High Performances ASIC based Elliptic Curve Cryptographic Processor over $GF(2^m)$

Z. Guitouni[1,3], R.Chotin-Avot[2], M. Machhout[3], H. Mehrez[2] and R. Tourki[3]
[1]Higher Institute of applied Sciences and Technology of Mahdia, Tunisia.
[2] University of Paris VI, LIP6/SOC Laboratory of University Pierre & Marie Curie, France
[3]Electronic and Micro Electronic Laboratory, Faculty of Sciences of Monastir, Tunisia

ABSTRACT
Elliptic Curve Cryptography (ECC) has gained increasing acceptance in the industry, the academic community and the cryptography applications. This interest is mainly due to the high level of security with relatively small keys provided by ECC. In this paper, a high-performance ASIC based ECC key generation processor is proposed. This processor supports generic elliptic curves over $GF(2^m)$ with sizes (m) ranging from 113 to 256 bits. The proposed processor is based on programmable cellular automata. For real time implementation, the processor was simulated using active-HDL and synthesized using Synopsys Design Compiler. Further, the processor is implemented by an ASIC CMOS 120 nm technology. The results on the layouted processor over $GF(2^{256})$ show a high performance, confirming the efficiency of the processor.

Keywords
Elliptic curve cryptography, cellular automata, finite fields, ASIC and Montgomery point multiplication algorithm.

1. INTRODUCTION
Elliptic curves cryptosystem (ECC) is a potential public key cryptosystem to become the dominant encryption method for information and communication system. The ECC was proposed in 1985 by Neal Koblitz [1] and Victor Miller [2], and the security of it rests on the discrete logarithm problem over the points on an elliptic curve. The ECC provides higher strength-per-bit than any other current public-key cryptosystems [3]. Because of its higher strength-per-bit, Elliptic Curve Cryptosystems are being increasingly used in embeded systems (e.g. IC card and mobile devices) instead of RSA, which is most used for public-key cryptosystems.

In the last decade, the approach of hardware implementing Elliptic Curve Cryptography algorithms knew a very intensive race, due essentially to the requirements of security, speed and area constraints. Different security organizations like ISO, ANSI, IEEE and NIST, have been working to standardize the use of ECC.

For the implementation of ECC, finite fields GF(p) and $GF(2^m)$ have been used, where p is prime and m is positive integer. In particular, $GF(2^m)$, which is an m-dimensional extension field of GF(2), is suitable for hardware implementation because there is no carry propagation in arithmetic operations. The function used for this purpose is the scalar multiplication K.P, where K is an integer and P is a point on an elliptic curve.

Recently, Hardware and firmware implementation of ECC over different fields $GF(2^m)$ have been reported in numerous works. Leung et al. [4] Presented a microcoded FPGA-based elliptic curve processor. This design is parameterized for arbitrary key sizes and allows the rapid development of different control flows. They have used a normal basis for the Galois field operations, and the point multiplication can be computed in 14.3 ms for $GF(2^{281})$. Morales-Sandoval and Feregrino-Uribe [5] proposed a hardware architecture that can perform three different ECC algorithms. The main functional units in their cryptosystem are coprocessor for scalar multiplication, random number generator, algorithm units, and main controller. Its scalar multiplication can be computed in 4.7 ms for $GF(2^{191})$. Orland and Paar [6] designed a reconfigurable elliptic curve processor over $GF(2^{167})$, the processor consists of main controller and arithmetic units. Chang Hoon et al [7] described an FPGA implementation of high performance ECC processor over $GF(2^{163})$. The proposed architecture is based on Lopez- Dahab elliptic curve point multiplication algorithm and Gaussian normal basis for $GF(2^{163})$ and drive parallelized elliptic curve in point doubling and point addition algorithms with uniform addressing. Dan Young-ping et al, proposed a parallel hardware processor to compute elliptic curve scalar multiplication in polynomial basis representation over $GF(2^{163})$ [8]. Bednara et al [9] designed an FPGA-based cryptographic processor architecture that allows using multiple squares, adders and multipliers. They are looking for a hybrid coordinate representation in affine projective Jacobian and Lopez-Dahab form. Two prototypes were synthesized for $GF(2^{191})$. In Ref [10], Cheung et al proposed an ECC design for various field operations, which is, however, not optimized for fixed field. The implementations of ECC in an integrated circuit (ASIC), are presented in works [11] and [12]. In [13], we described the coupled FPGA/ASIC implementation of elliptic curve crypto-processor over $GF(2^{163})$. In our scheme, we have developed the arithmetic unit over the finite field GF (2^{163}) and the elliptic curve operations. We have provided a comparison with some ECC hardware implementations in terms of occupation (Slices) and performances. The proposed ECC implementation outperforms all other implementations used for comparative purposes.

In this work we present the results of the implementation of generic ECC Algorithms over $GF(2^m)$ with sizes (m) ranging from 113 to 256 bits. Different algorithms of ECC are described in VHDL language and synthesised using Synopsys. In our design we selected an ASIC CMOS 120 nm technology for the implementation of the ECC processor.

The rest of the paper is organized as follows: the elliptic curve arithmetic hierarchy is described in section 2; section 3 presents the ASIC Implementation of the finite field arithmetic design. In section 4, the results of the implementation of the ECC arithmetic based on projective systems are discussed. The implementation of proposed ECC key generation processor is described. Section 6 concludes the paper.

2. ECC ARITHMETIC HIERARCHY

Elliptic curves have an algebra that allows the manipulation of points along a curve in controlled manners [14]. Point addition takes two points on the curve and constructs another one. By subtracting one among the original points from the sum, this could lead to a computation of another original point. Point doubling takes a single point and computes the addition of the point to itself. Finally, point multiplication combines the two point operations and allows us to multiply a scalar integer against a point. An elliptic curve, defined over GF (2^m) where m is a prime, is the set of solution points (x, y) to an equation of the form:

$$y^2 + x.y = x^3 + a.x^2 + b \qquad \textbf{(Eq. 1)}$$

With a, b $\in$ GF(2^m).

The set of points on an elliptic curve, together with a special point called the point of infinity, formed an abelian group structure by the following operations. The first one, the point addition operation is given by: Let P=(x_1,y_1) and Q=(x2,y2) $\in$ GF(2^m), the point addition P+Q= R(x_3,y_3), with:

$$x_3 = \lambda_2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda.\,(x_1 + x_3) + y_1 + x_3 \qquad \textbf{(Eq. 2)}$$

$$\lambda = (y_2 + y_1)/(x_2 + x_1)$$

The second operation is the point doubling operation R(x_3,y_3) = 2*P with:

$$x_3 = \lambda^2 + \lambda + a$$

$$y_3 = x_1^2 + (\lambda + 1).\, x_3 \qquad \textbf{(Eq. 3)}$$

$$\lambda = x_1 + y_1/x_1$$

The ECC security is based on the discrete logarithm problem, called the Elliptic Curve Discrete Logarithm Problem (ECDLP). Thus, a cryptosystem could be built using this approach. The ECDLP consists of giving two points P, Q $\in$ E (GF (2^m)), to find the positive integer k such as Q = k*P. On the contrary, knowing the scalar k and the point P, the operation k*P is relatively easy to compute [15]. The hierarchy of an Elliptic Curve Point Multiplication is depicted in fig.1.

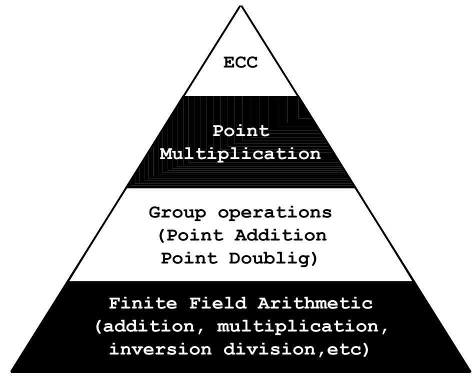


Fig.1 Elliptic curve hierarchy

3. ASIC IMPLEMENTATION OF FINITE FIELD ARITHMETIC

3.1 Finite field description

Finite field GF(2^m) arithmetic is fundamental to the implementation of a number of modern cryptographic systems [16]. The finite field arithmetic operations have been widely used in the areas of data communication and network security applications. Most arithmetic operations needed for security applications, such as exponentiation, inversion, division and multiplication.

In hardware field, elements can be easily implemented as a bit vector, which makes this kind of finite fields interesting for hardware implementations. In this section, we present the hardware implementation of the finite field operations.

3.1.1 Modular multiplication Implementation

In finite field GF(2^m), the operation of multiplication can be carried out by multiplying two elements of this field A(x) and B(x) and then performing reduction modulo P(x) or alternatively by interleaving multiplication and reduction, the multiplication is shown as follows:

$$(b(x)a_{m-1}x^{m-1} + \ldots + b(x)a_2x + b(x)a_0) \bmod P(x). \qquad \textbf{(Eq. 4)}$$

For the implementation of the modular multiplication, many algorithms are proposed. In this section, we will describe three modular multiplications methods in GF(2^m).

3.1.1.1 Cellular Automata Multiplier

In this subsection, we briefly discuss the properties of Programmable Cellular Automata (PCA). And we will study the modular multiplications methods in GF(2^m) using cellular automata

A. *Programmable cellular automata*

The Programmable cellular automata (PCA) [17], is a one dimensional CA whose the state transition rule is not fixed for each cell, but switched by control signals. So, different functions can be generated depending on the value of these signals. In fig.2, we present a standard 3-neighbrhood PCA with non-complemented additive rules. Using a cell structure like this, all possible additive rules can be achieved. The combinations of the control signals of Cl, Cm and Cr

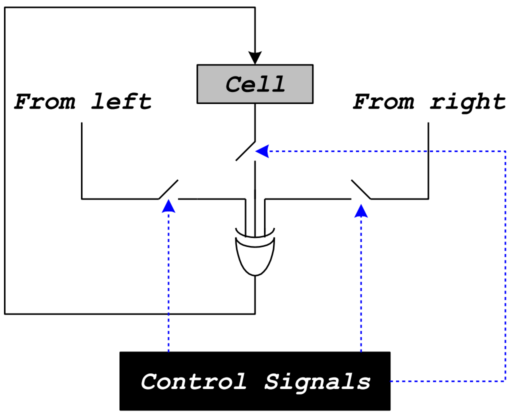


Fig.2 A 3-neighborhood PCA

B. PCA Multiplier

In [18], H. Li and C.N Zhang present a low complexity programmable cellular automata based versatile modular multiplier in GF(2^m). The algorithm of the multiplication is shown in fig.3.

Input: A(x), B(x), P(x)
Output: Z = A. B mod P(x)
(1)Reset PCA
(2)Config Coefficients of B(x) as Cm, and Coefficients of P(x) as Cr
(3)Run PCA m clock cycle

Fig.3 PCA based modular multiplication algorithm

In fig.4, we present the general architecture of the serial multiplier based on PCA in GF(2^m).

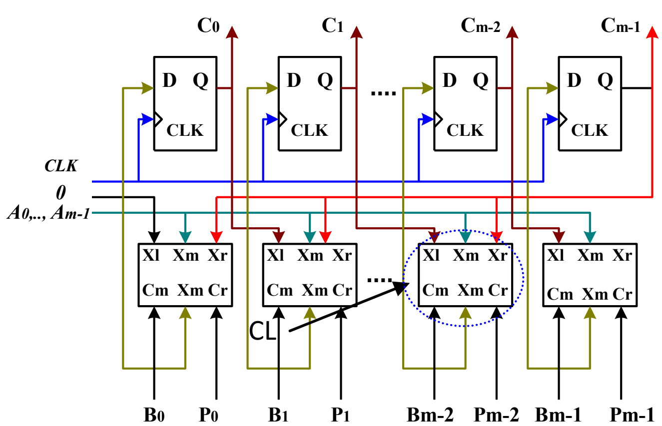


Fig.4 PCA Multiplier architecture in GF(2m)

According to fig.4, we noticed the architecture of the serial multiplier consist of a logical block (LB) of m combination logic (CL) and m bascules. The execution time to compute the complete modular multiplication in GF(2^m) with this architecture is equal of m x T, where T is the critical time of this architecture [31].

3.1.1.2 Interleaved multiplication

The idea of interleaved modular multiplication is very simple: the first operand is multiplied with the second operand bitwise and added to the intermediate result. The intermediate result is reduced with respect to the modulus. For this purpose two subtractions per iteration are required. The pseudo code implementation of interleaved modular multiplication is shown in fig.5.

Input: X, Y, M with $0 \le X, Y \le M$
Output: Z = X . Y mod M
n: number of bits of X
x_i: i^{th} bit of X
Z = 0
for (i = n–1; i $\ge$ 0; i = i–1) loop
Z = 2.Z;
I = x_i. Y;
Z= Z+ I;
if (Z $\ge$ M) Z = Z – M
if (Z $\ge$ M) Z = Z – M
end loop;

Fig.5 Interleaved multiplication algorithm

3.1.1.3 Montgomery Modular Multiplication

The Montgomery modular multiplication algorithm was designed to avoid division in modular multiplications. Given two n-bit inputs, X and Y, this algorithm gives Z = X.Y. R^{-1} mod M, where R equals to 2^m and M is the m-bit modulo. Fig.6, gives a pseudo code implementation of Montgomery modular multiplication.

Input: X, Y < M < 2^n with $2^{n-1} < M < 2^n$ and M = 2t + 1;
with $t \in n$
Output: Z = X.Y. 2^{-n} mod M
Z = 0
for (i = 0; i < n; i++) loop
Z = P + x_i. *Y;*
If (z_0 = 1) P = P + M;
Z = Z div 2;
end loop
if (Z $\ge$ M) Z = Z – M;

Fig.6 Montgomery multiplication algorithm

3.1.2 Inversion in Galois Field

In finite field GF(2^m), the inversion is a complex operation that is computed only once in a k.P operation. To calculate the multiplicative inverse operation for an element $A \in GF(2^m)$, Extended Euclidean Theorem can be applied [19]. The hardware architecture related to this operation is presented in Fig.7.

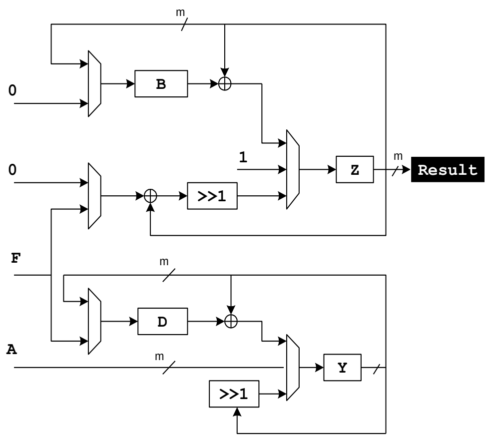


Fig.7 Hardware implementation of an Inverter in GF (2^m)

3.1.3 Division in Galois Field

Typically, in GF(2^m) the division x/y is implemented as two consecutive operations, the inversion y^{-1} and then the multiplication x. y^{-1}. There are well known algorithms for field inversion (see. Section 3.1.2), like The Modified Almost Inversion Algorithm, the Fermat's theorem or the Ito-Tsujii algorithm [21].

The algorithm proposed by S. C. Shantz [22] shown in fig.8 can perform a direct division x/y mod F in at most 2m-2 clock cycles. That is, this algorithm requires almost the same time to compute a single inversion but saves the additional time for the field multiplication in the operation x. y^{-1}.

Input: X1(x); Y1(x) $\in$ GF(2^m), X1(x) $\neq$ 0 and F(x) the irreducible polynomial of degree m
Output: U(x) = Y1(x)/X1(x)mod F(x)
$A(x) \longleftarrow X1(x)$
$B(x) \longleftarrow F(x)$
$U(x) \longleftarrow Y1(x)$
$V(x) \longleftarrow 0$
while A(x) $\neq$ B(x) do
if x divides to A(x) then
$A(x) \longleftarrow A(x)x^{-1}$
$U(x) \longleftarrow U(x)x^{-1} \bmod F(x)$
Else
if x divides to B(x) then
$B(x) \longleftarrow B(x)x^{-1}$
$V(x) \longleftarrow V(x)x^{-1} \bmod F(x)$
Else
if grade of A(x) is greater than grade of B(x) then
$A(x) \longleftarrow (A(x) + B(x))x^{-1}$
$U(x) \longleftarrow (U(x) + V(x))x^{-1} \bmod F(x)$
else
$B(x) \longleftarrow (A(x) + B(x))x^{-1}$
$V(x) \longleftarrow (U(x) + V(x))x^{-1} \bmod F(x)$
end if
end if
end if
end while

Fig.8 Division algorithm in GF(2^m)

3.2 ASIC Implementation of Finite Field

3.2.1 Modular Multiplication Implementation

We prototyped the modular multiplication methods on an ASIC CMOS 120 nm technology. The architectures were described using VHDL language. These modules were simulated using Active-HDL and synthesized using Synopsys Design Compiler. Synthesis results are shown in table.1. In this section, three criteria's are described: the area occupation (mm²), the static power consumption (mW) and the frequency (MHz). In order to testing the sensitivity of the different architectures in function of the number m, in our implementation we selected 4 values for m: 32, 64, 128 and 256.

Table.1 Modular multiplication performances

Performances	Size	CA	Interleaved	Montgo
Area (Slices)	32	06245	06085	06241
	64	09908	11483	12038
	128	17434	22507	23669
	256	31942	44553	46895
Dynamic Power (m W)	32	1,75	2,13	01.75
	64	2,56	3,40	03.81
	128	3,78	4,90	07.54
	256	4,74	7,61	13.91
Frequency (MHz)	32	769	769	1250
	64	666	625	714
	128	555	454	666
	256	370	357	625

In table.2, we present the comparison of our implementation results for the cellular automata multiplication method and the sum results published for the implementations of the modular multiplication methods. According to table.2, our proprietary implementation of the modular multiplication is faster, it has the best time was by about 0.44 μs in GF(2^{163}) and 0.65 μs in GF(2^{233}). Also, our result has the best area 21337 slices in GF(2^{163}).

3.2.2 Inversion and division Implementation

In table.3, we present the performances of the extended Euclidean algorithm for the inversion and division operations over GF(2^m), in table.3 three criteria's are described the Area (mm²) and the dynamic power (mW) and frequency.

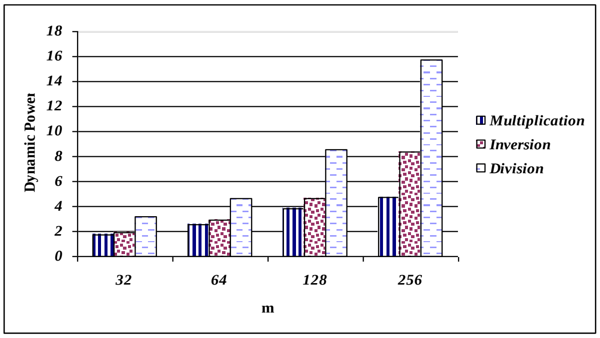
According to table.3, we noticed the important consumption of the inversion and division operations in term of area and power compared with the multiplication operation. Then inversion presents the least area occupation. But, the division presents the least dynamic power consumption.

Table.3 Inversion and division performances

Operation	Size	Frequency	Slices	Power (mw)
Inversion	32	555	12399	01.90
	64	500	23286	01.90
	128	384	44719	04.62
	256	357	87530	08.38
Division	32	833	09184	03.19
	64	625	17198	04.64
	128	588	33539	08.55
	256	555	66095	15.76

3.2.3 Implementations comparison

In this section, we describe the finite field implementations comparison in term of the total area occupation and the power consumption. In fig.9, we present the histograms of the performance for different size of integer m.



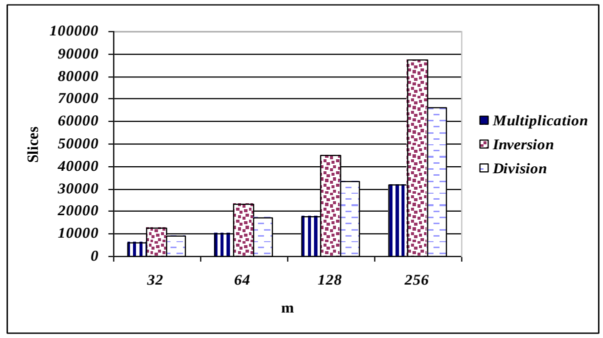


Fig.9 Finite field implementations comparison

According to fig.9, we noticed the high power consumption of the inverter and division operations compared with the multiplication. Hence, projective coordinates are used instead of the inversion operation, which is hard to compute and implement on hardware device. In order to eliminate the inversion operation, in the rest of the paper, we will devote to implement the elliptic curve processor on the projective coordinate.

4. IMPLEMENTATION OF THE ECC ARITHMETIC BASED ON PROJECTIVE SYSTEMS

Elliptic curves have an algebra that allows the manipulation of points along a curve in controlled manners [22]. Point addition takes two points on the curve and constructs another one. By subtracting one among the original points from the sum, this could lead to a computation of another original point. Point doubling takes a single point and computes what would amount to the addition of the point to itself. Finally, point multiplication combines the two point operations and allows us to multiply a point integer. In this section, we present the implementation of the elliptic curve arithmetic based on projective coordinate. In the next subsection, we describe the projective systems.

4.1 Projective coordinate

Compared to field multiplication in affine coordinates, inversion is by far the most expensive basic arithmetic operation in GF (2^m). Inversion can be avoided by means of projective coordinate [23] representation. A point P in projective coordinates is represented using three coordinates X, Y, and Z. This representation greatly helps to reduce internal computational operations. It is customary to convert the point P back from projective to affine coordinates in the final step. To get projective equation of an elliptic curve (E), a transformation on Equation (1) must be performed. It consists of multiplying this equation by a power of Z to clear the denominator. Each form of projective systems developed has positive and negative aspects for practical issues. The different projective coordinate systems are derived by substituting the affine coordinates (x, y) by (X/Z^c, Y/Z^d) with d, c being constants [24]. The formulas for adding two distinct points and for doubling a point appear to be different. However, the doubling operation can be rewritten in terms of an addition operation. To compute the point multiplication, we experimented with two different point multiplication algorithms.

In this subsection, three proposed projective coordinate designs serving for computing the elliptic curve point multiplication are presented and developed: Jacobian, Lopez & Dahab and Montgomery projective coordinates [25]. We experimented with different point multiplication algorithms: the double and add algorithm is used to perform point multiplication. While, for projective Montgomery method, a special double and add algorithm is applied.

Let P=(X1, Y1, Z1) and Q=(X2, Y2, Z2) be points of an elliptic curve E, the addition point P+Q=(X3, Y3, Z3), the doubling of point P1 is 2*P1= (X3, Y3, Z3). We show the concrete algorithms for computing point addition and doubling for each method. In this section are discussed various ways for making indistinguishable the addition formula on elliptic curves.

4.1.1 Jacobian projective coordinates

In the Jacobian projective coordinates system (c=2, d=3), a standard point is represented by means of three variables. A projective point P=(X, Y, Z) on the curve satisfies the next equation:

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6 \quad \textbf{(Eq. 5)}$$

The addition and doubling Elliptic curve arithmetic operations can be performed as shown in algorithm 5 and 6.

Input:$P = (X_1,Y_1,Z_1), Q = (X_2,Y_2,Z_2) \in E(GF(2^m))$
Output: $P(x_3,Y_3,Z_3) = P + Q$
1. $W = X_1 + X_2. Z_1^2$
2. $R = Y_1 + Y_2. Z_1^3$
3. $Z_3 = Z_1.W$
4. $T = R + Z_3$
5. $X_3 = a. Z_3^2 + R.T + W^3$
6. $Y_3 = T.X_3 + W^2 [R.X_1 + W.Y_1]$

Fig.10 Jacobian point addition method.

Input: $P = (X_1, Y_1, Z_1) \in E(GF(2^m))$, c such that $c^2 = b$
Output: $P(X_3,Y_3,Z_3) = 2*P$
1. $X_3 = X_1^4 + b. Z_1^8$
2. $Z_3 = X_1. Z_1^2$
3. $U = X_1^2 + Z_1.Y_1 + Z_3$
4. $Y_3 = U.X_3 + Z_3. X_1^4$

Fig.11 Jacobian point addition method.

4.1.2 Lopez & Dahab coordinates

In the Lopez & Dahab coordinates system (c=1, d=2), A projective point P=(X, Y, Z) on the curve satisfies the next equation:

$$Y^2+XYZ = X^3Z + a X^2Z^2 + bZ^4 \quad (\textbf{Eq. 6})$$

The addition and doubling Elliptic curve arithmetic operations can be efficiently implemented as illustrated in algorithms 7 and 8.

Input: $P = (X_1, Z_1), Q = (X_2 , Z_2) \in E(GF(2^m))$
Output: $P(X_3,Y_3,Z_3) = P + Q$
1. $A = Y_1 + Y_2. Z_1^2$
2. $B = X_1 + X_2.Z_1$
3. $C = Z_1 .B$
4. $D = B^2.C$
5. $E = A.C$
6. $Z_3 = A^2 + D + E$
7. $X_3 = E.(X_3 + X_2.Z_3) + Z_3.(X_3 + Y_2.Z_3)$
8. $Y_3 = C^2$

Fig.12 Lopez & Dahab point addition method.

Input: $P = (X_1, Y_1, Z_1) \in E(GF(2^m))$, c such that $c^2 = b$
Output: $P(X_3,Y_3,Z_3) = 2*P$
1. $A = Z_1^2$
2. $B = b. Z_1^4 = (c.A)^2$
3. $C = X_1^2$
4. $D = C^2$
5. $X_3 = D + B$
6. $Y_3 = X_3.(Y_1^2 + a.Z_3 + B)\ \ B.Z_3$
7. $Z3 = X_1^2 . Z_1^2 = A.C$

Fig.13 Lopez & Dahab point doubling method.

4.1.3 Montgomery method

Let us consider the points $P(X_1,Y_1,Z_1)$, $R(X_2,Y_2,Z_2)$, $Q(X_3,Y_3,Z_3)$, belonging to the curve $E(GF(2^m))$, where R = 2 * P and Q = P+R, the computations become, through the use of Montgomery method respectively as follows:

Input: $P(X_1,Z_1),Q(X_2,Z_2) \in E(GF(2^m))$
Output: $Q(X_3,Z_3) = P + Q$
1. $M = (X_1 \cdot Z_2) + (Z_1 \cdot X_2)$
2. $Z_3 = M^2$
3. $N = (X_1 \cdot Z_2) \cdot (Z_1 \cdot X_2)$
4. $M = x \cdot Z_3$
5. $X_3 = M + N$

Fig.14 Montgomery point addition method.

Input: $P(X_1,Z_1) \in E (GF(2^m))$, c such that $c^2 = b$
Output: $R(X_3,Z_3) = 2* P$
1. $T = X_1^2$
2. $M = c \cdot Z_1^2$
3. $Z_3 = T \cdot Z_1^2$
4. $M = M^2$
5. $T = T^2$
6. $X_3 = T +M$

Fig.15 Montgomery point doubling method.

4.2 ASIC Implementation of Elliptic Curve Operations

In fig.16, the total area occupations (mm²) of the ECC operations (point addition and point doubling) are presented. In fig.16, are reported the results obtained for the three projective coordinate systems.

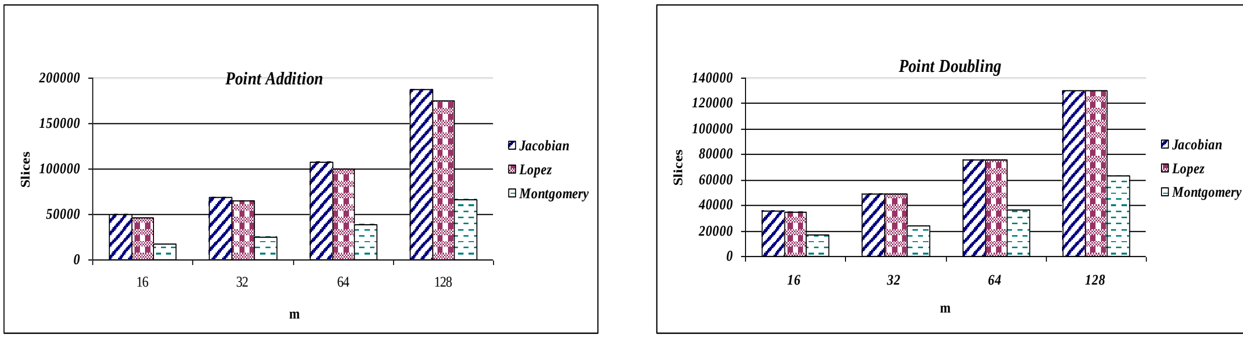


Fig.16. Area occupation comparison histograms.

In fig.17, we present the dynamic power consumption of the ECC arithmetic in different projective coordinate systems.

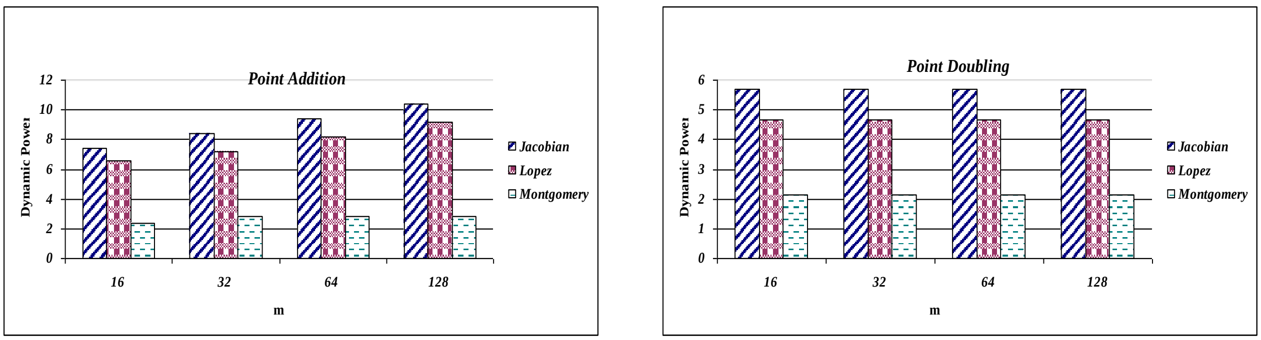


Fig. 17. Power consumption comparison histograms.

According to fig.16 and fig.17, as can be noticed, the Montgomery method, based on projective coordinates, is the best for elliptic curve arithmetic over GF (2^m) in term of area occupation and power consumption. In the next of this work, we selected the Montgomery method for the implementation of the ECC processor over GF(2^m).

5. IMPLEMENTATION OF THE ECC PROCESSOR

5.1 Elliptic Curve Point Multiplication

There are a variety of procedures allowing to accomplish point multiplication, the most basic being the double and add method. It is essentially the square and multiply technique for exponentiation converted to point multiplication [26].

Input: $k = (k_{n-1},k_{n-2}...,k_1,k_0)2$ with $k_{n-1} = 1$, $P(X_1,Z_1) \in E$ (GF(2^m))
Output: Q = k*P
Procedure:
1. $P_1 \leftarrow P$; $P_2 \leftarrow 2*P$
2. For i from $n - 2$ downto 0 do
3. if (k_i = 1) then
4. $P_1 \leftarrow P_1 + P_2$; $P_2 \leftarrow 2*P_2$
5. else
6. $P_2 \leftarrow P_2 + P_1$; $P_1 \leftarrow 2*P_1$
7. end of
8. end for
9. Return P_1

end algorithm.

The Elliptic curve point multiplication kP, where k is an integer and P is a point on the curve, is a fundamental operation in elliptic curve cryptosystems. It is defined as adding a point to itself a set number of times.

For the implementation of elliptic curve point multiplication, many methods are proposed. In our scheme, we selected the Montgomery method for the implementation of the point multiplication (see fig.18).

As mentioned above, we will study the scalar multiplication method based on Montgomery algorithm. The main advantages of this algorithm are: it does not have any extra storage requirements; the same operations are performed in every iteration of the main loop, thereby potentially increasing resistance of timing attacks and power analysis attacks. The algorithm is shown below [13].

According to fig.18, we noticed the Montgomery method is based on the formulas for doubling and addition (steps 4 and 6). In the next subsection, we describe the proposed architecture for the implementation of elliptic curve point multiplication over finite field GF(2^m).

5.2 Proposed processor architecture

The main units of the proposed Elliptic Curve Point Multiplication processor are shown in fig.19, including the input and the output interfaces for storing the input and the output data implemented as FIFOs. The control module consists of a finite state machine description. It generates the control signals for the initialization operations of finite field, the point addition and point doubling operations, and the conversion to affine coordinates operations, relying on the key values by the Montgomery Algorithm. The elliptic curve operator is formed by the point addition and the point doubling modules. Finally the arithmetic and logical unit (ALU) allow parallel execution of finite field addition, inversion a

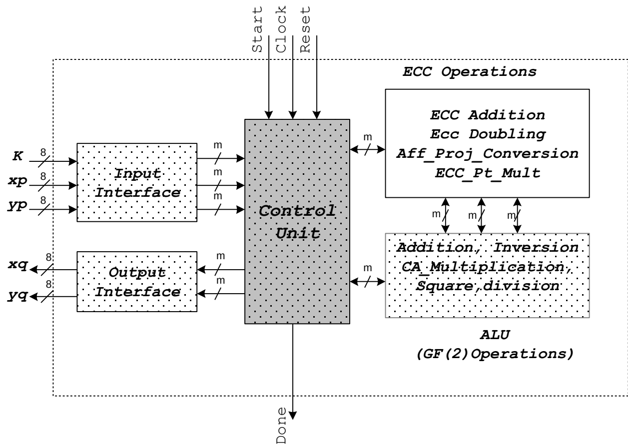


Fig.19 Proposed elliptic curve pro ∈ cessor

5.3 ASIC implementation of elliptic curve processor

We designed an ECC IP using the VHDL language and synthesized using Synopsys Design Compiler. In table.4, we present the implementation results of the ellpitic curve point multiplication over GF(2^m) , where $m \in \{113, 163, 191, 233, 256\}$. In table.4, three criteria's are described: the area occupation (mm²), the static power consumption (mW) and the frequency (MHz).

Table.4 The ECC processor performances

Size	Frequency (MHz)	Area (Slices)	Power (mw)
113	377	135420	12.65
163	370	207577	17.52
191	357	224808	19.76
233	333	275482	22.26
256	312	323449	23.10

5.4 Layout of ECC Processor

The resulting netlist of the ECC processor over GF(2256) is used as input to Cadence in order to perform mapping and routing with a 120 nm CMOS technology. The results obtained from these operations are reported in table.5 and fig.20. The final ASIC has been implemented using CMOS 120 nm technology.

The result in synthesis operating frequency is about 312 Mhz. The ECC processor areas are about 1.29 mm2. The total Input/output is equal to 44. The core dimension of the ECC processor is about 0.74 mm x 0.74 mm, and the core is about 0.55 mm2. The proposed ECC implementation provides a time of 0.85 ms over GF(2^{256}) and 0.29 ms over GF(2^{163}).

Table.5 ASIC implementation of ECC processor

.Core dimension	: 0.74 mm x 0.74 mm
Core area	: 0.55 mm2
Circuit dimension	: 1.36 mm x 1.36 mm
Circuit area	: 1.29 mm2
Total In/out	: 44

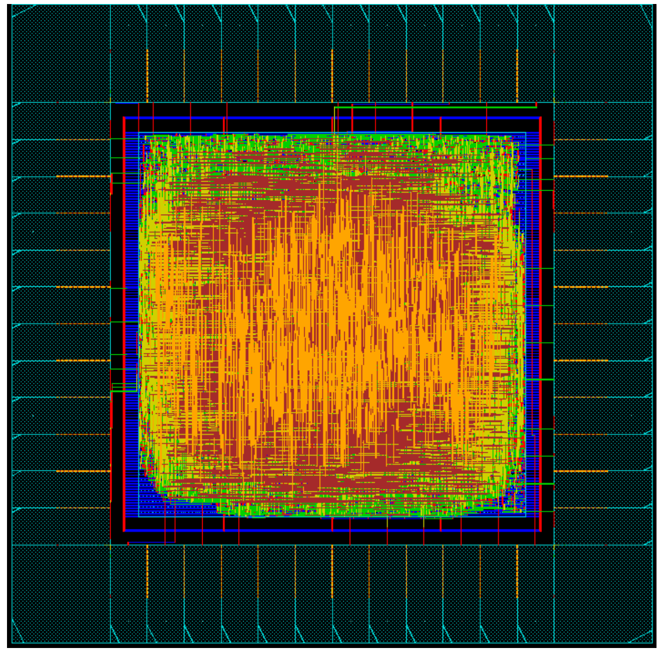


Fig.20 Layout of the design over GF(2^{256})

6. CONCLUSION

In this paper, we proposed a high performance of the generic elliptic curve key generation processor over GF(2m) scheme based on the Montgomery scalar multiplication algorithm. The proposed processor is performed using polynomial basis. The Finite Field operations use a cellular automata multiplier and Extended Euclidean Theorem for inversion. The elliptic curve arithmetic based on projective systems is used to compute the point multiplication in GF (2m). Our presented elliptic Curve arithmetic architectures improved point addition and point doubling for speed, low-power and less-Area applications. Finally, a completely parameterized processor of VHDL Elliptic Curve point multiplication was developed and tested. The second part of the paper, present's the first design of the ECC processor using a 120 nm CMOS technology. The ASIC area is about 1.29 mm2. This processor operates with clock frequency of 312 Mhz and provides a time of 0.85 ms over GF(2^{256}).

7. REFERENCES

[1] N. Koblitz, "Elliptic curve cryptosystems", Mathematics of Computation, number 48, pages 203-209, 1987.

[2] V.S. Miller, "Use of elliptic curve in cryptography", Advances in Cryptology– Proceedings of CRYPTO'85, Springer Verlag Lecture Notes in Computer Science 218, pages 417-426, 1986.

[3] Certicom research, "The Elliptic Curve Cryptosystem", Certicom, April 1997.

[4] K.H. Leung et al., FPGA implementation of a microcoded elliptic curve cryptographic processor, IEEE Symposium on Field Programmable Custom Computing Machines, 2000, pp 68-76.

[5] M.Morales-Sandoval, C.Feregrino-Uribe, on the hardware design of an elliptic curve cryptosystem, Proceeding of the 5th Mexican International Conference in Computer Science, 2004, pp60-70.

[6] G.Orlando, C.Paar, A high performance reconfigurable elliptic curve processor for GF(2m), Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000), pp 41-56.

[7] Chang Hoon Kim, Soonhak kown and Chun Pyo Hong, FPGA implementation of high performance ECC processor over GF(2163), Journal of Systems Architecture, Vol 54(, pp 893-900, 2008.

[8] Dan Young-ping, Zou Xue-cheng, Han Yu and Yi Li-hua, Design of highly efficient elliptic curve crypto-processor with two multiplications over $GF(2^{163})$, The journal of china Universities of Posts and Telecommunications, Vol 16(2), pp 72-79, 2009.

[9] M Bednara, M Daldrup, J von zur Gathen and J Shokrollahi, Reconfigurable implementation of elliptic curve crypto algorithms. Reconfigurable Architectures Workshop, 16th International Parallel and Distributed Processing Sympsium, April 2002.

[10] Cheung R C C, Telle N J, Luk W, et al, Customizable elliptic curve cryptosystems, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol 13 (2), pp 1048-1059, 2005.

[11] Sakyama K, Batina L, Preneel B, et al, Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF(2^n), IEEE Transactions on Computers, vol 56 (9), pp 1269-1282, 2007.

[12] Sozzana F, Bertoni G, S Turcato, et al, A parallelized design for an elliptic curve cryptosystem coprocessor, Proceeding of the International Conference on Information Technology, IEEE Computer Society, pp 626-630, 2005.

[13] Mohsen Machhout, Zied Guitouni, Kholdoun Torki, Lazhar Khriji and Rached Tourki, Coupled FPGA/ASIC Implementation of Elliptic Curve Crypto-Processor, IJNSA International Journal of Network Security & Its Applications, Vol.2 No.3, Juillet 2010.

[14] U.S. Department of Commerce, National Institute of Standards and Technology, Digital Signature Standard (DSS), Federal Information Processing Standards Publication FIPS PUB 186-2, January 2000.

[15] T. Izu1, B. Moller, and T. Takagi, "Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks", Progress in Cryptology – INDOCRYPT 2002,. Springer-Verlag LNCS 2551, 2002, pp. 296–313.

[16] Hyun-Sung Kim1 and Kee-Young Yoo, Multiplier for Public-Key Cryptosystem Based on Cellular Automata, MMM-ACNS 2003, LNCS 2776, pp. 436–439, 2003. Springer

[17] Jun-Cheol Jeon, Kee-Won Kim et all, Cellular Automata Architecture for Elliptic Curve Cryptographic Hardware, ICCS 2006, Part III, LNCS 3993, pp. 329 – 336, 2006, Springer.

[18] H. Li and C.N Zhang, "Efficient cellular automata versatile multiplier for $GF(2^n)$", ttp://www.iis.sinica.edu.tw /JISE/2002/ 2002 07_01.pdf.

[19] A. Daly, W. Maranane, T. Kerins and E. Popocivi, "Fast Modular Division for Application in ECC on Reconfiguration Logic", Field Programmable Logic and application, 13th International Conference, (FPL '03),2003, pp. 786-795.

[20] D. Hankerson, L. Lopez, and A. Menezes, Software Implementation of Elliptic Curve Cryptography Over Binary Fields, in Proc. of the Second International Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000, volume 1965 of Lecture Notes in Computer Science, pp. 1{24,Worcester, MA, August 2000, Springer.

[21] S. C. Shantz, From Euclid's GCD to Montgomery Multiplication to the Great Divide, Technical Report TR-2001-95, Sun Microsystems Laboratories, 2001.

[22] M. Morales-Sandoval, "Hardware architecture for Elliptic Curve Cryptography and Lossless Data Compression", Computer Science Department National Institute for Astrophisics, Optics and Electronics Tonantzintla. Puebla México, December 2004.

[23] M. Dion. "Implantation d'ECDSA sur une Carte à Puce".Université de Montréal, Département d'informatique et de Recherche Opérationnelle. Mai 1999.

[24] E.Oswald, "Introduction to elliptic curve Cryptography", Institue for Applied information Processing and communication, July 2005. 2

[25] Sining Liu, Francis Bowen, Brian King, and Wei Wang, " Elliptic curve Cryptosystem implementation Based on a look-Up Table sharing Scheme", In Proc. IEEE International Symposium on Circuits and Systems (ISCAS'06), 2006, pp. 4.

[26] Dupont L. Roy, S. Chouinard, J.Y. , «A FPGA Implementation of an Elliptic Curve Cryptosystem", In Proc. IEEE International Symposium on Circuits and Systems (ISCAS'06), 2006, pp. 4.

[27] B. Ansari, M. Anwar Hasan, High performance architecture of elliptic curve scalar multiplication, Tech. Report CACR2006-01, 2006.

[28] F. Sozzani, G. Bertoni, S. Turcato, L. Breveglieri, A parallelized design for an elliptic curve cryptosystem coprocessor, in: Symposium on Information Technology: Coding and Computing (ITCC), 1, 2005, pp. 626–630.

[29] A. K. Daneshbeh, M.A. Hasan, Area efficient high speed elliptic curve cryptoprocessor for random curves, in: IEEE Symposium on Information Technology: Coding and Computing (ITCC), 2, 2004, pp. 588–592.

[30] A. Satoh, K. Takano, A scalable dual-field elliptic curve cryptographic processor, IEEE Transactions Computers 52 (4) (2003) 449–460.

[31] Z. Guitouni1, R. Chotin-Avot, M. Machhout, H. Mehrez and R. Tourki, Design and FPGA implementation of modular multiplication methods using cellular automata, IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era, (DTIS'10).