JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.16, NO.1, FEBRUARY, 2016
http://dx.doi.org/10.5573/JSTS.2016.16.1.118

ISSN(Print) 1598-1657
ISSN(Online) 2233-4866

Efficient Algorithm and Architecture for Elliptic Curve Cryptographic Processor

Tuy Tan Nguyen and Hanho Lee

***Abstract*—This paper presents a new high-efficient algorithm and architecture for an elliptic curve cryptographic processor. To reduce the computational complexity, novel modified Lopez-Dahab scalar point multiplication and left-to-right algorithms are proposed for point multiplication operation. Moreover, bit-serial Galois-field multiplication is used in order to decrease hardware complexity. The field multiplication operations are performed in parallel to improve system latency. As a result, our approach can reduce hardware costs, while the total time required for point multiplication is kept to a reasonable amount. The results on a Xilinx Virtex-5, Virtex-7 FPGAs and VLSI implementation show that the proposed architecture has less hardware complexity, number of clock cycles and higher efficiency than the previous works.**

***Index Terms*—Elliptic curve cryptography, point multiplication, bit-serial, architecture**

I. INTRODUCTION

Elliptic curve cryptography (ECC), an approach based on the algebraic structure of elliptic curves over finite-fields, is a public key cryptography that has attracted great attention in recent years. It offers security similar to traditional systems, such as Rivest, Shamir, & Adleman (RSA), but with significantly smaller key lengths [1]. For example, 163-bit ECC is considered equivalent to 1024-bit RSA [1, 2]. Thanks to this advantage, ECC can be implemented to strictly consider resource limitations.

Manuscript received Sep. 7, 2015; accepted Dec. 24, 2015
Dept. of Information and Communication Engr., Inha University, Incheon, 22212, Korea
E-mail : hhlee@inha.ac.kr

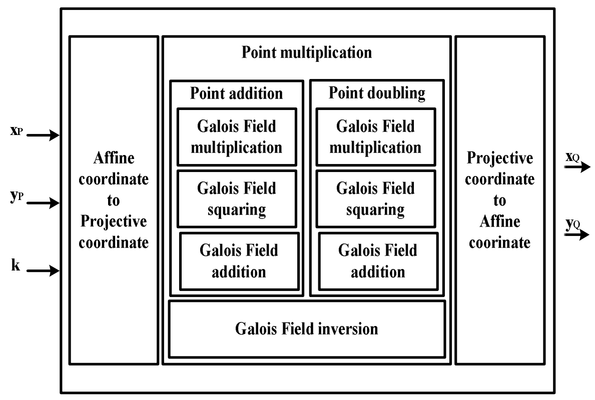

Fig. 1. Overview architecture for ECC point multiplication.

An elliptic curve E over a Galois field (GF) is the set of solutions to Eq. (1). A point $P(x_P, y_P)$ is a pair of elements that satisfies

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

The underlying operation in ECCs is scalar point multiplication, $Q = kP$, the multiplication of an elliptic curve point P by a scalar k to give the resultant point Q [3, 4]. To perform this operation, the point addition and point doubling operations are combined effectively.

Three main steps to calculate the point multiplication are: 1) convert point P from affine coordinate to project coordinate; 2) compute $Q = kP$ as projective coordinate; and 3) convert point Q from projective coordinate to affine coordinate. Fig. 1 describes the main operations to perform ECC point multiplication.

Recently, several field-programmable gate array (FPGA)-based ECC processors and hardware accelerators have been presented in the literature [5, 7]. In these papers, the authors showed various techniques to improve the performance of the ECC operation. The typical goal is to reduce the latency of ECC point multiplication in terms of the number of required clock cycles, and reduce the hardware complexity.

In this paper, we propose a novel idea for an ECC algorithm and architecture in projective coordinates to reduce hardware complexity and keep latency to a reasonable amount. Specifically, we use the bit-serial multiplication concept to reduce hardware complexity. Moreover, a delay-efficient architecture, in which the field multiplication operations and the field squaring operations are performed in parallel, is proposed in order to balance hardware complexity and latency efficiency.

The rest of this paper is organized as follows. In Section II, we provide some background information, including finite-field and ECC point multiplication algorithms. The proposed algorithm is presented in Section III. Section IV presents the proposed ECC point multiplication architecture and the novel design techniques. In Section V, the results and a performance comparison are presented. Finally, a conclusion is provided in Section V.

II. BACKGROUND

1. Finite-field Arithmetic

Finite-fields, which are defined as a set of elements denoted as $GF(2^m)$, play an important role in the calculations for cryptography systems. The operations in the finite-field include multiplication, squaring, addition, subtraction and inversion. Addition and subtraction are implemented simply by using XOR addition of the two m-bit operands [8]. The field squaring operation over $GF(2^m)$ can be implemented based on the approach presented by Rodríguez-Henríquez et al. [6]. To calculate the field inversion, polynomial greatest common divisor (GCD) and inverse computation [9] are used. Among these arithmetic operations, multiplication is the most important finite field arithmetic operation [11] because of its time consuming and large hardware requirement.

Two elements, $a(x)$ and $b(x)$, in $GF(2^m)$ can be expressed as:

$$\begin{aligned} a(x) &= a_0 + a_1x + ... + a_{m-1}x^{m-1} \\ b(x) &= b_0 + b_1x + ... + b_{m-1}x^{m-1} \end{aligned} \tag{2}$$

The product $c(x)$ can be computed as follows:

$$\begin{aligned} c(x) &= a(x)b(x) \bmod f(x) \\ c(x) &= \left(\sum_{i=0}^{m-1} b_i a(x)x^i \right) \bmod f(x) \end{aligned} \tag{3}$$

where $f(x) = x^m + f_{m-1}x^{m-1} + ... + f_1x + f_0$

Two popular field multiplication approaches to implement GF multiplication are digit-serial multiplication [8] and bit-serial multiplication [10]. The former is used with the goal of reducing latency in terms of the number of required clock cycles for one field multiplication operation. However, this approach requires a lot of hardware. Consequently, it is not suitable for resource-constrained systems. In order to reduce hardware complexity, bit-serial multiplication was introduced. In this approach, the operand $b(x)$ is processed from its least significant bit (LSB), and one bit at each cycle is considered. The product is the modulo of the sum $\sum_{i=0}^{m-1} b_i a(x)x^i$ and the irreducible polynomial $f(x)$.

2. ECC Point Multiplication Algorithm

There are several different algorithms for performing elliptic curve point multiplication [4, 12]. The three most used algorithms are the following: double and add; nonadjacent form (NAF) addition-subtraction chain; and Montgomery ladder product [1]. Among these algorithms, the Montgomery ladder method for elliptic curve scalar point multiplication is the most popular. It can be used not only in affine coordinates but also in projective coordinates. Given a point P on elliptic curve E, and an m-bit scalar $k = k_0 2^0 + k_1 2^1 + ... + k_{m-1}2^{m-1}$, the scalar product kP is defined by $0P = 0$, $1P = P$, $2P = P + P$, and so on.

Algorithm 1, a Lopez-Dahab (LD) algorithm, computes scalar point multiplication kP from point $P(x_P, y_P)$, which is on the curve. The neutral element is the point at infinity, and the doubling and adding operations are the corresponding curve operations. In the case of curve $y^2 + xy = x^3 + ax^2 + b$ over $GF(2^m)$, the following

Algorithm 1. LD scalar point multiplication [1]

```
Input: $k = (k_{m-1}...k_1k_0)$, $P(x_P, y_P)$
Output: $Q = kP$
 1. $A = 0$; $B = P$;
 2. for $i = 1$ to $i = m$ do
      if( $k_{m-1} = 0$)
         $B = A + B$;
         $A = 2A$;
      else
         $A = A + B$;
         $B = 2B$;
      end if
    end for
 3. Return $Q = A$;
```

property holds true during the execution of Algorithm 1 [3, 4]:

$A(x_A, y_A)$ and $B(x_B, y_B)$ are two different points on the curve. The x-coordinates $A + B$, x_{A+B} and x_{A+A}, are related by the following formulas [1].

$$x_{A+B} = x_P + x_B(x_A + x_B)^{-1} + (x_B(x_A + x_B)^{-1})^2 \quad (4)$$

$$x_{A+A} = x_A^2 + b/x_A^2 \quad (5)$$

ECC can be implemented using either affine coordinate or projective coordinate. However, the number of field inversions can be reduced in projective coordinate. Therefore, this coordinate is widely used such as in [1], [3, 4] and [8]. Assuming that standard projective coordinates are used, x_A and x_B can be expressed by using the forms $x_A = X_A/Z_A$, $x_B = X_B/Z_B$. The corresponding formulas for ECC point addition and point doubling operation in projective coordinate can be found in [1]:

$$Z_{A+B} = (X_AZ_B + X_BZ_A)^2 \quad (6)$$

$$X_{A+B} = x_PZ_{A+B} + X_AX_BZ_AZ_B \quad (7)$$

$$Z_{A+A} = X_A^2Z_A^2 \quad (8)$$

$$X_{A+A} = X_A^4 + bZ_A^4 \quad (9)$$

Left-to-right scalar point multiplication or a binary algorithm was presented by Li et al. [13]. Doubling and addition can be classified into three types: addition, doubling after addition, and doubling after doubling. Using this algorithm, the scalar k is represented in binary. The algorithm iterates through each bit of k. If the particular bit of k is '1', then a point addition is also performed. The run-time of the algorithm depends on the Hamming weight of k.

III. Modified Lopez-Dahab and Left-To-Right Algorithm

In this section, we propose a novel modified LD and a left-to-right algorithm to perform the ECC point multiplication, as shown in Algorithm 3. There are four main steps. The first one is converting point P from affine coordinate to projective coordinate. Two important steps are performing the ECC operation as projective coordinate and the modification step to get the correct result, compared with the original LD algorithm. The final step is converting from projective coordinate to affine coordinate and returning the result point, Q.

The proposed modified LD and the left-to-right algorithm show the advantages to computing ECC point multiplication by reducing the number of addition operations. Step 1 of the proposed algorithm shows the conversion from affine coordinate to projective coordinate. Initially, the assignments $X_2 = x_P$ and $Z_2 =$ '1' are performed. The initial values of X_1 and Z_1 depend on the value of the LSB of the key k_0. If $k_0 =$ '1', X_1 and Z_1 are assigned to x_P and '1', respectively. Otherwise, both X_1 and Z_1 are assigned to '0'. Based on the value of the key bit k_i, either the ECC addition or ECC doubling in the projective will be performed. The advantage of this assignment strategy is reducing the number of redundant point addition operations in case the value of the key bit k_i is '0'. Therefore, the time required for point multiplication is reduced significantly.

As can be seen from step 2 of the algorithm, whenever the key bit is equal to '0', the algorithm only executes the point doubling operation, which consists of four field squaring operations ($S_1 = X_2^2$, $S_2 = S_1^2$, $S_3 = Z_2^2$ and $S_4 = S_3^2$), one field addition ($X_2 = S_2 + S_4$) and one field multiplication ($Z_2 = S_1S_3$). Otherwise, the point addition operation, which requires four field multiplication operations ($T_1 = X_1Z_2$, $T_2 = X_2Z_1$, $T_3 = x_PZ_1$ and $T_4 = T_1T_2$), two field addition operations ($T_1 + T_2$ and $T_3 + T_4$), and one field squaring operation ($(T_1 + T_2)^2$), are calculated after finishing the point doubling. After finishing all iterations, there is a difference between the value for X_1 of proposed algorithm and the traditional value obtained from the LD algorithm. Therefore, a modification

Algorithm 2. Left-to-right Algorithm [13]

```
Input: $k = (k_{m-1}\ldots k_1k_0)$, $P(x_P, y_P)$
Output: $Q = kP$
1. $Q=\infty$
2. for $i = m - 1$ 0 do
   2.1 $Q = 2P$
   2.2 if ($k_i = 1$)
       $Q = Q + P$
3. Return (Q)
```

operation is required to return the same value with the result obtained from the traditional projective coordinate version of LD scalar point multiplication.

Because the initial values of point (X_1, Z_1) and (X_2, Z_2) of proposed algorithm are different from that in original LD algorithm, there is a constant gap of $2P$ between the result of our algorithm and the original LD one. A value of (X_3, Z_3) = $2P$ is assigned and added to the final result of step 2 of proposed algorithm to get the same result with original LD algorithm. The conversion from projective coordinate to affine coordinate follows the modification steps, resulting in the value of point Q.

Compared with conventional LD scalar point multiplication, depending on the number of '0' bits and '1' bits in the key, the proposed algorithm can significantly reduce the number of point addition operations. Assume the number of '0' bits in a key with a length of m bits is n, the number of point addition operations in projective coordinates can be reduced by n. Explained another way, the proposed method can reduce the number of field calculations, in which field multiplication requires a large number of clock cycles. Therefore, the proposed algorithm can appreciably reduce latency, as well as hardware requirements, to perform the whole ECC point multiplication.

Table 1 shows the number of field multiplication and field squaring calculations with the conventional LD algorithm and the proposed modified LD algorithm. Assume that there are n '0' bits in the key k. To perform ECC point multiplication, the traditional algorithm, such as that in Deschamps et al. [14], requires $6m – 6$ and $7m – 7$ field multiplications and field squarings, respectively. In the proposed algorithm, when the number of '0' bits in the key increases by one, the number of field multiplications is reduced by four, and the number of

Algorithm 3. Modified LD and Left-to-Right algorithm

```
Input: $k = (k_{m-1}\ldots k_1k_0)$, $P(x_P, y_P)$
Output: $Q = kP$
1. From affine coordinate to projective coordinate:
   $X_2 = x_P$, $Z_2 = 1$
   $X_3 = x_P^4 + b$, $Z_3 = x_P^2$
   if ($k_0 = 1$)
       $X_1 = x_P$, $Z_1 = 1$
   else
       $X_1 = 0$, $Z_1 = 0$
   end if
2. Calculate in projective coordinate:
   2.1 for $i = 1$ to $i = m – 1$ do
       $S_1 = X_2^2$
       $S_2 = S_1^2$
       $S_3 = Z_2^2$
       $S_4 = S_3^2$
       $X_2 = S_2 + S_4$
       $Z_2 = M_1(S_1, S_3)$
   2.2 if($k_i = 1$)
       $T_1 = M_2(X_1, Z_2)$
       $T_2 = M_3(X_2, Z_1)$
       $Z_1 = (T_1 + T_2)^2$
       $T_3 = M_4(x_p, Z_1)$
       $T_4 = M_5(T_1, T_2)$
       $X_1 = T_3 + T_4$
       end if
    end for
3. Adjust the result:
   $T_1 = M_2(X_3, Z_2)$
   $T_2 = M_3(X_2, Z_3)$
   $Z_1 = (T_1 + T_2)^2$
   $T_3 = M_4(x_p, Z_3)$
   $T_4 = M_5(T_1, T_2)$
   $X_1 = T_3 + T_4$;
4. Convert from projective to affine coordinate:
   if ($Z_2 = 0$)
      $X_1 = x_P$
      $Z_1 = x_P + y_P$
   else
      $X_1 = X_1/Z_1$
      $X_2 = X_2/Z_2$
      $T_2 = M_2(X_1 + x_P, X_2 + x_P)$
      $T_3 = M_3(X_1 + x_P, x_P^{-1})$
      $T_4 = T_2 + x_P^2 + y_P$
      $T_2 = M_2(T_3, T_4)$
      $Z_1 = T_2 + y_P$;
   end if
5. Return output value
   $x_Q = X_1$, $y_Q = Z_1$
```

field squarings is reduced by one. Therefore, the proposed algorithm can reduce the number of field calculations, resulting in reduced hardware complexity and latency. Generally, with n '0' bits in the key, the

Table 1. Comparison for required field multiplication and field squaring to calculate the ECC point multiplication

No. of "0" bits in the key	Multiplication		Squaring	
	Proposed	[8]	Proposed	[8]
1	*6m - 10*	*6m - 6*	*7m – 8*	*7m – 7*
2	*6m – 14*	*6m - 6*	*7m – 9*	*7m – 7*
…	…	…	…	…
n	*6m – 6 – 4n*	*6m - 6*	*7m – 7 – n*	*7m – 7*
…	…	…	…	…
m	*2m –36*	*6m - 6*	*6m - 6*	*7m – 7*

proposed algorithm can reduce calculations by $\frac{4n}{6m-6}$ field multiplications and $\frac{n}{7m-7}$ field squarings, compared to the number of field multiplications and field squarings by Mahdizadeh and Masoumi [8].

IV. Proposed ECC Point Multiplication Architecture

For the design of the ECC point multiplication architecture, we considered two parts. The first one involves calculations for converting between affine coordinate and projective coordinate; the other involves calculations in the projective coordinate system. Fig. 2 shows the proposed ECC processor architecture, in which the bit-serial multiplier in Fig. 3 is used.

To balance system latency and hardware cost, the number of computation units is selected in a way that allows computing the field operations simultaneously. In the proposed architecture, we use five field multipliers to perform the main loop. Step 2.1 of the proposed LD algorithm is the point doubling operation in projective coordinates. In the proposed architecture, two squaring operations, $S_1 = X_2^2$ and $S_3 = Z_2^2$, are performed in parallel by using squarer 1 and squarer 3. After that, $S_2 = S_1^2$ and $S_4 = S_3^2$ are also accomplished at the same time using squarer 2 and squarer 4. The results from squarer 1 and squarer 3 are fed to the multiplier 5 to calculate the multiplication $Z_2 = S_1S_3$, while the results from squarer 2 and squarer 4 are driven to the 163-bit XOR gate array to obtain X_2. After this step, the new value of X_2 and Z_2 are driven out. As mentioned, the most important module in the design of ECC point multiplication is the field multiplier. The number of clock cycles to calculate the field multiplication using the bit-serial approach is larger than the number of clock cycles using the digit-serial one. However, bit-serial multiplication requires less hardware than digit-serial multiplication. We use bit-serial multiplication and control the field operations so that the hardware cost and latency remain balanced. As shown in Fig. 2, multiplier 1 and multiplier 2 perform the calculations $T_1 = X_1Z_2$ and $T_2 = X_2Z_1$. The operations of multiplier 1 and multiplier 2 are accomplished simultaneously. After the signal *m_done*1 and *m_done*2

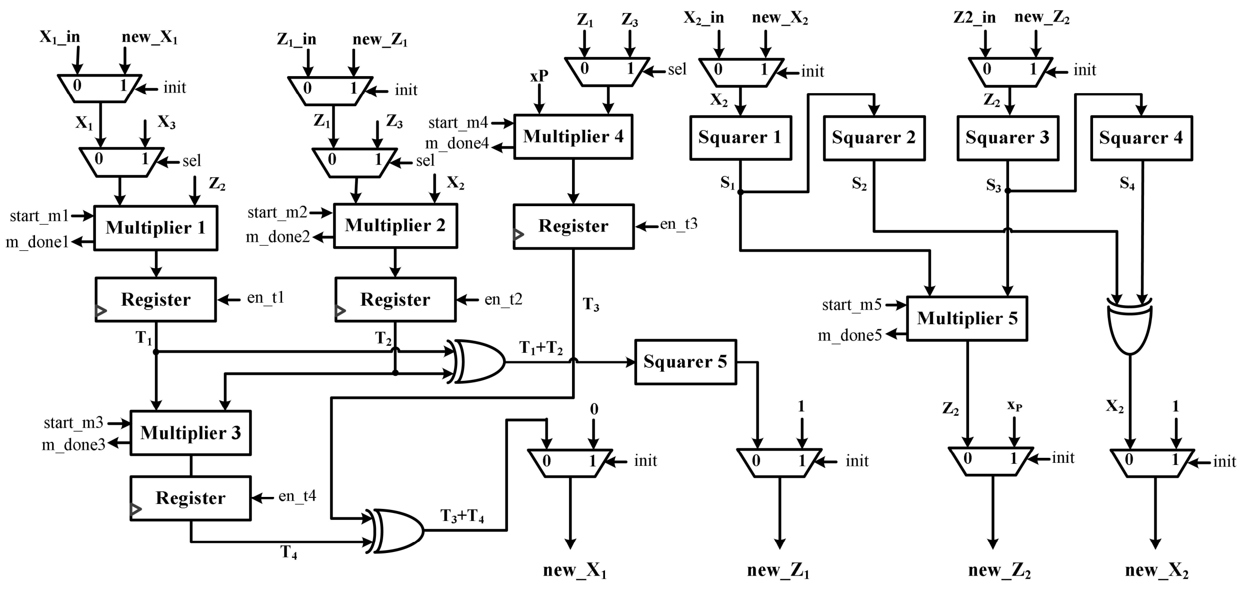

Fig. 2. Proposed architecture for ECC point multiplication.

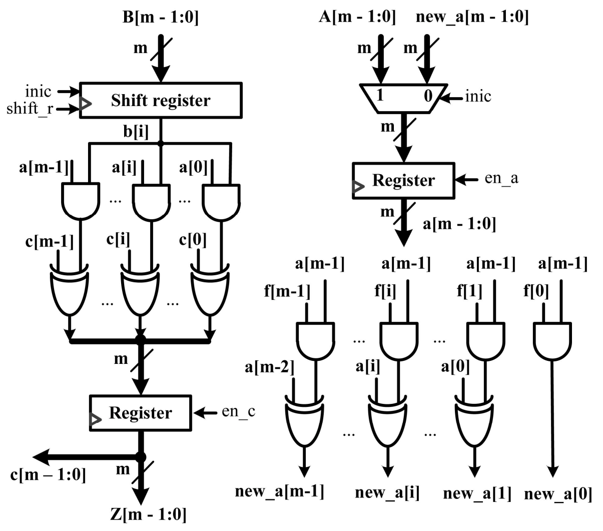

Fig. 3. Bit-serial multiplier in $GF(2^m)$ [14].

are enabled, multiplier 3 and multiplier 4 start performing the calculations $T_3 = x_P Z_1$ and $T_4 = T_1 T_2$ in parallel. In this way, the delay of the proposed algorithm is reduced twice each iteration. The outputs from multiplier 3 and multiplier 4 are put into the 163-bit XOR gate, where output value is either sent to the MUX or squarer 5 to get the new value of X_1 and Z_1. After completing all iterations, the modification operation is executed. The strategy in the design of the modification is for the hardware cost to be reduced as much as possible. Hence, we reuse the multipliers in the main loop of the algorithm to execute part 3 of the proposed algorithm. The conversion from projective coordinate to affine coordinate follows the modification step.

V. RESULTS AND COMPARISON

In this section, we compare the performance of different hardware implementations for point multiplication, which is the main operation for ECC. We used Verilog HDL to synthesize our design. Table 2 shows the comparison between bit-serial multiplication and digit-serial multiplication using a Xilinx Virtex-5 FPGA. Multiplication using bit-serial multiplication requires 566 flip-flops (FFs) and look-up tables (LUTs), while the digit-serial approach, depending on digit size, requires more hardware than the bit-serial approach. It is certain that when the digit size increases, the number of required LUTs to perform the field multiplication operation increases rapidly. From Table 2, the number of

Table 2. Performance comparison of bit-serial and digit-serial multiplications over $GF(2^{163})$ in a Virtex-5 FPGA

	Bit-serial	Digit-serial [1]	
		digit size=4	*digit size=8*
FFs	*566*	*497*	*496*
LUTs	*566*	*669*	*1,001*
# of clock cycles	*163*	*41*	*21*
Time (ns)	*82*	*73*	*56*
AreaxTime (A x T)	*46.4*	*48.8*	*56.1*

Table 3. Performance comparison of ECC processors in FPGAs

	Proposed		[1]	[8]
Device	*Virtex-7*	*Virtex-5*	*Virtex-5*	*Virtex-4*
Mult. operation	Bit-serial	Bit-serial	Bit-serial	Digit-serial
Slices	*4,665*	*4,815*	-	*14,203*
LUTs	*3,806*	*4,807*	*5,100*	*26,557*
Freq. (MHz)	*800*	*550*	*550*	*263*
Throughput (Mbps)	*2.51*	*1.72*	*1.63*	*12.5*
# of clock cycles	*52,012*	*52,012*	*54,943*	*3,404*
Time (μs)	*65.0*	*94.6*	*98.1*	*11.6*
Efficiency	*657*	*358*	*313*	*471*

LUTs based on a digit size equal to 8 is about twice as many as when the digit size is 4.

Table 3 shows the hardware complexity and efficiency comparison for the proposed architecture using the modified LD, the left-to-right algorithm, and its predecessors. The hardware cost can be defined as the number of LUTs and slices.

To evaluate the efficiency of the design, we used the same parameters defined by Mahdizadeh and Masoumi [8].

$$Efficiency = \frac{Throughput}{Area} \frac{(Mbps)}{(Slices)}$$

where

$$Throughput = \frac{Working\ frequency\ x\ Number\ of\ bits}{Number\ of\ cycles}$$

The algorithmic efficiency shown in the last row of Table 3 is defined as throughput divided by area. However, the previous designs [1, 8] did not show the slice count. Therefore, we used throughput divided by the number of LUTs to calculate the efficiency.

As can be seen from Table 3, to perform point

Table 4. Gate count and delay comparison for the proposed architecture and the others in VLSI technologies

	Proposed	[15]	[16]	[17]
Tech (nm)	*65*	*65*	*130*	*130*
Mult. operation	Bit-serial	Bit-serial	Bit-serial	Bit-serial
# of clock cycles	*52,012*	*106,700*	*219,148*	*275,816*
Area (GE*)	*12,102*	*11,571*	*11,720*	*12,506*
Freq. (KHz)	*1,130*	*106*	*400*	*1,130*
Time (ms)	*46*	*1,006.6*	*547.87*	*244.08*

*GE: gate equivalent (NAND gate)

multiplication, the implementation of the proposed architecture on Virtex-5 has less hardware complexity and lower number of clock cycles than the architecture of Sutter et al. [1] using the same bit-serial multiplication approach (digit size = 1). The efficiency of our work shows 14% higher than Sutter et al. [1]. Also, we have implemented the proposed architecture using Xilinx Virtex-7 FPGA. The proposed architecture using Virtex-7 requires only 3,806 LUTs, which is 26% less hardware complexity than the architecture of Sutter et al. [1] using the same bit-serial multiplication approach (digit size = 1). Noticeably, the proposed architecture uses about one-seventh of the hardware resources, compared with the architecture of Mahdizadeh and Masoumi [8].

The VLSI implementation results of the proposed architecture using TSMC 65-nm CMOS standard technology is shown in Table 4. The proposed architecture is modeled in the Verilog HDL and simulated to verify its functionality. After complete verification of the design functionality, it is then synthesized using appropriate time and area constraints. Synthesis step is carried out using the SYNOPSYS design tool and 65-nm CMOS technology. The estimated total number of NAND gates equivalent (GE) is 12,102 from the synthesized results, and a maximum clock frequency is 800 MHz. However, we use a clock frequency of 1,130 KHz in order to compare with other works [15-17]. As can be seen, the proposed architecture uses a similar number of NAND gates compared to other works. However, the proposed architecture requires only 52,012 clock cycles, which is a half, a quarter, and one-fifth of that in [15-17], respectively, to finish one point multiplication. Furthermore, at a clock frequency of 1,130 KHz used in Lee et al. [17], the proposed architecture finishes a point multiplication after 46 ms.

VI. Conclusion

This paper presented a novel efficient algorithm and architecture to compute ECC point multiplication. The result shows that the proposed architecture outperforms the existing architectures in terms of efficiency. Furthermore, to perform ECC point multiplication, our work incurs a lower hardware cost than the others. Therefore, the proposed algorithm and architecture is suitable for applications that must strictly consider hardware cost as well as system efficiency.

Acknowledgments

This research was supported by Basic Science Research Program through the NRF funded by the Ministry of Science, ICT and Future Planning (2013R1A2A2A01068628).

References

[1] G. D. Sutter, J.-P. Deschamps, and J. L. Imaña, "Efficient Elliptic Curve Point Multiplication using Digit-Serial Binary Field Operations," *IEEE Trans. on Industrial Electronics*, vol. 60, no.1, pp. 217-225, Jan. 2013.

[2] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," *Des. Codes Cryptography*, vol. 19, no. 2–3, pp. 173–193, Mar. 2000.

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

[4] J.-P. Deschamps, J. L. Imaña, and G. D. Sutter, *Hardware Implementation of Finite-Field Arithmetic*. New York: McGraw-Hill, 2009, ser. Electronic Engineering Series.

[5] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on FPGA," *IEEE Trans. on Very Large Scale Integrated (VLSI) Systems*, vol. 16, no. 2, pp. 198-205, Feb. 2008.

[6] F. Rodríguez-Henríquez, N. A. Saqib, and A. Díaz-Pérez, "A fast parallel implementation of elliptic curve point multiplication over GF(2m)," *Micro- process. Microsyst.*, vol. 28, no. 5–6, pp. 329–339, Aug. 2004, Special issue on FPGAs: Applications and Designs.

[7] S. M. Shohdy, A. B. El-Sisi, and N. Ismail, "FPGA

implementation of elliptic curve point multiplication over GF(2^{191})," *Proc. 3rd Int. Conf. Workshops Adv. ISA*, Berlin, Heidelberg, Germany, pp. 619–634, Jun. 2009.

[8] H. Mahdizadeh and M. Masoumi, "Novel Architecture for Efficient FPGA Implementation of Elliptic Curve Cryptographic Processor Over GF(2^{163})," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 12, pp. 2330-2333, Dec. 2013.

[9] J.-C. Bajard, L. Imbert, and C. Nègre, "Arithmetic Operations in Finite Fields of Medium Prime Characteristic Using the Lagrange Representation," *IEEE Trans. on Computer*, vol. 55, no. 9, pp. 1167-1177, Sep. 2006.

[10] A. Hariri and A. Reyhani-Masoleh, "Bit-Serial and Bit-Parallel Montgomery Multiplication and Squaring over GF(2^m)," *IEEE Trans. on Computer*, vol. 58, no. 10, pp. 1332-1345, Oct. 2009.

[11] C.W. Chiou, C.-Y. Lee, J.-M. Lin, T.-W. Hou, C.-C. Chang, "Concurrent error detection and correction in dual basis multiplier over GF(2^m)," *IET Circuits, Devices & Systems*, vol. 3, no. 1, pp. 22-40, Feb. 2009.

[12] G. Meurice de Dormale and J.-J. Quisquater, "High-speed hardware implementations of elliptic curve cryptography: A survey," *J. Syst. Archit.*, vol. 53, no. 2–3, pp. 72–84, Feb./Mar. 2007.

[13] H. Li, K. Wu, G. Xu, H. Yuan and P. Luo, "Simple Power Analysis Attacks Using Chosen Message against ECC Hardware Implementations," *IEEE World Congress on Internet Security*, pp. 68-72, Feb. 2011.

[14] J. P. Deschamps, J. L. Imaña, and G. D. Sutter,, "Hardware Implementation of Finite-Field Arithmetic" *McGrawHill*, ISBN 978-0-0715-4581-5, Mar. 2009.

[15] R. Azarderakhsh, K. U. Järvinen, and M. M.-Kermani, "Efficient Algorithm and Architecture for Elliptic Curve Cryptography for Extremely Constrained Secure Application," *IEEE Trans. on Circuits and Systems-I*, vol. 64, no. 4, pp. 1144-1155, Apr. 2014.

[16] U. Kocabas, J. Fan, and I. Verbauwhede, "Implementation of binary Edwards curves for very-constrained devices," *Proc. 21st Int. Conf. Application-Specific Systems Architectures and Processors* (ASAP2010), pp. 185–191, Jul. 2010.

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

Tuy Nguyen Tan received a BSc in Electronics and Telecommunications in 2009 from Danang University of Technology, Danang City, Vietnam. From 2008, he was a member of Silicon Design Solutions (eSilicon), Danang branch, where he joined the internship program, and then worked as a circuit design engineer. He was responsible for designing the read and write assistant circuit for SRAM. From 2010 to 2013, he worked as a Technical Supervisor at GTel Mobile JSC, a member of the VimpelCom Group, Russia. He is currently working toward an MSc at Inha University, Korea. His interests include not only digital VLSI circuits and systems design for communications, such as cryptography systems, but also their efficient hardware implementation.

Hanho Lee received a PhD and MSc, both in Electrical & Computer Engineering, from the University of Minnesota, Minneapolis, in 2000 and 1996, respectively. In 1999, he was a Member of Technical Staff-1 at Lucent Technologies, Bell Labs, Holmdel, New Jersey. From April 2000 to August 2002, he was a Member of the Technical Staff at Lucent Technologies (Bell Labs Innovations), Allentown. From August 2002 to August 2004, he was an Assistant Professor in the Department of Electrical and Computer Engineering, University of Connecticut, USA. Since August 2004, he has been with the Department of Information and Communication Engineering, Inha University, where he is currently a Professor. He was a visiting researcher at the Electronics and Telecommunications Research Institute (ETRI), Korea, in 2005. From August 2010 to August 2011, he was a visiting scholar at Bell Labs, Alcatel-Lucent, Murray Hill, New Jersey, USA. His research interests include VLSI architecture design for digital signal processing, forward error correction architectures, cryptographic systems, and communications.