HARDWARE IMPLEMENTATION OF ELLIPTIC CURVE ARITHMETIC IN GF ($2^m$)

W.El hadj youssef, Z.Guitouni, M.Machhout and R.Tourki

Elhadjyoussef_wajih@yahoo.fr
Ziedguitouni@yahoo.fr
Machhout@yahoo.fr
Rached.Tourki@planete.tn

Abstract: In this paper, we present an Elliptic Curve Point Multiplication processor over base Fields GF ($2^m$). Our design operates in a polynomial basis. It is fully parameterizable in the irreducible polynomial and the chosen Elliptic Curve over any base Galois Field up to a given size. High performance is achieved by the use of a dedicated Galois Field arithmetic coprocessor implemented on FPGA. We designed a parallelized version of the Montgomery point multiplication algorithm implemented on a reconfigurable hardware platform (Xilinx VirtexIV XC4VLX25 -12FF676).

Key words: Elliptic Curve Cryptography, Finite Field Arithmetic, Scalar Multiplication.

INTRODUCTION

Elliptic curves have been studied since the second half of 19th century as geometric algebraic entities without any cryptographic purpose. Application of elliptic curves in public key cryptography was independently proposed by Neals Koblitz and Victor Miller in 1985 [ERN 04]. Koblitz and Miller proposed to use an elliptic curve defined on a finite field, and to define a point addition operation such that the points of the elliptic curve and the point addition operation formed an abelian group. On this group, the discrete logarithm problem, called the Elliptic Curve Discrete Logarithm Problem (ECLDP), can be defined and so, a cryptosystem could be built on this problem.

The Elliptic Curve Cryptography (ECC) covers all relevant asymmetric cryptographic primitives like digital signatures and key agreement algorithms. It is standardized by many organizations [MIL 86]. The function used for this purpose is the scalar multiplication $K.P$, where $K$ is an integer and $P$ a point on an elliptic curve. The scalar multiplication can be computed with point addition and point doubling on elliptic curves which are based on computations in finite fields.

In this paper, we detailed the implementation of the ECC on FPGA. We developed all arithmetic operations on the finite field GF ($2^m$). To compute the scalar multiplication two groups of algorithms are required. The first computes the point operations on the elliptic curve in affine plane and the second in the projective plane. The scalar multiplication in projective coordinate eliminates the use of inversion, which is the hardest computing operation.

This article is structured as follows: In section 1, we describe arithmetic operations on an elliptic curve over a finite field. This section introduces the mathematical concepts necessary to understand and implement these arithmetic operations. Section 2 briefly presents the mathematical background, which is needed to explain the design considerations of the hardware architecture in section 3. In addition, the finite field arithmetic in GF ($2^m$) and elliptic curve arithmetic are discussed in this section. Section 4 describes the Elliptic Curve Arithmetic in projective coordinate. The results and interpretations are summarized in Section 5. This Section presents the synthesis results and timing for elliptic curve and field arithmetic. Finally, Section 6 concludes the paper.

1. The elliptic curve cryptography

An elliptic curve [MIL 86], defined on a field $K=GF (2^m)$ where $m$ is a prime, is the set of solution points $(x, y)$ to an equation of the form:

$$y^2 + x.y = x^3 + a.x^2 + b$$  \hspace{1cm} (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 can be equipped with an Abelian group structure [JOY 95] by the following point addition operation:

If \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \in \text{GF}(2^m) \),
ADD \( (P, Q) = P + Q = R \), with \( R = (x_3, y_3) \), where
\[
\begin{align*}
x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\
y_3 &= \lambda(x_1 + x_3) + y_1 + x_3 \\
\lambda &= (y_2 + y_1) / (x_2 + x_1)
\end{align*}
\]  
(2)

And by following point doubling operation
DOUBLE \( (P) = 2P = R \), where:
\[
\begin{align*}
x_3 &= \lambda^2 + \lambda + a \\
y_3 &= x_1^2 + (\lambda + 1)x_3 \\
\lambda &= x_1 + y_1 / x_1
\end{align*}
\]  
(3)

In this case, ADD requires one inversion, two multiplications, one squaring and eight additions. The DOUBLE operation requires five additions, two squaring, two multiplications and one inversion. All operations are on \( \text{GF}(2^m) \) [NGU 03].

ECC’s security is based on the discrete logarithm problem, called the Elliptic Curve Discrete Logarithm Problem (ECLDP). Thus, a cryptosystem could be built on this problem. The ECLDP consists on given two points \( P, Q \in \text{GF}(2^m) \), to find the positive integer \( k \) such as \( Q = kP \) [IZU 02]. This problem is of exponential complexity. On the contrary, knowing the scalar \( k \) and the point \( P \), the operation \( kP \) is relatively easy to compute. The hierarchy of arithmetic for an Elliptic Curve point multiplication is depicted in Figure. 1.

![Figure.1. ECC arithmetic hierarchy](image)

**2. Mathematical Background**

There are several cryptographic schemes based on elliptic curves [ERN 02]. These schemes work on a subgroup of points of an EC over a finite field. Arbitrary finite fields are approved to be suitable for ECC.

There are several bases known for \( \text{GF}(2^m) \). The most common bases, which are also proposed by the leading standards concerning ECC (IEEE 1363 and ANSI X9.62), are polynomial bases and normal bases. The finite field \( \text{GF}(2^m) \) is the underlying field on which elliptic curves are based throughout this work. It can be viewed as a vector space of dimension \( m \) over the field \( \text{GF}(2^m) \) [HAN 04].

In hardware field, elements can be easily implemented as a bit vector, which makes this kind of finite fields interesting for hardware implementations. For this work the representation is a polynomial basis only.

**2.1. The Finite Field \( \text{GF}(2) \)**

The smallest imaginable finite field is \( \text{GF}(2^n) \approx Z/2 \), which have two elements only: The additive and the multiplicative neutral elements 0 and 1 respectively. Its addition and multiplication tables assemble the truth tables of the binary XOR (⊕) and the binary AND (⊗) operation respectively. The elements can directly represented by a single bit.

**2.2. Polynomial Rings over \( \text{GF}(2) \)**

The set \( \text{GF}(2^m) = \sum_{i=0}^{m} a_i x^i ; a_i \in \text{GF}(2) \), of polynomials with coefficient in \( \text{GF}(2) \) together with the additive neutral element \( 0x^0 \), the multiplicative neutral element \( 1x^0 \), and polynomial addition as well as multiplication operations constitutes a ring over \( \text{GF}(2) \). Since the degree of a coefficient is given by it’s bit position, an element of \( \text{GF}(2^m) \) can effectively be represented by it’s coefficients stored in a bit vector.

**2.3. Finite Fields \( \text{GF}(2^m) \)**

Given an irreducible polynomial \( F \in \text{GF}(2^m) \) of degree \( m \), finite fields of extension degree \( m \) are constructed by modular arithmetic out of the previously defined polynomial rings as follows:

\[
\text{GF}(2^m) = \text{GF}(2) \backslash x / F(x).
\]

The set, which is underlying the Galois field, is thus the finite set of residue classes of polynomials modulo the prime polynomial \( F(x) \). The canonical representative of a polynomial’s residue class is the remainder of the polynomial division \( A(x) / F(x) \): it is a polynomial of degree less than \( m \). The computation of the canonical representative is called polynomial reduction.

**2.4. Addition in \( \text{GF}(2^m) \)**

For the chosen field \( \text{GF}(2^m) \) the addition of two numbers is very easy to compute, 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.

**2.5. Multiplication in \( \text{GF}(2^m) \)**

The multiplication of two polynomials \( A,B \in \text{GF}(2^m) \) is given by

\[
C = A \cdot B = \sum_{i=0}^{2m-2} c_i x^i \mod F
\]  
(4)
Denoting  
\[ c_k = \bigoplus_{i=0}^{2m-2} a_i \otimes b_{k-i} \quad \text{for} \quad 0 \leq k \leq 2m-2 \]

With \( P \) as the corresponding prime polynomial and \( a_i = 0, b_j = 0 \), for \( i \geq m \). Since \( \sum_{i=0}^{2m-2} c_i x^i \) has a maximum degree of \( 2m-2 \) the reduction of \( m-1 \) bits has to be performed.

2.6. Inversion in GF (2\(^m\))

The inverse operation in GF (2\(^m\)) corresponds to computing a multiplicative inverse of a polynomial modulo an irreducible polynomial \( P(x) \) of degree \( m \). To compute the multiplicative inverse for an element \( A \in \text{GF} (2^m) \), the extended Euclidean algorithm’s Theorem can be applied.

3. Design and implementation of the elliptic curve cryptosystems

3.1. Finite Field Arithmetic

In this section we present hardware implementation of finite field operation in GF (2\(^m\)). For further information we refer to [MAD 02].

3.1.1. 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.

3.1.2. Multiplication

Multiplication in GF (2\(^m\)) [HUT 03] with polynomial basis representation is defined as follows. Inputs \( A=(a_0,a_1,\ldots,a_{m-1}) \) and \( B=(b_0,b_1,\ldots,b_{m-1}) \in \text{GF}(2^m) \), and the product \( C=A\cdot B \) are treated as polynomials \( A(x),B(x), \) and \( C(x) \) with respective coefficients. The dependence between these polynomials is given by \( C(x) = A(x) \cdot B(x) \mod F(x) \), where \( F(x) \) is a constant irreducible polynomial of degree \( m \). The algorithm for multiplication in GF (2\(^m\)) is given below as algorithm.1, and its Hardware implementation is presented in Figure.2.

Algorithm.1. Multiplication in F\(_2^m\)  

**Require:**  
A(x), B(x) \( \in \text{GF}(2^m) \), A(x) \( \neq 0 \) and F(x) the irreducible polynomial of degree \( m \)  

**Ensure:**  
\( C(x) = A(x)^{-1} \mod F(x) \)

1: B(x) <= 1  
2: C(x) <= 0  
3: U(x) <= A(x)  
4: V (x) <= F (x)  
5: loop  
6: while U(0) = 0 do  
7: U(x) <= U(x) x\(^{-1}\)  
8: B(x) <= (B(x) + x_0 F (x)) x\(^{-1}\)  
9: end while  
10: if U(x) = 1 then  
11: return B(x)  
12: end if  
13: if grade (U(x) < grade (V (x))) then  
14: U(x) \( \triangleq \) V (x)  
15: C(x) \( \triangleq \) B(x)  
16: end if  
17: U(x) <= U(x) + V (x)  
8: B(x) <= (B(x) + x_0 F (x)) x\(^{-1}\)  
9: end while  
10: if U(x) = 1 then  
11: return B(x)  
12: end if  
13: if grade (U(x) < grade (V (x))) then  
14: U(x) \( \triangleq \) V (x)  
15: C(x) \( \triangleq \) B(x)  
16: end if  
17: U(x) <= U(x) + V (x)  

3.1.3. Inversion

The algorithm of inversion [DAL 03] is given below as algorithm.2, and its Hardware implementation is presented in Figure.3.

Algorithm.2. Inversion in F\(_2^m\)  

**Require:**  
A(x) \( \in \text{GF}(2^m) \), A(x) \( \neq 0 \) and F(x) the irreducible polynomial of degree \( m \)  

**Ensure:**  
\( C(x) = A(x)^{-1} \mod F (x) \)

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

4. The Elliptic curve arithmetic

The field arithmetic [BED 02] 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 [HAN 00]. A diagram of such inverter is shown in figure 3. The serial multiplication is based on a shift and add operation with interleaved polynomial reduction. The main advantage of using a serial multiplier is that it consumes less resource compared with other approaches. The architecture of this multiplier is shown in figure 2. A field multiplication can be achieved in $m$ clock cycles.

4.1. Projective Coordinates

Compared with 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 [DIO 99] 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. This is due to the fact that affine coordinate representation involves the usage of only two coordinates and therefore is more useful for external communication saving some valuable bandwidth.

In standard projective coordinates the projective point $(X: Y: Z)$ with $Z \neq 0$ corresponds to the affine coordinates $x = X / Z$ and $y = Y / Z$. The projective equation of the elliptic curve is given as:


Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points that belong to the curve of Equation 1 then $P + Q = (X_3, Y_3)$. Let the $x$ coordinate of $P$ be represented by $XZ$. Then, when the point $2P = (X_1, Y_1, Z_1)$ is converted to projective coordinate representation, it becomes:

$$X_3 = X_1^4 + b.Z_1^4 \quad (6)$$
$$Z_3 = X_1^2.Z_1^2$$

The computation of Eq.6 requires one general multiplication, one multiplication by the constant $b$, four squaring and one addition. Figure 5 is the sequence of instructions needed to compute a single point doubling operation $Mdouble(X_1, Z_1)$ efficiently [MOR 04].

Algorithm 3. Montgomery point doubling

**Input:** $P = (X_1, Z_1) \in E(F_2^m)$, $c$ such that $c^2 = b$

**Output:** $P = 2 \cdot P$

**Procedure:** Mdouble($X_1, Z_1$)

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. $X_3 = T + M$

Figure 4. Montgomery point doubling

The coordinates of $P + Q$ in projective coordinates can also be computed as the fraction $X_3 / Z_3$ and are given as:

$$Z_3 = (X_1.Z_2 + X_2.Z_1)^2$$
$$X_3 = x.Z_3 + (X_1.Z_2)(X_2.Z_1) \quad (7)$$

The required field operations for point addition [MEN 04] are as follows: three general multiplications, one multiplication by $x$, one squaring and two additions. This operation can be efficiently implemented as shown in Figure 5.
Input: \( P = (X_1, Z_1), Q = (X_2, Z_2) \in E(F_2^m) \)

Output: \( P = P + Q \)

Procedure: Madd\((X_1, Z_1, X_2, Z_2)\)
1. \( M = (X_1 \cdot Z_2) + (Z_1 \cdot X_2) \)
2. \( Z_3 = M \)
3. \( N = (X_1 \cdot Z_2) \cdot (Z_1 \cdot X_2) \)
4. \( M = x \cdot Z_3 \)
5. \( X_3 = M + N \)

Figure 5. Montgomery point addition

4.2. Montgomery point multiplication

A method based on the formulas for doubling and addition is shown in Figure 4 and Figure 5. The computation of \( KP \) is performed by expressing \( K \) in binary form \( K=K_{i_{m-1}}...K_1K_0 \) and applying “double and add method”:

\[
KP = 2\left(\ldots2\left(K_iP + K_{i-1}P\right) + \ldots\right) + K_0P \quad (8)
\]

Both Mdouble and Madd operations [ROD 04] are executed in each iteration of the algorithm. If the test bit \( k_i \) is ‘1’, the manipulations are made for Madd \((X_1, Z_1, X_2, Z_2)\) and Mdouble \((X_2, Z_2)\) else Madd \((X_2, Z_2, X_1, Z_1)\) i-e Mdouble and Madd with reversed arguments.

The approximate running time of the algorithm is \(10mM + (1I + 10M)\) where \( M \) represents a field multiplication operation, \( m \) stands for the number of bits and \( I \) correspond to inversion [ROD 04]. It is noted that the factor \((1I + 10M)\) represents time needed to convert from standard projective to affine coordinates. In the next subsection we discuss how to obtain an efficient parallel implementation of the above algorithm.

Algorithm 5. Montgomery point Multiplication

Input: \( k = (k_{m-1}, k_{m-2}, \ldots, k_1, k_0) \) with \( k_{m-1} = 1 \),
\( P(x, y) \in E(F_2^m) \)

Output: \( Q = kP \)

Procedure: MontPointMult \((P, k)\)
1. Set \( X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2 \)
2. For \( i \) from \( n - 2 \) downto \( 0 \) do
   2.1 if \( (k_i = 1) \) then
      Madd\((X_1, Z_1, X_2, Z_2)\);
      Mdouble\((X_2, Z_2)\);
   2.2 Else
      Madd\((X_2, Z_2, X_1, Z_1)\);
      Mdouble\((X_1, Z_1)\);
3. \( x_3 \leftarrow X_1/Z_1 \)
4. \( y_3 \leftarrow (x + X_1/Z_1)((X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1Z_2)(xZ_1Z_2)^{-1} + y \)
5. Return \((x_3, y_3)\)

5. Results and interpretation

We have prototyped the arithmetic logic unit in a Xilinx VirtexIV XC4VLX25 -12FF676 FPGA. All operations were defined over the finite field \( GF(2^{163}) \) and \( P(x) = x^{163} + x^8 + x^7 + x^3 + 1 \) the irreducible polynomial. The arithmetic unit was described using VHDL language. This module was simulated using Active-HDL and synthesized using the Xilinx ISE software. Synthesis results are shown in table 1 and table 2. We show the timing results obtained for elliptic curve and field arithmetic. In this section we give timings for our implementation and analyze the results.
Figure 7 shows the behavior of the Montgomery point multiplication.

<table>
<thead>
<tr>
<th>Module</th>
<th>% CLB</th>
<th>Time (ms)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>3 %</td>
<td>0.000804</td>
<td>134.56</td>
</tr>
<tr>
<td>Inverter</td>
<td>7 %</td>
<td>0.007584</td>
<td>230.56</td>
</tr>
</tbody>
</table>

Table.1. Synthesis serial multiplication for GF(2^{163})

The Elliptic curve arithmetic logic was described using VHDL language. Synthesis results are shown in table2. In this section we give Slices and Maximum frequency for our implementation and analyze the results.

<table>
<thead>
<tr>
<th>Module</th>
<th>% CLB</th>
<th>Time (ms)</th>
<th>Power(mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>19 %</td>
<td>0.0043474</td>
<td>138.03</td>
</tr>
<tr>
<td>Doublement</td>
<td>22 %</td>
<td>0.0041449</td>
<td>138.44</td>
</tr>
<tr>
<td>Point</td>
<td>32 %</td>
<td>9.89</td>
<td>230.62</td>
</tr>
</tbody>
</table>

Table.2. Synthesis results for E (F_{2^{163}})

Table.1 represents timing performances and the occupied resources of Inverter and multiplier architectures. In this table a comparison of the existing FPGA’s implementations of Finite field arithmetic over GF(2^{m}) is given. Our Binary Multiplier in GF(2^{163}) occupies 372 (3%) CLB slices and one field multiplication is performed in 0.000804 ms (202.143 MHz). Inversion occupies 791 CLB slices (7%) and one field inversion is performed in 0.00758483 ms (226.209 MHz).

Elliptic curve point addition and point doubling do not participate directly as a single computational unit in this design but parallel computations for both point addition and point doubling are designed together. The point addition occupy 2039 (19 %) CLB slices and it takes 0.0043474 ms (196.955 MHz) for one computational cycle. The point doubling occupy 2436 (22 %) CLB slices and it takes 0.0041449 ms (196.955 MHz) for one computational cycle.

The elliptic curve scalar multiplication over GF(2^{163}) occupies 3509 (32%) CLB slices and it takes 9.89 ms (165,203 MHz).

Generally, we speak about memory occupation and time execution. In this paper we introduce the constraints of power consumption. The XPow er allows you to analyze total device power. The XPower utilizes device knowledge and design data to estimate device and individual net power consumption.

Table.1 and table.2 introduced the power consumption of finite field arithmetic’s and the elliptic curve operations. We notice that further in addition to time computation and memory occupation, the inversion operation consumes 230.56 mW. Hence, the use of projective coordinates in order to eliminate the use of inversion which is the hardest operation to compute and to implement on hardware device.

Table 3 shows a performance comparison of the scalar multiplication timing result obtained with some hardware implementations.

<table>
<thead>
<tr>
<th>Reference</th>
<th>F_q</th>
<th>Platform</th>
<th>Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>F_{2^{163}}</td>
<td>Xilinx XC4VLX25</td>
<td>9.89</td>
</tr>
<tr>
<td>[ERN 01]</td>
<td>F_{2^{270}}</td>
<td>Xilinx XC4085XLA</td>
<td>6.8</td>
</tr>
<tr>
<td>[ERN 02]</td>
<td>F_{2^{113}}</td>
<td>Amtel AT94K40</td>
<td>10.9</td>
</tr>
<tr>
<td>[OKA 00]</td>
<td>F_{2^{163}}</td>
<td>Altera EPIF10K250</td>
<td>80</td>
</tr>
</tbody>
</table>

Table.3. Timing comparison for scalar multiplication

6. Conclusion

In this paper we have presented a hardware implementation of ECC, in GF (2^{163}). We have developed the arithmetic unit over the finite field of GF (2^{163}) (Multiplication, Inversion) and the elliptic curve operations (Addition, Doubling). The Montgomery Curve Point Multiplication algorithm is used to compute the scalar multiplication K.P.

We have provided a comparison with some ECC hardware implementations reporting the results in terms of CLBs area and performances. The proposed ECC implementation provides a Time of 9.8 ms with a 230.6 mw of power consumption.

REFERENCES


[ERN 01] M.Ernest et al.”Rapid Prototyping for Hardware Accelerated Elliptic Curve Public Cryptosystems”.Inc Proc. of 12th IEEE workshop on Rapid System


