High-speed Hardware Implementations of Point Multiplication for Binary Edwards and Generalized Hessian Curves

Bahram Rashidi[1], Reza Rezaeian Farashahi[2], Sayed Masoud Sayedi[3]
[1,3]Dept. of Elec. & Comp. Eng., Isfahan University of Technology, Isfahan 84156-83111, Iran
[2]Dept. of Mathematical Sciences, Isfahan University of Technology, Isfahan 84156-83111, Iran
[1]b.rashidi@ec.iut.ac.ir, [2]farashahi@cc.iut.ac.ir, [3]m_sayedi@cc.iut.ac.ir,

***Abstract*— In this paper high-speed hardware architectures of point multiplication based on Montgomery ladder algorithm for binary Edwards and generalized Hessian curves in Gaussian normal basis are presented. Computations of the point addition and point doubling in the proposed architecture are concurrently performed by pipelined digit-serial finite field multipliers. The multipliers in parallel form are scheduled for lower number of clock cycles. The structure of proposed digit-serial Gaussian normal basis multiplier is constructed based on regular and low-cost modules of exponentiation by powers of two and multiplication by normal elements. Therefore, the structures are area efficient and have low critical path delay. Implementation results of the proposed architectures on Virtex-5 XC5VLX110 FPGA show that then execution time of the point multiplication for binary Edwards and generalized Hessian curves over GF(2^{163}) and GF(2^{233}) are 8.62µs and 11.03µs respectively. The proposed architectures have high-performance and high-speed compared to other works.**

Keywords: Elliptic Curve Cryptosystems, Point multiplication, Finite Fields, Gaussian normal basis, Binary Edwards curves, generalized Hessian curves, FPGA.

1. Introduction

The elliptic curve cryptosystem (ECC) was presented in mid 1980s independently by Neil Koblitz [1] and Victor Miler [2]. ECC is a public key (PKC) scheme with a relatively small key size in which elliptic curves over finite fields (Galois fields (GF)) are applied. The point multiplication or scalar multiplication is the main operation in this cryptosystem. Therefore, efficient implementation of this operation can lead to high-performance and high-speed crypto-processors. In recent years, for hardware implementation of ECC mostly binary finite fields GF(2^m) is considered, because addition operation in the binary field is carry free and can be realized by a simple bit-wise XOR, with a time delay of one XOR gate. In addition, in the normal basis, the squaring operation is implemented by a simple cyclic shift. It makes the binary finite fields with normal basis representation a suitable choice for efficient hardware implementation of ECC.

Many different FPGA-based hardware implementations of the point multiplication on binary elliptic curves have been reported [3-14]. For example, the proposed architecture in [6] is based on a modified Lopez-Dahab elliptic curve point multiplication algorithm, and uses Gaussian normal basis (GNB) for GF(2^{163}) field arithmetic. In [7] the structure is implemented in parallel. Also, the critical path of the Lopez-Dahab point multiplication architecture is reorganized and reordered so that the operations in the critical path are diverted into the noncritical paths. In [11] a theoretical model is used to approximate the delay of different field operations in point multiplication architecture implemented on *k*-input lookup-tables on FPGA. In addition, a suitable scheduling for performing point addition and doubling operations in the pipelined data path of the architecture is illustrated. The works presented in [12-14] are based on binary Edwards and generalized Hessian curves. In [12] the design of an FPGA-based binary Edwards curves processor is explained. In [13] by using parallelization technique in higher levels full resource utilization is achieved in computing point addition and point doubling formulas for both binary Edwards and generalized Hessian curves. Here, differential formulations for computing point multiplication are used through which a LUT-based pipelined and efficient digit-level GNB multiplier is employed. In [14] to reduce the latency of point multiplication, an analysis of data-flow and maximum number of parallel multipliers is employed. And the addition and doubling formulations are modified and a new proposed digit-level hybrid-double GNB multiplier is employed to remove the data dependencies and hence reduce the latency of point multiplication.

The present paper focuses on the hardware implementation of a high-speed and high-performance architecture of Montgomery ladder point multiplication for binary Edwards and generalized binary Hessian curves over GF(2^m). In the proposed structure, to reduce the number of clock cycles, field multipliers are in parallel form for computations of point addition and point doubling. In addition, our previously proposed efficient pipelined digit-serial Gaussian normal basis multiplier [15] is used. The multiplier has a highly regular structure with low critical path delay and low hardware resources. The proposed structures of point multiplication use only three and four units of field multipliers for the particular and general forms of binary Edwards curves respectively. Also, three field multipliers is used for generalized Hessian curves. The multipliers are shared and scheduled for lower number of clock cycles during point multiplication process.

The rest of the paper is organized as follows. In section 2, a mathematical background on the finite fields and a brief description of binary generic curves (BGCs), binary Edwards and generalized binary Hessian curves are presented. The structure of the digit-serial Gaussian normal basis multiplier is presented in section 3. The proposed structures of Montgomery ladder point multiplication for binary Edwards and generalized Hessian curves are presented in sections 4 and 5. Next, a comparison between this work and other related works is given in section 6. The paper is concluded in section 7.

2. Mathematics background

2.1. Gaussian Normal Basis

The binary finite fields are attractive fields for hardware implementation. In these fields the addition operation is implemented by a simple bit-wise XOR operation. Moreover, the efficiency of the multiplication operation depends on the method of representing elements of the binary finite field. There are two main applicable bases called polynomial basis (PB) and normal basis (NB). The multiplication is performed efficiently in PB, but the squaring is much more efficient in NB in terms of hardware implementation. There are special types of NB representation where the multiplication is also implemented efficiently. In this paper, we consider the Gaussian normal basis representation (GNB) for a binary finite field.
In the following a brief discussion on NB representation is presented. Let GF(2^m) be the binary finite field of order 2^m. The element β is a normal element of GF(2^m) if the set $\{\beta, \beta^2, \dots, \beta^{2^{m-1}}\}$ is a basis for the vector space GF(2^m) over GF(2). For any binary finite field such a basis exists, and it is called normal basis. Using β, every element A in GF(2^m) can be represented as $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in$ GF(2). For simplicity, we write $A = (a_{m-1}, a_{m-2}, \dots, a_1, a_0)$. The squaring of element A is computed as:

$$A^2 = a_{m-1}\beta + \sum_{i=1}^{m-1} a_{i-1}\beta^{2^i}$$

which can be performed by a simple cyclic shift as:

$$A^2 = (a_{m-2}, a_{m-3}, \dots, a_0, a_{m-1}).$$

Let C be the multiplication of elements A, B in GF(2^m), as:

$$C = AB = \sum_{i=0}^{m-1} a_i \beta^{2^i} \sum_{j=0}^{m-1} b_j \beta^{2^j} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \beta^{2^i+2^j}.$$

The element $\beta^{2^i+2^j}$ is represented by

$$\beta^{2^i+2^j} = \sum_{k=0}^{m-1} M_{ij}^{(k)} \beta^{2^k}, \quad M_{ij}^{(k)} \in \text{GF}(2).$$

Thus,

$$C = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \sum_{k=0}^{m-1} M_{ij}^{(k)} \beta^{2^k} = \sum_{k=0}^{m-1} c_k \beta^{2^k}, \qquad c_k = \sum_{i=0}^{m-1} \sum_{j-0}^{m-1} a_i b_j M_{ij}^{(k)}.$$

Here $M^{(k)}$ is an m by m symmetric matrix with entries $M_{ij}^{(k)}$ in GF(2). The matrix $M^{(k)}$ is computed by the k-cyclic right and down shift to entries of matrix $M^{(0)}$; i.e., for the indices $i, j, k = 0, \dots, m-1$, considered modulo m, we have $M_{i+1\,j+1}^{(k+1)} = M_{ij}^{(k)}$. Matrix $M^{(0)}$ is called *multiplication matrix* that we denote it by M. The complexity of the hardware implementation of a normal basis multiplication is related to the number of nonzero entries of the matrix M, that is a crucial parameter for the speed of the system. The Gaussian normal basis representation is a special class of normal basis representation that make multiplication simpler and more efficient [16-17].

For the binary finite field GF(2^m) where m is not divisible by 8, a Gaussian normal basis representation exists. More precisely, for GF(2^m) and a given positive integer T, let $p = mT + 1$ be a prime number and let k be the multiplicative order of 2 module p. If $\gcd(mT/k, m) = 1$, then a GNB of type T exists. The time and area complexity of the multiplication operation in GF(2^m) depends on the type of the normal basis which is related to the number of nonzero entries of the multiplication matrix.
The GNB is considered in several standards such as IEEE P1363 [17] and NIST [18]. For example the binary finite fields GF(2^{163}), GF(2^{233}), GF(2^{283}), GF(2^{409}) and GF(2^{571}) of corresponding types 4,2,6,4 and 10 are recommended by these two standards. In this work, we consider the GNBs of even types with odd values of m which are applicable for cryptographic applications.

2.2. Binary Elliptic Curves

Elliptic curves defined over GF(2^m) are called binary elliptic curves. An ordinary binary elliptic curve is given by the traditional Weierstrass equation

$$E : y^2 + xy = x^3 + ax^2 + b,$$

where $a, b \in \mathrm{GF}(2^m)$ and $b \neq 0$. The set of GF(2^m)-rational points on E including the point at infinity O is denoted by $E\big(\mathrm{GF}(2^m)\big)$, i.e., the set

$$E_{a,b}\big(\mathrm{GF}(2^m)\big) = \{(x, y): x, y \in \mathrm{GF}(2^m),\ y^2 + xy = x^3 + ax^2 + b\} \cup \{O\}.$$

This set of points by the well know addition law [19] forms an Abelian additive group where O is the neutral element. The point addition (PA) formulas carry out the addition of two different points, and the point doubling (PD) formulas output the addition of a point with itself. The addition and doubling formulas are performed in several coordinate systems such as affine, projective, Jacobian and Lopez-Dahab.
The most important operation of elliptic curve cryptography is point multiplication or scalar multiplication. The scalar multiplication includes a sequence of point additions and point doublings. One efficient and popular point multiplication algorithm is the Montgomery ladder point multiplication [20] that is generalized to binary elliptic curves by Lopez-Dahab [21]. Algorithm 1 shows the Montgomery ladder point multiplication that uses differential addition and doubling formulas.

Algorithm1: Montgomery ladder point multiplication
Input: $\boldsymbol{k} = (\boldsymbol{k_{m-1}}, \boldsymbol{k_{m-2}}, \dots, \boldsymbol{k_2}, \boldsymbol{k_1}, \boldsymbol{k_0})_2$ **with** $\boldsymbol{k_{m-1}} = \boldsymbol{1}$.
$\boldsymbol{P} = (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{E}/\mathrm{GF}(2^m)$.
Output: $w(\boldsymbol{kP})$
1. **Initial values**
$\boldsymbol{w_0} \coloneqq w(\boldsymbol{P}), \boldsymbol{W_1} \coloneqq \boldsymbol{w_0},\ \boldsymbol{Z_1} \coloneqq 1$;
$(\boldsymbol{W_2}, \boldsymbol{Z_2}) \coloneqq$ Double$(\boldsymbol{W_1}, \boldsymbol{Z_1})$;
2. **Loop iterations part**
For $\boldsymbol{i} = \boldsymbol{m-2}$ **to** 0 **do**
If $\boldsymbol{k_i} = \boldsymbol{1}$ **then**
$(\boldsymbol{W_1}, \boldsymbol{Z_1}) \coloneqq$**dAdd** $(\boldsymbol{W_1}, \boldsymbol{Z_1}, \boldsymbol{W_2}, \boldsymbol{Z_2}, \boldsymbol{w_0})$;
$(\boldsymbol{W_2}, \boldsymbol{Z_2}) \coloneqq$ **Double** $(\boldsymbol{W_2}, \boldsymbol{Z_2})$;
Else
$(\boldsymbol{W_2}, \boldsymbol{Z_2}) \coloneqq$ **dAdd** $(\boldsymbol{W_1}, \boldsymbol{Z_1}, \boldsymbol{W_2}, \boldsymbol{Z_2}, \boldsymbol{w_0})$;
$(\boldsymbol{W_1}, \boldsymbol{Z_1}) \coloneqq$**Double** $(\boldsymbol{W_1}, \boldsymbol{Z_1})$;
End if;
End for;
Return $w(\boldsymbol{kP}) <= (\boldsymbol{W_1}, \boldsymbol{Z_1})$ **and** $w((\boldsymbol{k}+\boldsymbol{1})\boldsymbol{P}) <= (\boldsymbol{W_2}, \boldsymbol{Z_2})$

Here, we briefly describe the differential coordinate system used in Algorithm 1. Let w be a rational function on the elliptic curve E over GF(2^m). So, for P in $E\big(\mathrm{GF}(2^m)\big)$, $w(P)$ belongs to GF(2^m). Suppose, for each point P in E, we have $w(P) = w(-P)$, where $-P$ is the additive inverse of P. The differential addition formulas compute $w(P+Q)$ for a given $w(P)$ and $w(Q)$ and $w(P-Q)$ and the differential doubling formulas output $w(2P)$ for a given $w(P)$. In Algorithm 1, the differential addition and doubling formulas are performed by the functions "**dAdd**" and "**Double**" respectively. And for a positive integer k and a point P in E, $w(kP)$ is computed by using the Montgomery ladder scalar multiplication. It computes $w(kP)$ in recursive method by computing $w(2iP)$, $w((2i+1)P)$ from $w(iP)$, $w((i+1)P)$ and $w(P)$ using the differential addition and doubling

formulas. To avoid the costly field inversion operation, the projective coordinate system is applied, that is, for an affine point P in E we write $w(P) = \frac{W}{Z}$, where $W, Z \in$ GF(2^m) and we represent the point P as $(W:Z)$. Algorithm 1, for the inputs k and $w_0 = w(P)$ outputs (W_1, Z_1) as the projective representation of $w(kP)$. Furthermore, in the final step of the algorithm (W_2, Z_2) represents $w((k+1)P)$. This helps to recover the point kP.
Notice, for the binary elliptic curve E in Weierstrass equation, for a point $P = (x, y)$ in E, the function w is normally defined by $w(P) = x$. Also, Lopez-Dahab formulas are used in Montgomery ladder.

2.3. Binary Edwards Curves

Elliptic curves are traditionally represented by the Weierstrass equations. Moreover, they are represented by alternative curve shapes to provide further efficiency and speed for cryptographic applications. The binary Edwards curves (BEC) are elliptic curves over binary fields presented in [22]. These curves provided the first complete addition formulas for binary elliptic curves, i.e., the formulas that compute the addition of all pairs of input points, with no exceptional cases. Moreover, the family of complete binary Edwards curves is complete, which means every ordinary elliptic curve can be represented by a complete binary Edwards curve. A binary Edwards curve over GF(2^m) is given by

$$E_{d_1,d_2}: d_1(x+y) + d_2(x^2+y^2) = xy + xy(x+y) + x^2y^2$$

where $d_1, d_2 \in$ GF(2^m) with $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$. The curve has a symmetric equation with a symmetric addition law. The point (0, 0) is the neutral element of the addition law and the point (1, 1) is of order 2. The negative of the point (x, y) is the point (y, x). The addition law on binary Edwards is unified that means the doubling of a point on the curve can be computed by the addition of a point with itself. Furthermore, the addition law on binary Edwards over GF(2^m) is complete if $\mathbf{Tr}(d_2) = 1$. Here **Tr** is denoted the trace function over GF(2^m); where for α in GF(2^m), $\mathbf{Tr}(\alpha) = \sum_{i=0}^{m-1} \alpha^{2^i}$.
The explicit formulas are presented in [22] for affine addition, projective addition, and mixed addition on binary Edwards curves. In comparison with the Weierstrass curves, the formulas are not faster but have the properties of being unified and, for $\mathbf{Tr}(d_2) = 1$, complete. The dedicated doubling formulas are also presented in affine and projective coordinates. The doubling formulas are quite comparatively fast and are also complete for complete binary Edwards curves. The recently presented formulas for mixed point addition in [23] are not complete. In this case we have exception points.
The fast explicit formulas for differential addition on binary Edwards curves are presented in [22]. For the binary Edwards curve E_{d_1,d_2}, for a point $P = (x, y)$ in E_{d_1,d_2}, the function w is defined by $w(P) = x + y$. Notice, $w(-P) = w(P)$, since $-P = (y, x)$. Assume that for the points P_1 and P_2 on E_{d_1,d_2}, $w(P_1)$ and $w(P_2)$ are given as fractions $\frac{W_1}{Z_1}$ and $\frac{W_2}{Z_2}$. Let $P = P_1 - P_2$ with $w_0 = w(P)$, as a field element is given. For the point P_a, the addition of P_1, P_2, the value $w(P_a) = \frac{W_a}{Z_a}$ is computed as follows:

$$C = W_1 \times (Z_1 + W_1), D = W_2 \times (Z_2 + W_2), E = Z_1 \times Z_2, \quad F = W_1 \times W_2,$$
$$V = C \times D, Z_a = V + (e_1 \times E + e_2 \times F)^2, W_a = V + w_0 \times Z_a$$

where, $e_1 = \sqrt{d_1}$ and $e_2 = \sqrt{\frac{d_2}{d_1} + 1}$.
For the point P_d, the doubling of P_1, the value $w(P_d) = \frac{W_d}{Z_d}$ is computed as follows:

$$C = W_1 \times (Z_1 + W_1), W_d = C^2,$$
$$Z_d = W_d + ((e_3 \times Z_1 + e_4 \times W_1)^2)^2$$

where, $e_3 = \sqrt[4]{d_1}$ and $e_4 = \sqrt[4]{\frac{d_2}{d_1} + 1}$.

If $d_1 = d_2$, then the explicit formulas for the point addition are
$C = W_1 \times (Z_1 + W_1), D = W_2 \times (Z_2 + W_2), E = Z_1 \times Z_2, \quad V = C \times D,$
$Z_a = V + d_1 \times E^2, W_a = V + w_0 \times Z_a$

The explicit formulas for point doubling in this case are

$$C = W_1 \times (Z_1 + W_1), W_d = C^2, Z_d = d_1 \times (Z_1^2)^2 + W_d$$

The scalar multiplication kP, for the given positive integer k and the point $P = (x, y)$ in E_{d_1,d_2}, is performed using the Montgomery ladder by Algorithm 1. Note that, in the initial values step of the Algorithm 1, the element $w_0 = w(P)$ equals $x + y$. The Algorithm 1 outputs (W_1, Z_1), (W_2, Z_2) as the projective representations of $w(kP)$ and $w((k + 1)P)$.

2.4. Generalized binary Hessian Curves

The Hessian curve is an alternative symmetric curve shape representing an elliptic curve. The use of Hessian form in cryptography has been studied because of its faster arithmetic compared to that of Weierstrass form. Over a finite field, the family of generalized Hessian curves [24] covers more isomorphism classes of elliptic curves and it is equivalent to the family of all elliptic curves with a point of order 3. These curves provide efficient unified addition formulas which makes them interesting against side-channel attacks. They also have complete addition formulas with suitably chosen parameters. A generalized Hessian curve over a binary finite field GF(2^m) is defined by a symmetric cubic equation as follows:

$$H_{c,d} : x^3 + y^3 + c = dxy.$$

where $c, d \in$ GF(2^m), $c \neq 0$ and $d^3 \neq 27c$. Clearly, this family covers the Hessian elliptic curves where $c = 1$. Notice that the Hessian addition formulas, called the Sylvester formulas, work for the family of generalized Hessian, but these formulas are not unified. A suitable modification of the Sylvester formulas gives fast and efficient unified addition formulas for generalized Hessian curves [24]. The neutral element of the group of points on $H_{c,d}$ is the point at infinity $(1 : 1 : 0)$. For the point $P = (x, y)$ on $H_{c,d}$ the additive inverse is given by $-P = (y, x)$. The explicit formulas are presented [23] for affine addition, projective addition, mixed addition, doubling and tripling for binary generalized Hessian curves. The unified formulas are one of the fastest known addition formulas on binary elliptic curves. Furthermore, the addition formulas are complete for generalized Hessian curves over GF(2^m) when c is not a cube in GF(2^m). Moreover, very competitive differential addition and doubling formulas have been presented [24] for generalized binary Hessian curves, where the mixed differential addition and doubling formulas are also complete. For the binary generalized Hessian curve $H_{c,d}$, the function w is defined on the curve as follow.

For a point $P = (x, y)$ on the curve, $w(P)$ is given by $w(P) = c + dxy$, i.e., $w(P) = x^3 + y^3$. Clearly $w(-P) = w(P)$, since $-P = (y, x)$. The mixed w-coordinate differential addition and doubling on generalized binary Hessian curve are given as follows. Assume that, for the points P_1 and P_2 on $H_{c,d}$, the values $w(P_1)$ and $w(P_2)$ are written as $w(P_1) = \frac{W_1}{Z_1}$ and $w(P_2) = \frac{W_2}{Z_2}$. Let $w_0 = w(P_1 - P_2)$. For the point addition $P_a = P_1 + P_2$ and the point doubling $P_d = 2P_1$, the field elements $w(P_a) = \frac{W_a}{Z_a}$ and $w(P_d) = \frac{W_d}{Z_d}$ are computed as follows:

$$A = W_1 \times Z_2, B = W_2 \times Z_1, C = A \times B, U = h_2 \times C,$$
$$Z_a = (A + B)^2, W_a = U + w_0 \times Z_a$$

For doubling, we have

$$A = W_1^2, B = Z_1^2, C = A + h_1 \times B, D = h_2 \times B, W_d = C^2, Z_d = A \times D$$

where $h_1 = \sqrt{c^3(d^3 + c)}$ and $h_2 = d^3$.

Now, the Algorithm 1 performs the Montgomery ladder computation of the point multiplication kP, where k is a positive integer and $P = (x, y)$ is the input point in $H_{c,d}$. Notice, the initial value of the Algorithm 1 is the element $w_0 = w(P) = c + dxy$ and the outputs are (W_1, Z_1) and (W_2, Z_2) as the projective representation of $w(kP)$ and $w(kP + P)$.

3. Structure for Gaussian normal basis multiplier

The multiplication operation in GF(2^m) has a high complexity structure compared to addition and squaring operations. In recent years, several architectures of the normal basis and Gaussian normal basis (GNB)

multipliers are presented in [25-38]. In this section, the structure of the digit-serial Gaussian normal basis multiplier that presented in [15] is explained.
Let A and B be two elements in GF(2^m) and $B = [b_{m-1}, b_{m-2}, \dots, b_2, b_1, b_0]$. Element B is divided into d words of w bits where $d = \left\lceil \frac{m}{w} \right\rceil$. In other words, we have:

$$B = B_1 + B_2 + B_3 + \cdots + B_w,$$

where, for $i = 1, \dots, w$,

$$B_i = \sum_{k=1}^{d} b_{m-(k-1)w-i}\, \beta^{2^{m-(k-1)w-i}}.$$

Here we let $b_i = 0$ if $i \leq 0$. Multiplication of elements A, B in GF(2^m) is written as:

$$C = AB = A\sum_{i=1}^{w} B_i = \sum_{i=1}^{w}\left(\sum_{k=1}^{d} b_{m-(k-1)w-i}\, A^{2^{-(w-i)}} \beta^{2^{m-kw}}\right)^{2^{w-i}}.$$

or we can rewrite it as following:

$$C = \sum_{i=1}^{w} {C_i}^{2^{w-i}} = \left(\left(\dots\left(\left({C_1}^2 + C_2\right)^2 + C_3\right)^2 + \cdots\right)^2 + C_w\right),$$

where for $i = 1, \dots, w$,

$$C_i = \sum_{k=1}^{d} b_{m-(k-1)w-i}\, A^{2^{-(w-i)}} \beta^{2^{m-kw}}$$

To have a low-complexity and regular architecture of multiplication by $\beta^{2^{(m-kw)}}$ the computation of $y = x\beta^{2^{(m-kw)}}$ is performed in three steps. In the first step the exponentiation of the input x by $2^{-(m-kw)}$ is done. Then in the second step multiplication by β is computed. And finally in the third step the exponentiation of the result by $2^{(m-kw)}$ is performed. These operations result as follows:

$$y = x\beta^{2^{(m-kw)}} = \left(\left(x^{2^{-(m-kw)}}\right)\beta\right)^{2^{(m-kw)}}.$$

Here, the multiplication by β is the main part of the implementation, because the two other steps of exponentiation by $2^{-(m-iw)}$ and $2^{(m-iw)}$ are free hardware, implemented only by cyclic shift. The details of multiplication by β are given in [15].
Considering above three mentioned steps, C_i is rewritten as:

$$C_i = \sum_{k=1}^{d} b_{m-(k-1)w-i}\left(\left(\left(A^{2^{-(w-1)}}\right)^{2^{(i-1)}}\right)^{2^{-(m-kw)}} \beta\right)^{2^{m-kw}}.$$

In the computation of C_i, the exponentiation by $2^{-(m-kw)}$ is performed in the following regular form:

$$\left(\left(A^{2^{-(w-1)}}\right)^{2^{(i-1)}}\right)^{2^{-(m-kw)}} = \left(\dots\left(\left(\left(A^{2^{-(w-1)}}\right)^{2^{(i-1)}}\right)^{2^{-(m-w)}}\right)^{2^{w}} \dots\right)^{2^{w}}.$$

in which, first exponentiation by $2^{-(m-w)}$ is computed, and then for $k = 2,3, \dots, d$, exponentiation by $2^{-(m-kw)}$ are generated sequentially using d-1 exponentiation by 2^w.
Fig.1 shows the structure for the digit-serial GNB multiplier over GF(2^m).

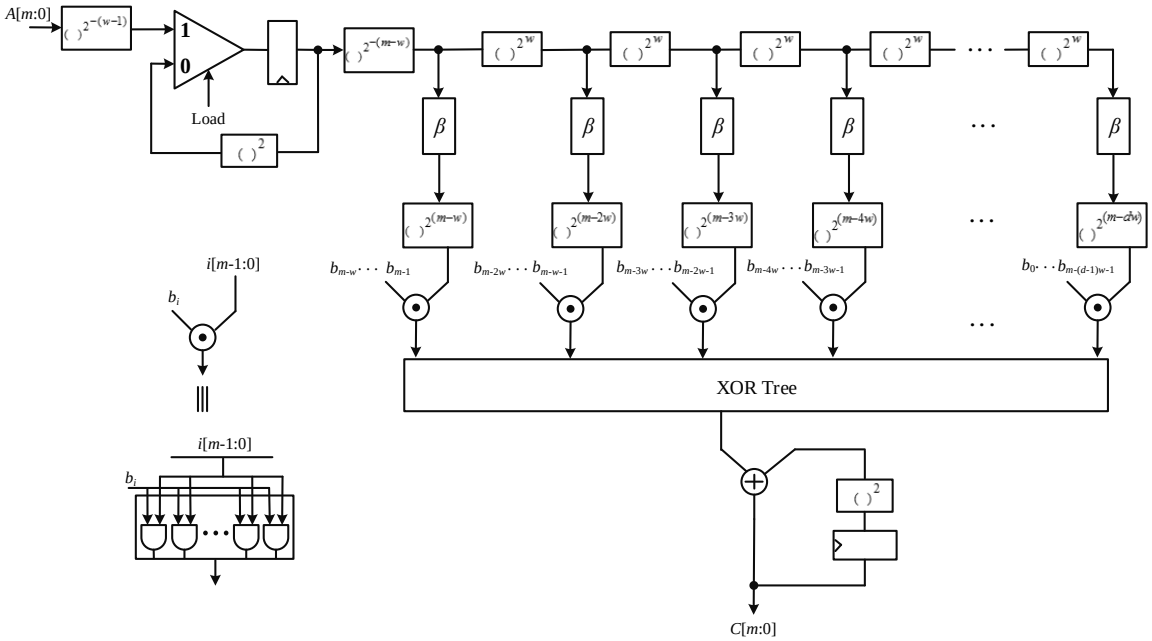

Fig.1. Structure of the digit-serial GNB multiplier over GF(2^m)

As seen in the figure, a regular architecture for hardware implementation is provided. In this structure the blocks of exponentiation by powers of 2 in the normal basis representation are implemented by wired cyclic shift. This property is an important factor for the efficiency of the structure. In the point multiplication architecture, to achieve a lower path delay, the structure of digit-serial GNB is pipelined. To apply this technique, the XOR tree is pipelined by adding two registers behind the last bit-wise XOR operation in the XOR tree.

Fig.2 shows an example of digit-serial GNB multiplier over GF(2^7) with T=4, w=3 and d =3. The GNB multiplication of elements A and B in GF(2^7) with $B = B_1 + B_2 + B_3$, where $B_1 = b_6\beta^{2^6} + b_5\beta^{2^5} + b_4\beta^{2^4}$, $B_2 = b_3\beta^{2^3} + b_2\beta^{2^2} + b_1\beta^2$, and $B_3 = b_0\beta$ is expressed as follows:

$$C = AB = \left(\left(C_1{}^2 + C_2\right)^2 + C_3\right),$$

where,

$$C_1 = \left(\left(A^{2^{-2}}\right)^{2^{-4}}\beta\right)^{2^4} b_6 + \left(\left(\left(A^{2^{-2}}\right)^{2^{-4}}\right)^{2^3}\beta\right)^2 b_3 + \left(\left(\left(\left(A^{2^{-2}}\right)^{2^{-4}}\right)^{2^3}\right)^{2^3}\beta\right)^{2^{-2}} b_0,$$

$$C_2 = \left(\left(\left(A^{2^{-2}}\right)^2\right)^{2^{-4}}\beta\right)^{2^4} b_5 + \left(\left(\left(\left(A^{2^{-2}}\right)^2\right)^{2^{-4}}\right)^{2^3}\beta\right)^2 b_2 + \left(\left(\left(\left(\left(A^{2^{-2}}\right)^2\right)^{2^{-4}}\right)^{2^3}\right)^{2^3}\beta\right)^{2^{-2}} b_{-1},$$

$$C_3 = \left(\left(\left(A^{2^{-2}}\right)^{2^2}\right)^{2^{-4}}\beta\right)^{2^4} b_4 + \left(\left(\left(\left(A^{2^{-2}}\right)^{2^2}\right)^{2^{-4}}\right)^{2^3}\beta\right)^2 b_1 + \left(\left(\left(\left(\left(A^{2^{-2}}\right)^{2^2}\right)^{2^{-4}}\right)^{2^3}\right)^{2^3}\beta\right)^{2^{-2}} b_{-2}.$$

where, in computing C_2, C_3, the bits b_{-1} and b_{-2} are set to zero.

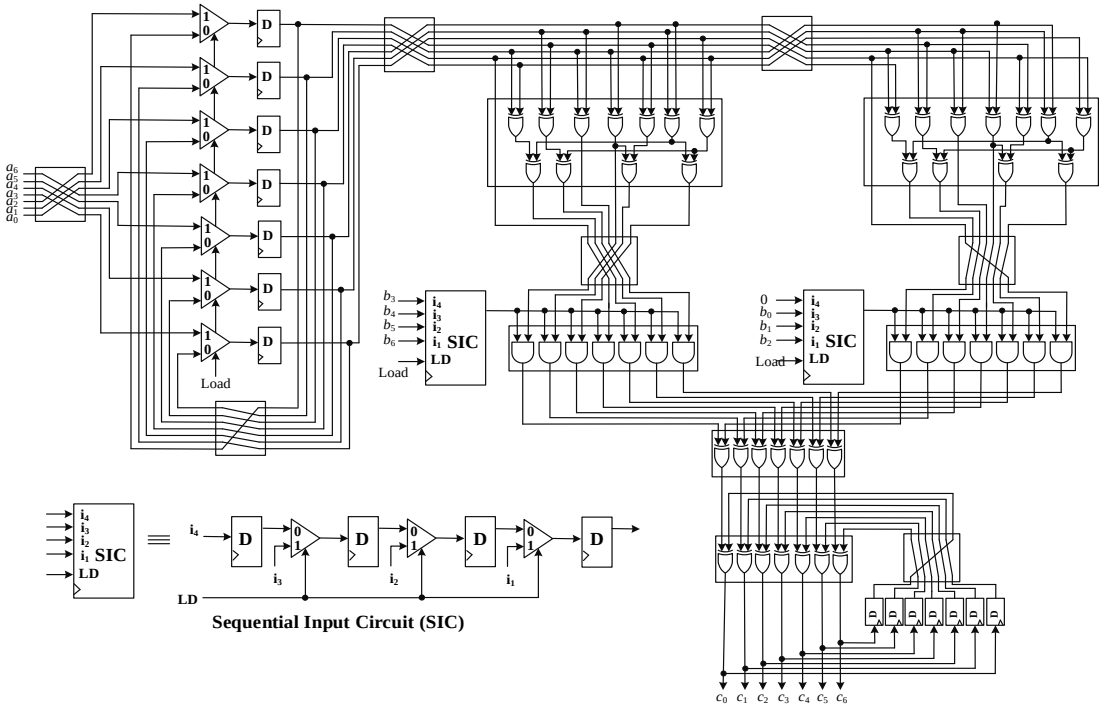

Fig.2. Structure of the digit-serial GNB multiplier over GF(2^7) with w=3 and d=3

The critical path of digit-serial GNB multiplier over GF(2^m) of type T equals T_A+ ($\lceil log_2^T \rceil$+$\lceil log_2^{(d+1)} \rceil$)$T_X$, where T_X and T_A denote the time delays of a 2-input AND gate and a 2-input XOR gate respectively. Also, the multiplier requires dm number of AND gates and at most (dm+(T-1)(m-1)d) XOR gates. In Fig. 2 the critical path delay is $T_A + (\lceil log_2^4 \rceil + \lceil log_2^{(3+1)} \rceil)T_X = T_A + 5T_X$.

4. Proposed Hardware Structure of Point Multiplication on BECs

In the binary Edwards curves, the point addition and point doubling formulas are performed in parallel by using three levels of multiplications. In more details, for example in the case of $d_1 \neq d_2$, computation of PA and PD in mixed w-coordinate requires 10 field multiplications. The parallel PA and PD operations are computed in at least three steps due to the data dependency of the formulas. And, in each step at most four field multiplier units are used. In the first step the four multiplications $C = W_1 \times (Z_1 + W_1)$, $D = W_2 \times (Z_2 + W_2)$, $E = Z_1 \times Z_2$ and $F = W_1 \times W_2$ are computed in parallel by multipliers M_2,M_1, M_3 and M_4 respectively. In the second step $V = C \times D$, $F \times e_2$ and $E \times e_1$ are performed similarly. Finally in the last step $Z_1 \times e_3$, $W_1 \times e_4$ and $w_0 \times Z_a$ are computed by M_1, M_2 and M_4 respectively. Fig.3 (a) and (b) show the proposed scheduling of parallel computation of the PA and PD of binary Edwards curves for two cases of $d_1 \neq d_2$ and $d_1 = d_2$ respectively. In the proposed scheduling the resource allocation is performed properly to reduce the number of clock cycles.

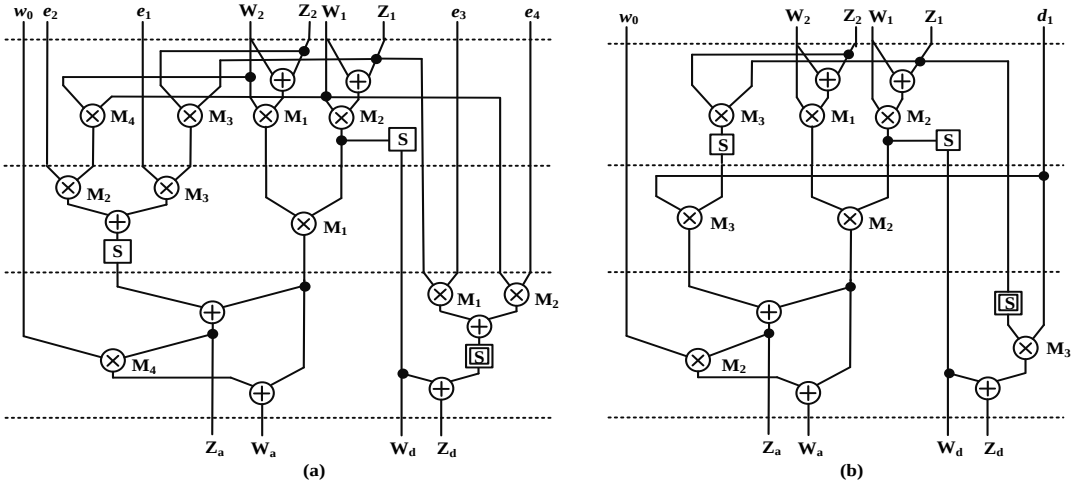

Fig.3. Proposed scheduling of parallel computation of PA and PD on binary Edwards curves for the case $d_1 \neq d_2$ (a), and for the case $d_1 = d_2$ (b)

The proposed architecture for field multiplier has digit-serial or word-level structure. The result of any multiplier is completed at w clock cycles, where w is the number of words of input operands. Moreover, the structure of field multiplier is pipelined to fine the critical path delay, so each multiplication operation is computed in w+1 clock cycles. Thus, the total computation of PA and PD is performed in parallel in three steps by 3(w+1) clock cycles. For example for the case $d_1 \neq d_2$ and w=4 the total computation requires 15 clock cycles as follows.

$$\text{Cycles } 1-5: \begin{cases} C=W_1\times(Z_1+W_1)\overset{by}{\rightarrow}\mathrm{M}_2 \\ D=W_2\times(Z_2+W_2)\overset{by}{\rightarrow}\mathrm{M}_1 \\ E=Z_1\times Z_2\overset{by}{\rightarrow}\mathrm{M}_3 \\ F=W_1\times W_2\overset{by}{\rightarrow}\mathrm{M}_4 \end{cases}, \text{Cycles } 6-10: \begin{cases} V=C\times D\overset{by}{\rightarrow}\mathrm{M}_1 \\ F\times e_2\overset{by}{\rightarrow}\mathrm{M}_2 \\ E\times e_1\overset{by}{\rightarrow}\mathrm{M}_3 \end{cases}, \text{Cycles } 11-15: \begin{cases} Z_1\times e_3\overset{by}{\rightarrow}\mathrm{M}_1 \\ W_1\times e_4\overset{by}{\rightarrow}\mathrm{M}_2 \\ w_0\times Z_a\overset{by}{\rightarrow}\mathrm{M}_4 \end{cases}$$

More details of four field multipliers M_1, M_2, M_3 and M_4 performance for PA and PD computation are shown in Table 1.

Table 1: performance of multipliers M_1, M_2, M_3 and M_4 for the case $d_1 \neq d_2$ and w=4

Cycles	M_1	M_2	M_3	M_4
1	Part 1 of $W_2(Z_2+W_2)$	Part 1 of $W_1(Z_1+W_1)$	Part 1 of Z_1Z_2	Part 1 of W_1W_2
2	Part 2 of $W_2(Z_2+W_2)$	Part 2 of $W_1(Z_1+W_1)$	Part 2 of Z_1Z_2	Part 2 of W_1W_2
3	Part 3 of $W_2(Z_2+W_2)$	Part 3 of $W_1(Z_1+W_1)$	Part 3 of Z_1Z_2	Part 3 of W_1W_2
4	Part 4 of $W_2(Z_2+W_2)$	Part 4 of $W_1(Z_1+W_1)$	Part 4 of Z_1Z_2	Part 4 of W_1W_2
5	Part 5 of $W_2(Z_2+W_2)$	Part 5 of $W_1(Z_1+W_1)$	Part 5 of Z_1Z_2	Part 5 of W_1W_2
6	Part 1 of CD	Part 1 of Fe_2	Part 1 of Ee_1	---
7	Part 2 of CD	Part 2 of Fe_2	Part 2 of Ee_1	---
8	Part 3 of CD	Part 3 of Fe_2	Part 3 of Ee_1	---
9	Part 4 of CD	Part 4 of Fe_2	Part 4 of Ee_1	---
10	Part 5 of CD	Part 5 of Fe_2	Part 5 of Ee_1	---
11	Part 1 of Z_1e_3	Part 1 of W_1e_4	---	Part 1 of w_0Z_a
12	Part 2 of Z_1e_3	Part 2 of W_1e_4	---	Part 2 of w_0Z_a
13	Part 3 of Z_1e_3	Part 3 of W_1e_4	---	Part 3 of w_0Z_a
14	Part 4 of Z_1e_3	Part 4 of W_1e_4	---	Part 4 of w_0Z_a
15	Part 5 of Z_1e_3	Part 5 of W_1e_4	---	Part 5 of w_0Z_a

Computation graphs of the proposed method for the cases $d_1 \neq d_2$ and $d_1 = d_2$ are shown in Fig.4 (a) and Fig.4 (b) respectively. As seen in Fig.4 the multipliers are in parallel and multiplication operations are performed concurrently. In case $d_1 \neq d_2$, the multipliers M_1 and M_2 are busy in all 15 clock cycles during multiplication computations of PA and PD. Also, multipliers M_3 and M_4 are busy for 10 clock cycles. Therefore, the utilization factor of M_1 and M_2 is $\frac{15}{15}\times 100 = 100\%$ and of M_3 and M_4 is $\frac{10}{15}\times 100 = 66.67\%$. Also for the case $d_1 = d_2$ the utilization factors of M_1, M_2 and M_3 are $66.67\%, 66.67\%$ and 100% respectively.

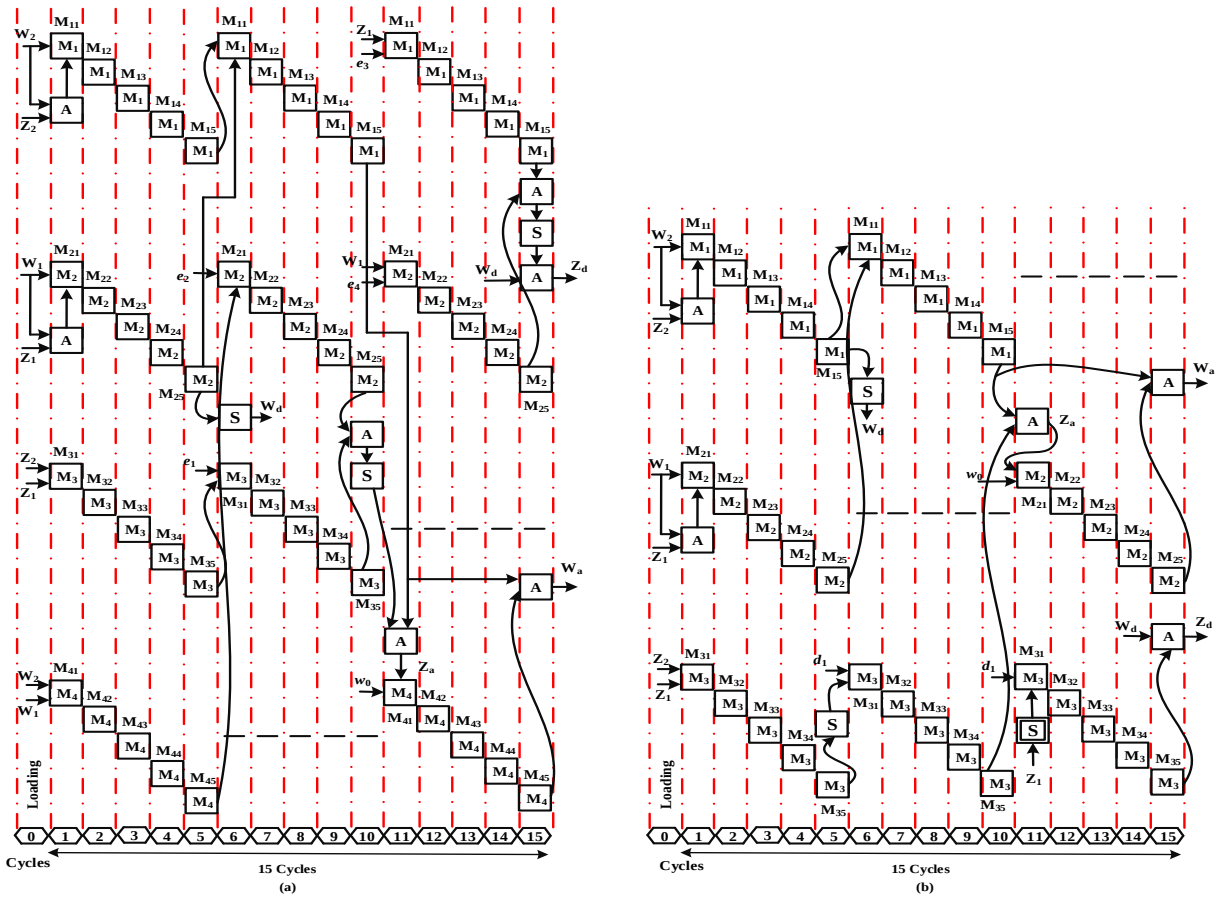

Fig.4. Computation graph of PA and PD in BEC for the cases (a) $d_1 \neq d_2$, and (b) $d_1 = d_2$

The proposed structures of Montgomery ladder point multiplication for BECs cases $d_1 \neq d_2$ and $d_1 = d_2$ are shown in Fig.5 and Fig.6 respectively. In these architectures the Montgomery point multiplication is implemented by using four and three multipliers for $d_1 \neq d_2$ and $d_1 = d_2$ respectively.

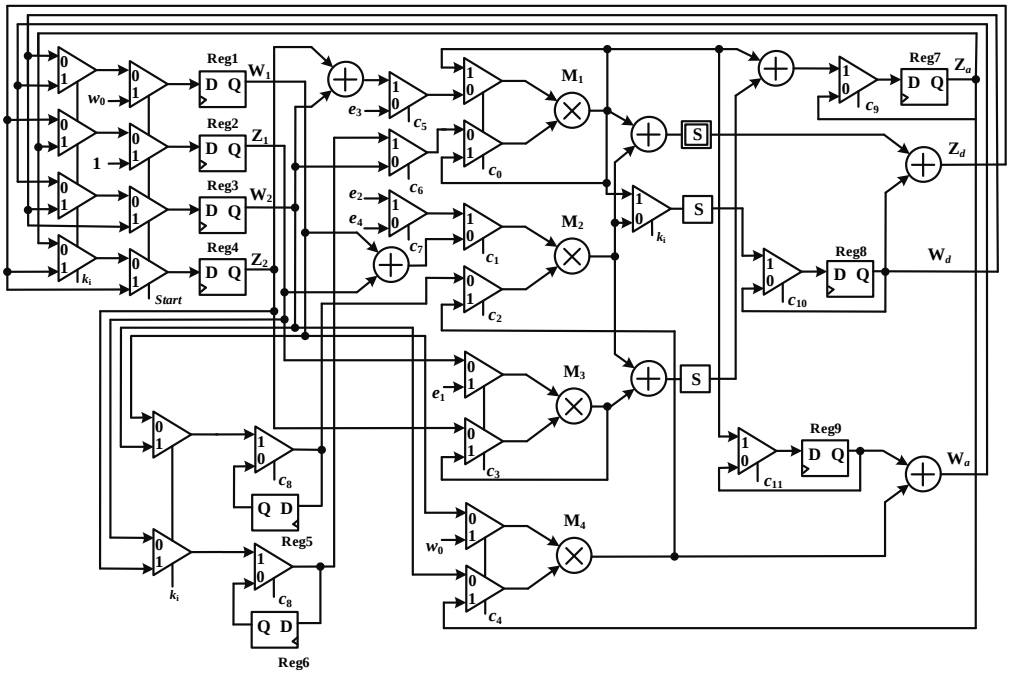

Fig.5. Proposed structure for implementation of the Montgomery ladder point multiplication for BECs case $d_1 \neq d_2$

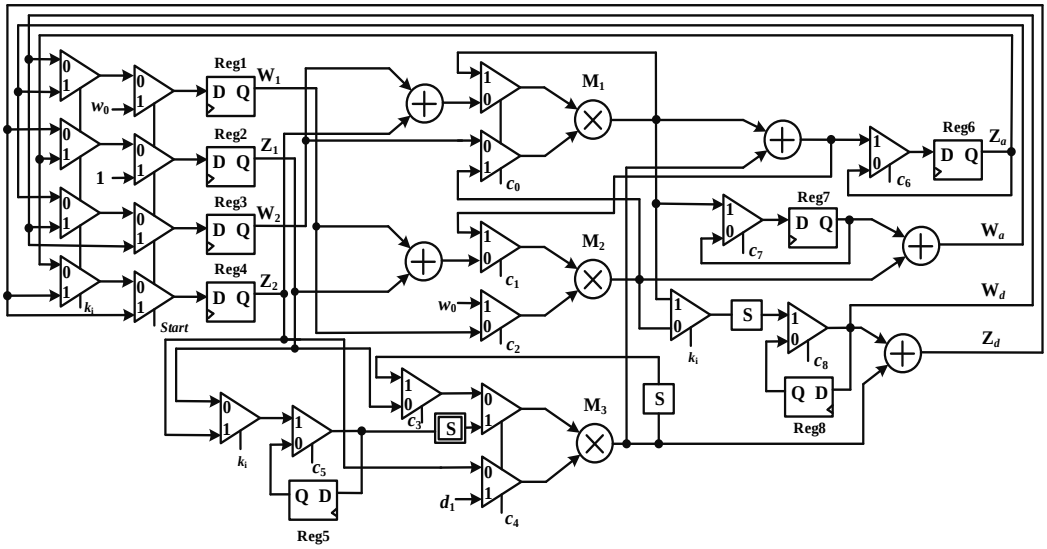

Fig.6. Proposed structure for implementation of the Montgomery ladder point multiplication for BECs case $d_1 = d_2$

As shown in Algorithm 1, in the first step of the algorithm, the initial values of W_1, Z_1 are set by the input point P, and the values W_2 and Z_2 are computed by the point doubling (W_2, Z_2) = **Double**(W_1, Z_1). The point doubling for the binary Edwards curve for the cases $d_1 \neq d_2$ and $d_1 = d_2$ is performed by three and one multiplication respectively. In Fig.5 and Fig.6 the *Start* signal is considered to compute the initial values in step 1 of the algorithm. In the first clock cycle *Start* is set to '1', and W_1=w_0 and Z_1=1 are initialized and stored in Reg1 and Reg2 respectively. Then *Start* is set to '0' and the doubling operation is performed by multipliers M_1 and M_2 to compute W_2 and Z_2. After completion of doubling operations, the coordinates $W_2 = W_d$ and $Z_2 = Z_d$ are ready for loop iteration. At this time, *Start* is set again to '1' and the computed initial values of W_1, Z_1, W_2, and Z_2 are stored in Reg1, Reg2, Reg3 and Reg4 respectively.

In the second step of the algorithm, loop iterations are computed based on bits k_i, the i[th] bit of the scalar number k. The two multiplexers with control signal k_i are to select the input arguments of the point doubling. For $k_i = '1'$ the values W_2 and Z_2 are set as the input arguments of the doubling computation. Otherwise W_1 and Z_1 are selected. Registers Reg5 and Reg6 are used to store the input values of point doubling, which are required in the loop iterations. The multiplexer with control signal k_i at the outputs of M_1 and M_2 is to select of doubling computations. In addition, the four multiplexers with control signal k_i are used to determine the target registers of PD and PA outputs.

In the proposed implementation of Algorithm 1 for BECs the initial values computations take 2(w+1) and (w+1) clock cycles in the cases $d_1 \neq d_2$ and $d_1 = d_2$ respectively. In addition, each loop process takes 3(w+1)+1 clock cycles. In the first cycle, the values W_1, Z_1, W_2, and Z_2 are loaded into the registers Reg1, Reg2, Reg3 and Reg4 respectively. And then, three levels of the multiplication operations are performed in 3(w+1) clock cycles. Therefore the total clock cycles of Algorithm 1 for BEC over GF(2^m) are (m-1)(3(w+1)+1)+2(w+1)+1 and (m-1)(3(w+1)+1)+(w+1)+1 in the cases of $d_1 \neq d_2$ and $d_1 = d_2$ respectively.

5. Proposed Hardware Structure of Point Multiplication on GBHCs

The structure of generalized Hessian curves is designed similarly to that of binary Edwards curves. The point addition and doubling computations of the GBHCs are performed in three levels of multiplication shown in Fig.7. The proposed scheduling is designed properly to have the lowest number of clock cycles due to data dependency of the operation. For example where the number of words w equals 4 the total PA and PD computations are performed in 15 clock cycles. In more details, the output of each pipelined multiplier is ready at 5 clock cycles, so each step of the scheduling is performed at 5 clock cycles as follows:

$$\text{Cycles } 1-5: \begin{cases} A = W_1 \times Z_2 \xrightarrow{by} \mathrm{M}_1 \\ B = W_2 \times Z_1 \xrightarrow{by} \mathrm{M}_2 \end{cases}, \text{Cycles } 6-10: \begin{cases} l = (A+B)^2 \times w_0 \xrightarrow{by} \mathrm{M}_1 \\ C = A \times B \xrightarrow{by} \mathrm{M}_2 \\ U = h_2 \times Z_1^2 \xrightarrow{by} \mathrm{M}_3 \end{cases}, \text{Cycles } 11-15: \begin{cases} h_1 \times Z_1^2 \xrightarrow{by} \mathrm{M}_1 \\ h_2 \times C \xrightarrow{by} \mathrm{M}_2 \\ W_1^2 \times U \xrightarrow{by} \mathrm{M}_3 \end{cases}$$

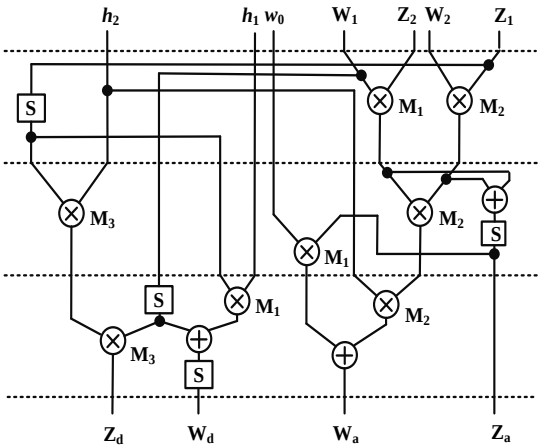

Fig.7. Proposed scheduling of PA and PD computation for GBHCs

Fig.8 shows the computation graph of a loop process of Algorithm 1 for GBHCs. As shown in the figure, in each loop iteration multipliers M_1, M_2 and M_3 operate in parallel. M_1 and M_2 are busy in all 15 clock cycles of PA and PD computation and M_3 is busy for 10 clock cycles. So, the utilization factor of M_1, M_2 and M_3 are 100%, 100% and 66.67% respectively.

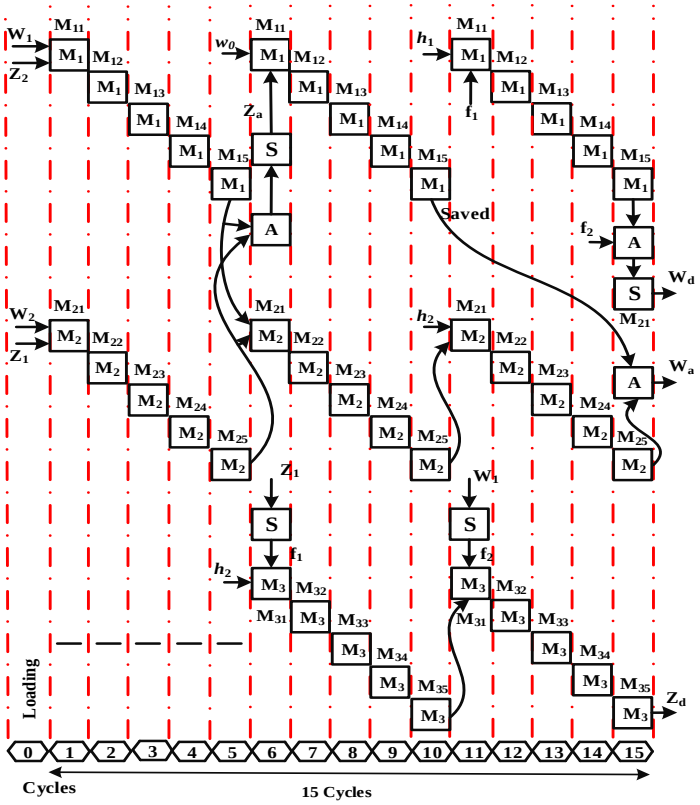

Fig.8. Computation graph of PA and PD for GBHCs

The proposed architecture of Montgomery ladder point multiplication on GBHCs is shown in Fig.9. The values of input coordinates W_i, Z_i, for i=1,2, are fixed during point doubling computation. These input coordinates are stored in registers Reg5 and Reg6. The outputs of PA and PD should be ready simultaneously at the end of each loop iteration. The output Z_a is computed at the end of cycle 5 and stored in Reg7 until other coordinates Z_d, W_d and W_a are calculated for the next loop iteration. For computation of W_a, register Reg8 is used to store the output of M_1 at cycle 10 and to add it to the output of M_2 at cycle 15.

The initial values computation of Algorithm 1 for GBHCs which is implemented by M_1 and M_3 takes $2(w+1)$ clock cycles. Also, each loop process takes $3(w+1)+1$ clock cycles. At first cycle inputs W_1, Z_1, W_2, and Z_2 are loaded in registers Reg1, Reg2, Reg3 and Reg4 respectively. And then the three levels of multiplication

operation take 3(w+1) cycles. Therefore, the total clock cycles of Algorithm 1 for GBHCs over GF(2^m) is (m-1)(3(w+1)+1)+2(w+1)+1.

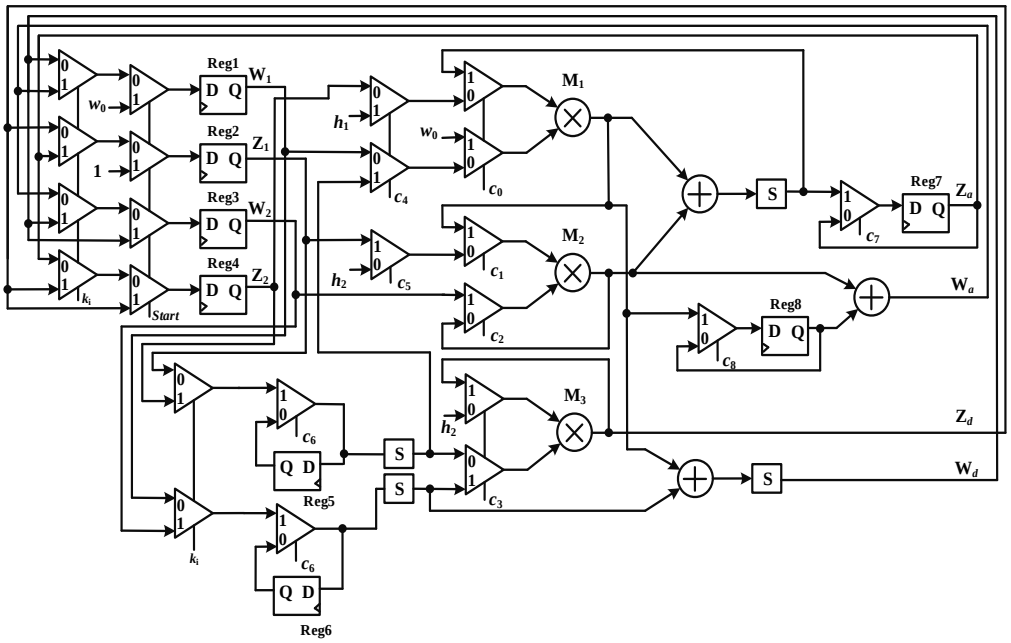

Fig.9: Proposed architecture for implementation of the Montgomery ladder point multiplication for GBHCs

6. Comparison and Results

In this section a comparison between the presented works and other FPGA-based hardware implementations of the point multiplication on binary elliptic curves is presented. The comparison is based on hardware resources, maximum frequency, execution time and efficiency. Parameters of several well-known point multiplication architectures on binary curves are summarized in Table 2. In the table, the efficiency is computed by the following formula:

$$\text{Efficiency} = \frac{Number\ of\ bits}{Computation\ time \times Slices}$$

All the structures are implemented over fields GF(2^{163}) and GF(2^{233}) that are recommended by NIST for ECC applications. The proposed method for BECs ($d_1 \neq d_2$) over GF(2^{233}) has better hardware consumption and timing performance than that of work [12]. For implemented BECs ($d_1 \neq d_2$) over GF(2^{163}) on Virtex-5 FPGA, the computation time and maximum operation frequency of present work are 67% and 25% better than those of work in [13], but hardware resources in [13] are less than present work. However, the proposed work has better efficiency than that of method presented in [13]. Also, there are similar comparison results for BECs ($d_1 = d_2$) and GBHCs over GF(2^{163}) . The proposed method for BECs ($d_1 \neq d_2$) over GF(2^{233}) and GF(2^{163}) implemented on Virtex-4 FPGA have, respectively, 40% and 30% less computation time compared to that of [14]. Moreover, efficiency and hardware resources in the proposed method are improved. The work presented in [14], for GBHCs over GF(2^{163}) and GF(2^{233}) have 24% and 35% computation time more compared to that of this work for the same digit size.

Table 2: Comparison of the proposed structures and other works on binary curves

Works	Field size	Device	Area	Fmax (MHz)	Latency (Cycle)	Time(μs)	Efficiency
[3] BGCs d=41	163	EP2S180F1020C3	18489 ALMs	144.74	---	51.67	171
[4] BGCs d=41	163	Virtex-4 XC4VLX200	19604 Slices	251.054	---	11.92	698
[5] BGCs	163	Virtex-4 XC4VLX200	16209 Slices	153.9	---	19.55	514
[6] BGCs d=55	163	Virtex-4 XC4VLX80	24363 Slices	143	---	10	669
[7] BGCs d=55	163	Virtex-4 XC4VLX200	17929 Slices	250	---	9.6	947
[8] BGCs	163	Virtex-4 XC4VLX80	20807 Slices	185	---	7.72	1015
[9] BGCs	163	Virtex-4 XC4VFX100	3568 Slices	253	---	34	1344
[9] BGCs	283	Virtex-4 XC4VFX100	6128 Slices	157	---	94	491
[10] BGCs	163	Virtex-5 XC5VLX110	7978 Slices	154.35	---	59.15	345
[10] BGCs	233	Virtex-5 XC5VLX110	7978 Slices	154.35	---	84.19	347
[11] BGCs	163	Virtex-4 XC4VLX200	8095 Slices	131	---	10.7	1882
[11] BGCs	163	Virtex-5 XC5VSX240	3513 Slices	147	---	9.5	4884
[12] BECs $(d_1 \neq d_2)$	233	Virtex-4 XC4VLX140	21816 Slices	47.384	9008	190	56
[13] BECs $(d_1 \neq d_2)$ d=33	163	Virtex-5 XC5VLX110	4681 Slices	265.8	7542	28.3	1230
[13] BECs $(d_1 \neq d_2)$ d=41		Virtex-5 XC5VLX110	5788 Slices	264.5	6709	25.3	1113
[13] BECs $(d_1 = d_2)$ d=33		Virtex-5 XC5VLX110	4681 Slices	265.8	5911	22.2	1569
[13] BECs $(d_1 = d_2)$ d=41		Virtex-5 XC5VLX110	5788 Slices	264.5	5243	19.8	1422
[13] GBHCs d=33		Virtex-5 XC5VLX110	4681 Slices	268.2	5415	20.1	1732
[13] GBHCs d=41		Virtex-5 XC5VLX110	5788 Slices	267.1	4747	17.7	1591
[13] BECs $(d_1 \neq d_2)$ d=55		Virtex-4 XC4VLX100	12834 Slices	---	---	22.9	555
[13] BECs $(d_1 = d_2)$ d=55		Virtex-4 XC4VLX100	12834 Slices	---	---	23.3	545
[13] BHCs (c=1) d=55		Virtex-4 XC4VLX100	12834 Slices	---	---	20.8	610
[14] BECs $(d_1 \neq d_2)$ d=33	163	Virtex-4 XC4VLX160	27778 Slices	217.2	3808	17.5	335
[14] BECs $(d_1 \neq d_2)$ d=26	233	Virtex-4 XC4VLX160	29252 Slices	198.4	7212	36.3	219
[14] GBHCs d=33	163	Virtex-4 XC4VLX160	15992 Slices	218.2	3471	15.9	641
[14] GBHCs d=26	233	Virtex-4 XC4VLX160	16940 Slices	205.1	6791	33.1	416
Proposed BECs $(d_1 \neq d_2)$ d=33	**163**	**Virtex-4 XC4VLX100**	**22957 Slices**	**253.873**	**3091**	**12.18**	**583**
Proposed BECs $(d_1 \neq d_2)$ d=41		**Virtex-4 XC4VLX100**	**27365 Slices**	**247.396**	**2603**	**10.52**	**566**
Proposed BECs $(d_1 = d_2)$ d=33		**Virtex-4 XC4VLX100**	**17125 Slices**	**254.996**	**3085**	**12.1**	**787**
Proposed BECs $(d_1 = d_2)$ d=41		**Virtex-4 XC4VLX100**	**20853 Slices**	**247.750**	**2598**	**10.49**	**745**
Proposed GBHCs d=33		**Virtex-4 XC4VLX100**	**17052 Slices**	**254.808**	**3091**	**12.13**	**788**
Proposed GBHCs d=41		**Virtex-4 XC4VLX100**	**20752 Slices**	**247.037**	**2603**	**10.54**	**745**
Proposed BECs $(d_1 \neq d_2)$ d=33		**Virtex-5 XC5VLX110**	**9624 Slices**	**331.363**	**3091**	**9.33**	**1815**
Proposed BECs $(d_1 \neq d_2)$ d=41		**Virtex-5 XC5VLX110**	**11397 Slices**	**302.081**	**2603**	**8.62**	**1659**
Proposed BECs $(d_1 = d_2)$ d=33		**Virtex-5 XC5VLX110**	**7314 Slices**	**331.363**	**3085**	**9.31**	**2394**
Proposed BECs $(d_1 = d_2)$ d=41		**Virtex-5 XC5VLX110**	**8645 Slices**	**302.093**	**2598**	**8.6**	**2192**
Proposed GBHCs d=33		**Virtex-5 XC5VLX110**	**7313 Slices**	**331.363**	**3091**	**9.33**	**2389**
Proposed GBHCs d=41		**Virtex-5 XC5VLX110**	**8645 Slices**	**302.093**	**2603**	**8.62**	**2187**
Proposed BECs $(d_1 \neq d_2)$ d=26	**233**	**Virtex-4 XC4VLX100**	**18278 Slices**	**333.970**	**7213**	**21.6**	**590**
Proposed BECs $(d_1 \neq d_2)$ d=59		**Virtex-4 XC4VLX100**	**37053 Slices**	**277.691**	**3723**	**13.41**	**467**
Proposed BECs $(d_1 = d_2)$ d=26		**Virtex-4 XC4VLX100**	**13786 Slices**	**333.970**	**7203**	**21.57**	**784**
Proposed BECs $(d_1 = d_2)$ d=59		**Virtex-4 XC4VLX100**	**31702 Slices**	**277.681**	**3718**	**13.39**	**549**
Proposed GBHCs d=26		**Virtex-4 XC4VLX100**	**14052 Slices**	**333.970**	**7213**	**21.6**	**768**
Proposed GBHCs d=59		**Virtex-4 XC4VLX100**	**27933 Slices**	**277.681**	**3723**	**13.41**	**622**
Proposed BECs $(d_1 \neq d_2)$ d=26		**Virtex-5 XC5VLX110**	**6547 Slices**	**391.932**	**7213**	**18.40**	**1934**
Proposed BECs $(d_1 \neq d_2)$ d=59		**Virtex-5 XC5VLX110**	**14343 Slices**	**337.603**	**3723**	**11.03**	**1473**
Proposed BECs $(d_1 = d_2)$ d=26		**Virtex-5 XC5VLX110**	**4987 Slices**	**391.932**	**7203**	**18.38**	**2542**
Proposed BECs $(d_1 = d_2)$ d=59		**Virtex-5 XC5VLX110**	**11494 Slices**	**337.603**	**3718**	**11.01**	**1841**
Proposed GBHCs d=26		**Virtex-5 XC5VLX110**	**5045 Slices**	**391.932**	**7213**	**18.40**	**2510**
Proposed GBHCs d=59		**Virtex-5 XC5VLX110**	**8875 Slices**	**337.603**	**3723**	**11.03**	**2380**

7. Conclusions

In this paper, hardware architectures of high-speed Montgomery ladder point multiplication for binary Edwards and generalized Hessian curves over GF(2^m) are presented. To reduce the number of clock cycles, the proposed structures are designed based on concurrent computation of the point addition and point doubling by using parallel digit-serial Gaussian normal basis multipliers. The multipliers have highly regular structures with low hardware resources and low critical path delays. The results show an overall improvement in terms of execution time, hardware resources, and efficiency in comparison with previously reported works.

References

[1] Koblitz, N. (1987) Elliptic curve cryptosystems. in Mathematics of Computation, 203-209.
[2] Miller , V. S. (1986) Use of elliptic curve in cryptography. in Advances in Cryptology, Crypto'85 Proceedings, 417-426.
[3] Järvinen, K., and Skytta, J. (2008) On Parallelization of High-Speed Processors for Elliptic Curve Cryptography. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 16, No. 9, 1162-1175.
[4] Masoumi, M., Mahdizadeh, H. (2012) Efficient Hardware Implementation of an Elliptic Curve Cryptographic Processor over GF (2^{163}). World Academy of Science, Engineering and Technology 65, 1223-1230.
[5] Chelton, W. N. and Benaissa, M. (2008) Fast elliptic curve cryptography on FPGA. IEEE Transactions on Very Large Scale Integration (VLSI) Systems,Vol. 16, No. 2, 198-205.
[6] Choi, H. M., Hong, C. P., Kim, C. H. (2008) High Performance Elliptic Curve Cryptographic Processor Over GF(2^{163}). 4th IEEE International Symposium on Electronic Design, Test & Application, pp. 290-295.
[7] Mahdizadeh, H. and Masoumi, M. (2013) Novel Architecture for Efficient FPGA Implementation of Elliptic Curve Cryptographic Processor Over GF(2^{163}). IEEE Trans. on VLSI Systems, Vol. 21, Iss. 12, 2330-2333.
[8] Zhang, Y., Chen, D., Choi, Y., Chen, L., and Ko, S.-B. (2010) A high performance ECC hardware implementation with instruction-level parallelism over GF (2^{163}). Microprocess. Microsyst., Vol. 34, No. 6, 228-236.
[9] Fayed, M. A., Watheq, El-Kharashi, M., Gebali, F. (2007) A High-Speed, High-Radix, Processor Array Architecture for Real-Time Elliptic Curve Cryptography over GF(2^m). IEEE International Symposium on Signal Processing and Information Technology, 56-61.
[10] Cinnati Loi, K. C., Sen A., and Ko, S.B. (2014) FPGA Implementation of Low Latency Scalable Elliptic Curve Cryptosystem Processor in GF(2^m). IEEE international Symposium on Circuits and Systems (ISCAS), 822-825.
[11] Roy, S.S., Rebeiro, C., and Mukhopadhyay, D. (2013) Theoretical Modeling of Elliptic Curve Scalar Multiplier on LUT-Based FPGAs for Area and Speed. IEEE Trans. on VLSI Systems, Vol. 21, No. 5, 901-909.
[12] Chatterjee, A., Sengupta I. (2012) Design of a high performance Binary Edwards Curve based processor secured against side channel analysis", Integration, the VLSI Journal, Vol. 45, No. 3, 331-340.
[13] Azarderakhsh, R. and Reyhani-Masoleh, A. (2012) Efficient FPGA Implementations of Point Multiplication on Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis. IEEE Trans. on VLSI Systems, Vol. 20, No. 8, 1453-1466.
[14] Azarderakhsh, R. and Reyhani-Masoleh, A. (2015) Parallel and High-Speed Computations of Elliptic Curve Cryptography Using Hybrid-Double Multipliers. IEEE Trans. on VLSI Systems, Vol. 26, Iss. 6, 1668-1677.
[15] Rashidi, B., Sayedi, S.M., Rezaeian Farashahi, R. (2016) Efficient and Low-complexity Hardware Architecture of Gaussian Normal Basis Multiplication over GF(2^m) for Elliptic Curve Cryptosystems. IET Circuits Devices Syst., Vol. 10, 1-10.
[16] Ash, D.W., Blake, I.F., and Vanstone, S.A. (1989) Low Complexity Normal Bases. Discrete Applied Math., 25, 191-210.
[17] IEEE P1363: Editorial Contribution to standard for Public Key Cryptography, 2003.
[18] Federal Information Processing Standards Publications (FIPS) 186-2, U.S. Department of Commerce/NIST: Digital Signature Standard (DSS), 2000.
[19] Hankerson, D., Menezes, A., Vanstone, S. (2004) Guide to Elliptic Curve Cryptography. Springer-Verlag, New York, Edition 1.
[20] Montgomery, P.L. (1987) Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48, 243-264.
[21] Lopez, J. Dahab, R. (1999) Fast multiplication on elliptic curves over GF(2^m) without precomputation. in Proc. of First International Workshop Cryptographic Hardware and Embedded Systems (CHES'99), (Springer-Verlag), 316-327.
[22] Bernstein, D. Lange, T. and Rezaeian Farashahi, R. (2008) Binary Edwards curves. in Proc. Workshop Cryptograph. Hardware Embedded Syst., Vol. 5154, 244–265.
[23] Kim, K., Lee, C., Negre, C. (2014) Binary edwards curves revisited. INDOCRYPT 2014. LNCS, vol. 8885, 393-408.
[24] Rezaeian Farashahi, R. and Joye, M. (2010) Efficient arithmetic on Hessian curves. in Proc. 13th Int. Conf. Practice Theory of Public Key Cryptography, 243–260.
[25] suk cho, Y., Yeon Choi, J. (2013) A new Word-parallel bit-serial Normal basis multiplier over GF(2^m). International Journal of control and Automation, Vol. 6, No. 3, 209-216.
[26] Reyhani-Masoleh, A. (2006) Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases. IEEE Trans. Computers, Vol. 55, Iss. 1, 34-47.
[27] Chiou, C.W., Chang, H.W., Liang, W.-Y., Lee, C.-Y., Lin,, J.-M., Yeh,Y.-C. (2012) Low-complexity Gaussian normal basis multiplier over GF(2^m). IET Inf. Secur., Vol. 6, Iss. 4, 310-317.
[28] Lee, C.-Y., Wun Chiou, C. (2012) Scalable Gaussian Normal Basis Multipliers over GF(2^m) Using Hankel Matrix-Vector Representation. J Sign Process Syst. Vol 69, Iss. 2, 197-211.
[29] Azarderakhsh, R. and Reyhani-Masoleh, A. (2010) A Modified Low Complexity Digit-Level Gaussian Normal Basis Multiplier. Proc. Third Int'l Workshop Arithmetic of Finite Fields (WAIFI), 25-40.
[30] Azarderakhsh, R. and Reyhani-Masoleh, A. (2013) Low-Complexity Multiplier Architectures for Single and Hybrid-Double Multiplications in Gaussian Normal Bases. IEEE Trans. Comput., Vol. 62, Iss. 4, 744-757.
[31] Chuang, T.-P., Wun Chiou, C., Lin, S.-S., Lee, C.-Y. (2012) Fault-tolerant Gaussian normal basis multiplier over GF(2^m). IET Inf. Secur., Vol. 6, Iss. 3, 157-170.
[32] Lee, C.-Y. (2010) Concurrent error detection architectures for Gaussian normal basis multiplication over GF(2^m). Integration, the VLSI journal, Vol. 43, Iss. 1, 113-123.
[33] Wang, Z., Wang, X., and Fan, S. (2010) Concurrent Error Detection Architectures for Field Multiplication Using Gaussian Normal Basis. Proc. of Information Security, Practice and Experience (ISPEC), LNCS 6047, 96-109.
[34] Wun Chiou, C. Lin, J.M., Li, Y.K., Lee, C.-Y., Chuang, T.P., and Yeh, Y.C. (2014) Pipeline Design of Bit-Parallel Gaussian Normal Basis Multiplier over GF(2^m). Advances in Intelligent Systems and Computing, Springer, Vol. 238, 369-377.
[35] Chiou, C. W., Chang, C. C., Lee, C. Y., Lin, J. M., & Hou, T. W. (2009) Concurrent error detection and correction in Gaussian normal basis multiplier over GF(2^m)", IEEE Trans. Comput., Vol. 58, Iss. 6, 851-857.
[36] Lee, C. and Chang, P. (2009) Digit-Serial Gaussian Normal Basis Multiplier over GF(2^m) Using Toeplitz Matrix-Approach. Proc. Int'l Conf. Computational Intelligence and Software Eng. (CiSE), 1-4.
[37] Azarderakhsh, R., Mozaffari Kermani, M., Bayat-Sarmadi, S, Lee, C.-Y. (2014) Systolic Gaussian Normal Basis Multiplier Architectures Suitable for High-Performance Applications. IEEE Trans. on VLSI Systems, Vol. 99, 1-4.
[38] Wang, Z., Fan, S. (2012) Efficient montgomery-based semi-systolic multiplier for even-type GNB of GF(2^m). IEEE Trans. Comput., Vol. 61, Iss. 3, 415-419.