Hardware Implementation of Elliptic Curve Point Multiplication over $GF(2^m)$ for ECC protocols

Moncef Amara
University of Paris 8
LAGA laboratory
Saint-Denis / France

Amar Siad
University of Paris 8
LAGA laboratory
Saint-Denis / France

Abstract

The Elliptic Curve Cryptography covers all relevant asymmetric cryptographic primitives like digital signatures and key agreement algorithms. In the present work, we develop a design of elliptic curve operations over binary Fields $GF(2^m)$. The function used for this purpose is the scalar multiplication kP which is the core operation of ECCs. Where k is an integer and P is a point on an elliptic curve. The EC Point multiplication processor defined in affine coordinates is achieved by using a dedicated Galois Field arithmetic implemented on FPGA using VHDL language.

1. Introduction

Elliptic Curve Cryptography (ECC) is a relatively new crypto-system, suggested independently, from the second half of 19 th century, by Neals Koblitz [4] and Victor Miller [8]. At present, ECC has been commercially accepted, and has also been adopted by many standardizing bodies such as ANSI, IEEE [3], ISO and NIST [1]. Since then, it has been the focus of a lot of attention and gained great popularity due to the same level of security they provide with much smaller key sizes than conventional public key crypto-systems have.

The ECC covers all relevant asymmetric cryptographic primitives like digital signatures (ECDSA), key exchange and agreement protocols. Point multiplication serves as the basic building block in all ECC primitives and is the computationally most expensive operation.

The aim of this work is to develop an EC point multiplication processor, intended to the conception of the cryptographic applications, like digital signatures (ECDSA) and key agreement (Diffie-Hellman) protocols.

The paper is organized as follows. After a brief introduction, an overview of the use of elliptic curve in cryptography application is given in section 2. We present in Section 3, mathematical background on elliptic curve over finite-field $GF(2^m)$. The point multiplication method is explained in Section 4, and Elliptic Curve Arithmetic's in $GF(2^m)$ based on affine coordinates are presented in Section 5. The EC Point multiplication processor given in Section 6. In Section 7, synthesis results of elliptic curves and finite field arithmetic operations are presented. Finally, conclusion is summarized in Section 8.

2. Elliptic curve cryptography

Elliptic curves (Figure.1), defined over a finite-field provide a group structure that is used to implement the cryptographic schemes. The elements of the group are the rational points on the elliptic curve, together with a special point O (called the "point at infinity").

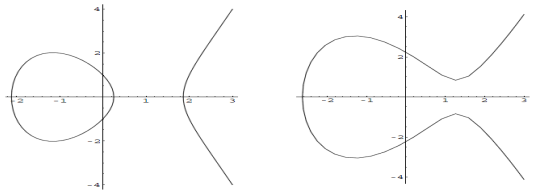

Figure 1. Graphs of elliptic curves $y^2 = x^3 - 4x + 1$ (on the left) and $y^2 = x^3 - 5x + 5$ (on the right) over $\mathbb{R}$ [6].

A major building block of all elliptic curve cryptosystems is the scalar point multiplication, an operation of the form $k.P$ where k is a positive integer and P is a point on the elliptic curve. Computing $k.P$ means adding the point P exactly $k - 1$ times to itself, which results in another point Q on the elliptic curve. The inverse operation, i.e., to recover k when the points P

and $Q = k.P$ are given, is known as the *Elliptic Curve Discrete Logarithm Problem* (ECDLP).

Table 1. Key length for public-key and symmetric-key cryptography.

Symmetric-key	ECC	RSA/DLP
64 bit	128 bit	700 bit
80 bit	160 bit	1024 bit
128 bit	256 bit	2048-3072 bit

To date, no subexponential-time algorithm is known to solve the ECDLP in a properly selected elliptic curve group. This makes Elliptic Curve Cryptography a promising branch of public key cryptography which offers similar security to other "traditional" DLP-based schemes in use today, with smaller key sizes and memory requirements, e.g., 160 bits instead of 1024 bits (as shown in Table 1).

2.1. Diffe-hellman

The Diffie-Hellman protocol is the basic public-key crypto-system proposed for secret key sharing. If A (Alice) and B (Bob) first agree to use a specific curve, field size, and type of mathematics. They then share the secret key by process as follows. We can see that we just need scalar multiplication in order to implement the Diffie-Hellman protocol.

Algorithm 1 *Diffie-Hellman Protocol*

1: A and B each chose random private key k_a and k_b
2: A and B each calculate k_aP and k_bP, and send them to opposite side.
3: A and B both compute the shared secret $Q = k_a(k_bP) = k_b(k_aP)$.

2.2. Elliptic curve digital signature algorithm

EC Digital Signature Algorithm is the elliptic curve analogue of the DSA, this protocol needs not only the elliptic curve operations, such as scalar multiplication, field multiplication and field inverse multiplication, but also integer multiplication, inverse operation, modular operation and a hash function. In the ECDSA, A (Alice) generates the signature with his secret key and B (Bob) verifies the signature with A's public key. Algorithm.2 is the ECDSA protocol which A signs the message m and B verifies A's signature.

Algorithm 2 *ECDSA Protocol*

Key generation : (A)

1: Select a random integer d from $[1, n-1]$.
2: Compute $Q = d.P$.
3: A's public key is Q; A's private key is d.

Signature generation : (A)

1: Select a random integer k from $[1, n-1]$.
2: Compute $k.G = (x_1, y_1)$ and $r = x_1$(mod n).
3: If $r = 0$ then go to step 1.
4: Compute k^{-1}(mod n).
5: Compute $s = k^{-1}(SHA-1(m) + dr)$(mod n).
6: If $s = 0$ then go to step 1.
7: Send m and (r, s), which is A's signature for the message m, to B.

Signature verification : (B)

1: Verify that r and s are integers in $[1, n-1]$.
2: Compute $e = SHA-1(m)$.
3: Compute $w = s^{-1}$(mod n).
4: Compute $u_1 = e * w$(mod n) and $u_2 = r * w$(mod n).
5: Compute $u_1P + u_2Q = (x_1, y_1)$ and $v = x_1$(mod n).
6: If $s = 0$ then go to step 1.
7: Accept the signature if and only if $v = r$.

3. Background mathematics

In this section, an introduction to the abstract algebra and elliptic curves used in this implementation is presented.

3.1. Groups and fields

A group $(G, +)$ consists of a set of numbers G together with an operation $+$ that satisfies the following properties.

1. Associativity:$(a + b) + c = a + (b + c)$ for all $a, b, c \in G$.
2. Identity: there is an element $0 \in G$ such that $a + 0 = 0 + a$ for all $a \in G$.
3. Inverse: for every $a \in G$, there exists an element $-a \in G$ such that $(-a) + a = a + (-a) = 0 \in G$.

A field $(\mathbb{F}, +, \times)$ is a set of numbers F together with two operations and that satisfies the following properties.

1. $(\mathbb{F}, +)$ is an abelian group with identity 0.
2. $(\times)$ is associative.

3. there exists an identity $1 \in F$ with $1 \neq 0$ such that $1 \times a = a \times 1 = a$ for all $a \in F$.
4. the operation $\times$ is distributive over $+$, i.e., $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in F$.
5. $a \times b = b \times a$ for all $a, b \in F$.
6. for every $a \neq 0, a \in F$ there exists an element $a^{-1} \in F$ such that $a^{-1} \times a = a \times a^{-1} = 1$.

If the field has a finite set of elements, it is called a finite (or Galois) field. Numbers in the field $\mathbb{F}_2$ can be represented by $\{0, 1\}$ and numbers in $\mathbb{F}_{2^n}$ can be represented as n-bit binary numbers.

3.2. Elliptic curves over $\mathbb{F}_{2^n}$

In this section, a group operations on elliptic curves over $\mathbb{F}_{2^n}$ is described.

A nonsupersingular elliptic curve E over $\mathbb{F}_{2^n}$, $E(\mathbb{F}_{2^n})$ is the set of all solutions to the following equation [7].

$$y^2 + xy = x^3 + a_2x^2 + a_6 \quad (1)$$

where $a_2, a_6 \in \mathbb{F}_{2^n}$, and $a_6 \neq 0$. Such an elliptic curve is a finite abelian group. The number of points in this group is denoted by $\#(E(\mathbb{F}_{2^n}))$.

Curve addition: If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points on the elliptic curve [i.e., satisfy (1)] and $P \neq -Q$ then $(x_3, y_3) = R = P + Q$ can be defined geometrically (Figure.2). In the case that $P \neq Q$ (i.e., point addition), a line intersecting the curve at points P and Q and must also intersect the curve at a third point $-R = (x_3, -y_3)$). If $P = Q$ (point doubling), the tangent line is used (Figure.2).

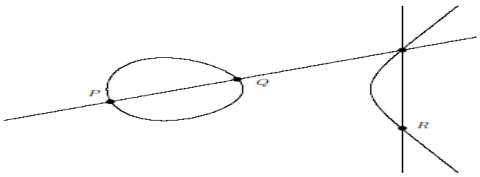

Figure 2. Group law of elliptic curve.

For E given in affine coordinates:
if $P \neq Q$

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \\ \text{où } \lambda &= \tfrac{(y_2+y_1)}{(x_2+x_1)} \end{aligned} \quad (2)$$

if $P = Q$

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a \\ y_3 &= x_1^2 + (\lambda + 1)x_3 \\ \text{où } \lambda &= x_1 + \tfrac{y_1}{x_1} \end{aligned} \quad (3)$$

4. Elliptic curve point multiplication

There are different ways to implement point multiplication: binary, signed digit representation (NAF), Montgomery method. A scalar multiplication is performed in three different stages. At the top level, the method for computing the scalar multiplication must be selected, in the second level, the coordinates to represent elliptic points must be defined. From this representation, the add operation is defined. Possible coordinates are: affine, projective, Jacobeans and L'opez-Dahab. The lower level, but the most important, involves the primitive field operations on which the curve is defined. Basic field operations are sum, multiplication, squaring and division.

4.1. Binary method

The most simplest and straightforward implementation is the binary method (as shown in Algorithm.3 and .4); The binary method scans every bit of scalar k and, depending on its value, 0 or 1, it performs an ECC-DOUBLE operation or both a ECC-DOUBLE and an ECC-ADD operation. Algorithm.3 scans every bit of k from right to left. This allows to perform the operations ECC-DOUBLE and ECC-ADD in parallel.

For an elliptic curve defined on $\mathbb{F}_{2^m}$ using affine coordinates, the operations ECC-ADD and ECC-DOUBLE are performed according to equations (2) and (3) respectively. The operation ECC-ADD requires one inversion, two multiplications, one squaring and eight additions. The operation ECC-DOUBLE requires five additions, two squaring, two multiplications and one inversion, all of them, operations on $\mathbb{F}_{2^m}$.

Algorithm 3 *Binary method: right to left* [7]

Input: $P(x, y)$, $x, y \in GF(2^m)$, $k = (k_{m-1}, k_{m-2}, \ldots, k_0)$
Output: $R = k.P$

```
1: R ← 0
2: S ← P
3: for i ← 0, m − 1 do
4:     if k_i = 1 then
5:         if R = 0 then
6:             R ← S
7:         else
8:             R ← R + S
9:         end if
10:    end if
11:    S ← 2S
12: end for
13: return R
```

Algorithm.4 is a second version of Binary method (left to right).

Algorithm 4 *Binary method: left to right.*

Input: $P(x, y)$,$x, y \in GF(2^m)$,$k = (k_{m-1}, k_{m-2}, \ldots, k_0)$
Output: $R = kP$

1: $R \leftarrow 0$
2: **for** $i \leftarrow m-1, 0$ **do**
3: $\quad R \leftarrow 2R$
4: $\quad$ **if** $k_i = 1$ **then**
5: $\quad\quad R \leftarrow R + P$
6: $\quad$ **end if**
7: **end for**
8: **return** R

5. Field arithmetic over $\mathbb{F}_{2^m}$

The field operations required to implement the elliptic curve group operation are addition, multiplication (squaring) and inverse in $\mathbb{F}_{2^m}$.

5.1. Polynomial basis representation

The standard polynomial basis representation is used for our implementations with the reduction polynomial:

$F(x) = x^m + G(x) = x^m + \sum_{i=0}^{m-1} g_i x^i$

where $g_i \in \{0, 1\}$ for $i = 1, \cdots, m-1$ and $g_0 = 1$

Let α be a root of $F(x)$, then we represent $A \in \mathbb{F}_{2^m}$ in polynomial basis as:

$$A(\alpha) = \sum_{i=0}^{m-1} a_i \alpha^i, a_i \in \mathbb{F}_2 \quad (4)$$

5.2. Addition

$\mathbb{F}_{2^m}$ addition is the simplest of all operations, since it is a bitwise addition in $\mathbb{F}_2$ which maps to an XOR operation $\oplus$ in software or hardware.

$$C \equiv A + B \bmod F(\alpha) \equiv (a_{m-1} \oplus b_{m-1})\alpha^{m-1} + \cdots + (a_1 \oplus b_1)\alpha + (a_0 \oplus b_0)$$

Algorithm 5 : *Addition in* $\mathbb{F}_{2^m}$ [7]

Input: $A(x)$, $B(x)$ two binary polynomial of degrees $m-1$
Output: $C(x) = A(x) + B(x)$.

1: **for** $i \leftarrow 0, m-1$ **do**
2: $\quad C[i] \leftarrow A[i] \oplus B[i]$
3: **end for**
4: **return** C

5.3. Multiplication in $\mathbb{F}_{2^m}$

Right-to-left shift-and-add method: The multiplication of two elements $A, B \in \mathbb{F}_{2^m}$, with
$A(\alpha) = \sum_{i=0}^{m-1} a_i \alpha^i$ and
$B(\alpha) = \sum_{i=0}^{m-1} b_i \alpha^i$ is given as:

$$C(\alpha) = \sum_{i=0}^{2m-2} c_i \alpha^i \equiv A(\alpha) \cdot B(\alpha) \bmod F(\alpha)$$

Algorithm 6 :*Multiplication in* $\mathbb{F}_{2^m}$ *(Right-to-left shift-and-add method)* [7]

Input: $A(x)$,$B(x)$ two binary polynomial of degree $\leq m-1$
Output: $C(x) = A(x) \cdot B(x)$.

1: **if** $a_0 = 1$ **then**
2: $\quad C \leftarrow B$
3: **else**
4: $\quad C \leftarrow 0$
5: **end if**
6: **for** $i \leftarrow 1, m-1$ **do**
7: $\quad b \leftarrow b \cdot x \bmod f(x)$
8: $\quad$ **if** $a_i = 1$ **then**
9: $\quad\quad c \leftarrow c + b$
10: $\quad$ **end if**
11: **end for**
12: **return** C

5.4. Squaring

Since squaring a binary polynomial is a linear operation, it is much faster than multiplying two arbitrary polynomials; i.e., if:

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_2x^2 + a_1x + a_0$$

then

$$A(x)^2 = a_{m-1}x^{2m-2} + \cdots + a_2x^4 + a_1x^2 + a_0$$

The binary representation of $A(x)^2$ is obtained by inserting a 0 bit between consecutive bits of the binary representation of $A(x)$ as shown in Figure.3

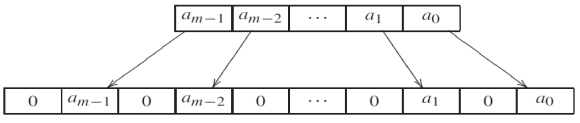

Figure 3. Squaring a binary polynomial

5.5. Inversion

Field inversion is an implementation of the Modified Almost Inversion Algorithm listed in algorithm.7, MAIA is a variant of the Extended Euclidean Algorithm, commonly used to compute inverses in the integer numbers.

This algorithm was selected because it is considered less complex than other variants of the Extended Euclidean Algorithm that can be found in [2].

Algorithm 7 :*Modified Almost Inverse Algorithm: Inversion in* $\mathbb{F}_{2^m}$ [7]

Input: $A(x) \in \mathbb{F}_{2^m}$,$A(x) \neq 0$ and $P(x)$ the irreducible polynomial of degree m
Output: $C(x) = A(x)^{-1} \bmod P(x)$.

1: $B(x) \leftarrow 1$
2: $C(x) \leftarrow 0$
3: $U(x) \leftarrow A(x)$
4: $V(x) \leftarrow P(x)$
5: **loop**
6: **while** $U(0) = 0$ **do**
7: $U(x) \leftarrow U(x)x^{-1}$
8: $B(x) \leftarrow (B(x) + x_0 P(x))x^{-1}$
9: **end while**
10: **if** $U(x) = 1$ **then**
11: **return** $B(x)$
12: **end if**
13: **if** $gradeU(x) < gradeV(x))$ **then**
14: $U(x) \leftarrow V(x)$
15: $C(x) \leftarrow B(x)$
16: **end if**
17: $U(x) \leftarrow U(x) + V(x)$
18: $B(x) \leftarrow B(x) + C(x)$
19: **end loop**

6. Elliptic curve processor

Figure.4 shows a structure of ECC processor. It consists of a main control block, an ECC add and double block and an ECC block for arithmetic operations. The ECC processor we have implemented is defined over the field $GF(2^{163})$, which is a SEC-2 recommendation [9], with this field being defined by the field polynomial $F(x) = x^{163} + x^7 + x^6 + x^3 + 1$. We given also simulation results for ECC processor defined over $GF(2^{233})$.

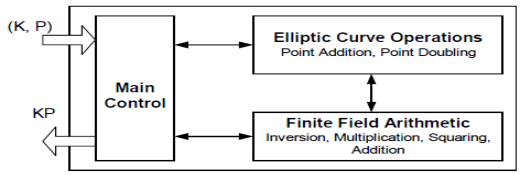

Figure 4. Elliptic curve point multiplication processor.

Figure.5 shows the hardware implementation of point addition operation, corresponding to equation (2).

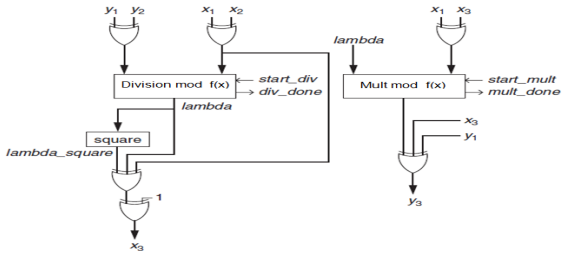

Figure 5. Hardware implementation of point addition operation.

The field arithmetic unit consists of a field serial multiplier and an inverter. The inverter is based on the Modified Almost Inverse Algorithm; this module dominates the time execution in both Add and doubling operations. The serial multiplication is based on a shift and add operation. It can be achieved in m clock cycles.

Bellow, we present hardware implementation of finite field operation in $GF(2^m)$.

6.1. Addition

The addition in the finite field of $GF(2^m)$ is very easy to compute. For the chosen field the addition of two numbers is the simplest operation, since it is only a XOR combination of the bits of two addends. Therefore we need only m XOR gates and one clock cycle for this operation.

6.2. Multiplication

Multiplication in $GF(2^m)$ with polynomial basis representation is defined in section 5.3. Inputs $A = (a_0, a_1, \cdots, a_{m-1})$ and $B = (b_0, b_1, \cdots, b_{m-1}) \in GF(2^m)$, and the product $C = AB = (c_0, c_1, \cdots, c_{m-1})$ are treated as polynomials $A(x), B(x)$, and $C(x)$ with respective coefficients. The dependence between these polynomials

is given by $C(x) = A(x).B(x) \bmod F(x)$, Where $F(x)$ is a constant irreducible polynomial of degree m. The hardware implementation for multiplication in $GF(2^m)$ is presented in Figure 6.

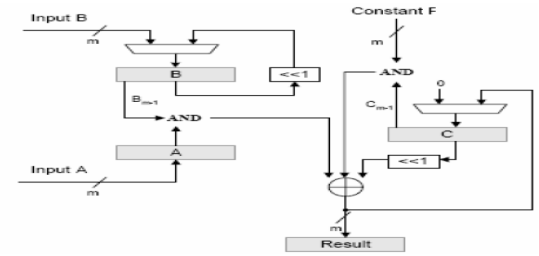

Figure 6. Serial Multiplier in $GF(2^m)$.

6.3. Inversion

The algorithm of inversion is given in section 5.5, and its Hardware implementation is presented in Figure 7.

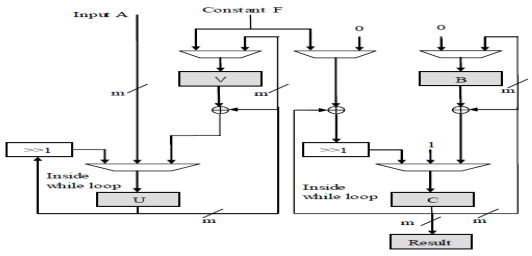

Figure 7. Inverter in $GF(2^m)$.

7. Simulation and results

7.1. FPGA

Field programmable gate array (FPGA) devices provide an excellent technology for the implementation of general purpose cryptographic devices. Compared with application specific integrated circuits (ASIC), FPGA as offer low non-recurring engineering costs, shorter design time, greater flexibility and the ability to change the algorithm or design.

7.2. NIST-recommended elliptic curves

The NIST elliptic curves over $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$ are listed in Table (II). The following notation is used. The elements of $\mathbb{F}_{2^m}$ are represented using a polynomial basis representation with reduction polynomial $f(x)$. The reduction polynomials for the fields $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$ are $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ and $f(x) = x^{233} + x^{74} + 1$ respectively. An elliptic curve E over $\mathbb{F}_{2^m}$ is specified by the coefficients $a, b \in \mathbb{F}_{2^m}$ of its defining equation $y^2 + xy = x^3 + ax^2 + b$. The number of points on E defined over $\mathbb{F}_{2^m}$ is nh, where n is prime, and h is called the co-factor. A random curve over $\mathbb{F}_{2^m}$ is denoted by B-m.

Table 2. NIST-recommended elliptic curves over $\mathbb{F}_{2^{163}}, \mathbb{F}_{2^{233}}$ [2].

B-163:	$m = 163, f(z) = z^{163} + z^7 + z^6 + z^3 + 1,$ $a = 1, h = 2$
b	= 0x 00000002 0A601907 B8C953CA 1481EB10 512F7874 4A3205FD
n	= 0x 00000004 00000000 00000000 000292FE 77E70C12 A4234C33
x	= 0x 00000003 F0EBA162 86A2D57E A0991168 D4994637 E8343E36
y	= 0x 00000000 D51FBC6C 71A0094F A2CDD545 B11C5C0C 797324F1
B-233:	$m = 233, f(z) = z^{233} + z^{74} + 1,$ $a = 1, h = 2$
b	= 0x 00000066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42 81FE115F 7D8F90AD
n	= 0x 00000100 00000000 00000000 00000000 0013E974 E72F8A69 22031D26 03CFE0D7
x	= 0x 000000FA C9DFCBAC 8313BB21 39F1BB75 5FEF65BC 391F8B36 F8F8EB73 71FD558B
y	= 0x 00000100 6A08A419 03350678 E58528BE BF8A0BEF F867A7CA 36716F7E 01F81052

The architecture has been tested on ISE 9.2i Software using XILINX FPGA xc5vlx50-3-ff1153 device and simulate with ISE Simulator.

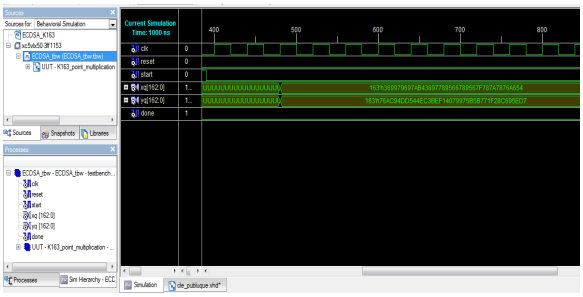

Figure 8. Final result of the scalar multiplication $k.P$ for $E(\mathbb{F}_{2^{163}})$

Table 3. The x and y input coordinates of the point P and an arbitrary value of k.

k	= 0x 00000001 33E3CAE7 2CD0F448 B2954810 FB75B5E3 D8F43D07
P_x	= 0x 00000003 69979697 AB438977 89566789 567F787A 7876A654
P_y	= 0x 00000004 035EDB42 EFAFB298 9D51FEFC E3C80988 F41FF883

Table 3 show the input parameters of the ECC scalar multiplication for a "163 bits" arbitrary value of k, and in Table 4, we give the implementation results corresponding.

Table 4. Synthesis results for $E(\mathbb{F}_{2^{163}})$.

point multiplication $G(F_{2^{163}})$		
Slice Logic Utilization:		
Number of Slice Registers:	2163	7%
Number of Slice LUTs:	2735	9%
Number used as Logic:	2735	9%
IO Utilization:		
Number of bonded IOBs:	330	58%
Maximum Frequency:	169.477MHz	

In Table 5, we give the implementation results for $\mathbb{F}_{2^{233}}$.

Table 5. Synthesis results for $E(\mathbb{F}_{2^{233}})$.

point multiplication $G(F_{2^{233}})$		
Slice Logic Utilization:		
Number of Slice Registers:	3073	10%
Number of Slice LUTs:	3637	12%
Number used as Logic:	3637	12%
IO Utilization:		
Number of bonded IOBs:	470	83%
Maximum Frequency:	136.323MHz	

8. Conclusion

In this work, the elliptic curve point multiplication is considered, we have analyzed the ECC protocols and designed the ECC processor over the field $GF(2^{163})$. The ECC processor can calculate various operations for implementing ECC protocols, which are a scalar multiplication, an Elliptic Curve point addition, a polynomial multiplication and a polynomial inverse multiplication. It is synthesized and tested with Xilinx FPGA and its average operation frequency for scalar multiplication is 169.477MHz.

9. Acknowledgment

This work was supported by the University of Paris 8.

References

[1] DSS. Digital signature standard (dss). *Federal Information Processing Standards Publication 186-2, National Institute of Standards and Technology*, 2000.

[2] D. H. J. L. Hernandez and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. *In Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES), volume 1965 of Lecture Notes in Computer Science*, 2001.

[3] IEEE.P1363. Standard specifications for public key cryptography. 2000.

[4] N. Koblitz. Elliptic curve crytosystem. *Mathematics of Computation*, 48:203–209, 1987.

[5] S. S. Kumar. *Elliptic curve cryptography for constrained devices*. PhD thesis, Ruhr-University Bochum, June 2006.

[6] Lejla.BATINA. *Arithmetic and Architectures for Secure Hardware Implementations of Public-Key Cryptography*. PhD thesis, KATHOLIEKE UNIVERSITEIT LEUVEN, December 2005.

[7] D. H. A. Menezes and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[8] V. S. Miller. Use of elliptic curves in cryptography. *Advances in Cryptology - CRYTO '85, Lecture Notes in Computer Science, Springer-Verlag, Hugh C. Williams (Ed.)*, 128:417–426, 1985.

[9] SEC.2. Recommended elliptic curve domain parameters. standard for efficient cryptography. *The SECG Group*, 2000.