High-Performance Architecture for Elliptic Curve Cryptography over Prime Fields on FPGAs

Jyu-Yuan LAI, Yu-Shiang WANG, and Chih-Tsun HUANG*

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30013

Received May 18, 2012; final version accepted August 23, 2012

We present a high-performance Elliptic Curve Cryptographic (ECC) processor that supports arbitrary prime field and curve parameters. A novel pipeline architecture of the Montgomery multiplication by using the inherent DSP blocks in modern FPGAs is proposed to speed up the point scalar multiplication. In addition, the improved operation scheduling is presented to optimize the operation cycles further. With Xilinx Virtex-5 FPGA devices, a 256-bit point scalar multiplication can be performed in 0.86 ms at 263 MHz, about 3.14 to 11.27 times faster than other designs with comparable functionalities. Our processor therefore outperforms others significantly in terms of throughput, area, and cost-effectiveness.

KEYWORDS: elliptic curve cryptography (ECC), prime field, FPGA implementation, high-performance architecture

1. Introduction

Digital security has become an urgent need nowadays for modern vital applications. Elliptic Curve Cryptography (ECC), independently proposed by Miller [1] and Koblitz [2], has been considered mature as compared with conventional public-key cryptosystems, e.g., RSA cryptography. As an alternative to RSA, ECC provides much higher levels of security strength with the same key length. Therefore, ECC has been widely adopted in modern security standards [3–5].

There are two distinct requirements for practical use of ECC. Ultra low-cost design (e.g., for power, energy, and hardware area) is demanded for portable devices and ubiquitous applications, e.g., the appearing RFID applications. At the other end, extremely high-speed and powerful cryptographic engines are required for server-side applications to deliver the maximized performance. Both of the diverse criteria can be optimized with hardware accelerator to achieve the best cost-effectiveness.

Recently, many high-performance ECC designs on FPGA (Field-Programmable Gate Array) devices have been published (e.g., ECC over binary field $GF(2^m)$ in [6, 7], prime field $GF(p)$ in [8–11], and dual-field in [12]). FPGA inherently provides the fast operating frequency, high-performance embedded arithmetic components, and great reconfigurability with lower inventory cost, making it an attractive alternative to ASIC approach for server-side security applications.

In [6], a two-stage-pipelined word-serial multiplier has been presented for specific irreducible polynomial. Different levels of parallelism among design hierarchies for ECC has been explored in [7]. A 256-bit full-word Montgomery multiplier has been presented to reduce the operation cycles significantly in [8]. In addition, a dual-field ECC processor utilizing instruction-level parallelism with one to four Modular Arithmetic Logic Units (MALUs) has been proposed in [12].

Several approaches have also utilized the inherent DSP (Digital Signal Processing) blocks in FPGAs to optimize the overall throughput and cost [9–11]. By taking the advantage of the high-speed DSP blocks, the design in [9] can improve the operating frequency to 490 MHz. In [10], the Hiasat multiplier with DSP blocks performed one 127-bit modular multiplication in one clock cycle. Furthermore, the GLV (Gallant, Lambert, and Vanstone) method [13] was applied to replace a larger point scalar multiplication with two smaller ones to reduce the overall complexity. However, both of the two fast ECC designs was designed for supporting specific prime fields, i.e., P -224 and P -256 recommended by NIST [14] in [9] and the Mersenne prime of type $2^n - 1$ in [10], which limits their flexibility substantially. Recently, the design based on the residue number system has been shown in [11], which achieves a higher degree of parallelism with a large amount of DSP blocks, resulting in the fast ECC implementation for general prime fields.

This work was supported in part by National Science Council, Taiwan, ROC, under contract NSC 98-2220-E-007-025, NSC 98-2622-E-007-014-A2, and by Ministry of Economic Affairs, Taiwan, ROC, under contract 97-EC-17-A-01-S1-002.

* Corresponding author. E-mail: cthuang@cs.nthu.edu.tw

In this paper, we present a high-performance processor for ECC over $GF(p)$ by using the inherent DSP blocks for extremely fast Elliptic Curve (EC) arithmetic. Arbitrary prime fields and elliptic curves are supported. We propose a novel architecture of Montgomery multiplier by cascading the DSP blocks to reduce the number of cycles and to eliminate extra routing delay. In addition, to further optimize the operation cycles, the improved operation scheduling for the finite field operations in EC arithmetic is presented by overlapping successive Montgomery multiplications, i.e., the modular reduction stage of the present Montgomery multiplication can be done with the multiplication stage of the next one in parallel. Successive iterations in the point scalar multiplication can be overlapped as well. Our ECC processor can perform a 256-bit point scalar multiplication in 0.86 ms at 263 MHz on Xilinx Virtex-5 FPGAs. The comparison with other ECC implementations justifies the performance, cost-effectiveness, and the flexibility of our approach.

2. The Word-based Montgomery Multiplier

Our processor focuses on the ECs over $GF(p)$ in IEEE 1363 *Standard Specification for Public-Key Cryptography* [3]. The standardized EC over $GF(p)$ is $y^2 = x^3 + \alpha x + \beta$, where $x, y \in GF(p)$ and $4\alpha^3 + 27\beta^2 \neq 0 \pmod{p}$. EC point scalar multiplication, the most important operation in ECC, can be decomposed into iterative point doubles and point additions which involve finite field operations. We adopt the addition-and-subtraction method in [3] for the point scalar multiplication. In addition, Jacobian's projective coordinate $(x, y, z) \rightarrow (x/z^2, y/z^3)$ over $GF(p)$ is used in our processor to effectively replace the field inversion with several field multiplications.

Montgomery multiplication algorithm is a well-known fast modular multiplication method. Furthermore, in order to utilize the inherent high-performance DSP blocks in modern FPGAs to accelerate the computation, we proposed the word-based Montgomery multiplication, as shown in Algorithm 1. Different from the Multiple-word High-Radix Montgomery Multiplication (MWR 2^k MM) in [15] and the high-radix scheme in [16], we partition the operands according to the word width of the arithmetic components in the DSP blocks in order to maximize the utilization of the DSP blocks and to reduce the number of cycles. In addition, we applied the algorithm to pipelined architecture by cascading the DSP blocks to improve the frequency.

Algorithm 1 A word-based Montgomery multiplication.

Input: A, B, p , and p' , where p is an n -bit prime, $A, B \in GF(p)$, and $2^n \times 2^{-n} - p \times p' = 1$.

Output: $S = A \times B \times 2^{-n} \pmod{p}$.

```

1:  $\tau = (p' \pmod{2^k}) \times p$ .
2: for  $i = 0$  to  $w - 1$  do // Outer Loop (Multiplication Stage)
3:    $c_0 = c_{-1} = 0$ 
4:   for  $j = 0$  to  $e - 1$  do // Inner Loop 1
5:      $(c_j, s_j) = s_j + a_i \times b_j + c_{j-1}$ 
6:   end for
7:    $q = S \pmod{2^k}$ 
8:    $c_0 = c_{-1} = 0$ 
9:   for  $j = 0$  to  $e - 1$  do // Inner Loop 2
10:     $(c_j, s_j) = s_j + q \times \tau_j + c_{j-1}$ 
11:  end for
12:   $S = S/2^k$ 
13: end for
14:  $S = S \pmod{p}$  // Reduction Stage

```

In Algorithm 1, let $m \times k$ -bit multiplier be used, we partition the multiplicand A into w words, and the multiplier B into e words, respectively, where $w = \lceil \frac{n}{k} \rceil$, $e = \lceil \frac{n+k}{m} \rceil$, and n is the field size. Therefore, $A = \sum_{i=0}^{w-1} a_i 2^{ik}$ and $B = \sum_{i=0}^{e-1} b_i 2^{im}$, where $b_{e-1} = 0$. As a result, an $n \times n$ -bit Montgomery multiplication is divided into multiple $m \times k$ -bit atomic multiplications. In addition, to effectively reduce the operation cycles, 24×16 -bit multipliers are used for our 256-bit Montgomery multiplication (i.e., $m = 24$, $k = 16$, $w = 16$, and $e = 12$), where the inherent DSP block in Xilinx Virtex-5 family comes with a 25×18 -bit multiplier.

Figure 1 shows the data dependency graph of Algorithm 1. Each node consists of one 24×16 -bit multiplication and at most two 40-bit additions. In addition, node $A_{i,j}$ and $B_{i,j}$ represent the computation of $(c_j, s_j) = s_j + a_i \times b_j + c_{j-1}$ and $(c_j, s_j) = s_j + q \times \tau_j + c_{j-1}$, respectively (see line 5 and 10 in Algorithm 1). Because the output of node $A_{i,j}$ is one of the inputs of node $B_{i,j}$, the Inner Loop 1 and Inner Loop 2 in each iteration of the Outer Loop can be overlapped for $e - 1$ stages. Moreover, the intermediate result S discards the least significant k bits at the end of every iteration of the Outer Loop (see line 12), the node $A_{i,j}$ for $0 < i \leq w - 1$ is fed by the node $B_{i-1,j+1}$ instead of $B_{i-1,j}$, i.e., the Inner Loop 2 and the successive Inner Loop 1 is overlapped for $e - 2$ stages.

The proposed Processing Elements (PEs) are shown in Fig. 2, i.e., PEA and PEB perform the computation of node $A_{i,j}$ and $B_{i,j}$ in Fig. 1, respectively. Both PEs utilize the high-performance DSP blocks in modern FPGAs as the

Fig. 1. The data dependency graph of Algorithm 1.

Fig. 2. The proposed processing elements (a) PEA and (b) PEB.

Fig. 3. (a) The proposed 4-stage pipelined multiplier and (b) the modular reduction for the Montgomery multiplication.

kernel component, e.g., the DSP48E in Xilinx Virtex-5 family. To optimize the operating frequency, the embedded registers in the DSP block are used to keep the intermediate results s_j . Additional routing delay introduced by using the registers in generic FPGA slices can therefore be eliminated. Note that PEB requires additional 16-bit register to store the least significant word of S , i.e., q , for the Inner Loop 2 in Algorithm 1. It also needs another 8-bit register for the most significant 8-bit data of each word of S after discarding the least significant 16-bits in each iteration of the Outer Loop.

Figure 3(a) shows the proposed pipeline architecture for the Montgomery multiplication with four pipeline stages. At most four node $A_{i,j}$ and four node $B_{i,j}$ can be performed simultaneously while no precedence violation exists. Each PE propagates s_j to the next one and c_j back to itself in every cycle. For the simplification, Fig. 4 shows an example of the operation cycles for the Outer Loop in Algorithm 1 by using 2-stage pipelined Montgomery multiplier when $w = 5$ and $e = 7$, i.e., there are two PEAs and two PEBs, and the operands A and B are divided into five and seven words, respectively. Let x denotes the number of pipeline stages, there are at least three cycles between PEA_i and PEA_{i+1} , and $(e - 3x)$ additional cycles are required for waiting available PEs between every x PEAs when x is less than $\lceil e/3 \rceil$. Finally, the last iteration of the Outer Loop in Algorithm 1 requires $(e + 1)$ cycles. Therefore, the number of cycles for the Outer Loop can be formulated as

$$C_m = \begin{cases} 3 \times (w - 1) + (e - 3x) \times (\lceil \frac{w}{x} \rceil - 1) + (e + 1), & x < \lceil \frac{e}{3} \rceil, \\ 3 \times (w - 1) + (e + 1), & \text{Otherwise.} \end{cases} \quad (1)$$

Fig. 4. The operation cycles for the Outer Loop in Algorithm 1 by using 2-stage pipeline ($x = 2$) Montgomery multiplier when $w = 5$ and $e = 7$.

As a result, the 4-stage ($x = 4$) pipelined multiplier is used, which requires 58 cycles for the multiplication stage (i.e., the entire Outer Loop, lines 2–13 in Algorithm 1) of the 256-bit Montgomery multiplication by using the 24×16 -bit multiplier in the DSP block in Virtex-5 FPGAs (where $w = 16$ and $e = 12$).

Our pipeline multiplier generates $(n + k)$ bits of the product S . Let the most significant k bits of S be d , we subtract $d \times p$ and $(d + 1) \times p$ from S simultaneously by using two parallel PEAs and select the final result according to the sign bit, as shown in Fig. 3(b). The product S and the prime p are stored in two circular shift registers, X and Y , respectively. The least significant word of X is subtracted by $d \times p_j$ and $(d + 1) \times p_j$, and then stored into the most significant word of X and Y in every cycle. Therefore, the number of cycles for the modular reduction stage can be represented as $C_r = e$ (i.e., line 14 in Algorithm 1), and the overall Montgomery multiplication requires $C_m + C_r$ cycles.

3. Operation Scheduling

Our ECC processor performs point scalar multiplication by using the addition-and-subtraction method in [3] for arbitrary prime fields and elliptic curves. The point scalar multiplication is decomposed to iteratively point double (PDBL) or point double with point addition/subtraction (PADDSUB). To optimize the performance, we implement the Montgomery multiplier (see Fig. 3) and the modular addition with the DSP blocks. An n -bit modular addition can be performed in $C_a = 2 \times \frac{n}{r}$ cycles, where r is the word width of the carry propagation adder in the DSP blocks. As a result, a 256-bit modular addition requires 12 cycles where the DSP block in Xilinx Virtex-5 family comes with a 48-bit carry propagation adder.

With Jacobian's projective coordinate, we decompose the PDBL, i.e., $P_2(x_2, y_2, z_2) = 2P_1(x_1, y_1, z_1)$, into eleven atomic multiplications and eight additions. In addition, mix-coordinate PADDSUB, i.e., $P_2(x_2, y_2, z_2) = P_0(x_0, y_0, z_0) + P_1(x_1, y_1, 1)$, is adopted to reduce the complexity, which can be implemented by twelve multiplications and eight additions. Three types of operation scheduling are proposed for PDBL and PADDSUB by using one modular multiplier and adder (see Figs. 5, 6, and 7). First, the preliminary scheduling (Type-I) performs the modular multiplication and addition/subtraction simultaneously as long as no precedence violation occurs, and accomplishes the modular reduction after the multiplication, as shown in Fig. 5(a). $C_m + C_r + 2$ cycles are required in each stage where the additional two cycles are introduced by moving data from and to the RAM. By using the addition-and-subtraction method in [3], there are approximately two-third of iterations executing PDBL and PADDSUB while the scalar is in NAF (non-adjacent form) representation, and the rest of iterations executing only PDBL. As a result, Type-I scheduling can perform a point scalar multiplication in $C_{psm,I} = 2/3(n - 1)(11(C_m + C_r + 2) + C_a) + 1/3(n - 1)(22(C_m + C_r + 2) + 3C_a)$ cycles.

To reduce the operation cycles, Type-II scheduling performs the reduction stage of the present Montgomery multiplication (Stage t) with the next multiplication (Stage $t + 1$) at the same time, as long as no data dependency exists between any successive stages [see Fig. 5(b)]. Figure 6 shows its detail scheduling for point double and point addition.

Fig. 5. (a) Preliminary (Type-I) and (b) Type-II operation scheduling technique.

Stage	MUL	ADD
PDBL 1	$P_0 = z_1^2$	—
PDBL 2	$P_3 = x_1^2$	$2y_1 = y_1 + y_1$
PDBL 3	$P_1 = P_0^2$	$2P_3 = P_3 + P_3$
PDBL 4	$P_4 = (2y_1)^2$	$M_3 = P_3 + 2P_3$
PDBL 5	$P_2 = \alpha \times P_1$	—
PDBL 6	$S = x_1 \times P_4$	$M_1 = M_3 + P_2$
PDBL 7	$P_5 = M_1^2$	$2S = S + S$
PDBL 8	$P_6 = P_4^2$	$x_2 = P_5 - 2S$
PDBL 9	$z_2 = 2y_1 \times z_1$	—
PDBL 10	$T = P_6 \times 2^{-1}$	$P_7 = S - x_2$
PDBL 11	$P_8 = M_1 \times P_7$	—
PDBL 12	—	$y_2 = P_8 - T$

(a)

Stage	MUL	ADD
PADD 1	$Q_0 = z_0^2$	—
PADD 2	—	—
PADD 3	$Q_1 = Q_0 \times z_0$	—
PADD 4	$U_1 = x_1 \times Q_0$	—
PADD 5	$S_1 = y_1 \times Q_1$	$W = x_0 - U_1$
PADD 6	$Q_1 = W^2$	$R = y_0 - S_1$
PADD 7	$Q_3 = R^2$	$T = x_0 + U_1$
PADD 8	$Q_4 = T \times Q_1$	$M = y_0 + S_1$
PADD 9	$z_2 = z_0 \times W$	$x_2 = Q_3 - Q_4$
PADD 10	$Q_5 = Q_1 \times W$	$2x_2 = x_2 + x_2$
PADD 11	—	$V = Q_4 - 2x_2$
PADD 12	$Q_7 = V \times R$	—
PADD 13	$Q_6 = M \times Q_5$	—
PADD 14	—	$2y_2 = Q_7 - Q_6$
PADD 15	$y_2 = 2y_2 \times 2^{-1}$	—

(b)

Fig. 6. Type-II operation scheduling for (a) point double and (b) point addition.

Stage	MUL	ADD
...	...	...
PDBL 1	$P_0 = z_1^2$	—
PDBL 2	$P_3 = x_1^2$	$2y_1 = y_1 + y_1$
PDBL 3	$P_1 = P_0^2$	$2P_3 = P_3 + P_3$
...	...	...
PDBL 10	$T = P_6 \times 2^{-1}$	$P_7 = S - x_2$
PDBL 11	$P_8 = M_1 \times P_7$	—
PDBL 12	$P_0 = z_2^2$	$y_2 = P_8 - T$
PDBL 2	$P_3 = x_2^2$	$2y_2 = y_2 + y_2$
PDBL 3	$P_1 = P_0^2$	$2P_3 = P_3 + P_3$
...	...	...

(a)

Stage	MUL	ADD
...	...	...
PDBL 9	$z_2 = 2y_1 \times z_1$	—
PDBL 10	$T = P_6 \times 2^{-1}$	$P_7 = S - x_2$
PADD 1	$Q_0 = z_2^2$	—
PDBL 11	$P_8 = M_1 \times P_7$	—
PADD 3	$Q_1 = Q_0 \times z_0$	$y_2 = P_8 - T$
PADD 4	$U_1 = x_1 \times Q_0$	—
...	...	...
PADD 14	$P_0 = z_1^2$	$2y_2 = Q_7 - Q_6$
PDBL 2	$P_3 = x_1^2$	—
...	...	...

(b)

Fig. 7. Successive iterations with Type-III operation scheduling for (a) (PDBL)-(PDBL) and (b) (PDBL-PADDSUB)-(PDBL).

Fig. 8. Operation cycles for Type-I, Type-II, and Type-III operation scheduling by using Xilinx Virtex-5 FPGAs.

Although additional bubble stage is required to fix the precedence violation, e.g., the stage PADD 2 in Fig. 6(b), the number of cycles for each stage can be reduced from $C_m + C_r + 2$ to $C_m + 2$. Consequently, a point scalar multiplication can be performed in $C_{psm,II} = 2/3(n-1)(11(C_m + 2) + C_a) + 1/3(n-1)(25(C_m + 2) + 3C_a)$ cycles.

To further optimize the performance, Type-III operation scheduling overlaps successive iterations with the following conditions: (1) As shown in Fig. 7(a), for (PDBL)-(PDBL), PDBL 12 of the present iteration can be overlapped with PDBL 1 of the next iteration; (2) For (PDBL-PADDSUB)-(PDBL), PDBL 11 can be inserted to PADD 2. In addition, PADD 14 of current iteration can be overlapped with PDBL 1 of the next iteration. Furthermore, PADD 15 can be omitted since $2y_2$ can be fed into the next PDBL smoothly by ignoring the addition operation in PDBL 2, as shown in Fig. 7(b). Therefore, one stage can be effectively removed for PDBL, and two stages can be reduced for PDBL-PADDSUB, respectively. As a result, a point scalar multiplication can be performed in $C_{psm,III} = 2/3(n-1)(10(C_m + 2)) + 1/3(n-1)(23(C_m + 2) + C_a)$ cycles.

Figure 8 compares the operation cycles for Type-I, Type-II, and Type-III operation scheduling by using Xilinx Virtex-5 FPGAs ($C_m = 58$, $C_r = 12$, and $C_a = 12$). With the techniques shown in Figs. 5(b) and 7, Type-III operation scheduling reduces the operation cycles by 18.49% as compared with Type-I scheduling.

4. Implementation Results and Comparison

For a fair comparison with various FPGA devices, our ECC processor (with Type-III operation scheduling) has been synthesized on different FPGA families, namely, Xilinx Virtex-2 XC2VP30, Virtex-4 XC4VFX12, and Virtex-5

Table 1. Comparison among ECC designs on Xilinx Virtex-2 platform for 256-bit $GF(p)$ point scalar multiplication.

	Prime Field	f_{max} (MHz)	Cycles	Time (ms, T)	Area [Normalized] (A)	AT
Ours	Arbitrary	112	250,928	2.24	3,423 Slices + 14 18×18-bit MULs [5,397 Slices]	1.00
[8]	Arbitrary	40	151,360	3.86	15,755 Slices + 256 18×18 -bit MULs [51,851 Slices]	16.86
[12]	Arbitrary	100	269,886	2.70	10,847 Slices	2.42

Table 2. Comparison among ECC designs on Xilinx Virtex-4 [9] and Altera Stratix II [11] platforms for 256-bit $GF(p)$ point scalar multiplication.

	Prime Field	f_{max} (MHz)	Cycles	Time (ms, T)	Area [Normalized] (A)	AT
Ours	Arbitrary	227	248,292	1.09	2,901 Slices + 14 DSPs [11,567 Slices]	1.17
[9]	NIST-256	490	303,450 ¹	0.62 ¹	1,715 Slices + 32 DSPs [21,523 Slices]	1.24 ¹
			243,000 ²	0.50 ²		1.00 ²
[11]	Arbitrary	157.2	106,896*	0.68	9,177 ALMs + 96 DSPs [68,601 Slices [†]]	4.33

¹Double-and-Add method; ²Window-based method.*Estimated from the given information; [†]Assuming that a Stratix II ALM and a Virtex IV slice are equivalent.Table 3. Comparison among ECC designs on Xilinx Virtex-5 platform for 256-bit $GF(p)$ point scalar multiplication.

	Prime Field	f_{max} (MHz)	Cycles	Time (ms, T)	Area [Normalized] (A)	AT
Ours	Arbitrary	263	225,720	0.86	3,657 Slices + 10 DSPs [13,577 Slices]	1.07
[10]	Arbitrary	100	970,000*	9.70	2,025 Slices + 0 DSPs [2,025 Slices]	1.81
	$2^n - 1$	50	9,411*	0.19	1,439 Slices + 56 DSPs [56,991 Slices]	1.00

*Estimated from the given information.

XC5VLX110. Table 1 compares different ECC designs on Xilinx Virtex-2 devices for point scalar multiplication. Note that because the word widths of the inherent arithmetic components in Virtex-2 and Virtex-4 FPGAs are smaller than the one in Virtex-5 FPGAs, the number of cycles for point scalar multiplication by using Virtex-5 FPGAs is less than those with Virtex-2 and Virtex-4 FPGAs (see Tables 1, 2, 3). In Table 1, our implementation is the fastest one in terms of the operation time. Note that [12] proposed a dual-field architecture. The approach in [8] reduces the operation cycles significantly by utilizing a full-word 256×256 Montgomery multiplier. However, it results in a much larger hardware. To further compare the area, we normalize the inherent multiplier (MUL) in Virtex-2 FPGAs to 141 slices according to our synthesis experiment. The comparison shows that our word-based ECC processor provides the fastest operation time as well as the best cost-effectiveness, i.e., the AT product.

In Tables 2 and 3, we also summarize the comparison among the ECC implementations on modern FPGAs, i.e., [9] was on Xilinx Virtex-4 FPGA, [10] was on Xilinx Virtex-5, and [11] was on Altera Stratix II FPGA. Although in Table 2, the FPGAs adopted in [9] and [11] are different, they are both fabricated by using 90 nm CMOS technology. Also, one can normalize an Adaptive Logic Module (ALM) in Stratix II FPGAs to a slice in Virtex-4 FPGA, as stated in [11]. In addition, we also implemented functional-compatible modules for the DSP blocks in Virtex-4 and Virtex-5 FPGAs for the area comparison. Each DSP block in Virtex-4 can be mapped into 619 slices, and the one in Virtex-5 is normalized to 992 slices, respectively.

Both [9] and [10] proposed ECC designs supporting special prime fields, e.g., the specific prime recommended by NIST [14] in [9] and the Mersenne prime of the type $2^n - 1$ in [10]. Note that [10] also provided architectures for generic prime fields. With the special prime fields, the complexity of modular reduction can be minimized significantly. Therefore, [9] can further pipeline the DSP blocks to improve the operating frequency to 490 MHz. In addition, the Hiasat multiplier over $GF(p^2)$, which performs a 127-bit modular multiplication in one cycle, is used in [10] to reduce the operation cycles effectively. As a result, a 256-bit point scalar multiplication can be performed in 0.19 ms in [10], which is the fastest design to the best of our knowledge. However, the architectures dedicated to specific prime fields will limit their flexibility.

Tables 2 and 3 show that our processor supporting arbitrary prime fields and elliptic curves provides comparable cost-effectiveness compared with the fast ECC designs for specific fields in [9] and [10]. In addition, compared with the design supporting arbitrary prime fields in [8, 10, 12], our processor is 3.14 times to 11.27 times faster in terms of the operation time. Although the design in [11] is 1.6 times faster than ours, our design is 3.7 times more cost-effective in AT product. The comparison justifies that our processor outperforms other designs of arbitrary primes for the speed or the cost-effectiveness.

5. Conclusion

This paper has presented a high-performance ECC architecture featuring arbitrary prime fields and elliptic curves. The proposed word-based pipelined Montgomery multiplier operates at a high clock rate with the inherent DSP blocks on modern FPGA devices. The number of operation cycles is effectively reduced by the improved operation scheduling for the finite field operations in EC arithmetic. A 256-bit point scalar multiplication can be done in 0.86 ms at 263 MHz on Xilinx XC5VLX110. The comparison indicates that our ECC processor outperforms other FPGA designs significantly with the capability of supporting arbitrary prime fields and elliptic curves, which delivers the maximized performance and the flexibility for widespread security applications as well.

REFERENCES

- [1] Miller, V. S., "Use of elliptic curve in cryptography," in *Advances in Cryptology—Crypto'85 Proceedings*, 417–426 (1986).
- [2] Koblitz, N., "Elliptic curve cryptosystems," in *Mathematics of Computation*, 203–209 (1987).
- [3] IEEE, *IEEE 1363 Standard Specifications for Public-Key Cryptography*, IEEE Standards Department, Piscataway (2000).
- [4] ANSI, *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, American National Standards Institute (ANSI), Washington, DC (1998).
- [5] SECG, *SEC 2: Recommended Elliptic Curve Domain Parameters*, Standards for Efficient Cryptography Group (2000).
- [6] Ansari, B., and Hasan, M. A., "High-performance architecture of elliptic curve scalar multiplication," *IEEE Trans. Computers*, **57**: 1143–1153 (2008).
- [7] Järvinen, K., and Skyttä, J., "On parallelization of high-speed processors for elliptic curve cryptography," *IEEE Trans. VLSI Systems*, **16**: 1162–1175 (2008).
- [8] McIvor, C. J., McLoone, M., and McCanny, J. V., "Hardware elliptic curve cryptographic processor over $GF(p)$," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, **53**: 1946–1957 (2006).
- [9] Güneysu, T., and Paar, C., "Ultra high performance ECC over NIST primes on commercial FPGAs," in *Cryptographic Hardware and Embedded Systems (CHES) 2008, LNCS*, Vol. 5154, 62–78, Springer-Verlag (2008).
- [10] Hamilton, M., and Marnane, W. P., "FPGA implementation of an elliptic curve processor using the GLV method," in *Proc. IEEE Int'l Conf. Reconfigurable Computing and FPGAs (ReConFig)*, Cancun, 249–254 (2009).
- [11] Guillermin, N., "A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$," in *Cryptographic Hardware and Embedded Systems (CHES) 2010, LNCS*, Vol. 6225, 48–64, Springer-Verlag (2010).
- [12] Sakiyama, K., De Mulder, E., Preneel, B., and Verbauwhede, I., "A parallel processing hardware architecture for elliptic curve cryptosystems," in *Proc. IEEE Int'l Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Toulouse, Vol. 3, 904–907 (2006).
- [13] Gallant, R. P., Lambert, R. J., and Vanstone, S. A., "Faster point multiplication on elliptic curves with efficient endomorphisms," *CRYPTO 2001*, **2139**: 190–200 (2001).
- [14] NIST, *Recommended Elliptic Curves for Federal Government Use*, National Institute of Standards and Technology (NIST), Gaithersburg (1999).
- [15] Tenca, A. F., Todorov, G., and Koç, Ç. K., "High-radix design of a scalable modular multiplier," in *Cryptographic Hardware and Embedded Systems (CHES) 2001, LNCS*, Vol. 2162, 185–201, Springer-Verlag (2001).
- [16] Blum, T., and Paar, C., "High-radix montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. Comput.*, **50**: 759–764 (2001).