International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:5, 2012

Efficient Hardware Implementation of an Elliptic Curve Cryptographic Processor over GF (2^{163})

Massoud Masoumi, Hosseyn Mahdizadeh

***Abstract*—**A new and highly efficient architecture for elliptic curve scalar point multiplication which is optimized for a binary field recommended by NIST and is well-suited for elliptic curve cryptographic (ECC) applications is presented. To achieve the maximum architectural and timing improvements we have reorganized and reordered the critical path of the Lopez-Dahab scalar point multiplication architecture such that logic structures are implemented in parallel and operations in the critical path are diverted to noncritical paths. With *G=41*, the proposed design is capable of performing a field multiplication over the extension field with degree 163 in 11.92 μs with the maximum achievable frequency of 251 MHz on Xilinx Virtex-4 (XC4VLX200) while 22% of the chip area is occupied, where *G* is the digit size of the underlying digit-serial finite field multiplier.

***Keywords*—**Elliptic Curve Cryptography, FPGA implementation, Scalar point multiplication

I. Introduction

ELLIPTIC CURVE CRYPTOGRAPHY (ECC) is a public key cryptography system superior to the well-known RSA cryptography: for the same key size, it gives a higher security level than RSA [1, 2]. Intuitively, there are numerous advantages of using field-programmable gate-array (FPGA) technology to implement in hardware the computationally intensive operations needed for ECC. These advantages are comprehensively studied and listed by Wollinger, et. al. in [3]. In particular, performance, cost efficiency, and the ability to easily update the cryptographic algorithm in fielded devices are very attractive for hardware implementations. Several recent FPGA-based hardware implementations of ECC have achieved high-performance throughput and efficiency. In this work we present a new architecture as well as an efficient ECC FPGA implementation over $GF(2^{163})$ that has considerable advantages compared to other implementations as regards to speed and area. The proposed architecture is based on a modified Lopez-Dahab elliptic curve point multiplication algorithm [4] in which we have reorganized and reordered the data path carefully to achieve maximum performance and efficiency. As we know, the efficiency of an algorithm is measured by the scarce resources it consumes. Typically the measure used is time, but sometimes other measures such as space and number of processors are also considered. Our basic strategy for architectural timing improvement is to reorganize the critical path such that logic structures are implemented in parallel. Usually, this technique is used whenever a function that currently evaluates through a serial string of logic can be broken up and evaluated in parallel.

M. M. and H. M. are with Islamshahr Islamic Azad University, Tehran, Iran. (email:m_masoumi@eetd.kntu.ac.ir, m.mahdizade@yahoo.com)

By using a modified field multiplier and two squarer modules for separating the paths in which squaring is repeated several times we have designed an efficient architecture for the Itoh-Tsujii Multiplicative Inverse Algorithm (ITMIA) [5]. In the design of the ECC processor, we have separated sequentially executed operations into parallel operations and have carefully reordered paths to divert operations in the critical path to noncritical paths in order to minimize the combinatorial delay of the critical path. The architecture of the ECC processor has been designed in such a way that the calculations of point addition are separated and are performed independent of the key which in turn considerably reduces the processing delay. The results we obtained show that by using the mentioned optimization techniques and by implementing a modified *G*-bit digit serial finite-field multiplier, with $G = 41$ our proposed design is able to compute $GF(2^{163})$ elliptic curve scalar point multiplication operations in 11.92 μs with the maximum achievable frequency of 251 MHz on Xilinx Virtex-4 (XC4VLX200) while 19606 slices or 22% of the chip area is occupied which makes the design suitable for high speed applications. The organization of the article is as follows: In Section 2, a brief introduction of the mathematical background of ECC is presented. In Section 3, the algorithm optimization decomposition in parallel and resource occupation for implementation of the modular arithmetic logic unit and the finite field arithmetic units in hardware are detailed. In Section 4 the proposed architecture for ECC processor is illustrated. In section 5, the implementation results and performance obtained are compared with those in other published works. Finally, in the conclusions we summarize the results of our discussions.

II. Mathematical Background

A. Mathematical Background

It has been turned out that the form of cubic equation appropriate for elliptic curve cryptographic applications which has been recommended by NIST is [1, 6]

$$y^2 + xy = x^3 + ax^2 + b \left(mod\, P(x)\right) \quad (1)$$

where it is understood that the variables x and y and the coefficients a and b are elements of $GF(2^m)$ and calculations are performed in $GF(2^m)$. Let us consider the finite field $GF(2^{163})$ generated using the irreducible polynomial $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$1. This is a NIST recommended field for ECC applications. An elliptic curve group over $GF(2^m)$ consists of the points on the corresponding elliptic curve, together with a point at infinity, $\mathcal{O}$. The set of points that satisfy the Eq. (1) together with the element $\mathcal{O}$ forms an addition Abelian group with respect to the elliptic point addition operation. $\mathcal{O}$ serves as the additive identity.

Thus, $\mathcal{O} = -\mathcal{O}$ and for any point P on the curve, $P + \mathcal{O} = P$ and $P + (-P) = \mathcal{O}$. It can be shown that a finite Abelian group can be defined based on the set $E_{2^m}(a,b)$, provided that $b \neq 0$. The rules for addition can be stated as follows. For all points $P, Q \in E_{2^m}(a,b)$:

1) $P + \mathcal{O} = P$.

2) If $P = (x_P, y_P)$, then $-P + (x_P, y_P) = \mathcal{O}$. The point $(x_P, x_P + y_P)$ is the negative of P, denoted as $-P$.

3) If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ with $P \neq Q$ and $P \neq -Q$, then $R = P + Q = (x_R, y_R)$ is determined by the following rules:

$$x_R = \lambda^2 + \lambda + x_P + x_Q + a \quad (2)$$
$$y_R = \lambda(x_P + x_R) + x_R + y_P$$

where $\lambda = \frac{y_Q + y_P}{x_Q + x_P}$

4) If $P = (x_P, y_P)$ then $R = 2P = (x_R, y_R)$ is determined by the following rules:

$$x_R = \lambda^2 + \lambda + a \quad (3)$$
$$y_R = x_P^2 + (\lambda + 1)x_R$$

where

$$\lambda = x_P + \frac{y_P}{x_P}$$

B. Elliptic Curve Cryptography

It has been shown that the points on an elliptic curve can be represented using either two or three coordinates. In affine-coordinate representation, a finite point on $E(GF(2^m))$ is specified by two coordinates $x, y \in GF(2^m)$ satisfying Eq. (2) and (3). We can make use of the concept of a projective plane over the field $GF(2^m)$ [2]. In this way, one can represent a point using three rather than two coordinates. Then, given a point P with affine-coordinate representation x, y there exists a corresponding projective-coordinate representation X, Y and Z such that, $P(x;y) = P(X;Y;Z)$. As a means of avoiding the expensive field inversion operation, it is more convenient to work with Lopez-Dahab (LD) projective coordinates which is highly attractive for hardware implementations. The Lopez-Dahab algorithm is shown in Fig. 1.

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$ with $k_{t-1} = 1$, $P = (x_P, y_P) \in E(F_{2^m})$.
OUTPUT: kP.
1. $X_1 \leftarrow x_P$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x_P^4 + b$, $Z_2 \leftarrow x_P^2$. {Compute $(P, 2P)$}
2. For i from $t-2$ downto 0 do
  2.1 If $k_i = 1$ then
    $T \leftarrow Z_1$, $Z_1 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_1 \leftarrow x_P Z_1 + X_1 X_2 T Z_2$.
    $T \leftarrow X_2$, $X_2 \leftarrow X_2^4 + bZ_2^4$, $Z_2 \leftarrow T^2 Z_2^2$.
  2.2 Else
    $T \leftarrow Z_2$, $Z_2 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_2 \leftarrow x_P Z_2 + X_1 X_2 Z_1 T$.
    $T \leftarrow X_1$, $X_1 \leftarrow X_1^4 + bZ_1^4$, $Z_1 \leftarrow T^2 Z_1^2$.
3. $x_3 \leftarrow X_1/Z_1$.
4. $y_3 \leftarrow (x_P + X_1/Z_1)[(X_1 + x_P Z_1)(X_2 + x_P Z_2) + (x_P^2 + y)(Z_1 Z_2)](x_P Z_1 Z_2)^{-1} + y_P$.
5. Return (x_3, y_3)

Fig. 1 The Lopez-Dahab scalar point multiplication over $GF(2^m)$ [4]

III. Hardware Architectures For Finite Field Operations Over GF(2^M)

A. Finite Field Reduction

Assuming that we have already computed the product polynomial $D(x) = A(x)B(x)$ and we want to obtain the modular product of $C(x)$ such that

$$C(x) = D(x) \bmod P(x) \quad (4)$$

Recall that the polynomial product D and the modular product C; have $2m-1$ and m; coordinates, respectively, i.e.,

$$D = [d_{2m-2}, d_{2m-3}, \ldots, d_{m+1}, d_m, \ldots, d_1, d_0];$$
$$C = [c_{m-1}, c_{m-2}, \ldots, c_1, c_0]; \quad (5)$$

One of the most efficient approaches for hardware implementation is reduction in finite fields using fast reduction algorithm corresponding to the field polynomial. Fig. 2 represents the implementation of the reduction modulo $P(x)$ used in this article. It has been assumed that the maximum degree of $D(x)$ is equal to $162+G$ in which the sentences with degree $163 \leq i \leq 162+G$ are mapped to the sentences with degree $i < 163$.

Input: $D = [d_{162+G}, d_{161+G}, \ldots, d_1, d_0]$,
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$;
Output: $C = [c_{162}, c_{161}, \ldots, c_1, c_0]$;
if $G = 0$ then
$C \leftarrow D$;
else
$[c_{162}, \ldots, c_G] \leftarrow [d_{162-G}, \ldots, d_0]$;
$[c_{G-1}, \ldots, c_0] \leftarrow 0$;
for i from 1 to G do
$c_{i-1} \leftarrow c_{i-1}\ xor\ d_{163+i-1}$;
$c_{3+i-1} \leftarrow c_{3+i-1}\ xor\ d_{163+i-1}$;
$c_{6+i-1} \leftarrow c_{6+i-1}\ xor\ d_{163+i-1}$;
$c_{7+i-1} \leftarrow c_{7+i-1}\ xor\ d_{163+i-1}$;
Return C;

Fig. 2 Reduction algorithm for $C(x) = D(x) \bmod P(x)$.

B. Finite Field Multiplication

Field multiplication is by far the most costly arithmetic operation which directly affects the working frequency and speed of the ECC processor [2]. One can make an speed-area trade-off by using a serial-parallel strategy, in which the multiplication of two arbitrary field elements is accomplished by using a procedure inspired in the well-known digit-serial/parallel (LSD) finite field multipliers. In this work, we have designed LSD multiplier directly at digit-level. Based on [7], LSD multiplication algorithms are classified as least significant digit (LSD) first and most significant digit (MSD) first algorithms. It has been shown that the LSD first algorithm consumes fewer gates and has shorter critical path compared with the MSD first algorithm. Various approaches have been proposed for efficient implementation of the LSD multiplier. With digit size G, the total number of digits in $GF(2^m)$ will be $n = \lceil m/G \rceil$.

Assume $A = \sum_{j=0}^{m-1} a_j \alpha^j$ and $B = \sum_{j=0}^{m-1} b_j \alpha^j$ such that

$$B_i = \begin{cases} \sum_{j=0}^{G-1} b_{G*i+j}\alpha^j & 0 \le i \le n-2 \\ \sum_{j=0}^{m-1-G(n-1)} b_{G*i+j}\alpha^j & i = n-1 \end{cases} \quad (6)$$

$$C = A * B \bmod P(x) = \sum_{j=0}^{m-1} c_j \alpha^j$$

$$= \begin{pmatrix} B_0 A + B_1\left(A\alpha^G \bmod f(x)\right) + B_2\left(A\alpha^G \cdot \alpha^G \bmod P(x)\right) \\ +B_{n-1}\left(A\alpha^{G*(n-2)} \cdot \alpha^G \bmod P(x)\right) \end{pmatrix} \bmod P(x) \quad (7)$$

The LSD algorithm is represented in Fig. 3.

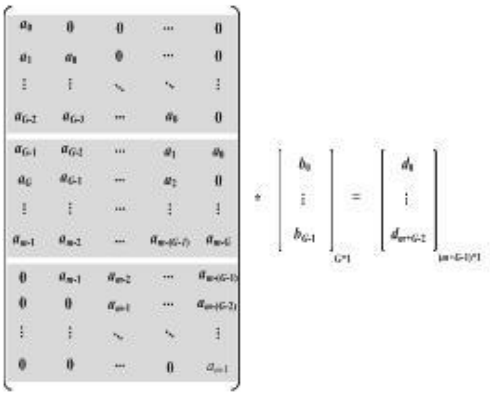
Input: $A,B \in GF(2^m)$
Output: $C \in GF(2^m)$, $C = AB$ over $GF(2^m)$
Set: $A^{(0)} = a$, $D^{(0)} = 0$, $n = \lceil m/G \rceil$
for i **from 1 to** n **do**
1) $A^{(i)} = A^{(i-1)}\alpha^G \bmod P(x)$,
2) $D^{(i)} = A^{(i-1)} \cdot B_{i-1} + D^{(i-1)}$
Where
$A^{(i)} = \sum_{j=0}^{m-1} A_j^{(i)}\alpha^j$
$D^{(i)} = \sum_{j=0}^{m+G-2} d_j^{(i)}\alpha^j$ **and**
$B_i = \begin{cases} \sum_{j=0}^{G-1} b_{G*i+j}\alpha^j & 0 \le i \le n-2 \\ \sum_{j=0}^{m-1-G*(n-1)} b_{G*i+j}\alpha^j & i = n-1 \end{cases}$
end for
3) Return $C = D^{(n)} \bmod P(x)$

Fig. 3 The LSD multiplication algorithm [7]

Consider the two-step classical multiplication in $GF(2^m)$ which involves in a polynomial multiplication and a reduction modulo an irreducible polynomial. The product of the polynomials $A(x)$ and $B(x)$, $D(x)=A(x)\times B(x)$, is a polynomial with maximum degree $2m-2$ and can be written as follows.

$$D = \begin{cases} \sum_{i=0}^{k} a_i b_{k-i}\ ;\ k = 0, \cdots, m-1 \\ \sum_{i=k}^{2m-2} a_{k-i+(m-1)} b_{i-(m-1)}\ ;\ k = m, \cdots, 2m-2 \end{cases} \quad (8)$$

We implemented the above scheme in a matrix form. Thus, we put A in a three-section multiplicand matrix. The upper part is a lower triangular submatrix. The middle part is a $(m-G+1)\times G$ submatrix. The lower part is an upper triangular submatrix.



(9)

By converting Eq. (8) into matrix form Eq. (10), the G^{th} term of polynomial $D(x)$, d_G, can be expressed as Eq. (10).

$$d_G = a_{G-1}b_0 + a_{G-1}b_1 + a_0 b_{G-1} \quad (10)$$

where G is the digit size of the underlying LSD multiplier. As it is seen in Fig. 3, there are three steps for implementing the LSD algorithm. Steps 1 and 2 of the LSD multiplier as is represented in Eq. (11) can be implemented in parallel.

$$\begin{aligned} A^{(i)} &= A^{(i-1)}\,\alpha^G \bmod P(x) \\ D^{(i)} &= A^{(i-1)}\,B_{i-1} + D^{(i-1)} \end{aligned} \quad (11)$$

Step 1 of the LSD algorithm reduces $m+G$ bits to m bits and step 2 shows the partial products. The final result is obtained in step 3 in which $m+G-1$ bits are reduced to m bits. The implementation architecture for different stages of the LSD multiplier is depicted in Fig. 4. Step 1 is performed by the left side of Fig. 4, step 2 is performed by the multiplication function and step 3 is performed by the right side of Fig. 4.

C. Finite-Field Multiplicative Inversion

Based on Fermat's Little Theorem (FLT) and using an ingenious rearrangement of the required field operations, the Itoh-Tsujii Multiplicative Inverse Algorithm (ITMIA) was presented in [5]. The main advantage of ITMIA algorithm in comparison with the Extended Euclidian Algorithm is that it does not require a separate inversion module. When computing the multiplicative inverse using ITIMA algorithm, 81 squaring must be iteratively performed in the algorithm's addition chain. These iterative computations are done sequentially and therefore further parallelism is not possible [2]. Now, to design an efficient multiplicative inversion block based on the ITMIA, it is necessary to think how to reduce its critical path. In other word, the critical path of the multiplier and the critical path of the inversion block should be along each other. If we use only one squarer module in the inversion

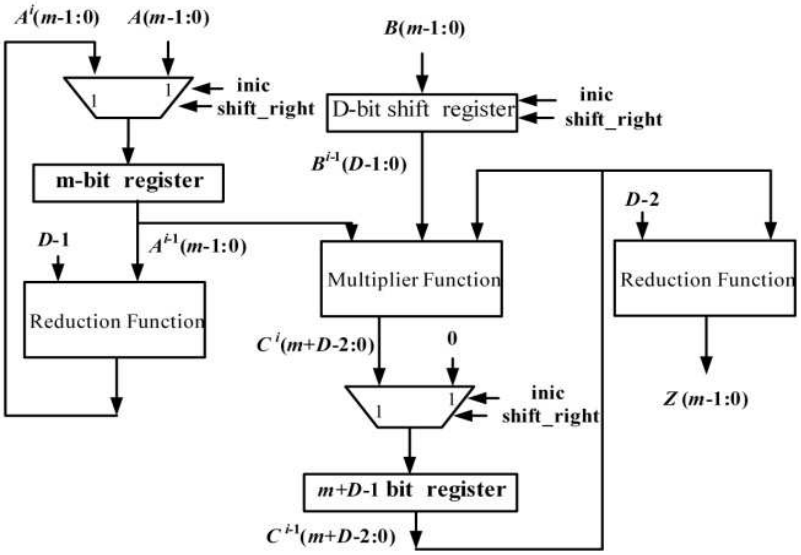


Fig. 4 Block diagram of the LSD multiplier implemented in this work

D.Finite-Field Multiplicative Inversion

Based on Fermat's Little Theorem (FLT) and using an ingenious rearrangement of the required field operations, the Itoh-Tsujii Multiplicative Inverse Algorithm (ITMIA) was presented in [5]. The main advantage of ITMIA algorithm in comparison with the Extended Euclidian Algorithm is that it does not require a separate inversion module. When computing the multiplicative inverse using ITIMA algorithm, 81 squaring must be iteratively performed in the algorithm's addition chain. These iterative computations are done sequentially and therefore further parallelism is not possible [2]. Now, to design an efficient multiplicative inversion block based on the ITMIA, it is necessary to think how to reduce its critical path. In other word, the critical path of the multiplier and the critical path of the inversion block should be along each other. If we use only one squarer module in the inversion block, this module should accomplish squaring for the input of the inversion block, output of the multiplier and also its own output (for consecutive squaring) and therefore, we are forced to use a 3 to 1 multiplexer at the input of the squarer. Output of this squarer together with a number of combinational gates such as AND, OR, and NOT gates are connected to the input of the multiplier. As a result of such architecture, the critical path will place on the squarer which will create a bottleneck for reducing the clock cycle time. We can break this critical path by changing the architecture so that a 2 to 1 multiplexer is used in place of a 3 to 1 multiplexer at the cost of adding another squarer in the inverter architecture. The first squarer is used for squaring at stages 1, 3, and 8 and also for the final stage squaring, while the other required squarings in Table I is accomplished with the second squarer, since at the stages 1, 3, and 8, $u_0 = 1$ and only one squaring need to be performed while at the other stages several squaring are performed (see appendix for more details). The schematics of the designed architecture for multiplicative inversion over finite field $GF(2^{163})$ is shown in Fig. 5.

TABLE I
$B_i(A)$ COEFFICIENT GENERATION FOR M-1 =162 [2]

i	u_i	*rule*	$\left[\beta_{u_{i_1}}(a)\right]^{2^{u_{i_2}}} \cdot \beta_{u_{i_2}}(a)$	$\beta_{u_i}(a) = a^{2^{u_i}-1}$
0	1	-	-	$\beta_{u_0}(a) = a^{2^1-1}$
1	2	$2u_0$	$\left[\beta_{u_0}(a)\right]^{2^{u_0}} \cdot \beta_{u_0}(a)$	$\beta_{u_1}(a) = a^{2^2-1}$
2	4	$2u_1$	$\left[\beta_{u_1}(a)\right]^{2^{u_1}} \cdot \beta_{u_1}(a)$	$\beta_{u_2}(a) = a^{2^4-1}$
3	5	u_0+u_2	$\left[\beta_{u_2}(a)\right]^{2^{u_0}} \cdot \beta_{u_0}(a)$	$\beta_{u_3}(a) = a^{2^5-1}$
4	10	$2u_3$	$\left[\beta_{u_3}(a)\right]^{2^{u_3}} \cdot \beta_{u_3}(a)$	$\beta_{u_4}(a) = a^{2^{10}-1}$
5	20	$2u_4$	$\left[\beta_{u_4}(a)\right]^{2^{u_4}} \cdot \beta_{u_4}(a)$	$\beta_{u_5}(a) = a^{2^{20}-1}$
6	40	$2u_5$	$\left[\beta_{u_5}(a)\right]^{2^{u_5}} \cdot \beta_{u_5}(a)$	$\beta_{u_6}(a) = a^{2^{40}-1}$
7	80	$2u_6$	$\left[\beta_{u_6}(a)\right]^{2^{u_6}} \cdot \beta_{u_6}(a)$	$\beta_{u_7}(a) = a^{2^{80}-1}$
8	81	u_0+u_7	$\left[\beta_{u_7}(a)\right]^{2^{u_0}} \cdot \beta_{u_0}(a)$	$\beta_{u_8}(a) = a^{2^{81}-1}$
9	162	$2u_8$	$\left[\beta_{u_8}(a)\right]^{2^{u_8}} \cdot \beta_{u_8}(a)$	$\beta_{u_9}(a) = a^{2^{162}-1}$

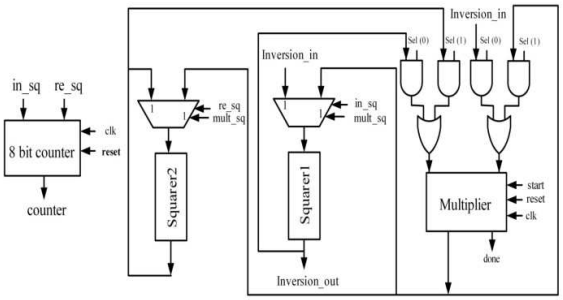


Fig. 5 Schematic of the designed architecture for finite field multiplicative inversion

IV. The Proposed Architecture for the ECC Processor

As was mentioned, the most important strategy for architectural timing improvements is to reorganize and reorder the critical path such that logic structures are implemented in parallel and to divert operations in the critical path to a noncritical path. This technique should be used whenever a function that currently evaluates through a serial string of logic can be broken up and evaluated in parallel .This assumption can dramatically speed up the implementation of a large design. For the design of architecture for ECC scalar multiplier, two different parts are considered; the first part that involves in calculations in the affine coordinate system and the other part that involves in the calculations for converting projective coordinate to affine coordinates. For projective calculations, parts 1 and 2 of the LD algorithm are considered. In the design of this part of the processor, the number of computational units is chosen in such a way that allows parallel computations to be performed. Hence, we use three field multipliers to implement the main loop of the algorithm in which point addition and doubling are carried out. So, according to section 2.1 of the LD algorithm, at the first stage, the three multiplications X_1Z_2, X_2Z_1, TZ_2 (T$\rightarrow X_2$) are performed in parallel by using three multipliers as is shown in Fig. 6, and then, the three other multiplications $x_P Z_1$, $X_1X_2T\ Z_2$ ($T\leftarrow Z_1$), bZ_2^4 are accomplished in parallel at the second stage. Hence, the delay of each iteration is reduced from six field multiplication delay to two field multiplications. For this part of the processor (computations in the projective coordinates) we have used five squarers and two adders, as is shown in Fig. 6. Four squarers are used for computing Z_1^4, X_1^4, Z_2^4 and X_2^4 while the fifth squarer is used for $(X_1Z_2+X_2Z_1)^2$. In addition, It is essential after the first field multiplication to save the result of $(X_1Z_2+X_2Z_1)^2$ and $(X_2^4+bZ_2^4)$ in the registers t_1 and t_2 respectively for the later calculations. The most important modules in the design of the scalar point multiplier processor are field multiplication, field inversion and field squaring. The key point here is that the critical path must be placed on the longest path among these modules. Since the inverter module was designed such that its critical path is coincided with the multiplier's critical path and since the multiplier's path is larger than the squarer's path, the critical path need to be placed on the multiplier. Please notice that if resource sharing is used in implementing the field squarer, the number of required computational elements will decrease; however, since for squaring of different values we are forced to use multiplexers at the input of this computational unit that are controlled with conditional statements, the critical path length will increase. To avoid long critical path, the architecture should be designed synchronous and by using combinational logic. In addition, in the design of the projective calculations, separate calculations have not been performed for using the initial values of part 1 of the LD algorithm, since if further computational modules are designed for these calculations, the complexity of the critical path and the amount of required area will increase. We can avoid additional or unnecessary calculations by using calculations of part 2 of the algorithm for obtaining the results for part 1.

In the proposed design, calculations of part 1 need to be performed whenever the most significant bit of the key is 1. So, when $k_i = 1$, if the values of Eq. (12) are used in the calculations of part 2.1, then the required initial values of the LD algorithm are obtained in accordance with part 1 of the LD algorithm.

$$X_1\leftarrow 1,\ Z_1\leftarrow 0,\ X_2\leftarrow x_P,\ Z_2\leftarrow 1 \tag{12}$$

The results of the calculation in section 2.1 of the LD algorithm are obtained as Eq. (13) by using the values of Eq. (13).

$$X_1\leftarrow x_P,\ Z_1\leftarrow 1,\ X_2\leftarrow x_P^4+b,\ Z_2\leftarrow x_P^2 \tag{13}$$

As it is seen in Fig. 7, whenever the key bit is equal to 1, the values of '1', '0', and x_P are entered into the multiplexers to connect to the appropriate inputs to make the terms of Eq. (13). After designing the computational units for projective coordinates, its input and output ports should be connected together based on the key bits to complete the iteration in the LD algorithm. When designing the architecture for calculations in the projective coordinate system in part 2.1 of the LD algorithm, to set up part 2.2 of the algorithm which works with zero bits of the key, it is enough to swap X_1 and Z_1 with X_2 and Z_2 respectively when the key bits change. So, we need to use a 2 to 1 multiplexer that is controlled with the key bits. Therefore, in order to avoid long critical path, another strategy should be considered. As it is seen from the architecture of Fig. 7, in order to prevent further complexity when swapping X_1 and Z_1 with X_2 and Z_2, the input-output paths of point addition and doubling have been separated from each other. The idea behind this subject is to connect the outputs of point addition and doubling to the inputs of the adder, independent of the values of the key bits .For example, if we consider the following point addition operation for $k_i = 1$, inputs to this operation are X_1, X_2, Z_1 and Z_2 and outputs are saved in X_1 and Z_1.

$$T\leftarrow Z_1,\ Z_1\leftarrow (X_1Z_2+X_2Z_1)^2,\ X_1\leftarrow x_PZ_1+X_1X_2TZ_2 \tag{14}$$

When a key bit changes from $k_i = 1$ to $k_i = 0$, this change will lead to change in the terms $X_1Z_2+X_2Z_1$ and $X_1X_2TZ_2$. However, since whenever the value of any key bit changes only X_1 and Z_1 are swapped with X_2 and Z_2, the terms $X_1Z_2+X_2Z_1$ and $X_1X_2TZ_2$ will remain unchanged. So, the point addition operation can be repeated in the iterative part of the algorithm without involvements of the key bits and only after the end of the loop, the registers are swapped with each other. The point doubling operation for $k_i = 1$ is performed in accordance with Eq. (15).

$$T\leftarrow X_2,\ X_2\leftarrow X_2^4+bZ_2^4,\ Z_2\leftarrow T^2Z_2^2 \tag{15}$$

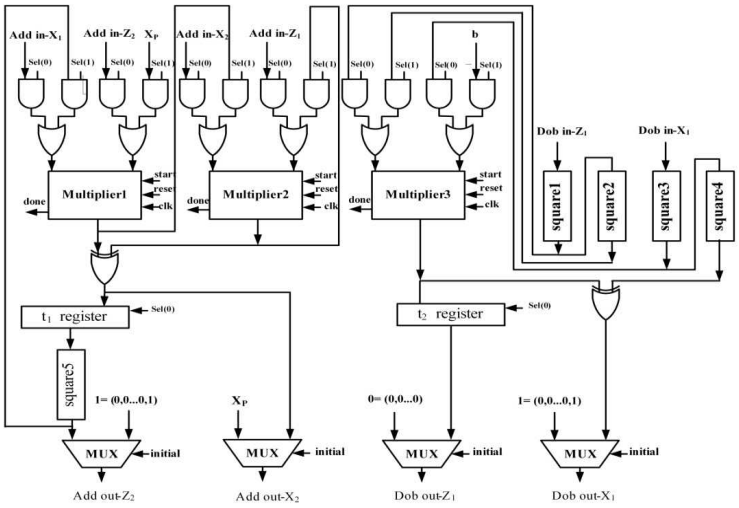


Fig. 6 The architecture designed for the computation of point addition and point doubling in projective coordinates of the LD algorithm

This operation for $k_i = 0$ is done by swapping X_2 and Z_2 with X_1 and Z_1 respectively. Therefore, output registers are swapped in order to provide proper inputs for the point doubling operation based on the key bits in the iterative part of the LD algorithm. In order to realize that when the initial values are entered into the calculations and also to be aware of the iterations of the LD algorithm based on the key bits, it is necessary to combine the module designed in Fig. 8 with a key shift register in a new structure. The aim of this work is that the inputs and outputs of the architecture of Fig. 8 are properly connected to each other when all values of the key are scanned. The new design is shown in Fig. 7. The second part of the processor involves in calculations that convert projective coordinates to affine coordinates. It is obvious from the LD algorithm that parts 3 and 4 of this algorithm require many calculations to be implemented. In addition, most of the calculations are performed in a sequential manner. A possible sequence of the instructions from standard Projective to affine coordinates is proposed in [2] in which only one inversion unit is used for converting projective coordinates to affine coordinates. As it is seen form the LD algorithm, by calculating $(x_P Z_1 Z_2)^{-1}$, another inversion, X_1/Z_1, can be calculated using $(x_P Z_2 X_1)*(x_P Z_1 Z_2)^{-1}$. In this approach, the number of field inverters is reduced with the cost of increase in the number of field multipliers. However, considering the sequence of the algorithm and due to repeated referrals to these multipliers, if we use several field multipliers the length of the critical path will increase. For implementing this algorithm, ten field multiplications should be performed. In addition, for performing twelfth to seventeenth steps, we need to wait for the calculation of $(x_P Z_1 Z_2)^{-1}$ and therefore a long computational delay will be inevitable. As it is seen from the second part of the scalar multiplier processor which involves in converting projective coordinates to affine coordinates, there are many computations that should be done sequentially.

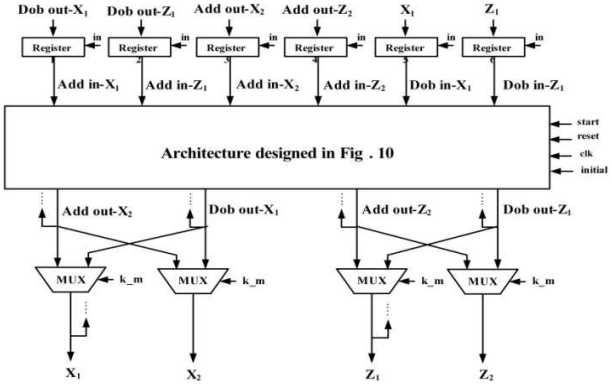


Fig. 7 Architecture of point addition and doubling iteration based on the key bits

As was mentioned, in order to keep the critical path on the multiplier, we need to design this part of the algorithm with combinational logic as much as possible. Another approach for the implementation is based on step 4 of the LD algorithm as it is seen in Eq. (16).

$$y_3 \leftarrow (x_P+X_1/Z_1)[(X_1+xZ_1)(X_2+x_PZ_2)+(x_P^2+y_p)(Z_1 Z_2)](x_PZ_1Z_2)^{-1} + y_P \quad (16)$$

There are two field inversions and five field multiplications in Eq. (17). One way to implement the above function is to use two field inverter and three parallel field multiplier units. However, this causes that these multipliers to remain unused in other stages since the results of multiplications in the next steps are dependent to the results of the previous steps. This subject will cause an unbreakable delay which prohibits further speed up. Another design that leads to more efficient

implementation is to enter $(Z_1Z_2)^{-1}$ in the square brackets of Eq. (16). This will result in Eq. (17).

$$y_3 \leftarrow (x_P + X_1/Z_1)[(X_1/Z_1 + x_P)(X_2/Z_2 + x_P) + (x_P^2 + y)] \quad (x_P)^{-1} + y_P \tag{17}$$

Therefore, first we calculate Z_1^{-1}, Z_2^{-1} and x_P^{-1} using three parallel field inverters concurrently and then implement five required multiplications of Eq. (17) by using two multipliers that are implemented in parallel in three stages. Also, for this part of calculations we also need five adder units. The final value of variable x in affine coordinate system in accordance with part 2 of the LD algorithm is $x_3=X_1/Z_1$ for which we have to calculate Z_1^{-1} by using an inverter and then multiply the result by X_1. Since $X_1*Z_1^{-1}$ is used for the next multiplication, $(X_1/Z_1+x_P)*(X_2/Z_2+x_P)$, it is necessary to save the result of $X_1*Z_1^{-1}$. However, saving this value in a register and using it in next clock cycles will increase the critical path. To avoid this, this register should be eliminated. Since in the conversion of coordinates, implementation of the multipliers have been done in a parallel combinational manner (i.e., five multiplications are performed in three stages using two multipliers), in the second stage of multiplication the result of first multiplication will be lost. However, in the third stage of multiplication one of the multipliers is unused and could be used for calculating $X_1*Z_1^{-1}$. So, the multiplication $X_1*Z_1^{-1}$ is repeated in the third stage to eliminate the need for saving data in this section of the processor. Finally, one of the important steps that must be considered in the design of scalar multiplier is to select the word length (G). Due to iterative calculations in the projective coordinate system (part 2 of the LD algorithm), fast performing of calculations is very important in the design of an efficient ECC processor. So, choosing large G values for the multipliers used in the design of the first part of the processor (i.e., the multipliers in Fig. 7 or projective calculations) will be more appropriate. The word lengths that were used in this part of the processor is G_1= 41. Since calculations of the third and fourth part of the LD algorithm are used only once at the end of the algorithm and there is no iteration as part 2 of the algorithm, there is no need to select large values for G. Instead, since there are relatively a large number of computational units in this part of the processor, a relatively small value for G should be chosen to reduce the required implementation area. The word's length used in this part of the processor is G_2=11.

V. Implementation Results

The ECC processor was implemented using synthesizable VHDL codes on Xilinx XC4VLX200. Performance of the proposed scalar multiplication for is shown in table II. The proposed design completes one scalar point multiplication in $326 * (\lceil m/G_1 \rceil) + 12 * (\lceil m/G_2 \rceil) + 1509$ cycles. The number of required clock cycles for ECC point multiplication with G_1 = 41 and G_2 =11 is 2993 cycles. The term "$\lceil m/G_1 \rceil$" indicates the number of cycles required to perform finite field multiplication in part 2 of the LD algorithm or calculations in the projective coordinate system. The term "$\lceil m/G_2 \rceil$" indicates the number of cycles required to perform finite field multiplication in parts 3 and 4 of the LD algorithm or calculations for converting projective coordinates to affine coordinates. In order to decide how efficient a design is, we utilize the efficiency defined as $\frac{Thoughput}{Area}$ ($\frac{\frac{Mbit}{s}}{slices}$) as a figure of merit, where Throughput is defined as $\frac{working\ frequency \times Number\ of\ Bits}{Number\ of\ Cycles}$ and hardware area can be defined as number of four inputs LUTs as well as CLB slices. Table II presents performance of the proposed scalar multiplier. The last column in this table shows the algorithmic efficiency defined as throughput/area. It would be more accurate to use throughput/#slices, but slice counts were not reported by the authors of some other designs. Therefore, we have used throughput/#LUTs. In Table III, a number of high speed elliptic curve processors (ECP) are compared with the proposed one. As it is seen from table III, the proposed design is more efficient than the other designs reported in the open literature expect one of the proposed schemes reported in [15]. Please note that although that design utilizes 4.82 times less LUT compared with our design, it is almost 4 times slower than our design with G = 41. The design proposed by Kim et.al. in [19] is almost 15% faster than our design but in consumes 25% more resources than our implementation.

TABLE II
Performance of the Proposed Scalar Multiplier

G_1	G_2	Freq. (MHz)	Time (µs)	No. of Cycles	Area (Slices)	Area (LUT)	Efficiency
41	11	251.054	11.92	2993	19604	36727	372.1

TABLE III
Performance of the Scalar Multipliers

Ref.	FPGA	Freq. (MHz)	Time (µs)	Area (Slices)	Area (LUT)	Efficiency
[8]	XCV400E	76.7	210	-	3002	265
[9]	XCV2000E	66.4	144	-	20068	56
[10]	VinexII V8000	90.2	106	18079	-	44
[11]	XCV2600E	46.5	63	18314 + 24 RAMs	-	-
[12]	XC2V600-4	54	60	-	-	-
[13]	Virtex II pro 30	100	280	8450	-	-
[14]	Virtex-4 VLX200	153.9	19.55	16209	26364	316
[15]	XC2V2000	100	46.5	3416	7559	532
[16]	XC2V6000	93.3	34.11	13376	2812	340
[17]	Virtex 2000E	66	75	-	10017	70
[18]	Stratix II	-	49	-	-	-
[19]	XC4VLX80	143	10	24,363	-	-

VI. Conclusions

A high-performance ECC processor was implemented using FPGA technology. We used a careful parallel implementation

strategy to reduce the critical path of the Itoh-Tsujii's Finite-Field Inversion. In addition, in the design of the ECC processor, by using three parallel multiplier units and reducing the number of unused cycles in each stage we reduced the processor delay which is mainly related to the calculations in the projective coordinate system. Separation of point doubling path from point addition path and using appropriate initial values for the initial setup of the processor reduced the complexity of the processor. The results show that the designed architecture can be well suited to the applications that require high performance.

REFERENCES

[1] D. Hankerson, A. Menezes, S. Vanstone, Guide to elliptic curve cryptography, Springer, 2004.
[2] Rodriguez-Henriquez et.al, Cryptographic Algorithms on Reconfigurable Hardware, Springer, 2006.
[3] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: State-of-the-art and Implementations Attacks," ACM Trans. on Embedded Computing Sys., 3(3):534- 574, 2004.
[4] J. Lopez and R. Dahab, "Fast multiplication on elliptic curves over GF(2m) without precomputation,", CHES, MA, USA, 1999.
[5] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in GF(2m) Using Normal Basis," Information and Computing, 78:171-177, 1988.
[6] W. Stallings, Cryptography and Network Security, 4th Ed., Prentice-Hall, 2006.
[7] S. Kummar, T. Wollinger, and C. Paar, "Optimum Digit Serial GF(2m) Multipliers for Curve Based Cryptography," IEEE Trans. Comp., vol. 55, no 10. 2006
[8] G. Orlando and C. Paar., "A high-performance reconfigurable elliptic curve processor for GF(2m)," CHES, MA, USA, 2000.
[9] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, "An end-to-end systems approach to elliptic curve cryptography," CHES, CA, USA, 2002.
[10] K. Jarvinen, M. Tommiska, and J. Skytta, "A scalable architecture for elliptic curve point multiplication," ICFPT, Brisbane, Australia, 2004.
[11] F. Rodriguez-Henriquez, N. A. Saqib, and A. Diaz-Perez, "A fast parallel implementation of elliptic curve point multiplication over GF(2m)," Microprocessors Microsyst., vol. 28, pp. 329–339, 2004.
[12] R. C. C. Cheung, N. J. Telle, W. Luk, and P. Y. K. Cheung, "Customizable elliptic curve cryptosystems," IEEE Trans. Very Large Scale Integr. (VLSI) Syst.", vol. 13, no. 9, pp. 1048–1059, Sep. 2005.
[13] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Superscalar coprocessor for high-speed curve-based cryptography," CHES, Yokohama, Japan, 2006
[14] W.N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on FPGA, "IEEE Trans. on Very Large Scale Integration (VLSI) Systems," vol. 16, no. 2, Feb. 2008, pp. 198-205.
[15] B. Ansari and a. Hasan, "High-Performance Architecture of Elliptic Curve Scalar multiplication", IEEE Trans. on Comp., Vol. 57, No. 11, pp. 1443-1453, Nov. 2008.
[16] Yong-ping Dan et. al., "High-performance hardware architecture of elliptic curve cryptography processor over GF(2163), J. Zhejiang Univ. Sci., A 2009 10(2):301-310
[17] J. Lutz and Hasan, A., "High performance FPGA based elliptic curve cryptographic coprocessor," ITCC, Las Vegas, USA, Apr. 5-7, vol. 2, pp. 486-492, 2004,
[18] K. Jarvinen and J. Skytta, "On parallelization of high-speed processors for elliptic curve cryptography," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 16 (9) (2008) 1162–1175.
[19] C. H. Kim, S. Kwon, C. P. Hong, "FPGA implementation of high performance elliptic curve cryptographic processor over GF(2163)," J. of Sys. Architecture, 54 (10) (2008) 893–900