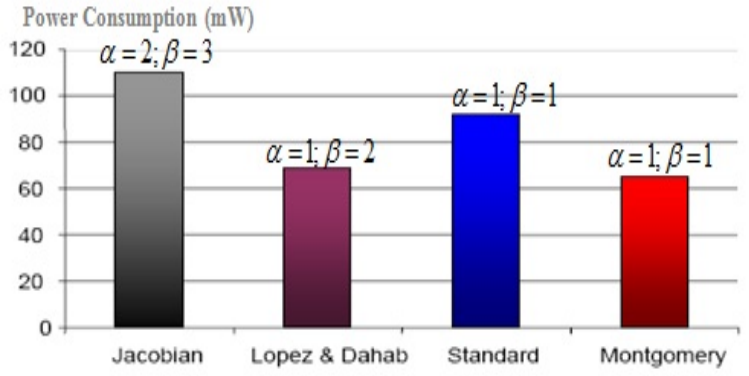

Fig. 4. Power consumption histogram of projective systems

Figure 4 gives the consumption of every method, we remark that the method of Lopez-Dahab and Montgomery have the minimum consumption.
The choice of the appropriate method among the four methods cited below, is determined by the target platform. Indeed, if we need speed, regardless of occupation, we must choose the method of Montgomery; whereas if the platform

has a reduced memory space, as is the case with smart cards, we must opt for the method of Lopez-Dahab. In our implementation we will choose ...

4 Montgomery Point Multiplication Processor Design and Implementation

Montgomery Point Multiplication algorithm computes the point multiplication in a fixed time, this can be a gain because it makes it more resilient to side-channel attacks based on timing or power consumption measurements. In this section, the properties of the our proposed cryptosystem is given. We will first present the main blocks needed to compute scalar multiplication and then we will detail the point addition and point doubling computation.

4.1 Montgomery Point Multiplication Processor Design

In this part, we will introduce the main blocks used to compute Montgomery Point Multiplication. Then we will detail the data path of our cryptosystem in figure 5 which shows the data manipulating by every component.
The seven based blocks required for executing scalar point multiplication are:

- Input Interface (II): in order to decrease input length and provide a word-by-word operation, the input operands are divided into smaller bit length words, data acquisition will be done step by step, as it's shown in 5 we received the 163×2, 30-bits by 30-bits, in this way we don't need larger number of I/O devices for our cryptosystem.
- Affine to project conversion (Aff vs Proj): this block transform the coordinate of a point P(x,y) to a point P'(X,Y,Z). It used one multiplier and one register to do the entire conversion, it's based on the reuse of multiplier block and the storage of result in every step. Scalar multiplication can't be computed if this block did'nt sent the signal "Start" to the controller.
- Controller: is the main block which interacts with the user to get inputs data (k and point P), passing them to the Point Addition Block and Point Doubling Block. It provides the necessary control signals to all components. The controller wait that "Aff vs Proj" block end its conversion, then receiving "Start" signal, it begin to synchronize the other blocks.
- Montgomery Point Addition (MPA): it perform field addition. It uses modular arithmetic multiplication and Xoring operations, then it store results in register files. To compute MPA, we need 5 multiplications, but we will use only 2 multipliers because we have dependance in operation calculus and also we have to find a compromise between area and speed.
- Montgomery Point Doubling (MPD): it compute field doubling. It is based on modular arithmetic multiplication, Xoring operations and registers to store results. MPD computation is based on 6 multiplications, but in our case we will use only 2 multiplication blocks because calculus is independent in some case and dependent in other cases, so if we study this dependance we can decrease number of blocks need.

– Projective to affine conversion (Proj vs Aff): After doing all operations of scalar multiplication, this block did the conversion from projective to affine coordinates. Then it sent the result to the controller which sent it to the OI. It uses two multiplication and two inversion blocks to compute the entire conversion.
– Output Interface (OI): in order to decrease output length and used I/O devices, final results are divided into groups of smaller bit length of 30-bits. In this way, final output will be send by the OI word by word. As it's mentioned in 5, we have two outputs Out-X and Out-Y, every one have a length of 163-bits and they were sent by word of 30-bits.

The processor design is presented by the figure 5.

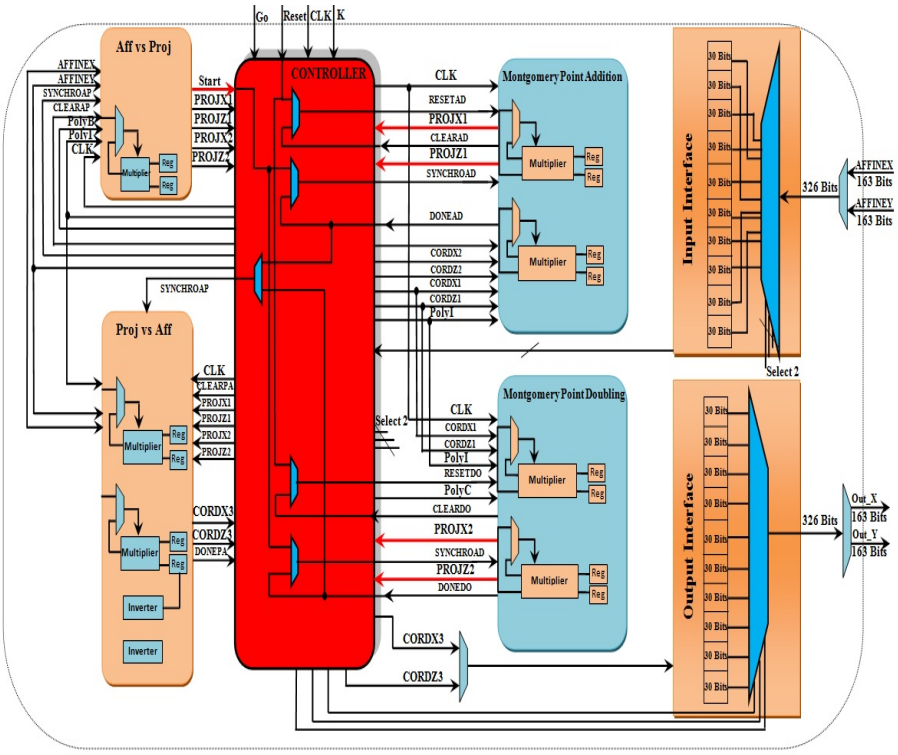

Fig. 5. ECC Hardware Processor Design

Addition of 2 points or Doubling require inversion computation which is the most costly operation, and in affine coordinates, inversions are very expensive. We used projective coordinates in order to replace inversions by the multiplication operations and then perform one inversion at the end.
In order to provide a word-by-word operation the input AFFINEX and AFFINEY

was divided into ten 30-bit words by the *Input Interface* which send data to the *Controller*.
Based on Figure 5, receiving the scalar k, the point P and signals of activation (CLK, GO and Reset), the *controller* send affine coordinates (AFFINEX, AFFINEY) and irreducible polynomial to the *Aff vs Proj blocs* with the signals of activation (CLK, SYNCHROAP and Reset). Receiving the necessary data, *Aff vs Proj blocs* convert affine coordinates to projective coordinates then it send a signal "Start" to the *controller* in order to activate simultaneously MPA and MPD blocks by sending needed data and signals (CORDX1, CORDX2, CORDZ1, CORDZ2, CLK, RESET and SYNCHRO). The MPA compute point addition, it used two multiplier, so in every step it activate them, here we use parallel approach and serial approach alternately. We applied the same approach to point doubling so the MPD in the same step activate the two multiplier in parallel, then in the next step it reactivate the same two multiplier till the MPD block compute the entire calculus. After computing point addition and point doubling, the MPA and MPD send results (PROJX1, PROJZ1, PROJX2 and PROJZ2) to the *controller*. It the *controller* activate the *Proj vs Aff blocs* to do the conversion from projective to affine coordinates. Having the results of point addition from MPA and pont doubling from MPD, and following the same methodology as the other blocks, the *controller* send needed data to the *Proj vs Aff blocs* (PROJX1, PROJZ1, PROJX2 and PROJZ2) with activation signals (CLK, Reset and SYNCHROPA). The *Proj vs Aff blocs* activate its basic operations to compute conversion which are multiplication and inversion.
Computing the entire scalar multiplication, final results will be sent from Output Interface word-by-word, it will be split to a group of of 32-bits word to minimize I/O devices use.

More details about the implementation of the main components of ECC scalar multiplication computation are provided in the following Section.

4.2 Montgomery Point Multiplication Processor Implementation

Our hardware design consists on components full time function. To achieve scalar point multiplication operations, we need concurrent and cascaded operations. Thus, we have dependent and independent operations. We will apply this approach in both Point Addition Operation and Point Doubling Operation.
We should note that the control of all operations is performed by the *Controller block*. In this section we will details Controller function, Point Addition Operation and Point Doubling Operation.

Controller function. The Controller is the main component in our design, it is the responsible of the communication between all components. In this section we will describe the way that the controller synchronize between components. Figure 6 gives the hard finite state machine of the Controller.

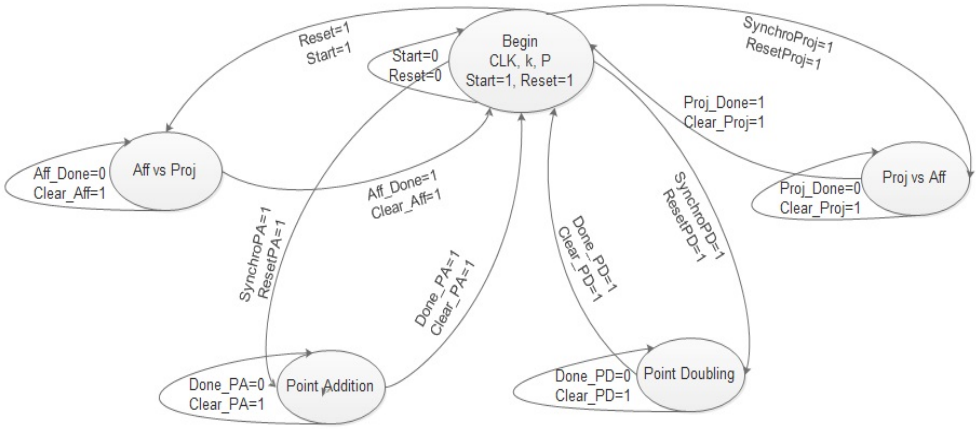

Fig. 6. State machine of Controller

Receiving data in (k and P) and signals CLK, Reset=1 and Start=1, the Controller activate "Aff vs Proj" block to convert P coordinates from affine to projective coordinates. During conversion, Clear-Aff takes 1 and Aff-Done takes 0. If "Aff vs Proj" complete the conversion Clear-Aff takes 0 and Aff-Done takes 1. Receiving Aff-Done=1, the Controller sends signals of activation to Point Addition block and Point Doubling block respectively: SynchroPA=1, ResetPA=1, SynchroPD=1 and ResetPD=1. Starting point addition computation and point doubling computation, signals Clear-PA and Clear-PD take 1 and signals Done-PA and Done-PD takes 0. Point Addition component and Point Doubling component are activated firstly by signals SynchroPA, ResetPA, SynchroPD and ResetPD sent by the Controller, then they will be activated and disabled by output signals Clear-PA and Clear-PD. When the scalar multiplication is completed, the Controller activate the conversion from projective to affine coordinates. It sends signals SynchroProj=1 and ResetProj=1 to start conversion. "Proj vs Aff" bloc makes: Clear-Aff=1 and Aff-Done=0. When finishing conversion, Clear-Aff takes 0 and Aff-Done takes 1. During scalar multiplication computation, operands are stored in different register files. Thus, in every step, the Controller load operands then store results to be used in the next operands, until the scalar multiplication is complete.

Point Addition Operation. In this section, we will illustrate point addition operation. Let's take two point defined as: $P_1, P_2 \in E[GF(2^m)]$ and which are presented in projective coordinates and $P_3 = P_1 + P_2$. Point Addition Algorithm are given by algorithm 2, so to compute point addition we need 5 multiplications and 1 Xoring block. Point addition hardware architecture is illustrated in Figure 7.

Algorithm 2 : ECC Point Addition In Projective coordinates

Input: x, X_1, Z_1, X_2, Z_2
Output: X_3, Z_3
Step1
$X1 = X_1 \times Z_2$
$Z1 = Z_1 \times X_2$
Step2
$T2 = X1 \times Z1$
$Z_3 = (Z1 + X1)^2$
Step3
$X1 = Z_3 \times x$
$X_3 = (Z_3 \times x) + T2$
return X_3, Z_3

Point Doubling Operation is the second main operation of scalar point multiplication, so it will be studied in the next section.

Point Doubling Operation. Point doubling is implemented using the same approach used in point addition implementation. Let's take a point $P_1 \in E[GF(2^m)]$ presented in projective coordinates. To compute $P = 2 * P$ we use algorithm 3 and we need 6 multiplications and 1 Xoring. Figure 7 give the hardware architecture of Point doubling.

Algorithm 3 : ECC Point Doubling In Projective coordinates

Input: c, X_1, Z_1
Output: X_3, Z_3
Step1
$X = X_1^2$
$Z = Z_1^2$
Step2
$X = X^2$
$T_1 = Z \times c$
Step3
$Z_3 = Z \times X$
$T_1 = T_1^2$
$X_3 = X + T_1$
return X_3, Z_3

To implement Point Addition and Point Doubling we have optimized the components number needed to compute them, this optimization will be explained in the next section.

Point Addition and Point Doubling optimization. The Figure 7 resumes the parallel computation of point addition and point doubling. As we can see,

firstly, we compute the first step of point addition and point doubling respectively which is based on two multiplications every one, so multiplier1, multiplier2, multiplier3 and multiplier4 are activated in parallel. Here, by analyzing Algorithms 3 and 2, in step1, we calculated $X1$ and $Z1$ in point addition and X and Z in point doubling using inputs values. Then, in step2 of respectively point addition and point doubling, $T2$ and Z_3 depends on $X1$ and $Z1$, X and T_1 depends on X and Z. It's the same approach for step3, so we remark that calculus in every step depends on the previous one, and in the same step calculus is independent, from this remark we had the idea to use only 2 multipliers to compute point Addition and 2 multipliers to compute point doubling. In Every step, results are stored in different registers, then they will be used in the next steps. Thus, step 2 will be performed by reactivation of the four multipliers, and in step 3, we activate only multiplier1, multiplier3 and multiplier4. The serialization here is done by the use of the same component (Mult1, Mult2, Mult3, Mult4 and XOR) in every step. Our aim here is to handle robustly serialization approach and parallelism approach in order to find a well compromise between area and execution time.

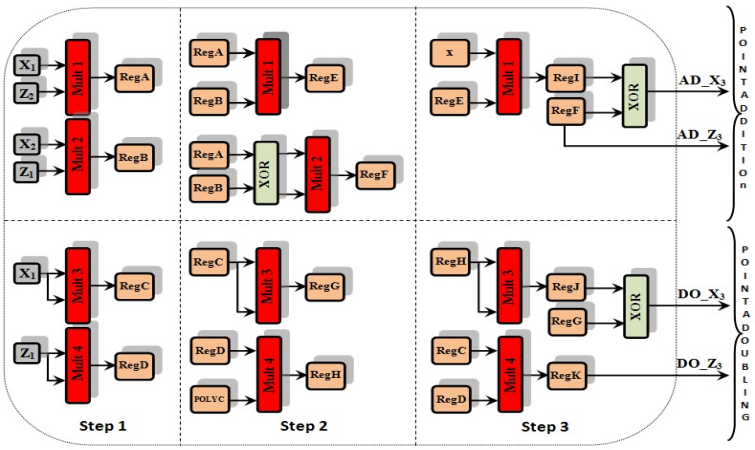

Fig. 7. Sequence of ECC Point Addition and Point Doubling Operations

As it's mentioned earlier, we will use only 4 multiplication blocks (2 for Point Addition and 2 for Point Doubling).

The study of the calculus dependance in Algorithm 2 and Algorithm 3, let as minimize number of blocs need and verify components full time function as it mentioned in Table 3. The reduction of block number, decrease the area occupation of both point addition and point doubling.

Table 3. Number of required operation in each step of both Point Addition and Point Doubling

	Point Addition		Point Doubling	
	Mult1	Mult2	Mult3	Mult4
Step1	✓	✓	✓	✓
Step2	✓	✓	✓	✓
Step3	✓		✓	✓

Let's now take the example of Point Addition computation where we activated the 2 multipliers in every step and we reactivated them in the next step. Figure 8 shows the chronogram of signals which make the synchronization between multipliers used to compute point addition operation.

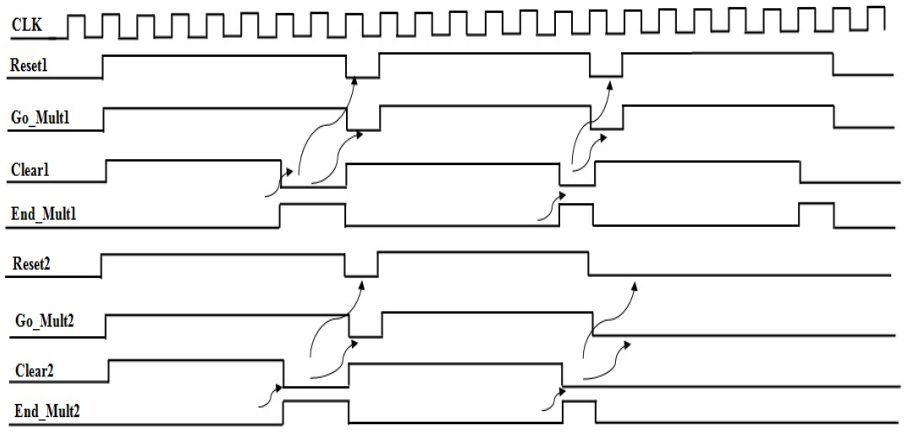

Fig. 8. Point Addition Chronogram

As we can see in Figure 8, every multiplier receives two signals (Reset and Go-Mult) from Controller to begin calculus and they make the signal Clear='1' and End-Mult='0'. In the first step, Mult1 and Mult2 compute the first two multiplications. When they finish, the signal End-Mult takes '1' and Clear takes '0' to indicate the end of multiplication. Then, the two multipliers will be disabled. They will be reactivated in the next step by the signal Clear.

We applied the approaches presented earlier to compute Point Doubling. The performance of our hardware implementation is presented in the next section, and a comparison with the state of the art is given.

5 Experimental Results

We have synthesized the architecture using the ISE 14.5 FPGA and ASIC. Arithmetic units were synthesized for the Koblitz curve recommended by NIST [20], for the finite field $GF(2^m)$ using the irreducible polynomial $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$. Our synthesis results for ECC Point Multiplication over $GF(2^{163})$ field are summarized in Table 4. Results are given in term of area requirement, frequency and time.

Table 4. ECC Point Multiplication Implementation results

Designs	**Curve**	**Platform**	**Area**	**Freq.** (MHz)	**Time** (μs)
Results1	$GF(2^{163})$	virtex5ML50	9670 Slices	221	2.58
Results2	$GF(2^{163})$	ASIC	0.121 mm^2	990	0.576

Using virtex5ML50 FPGA platform, our design used only 9670 slices to perform scalar point multiplication, achieving maximum of frequency of 221 MHz in 2.58 μs. In ASIC platform, it used 0.121 mm^2 area cells with a frequency of 990 MHz and time of 0.576 μs, it consumed 39 mW.

Table 5. ECC Implementation results comparison

Designs	**Curve**	**FPGA**	**Area** Slices	**Freq.** (MHz)	**Time** (μs)	**Time/Area** (10^{-4})
Our Design	$GF(2^{163})$	virtex5ML50	9670	221	2.58	2.667
Our Design	$GF(2^{163})$	XC4VLX200	9300	99	5.75	6.18
Our Design	$GF(2^{163})$	XC4VLX80	9213	118	4.83	5.24
ECC [2]	$GF(2^{163})$	XC4VLX80	24263	143	10	4.12
ECC [3]	$GF(2^{163})$	XC4VLX200	16209	153.9	19.55	12.06
ECC [4]	$GF(2^{163})$	XC4VSX35	10488	99	144	137.29
ECC [5]	$GF(2^{163})$	XC2V2000	3416	100	41	120.02
ECC [6]	$GF(2^{163})$	XC4VLX80	20807	185	7.7	3.70
ECC [7]	$GF(2^{163})$	XC4VLX80	8070	147	9.7	12.02
ECC [8]	$GF(2^{163})$	Virtex4	12834	196	17.2	13.40
ECC [9]	$GF(2^{163})$	Virtex5	6150	250	5.5	8.94
ECC [10]	$GF(2^{163})$	XC4VLX200	14203	263	11.6	8.17
ECC [11]	$GF(2^{163})$	XC4VLX200	10417	121	9	8.63
ECC [12]	$GF(2^{163})$	Virtex5	10363	153	5.1	4.92

Table 5 give a comparison between our results and the state of the art implementations of scalar multiplication. The different parameters used are Area, Frequency and Time consumption.
We note that, comparing our results to the older implementations done in 2008, such that [2], [3] and [4], we find that they used respectively 24263 slices, 16209

slices and 10488 slices. We present a gain, in area occupation, of 62.03%, 42.62% and 12.16% respectively. [2] and [3] have a higher frequency, but with an important execution time. Implementation in [5] used the less area occupation which is 3416 slices, but our implementation outperform it in term of frequency and time.
We implement our design in the same platform of [6] which is XC4VLX80, we have gain of 55.71% in area and 66.49% in time with a decrease of frequency by 36.22%.
Results in [7] uses less memory of our design but they compute scalar multiplication in 9.7 μs whereas our design compute it in only 4.83 μs. We outperform implementation in [8], [10] and [11] in term of area and necessary time to compute scalar multiplication, with a loose in frequency.
Implementation of [12] is recent, it's done in 2015 using Virtex5 platform, we present better results comparing to them in all parameters. Thus, we decrease the area by 6.68%, we increase frequency by 44.4% and we decrease time by 49.4%. Our architecture prove that it reaches high performance running in FPGAs circuit.

In order to have a relevant performance comparisons between our synthesis results and related works results, we will add another parameter which is Area/Time, it gives the ratio between Area and Time. Figure 9 gives different ratios of the mentioned state of the art works.

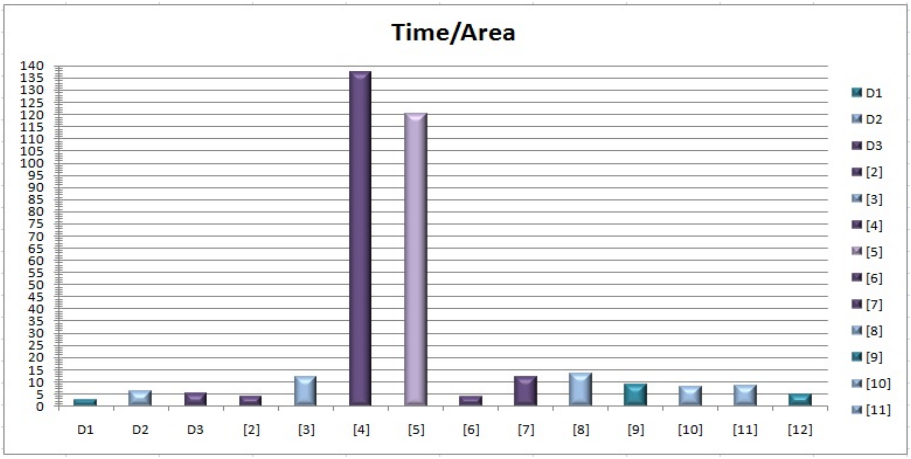

Fig. 9. Area/Time diagram of different works

It's clear in Figure 9 that, using different platforms XC4VLX80, XC4VLX200 and Virtex5, our designs have the minimum ratio area per time. Our design (D1, D2 and D3) focus on cost/area minimization, but also we should have a balance between area and time.

Our implementations have the best Area/Time ratio, we prove that our design is the most efficient comparing to other designs.

6 Conclusion and Future works

6.1 Conclusion

Elliptic curve cryptosystems give the most security per bit compared to the other public-key cryptosystems. Thus, ECC have the same benefits of the other cryptosystems, but it outperform them due to its shorter key lengths, speed and memory saving. ECC is the most efficient for applications with lacks of power and small storage memory. In this paper, we focus on area minimization with a respect to the execution time. We proposed an efficient hardware architecture to compute ECC scalar multiplication. Our architecture was implemented in different FPGA platforms to do a comparison between our design and the state of the art designs, and then in ASIC platform. We prove that our design have better performance compared to the others.

6.2 Future works

The elliptic curve cryptosystems will be the most used in the future because of its shorter key and its better security compared to the other public cryptosystems. One of the applications that we can implement is the digital signature system based on elliptic curve (ECDSA) which represents one of the main digital signature systems.
Furthermore, based on our proposed architecture described below, we can propose lightweight coprocessor for 16-bit microcontrollers that implements high security elliptic curve cryptography and which is more suitable for Wireless Sensor Networks (WSN), mobile phone, smart card etc.

References

1. W. Diffie and M. E. Hellman, *New directions in cryptography.* IEEE Transactions on Information Theory, IT-22:644-654, 1976.
2. Chang Hoon Kim, Soonhak Kwon and Chun Pyo Hong. *FPGA implementation of high performance elliptic curve cryptographic processor over* $GF(2^{163})$. Journal of Systems Architecture, 2008.
3. W.N. Chelton and M. Benaissa. *Fast elliptic curve cryptography on FPGA*, IEEE Trans. Very Large Scale Integr. VLSI Syst. 16 (2) (2008) 198-205.
4. S. Antao, R. Chaves and L. Sousa. *Efficient FPGA elliptic curve cryptographic processor over* $GF(2^m)$, ICECE Technology, FPT 2008, pp. 357-360. International Conference on 2008.
5. B. Ansari and M. Anwar. *High-performance architecture of elliptic curve scalar multiplication*, IEEE Transactions on Computers 57 (11) (2008) 1443-1452.

6. Y. Zhang, D. Chen, Y. Choi, L. Chen and S. Ko. *A high performance pseudo-multicore ECC processor over* $GF(2^{163})$, in: Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), 2010, pp. 701-704.
7. C. Rebeiro, S.S. Roy and D. Mukhopadhyay. *Pushing the limits of high-speed* $GF(2^m)$ *elliptic curve scalar multiplication on FPGAs*, in: Cryptographic Hardware and Embedded Systems, CHES 2012, vol. 7428, 2012, pp. 494-511.
8. R. Azarderakhsh and A. Reyhani-Masoleh. *Efficient FPGA implementations of point multiplication on binary edwards and generalized hessian curves using gaussian normal basis*, IEEE Trans. Very Large Scale Integr. VLSI Syst. 20 (8) (2012) 1453-1466.
9. G. Sutter, J. Deschamps and J. Imana. *Efficient elliptic curve point multiplication using digit-serial binary field operations*, IEEE Trans. Ind. Electron. 60 (1) (2013) 217-225.
10. Mahdizadeh, H. and Masoumi, M. *Novel Architecture for Efficient FPGA Implementation of Elliptic Curve Cryptographic Processor Over* $GF(2^{163})$, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on (Volume:21 , Issue: 12, p 2330-2333, IEEE Circuits and Systems Society, 2013.
11. Shuai Liu, Lei Ju, Xiaojun Cai, Zhiping Jia and Zhiyong Zhang. *High Performance FPGA Implementation of Elliptic Curve Cryptography over Binary Fields*, Trust, Security and Privacy in Computing and Communications (TrustCom), Beijing 2014 IEEE 13th International Conference.
12. Zia U. A. Khan and M. Benaissa. *High Speed ECC Implementation on FPGA over* $GF(2^m)$. International Conference on Field-programmable Logic and Applications (FPL)2-4th September, 2015.
13. Erich Wenger and Michael Hutter, *Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations*. Information Security Technology for Applications Volume 7161 of the series Lecture Notes in Computer Science pp 256-271, 2012.
14. Julio López and Ricardo Dahab, *Fast Multiplication on Elliptic Curves over* $GF(2^m)$ *Without Precomputation,* In CHES 99, Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems, London, UK: Springer-Verlag, pp. 316-327, 1999.
15. Yu Zhang, Dongdong Chen, Younhee Choi, Li Chen and Seok-Bum Ko. *A high performance ECC hardware implementation with instruction-level parallelism over* $GF(2^{163})$, Microprocessors and Microsystems 34 (2010) 228-236.
16. Sutter G.D., Deschamps J. and Imana, J.L. . *Efficient Elliptic Curve Point Multiplication Using Digit-Serial Binary Field Operations*, Industrial Electronics, IEEE Transactions on (Volume:60 , Issue: 1), IEEE Industrial Electronics Society, p 217 - 225, 2012.
17. Alfred J. Menezes, *Elliptic Curve Public Key Cryptosystems*, The Springer International Series in Engineering and Computer Science, 1993.
18. M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning Publications, January 1, 1998. https://www.manning.com/books/implementing-elliptic-curve-cryptography
19. T. El Gamal. , *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Trans. Info. Theory, IT-31, 1985, pp 469-472.
20. IEEE 1363, *Standard Specifications for Public key Cryptography*, 2000.
21. N. Koblitz, *Elliptic curve cryptosystems*. Mathematics of Computation, 48:203-209, 1987.

22. V. Miller, *Uses of elliptic curves in cryptography*. In H. C. WIlliams, editor, Advances in Cryptology CRYPTO '85, volume LNCS 218, pages 417-426, Berlin, Germany, 1986. Springer-Verlag.
23. D. Hankerson, J. Lopez Hernandez, and A. Menezes, *Software Implementation of Elliptic Curve Cryptography Over Binary Fields*. In Çetin K. Koç and Christof Paar, editors, Workshop on Cryptographic Hardware and Embedded Systems CHES 2000, pages 1-24. Springer Verlag, August 2000. LNCS 1717.
24. J. Lopéz and R. Dahab, *Improved algorithms for elliptic curve arithmetic in* $GF(2^n)$. In Selected Areas in Cryptography - SAC '98, pages 201-212, 1999. LNCS 1556.
25. D.V. Chudnovsky and G.V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*. In Advances in Applied Mathematics, volume 7, pages 385-434, 1987.
26. H. Cohen, A. Miyaji, and T. Ono, *Efficient elliptic curve exponentiation using mixed coordinates*. In K. Ohta and D. Pei, editors, Advances in Cryptology, ASIACRYPT 98, pages 51-65. Springer Verlag, 1998. LNCS 1514.
27. S. Baktri, E. Savas, *Highly-Parallel Montgomery Multiplication for Multi-core General-Purpose Microprocessors*, Department of Computer Engineering, Bahcesehir University, 2011, http://eprint.iacr.org/2012/140.pdf
28. Praveen Bhide and Prof D.V Manjunatha. *Design and Implementation of Elliptic Curve Cryptographic Processor*. Second International Conference on Recent Advances in Science and Engineering-2015.
29. M.N. Hassan and M. Benaissa. *Efficient time-area scalable ECC processor using coding technique*, in: M.A. Hasan, T. Helleseth (Eds.), Arithmetic of Finite Fields, vol. 6087, Springer, Berlin Heidelberg, 2010, pp. 250-268.
30. M. Benaissa and W.M. Lim. *Design of flexible* $GF(2^m)$ *elliptic curve cryptography processors*, IEEE Trans. Very Large Scale Integr. VLSI Syst. 14 (6) (2006) 659-662.
31. Certicom Research, *Recommended Elliptic Curve Domain Parameters*, "Standards for Efficient Cryptography 2 (SEC2)", January 27, 2010, Version 2.0