High-Performance Implementation of Point Multiplication on Koblitz Curves

Reza Azarderakhsh and Arash Reyhani-Masoleh

***Abstract*—Fast and high-performance computation of finite-field arithmetic is crucial for elliptic curve cryptography (ECC) over binary extension fields. In this brief, we propose a highly parallel scheme to speed up the point multiplication for high-speed hardware implementation of ECC cryptoprocessor on Koblitz curves. We slightly modify the addition formulation in order to employ four parallel finite-field multipliers in the data flow. This reduces the latency of performing point addition and speeds up the overall point multiplication. To the best of our knowledge, the proposed data flow of point addition has the lowest latency in comparison to the counterparts available in the literature. To make the cryptoprocessor more efficient, we employ a low-complexity and efficient digit-level Gaussian normal basis multiplier to perform lower level finite-field multiplications. Finally, we have implemented our proposed architecture for point multiplication on an Altera Stratix II field-programmable gate array and obtained the results of timing and area.**

***Index Terms*—Cryptoprocessor, elliptic curve cryptography (ECC), field-programmable gate array (FPGA), Koblitz curves, parallel processing, point multiplication.**

I. Introduction

INFORMATION security in the networked environments aims to optimally use wide variety of cryptographic algorithms. These algorithms need to operate efficiently using minimal available resources. Elliptic curve cryptography (ECC) [2], [3] has been identified as an efficient method for public key cryptography. The efficiency of ECC implementations is based on point (or scalar) multiplication which is the most resource-consuming operation. Point multiplication is an operation of successively adding a point along an elliptic curve to itself. Binary Koblitz curves are special class of generic curves that point multiplication can be efficiently computed using their special properties. These curves employ Frobenius map (instead of doubling) and point addition operation for computing point multiplication. In the recent past, considerable efforts have been made to accelerate the computation of point multiplication over binary elliptic curves. Those include parallelization [4], [1] and interleaving [5], [6]. Parallelization is a well-known approach to accelerate the ECC computations, employing multiple parallel field arithmetic units (FAUs; mainly multipliers) in the finite-field computations. It is worth mentioning that, in the case of dependences among lower level computations, achieving parallelization is a challenging task, and employing more than certain number of parallel arithmetic units will not increase the speed of ECC computations. Recently, several methods to perform parallel computations for point addition on Koblitz curves have been proposed in [4] and [6]–[8]. It has been claimed that the maximum number of the finite-field multipliers to achieve the highest parallelization in computing point multiplication on Koblitz curves is three parallel finite-field multipliers [4]. However, in this brief, we slightly modify the point addition formulation in such a way to employ four multipliers to reduce the latency of point addition. This techniques will increase the overall speed of point multiplication on Koblitz curves. In this effect, we modify the point addition formulation to employ four parallel finite-field multipliers to reduce the latency of point multiplication from $4M + 13$ to $3M + 13$, where M is the latency (number of clock cycles) for a digit-level Gaussian normal basis (GNB) multiplication operation. Therefore, the number of multiplications in the critical path reduced from four to three. For investigating the practical performance of the proposed architecture, we implement it on a field-programmable gate array (FPGA) for different digit sizes over $GF(2^{163})$ targeting high-performance applications. It is noted that our method can be applied to any finite-field representation, and for the sake of efficient implementation and comparison, we use GNB in this brief.

Manuscript received September 7, 2012; accepted November 16, 2012. Date of publication January 14, 2013; date of current version March 11, 2013. This brief was recommended by Associate Editor Y. Ha.

R. Azarderakhsh is with the Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: razarder@uwaterloo.ca).

A. Reyhani-Masoleh is with the Department of Electrical and Computer Engineering, The University of Western Ontario, London, ON N6A 5B9, Canada (e-mail: areyhani@uwo.ca).

Digital Object Identifier 10.1109/TCSII.2012.2234916

II. Preliminaries

A. GNB

It is well known that, for any positive integer $m \geq 1$, there exists a normal basis of $GF(2^m)$ over $GF(2)$ [9]. Let $N = \{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$ 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 a field element, for example, $A = (a_0, a_1, \ldots, a_{m-1}) \in GF(2^m)$, is $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where coefficient $a_i \in GF(2)$. In normal basis, squaring can be achieved by simple right cyclic shift of coordinates of A, i.e., $A^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = (a_{m-1}, a_0, a_1, \ldots, a_{m-2})$. The multiplication of two elements A and B, $C = A \times B$, over GNB is based on a multiplication matrix $R_{(m-1)\times T}$ [10]. Let A and B be two field elements represented by GNB over $GF(2^m)$. Then, their product in $GF(2^m)$ can be obtained from [10]

$$c_0 = a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{j=1}^{T} b_{R(i,j)} \right) \tag{1}$$

where $R(i,j)$, $0 \le R(i,j) \le m-1$, $1 \le i \le m-1$, $1 \le j \le T$ denotes the (i,j)th element of $R_{(m-1)\times T}$ matrix. Note that, to obtain the lth coordinates of C, i.e., c_l, one needs to add "l mod m" to all indices in (1).

B. Arithmetic on Binary Elliptic Curves

A nonsupersingular binary generic 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:

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

where $a, b \in GF(2^m)$ and $b \neq 0$ [11]. For all $P \in E_{a,b}(GF(2^m))$, $P + \mathcal{O} = \mathcal{O} + P = P$. The negative of point $P = (x,y)$ is $-P = (x, x+y)$, where $(x,y) + (x, x+y) = \mathcal{O}$. The scalar k, $1 \le k \le r-1$ (r is the order of point P, $rP = \mathcal{O}$), is defined as an integer, and the point multiplication in elliptic curve is defined over the Abelian group as $Q = kP$, where P and Q are two points on $E_{a,b}(GF(2^m))$ and $k > 1$ is an integer. The point P is called the base point and Q is the result point. Binary elliptic 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]. In $GF(2^m)$, Frobenius map ϕ is an endomorphism that raises every element to its power of two, i.e., $\phi : x \to x^2$. Then, Frobenius endomorphism can be carried out efficiently (cost free) if the elements of finite field are represented in normal basis [11]. Koblitz showed that point doublings can be performed efficiently by utilizing the Frobenius endomorphism if the binary curve is defined over $GF(2)$ and $a \in \{0,1\}$. Then, the Frobenius map can be defined as $\phi : (x,y) \to (x^2, y^2)$. Then, if one represents the scalar k in τ-adic nonadjacent form (τNAF), i.e., $k = \sum_{i=0}^{l-1} k_i \tau^i$ for $k_i \in \{0, 1, -1\}$ and $k_i k_{i+1} = 0$, then point multiplication can be computed as $kP = \sum_{i=0}^{l-1} k_i \tau^i(P)$. It results in the Hamming weight of τNAF to be the same as the one of the binary non adjacent form (NAF), i.e., $\approx (\log_2 k)/3$, and its length to be approximately 2 m, which is twice the length of the binary NAF. Solinas [12] proposed a method and reduced the length of the τNAF over the remainder of k to m. Recently, efficient hardware architectures for τNAF conversion have been proposed in [13] and [14].

C. Point Addition on Koblitz Curves

Point addition on Koblitz curve can be performed using different coordinate systems such as, Jacobian, standard projective, and Lopez–Dahab projective coordinates. Among them, the Lopez–Dahab coordinate system provides efficient point addition formulation. For the Lopez–Dahab coordinates [15], the triple coordinates (X,Y,Z) are used to represent $(X/Z, Y/Z^2)$ in affine coordinates having $Z \neq 0$ and $\mathcal{O} = (1,0,0)$. The curve equation in this coordinate is $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$,, where $a, b \in GF(2^m)$, and the costs of point addition and doubling are $13\mathbf{M} + 4\mathbf{S} + 9\mathbf{A}$ and $5\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$, respectively. Note that $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{A}$ are the costs of multiplication, squaring, and addition, respectively. In the Lopez-Dahap coordinates where one of the points represented in affine, the cost of mixed projective point addition, i.e., $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2)$, reduces to $9\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$. The explicit formulation is given as follows [11]:

$$Z : \begin{cases} A = Y_1 + y_2 Z_1^2, \quad B = X_1 + x_2 Z_1 \\ C = BZ_1 \\ Z_3 = C^2 \end{cases}$$

$$X : \begin{cases} D = x_2 Z_3 \\ X_3 = A^2 + C(A + B^2 + aC) \end{cases}$$

$$Y : Y_3 = (D + X_3)(AC + Z_3) + (y_2 + x_2) Z_3^2 \tag{3}$$

where $a \in \{0,1\}$ for Koblitz curves, and hence, its cost reduces to $8\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$.

D. Point Multiplication on Koblitz Curves

Algorithm 1 Point multiplication on Koblitz curves using double-and-add-or-subtract algorithm [11].

Inputs: A point $P = (x,y) \in E_K(GF(2^m))$ on curve and integer k, $k = \sum_{i=0}^{l-1} k_i \tau^i$ for $k_i \in \{0, \pm 1\}$.

Output: $Q = kP$.

1: **initialize**
  a: **if** $k_{l-1} = 1$ then $Q \leftarrow (x, y, 1)$
  b: **if** $k_{l-1} = -1$ then $Q \leftarrow (x, x+y, 1)$
2: **for** i **from** $l-2$ **downto** 0 **do**
  $Q \leftarrow \phi(Q) = (X^2, Y^2, Z^2)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + k_i P = (X, Y, Z) \pm (x, y)$
  end if
end for
3: **return** $Q \leftarrow (X/Z, Y/Z^2)$

The algorithm for computing point multiplication, i.e., $Q = kP$, on Koblitz curves is given in Algorithm 1, where the scalar k is presented in τNAF [11]. This algorithm requires, on average, $m-1$ Frobenius maps and $m/3 - 1$ point additions or subtractions. Since the Frobenius maps can be computed with free squarings in normal basis, the computation of point addition determines the efficiency of point multiplication. Therefore, our main focus is on high-performance computation of point addition employing multiple efficient digit-level finite-field multipliers. In the following, we study the parallelization of point addition on Koblitz curves.

III. High-Speed Parallelization of Point Addition

Parallelization for hardware implementation of point addition on Koblitz curves has been considered recently employing different number of field multipliers in [4], [8], and [16]. In [4], it is shown that employing two finite-field multipliers reduces the number of multiplications (and, hence, the latency of ECC point multiplication) in the data path to five. Also, it is shown in [4] that the maximum number of parallel finite-field

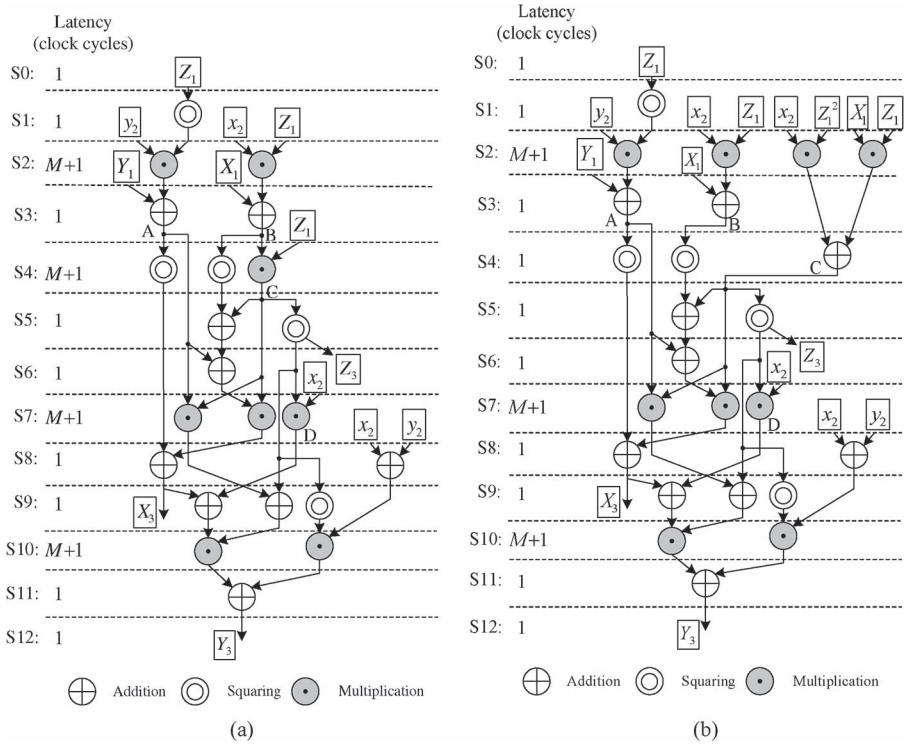

Fig. 1. Data dependence graph for parallel computation of point addition on Koblitz curves. (a) Using three finite-field multipliers adopted from [4]. (b) Proposed scheme employing four multipliers.

multipliers that can be employed to implement the fastest point multiplication is three. The data dependence graph for point addition employing three multipliers is depicted in Fig. 1(a) [4]. As one can see, the latency of point addition is $4M + 13$, where M is the latency for a multiplication. In Step S4, only one multiplier is operating, and the other two multipliers are idle. This is mainly because of the dependence of computing C to B [as shown in (3)]. This does not allow us to compute B and C in parallel. As can be seen from Fig. 1(a), a potential bottleneck occurs in computing C which uses only one multiplier in Step S4. This results in 66% multiplier utilization for the data dependence graph presented in Fig. 1(a) employing three parallel multipliers. The formulation of point addition can be modified to employ one additional parallel multiplication to reduce its latency, as stated in the following proposition.

Proposition 1: In computing the Z coordinate of the point addition formulation of (3), the data dependence in computing C can be eliminated by the following:

$$Z : \begin{cases} A = Y_1 + y_2 Z_1^2, & B = X_1 + x_2 Z_1 \\ C = x_2 Z_1^2 + X_1 Z_1, & Z_3 = C^2. \end{cases} \quad (4)$$

As one can see from (4), the computation of C can be performed in parallel with B at the cost of employing one more multiplier as compared to the formulation presented in (3). Therefore, we can employ four multipliers in parallel to compute point addition. The data dependence graph for computing point addition based on (4) is depicted in Fig. 1(b), which employs four parallel multipliers. As one can see in Step S2 in Fig. 1(b), four multipliers are operating in parallel. Therefore, the multiplication in Step S4 in Fig. 1(a) is eliminated. As can be seen in Fig. 1(b), the number of field multipliers in the data path is reduced to three multipliers with the overall latency of $3M + 13$ clock cycles.

TABLE I
COMPARISON OF THE COSTS OF PERFORMING POINT ADDITION IN THE MAIN LOOP ON KOBLITZ CURVES IN TERMS OF THE NUMBER OF MULTIPLICATIONS OVER $GF(2^{163})$, INCLUDING THE COST OF ADDITIONS AND SQUARINGS

Work	# of Multipliers (max)	Cost
E_K [4]	3	$(H(k)-1)\times(4M+13)$
This work	4	$(H(k)-1)\times(3M+13)$

A. Latency of Point Multiplication

The point multiplication on Koblitz curves needs to perform τNAF conversion, the main computation (addition and Frobenius map), and the coordinate conversion. In [14], an efficient circuitry is presented for τNAF conversion which requires $m + 6$ clock cycles for $m = 163$. Also, the latency of coordinate conversion from projective Lopez–Dahab to affine is $11M + 11$ based on the Itoh–Tsujii method [17]. Since these latencies are fixed for all implementations, we only compare the latency for the main processor in computing point additions, as given in Table I. We assume that four multipliers, two adders, and three squarers are available based on the data dependence graph depicted in Fig. 1(b) (one extra squarer is employed to perform Frobenius maps of all three coordinates, i.e., X_3, Y_3, and Z_3 in parallel). In this table, $H(k)$ is the Hamming weight of τNAF expansion of k, and the latency of the main loop is computed by multiplying the number of nonzero terms in k to the latency of a point addition. We assume that the point additions and point subtractions are divided evenly and 163 clock

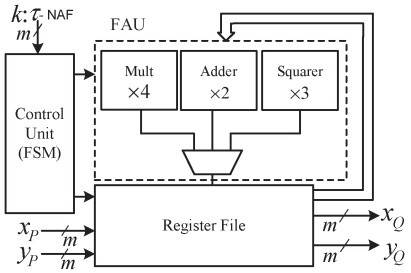

Fig. 2. Architecture of point multiplication cryptoprocessor.

cycles are required to perform Frobenius maps. Therefore, the total latency of point multiplication over $GF(2^{163})$ (without considering τNAF conversion) is computed as $(H(k)-1)\times(3M+13)+163+11M+11$, where $(H(k)-1)\times(3M+13)$ is the latency of computing point addition in the main loop.

IV. Proposed Cryptoprocessor for Point Multiplication

The architecture of the cryptoprocessor is depicted in Fig. 2 and is explained in the following.

A. FAU

The FAU performs three basic arithmetic operations employing four digit-level GNB multipliers, two $GF(2^m)$ adders, and two squarers. Multiplication in $GF(2^m)$ plays the main role in determining the efficiency of the point multiplication. We employ a low-complexity digit-level parallel-in parallel-out GNB multiplier presented in [1] and efficiently pipelined it in this work. This architecture, which implements (1), is optimized for even values of T, $T>2$ GNB [18]. In a digit-level parallel-in parallel-out GNB multiplier, the results will be available in parallel after $q=\lceil m/d\rceil$ clock cycles. Therefore, as the pipelining adds one extra clock cycle to the latency of multiplication, the latency of the multiplier is given by $M=\lceil m/d\rceil+1, 1\le d\le m$. For the given field size $m=163$ (which is type 4 GNB), digit size d is chosen in such a way to reduce the latency while increasing d. Therefore, we choose the digit sizes from the set $d=\{11,21,33,41,55\}$ for $m=163$. The time complexity of the employed digit-level GNB multiplier is $T_A+(\lceil\log_2 T\rceil+\lceil\log_2(d+1)\rceil)T_X$, and its area complexity is dm AND gates and $\le(d(m-1)/2)(T-1)+dm$ XORs, which is further reduced by a complexity reduction algorithm [1] to $n_p+v_p((T/2)-1)+dm$ XOR gates, where n_p, $n_p\leqslant\min\{(v_pT/2),\binom{m}{2}\}$ and $v_p=d(m-1)/2$. Note that the exact value of n_p can be obtained after employing common subexpression sharing and depends on the digit size d[1]. Also, the pipelined multiplier requires $(3+\ell)m$ registers, and its critical-path delay is $\max\{(T_A+(\lceil\log_2 T\rceil+\lceil\log_2 K\rceil)T_X),(\lceil\log_2(\ell+1)\rceil T_X)\}$, where ℓ is the level of accumulation and $K=\lceil d/\ell\rceil$. The $GF(2^m)$ adder uses m XOR gates to perform the addition and requires only a clock cycle to store the results in the registers. The squarer is simple rewiring in normal basis and requires a clock cycle to store its results in the registers. Note that the Frobenius map is performed for coordinates of X, Y, and Z independently.

TABLE II
Implementation Results of the Point Multiplication on Koblitz Curves on the Stratix II EP2S180F1020C3 Device

d	$M=q+1$	**Latency**[1] (L_{Total})	$f_{\max}$ (MHz)	Area (ALMs)	**P.M. Time** [μs]
11	16	3601	198.21	7,092	18.18
21	9	2405	195.33	11,848	12.33
33	6	1892	192.50	18,964	9.85
41	5	1721	188.71	23,084	9.15
55	4	1550	165.06	30,564	9.39

1. $L_{Total}=(H(k)-1)(3M+13)+163+11M+11$, $H(k)\approx 163/3\approx 54.33$.

B. Control Unit and the Register File

The control unit is designed with a finite-state machine (FSM) to perform the point multiplication with other units. First, the coordinates of P, i.e., (x_P,y_P), are loaded to the registers. Once k is available in the τNAF representation, at the input of the control unit, the FAU starts the computations based on the FSM stored in the control unit. The final and intermediate results are stored in the registers. The data bus width is set to 163 bits. The projective coordinates of $Q=kP$, i.e., (X,Y,Z), are converted to the affine coordinates of Q, i.e., $(x_Q,y_Q)=(X/Z,Y/Z^2)$, using one of the available multipliers and a squarer based on the Itoh–Tsujii's scheme [17] instead of using dedicated hardware. This resulted in saving in the entire area of the cryptoprocessor.

V. FPGA Implementations and Comparisons

FPGAs have advantages for prototyping and the proof of concepts. To have a fair comparison with previous works, we have selected the Altera Stratix II EP2S180F1020C3 device as the target FPGA for our implementations. The presented architecture for point multiplication of the cryptoprocessor presented in Section IV is coded in the Very high speed integrated circuit Hardware Description Language and synthesized for different digit sizes d, $d\in\{11,21,33,41,55\}$, for the Koblitz curve defined over $GF(2^{163})$.

We use the Altera Quartus II version 11 design software for our implementations. The results of the area and maximum clock frequencies of the implementations after the place and route are reported in Table II. As one can see, increasing the digit size results in the reduction of the latency of the point multiplication, i.e., $L_{\rm Total}$, at the cost of the increase in the area and decrease in the operating clock frequency. The point multiplication time is provided by dividing the total number of clock cycles ($L_{\rm Total}$) by the maximum operating clock frequency ($f_{\max}$). The pipelined (one level) digit-level GNB multiplier adds one clock cycle to the latency of multiplier, as can be seen in the second column of Table II (i.e., $M=q+1$). The latency of loading the operands to the multipliers is counted in the total latency, as shown in the data dependence graph illustrated in Fig. 1. The total latency of point multiplication is computed as $L_{\rm Total}=(H(k)-1)(3M+13)+163+11M+11$, $H(k)\approx 163/3\approx 54.33$ (without considering τ-adic conversion). Note that the fastest computation of point multiplication is obtained for $d=41$, which is 9.15 μs, employing 23 084 adaptive logic module (ALM) including control unit. In Table III, the best results in terms of time

TABLE III
COMPARISON OF RELATED WORKS FOR FPGA IMPLEMENTATIONS OF PARALLEL POINT MULTIPLICATION ON KOBLITZ CURVES OVER $GF(2^{163})$ USING DIGIT-LEVEL FINITE-FIELD MULTIPLIERS. NOTE THAT ALL OF THE ARCHITECTURES ARE IMPLEMENTED ON ALTERA STRATIX II TO HAVE FAIR COMPARISONS

Work	τ-adic Conv.	Basis	# of Mults.	d	Total Latency	$f_{\max}$	Area (ALMs)	Time [μs]	Area-Time (A×T)
[19]	NO	NB	4	24	2033	152.28	26,381	13.38	0.35
[4]	NO	NB	3	33	4248	146.71	22,416	28.95	0.64
[6]	NO	NB	4	17	1540	162.42	23,580	9.48	0.22
[6]	NO	NB	4	19	1422	164.42	25,366	8.64	0.22
This work Fig. 1b	NO	GNB	4	33	1892	192.50	18,964	9.85	0.18
This work Fig. 1b	NO	GNB	4	41	1721	188.71	23,084	9.15	0.21

It should be noted that the works presented in [19] and [4], reported their results with and without τ-adic conversion. Therefore, to have a fair comparison, we only compare the results with the ones without τ-adic conversion.

and area are summarized for point multiplications on Koblitz curve over $GF(2^{163})$, i.e., the National Institute of Standards and Technology (NIST) K-163. The latency of the proposed architecture for point addition is less than the counterparts proposed in [4] and [19], as shown in Table III. Our proposed scheme results in faster time of point multiplication in comparison to the one proposed in [4]. In [6] and [5], a new scheme known as interleaving is proposed to reduce the latency of point addition on Koblitz curves. This scheme reduces the latency of point addition about 50% of the one proposed in [4] employing four finite-field multipliers. It is worth mentioning that one can improve the timing results for this scheme employing the digit-level multiplier employed in this work. The work presented in [6] computes a point multiplication in 9.48 μs, employing four multipliers with $d = 17$ which occupies 23 580 ALMs. Also, with choosing $d = 19$, the time of point multiplication reduces to 8.64 μs occupying 25 366 ALMs. However, our proposed scheme with $d = 33$ occupies 18 964 ALMs and computes a point multiplication in 9.85 μs. In this brief, the implementation results highlight the fact that the finite-field multiplier employed here (adopted from [20] and [1]) outperforms the ones used in other works available in the literature. In comparison to the scheme proposed in [6], the following can be summarized: 1) The digit-level GNB multiplier employed in this work (adopted from [20] and [1]) operates at higher clock frequencies, and 2) in terms of time–area product which is mentioned by $(A \times T)$ in Table III, our proposed architecture is more efficient or favorably comparable with the counterparts.

VI. CONCLUSION

We have proposed a new fast data flow graph for the point addition formulation using the Lopez–Dahab mixed coordinates employing four parallel multipliers on Koblitz curves. It is shown that the data flow graph has three multipliers in its critical path as compared to four multipliers for the best scheme available in the literature. We have employed a low-complexity digit-level GNB multiplier (proposed in [1]) to perform finite-field multiplications. The analysis results show that our method results in smaller latencies in computing point addition. Moreover, the implementation results on Altera Stratix II indicate that our parallel multipliers operate at higher clock frequencies and the point multiplication results are more efficient in terms of time–area product and are favorably comparable with the ones available in the literature.

REFERENCES

[1] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1453–1466, Aug. 2012.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. CRYPTO 1985*, 1986, pp. 417–426.

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

[4] 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.

[5] K. Järvinen, "Optimized FPGA-based elliptic curve cryptography processor for high-speed applications," *Integr., VLSI J.*, vol. 44, no. 4, pp. 270–279, Sep. 2011.

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

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

[8] J. Adikari, V. S. Dimitrov, and R. J. Cintra, "A new algorithm for double scalar multiplication over Koblitz curves," in *Proc. IEEE ISCAS*, 2011, pp. 709–712.

[9] A. J. Menezes, I. F. Blake, S. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. Boston, MA: Kluwer, 1993.

[10] 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.

[11] D. R. Hankerson, S. A. Vanstone, and A. J. Menezes, *Guide to Elliptic Curve Cryptography*. New York: Springer-Verlag, 2004.

[12] J. A. Solinas, "Efficient arithmetic on Koblitz curves," *Des., Codes Cryptograph.*, vol. 19, no. 2/3, pp. 195–249, Mar. 2000.

[13] B. B. Brumley and K. U. Järvinen, "Conversion algorithms and implementations for Koblitz curve cryptography," *IEEE Trans. Comput.*, vol. 59, no. 1, pp. 81–92, Jan. 2010.

[14] J. Adikari, V. Dimitrov, and K. Jarvinen, "A fast hardware architecture for integer to τ-NAF conversion for Koblitz curves," *IEEE Trans. Comput.*, vol. 61, no. 5, pp. 732–737, May 2012.

[15] J. López and R. Dahab, "Fast multiplication on elliptic curves over GF(2m) without precomputation," in *Proc. Workshop CHES*, 1999, pp. 316–327.

[16] O. Ahmadi, D. Hankerson, and F. Rodríguez-Henríquez, "Parallel formulations of scalar multiplication on Koblitz curves," *J. Univ. Comput. Sci.*, vol. 14, no. 3, pp. 481–504, 2008.

[17] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases," *Inf. Comput.*, vol. 78, no. 3, pp. 171–177, Sep. 1988.

[18] *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Std. 1363-2000, Jan. 2000.

[19] V. S. Dimitrov, K. U. Järvinen, M. J. Jacobson, 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.

[20] R. Azarderakhsh and A. Reyhani-Masoleh, "A modified low complexity digit-level Gaussian normal basis multiplier," in *Proc. 3rd Int. WAIFI*, M. A. Hasan and T. Helleseth, Eds., Jun. 2010, vol. 6087, pp. 25–40.