# CREATING AN ELLIPTIC CURVE ARITHMETIC UNIT FOR USE IN ELLIPTIC CURVE CRYPTOGRAPHY

Apostolos P. Fournaris
Electrical and Computer Engineering Dpt
University of Patras
Patras-GREECE
apofour@ece.upatras.gr

Odysseas Koufopavlou
Electrical and Computer Engineering Dpt
University of Patras
Patras-GREECE
odysseas@ece.upatras.gr

## Abstract

*Elliptic Curve Cryptography (ECC) is a very promising cryptographic method, offering the same security level as traditional Public Key Cryptosystems (RSA, El Gamal) but with considerably smaller key lengths. To increase the performance of an EC Cryptosystem, dedicated hardware is employed for all EC point operations. However, the computational complexity and hardware resources of an Elliptic Curve processing unit are very high and depend on the efficient design of the Elliptic Curve's underlined $GF(2^k)$ Field. In this paper, we propose an EC arithmetic unit that is structured over a high peformance, low gate number $GF(2^k)$ arithmetic unit. This proposed $GF(2^k)$ arithmetic unit is based on one dimensional systolic architecture that can perform $GF(2^k)$ multiplication and inversion with only the performance cost of inversion. This is achieved by utilizing a multiplication/inversion algorithm based on the modified extended Euclidean algorithm.*

## 1. INTRODUCTION

One of the most promising types of cryptography is Public Key Elliptic Curve Cryptography (ECC) [2], [3]. Its security strength lies in the unfeasibility of solving the ECDLP (Elliptic Curve Discrete Logarithmic Problem) [1]. ECC is used in digital signature schemes or key exchange mechanisms. The ECDLP is based on the EC point multiplication operation. This operation is very complex and consists of a series of EC point addition and doubling operations. Each such operation follows the Group Law for point operations in the EC that is degraded to a series of arithmetic operations in the $GF(2^k)$ Field defining the EC. Therefore, the computational efficiency of an Elliptic Curve Cryptosystem, in terms of calculation speed and hardware resources, is highly dependent on the computation efficiency of EC point operations and $GF(2^k)$ Field operations.

Since, all the indicated operations, have a considerable computational cost, dedicated modules are usually employed for finding efficiently the operations' results. When the ECC application environment has very strict speed and resources requirements, the above modules can only be implemented in hardware. However, due to the high computational effort of the EC operations, the hardware architecture of such modules have to be very carefully constructed in order to meet the application constrains. There exist several works where EC operation arithmetic units are proposed. However, few of those works are applicable in very constrained security environments such as mobile and smart card applications. Thus, the efficient design of an EC operations arithmetic unit remains an open problem.

The core of EC arithmetic unit is the hardware modules that realize the $GF(2^k)$ Field operations required for each EC point operation. The three basic $GF(2^k)$ Field operations are addition-subtraction, multiplication, and inversion-division. Multiplication and inversion-division are considered the most computationally intensive, resource demanding and time consuming operations [4][6]. Many researchers have tried to design those operations in hardware and have managed some very promising results. However, in most such works, each $GF(2^k)$ Field operation is treated independently leading to architectures performing either multiplication or inversion-division. As a result $GF(2^k)$ arithmetic units employ multipliers and inverts-dividers separately, controlled by an internal control unit [4][6]. This approach results in a hardware resources increase and a throughput decrease. In other design approaches, the coordinate system of EC points is transformed (from affine to projective coordinates) in order to reduce the number of inversions at the cost of extra multiplications [1]. However, in that case, at least one inversion operation is needed at the end of point multiplication for transformation back to the original EC point coordinate system (from projective to affine coordinates). Therefore, the design of an inverter-divider can not be entirely avoided.

In this paper, an EC arithmetic unit is proposed besed on a innovative $GF(2^k)$ arithmetic unit that can perform all the GF($2^k$) arithmetic operations using reusability of the same hardware component. The architectures use a modified version of the Extended Euclidean Algorithm, similar to the one proposed in [5], that avoids the non constant iteration number problem without introducing complex control logic or high number of control signals. The algorithm can perform multiplication and inversion with appropriate initialization by reusing its algorithmic functions for both GF($2^k$) operations. Integrating the above algorithm, a one dimensional M/I systolic architecture is designed that can perform multiplication reusing the architectural structure of an inverter. More specifically, after *k* clock cycles the multiplication result reaches the output, after *2k - 1* clock cycles the inversion result is reaches the output and after *3k - 1* clock cycles the division result reaches the output. The one dimensional M/I architecture is the structural element of a proposed GF($2^k$) arithmetic unit's architecture. The above approach leads to an Elliptic Curve arithmetic unit consisting of a control unit, storage elements (registers) and the proposed GF($2^k$) arithmetic unit employing the proposed one dimensional M/I systolic architecture.

The paper is organized as follows. In section 2 Elliptic Curves and their arithmetic is presented. In section 3, the GF($2^k$) Field mathematical framework is analyzed and GF($2^k$) Field operations are presented. In this section, the utilized Modified EEA algorithms (sMEEA and inv/mulMEEA) for inversion and multiplication are presented. In section 4, the proposed one dimensional systolic M/I systolic architecture and resulting GF($2^k$) arithmetic unit is described and analyzed. The Elliptic Curve arithmetic unit is proposed in section 5 using the proposed GF($2^k$) arithmetic unit based on the M/I systolic architecture of section 4. The performance of all the architectures in terms of space (gate, flip flop, MUX number) and time (critical path delay, latency) complexity is discussed in section 6 and comparisons are made with other known designs. Section 7 concludes the paper.

## 2. ELLIPTIC CURVES IN GF($2^K$) FIELDS FOR CRYPTOGRAPHY

An Elliptic Curve *E* defined over a Field *F* is the set of solutions *(x,y)* where *x,y* ∊ *F*, of the *long Weierstrass equation* $E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5$ along with the *point at infinity* denoted as $\infty$. The variables $a_1$, $a_2$, $a_3$, $a_4$, $a_5$ ∊ *F* and $\Delta \neq 0$, where $\Delta$ is the discriminant of *E*.

We can say that two Elliptic Curves $E_1$ and $E_2$ defined over *F* are isomorphic if there exist a transformation between each *x* and *y* of $E_1$ and each $x'$, $y'$ of $E_2$ that has the form $x = u^2x' + r$, $y = u^3y' + su^2x' + t$, where *u, s, r, t* ∊ *F.* This transformation, referred as admissible change of variables, lead to a different equation defining the Elliptic Curve. This equation, denoted as *short Weierstrass equation,* can have several different forms according to the field *F* defining the Elliptic Curve *E.* For non supersingular Elliptic Curves defined over GF($2^k$) Fields, the characteristic *Char(GF($2^k$))=2* and the short Weierstrass equation has the form $E: y^2 + xy = x^3 + ax^2 + b$, where *a, b* ∊ *GF($2^k$)* and $\Delta = b \neq 0$. Those type of Elliptic Curves are widely used in Cryptography [1].

Three mathematic operations between points of the elliptic curve E can be identified. Point addition is the operation of adding one EC point $P_1$ to another EC point $P_2$ to obtain a third EC point $P_3$ ($P_3 = P_1 + P_2$). Point Doubling is a special case of point addition where an EC point $P_1$ is added to its self to obtain the EC point $P_3$ which is the double of $P_1$ ($P_3 = 2P_1$). Point multiplication is the operation of multiplying an EC point *P* with an integer *s* to obtain an EC point *Q* (*Q = sP*).

Each point multiplication operation requires a series of point addition and doubling operations following an algorithm known as double-and-add method. This method is shown below:

**Algorithm 1: Binary Point Multiplication algorithm (*Double and Add Method*)**

```
Input: $s = (s_{t-1}, \ldots, s_1, s_0)_2$, $P \in E(GF(2^k))$.
Output: $Q = sP$
1. $Q = \infty$.
2. For $i$ from 0 to $t-1$ do
        2.1 If $s_i = 1$ then $Q = Q + P$.
        2.2 $P = 2P$.
3. Return $Q$.
```

One point doubling operation is performed in each round of algorithm 1 and if the current bit of *s* is 1 a point addition operation follows.

To analyze the way point addition and doubling operations are performed, each point must be characterized by a set of values of the Finite Field (GF($2^k$)) defining the Elliptic Curve. In the two dimensional affine plane, each EC point is described by two values *x, y* ∊ *GF($2^k$)* called affine coordinates. In that case, the group law for affine coordinates is used for describing point addition and doubling results. Suppose that we have two EC points $P_1 = (x_1,y_1)$ and $P_2 = (x_2, y_2)$.

When $P_1$ • $P_2$ (point addition) the slope of the line between $P_1$ and $P_2$ would be $\lambda = \frac{y_2 + y_1}{x_2 + x_1}$ for $x_2$ • $x_1$ and the point $P_3 = P_1 + P_2 = (x_3, y_3)$ would be:

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

When $P_1 \bullet P_2$ but $x_2 = x_1$ then $P_3 = P_1 + P_2 = \infty$.

When $P_1 = P_2$ (Point Doubling) and $y_1 \bullet 0$, the slope of the tangent line in $P_1$ would be $\lambda = \frac{x_1^2 + y_1}{x_1}$ and the point $P_3 = P_1 + P_2 = 2P_1 = (x_3, y_3)$ would be:

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2}$$

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

When $P_1 = P_2$ (Point Doubling) and $y_1 = 0$, then $P_3 = 2P_1 = \infty$.

Since the point at infinity is the identity element of the $E(GF(2^k))$ Group, $P_3 = P_1 + \infty = P_1$. Point Subtraction can be performed using the point $-P_2$ instead of $P_2$, where $-P_2 = (x_2, x_2 + y_2)$.

Therefore, the problem of finding EC point addition and doubling results is reduced in finding the $x_3$, $y_3$ coordinates using GF($2^k$) Field mathematic operations. Three such operations stand out from the above analysis, addition, multiplication and division. Those operations highly influence the performance of the point addition and doubling operations and therefore also point multiplication. Their hardware efficiency plays a crucial role in the overall efficiency of an EC arithmetic unit system.

## 3. Arithmetic Operations In GF($2^k$) Fields

The GF($2^k$) field is isomorphic to $GF(2)[x]/(F(x))$, where $F(x)$ is a $k$ degree monic irreducible polynomial of the form :

$$F(x) = \sum_{i=0}^{k} f_i x^i$$

with coefficients $f_i \in \{0,1\}$.

In polynomial basis representation, an element α of a GF($2^k$) Field is a polynomial of degree less than or equal to $k$-$1$ defined over a basis $\{1, x, x^2, x^3, \ldots x^{k-1}\}$ with coefficients $a_i \in \{0,1\}$, where $x$ is a root of the irreducible polynomial $F(x)$. This can be written as

$$a(x) = \sum_{i=0}^{k-1} a_i x^i = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \ldots a_1 x + a_0$$

We can also represent an element $\alpha \in$ GF($2^k$) as a $k$ dimensional vector over GF(2). Therefore, in hardware a GF($2^k$) Field element can be represented as a binary number of its coefficients ($a_{k-1}, a_{k-2} \ldots a_1, a_0$). Addition between two or more GF($2^k$) Field elements becomes a modulo 2 addition of their coefficients. The operation is carry free and identical to subtraction. Addition and subtraction in GF($2^k$) Fields is a XOR operation.

### 3.1. GF($2^k$) field Multiplication

Suppose that $a(x)$, $b(x)$ polynomials are elements of a GF($2^k$) Field defined over the irreducible polynomial $f(x)$ in polynomial basis representation. Then, we define multiplication as

$$c(x) = \sum_{i=0}^{k-1} c_i x^i = a(x) \bullet b(x) = a(x) \cdot b(x) \bmod f(x)$$

where $c(x) \in$ GF($2^k$) and $c_i \in$ GF(2). GF($2^k$) Field Multiplication is a modular operation.

Using the fact that $x$ is a root of the irreducible polynomial ($f(x)=0$) the well known MSB and LSB bit serial multiplication algorithms can be introduced [6]. Those algorithms follow the shift and add principle but process the multiplier $b(x)$ beginning from the Least Significant bit (LSB) or the Most Significant bit (MSB). In our analysis we chose the LSB multiplication algorithm, presented below in its bit level representation were the superscript represents the number of rounds and the subscript the bit position [6],[11]:

**Algorithm 2: Bit level LSB multiplication**

```
Input: $a = \{a_{k-1} \ldots a_2, a_1, a_0\}$, $b = \{b_{k-1} \ldots b_1, b_0\}$,
    $f(x) = x^k + \sum_{i=0}^{k-1} f_i x^i = x^k + r(x)$, $r = \{f_{k-1} \ldots f_1, f_0\}$
Output: $c = a \bullet b$, $c(x)=a(x)b(x) \bmod f(x)$

For i, j = 0 to k - 1
    $a_i^{(j)} = \begin{cases} a_{i-1}^{(j-1)} \oplus a_{k-1}^{(i-1)} \cdot f_i, & 1 \le i \le k-1 \\ a_{k-1}^{(i-1)} \cdot f_0, & i = 0 \end{cases}$
    $c_i^{(j)} = c_i^{(j-1)} \oplus a_i^{(j-1)} b_{j-1}$
```

### 3.2. GF($2^k$) field Inversion

If $a^{-1}(x)a(x) \equiv 1(\bmod f(x))$, then $a^{-1}(x)$ is called the multiplicative inverse of $a(x)$. One of the most popular algorithms for calculating the multiplicative inverse of a GF($2^k$) Field element $a$ is the Extended Euclidean Algorithm (EEA). However, $a^{-1}$ in the EEA cannot be calculated in a constant number of steps and the use of systolic or pipelined techniques necessary for the achievement of small critical path delay and high throughput designs is difficult. Therefore, many modification of the original EEA have been proposed to solve this problem [8] [9] [10] [12]. One of the most promising such modifications of EEA (MEEA) is proposed in [9] and analyzed further in [5]. This algorithm, employs four $k$-degree polynomials, $u(x)$, $w(x)$, $s(x)$, $t(x)$ and an integer $e$ in order to come up with the multiplication result.

However, the use of the integer $e$ and the non regular way this number changes in each inversion round of MEEA poses a different design problem. This integer takes positive and negative values ($-k+1 \le e \le k-1$) and is increased or decreased by one in each round according to the conditions of the previous round. This asymmetry in the computational flow leads to complex control logic circuitry that affects the critical path delay and space complexity of a resulting inversion architecture. This control logic usually consists of a separate adder or Up/Down counter.

To solve this problem, a different encoding of the integer *e* is proposed in [5] that can be realized in hardware by a left/right one bit shifter (Shifter D). When shifted left, $d_0 = 1$ and when shifted right $d_{k-1} = 0$, where $d_0$ is the least significant bit of D and $d_{k-1}$ the most significant bit of D. The D shifter represents $|e|$. When the number of 1's in the D shifter is reduced, *e* is closer to zero. When the number of 1's in D shifter is increased *e* moves further away from zero. The possible left or right shifting of D can be controlled by a signal. This signal is identical to the estimated sign of integer *e* ($\text{sign}^{(i)}$). The integer's *e* sign of each round is estimated from the *sign* ($\text{sign}^{(i-1)}$) and $d_0$ value ($d_0^{(i-1)}$) of the previous round. The encoding of the *e* integer using the D shifter is shown in Table I.

Table I
D Shifter Encoding.

| $e^{(i-1)}$ | $D^{(i-1)} = \{d_{k-1},..d_0\}$ | $e^{(i)}$ | $D^{(i)}=\{d_{k-1},..d_0\}$ | $\text{sign}^{(i)}$ |
|---|---|---|---|---|
| -1 | {0…..0001} | -2 | {0…..0011} | 1 |
| 0 | {0…..0000} | -1 | {0…..0001} | 1 |
| 2 | {0…..0011} | 1 | {0…..0001} | 0 |
| 1 | {0…..0001} | 0 | {0…..0000} | 0 |
| -3 | {0…..0111} | 2 | {0…..0011} | 0 |

The resulting algorithm (sMEEA) in bit level is presented below. The symbol + is used for logical OR, ⊕ for XOR and · for AND whenever 1 bit operations are involved. Note that the sMEEA algorithm presented below is slightly different than the one in [5]. The sMEEA algorithm does not contain errors that were present in the structure of [5] algorithm.

**Algorithm 3: bit level sMEEA**

```
Input: $a(x), f(x)$
Output: $a^{-1}(x) \bmod f(x)$
Initialization: $s_i^{(0)}=f_i$, $v_i^{(0)}=0$, ($r_0^{(0)}=0$ and $r_i^{(0)} = a_{i-1}$ for $i>0$) ($u_0^{(0)}=1$ and $u_i^{(0)}=0$ for $i>0$) ($d_0^{(0)}=1$ and $d_i^{(0)}=0$ for $i>0$ ), $i \in \{0,k\}$ and $sign^{(0)}=1$ (meaning e is negative)
1. For $j = 1$ to $2k - 1$
   $c_1 = r_k^{(j-1)}$
   $c_2 = c_1 \cdot sign^{(j-1)}$
2. For $i = 0$ to $k$
   $r_i^{(j)} = r_{i-1}^{(j-1)} \oplus s_{i-1}^{(j-1)} \cdot c_1$
   $u_i^{(j)} = u_i^{(j-1)} \oplus v_i^{(j-1)} \cdot c_1$
   $s_i^{(j)}, v_i^{(j)} = \begin{cases} s_i^{(j-1)}, v_{i-1}^{(j-1)} & for \quad c_2 = 0 \\ r_i^{(j-1)}, u_{i-1}^{(j-1)} & for \quad c_2 = 1 \end{cases}$
   $sign^{(j)} = \begin{cases} sign^{(j-1)} \cdot \overline{r}_k^{(j-1)} & for \quad d_0^{(j-1)} = 1 \\ 1 & for \quad d_0^{(j-1)} = 0 \end{cases}$
   $d_i^{(j)} = \begin{cases} d_{i+1}^{(j-1)} \; and \; d_{k-1}^{(j)} = 0 & for \; sign^{(j)} = 0 \\ d_{i-1}^{(j-1)} \; and \; d_0^{(j)} = 1 & for \; sign^{(j)} = 1 \end{cases}$
Return: $a_i^{-1} = v_{k-i}^{(2k-1)} \quad for \quad i = 0 \; to \; k-1$
```

### 3.3. GF($2^k$) field Multiplication-Inversion algorithm

When sMEEA algorithm is compared with the LSB multiplication algorithm of subsection 3.1, similarities can be found [5]. The calculation of $v_i^{(j)}$ and $u_i^{(j)}$ or $r_i^{(j)}$ and $s_i^{(j)}$ employs the same function as that of the basic cell of LSB, $a_i^{(j)}$, $c_i^{(j)}$ and $b_i^{(j)}$. The function $x = y \oplus z \cdot v$ can be reused for multiplication and inversion by modifying the sMEEA accordingly. More specifically, a control signal is added for separating the one operation from the other. The resulting inv/mulMEEA is presented below:

**Algorithm 4: bit level inv/mulMEEA**

```
Input: $a(x), b(x), f(x), mul/inv$
Output: $a^{-1}(x) \bmod f(x)$ or $c(x)=a(x)b(x) \bmod f(x)$
Initialization: $s_i^{(0)}=f_i$, $v_i^{(0)}=0$, ($r_0^{(0)}=0$ and $r_i^{(0)}=a_{i-1}$ for $i>0$) ($u_0^{(0)}=1$ and $u_i^{(0)}=0$ for $i>0$) ($d_0^{(0)}=1$ and $d_i^{(0)}=0$ for $i>0$ and $sign^{(0)}=1$ for $mul/inv=0$ : inversion) ($d_0^{(0)}=0$ and $d_i^{(0)}=0$ for $i>0$ and $sign^{(0)}=0$ for $mul/inv=1$ : multiplication) $i \in \{0,k\}$

1. For $j=1$ to $2k-1$
   $c_{1a} = r_k^{(j-1)}$
   $c_2 = c_{1a} \cdot sign^{(j-1)}$
2. For $i=0$ to $k$
   $r_i^{(j)} = r_{i-1}^{(j-1)} \oplus s_{i-1}^{(j-1)} \cdot c_{1a}$
   $u_i^{(j)} = u_i^{(j-1)} \oplus v_i^{(j-1)} \cdot c_{1b}$
   $c_{1b}, g = \begin{cases} c_{1a}, v_{i-1}^{(j-1)} & for \quad mul/inv = 0 \\ b_{j-1}, r_i^{(j-1)} & for \quad mul/inv = 1 \end{cases}$
```

$$s_i^{(j)}, v_i^{(j)} = \begin{cases} s_i^{(j-1)}, g & for \quad c_2 = 0 \\ r_i^{j-1}, u_{i-1}^{(j-1)} & for \quad c_2 = 1 \end{cases}$$

$$sign^{(j)} = \begin{cases} sign^{(j-1)} \cdot \overline{r}_k^{(j-1)} & for \quad d_0^{(j-1)} = 1 \\ 1 & for \quad d_0^{(j-1)} = 0 \end{cases}$$

$$d_i^{(j)} = \begin{cases} d_{i+1}^{(j-1)} \quad and \quad d_{k-1}^{(j)} = 0 & for \ sign^{(j)} = 0 \\ d_{i-1}^{(j-1)} \quad and \quad d_0^{(j)} = 1 & for \ sign^{(j)} = 1 \end{cases}$$

Result: $\overline{a^{-1}(x)} = v^{(2k-1)}(x)$

$$a_i^{-1} = v_{k-i}^{(2k-1)}, c_i = u_i^{(k)} \quad for \quad i = 0 \ to \ k-1$$

In the inv/mulMEEA, if the signal *mul/inv* is set (*mul/inv = 1*) multiplication is performed instead of inversion. In that case, steps 1 and 2 of inv/mulMEEA become:

$$r^{(j)}(x) = x \cdot r^{(j-1)}(x) + x \cdot s^{(j-1)}(x) \cdot r_k^{(j-1)}$$

$$u^{(j)}(x) = u^{(j-1)}(x) + g(x) \cdot b_{j-1}$$

and since *r(x) = x · a(x)* and *s(x) = f(x)* in bit level the above equations become:

$$a_i^{(j)} = a_{i-1}^{(j-1)} \oplus f_{i-1} \cdot a_k^{(j-1)}$$

$$u_i^{(j)} = u_i^{(j-1)} \oplus a_i^{(j-1)} \cdot b_{j-1}$$

If we change *u(x)* into *c(x)* then the above equation is identical to the LSB algorithm in bit level. It must be noted, however, that the signal *mul/inv* can not be active more than *k* rounds since the multiplication process is concluded in *k* rounds.

## 4. Proposed One Dimensional Systolic Multiplication/Inversion Architecture and GF($2^k$) Arithmetic Unit Architecture

Using algorithm inv/mulMEEA a multiplication – inversion architecture can be proposed. This architecture is formed as a one dimensional systolic array that processes bits in a bit serial fashion. More specifically, the architecture consists of two different types of Processing Elements (PE), presented in Fig. 1.

The Type I PEs perform the calculations of *r(x)*, *s(x), u(x), v(x)* for multiplication and inversion and generate the appropriate control signals. The Type II PEs perform multiplication and inversion using the control signals of the Type I PEs. The extra circuitry for performing multiplication instead of inversion using the PE of Fig. 1 is one MUX and one extra signal $c_{1b}$ in each PE along with one additional MUX in Type I PE for choosing the correct value of $c_{1a}$.

To construct the one dimensional multiplication inversion (M/I) architecture, *k+1* PEs are required. Each such PE represents the calculations performed for one specific bit. The proposed one dimensional systolic M/I architecture is shown in Fig. 2. The values of *a(x)*, *f(x)* are inserted in parallel in the PEs. Each PE pumps the value of the various control signals generated by the Type I PE to the consecutive PE. The value of *b(x)* is inserted in serial fashion, one bit in each clock cycle and pumped similarly to all the PEs.

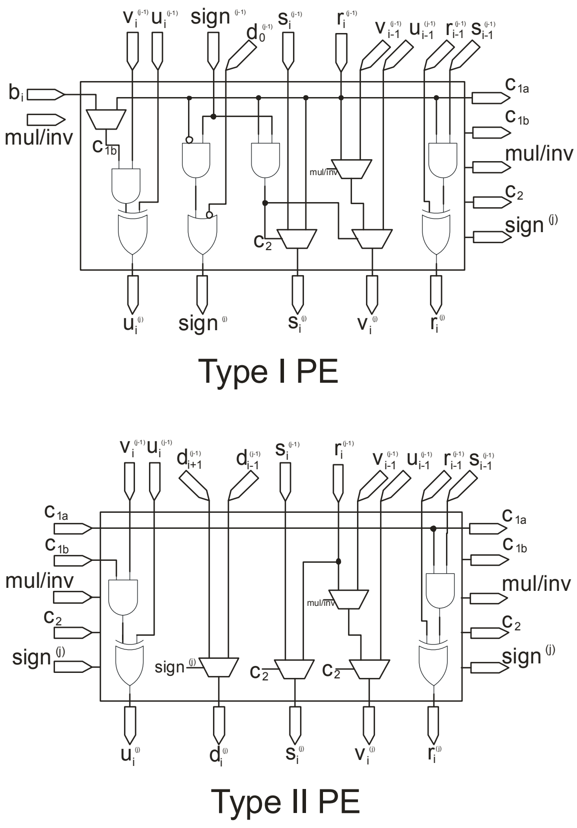
Type I PE
$v_i^{(j-1)}$ $u_i^{(j-1)}$ $sign^{(j-1)}$ $d_0^{(j-1)}$ $s_i^{(j-1)}$ $r_i^{(j-1)}$ $v_{i-1}^{(j-1)}$ $u_{i-1}^{(j-1)}$ $r_{i-1}^{(j-1)}$ $s_{i-1}^{(j-1)}$
$b_i$
mul/inv
$c_{1b}$
$c_2$
$c_{1a}$
$c_{1b}$
mul/inv
$c_2$
$sign^{(j)}$
$u_i^{(j)}$ $sign^{(j)}$ $s_i^{(j)}$ $v_i^{(j)}$ $r_i^{(j)}$
Type II PE
$v_i^{(j-1)}$ $u_i^{(j-1)}$ $d_{i+1}^{(j-1)}$ $d_{i-1}^{(j-1)}$ $s_i^{(j-1)}$ $r_i^{(j-1)}$ $v_{i-1}^{(j-1)}$ $u_{i-1}^{(j-1)}$ $r_{i-1}^{(j-1)}$ $s_{i-1}^{(j-1)}$
$c_{1a}$
$c_{1b}$
mul/inv
$c_2$
$sign^{(j)}$
$u_i^{(j)}$ $d_i^{(j)}$ $s_i^{(j)}$ $v_i^{(j)}$ $r_i^{(j)}$

**Figure 1. Types of Processing Elements.**

The intermediate values of *r(x), s(x), u(x), v(x)* are reinserted to each PE in each clock cycle. The calculations are concluded after *2k-1* clock cycles for inversion and *k* clock cycles for multiplication. The signal *init* is responsible for the correct initialization of the architecture. It controls a series of multiplexers (MUX). Those MUXs are responsible for loading the initial values *x•a(x)* and *f(x)* in the signals *r(x)* and *s(x)* when *init = 1* or the previous round's calculated output of each PE when *init = 1*. Special care has been paid in the correct initial value of the D shifter's $d_0$ bit and the initial value of the *sign*. It has been found that both initial values are always $\overline{mul / inv}$.

The proposed architecture of Fig. 2 is the basic structural element of a GF($2^k$) arithmetic unit since it can perform all GF($2^k$) arithmetic operations (multiplication, inversion) apart from addition. GF($2^k$) addition can be realized in hardware as a parallel structure of XOR gates and has very small computational cost in speed and hardware resources. Due to its small cost, it can be realized in a proposed GF($2^k$) arithmetic unit as an independent module that can function in parallel to the M/I architecture. The structure of the proposed GF($2^k$) Arithmetic unit is presented in Fig. 3.

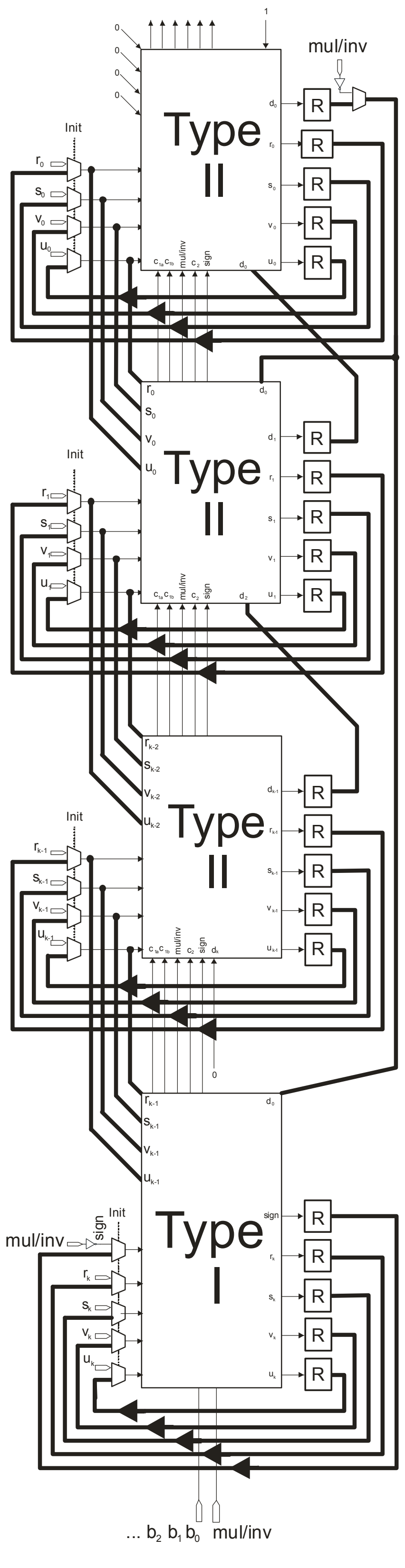

Figure 2. Proposed One Dimensional Systolic M/I Architecture

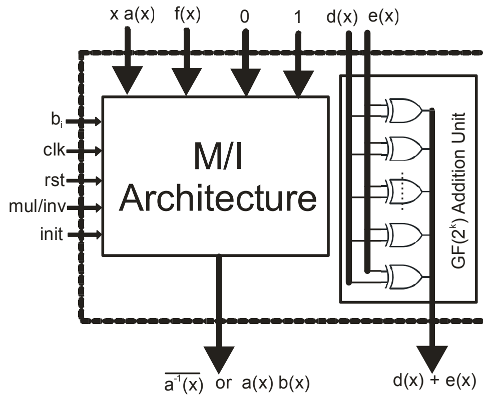

**Figure 3. Proposed GF($2^k$) Arithmetic Unit.**

The proposed GF($2^k$) arithmetic unit of Fig. 3 takes as input four polynomials in parallel. Those polynomials are, the *a(x)* polynomial for GF($2^k$) inversion along with the irreducible polynomial *f(x)* defining the GF($2^k$) Field and two polynomials *d(x)* and *e(x)* for GF($2^k$) addition. In addition to the above inputs, a *b(x)* polynomial can be used as input in bit serial fashion when multiplication is performed. There exist two outputs to the arithmetic unit. The first output is for GF($2^k$) addition and the second for GF($2^k$) inversion or multiplication. Note that the output in case of inversion is $\overline{a^{-1}(x)}$ meaning that the final permutation for finding $a^{-1}(x)$ is not done.

## 5. Proposed EC Arithmetic Unit Architecture

The proposed GF($2^k$) Arithemtic unit is the construction element of an EC arithmetic unit that can be employed in ECC. By applying a Control logic policy and a series of simple storage elements the functionality of the GF($2^k$) arithmetic unit is extended in order to perform all EC point operation. The generic structure of a proposed EC point operation arithmetic unit is presented in Fig. 4.

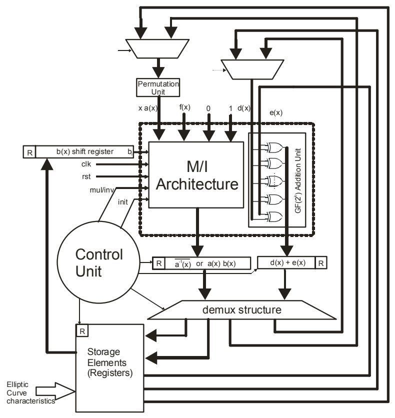

**Figure 4. Proposed EC point operation arithmetic unit.**

The architecture of Fig. 4 consists of a control unit, the proposed GF($2^k$) arithmetic unit, storage elements, registers and a multiplexer and demultiplexer structure. The role of the multiplexer and demultiplexer structure is to control the data flow of the GF($2^k$) arithmetic unit as is required by the EC point operation calculation procedure. The data that are calculated from the GF($2^k$) arithmetic unit are either stored in the strorage elements or are reinserted in the GF($2^k$) arithmetic unit for further processing.

## 5.1. Control Unit Sequence

The control unit is assigned with the overall management of the system. It is responsible for executing the algorithmic steps of EC point addition, EC point doubling and EC point multiplication operations. The control unit generates the appropriate control signals for the needed calculations and storage operations in the architecture's registers. The control unit calculates the required number of clock cycles for each GF($2^k$) Field operation by using counters. The exact functionality of a control unit can be described after an analysis of the EC point addition and doubling calculations presented in section 2.

The EC point addition operation can be broken in smaller and simpler GF($2^k$) Field operations than those presented in section 2. More specifically, for EC point addition assume that $\lambda$ can also be described as $\lambda = \frac{y_2 + y_1}{x_2 + x_1} = \frac{Y^+}{X^+} = Y^+ \cdot \frac{1}{X^+}$ where $Y^+ = y_2 + y_1$ and $X^+ = x_2 + x_1$. In that case, the EC point $P_3$ : ($x_3$, $y_3$) can be found as:

$$x_3 = \lambda^2 + \lambda + a + x_1 + x_2 = \left(\frac{Y^+}{X^+}\right)^2 + \frac{Y^+}{X^+} + (a + X^+)$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 = \left(\frac{Y^+}{X^+}\right)(x_3 + x_1) + (x_3 + y_1)$$

Using the above notation, the control unit can manipulate the EC point addition operation following the methodology proposed in Table II. In this Table, there is a full description of which GF($2^k$) Field operation (and what type of GF($2^k$) Field operations) needs to be performed on any specific clock cycle of the EC operation sequence and which GF($2^k$) Field operations are beginning on any specific clock cycle along with their estimated conclusion clock cycle time. Also, the storage operations that take place in each clock cycle are presented.

Similar analysis can be done for the EC point doubling operation. The methodology is presented in Table III. More specifically, for EC point doubling assume that $\lambda$ is described as $\lambda = \frac{x_1^2 + y_1}{x_1}$ and the point $P_3 = 2P_1 = (x_3, y_3)$ would be:

$$x_3 = \lambda^2 + \lambda + a \qquad y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

Table II
Control Unit EC Point Addition Calculation Sequence

| Current Round's starting operation | | Clk. no | Store operation | Concl. in clk no | Current output |
|---|---|---|---|---|---|
| Addition | Mult./ Inversion | | | | |
| $X^+ = x_1+x_2$ | - | 1 | - | 2 | - |
| $Y^+ = y_1+y_2$ | $1/(X^+)$ | 2 | $X^+$ | 3/ 2k+1 | $X^+ = x_1+x_2$ |
| $a + X^+$ | - | 3 | $Y^+$ | 4 | $Y^+ = y_1+y_2$ |
| $x_1 + x_3$ | - | 4 | $a + X^+$ | 5 | $a + X^+$ |
| $y_1 + x_3$ | - | 5 | $x_1 + x_3$ | 6 | $x_1 + x_3$ |
| - | - | 6 | $y_1 + x_3$ | - | $y_1 + x_3$ |
| .... | ..... | .... | ..... | .... | .... |
| | $\lambda = (Y^+)\left[1/X^+\right]$ | 2k + 1 | $1/(X^+)$ | 3k+1 | $1/(X^+)$ |
| .... | ..... | .... | ..... | .... | .... |
| $\lambda + a + X^+$ | $\lambda^2$ | 3k+1 | $\lambda$ | 3k +2 /4k+1 | $\lambda$ |
| - | - | 3k + 2 | $\lambda + a + X^+$ | - | $\lambda + a + X^+$ |
| .... | ..... | .... | ..... | .... | .... |
| $x_3 = \lambda^2 + \lambda + a + X^+$ | $\lambda(x_1 + x_3)$ | 4k+1 | $\lambda^2$ | 4k +2 / 5k +1 | $\lambda^2$ |
| - | - | 4k +2 | $x_3$ | - | - |
| ..... | ..... | ..... | ..... | ..... | ..... |
| $y_3 = \lambda(x_1 + x_3) + y_1 + x_3$ | - | 5k+1 | $\lambda(x_1 + x_3)$ | 5k+2 | $\lambda(x_1 + x_3)$ |
| - | - | 5k+2 | $y_3$ | - | $y_3$ |

Table III
Control Unit EC Point Doubling Calculation Sequence

| Current Round's starting operation | | Clk. no | Store operation | Concl. in clk no | Current output |
|---|---|---|---|---|---|
| Addition | Mult./ Inversion | | | | |
| $x_1 + x_3$ | $1/x_1$ | 1 | - | 2/2k | - |
| $x_3 + y_1$ | - | 2 | $x_1 + x_3$ | 3 | $x_1 + x_3$ |
| - | - | 3 | $x_3 + y_1$ | - | $x_3 + y_1$ |
| … | … | … | … | … | … |
| - | $(y_1)[1/x_1]$ | 2k | $1/x_1$ | 3k | $1/x_1$ |
| … | … | … | … | … | … |
| $\lambda = x_1 + (y_1/x_1)$ | - | 3k | $y_1/x_1$ | 3k + 1 | $(y_1/x_1)$ |
| $\lambda + a$ | $\lambda^2$ | 3k+1 | $\lambda$ | 3k+2 / 4k+1 | $\lambda$ |
| - | - | 3k + 2 | $\lambda + a$ | - | $\lambda + a$ |
| … | … | … | … | … | … |
| $x_3 = \lambda^2 + \lambda + a$ | $\lambda(x_1 + x_3)$ | 4k+1 | $\lambda^2$ | 5k + 1 | $\lambda^2$ |
| - | - | 4k + 2 | $x_3$ | - | $x_3$ |
| … | … | … | … | … | … |
| $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ | - | 5k +1 | $\lambda(x_1 + x_3)$ | 5k + 2 | $\lambda(x_1 + x_3)$ |
| - | - | 5k + 2 | $y_3$ | - | $y_3$ |

It can be remarked that both EC point operations described above give a result after the same number of clock cycles (*5k + 2*).

## 6. Performance Analysis.

The critical path delay of the proposed one dimensional systolic M/I architecture can be extracted from Fig. 1 and Fig. 2. If $T_{AND}$ is the delay of an AND, $T_{XOR}$ the delay of a XOR gate and $T_{MUX}$ the delay of a MUX, the critical path delay would be $T_{AND}+T_{XOR}+T_{MUX}$. The proposed one dimensional systolic M/I architecture requires *2k-1* clock cycles to complete one inversion, *k* clock cycles to complete one multiplication and *3k-1* clock cycles for division. Those measurements along with details about the area complexity (gate number) are shown in Table IV where the architecture is compared with well known division architectures. It is assumed that $T_{XOR3}=2T_{XOR}$.

It can be noted that the proposed architecture fares very well when compared to other known architectures. It achieves the smallest latency and critical path delay. It also has the smallest number of hardware components (XOR, AND Gates, MUXs, Flip Flops) since it does not require any extra circuitry like, adders or zero checkers.

Table IV
Area and Time complexity comparisons of one dimensional systolic architectures.

| Architect. | Guo [10] | Daneshbeh [12] | proposed |
|---|---|---|---|
| OR | *0* | *0* | *1* |
| NOT | *0* | *0* | *0* |
| AND | *26k* | *14k* | *2k+4* |
| XOR | *11k* | *6k* | *2k+1* |
| MUX | *35k+2* | *20k* | *8k+9* |
| Flip Flops | $46k+4k\lceil \log k+1 \rceil$ | *36k* | *10k+5* |
| other | *adder, zero check* | - | - |
| Lat. (div) | *8k-1* | *5k-4* | *3k-1* |
| Crit. Path | $2T_{XOR}+ 2T_{AND}+2\ T_{MUX}$ | $T_{XOR}+2T_{AND} + T_{MUX}$ | $T_{XOR}+T_{AND} +T_{MUX}$ |

From the proposed one dimensional systolic M/I architecture results, the gate number and critical path delay of the GF($2^k$) arithmetic unit can also be calculated. The critical path delay of the proposed GF($2^k$) arithmetic unit is the same as the M/I architecture and equal to $T_{AND}+T_{XOR}+T_{MUX}$ while the gate number is higher by *k* XOR gates due to the GF($2^k$) addition unit.

The EC arithmetic unit as already indicated in the previous section, performs one EC point addition in *5k + 2* clock cycles and one EC point doubling in the same time delay. Using the algorithm presented in section 2 for EC point multiplication and assuming that the the number of 1 and 0 in the binary form of *s* integer are the same (Hamming weight of *s = k/2*) then one EC point multiplication will be concluded in $7.5\ k^2 + 3k$ clock cycles.

## 7. Conclusions.

In this paper, we proposed an EC arithmetic unit based on a GF($2^k$) arithmetic unit. This proposed GF($2^k$) arithmetic unit utilizes a one dimensional M/I systolic architecture that performs multiplication and inversion using only the inversion computational flow. As a result, the proposed GF($2^k$) arithmetic unit has the space and time complexity of an inverter but can perform all GF($2^k$) operations. As analyzed in the paper this architecture can be the structural element of very competitive EC arithmetic unit. The overall proposed system can be applied in ECC applications, especially if the application environment is computationally constrained, since the proposed system has high speed and small gate number in comparison to other works.

### References

[1] D. Hankerson, A. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography,* Springer-Verlag & Hall/CRC, New York, 2004.

[2] N. Koblitz, "Elliptic Curve Cryptography", *Mathematics of Computation*, 48, pp 203-209, 1987.

[3] V. Miller, "Use of Elliptic Curves in Cryptography", *in Advances in Cryptography (Crypto '85)*, LNCS 218, pp 47- 426, 1986.

[4] E. D. Mastrovito, "VLSI architectures for computations in Galois fields", *Phd Thesis*, Linkoping Univ., Sweden, 1991.

[5] A. P. Fournaris and O. Koufopavlou, "Applying Systolic Multiplication-Inversion Architectures Based On Modified Extended Euclidean Algorithm For GF($2^k$) in Elliptic Curve Cryptography, *Computer and Electrical Engineering Journal*, vol.33, issue 5-6, pp 333-348, Elsevier Ltd, September 2007

[6] J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic, 1993.

[7] M. Olofsson, "VLSI Aspects on Inversion in Finite Felds", *Phd Thesis*, Linköping Univ., Sweden, 2002.

[8] J.-H. Guo and C.-L. Wang, "Hardware Efficient Systolic Architecture for Inversion and Division in GF($2^k$)", *IEE Proc. on Computer and Digit. Techniques*, p 272 – 278, 1998.

[9] Z. Yan and D. V. Sarwate, "New Systolic Architectures for Inversion and Division in GF($2^k$)", *IEEE Transactions on Computers*, vol 52, no 11, pp 1514 – 1519, November 2003.

[10] J.-H. Guo and C.-L. Wang, "Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF($2^m$)", *IEEE Transactions on Computers*, vol 47, no 10, p 1161 – 1167, October 1998.

[11] L. Song and K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers", *Journal of VLSI Sign. Processing*, vol 19, no 2, pp 149 – 166, July 1998.

[12] K. Daneshbeh and M. A. Hasan, "A Class of Unidirectional bit serial systolic architectures for Multiplicative Inversion and Division over GF($2^m$)", IEEE Transactions on Computers, vol. 54 , no. 3, pp 370-380, March 2005.