Efficient FPGA Implementations of Point Multiplication on Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis

Reza Azarderakhsh, *Student Member, IEEE*, and Arash Reyhani-Masoleh, *Member, IEEE*

***Abstract*—Efficient implementation of point multiplication is crucial for elliptic curve cryptographic systems. This paper presents the implementation results of an elliptic curve crypto-processor over binary fields $GF(2^m)$ on binary Edwards and generalized Hessian curves using Gaussian normal basis (GNB). We demonstrate how parallelization in higher levels can be performed by full resource utilization of computing point addition and point-doubling formulas for both binary Edwards and generalized Hessian curves. Then, we employ the w-coordinate differential formulations for computing point multiplication. Using a lookup-table (LUT)-based pipelined and efficient digit-level GNB multiplier, we evaluate the LUT complexity and time–area tradeoffs of the proposed crypto-processor on an FPGA. We also compare the implementation results of point multiplication on these curves with the ones on the traditional binary generic curve. To the best of the authors' knowledge, this is the first FPGA implementation of point multiplication on binary Edwards and generalized Hessian curves represented by w-coordinates.**

***Index Terms*—Binary Edwards curves (BECs), elliptic curve cryptography (ECC), Gaussian normal basis (GNB), generalized Hessian curves (GHCs).**

I. Introduction

THE use of elliptic curve cryptography (ECC) has been identified as an efficient and suitable methodology to achieve public key cryptography in embedded and resource-constrained environments [1]. Miller [2] and Koblitz [3] independently showed that a group of points on generic (Weierstrass form) curves over finite fields can be used for elliptic curve cryptosystems. The main advantage of ECC is that it offers a similar security level compared with the other traditional cryptosystems, employing smaller key size. The security of ECC-based cryptosystems relies on the difficulty of solving elliptic curve discrete logarithm problem (ECDLP) [1]. To date, several forms of elliptic curves over finite fields of characteristic two have been considered for hardware implementation of such cryptosystems in the literature; see, for example, [4]–[13]. They cover a wide variety of cases regarding different basis representations (e.g., polynomial basis and normal basis), different coordinate systems (e.g., affine, projective, or mixed), and different curve forms (e.g., generic and Koblitz). In these implementations, various hardware platforms such as field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) have been utilized. For different target applications, efficient implementations of ECC on these platforms with a balance between complexity of computations and availability of the resources are crucial to provide highly efficient cryptographic systems.

Binary Edwards curves (BECs) have been introduced recently by Bernstein *et al.* in [14]. They showed that all generic elliptic curves over binary fields can be written in Edwards form to obtain efficient complete and unified addition formulas which work for all pairs of inputs. In [15], a generalized form of binary Hessian curves is proposed which has similar characteristics to the BECs. Both of these curves offer unified and complete formulas for point operations which provides resistance against side-channel attacks (SCAs). Despite the efficiency of binary Edwards and generalized Hessian curves, a limited number of articles in the literature such as [16]–[18] have investigated their implementations. In [16], an ASIC implementation of point multiplication on a special case of BECs has been presented addressing energy consumption and simple power analysis attacks over $GF(2^m)$ using polynomial basis representation. An SCA resistance evaluation of BECs has been discussed in [17] employing unified addition formula for doubling. The work presented in [18] mainly focuses on software implementation of point multiplication on these curves employing different curve parameters.

In this paper, we present efficient FPGA implementations of point multiplication over newly proposed binary Edwards [14] and generalized Hessian curves [15] as well as traditional binary generic curves using a pipelined low-complexity and fast Gaussian normal basis multiplier. For theoretical aspects of all the arithmetic operations over these curves, the reader is referred to [14], [15], and [19]. We have shown how combining algorithmic techniques (such as parallelization) with platform-dependent strategies (lookup-table (LUT)-based analysis) can be used to develop an efficient FPGA implementation of ECC crypto-processor. Moreover, we have examined completeness of the mixed differential addition and doubling formulations for every pair of points for binary Edwards and generalized Hessian curves.

Similar to binary generic elliptic curves, the performance of ECC computations on binary Edwards and generalized Hessian curves is determined by an operation called point multiplication.

Manuscript received November 30, 2010; revised April 05, 2011; accepted May 10, 2011. Date of publication June 27, 2011; date of current version June 14, 2012. The work of A. Reyhani-Masoleh was supported in part by the Natural Sciences and Engineering Research Council (NSERC) through a Discovery Grant.

The authors are with the Department of Electrical and Computer Engineering, The University of Western Ontario, London, ON, Canada, N6A 5B9 (e-mail: razarder@uwo.ca; areyhani@uwo.ca).

Digital Object Identifier 10.1109/TVLSI.2011.2158595

The efficiency of point multiplication depends mainly on the lower level field arithmetic operations.

At arithmetic level, multiplication of two field elements in the binary field of characteristic two, i.e., $GF(2^m)$, is more complicated than the other operations (e.g., addition and squaring) and determines the performance of an ECC crypto-processor. Several architectures for finite field multipliers are designed and analyzed with different field representations mainly using polynomial basis [5], [20], [21] and normal basis [22], [23]. Implementing field multipliers using polynomial basis is more efficient in software than normal basis [24]. In hardware implementations, both bases have efficient results and normal basis is a suitable choice in applications with frequent squarings. Massey and Omura (MO) [25] invented a bit-level, parallel-in serial-out $GF(2^m)$ normal basis multiplier. Such a bit-level multiplier is slow as it generates the results of multiplication after m clock cycles. The fastest type of multipliers is the bit-parallel one whose results are available after the propagation delay through the gates in one clock cycle. We note that, for type-2 GNB (which is type-2 optimal normal basis), there are several efficient multipliers available in the literature. For instance, in [26], Sunar and Koç proposed a bit-parallel multiplier based on a permuted normal basis. An efficient and systolic type of their multiplier has been proposed later by Kwon [27] for type-2 GNB which is highly regular. Also, subquadratic style multipliers have been proposed in [28]–[30], which require smaller area but higher delays.

A digit-level multiplier is in between (in terms of both space and time complexities) the bit-serial and bit-parallel multipliers. A digit-level version of the MO multiplier [25] is investigated for FPGA implementation of ECC in [6]. Also, Kwon *et al.* [22] proposed an improved digit-level GNB multiplier which has been employed in [7] for FPGA implementation of ECC over $GF(2^{163})$. In order to satisfy high-speed and low-complexity requirements of an ECC crypto-processor, one needs to design an efficient architecture for finite field multiplication using normal basis [6]. Two digit-level GNB multipliers are proposed in [23], one of which is modified in [31] by introducing a subexpression sharing and complexity reduction algorithm for type $T > 2$ Gaussian normal bases. In this paper, we have used a pipelined version of this multiplier for efficient FPGA implementations of point multiplication on binary Edwards, generalized Hessian, and binary generic curves.

The main contributions of this paper can be summarized here.

- We propose an efficient hardware architecture for point multiplication on binary Edwards and generalized Hessian curves incorporating higher level parallelization and optimum lower level scheduling. This increases the overall performance considering maximum utilization of available resources.
- We incorporate w-coordinate version of Montgomery's ladder for point multiplication in binary Edwards and generalized Hessian curves using mixed differential representation.
- For the proposed crypto-processor architecture over $GF(2^m)$, we obtain the optimum digit sizes in terms of time–area tradeoffs for the proposed fast and low-complexity digit-level GNB multiplier.
- Finally, we perform efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves over $GF(2^{163})$ on a Xilinx Virtex-5 device and investigate the LUT-based time–area efficiency for different digit sizes. We have also implemented ECC on binary generic curve and compared its FPGA implementation results with the ones obtained for binary Edwards and generalized Hessian curves.

The remainder of this paper is organized as follows. In Section II, preliminaries of Gaussian normal basis representation and arithmetics on the binary Edwards and generalized Hessian curves are presented. In Section III, point multiplication and parallelization of PA and PD are explained. The proposed hardware architecture for elliptic curve crypto-processor is presented in Section IV. Here, a modified and pipelined digit-level GNB multiplier is also presented and analyzed in terms of time–area tradeoffs for different digit sizes. Section V presents the results of FPGA implementations for the proposed ECC crypto-processor. Finally, we conclude the paper in Section VI.

II. Preliminaries

A. Gaussian Normal Basis

It is well known that, for any positive integer $m \geq 1$, there exists a normal basis of $GF(2^m)$ over $GF(2)$[32]. Let $N = \left\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\right\}$ be a normal basis of $GF(2^m)$ for $\beta \in GF(2^m)$. Then, β is called a normal element of $GF(2^m)$ such that the set is the normal basis of $GF(2^m)$. Therefore, the representation of any element in N, say, $A = (a_0, a_1, \ldots, a_{m-1})$, is $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where coefficient $a_i \in GF(2)$ [33]. In normal basis, squaring can be achieved by simple right cyclic shift of A, i.e., $A^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = (a_{m-1}, a_0, a_1, \ldots, a_{m-2})$[1]. Note that this operation is fast without any cost if it is implemented in hardware. GNB is a special class of normal basis which is included in the IEEE 1363 [34] and NIST [1] standards for elliptic curve digital signature algorithm (ECDSA) and exists for every $m > 1$ that is not divisible by eight [32]. The complexities of type-T GNB multipliers in terms of time and area depend on $T > 1$. For the five binary fields recommended by NIST, i.e., $m = 163$, 233, 283, 409, and 571, the values of T are even and are 4, 2, 6, 4, and 10, respectively.

The use of type-4 GNB to obtain high-performance elliptic curve cryptosystem has been received much attention in [6], [7], and [9]. Also, in [35], type-2 normal basis is incorporated as part of global distributed effort to solve ECDLP over $GF(2^{131})$.

B. Arithmetic Over Binary Edwards and Generalized Hessian Curves

It is well known that a nonsupersingular binary generic (short Weierstraß) elliptic curve can be defined by a set of points (x, y) and a special point at infinity $\mathcal{O}$ (group identity) that satisfy the following equation:

$$E_{a,b}/GF(2^m) : y^2 + xy = x^3 + ax^2 + b \tag{1}$$

where $a, b \in GF(2^m)$ and $b \neq 0$[36]. These curves are also called anomalous binary curves or Koblitz curves if $a \in \{0,1\}$ and $b = 1$, i.e., defined over $GF(2)$[3].

BECs belong to a special class of generic elliptic curves defined over the binary field when $m \geq 3$ [14]. The merit of BECs over generic curves is that their PA formulas are complete and their implementations are comparable with the traditional ones of [19].

Definition 1: [14] Let $\mathbb{K}$ be a finite field of characteristic two, i.e., $\mathrm{char}(\mathbb{K}) = 2$ and d_1 and d_2 be the elements of $\mathbb{K}$ with $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$. The BECs with coefficients d_1 and d_2 is the affine curve

$$E_{B,d_1,d_2}/GF(2^m): \quad d_1(x+y) + d_2(x^2+y^2) = xy + xy(x+y) + x^2y^2 \quad (2)$$

where $d_1, d_2 \in GF(2^m)$. Given a point $P = (x, y)$, its negation $-P$ is obtained as (y, x), which has no cost [14]. The point $(0, 0)$ is the neutral element and $(1, 1)$ has order 2 [14]. The BECs are complete if $\mathrm{Tr}(d_2) = 1$, i.e., d_2 cannot be written as $e^2 + e$ for any e in $\mathbb{K}$, where Tr is the absolute trace of $GF(2^m)$ over $GF(2)$ [14].

Definition 2: [15] Let c and d to be elements of $\mathbb{K}$ such that $c \neq 0$ and $d^3 \neq 27c$. The generalized Hessian curve (GHC) $H_{c,d}$ over $\mathbb{K}$ is defined by the equation

$$H_{c,d}/GF(2^m): x^3 + y^3 + c = dxy \quad (3)$$

where $c = 1$ results in a Hessian curve, i.e., H_d. Note that the GHCs are complete if and only if c is not a cube in $\mathbb{K}$.

The standard formulas on generic curves [19] fail in computing the addition of two points on curves if one of the points or their addition is at infinity. These possibilities should be tested before designing an elliptic curve cryptosystem. Note that point addition and doubling formulas on binary Edwards and generalized Hessian curves work for all input pairs. This characteristic is called completeness. In what follows, we discuss the point addition and doubling using w-coordinates for binary Edwards and generalized Hessian curves.

C. Point Addition and Doubling Using Differential Formulations in w-Coordinates

Differential addition [37] is the computation of $Q + P$, given points of Q, P, and $Q - P$. In [14] and [15], the idea of Montgomery's ladder [37] is used to present fast formulas for w-coordinate differential addition on binary Edwards and generalized Hessian curves, respectively. Let us assume w to be a linear and symmetric function in terms of the coordinates x and y of the point P and is defined as $w_i = x_i + y_i$, where $w(P) = w(-P)$. Bernstein *et al.* [14] have defined w-coordinate differential addition for computing $w(Q+P)$ given $w(Q)$, $w(P)$, and $w(Q-P)$. Similarly, the w-coordinates differential doubling is the computation of $w(2P)$ given $w(P)$. Therefore, using w-coordinates of differential addition and doubling formulas, $w((2n+1)P)$ and $w(2nP)$ can be computed given $w(nP)$ and $w((n+1)P)$, recursively [14]. In the following, we revisit the differential addition and doubling formulas for binary Edwards and generalized Hessian curves using w-coordinates [14] and [15].

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two affine points on the BEC E_{B,d_1,d_2}. Let us define $P_3 = P_1 + P_2 = (x_3, y_3)$, $P_4 = 2P_2 = (x_4, y_4) = (x_2, y_2) + (x_2, y_2)$, and $P_0 = P_2 - P_1 = (x_0, y_0) = (x_2, y_2) - (x_1, y_1)$. Then, one can write $w_3 = w(P_1 + P_2)$, $w_4 = w(2P_2)$, and $w_0 = w(P_2 - P_1)$ as defined above. In the mixed coordinate representation, w_i can be written as the fractions W_i/Z_i in projective, as $w_1 = w(P_1) = W_1/Z_1$ and $w_2 = w(P_2) = W_2/Z_2$, and w_0 is given as an affine field element. Then, the mixed w-coordinate addition (`MDiffADD`) of these two points can be obtained from [14] as

$$\begin{aligned} C &= W_1 \cdot (Z_1 + W_1) \\ D &= W_2 \cdot (Z_2 + W_2) \\ E &= Z_1 \cdot Z_2 \\ F &= W_1 \cdot W_2 \\ V &= C \cdot D \\ W_3 &= V + w_0 Z_3 \\ Z_3 &= V + \left(\sqrt{d_1} \cdot E + \sqrt{d_2/d_1 + 1} \cdot F\right)^2 \end{aligned} \quad (4)$$

and the formulas for w-coordinate doubling (`DiffDBL`) [14] are

$$\begin{aligned} C &= W_2 \cdot (Z_2 + W_2), \\ W_4 &= C^2 \\ Z_4 &= W_4 + \left(\left(\sqrt[4]{d_1} \cdot Z_2 + \sqrt[4]{d_2/d_1 + 1} \cdot W_2\right)^2\right)^2. \end{aligned} \quad (5)$$

For the GHCs, the w-coordinate differential addition formulas can be written as follows [15]:

$$\begin{aligned} A &= W_1 \cdot Z_2 \\ B &= W_2 \cdot Z_1 \\ C &= AB \\ U &= d^3 \cdot C \\ Z_3 &= (A + B)^2 \\ W_3 &= U + w_0 \cdot Z_3 \end{aligned} \quad (6)$$

and, similarly for doubling, those are presented as follows [15]:

$$\begin{aligned} A &= W_2^2 \\ B &= Z_2^2 \\ C &= A + \sqrt{c^3(d^3 + c)} \cdot B \\ D &= d^3 \cdot B \\ W_4 &= C^2 \\ Z_4 &= AD. \end{aligned} \quad (7)$$

The costs of different coordinates to compute differential addition and doubling are given in Table I for binary Edwards [14], generalized Hessian [15], and generic curves [19]. Let $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{D}$ be the costs of multiplication of two field elements, a squaring, and a multiplication by a constant curve parameter, respectively. As illustrated in this table, the mixed w-coordinate offers fast and comparable PA and PD formulas. Therefore, we use the mixed w-coordinate differential addition and doubling formulas [14]. Note that the difference of two points for differential addition is given in affine, i.e., $w_0 = w(P_2 - P_1)$.

TABLE I
COST OF POINT OPERATIONS ON BECS, GHCS, AND BINARY GENERIC CURVES (BGCS) OVER $GF(2^m)$ [14], [15], AND [19]

Curve	Curve Parameter	Combined Addition and Doubling[1]	
		Projective Diff	Mixed Diff
BEC [14]	$d_1 \neq d_2$	$8\mathbf{M}+4\mathbf{S}+2\mathbf{D}$	$6\mathbf{M}+4\mathbf{S}+4\mathbf{D}$
	$d_1 = d_2$	$7\mathbf{M}+4\mathbf{S}+2\mathbf{D}$	$5\mathbf{M}+4\mathbf{S}+2\mathbf{D}$
GHC [15]	$c \neq 1$	$7\mathbf{M}+4\mathbf{S}+3\mathbf{D}$	$5\mathbf{M}+4\mathbf{S}+3\mathbf{D}$
	$c = 1$	$7\mathbf{M}+4\mathbf{S}+2\mathbf{D}$	$5\mathbf{M}+4\mathbf{S}+2\mathbf{D}$
BGC [19]	$b \neq 0$	$7\mathbf{M}+5\mathbf{S}+1\mathbf{D}$	$5\mathbf{M}+5\mathbf{S}+1\mathbf{D}$

1. $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{D}$, are the costs of multiplication of two field elements, a squaring, and a multiplication by a constant, respectively.

Moreover, the mixed w-coordinate addition and doubling formulas are complete which means there is no need to check for the exceptional cases [14]. In order to have efficient computation of point operations, i.e., PAs and PDs, one needs to employ an efficient point multiplication algorithm. In the Section III, we give an explanation of using Montgomery's ladder for point multiplication.

III. POINT MULTIPLICATION ON BINARY EDWARDS AND GENERALIZED HESSIAN CURVES

Here, we consider Montgomery's ladder [37] and its modified version [19] to present a point multiplication algorithm over w-coordinates for binary Edwards, generalized Hessian, and binary generic curves. Using combined PA and PD formulations, we explain how parallelization can increase the performance of point multiplication. At the end, the cost of recovering final coordinates of point multiplication is derived.

A. Point Multiplication

The elliptic curve point multiplication is defined in the Abelian group as $Q = k \cdot P = P + P + \cdots + P$, ($k$ times), where k is a positive integer, and Q and P are two points on the elliptic curve $Q, P \in E(GF(2^m))$ [19]. The efficiency of point multiplication depends on finding the minimum number of steps to reach kP from a given point $P = (x_0, y_0)$. In binary Edwards and generalized Hessian curves, point multiplication can be defined similar to the one on generic curves [19]. Let P be a point on a BEC E_{B,d_1,d_2} and let us assume $w(nP)$ and $w((n+1)P)$, $0 < n < k$ are known. Therefore, one can use the w-coordinate differential addition and doubling formulas to compute their sum as $w((2n+1)P)$ and double of $w(nP)$ as $w(2nP)$.

Among different algorithms to perform point multiplication on elliptic curves, the Montgomery's ladder [37] is widely used in the literature. It has a uniform double-and-add structure which makes it secure against nondifferential (simple) side-channel attacks [14], [17]. In [19], an efficient version of Montgomery's algorithm is proposed over $GF(2^m)$. The Montgomery's ladder algorithm for point multiplication using mixed w-coordinates is provided in Algorithm 1. As shown in in Step 1 of this algorithm, the point $P = (x_0, y_0)$ is converted to the mixed w-coordinates by computing $w_0 = w(P) = x_0 + y_0$ and setting $W_1 = w_0$ and $Z_1 = 1$. Assume the scalar k is represented in binary, i.e., $k = \sum_{i=0}^{l-1} k_i 2^i$, $k_i \in GF(2)$. Then, the initialization steps, i.e., Steps 1a and 1b of Algorithm 1 , produce $w(P) = (W_1, Z_1)$ and $w(2P) = (W_2, Z_2)$ using (5) [38]. For BECs, the formulations of (4) and (5) are implemented in the `MDiffADD` and `DiffDBL` functions of Algorithm 1 , respectively. Therefore, after $l-1$ iterations as presented in Steps 2a and 2b of Algorithm 1, the w-coordinates of kP and $(k+1)P$, i.e., $w(kP) = (W_1, Z_1)$ and $w((k+1)P) = (W_2, Z_2)$, will be available. Similarly, for GHCs $w_0 = w(P) = 1 + dx_0y_0$, $d \neq 0$ is computed in Step 1 and $(W_1, Z_1) = (w_0, 1)$ is initialized in Step 1a for point multiplication [15]. For this curve, the formulations of (6) and (7) are implemented in `MDiffADD` and `DiffDBL` functions of Algorithm 1, respectively.

Algorithm 1 Montgomery's Algorithm **[37]**for Point Multiplication Using w -Coordinates.

Inputs: A point $P = (x_0, y_0) \in E(GF(2^m))$ on a binary curve and an integer $k = (k_{l-1}, \ldots, k_1, k_0)_2$.
Output: $w(Q) = w(kP) \in E(GF(2^m))$.
1: `set` : $w_0 \leftarrow x_0 + y_0$ and initialize
  a: $W_1 \leftarrow w_0$ and $Z_1 \leftarrow 1$
  b: $(W_2, Z_2) =$ `DiffDBL`(W_1, Z_1)
2: **for** i **from** $l-2$ down to 0 **do**
  a: **if** $k_i = 1$ **then**
    i): $(W_1, Z_1) =$ `MDiffADD`$(W_1, Z_1, W_2, Z_2, w_0)$
    ii): $(W_2, Z_2) =$ `DiffDBL`(W_2, Z_2)
  b: **else**
    i): $(W_1, Z_1) =$ `DiffDBL`(W_1, Z_1)
    ii): $(W_2, Z_2) =$ `MDiffADD`$(W_1, Z_1, W_2, Z_2, w_0)$
  end if
  end for
3: **return** $w(kP) \leftarrow (W_1, Z_1)$ and $w((k+1)P) \leftarrow (W_2, Z_2)$

B. Parallelism in Point Multiplication Algorithm

Parallelism is an approach to reduce the number of field arithmetic operations, mainly multiplications, in the critical path by using multiple multipliers concurrently [6]. In addition, merging point operations, i.e., the PA and PD, can result in less data dependency and can reduce the latency of the point multiplication over binary Edwards and generalized Hessian curves. Computing the w-coordinates of PA and PD for BECs together in one step of the Montgomery's algorithm requires six general finite field multiplications and four field multiplications by constants, as reported in Table I. As summarized in this table, for GHCs, the cost of combined PA and PD is five field multiplications and two multiplications by constants [15]. In the following, we explain how parallel field operations can be utilized to reduce the latency of the point multiplication operation.

1) Scheduling Field Operations for PA and PD: We have obtained the data dependency graphs for the combined PA and PD on binary Edwards and generalized Hessian curves as illustrated in Fig. 1 (Fig. 1(a) for $d_1 \neq d_2$ and Fig. 1(b) for $d_1 = d_2$) and Fig. 2(a), respectively. As shown in these figures, the latency (in terms of number of clock cycles) of each step is the latency of an operation with the longest latency. As one can see in Fig. 1(a) and (b), the first four operations of PA, i.e., Step 0 to Step 3, on BEC should be performed before any PD operation. This is because computation of PD depends on the PA. For generalized binary Hessian curve [Fig. 2(a)], operations of PA

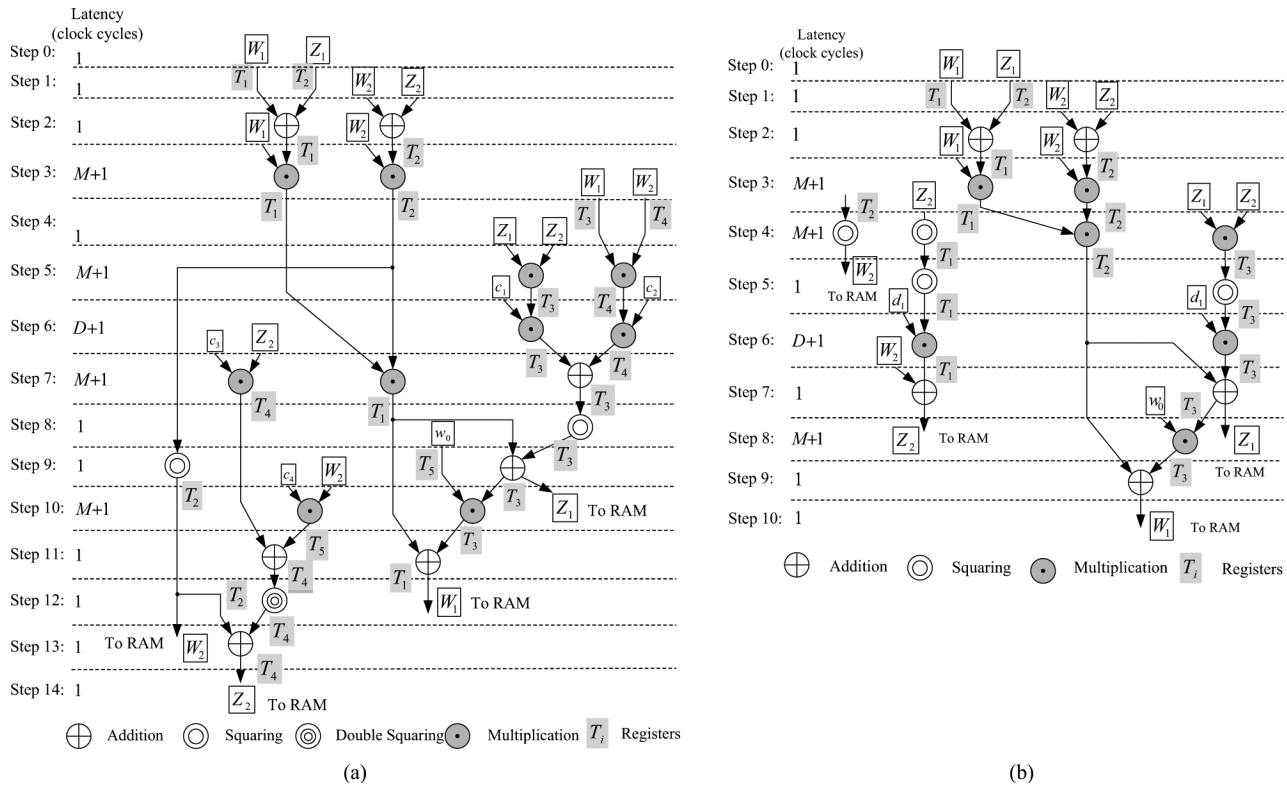

Fig. 1. Data dependency graphs for parallel computing of the combined PA and PD operations on BECs. (a) $d_1 \neq d_2$ and (b) $d_1 = d_2$ assuming $\mathcal{M} = 2$. It requires five registers of T_1, T_2, T_3, T_4, and T_5. The constant parameters, $c_1 = \sqrt{d_1}$, $c_2 = \sqrt{d_2/d_1 + 1}$, $c_3 = \sqrt{c_1}$, and $c_4 = \sqrt{c_2}$ are assumed to be precomputed and stored in the memory.

and PD can be performed in parallel at any time. Note that the latency of field additions and field squarings are negligible in comparison to the latency of the field multipliers. Therefore, we calculate the latency of the critical path in terms of number of field multiplications. Let M be the latency (in terms of number of clock cycles) for multiplying two field elements and D be the latency of multiplication of a field element by a constant (e.g., curve parameters, d_1 or d_2). Let us denote $\mathcal{M}$ as the number of parallel finite field multipliers. In the following, we investigate the parallelization using different numbers of multipliers $\mathcal{M} =$ 1,2, and 3.

2) BEC: For BECs with $d_1 \neq d_2$ and one available multiplier ($\mathcal{M} = 1$), the latency of the combined PA and PD is $6M + 4D$ as reported in Table I. Utilizing two multipliers, i.e., $\mathcal{M} = 2$, reduces the latency to $4M + 1D$ and $3M + 1D$ for $d_1 \neq d_2$ [Fig. 1(a)] and $d_1 = d_2$ [Fig. 1(b)], respectively. As one can see in Steps 3, 5, 6, 7, and 10 of Fig. 1(a), two independent multipliers are fully utilized. Thus, the utilization factor of two multipliers in Fig. 1(a) is 100%. Similarly, in Steps 3, 4, and 6 of Fig. 1(b), two multipliers are fully utilized. However, in Step 8 of Fig. 1(b), only one of the two multipliers is utilized [shown in Fig. 1(b)] and the other one is idle (not shown in Fig. 1(b)]. Therefore, the utilization factor of two multipliers in Fig. 1(b) is $7/8 \times 100 = 87.5\%$.

If three parallel multipliers, i.e., $\mathcal{M} = 3$, are employed, the latency will become $4M$ and $3M$ for $d_1 \neq d_2$ and $d_1 = d_2$, respectively. Therefore, adding one multiplier only reduces the latency by one multiplication by a constant. Moreover, one can figure out, the utilization factors for $d_1 \neq d_2$ and $d_1 = d_2$ will reduce to $10/12 \times 100 = 83.34\%$ and $7/9 \times 100 = 77.78\%$, respectively. In addition, employing four multipliers reduces the latency to $3M$ for $d_1 \neq d_2$ and has no impact for the case where $d_1 = d_2$. Note that employing more multipliers, i.e., $\mathcal{M} > 4$, does not decrease the latency. As a result, one can see the maximum utilization of the multipliers with low latency for the combined PA and PD operations is achieved only by choosing $\mathcal{M} = 2$. Multiplier utilization factors for data dependency graph of different curves are summarized in Table II. It is also worth noting that employing two multipliers for the case where $d_1 \neq d_2$ reduces the latency nearly 50% as compared to the case where only one multiplier is utilized.

TABLE II
MULTIPLIER UTILIZATION FACTORS FOR DATA DEPENDENCY GRAPH OF DIFFERENT CURVES

Curve	Utilization factor	
	$\mathcal{M} = 2$	$\mathcal{M} = 3$
BEC $d_1 \neq d_2$ (Fig. 1a)	100%	83.34%
BEC $d_1 = d_2$ (Fig. 1b)	87.5%	77.78%
GHC (Fig. 2a)	87.5%	77.78%
BGC (Fig. 2b)	100%	100%

3) GHC: For a GHC with $\mathcal{M} = 1$, the latency of combined PA and PD algorithm is $5M + 2D$. In such a case, the multiplier is always performed the operation and hence the utilization of multiplication for $\mathcal{M} = 1$ is 100%. The data dependency graph for GHC is illustrated in Fig. 2(a) using the combined PA and PD. In this figure, two multipliers, are available, i.e., $\mathcal{M} = 2$. As shown in Steps 2, 3, and 4 of Fig. 2(a), two multipliers operate in parallel, whereas, in Step 5 only one multiplier per-

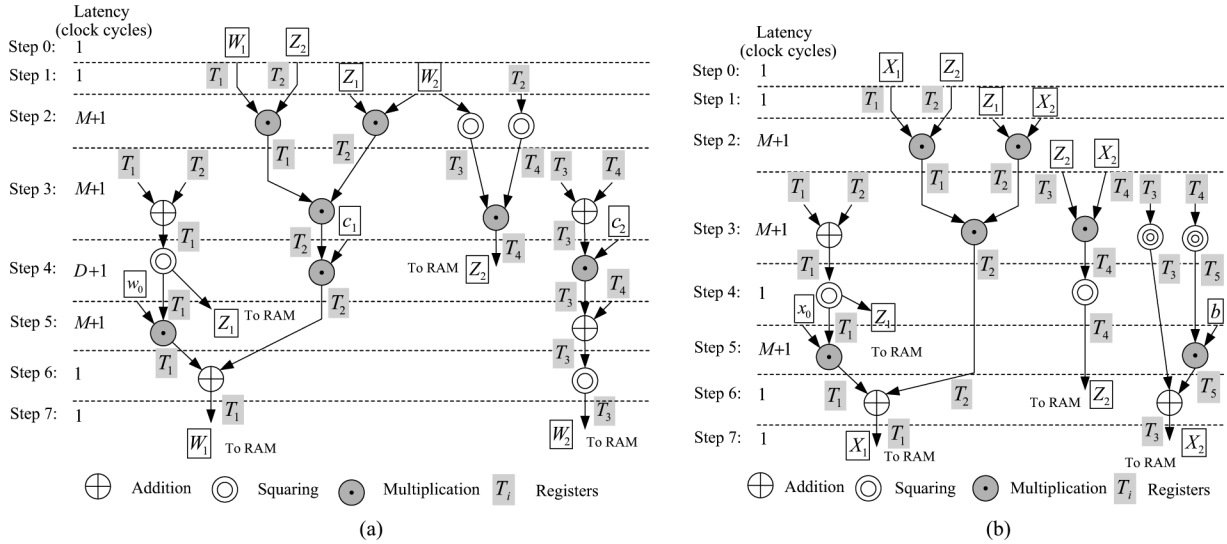

Fig. 2. Data dependency graph for parallel computing of the combined PA and PD operations for $\mathcal{M} = 2$ available multipliers on (a) GHCs, assuming $c_1 = d^3$, and $c_2 = 1/\sqrt{d^3}$ and (b) BGCs [39].

forms the multiplication. Therefore, the utilization for $\mathcal{M} = 2$ is $7/8 \times 100 = 87.5\%$. Also, the latency of computing the combined PA and PD operations in parallel is $3M + 1D$. Note that employing three parallel multipliers ($\mathcal{M} = 3$) reduces the latency to $2M + 1D$. However, one can figure out that only in a new step (including combination of Steps 2 and 3 in Fig. 2(a)) all three multipliers will be utilized and in Step 4, i.e., multiplication by constant, only one multiplier will perform the operation and the other two multipliers are idle. As a result, the utilization factor will reduce to $7/9 \times 100 = 77.78\%$. As one can figure out, increasing the number of multipliers from two to three reduces latency only 14% while increasing the required area about 33%.

4) Binary Generic Curve (BGC): For the sake of comparison, we have included a data dependency graph for BCGs employing two multipliers $\mathcal{M} = 2$ in Fig. 2(b) [39]. As seen from this figure, the latency of the combined PA and PD operations in parallel is $3M$. Incorporating three multipliers $\mathcal{M} = 3$ reduces the latency to $2M$ with multiplier utilization of 100% [7]. It is worth mentioning that employing more than three multipliers, i.e., $\mathcal{M} \geq 4$, will not reduce the latency of point multiplication. This has been investigated in a different way with $\mathcal{M} = 4$ to parallelize PA and PD operations as well as parallelizing finite field operations in [39]. We note that parallel computation of point multiplication over binary generic curves has been widely studied in the literature, for instance one can refer to [4]–[7], [11], and [39].

In the proposed architecture, multiplication by a constant is performed using one of the available multipliers. As a result, its cost is calculated the same as one of a multiplier.

As illustrated in Figs. 1 and 2, in each step, two words [e.g., W_1 and Z_1 in Step 0 of Fig. 1(a) and (b)] are read from the memory as the inputs (this is discussed in detail in Section IV-C). Consequently, this reduces the memory requirements. Scheduling has been made by two multipliers ($\mathcal{M} = 2$), two adders, and two squarers for efficient implementations. Also, addition and squaring can be performed in one clock cycle and multiplication using digit-level multiplier requires several $M = \lceil m/d \rceil$ clock cycles with an additional clock cycles for loading the inputs. Note that the order of operations are scheduled to achieve optimum number of clock cycles as illustrated in each step of data dependency graphs. At the end of point multiplication (the bottoms of data dependency graphs), the results of PAs and PDs for point multiplication are written to the memory. In what follows, we explain how to recover $Q = kP$ from P, $w(kP)$, and $w((k+1)P)$ at the end of the proposed Montgomery's point multiplication.

C. Recovering the Final Coordinates of x and y

In this paper, having w-coordinates in the last step of point multiplication, one can obtain $w(kP) = w_1 = W_1 \cdot Z_1^{-1}$ and $w((k+1)P) = w_2 = W_2 \cdot Z_2^{-1}$. The procedure of recovering the final point from w-coordinates is presented in [14]. At the end of differential addition, one has $w(kP)$, $w((k+1)P)$, and (x, y) for the base point P. First, one needs to check if $w_1^2 + w_1 \neq 0$ and then obtain $x_2^2 + x_2 = A'$ from the equation given in [14]. Since $\mathrm{Tr}(A') = 0$ [14], then employing linear half-trace H: $GF(2^m) \rightarrow GF(2)$ computation over $GF(2^{163})$, one has x_2 or $x_2 + 1$ as the output. With solving the curve equation for x_2 (or $x_2 + 1$), one can get y_2 (or $y_2 + 1$) whose cost is $I + 13M + 167S + 81A$ for $m = 163$. Note that inversion can be computed efficiently in normal basis using Itoh-Tsujii's scheme [40]. It requires $\lfloor \log_2(m-1) \rfloor + HW(m-1) - 1$ multiplications and $m-1$ squarings, where $HW(m-1)$ is the Hamming weight (number of ones) of the binary representation of $m-1$. Thus, for $m = 163$, the cost of an inversion is $9M + 162S$, where M and S are the costs (in terms of number of clock cycles in our analysis) to perform a finite field multiplication and squaring, respectively. Then, the total cost of recovering (x, y) coordinates of kP as a final point is $22M + 109$ clock cycles.

D. Latency of Point Multiplication Operations

The latency of point multiplication operations are summarized in Table III for $\mathcal{M} = 1, 2, 3$. The total latency consists

TABLE III
LATENCY OF THE OPERATIONS IN THE POINT MULTIPLICATION WITH $\mathcal{M} = 1, 2, 3$, WHERE M IS THE NUMBER OF CLOCK CYCLES REQUIRED FOR MULTIPLICATION OF TWO ARBITRARY FIELD ELEMENTS

Operation	Latency of Point Multiplication Operations			
Curve	BEC [14]		GHC [15]	BGC [19]
Parameter	$d_1 \neq d_2$	$d_1 = d_2$	$c = 1$	
Initialization	$1M + 5$	$1M + 5$	$1M + 3$	5
PA and PD, $\mathcal{M} = 1$	$10M + 21$	$7M + 16$	$7M + 10$	$6M + 10$
PA and PD, $\mathcal{M} = 2$	$5M + 15$	$4M + 11$	$4M + 8$	$3M + 8$
PA and PD, $\mathcal{M} = 3$	$4M + 9$	$3M + 7$	$3M + 9$	$2M + 5$
w-coordinate/affine $\mathcal{M} = 1$	$22M + 109$	$21M + 104$	$20M + 98$	$19M + 75$
w-coordinate/affine $\mathcal{M} = 2, 3$	$15M + 105$	$15M + 105$	$15M + 98$	$15M + 74$

of latencies of initialization (L_{initial}), computing PA and PD in the main loop (L_{loop}), and recovering the final point (L_R) for binary Edwards and generalized Hessian curves as follows:

$$L_{\text{Total}} = L_{\text{initial}} + (l-1) \times L_{\text{loop}} + L_R. \tag{8}$$

As shown in Table III, M is the number of clock cycles to multiply two field elements as well as a multiplication of a field element by a constant curve parameter. As an example, the latency of combined PA and PD with $\mathcal{M} = 2$ is calculated from Fig. 1(a) as $5M + 15$, by adding all clock cycles in 15 steps shown in Fig. 1(a), with an assumption of $D = M$.

IV. ARCHITECTURE OF THE ELLIPTIC CURVE CRYPTO-PROCESSOR

Here, we propose a hardware architecture for point multiplication over BECs, GHCs, and BGCs. A generic structure for the implementation of the point multiplication on FPGA platform is depicted in Fig. 3. The architecture is comprised of several blocks: a finite field arithmetic unit (FAU), a control unit, and memory. The FAU includes two field multipliers, two adders, and two squarers, as well as five 163-b registers to store intermediate results. The controller uses program instructions and implements finite state machine (FSM). The memory includes Block RAMs (BRAMs) and ROM to store the intermediate/final results and program instructions. The lower level (finite field) arithmetics are implemented in FAU and higher levels, i.e., PA and PD, are implemented in control logic as a FSM. In the following, we explain these blocks in detail.

A. FAU

In the binary field with characteristic two, $GF(2^m)$, addition is a bit-wise XOR and can be computed in one clock cycle. In normal basis, squaring of a field element is almost free (in hardware) in terms of both timing and area as it is equivalent to rewiring. The finite field multiplier plays the main role in determining the performance as it dominates the costs of point operations. Therefore, it is essential to design an efficient multiplier.

Bit-parallel multipliers can perform the finite field multiplication in one clock cycle. These multipliers are fast but require a large area complexity. Bit-serial multipliers require m clock cycles for the entire multiplication operation and they are efficient in terms of area but they are slow. Digit-level multipliers are the most suitable ones because the digit-size can be chosen for specific cryptographic applications based on the available resources. In this paper, we use a digit-level multiplier which is explained in the following.

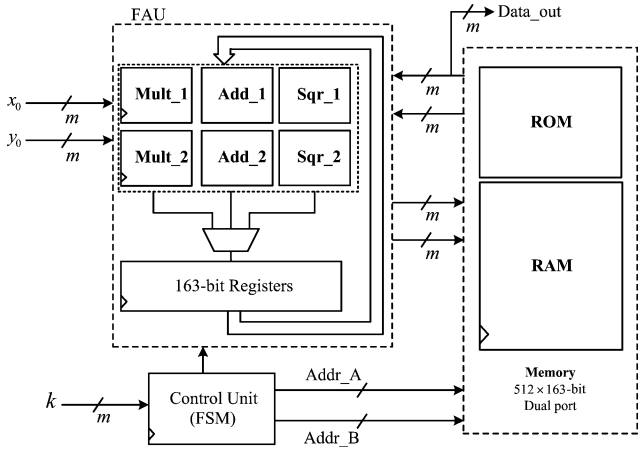

Fig. 3. Architecture of the proposed elliptic curve crypto-processor for BECs, GHCs, and BGCs.

B. Fast and Low-Complexity Digit-Level GNB Multiplier Over $GF(2^m)$

Here, we first present a pipelined low-complexity hardware architecture for digit-level GNB multiplier over $GF(2^m)$. Then, we evaluate the practical time–area efficiency of the presented multiplier by implementing it on a Xilinx Virtex-5 FPGA device.

1) Hardware Architecture: Let $A = (a_0, a_1, \ldots, a_{m-1})$ and $B = (b_0, b_1, \ldots, b_{m-1})$ be the field elements represented by type T GNB over $GF(2^m)$. Let $C = (c_0, c_1, \ldots, c_{m-1})$ denote their multiplication, i.e., $C = AB$. Reyhani–Masoleh in [23] has proposed a digit-level GNB multiplier with parallel output and digit-size $d, 1 \leq d \leq m$. It requires $M = \lceil m/d \rceil, 1 \leq M \leq m$, clock cycles to generate all the m coordinates of $C = AB$ simultaneously at the end of the final clock cycle. In [31], a modified and low-complexity version of the digit-level GNB multiplier proposed in [23] is presented. Here, we pipeline this architecture to make a faster VLSI architecture which operates at very high clock frequencies.

The used pipelined multiplier is depicted in Fig. 4. It consists of a ρ block, J blocks in Path-1, and the pipelined $GF(2^m)$ adder in Path-2. The ρ block includes two sub-blocks ρ_1 and ρ_2 and its structure depends on type T, $T \geq 2$, of GNB and multiplication matrix. Each J block consists of m two-input AND gates and each $GF(2^m)$ adder consists of binary trees of XOR gates. As illustrated in Fig. 4, the multiplier is pipelined by adding a stage of pipelined registers inside the $GF(2^m)$ adder

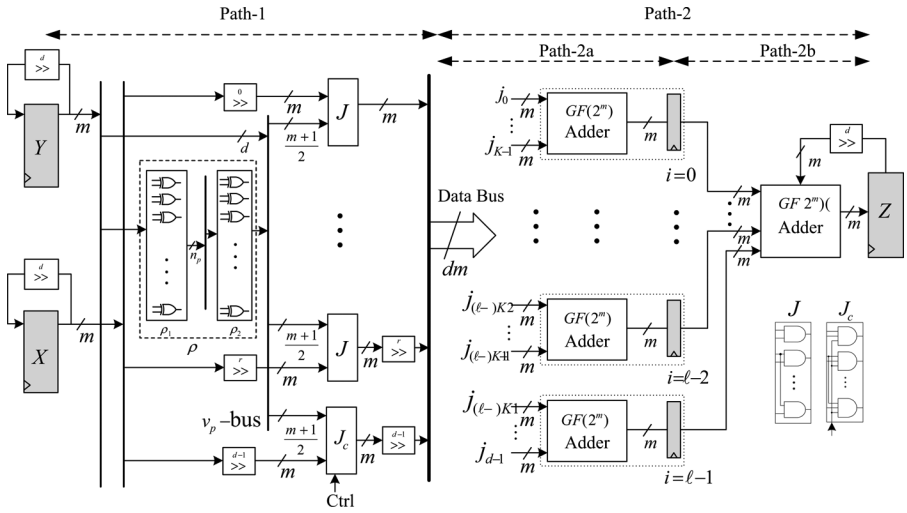

Fig. 4. Pipelined architecture of the low-complexity type T digit-level GNB multiplier with parallel-output [31].

TABLE IV
CRITICAL-PATH DELAY OF THE PIPELINED AND NONPIPELINED ARCHITECTURE OF PRESENTED DIGIT-LEVEL TYPE-4 GNB MULTIPLIER OVER $GF(2^{163})$

Non-Pipelined [23], [31]		Pipelined			
d	$D_{\text{Path-1}}+$ $D_{\text{Path-2}}$:	K	$D_{\text{Path-2a}}$: $\lceil \log_2 K \rceil T_X$	ℓ	$D_{\text{Path-2b}}$: $\lceil \log_2(\ell+1) \rceil T_X$
$2 \le d \le 3$	$T_A + 4T_X$	$2 < K \le 4$	$2T_X$	$2 \le \ell \le 3$	$2T_X$
$3 < d \le 7$	$T_A + 5T_X$	$4 < K \le 8$	$3T_X$	$3 < \ell \le 7$	$3T_X$
$7 < d \le 15$	$T_A + 6T_X$	$8 < K \le 16$	$4T_X$	$7 < \ell \le 15$	$4T_X$
$15 < d \le 31$	$T_A + 7T_X$	$16 < K \le 32$	$5T_X$	$15 < \ell \le 31$	$5T_X$
$31 < d \le 63$	$T_A + 8T_X$	$32 < K \le 64$	$6T_X$	$31 < \ell \le 63$	$6T_X$

in order to allow the multiplier to operate at very high clock frequencies. Therefore, instead of performing $GF(2^m)$ addition of dm inputs (as shown in Fig. 4), which are connected to the outputs of AND gates in J blocks, we perform the additions in two stages, i.e., over $\lceil dm/\ell \rceil$-inputs. The first stage contains $\ell\, GF(2^m)$ adders, each of which has at most $K = \lceil d/\ell \rceil$ m-bit inputs and are depicted by j_0 to j_{d-1} in the architecture. The outputs of the first adders are added with the output of the Z register using another $GF(2^m)$ adder in the second stage. Choosing the optimum value of ℓ plays an important role in designing the fast multiplier. This will be considered later in this section. It is shown in [23] and [31] that the critical-path delay of the non-pipelined multiplier is composed of the delays of the components located in Path-1 and path-2, i.e., $(\lceil \log_2 T \rceil T_X + T_A)$ and $(\lceil \log_2(d+1) \rceil T_X)$ for $1 \le d \le m$, respectively. Note that these are functions of the type of the multiplier T and the digit size d. As shown in Fig. 4, Path-2 is divided into Path-2a and Path-2b by inserting a stage of pipelined registers in between (hereafter we call it ℓ-level of accumulation). This technique reduces the number of logic gates in the critical-path and simplifies the routing.

2) Complexities: Here, we give the number of registers and time complexities of the pipelined digit-level GNB multiplier over $GF(2^m)$. The gate counts of the pipelined multiplier remains the same as the ones of the nonpipelined modified architecture presented in [31]. This requires dm AND gates and $n_p + v_p(T/2 - 1) + dm$ XOR gates, where n_p, $n_p \leqslant \min\left\{v_p T/2, \binom{m}{2}\right\}$ [31].

Proposition 1: The pipelined multiplier structure of Fig. 4 requires $(3+\ell)m$ registers and its critical-path delay is

$$\max\{(T_A + (\lceil \log_2 T \rceil + \lceil \log_2 K \rceil) T_X), (\lceil \log_2(\ell+1) \rceil T_X)\} \tag{9}$$

where ℓ is the level of accumulation and $K = \lceil d/\ell \rceil$.

Proof: As one can see from Fig. 4, ℓm registers are required between Path-2a and Path-2b for the pipeline purposes. As a result, the $(\ell + 3)m$ 1-bit registers required in the presented multiplier. The critical-path delay of Path-1, $D_{\text{Path}-1}$ is composed of the delays of the components in Path-1, i.e., T_X, $\lceil \log_2 T/2 \rceil T_X$, and T_A. The delay of Path-2a, $D_{\text{Path}-2\text{a}}$ is the delay of an m-bit $GF(2^m)$ adder with at most $K = \lceil d/\ell \rceil$ m-bit inputs, i.e., $\lceil \log_2 K \rceil T_X$, and the delay of Path-2b, $D_{\text{Path}-2\text{b}}$ is $\lceil \log_2(\ell+1) \rceil T_X$. Therefore, the critical-path delay of the presented architecture is $\max\{(D_{\text{Path}-1} + D_{\text{Path}-2\text{a}}), (D_{\text{Path}-2\text{b}})\}$ which completes the proof. ■

The critical-path delay of the pipelined and nonpipelined architecture of the presented multiplier in terms of number of levels of accumulation, ℓ and digit-size, d are illustrated in Table IV. It is noted that employing the proposed ℓ-level of accumulation using one stage of pipelined registers increases the latency of the multiplication by one clock cycle to $\lceil m/d \rceil + 1$.

3) LUT-Based Critical-Path Delay Analysis: Here, we investigate the critical-path delay of the presented pipelined

TABLE V
LUT-BASED CRITICAL-PATH DELAY (CPD) ($T_{\rm LUT}$) OF THE PRESENTED PIPELINED MULTIPLIER FOR DIFFERENT DIGIT SIZES (d) AND LEVELS OF ACCUMULATION (ℓ) FOR TYPE-4 GNB MULTIPLIER OVER $GF(2^{163})$ WHERE $K = \lceil d/\ell \rceil$

d	$D_{\text{Path-1}}$	$D_{\text{Path-2a}}$: $\lceil \log_6 K \rceil$	ℓ	$D_{\text{Path-2b}}$: $\lceil \log_6(\ell+1) \rceil$
$11 \le d \le 28$	$1T_{LUT}$	$1T_{LUT}$	$2 \le \ell \le 5$	$1T_{LUT}$
$33 \le d \le 163$	$1T_{LUT}$	$1T_{LUT}$	$6 \le \ell \le 28$	$2T_{LUT}$

scheme based on the six-input programmable LUTs available in Xilinx Virtex-5 FPGA device. To estimate resource consumption and critical-path delay, we need to convert the gate-oriented schematics to LUT-based schematics. Then, when the tree of XOR gates are converted into Γ-input ($\Gamma = 6$ in this case) LUT-oriented schematics the $\Gamma - 1$ XOR gates can be replaced by one LUT in the best case. For type $T \le 4$, each output of the ρ block is obtained by adding (XORing) of T inputs and considering the J block which includes an additional input for the AND operation. Therefore, such outputs can be implemented using six-input LUTs in $1T_{\rm LUT}$ delay. Then, the LUT-based critical-path delay of the Path-1 is $1T_{\rm LUT}$ for type $T \le 4$. The critical-path delay of Path-2 is summarized in Table V in terms of different levels of accumulation, ℓ and digit-size d. The critical-path delay of Path-2a and Path-2b are $\lceil \log_6 K \rceil T_{\rm LUT}$ and $\lceil \log_6(\ell+1) \rceil T_{\rm LUT}$, respectively. Therefore, K and ℓ should be chosen in such a way to have a balance for the LUT-based critical-path delay. For example, assume digit-size, $d = 55$ then the critical-path delay of the nonpipelined multiplier is $1T_{\rm LUT} + \lceil \log_6 56 \rceil T_{\rm LUT} = 4T_{\rm LUT}$. Employing $\ell = 10$ levels of accumulation results to have at most $K = \lceil 55/10 \rceil = 6$ inputs for each $GF(2^{163})$ adders in Path-2a. Then, the critical-path delay of the presented multiplier is $\max\{(1T_{\rm LUT} + \lceil \log_6 6 \rceil T_{\rm LUT}), (\lceil \log_6 11 \rceil) T_{\rm LUT}\} = 2T_{\rm LUT}$. Therefore, for practical implementations one needs to obtain optimum level of pipelining considering number of inputs of LUTs.

In this work, we have proposed an LUT-based pipelining scheme. We have tried several different pipelining techniques including the re-timing scheme of ISE tools but none of them was as efficient as the LUT-based analysis. Therefore, inserting pipelined registers in appropriate locations has a significant impact on the critical path delay of the proposed structure as the $GF(2^m)$ adder of the multiplier has the major critical path delay. In the following subsection, we implement the presented multiplier on FPGA.

4) Implementation: To evaluate the practical performance, the presented pipelined digit-level type-4 GNB multiplier over $GF(2^{163})$ is implemented on a Xilinx Virtex-5 FPGA device. First, feasible values for digit size d are chosen in such a way to decrease the critical-path delay while increasing the area (as a result of upper ceiling). Then, a careful LUT-based with floorplaning design is performed based on the given number of accumulators ℓ and digit-size d. The efficiency of the multiplier is measured in terms of reciprocal of the time-area products, i.e., $(\text{time} \times \text{area})^{-1}$ and is plotted for different digit sizes d, $11 \le d \le 82$, in Fig. 5. As shown in this figure, the local optimum (for time–area efficiency) in terms of digit sizes for the presented multiplier can be chosen as $d \in \{21, 24, 28, 33, 41, 55\}$. It is noted that two largest digit sizes of $d = 82$ and $d = 163$ degrade the maximum clock frequencies as the place and route (PAR) operation becomes complicated. Therefore, we exclude $d = 163$ from our analysis and keep $d = 82$ for comparison purposes. The presented multiplier is faster (i.e., operates at high clock frequencies) and is smaller than the digit-level MO multiplier employed in [6] for FPGA implementations of ECC over $GF(2^{163})$[23].

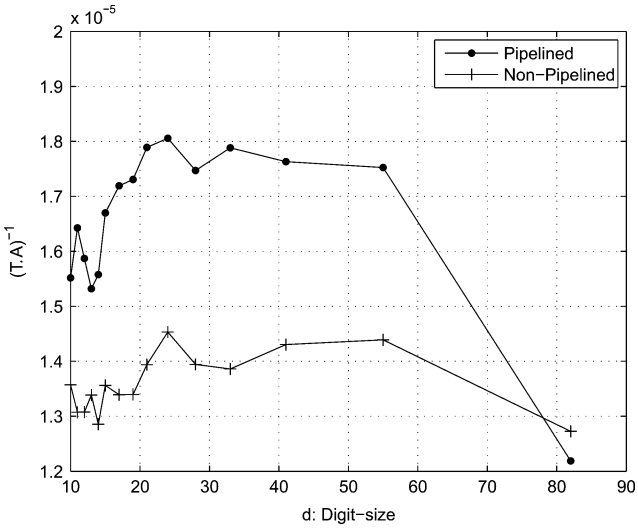

Fig. 5. Time–area ratio of the presented pipelined low-complexity digit-level GNB multiplier for type 4 over $GF(2^{163})$ for different digit sizes d.

C. Memory and Control Unit

1) Memory: The proposed architecture requires RAM to store intermediate and variables output as from the FAU and registers and ROM to store program instructions and constant values. As illustrated in Figs. 1 and 2, in each cycle two words (163-bit) from memory are accessed. Then, dual-port BRAMs are configured as two single-port BRAMs with independent data access [41]. One can perform two read operations per cycle by using a dual-port BRAM. This feature allows us to reduce the number of required BRAMs and achieve greater utilization of this resource. In the utilized Xilinx Virtex-5 FPGA device, 36-Kb (1024 36-b words) dual port BRAM blocks are available with a combined 72-b bus width (36-b per port). The dual-port RAM is assigned through Xilinx Synthesis Tool (XST). In Fig. 3, the storage RAM has been designed to allow the reading and writing of the 163-b words for $m = 163$. This results in minimizing the number of accesses to the memory. Therefore, as shown in Fig. 6, the storage RAM is constructed with $\lceil 163 \times 2/72 \rceil = 5$ BRAMs resulting in the storage of 512×163-b words to store the intermediate inputs as illustrated in the data flow diagrams of Figs. 1 and 2.

The basic field arithmetic operations, i.e., multiplication, addition, and squaring, are implemented in the FAU. The constants d_1, d_2, c_1, c_2, c_3, and c_4 are stored in the ROM. The ROM to store constants, is implemented with the same BRAM explained above by reserving a few addresses. A register file of 5×163-b

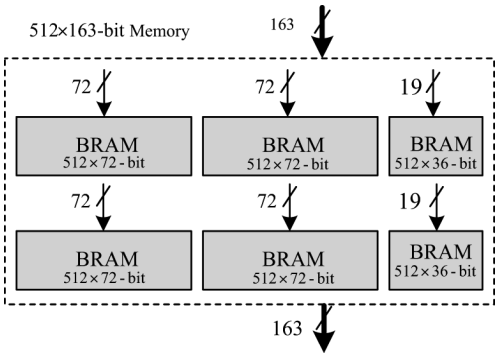

Fig. 6. Configuration of BRAMs for the proposed architecture.

registers (shown by T_i in Figs. 1 and 2) is incorporated in the FAU to reduce the overhead of the communication between the FAU and the RAM. It is noted that the load and store between the FAU and the memory storage require a single clock cycle. We count all of these clock cycles when calculating the total latency of the point multiplication. The ROM is also generated using Xilinx BRAMs as illustrated in Fig. 3. In Table III, the latency of the operations required to perform arithmetic operations are reported.

2) Control Unit: The control unit of the ECC crypto-processor controls the FAU and memory and it is implemented as a FSM. As shown in Fig. 3, the control unit has two address signals, Addr_A and Addr_B, which control the interface between the FAU and the memory. The program instructions are stored in ROM and the control unit fetches and decodes instructions and sends appropriate control signals to the other units based on the presented data dependency graphs of Figs. 1 and 2. Note that the ROM that stores the program instructions is instantiated using BRAMs as 1024×36-b words. Therefore, to store program instructions one extra BRAM is required. It is noted that the control unit decides where to store and conditionally swamp (based on k_i) the results of the combined PA and PD operations.

V. Comparisons and Implementations

Here, we discuss the results obtained in the previous sections and compare them with the counterparts in terms of side-channel analysis and implementation results.

A. Side-Channel Analysis

As mentioned before, Montgomery's Ladder is highly regular and suitable choice to protect scalar k against simple power analysis attacks [42]. Newly introduced BECs and GHCs have two special properties of being unified and complete [14]. The former is that the point addition formulations can be used for point doubling while the latter means that point addition formulations can be used for all pairs of inputs on the curve. Then, the point multiplication algorithm based on unified addition and doubling operations, will not cause side-channel leakage and hence it is protected against side-channel attack (SCA). Baldwin *et al.* [43] have investigated resistivity against simple power analysis (SPA) attacks of the unified operations for twisted Edwards curves over prime fields $GF(p)$. Also, this fact has been investigated in [17] using the unified addition formula of BECs. They have also taken advantage of incorporating a simple random order execution (i.e., randomly changing the storage location of the results) in the Montgomery's ladder that makes the differential power analysis (DPA) attack difficult [17]. In this work, we take advantage of completeness of w-coordinates differential PA and PD formulas on Montgomery's ladder which is also SPA-resistant.

The cost of explicit point addition is $8\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$ for generic curves [44], $13\mathbf{M}+3\mathbf{S}+3\mathbf{D}$ for BECs [14], and $8\mathbf{M}+3\mathbf{S}$ for GHCs [15]. Therefore, the GHCs offer the fastest addition formulas for binary elliptic curves. Although the explicit addition formulas for generic curves are faster than BECs, they are not complete and unified. Therefore, one can realize that the cost of one step of point multiplication on BECs using explicit addition formulas in [17] is higher than employing Montgomery's differential addition algorithm, i.e., combined differential PA and PD. It is interesting to note that one can reduce this cost by employing explicit addition formulas for GHCs.

B. Implementation Results and Discussion

We have selected the Xilinx Virtex-5 xc5vlx110-2ff1760 device as the target FPGA. In terms of available resources, xc5vlx110-2ff1760 contains 17 280 slices (69 120 LUTs and 69 120 registers), 128 BlockRAMs (BRAMs), and 800 input/output (I/O) pins. Each slice contains four flip-flops (FFs) and four LUTs [41].

Choosing Xilinx Virtex-5 FPGA would increase the performance and speed of our design. This is mainly due to the availability of six-input LUTs and large word size in its high 36-Kb BRAMs. Having six-input LUTs helps the design to be implemented with fewer logic levels and availability of large word size makes it easier to build large memory arrays (for storing large-bit field elements over $GF(2^m)$) with less routing delay. As a result, using Xilinx Virtex-5 FPGAs increases the speed by reducing both the critical-path delay and number of clock cycles (latency). Note that for the comparison purpose, we also implement the proposed design on a Xilinx Virtex-4 xc4vlx100 device (which offers four input LUTs) and compared it with the counterparts.

The presented architecture for elliptic curve crypto-processor of Section IV is coded in VHDL and synthesized for different digit sizes d, $d \in \{21, 24, 28, 33, 41, 55, 82\}$ using XST of Xilinx ISE 12.1 design software. The optimization goal for synthesize is set to the default value (i.e., speed). The results of the timing analysis of the implementations after the post place and route are reported in Tables VI and VII for BECs and GHCs, respectively. The number of required clock cycles for computing the point multiplication is also presented in these tables for the different digit sizes and different curve parameters, i.e., $d_1 = d_2$, $d_1 \neq d_2$, and $c = 1$. Moreover, the total latencies are found from (8) using $l = 163$ as the summation of the required clock cycles for the initialization, the total PA and PD in of the point multiplication, and the conversion as obtained from Table III.

The area requirements are stated in terms of the number of occupied slices (including LUTs and FFs) as reported in Tables VI and VII. Note that the proposed architecture for the FAU is the same for binary Edwards (with $d_1 = d_2$ and $d_1 \neq d_2$) and generalized Hessian curves, but they only differ in the control logic provided by instruction program (in ROM) and the number of

TABLE VI
IMPLEMENTATION RESULTS FOR BECs OVER $GF(2^{163})$ AND $\mathcal{M} = 2$

d	$M+1$	**Latency** Clock Cycles (L_{Total})		$f_{\max}$ (MHz)	**Area**						**P.M. Time** [μs]	
					LUTs		FFs		Slices			
		$d_1 \neq d_2$	$d_1 = d_2$		$d_1 \neq d_2$	$d_1 = d_2$	$d_1 \neq d_2$	$d_1 = d_2$	$d_1 \neq d_2$	$d_1 = d_2$	$d_1 \neq d_2$	$d_1 = d_2$
21	9	10041	7915	269.3	8158	8158	3097	2771	3181	3181	37.2	29.3
24	8	9208	7247	268.8	8750	8750	3423	3097	3371	3371	34.2	26.9
28	7	8375	6579	267.5	10309	10309	3423	3097	4078	4078	31.3	24.6
33	6	7542	5911	265.8	11139	11139	3249	3423	4681	4681	28.3	22.2
41	5	6709	5243	264.5	14235	14235	4075	3749	5788	5788	25.3	19.8
55	4	5876	4575	263.3	17432	17432	5053	4727	6536	6536	22.3	17.3
82	3	5043	3907	196.1	23301	23301	6357	6031	8872	8872	25.7	19.9

TABLE VII
IMPLEMENTATION RESULTS FOR GHC OVER $GF(2^{163})$ AND $\mathcal{M} = 2$

d	$M+1$	**Total Latency** Clock Cycles (L_{Total})	$f_{\max}$ (MHz)	**Area**			**P.M. Time** [μs]
				LUTs	FFs	Slices	
21	9	7419	272.3	8158	2934	3181	27.2
24	8	6751	271.8	8750	3260	3371	24.8
28	7	6083	269.3	10309	3260	4078	22.5
33	6	5415	268.2	11139	3586	4681	20.1
41	5	4747	267.1	14235	3912	5788	17.7
55	4	4079	266.2	17432	4890	6536	15.3
82	3	3411	196.1	23301	6194	8872	17.3

TABLE VIII
IMPLEMENTATION RESULTS FOR BGC OVER $GF(2^{163})$ AND $\mathcal{M} = 2$

d	$M+1$	**Total Latency** Clock Cycles (L_{Total})	$f_{\max}$ (MHz)	**Area**			**P.M. Time** [μs]
				LUTs	FFs	Slices	
21	9	5884	272.3	8158	3097	3181	21.6
24	8	5383	271.8	8750	3423	3371	19.8
28	7	4882	269.3	10309	3423	4078	18.1
33	6	4381	268.2	11139	3249	4681	16.3
41	5	3880	267.1	14235	4075	5788	14.5
55	4	3379	266.2	17432	5053	6536	12.7
82	3	2878	196.1	23301	6357	8872	14.7

required registers. Therefore, the area is equal for theses curves as presented in Tables VI and VII. The fastest point multiplications are computed for digit size $d = 55$ at approximately 17.3 μs and 15.3 μs for binary Edwards and generalized Hessian curves, respectively. The proposed architecture requires almost 6536 occupied slices (17 432 LUTs and 5053 FFs) and 6 BRAM blocks for $d = 55$. Similar implementation results are found for binary generic curve as illustrated in Table VIII.

It is noted that, from our implementations results (Tables VI–VIII), one can see that the slices occupation is usually larger than the number of LUTs divided by four ($\#LUT/4$) for Virtex-5. This is because the ISE design software starts the unrelated logic packing after the CLB pack factor (100% for the default value) is reached [41]. A higher percentage number will result in lower density packing and a lower pack factor results in a denser design with a difficult place and route and consequently higher delays.

Several implementations of ECC have been published in the literature targeting various applications with different requirements in terms of time–area tradeoffs. The implementation results of this work are reported in Table IX and are compared with the results for generic and Koblitz curves available in the literature. We note that because different curves and different FPGA technologies are used to implement different crypto-processors, meaningful quantitative comparisons of the area and time results are difficult. Therefore, as mentioned above we have implemented the crypto-processor for $d = 55$ on Virtex-4 device and its area and timing results are reported in Table IX. Moreover, as the finite field multiplier plays an important role in determining the performance of an ECC crypto-processor, we discuss the performance results in terms of efficiency of the finite field multiplier and fairly compared them with the counterparts.

It is worth mentioning that in these implementations, we have chosen normal basis as it offers free repeated squarings. Also, we could have taken more advantages of normal basis as it is utilized for Koblitz curves in [6] and [9]. However, by using normal basis, we have eliminated the extra hardware for squarings for the proposed ECC crypto-processor over BECs and GHCs. Moreover, recovering final coordinates (x, y) of $Q = kP$ (represented in w-coordinates) requires several repeated squarings and Half-trace computation, that their costs are reduced by using normal basis.

In [6], Järvinen *et al.* have presented the use of parallelization on different levels of point multiplication and have extensively studied the speed and area requirements for NIST B-163 and K-163 curves. For generic curves, the time-area performances are investigated using one, two, and four digit-level MO [25] multipliers over $GF(2^{163})$. As discussed in [23], the area complexity of a digit-level MO multiplier and its improved version is larger than the one presented in this work. Also, as one can realize, time complexity of our presented multiplier is less than digit-level MO multiplier as compared in [23]. In addi-

TABLE IX
COMPARISON OF ECC IMPLEMENTATIONS ON AN FPGA OVER $GF(2^{163})$

Work[1]	Device	Basis	Curve	d	$\mathcal{M}$	Area	Time [μs]
BGC [6]	Stratix II	NB	Generic	14	4	11,800 ALMs	48.88
BKC [6]	Stratix II	NB	Koblitz	11	4	13,472 ALMs	25.81
BKC [12]	Stratix II	NB	Koblitz	17	4	23,580 ALMs (26,647 ALUTs, 20575 FFs)	9.48
BGC [6]	Stratix II	NB	Generic	41	2	18,489 ALMs	51.56
BKC [6]	Stratix II	NB	Koblitz	41	2	19,498 ALMs	35.1
This work BGC	Virtex-5	NB	Generic	41	2	5,788 Slices (14,235 LUTs, 4,075 FFs)	14.4
This work BEC ($d_1 \neq d_2$)	Virtex-5	NB	Edwards	41	2	5,788 Slices (14,235 LUTs, 4,075 FFs)	24.9
This work BEC ($d_1 = d_2$)	Virtex-5	NB	Edwards	41	2	5,788 Slices (14,235 LUTs, 3,749 FFs)	19.5
This work GHC ($c = 1$)	Virtex-5	NB	Hessian	41	2	5,788 Slices (14,235 LUTs, 3,912 FFs)	17.4
BGC [7]	Virtex-4	NB	Generic	55	3	24,363 Slices	10.11
This work BGC	Virtex-4	NB	Generic	55	2	12,834 Slices (22,815 LUTs, 6,683 FFs)	17.2
This work BEC ($d_1 \neq d_2$)	Virtex-4	NB	Edwards	55	2	12,834 Slices (22,815 LUTs, 6,683 FFs)	22.9
This work BEC ($d_1 = d_2$)	Virtex-4	NB	Edwards	55	2	12,834 Slices (22,815 LUTs, 6,520 FFs)	23.3
This work GHC ($c = 1$)	Virtex-4	NB	Hessian	55	2	12,834 Slices (22,815 LUTs, 6,520 FFs)	20.8
This work BGC	Virtex-5	NB	Generic	55	2	6,536 Slices (17,305 LUTs, 4,075 FFs)	12.9
This work BEC ($d_1 \neq d_2$)	Virtex-5	NB	Edwards	55	2	6,536 Slices (17,432 LUTs, 5,053 FFs)	21.8
This work BEC ($d_1 = d_2$)	Virtex-5	NB	Edwards	55	2	6,536 Slices (17,432 LUTs, 4,727 FFs)	16.9
This work GHC ($c = 1$)	Virtex-5	NB	Hessian	55	2	6,536 Slices (17,305 LUTs, 4,890 FFs)	14.9

1. BGC: binary generic curve, BKC: binary Koblitz curve, BEC: binary Edwards curve, GHC: generalized Hessian curve.

tion, we have reached higher clock frequencies with LUT-based pipelining techniques as well. Further, the implementations in [6, Table VII] for generic curves over $GF(2^{163})$ require higher latency and subsequently larger computation time.

In [12], the same digit-level MO multiplier, has been used for point multiplication on Koblitz curves and has been compared with the results of using polynomial basis. The authors indicated that implementation results using polynomial basis is faster than the ones using normal basis having the same area [12, Table IV]. They have also taken advantage of operation interleaving in their implementations on Koblitz curves. However, it is worth mentioning that the large area consumption of the implementations results of using normal basis in [12] might be as a result of large number of pipelined registers and the implementations results of [12] can be improved using our proposed scheme. Therefore, if one employs our presented multiplier architecture incorporating the techniques proposed in [12], the results of point multiplication using normal basis would be comparable with the ones using polynomial basis. We further note that our implementations are not claimed to be the best possible and faster than counterparts using polynomial basis.

The point multiplication scheme proposed in [7] by Kim *et al.*. has been performed on NIST B-163 generic curve employing $\mathcal{M} = 3$ digit-serial GNB multipliers (proposed by Kwon *et al.* in [22]) with Montgomery's ladder on a four-input Virtex-4 FPGA. The maximum clock frequency that is reported for the ECC crypto-processor is $f_{\max} = 143$ MHz achieved with digit-size $d = 55$. Therefore, as the multiplier determines the upper bound for critical-path delay, one can estimate that the maximum operating frequency for the multiplier is 143 MHz. However, our presented multiplier operates at $f_{\max} =$ 196.5 MHz on Virtex-4 FPGA with only one level of pipelining. We further note that the proposed LUT-based pipelining technique has significant increase on $f_{\max}$. Moreover, the latency of point multiplication (i.e., the number of clock cycles) in [7] is $L_{\text{Total}} = 1 + 162 \times (2M + 2) + 149 = 1446$ employing three multipliers and hence the total time achieved for point multiplication is $T_{kP} = L_{kP}/f_{\max} = 1446/143 = 10.11$ μs with occupying 24 363 slices. Our implementation on Virtex-4 FPGA uses only two GNB multiplier and computes a point multiplication in 17.2 μs with using only 12 834 slices as reported in Table IX.

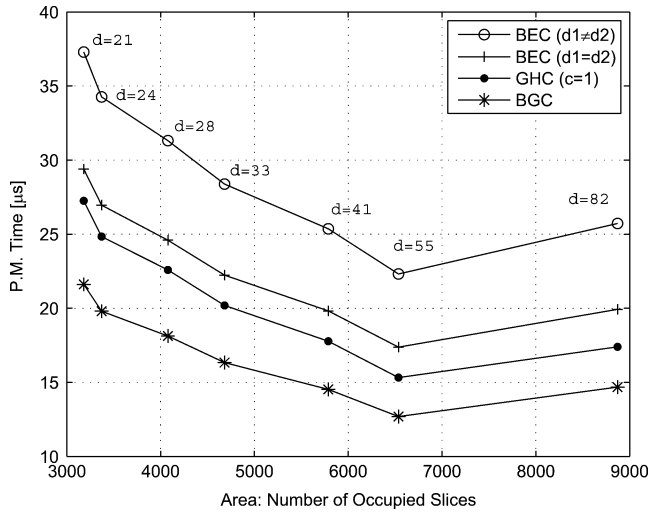

Fig. 7. Implementation results of point multiplication for BECs, GHCs, and BGCs reported in Tables VI–VIII on Xilinx Virtex-5 xc5vlx110-2ff1760 FPGA device. The points are related to digit sizes of $d = 21, 24, 28, 33, 41, 55, 82$.

Table IX shows a number of related designs (on NIST B-163 and K-163) which are implemented on different FPGA platforms using different types and number of multipliers. To have a fair comparison, we have implemented the ECC crypto-processor based on NIST B-163 generic curve using the presented GNB multiplier for different digit sizes. Data dependency graph of point multiplication of this curve has been illustrated in Fig. 2(b) as its latencies are summarized in Table III. Their implemented results are tabulated in Table VIII.

In Fig. 7, the implementation results are illustrated and point multiplication time is plotted versus area (number of occupied slices). As shown in this figure, increasing the area, as a result of increasing digit-size d, results in faster point multiplications. It is noted that larger digit sizes than 55, i.e., $d > 55$, are not efficient for the proposed architecture as it is seen from Fig. 7. Therefore, incorporating multiple smaller multipliers is more efficient than using of a large multiplier. As illustrated in

Table VIII and Fig. 7, our results indicate that the point multiplication over BGCs is faster than BECs and GHCs. This is because it has smaller latency which requires fewer number of clock cycles.

We further note that the implementations of point multiplication over binary generic curves (short Weierstraβ) require special hardware to handle point at infinity. Then, during each point operation, a check should be performed to ensure that the resulting point is not at infinity. It should be noted that the proposed ECC crypto-processor for BECs and GHCs works for all of the input pairs without any changes (i.e., it is complete). However, exceptional cases should be tested separately for the case employing NIST generic and Koblitz curves which requires extra hardware and time.

VI. Conclusion

In this paper, we have investigated the hardware implementation of point multiplication on BECs and GHCs over $GF(2^{163})$ using GNB. We have presented a pipelined digit-level Gaussian normal basis multiplier which operates in higher clock frequencies and studied its time–area tradeoffs for different digit sizes. The effect of parallelization using two multipliers for computing the point addition and point doubling on BECs and GHCs has been investigated. For point multiplication, the widely used Montgomery's ladder has been incorporated for differential w-coordinates. The proposed architecture has been implemented on FPGA to obtain the optimum digit-size. Also, we have examined the completeness of the point operations. For BECs and GHCs, the fastest point multiplication achieved with choosing $d = 55$. The proposed architecture requires 6536 occupied slices (17 432 LUTs and 5053 FFs), and computes a single point multiplication in 17.3 μs and 15.3 μs for BECs and GHCs, respectively. Our implementation results also indicate that the point multiplication over BGCs is faster than over BECs and GHCs. On the other hand, the point multiplication over BECs and GHCs is complete.

Acknowledgment

The authors would like to thank the reviewers for their constructive comments. The authors would also like to thank Canadian Microelectronics Corporation (CMC) Microsystems for providing the required infrastructure and CAD tools that have been used in this work.

References

[1] *U.S. Dept. of Commerce/NIST, Digital Signature Standard, FIPS Publications 186-2*, , Jan. 2000.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Adv. Cryptol.*, 1986, pp. 417–426.

[3] N. Koblitz, "Elliptic curve cryptosystems," *Math. Computation*, vol. 48, pp. 203–209, 1987.

[4] R. Cheung, N. Telle, W. Luk, and P. Cheung, "Customizable elliptic curve cryptosystems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 9, pp. 1048–1059, Sep. 2005.

[5] B. Ansari and M. Hasan, "High-performance architecture of elliptic curve scalar multiplication," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1443–1453, Nov. 2008.

[6] K. Järvinen and J. Skyttä, "On parallelization of high-speed processors for elliptic curve cryptography," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 9, pp. 1162–1175, Sep. 2008.

[7] C. H. Kim, S. Kwon, and C. P. Hong, "FPGA implementation of high performance elliptic curve cryptographic processor over $GF(2^{163})$," *J. Syst. Architecture*, vol. 54, no. 10, pp. 893–900, 2008.

[8] Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-curve-based security processor for RFID," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1514–1527, Nov. 2008.

[9] V. S. Dimitrov, K. U. Järvinen, M. J. , Jr, W. F. Chan, and Z. Huang, "Provably sublinear point multiplication on Koblitz curves and its hardware implementation," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1469–1481, Nov. 2008.

[10] W. Chelton and M. Benaissa, "Fast elliptic curve cryptography on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 2, pp. 198–205, Feb. 2008.

[11] M. Keller, A. Byrne, and W. P. Marnane, "Elliptic curve cryptography on FPGA for low-power applications," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 1, pp. 1–20, 2009.

[12] K. Järvinen and J. Skyttä, "Fast point multiplication on Koblitz curves: Parallelization method and implementations," *Microprocessors Microsyst.*, vol. 33, no. 2, pp. 106–116, 2009.

[13] Y. Zhang, D. Chen, Y. Choi, L. Chen, and S.-B. Ko, "A high performance ECC hardware implementation with instruction-level parallelism over $GF(2^m)$," *Microprocessors Microsyst.—Embedded Hardware Design*, vol. 34, no. 6, pp. 228–236, 2010.

[14] D. Bernstein, T. Lange, and R. Farashahi, "Binary Edwards curves," in *Proc. Workshop Cryptograph. Hardware Embedded Syst.*, 2008, vol. 5154, pp. 244–265.

[15] R. Farashahi and M. Joye, "Efficient arithmetic on Hessian curves," in *Proc. 13th Int. Conf. Practice Theory of Public Key Cryptography*, 2010, pp. 243–260.

[16] U. Kocabas, J. Fan, and I. Verbauwhede, "Implementation of binary Edwards curves for very-constrained devices," in *Proc. 21st Int. Conf. Application-specific Syst. Architectures Processors*, 2010, pp. 185–191.

[17] L. Batina, J. Hogenboom, N. Mentens, J. Moelans, and J. Vliegen, "Side-channel evaluation of FPGA implementations of binary Edwards curves," in *Proc. 17th IEEE Int. Conf. Electron., Circuits Syst.*, 2010, pp. 1255–1258.

[18] R. Moloney, A. O'Mahony, and P. Laurent, "Efficient implementation of elliptic curve point operations using binary Edwards curves," Cryptology ePrint Archive Rep. 2010/208, 2010. [Online]. Available: http://eprint.iacr.org/

[19] J. López and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," in *Proc. Workshop Cryptograph. Hardware Embedded Syst.*, 1999, pp. 316–327.

[20] M. A. Hasan, "Look-up table-based large finite field multiplication in memory constrained cryptosystems," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 749–758, Jul. 2000.

[21] S. Kumar, T. Wollinger, and C. Paar, "Optimum digit serial $GF(2^m)$ multipliers for curve-based cryptography," *IEEE Trans. Comput.*, vol. 55, no. 10, pp. 1306–1311, Oct. 2006.

[22] S. Kwon, K. Gaj, C. H. Kim, and C. P. Hong, "Efficient linear array for multiplication in $GF(2^m)$ using a normal basis for elliptic curve cryptography," in *Proc. Workshop Cryptograph. Hardware Embedded Syst.*, 2004, pp. 76–91.

[23] A. Reyhani-Masoleh, "Efficient algorithms and architectures for field multiplication using Gaussian normal bases," *IEEE Trans. Comput.*, vol. 55, no. 1, pp. 34–47, Jan. 2006.

[24] R. Dahab, D. Hankerson, F. Hu, M. Long, and M. Lopez, "Software multiplication using Gaussian normal bases," *IEEE Trans. Comput.*, vol. 55, no. 7, pp. 974–984, Jul. 2006.

[25] J. Massey and J. Omura, "Computational Method and Apparatus for Finite Arithmetic," U.S. Patent 4 587 627, 1986.

[26] B. Sunar and Ç. K. Koç, "An efficient optimal normal basis type II multiplier over $GF(2^m)$," *IEEE Trans. Comput.*, vol. 50, no. 1, pp. 83–87, Jan. 2001.

[27] S. Kwon, "A low complexity and a low latency bit parallel systolic multiplier over $GF(2^m)$ using an optimal normal basis of type II," in *Proc. 16th IEEE Symp. Comput. Arithmetic*, 2003, pp. 196–202.

[28] J. Gathen, A. Shokrollahi, and J. Shokrollahi, "Efficient multiplication using type 2 optimal normal bases," in *Proc. 1st Int. Workshop Arithmetic of Finite Fields*, 2007, vol. 4547, pp. 55–68.

[29] H. Fan and M. Hasan, "Subquadratic computational complexity schemes for extended binary field multiplication using optimal normal bases," *IEEE Trans. Comput.*, vol. 56, no. 10, p. 1435, Oct. 2007.

[30] D. Bernstein and T. Lange, "Type-II optimal polynomial bases," in *Proc. 3rd Int. Workshop Arithmetic Finite Fields*, 2010, vol. 6078, pp. 41–61.

[31] R. Azarderakhsh and A. Reyhani-Masoleh, "A modified low complexity digit-level Gaussian normal basis multiplier," in *Proc. 3rd Int. Workshop Arithmetic of Finite Fields*, 2010, vol. 6087, pp. 25–40.
[32] A. Menezes, I. Blake, S. Gao, R. Mullin, S. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. Boston, MA: Kluwer, 1993.
[33] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. Cambridge, U.K.: Cambridge Univ. Press, 1994.
[34] *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Std 1363-2000, Jan. 2000.
[35] J. Fan, D. Bailey, L. Batina, T. Guneysu, C. Paar, and I. Verbauwhede, "Breaking elliptic curves cryptosystems using reconfigurable hardware," in *Proc. 20th Int. Conf. Field Programmable Logic and Applications*, 2010, pp. 133–138.
[36] D. Hankerson, S. Vanstone, and A. Menezes, *Guide to Elliptic Curve Cryptography*. New York: Springer-Verlag, 2004.
[37] P. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Math. Computation*, pp. 243–264, 1987.
[38] D. J. Bernstein, "Batch binary Edwards," in *Proc. 29th Annu. Int. Cryptology Conf. Adv. Cryptol.*, 2009, pp. 317–336.
[39] F. Rodriguez-Henriquez, N. Saqib, and A. Díaz-Pérez, "A fast parallel implementation of elliptic curve point multiplication over $GF(2^m)$," *Microprocessors Microsyst.*, vol. 28, no. 5–6, pp. 329–339, 2004.
[40] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases," *Inf. Computing*, vol. 78, no. 3, pp. 171–177, 1988.
[41] "Xilinx Virtex-5 Device Data Sheet," Xilinx, ver. 5.0, Feb. 2009 [Online]. Available: www.xilinx.com/support/documentation/virtex-5.htm
[42] E. Brier and M. Joye, "Weierstraβ elliptic curves and side-channel attacks," in *Proc. Int. Conf. Practice Theory Public Key Cryptography*, 2002, pp. 183–194.
[43] B. Baldwin, R. Moloney, A. Byrne, G. McGuire, and W. P. Marnane, "A hardware analysis of twisted Edwards curves for an elliptic curve cryptosystem," in *Proc. 5th Int. Workshop Reconfigurable Computing: Architectures, Tools and Applications*, 2009, vol. 5453, pp. 355–361.
[44] E. Al-Daoud, R. Mahmod, M. Rushdan, and A. Kilicman, "A new addition formula for elliptic curves over $GF(2^m)$," *IEEE Trans. Comput.*, vol. 51, no. 8, pp. 972–975, Aug. 2002.

Reza Azarderakhsh (S'08) received the B.Sc. degree in electrical and electronic engineering from the Civil Aviation Technology College, Tehran, Iran, in 2002, and the M.Sc. degree in computer engineering (computer architecture) from Sharif University of Technology, Tehran, Iran, in 2005. He is currently working toward the Ph.D. degree in electrical and computer engineering at the University of Western Ontario, London, ON, Canada.

From 2004 to 2007, he was a Visiting Instructor with Civil Aviation Technology College, Tehran, Iran. In 2006, he won the TOPMED Scholarship and joined the Polytechnic University of Turin, Turin, Italy for a special double degree program in Electrical and Electronic Engineering with Sharif University of Technology. His current research interests include computer architecture, arithmetic of finite fields, cryptographic hardware, elliptic curve cryptography (ECC), and security of wireless networks.

Arash Reyhani-Masoleh (M'03) received the B.Sc. degree in electrical and electronic engineering from Iran University of Science and Technology, Tehran, Iran, in 1989, the M.Sc. degree in electrical and electronic engineering from the University of Tehran, Tehran, Iran, in 1991, both with the first rank, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2001.

From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology, Tehran, Iran. From June 2001 to September 2004, he was with the Centre for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, where he was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship in 2002. In October 2004, he joined the Department of Electrical and Computer Engineering, University of Western Ontario, London, ON, Canada, where he is currently a tenured Associate Professor. Currently, he serves as an associate editor for *Integration, the VLSI Journal*. His current research interests include algorithms and VLSI architectures for computations in finite fields, fault-tolerant computing, and efficient and reliable computations for cryptography and error-control coding.

Dr. Reyhani-Masoleh was the recipient of an NSERC Discovery Accelerator Supplement (DAS) in 2010. He is a member of the IEEE Computer Society.