

An Efficient Multiplier/Divider Design for Elliptic Curve Cryptosystem over $GF(2^m)^*$

MING-DER SHIEH, JUN-HONG CHEN, WEN-CHING LIN AND CHIEN-MING WU[†]

*Department of Electrical Engineering
National Cheng Kung University
Tainan, 701 Taiwan*

[†]*Chip Implementation Center
National Applied Research Laboratories
Hsinchu, 300 Taiwan*

E-mail: shiehm@mail.ncku.edu.tw

Using the concept of reciprocal polynomial, this paper shows that a field multiplication over $GF(2^m)$ can be implemented by extended Stein's algorithm, one of the algorithms used to accomplish division. In this way, a field multiplier can be efficiently embedded into a divider with very little hardware overhead for operand selection based on a fundamental change at the algorithmic level. When applying the developed combined multiplication and division (CMD) algorithm to Elliptic Curve Cryptography (ECC) using affine coordinates, we achieve about 13.8% reduction on the area requirement with almost no performance degradation compared to the one implemented with two distinct components. Experimental results also demonstrate that not only our CMD circuit has the area advantage (up to 12.7%) in comparison with other low-cost design but also the resulting area-efficient design of ECC system exhibits considerable improvement on the area-time (AT) complexity of previous work.

Keywords: elliptic curve cryptography (ECC), affine coordinates, combined multiplier and divider (CMD) design, extended stein's division algorithm, MSB-first multiplier, polynomial basis

1. INTRODUCTION

The Elliptic Curve Cryptography (ECC), proposed independently by Koblitz and Miller in 1985, has received increasing acceptance in practical applications [1-4]. As stated in [5, 6], ECC offers the most security per bit compared with other public key cryptosystems such as RSA cryptosystem, making it suitable for applications with constrained resources. For example, in some wireless applications, they demand a secured data communication but the devices have limited resources to offer such an option. For security consideration, the key length should be long enough to defend attack; therefore an area-efficient feature would be desirable in hardware implementation of ECC system. In this paper, based on the inherent data dependence in performing multiplication and division, the primitive operations required in ECC using affine coordinates, we show how to increase the hardware utilization of ECC over $GF(2^m)$ by efficiently combining the multiplication and division into a unified unit. Applying the combined multiplier and

Received October 31, 2007; revised April 17, 2008; accepted May 15, 2008.

Communicated by Liang-Gee Chen.

[†] This work was supported in part by the National Science Council of R.O.C. under grant No. NSC 96-2221-E-006-296-MY3, and was presented in part in Proceedings of IEEE Region 10 Conference TENCON, Taipei, Taiwan, Oct. 2008.

divider (CMD) design to ECC over $GF(2^m)$ can achieve obvious improvement on area requirement with almost no performance degradation compared to the one implemented with two distinct components.

In ECC, there are two different types of coordinates that can be used for point representation: affine and projective. Conventionally, the projective coordinates are preferred [7-12] because of the relatively poor performance of division compared with the multiplication. The price paid is the increased number of multiplications, temporary registers, and more complicated algorithms. In contrast, for ECC using affine coordinates it requires one division and two multiplications for each full addition operation; therefore the efficiency of divider will dominate the resulting ECC performance. Some researchers replaced the inversion with a series of multiplications by employing Fermat's theorem, which takes at least seven multiplications in $GF(2^m)$ if $m \geq 128$ [13]. From this viewpoint, we can acquire the speed advantage for such a replacement if the computation time of the employed inversion is less than seven times that of the multiplication.

In this paper, we focus on the design of low-cost ECC systems over $GF(2^m)$ using affine coordinates in which a full addition takes one division and two multiplications, and the two types of operations are executed sequentially. Assume that the division and multiplication over $GF(2^m)$ converge in $2m$ and m iterations, respectively [14-18]. The resulting hardware utilization of ECC systems would be only 50% if we implement the multiplication and division modules separately. To increase hardware utilization, Kim [19] proposed a compact finite processor over $GF(2^m)$ in which the multiplication and division share the same datapath. In addition to the area overhead for operation mode control, the design needs one more m -bit register to store the temporary value because the underlying multiplication algorithm starts its computation from the least significant bit (LSB).

In fact, most division algorithms such as the extended Stein's algorithms are possible to implement the multiplication because they possess the basic operations of multiplication. The difficulty comes from the different shift direction in the course of iterative operation. In this paper, we consider a fundamental change at the algorithmic level and show that the field multiplication can be implemented based on Stein's algorithm and completed in m iterations using the concept of reciprocal polynomial. With a unified shift direction, the modified multiplication algorithm can then be effectively merged into extended Stein's algorithm to get a combined multiplication and division (CMD) algorithm. By the proposed CMD algorithm, the multiplier and divider can then be efficiently combined. Note that the modified multiplication algorithm can also be combined with the variant Euclidean division algorithm [15] in a similar way.

Experimental results show that about 27.2% hardware area is saved when we employ the CMD design instead of implementing the divider and multiplier separately. The area reduction of our CMD design can improve up to 12.7% compared with the low-cost design in [19]. When applying the CMD design to ECC systems using affine coordinates, we achieve about 13.8% improvement in area requirement. Furthermore, our area-efficient design of ECC system exhibits considerable improvement (at least 23.9% improvement) on the area-time (AT) complexity of previous work [8, 11, 20]. To demonstrate the effectiveness of our development, we also analyze the required computation time to accomplish the point multiplication and show that the ECC employing our CMD in affine coordinates can outperform that implemented in projective coordinates.

The rest of this paper is organized as follows. In section 2, we briefly review the re-

lated finite field arithmetic in GF(2^m) and the ECC using affine coordinates. Section 3 presents the proposed CMD algorithm and section 4 shows the corresponding architecture design and complexity evaluation. Then, in section 5 we apply the developed CMD technique to realize the ECC systems using affine coordinates and show area and time complexity analysis. Section 6 shows our CMD and ECC implementation results and comparison with existing work. Finally, section 7 concludes this work.

2. BACKGROUND AND NOTATION

2.1 Finite Field Arithmetic in GF(2^m)

Point computation in elliptic curve is defined over finite fields GF(*p*) or GF(2^{*m*}). In this paper, we only consider the arithmetic in GF(2^{*m*}) in which an *m*-bit vector of binary information corresponds to an element in GF(2^{*m*}). Let *g*(*x*) be an irreducible polynomial expressed as

$$g(x) = \sum_{i=0}^m g_i x^i, \quad (1)$$

where $g_0 = g_m = 1$ and $g_i \in \{0, 1\}$, for $i = 1, \dots, m - 1$, or an equivalent vector form $g = (g_m, \dots, g_1, g_0)$. Any element $a \in \text{GF}(2^m)$ can then be uniquely represented as a vector $a = (a_{m-1}, \dots, a_1, a_0)$ or in the polynomial form:

$$a(x) = \sum_{i=0}^{m-1} a_i x^i, \quad (2)$$

where $a_i \in \{0, 1\}$, for $i = 0, \dots, m - 1$.

For the basic arithmetic operations in GF(2^{*m*}), the addition is simply a bit-wise exclusive (XOR) operation, and the multiplication can be performed efficiently based on Horner's rule [18]. The modular division operation however is relatively complicated and deserves special design efforts. Assume *a*(*x*) and *b*(*x*) are two elements in GF(2^{*m*}). We review the division algorithm [14] to obtain $(a(x)/b(x))_g$ based on extended Stein's algorithm in section 2.2, and the general multiplication algorithm to obtain $(a(x)b(x))_g$ in section 2.3, where the notation $(c(x))_g$ denotes the operation $c(x) \bmod g(x)$.

2.2 Extended Stein's Division Algorithm

In [14], we extended Stein's algorithm for computing $v(x) = (a(x)/b(x))_g$ over GF(2^{*m*}) based on polynomial basis. The division algorithm is shown below.

In Algorithm *DA*, (1) *k* is used to count the number of iterations, which implies that the division algorithm converges in $2m - 1$ iterations; (2) r_0 denotes the LSB of polynomial *r*(*x*); (3) δ represents the difference of upper bounds on $\deg(r)$ and $\deg(s)$, where $\deg(c)$ denotes the degree of *c*(*x*); (4) $r(x)/x$ denotes the right shift operation, $r/x \equiv (0, r_{m-1}, \dots, r_1)$, which decreases the degree of polynomial *r*(*x*) by one; and (5) the operation $u(x) \leftarrow (u(x)/x)_g$ can be undertaken as follows [14]:

Division algorithm, DA.

Initialization:	$(r(x), s(x), u(x), v(x), \delta) \leftarrow (b(x), g(x), a(x), 0, -1)$
Result:	$v(x) = (a(x)/b(x))_g$
Algorithm:	<ol style="list-style-type: none"> 1. for $k = 1 : 2m - 1$ { 2. if $r_0 = 1$ { 3. if $\delta < 0$ { 4. $(r(x), s(x), u(x), v(x)) \leftarrow (r(x) + s(x), r(x), u(x) + v(x), u(x))$ 5. $\delta \leftarrow -\delta$ 6. } 7. else { 8. $(r(x), u(x)) \leftarrow (r(x) + s(x), u(x) + v(x))$ 9. } 10. } 11. $(r(x), u(x)) \leftarrow (r(x)/x, (u(x)/x)_g)$ 12. $\delta \leftarrow \delta - 1$ 13. }

$$u_{m-1} = u_0 g_m = u_0 \quad (3)$$

$$u_j = u_{j+1} + u_0 g_{j+1}, \quad 0 \leq j \leq m - 2. \quad (4)$$

2.3 Multiplication Algorithm

In general, the field multiplication operation, $t(x) = (a(x)b(x))_g$, can be expressed as

$$\begin{aligned}
 t(x) &= (a(x)b(x))_g \\
 &= (a(x)(b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \dots + b_1x + b_0))_g \\
 &= (((\dots(((a(x)b_{m-1}x)_g + a(x)b_{m-2}x)_g + \dots)x)_g + a(x)b_1x)_g + a(x)b_0.
 \end{aligned} \quad (5)$$

Depending on the coefficients of $b(x)$, the repeated operations involve multiplying $a(x)$ by x and then taking mod $g(x)$. The recursive equation can be expressed as follows [17]:

$$t(x) = (t(x) \cdot x)_g + b_{m-k} \cdot a(x), \quad k = 1, 2, \dots, m \quad (6)$$

where k is the number of iterations. The following gives a field multiplication algorithm, algorithm MA_1 , for performing the operation $u(x) = (a(x) \cdot b(x))_g$ over $\text{GF}(2^m)$.

Multiplication algorithm, MA_1.

Initialization:	$(r(x), v(x), u(x)) \leftarrow (b(x), a(x), 0)$
Result:	$u(x) = (a(x) \cdot b(x))_g$
Algorithm:	<ol style="list-style-type: none"> 1. for $k = 1 : m$ { 2. $u(x) \leftarrow (u(x) \cdot x)_g$ 3. if $r_{m-1} = 1$ { 4. $u(x) \leftarrow u(x) + v(x)$ 5. } 6. $r(x) \leftarrow r(x) \cdot x$ 7. }

In Algorithm *MA_1*, (1) r_{m-1} is the MSB (most significant bit) of polynomial $r(x)$; (2) it converges in m iterations; (3) $r(x) \cdot x$ denotes a left shift operation, $r \cdot x \equiv (r_{m-2}, \dots, r_0, 0)$, which increases the degree of polynomial $r(x)$ by one; and (4) the operation $u(x) \leftarrow (u(x) \cdot x)_g$ can be rewritten as

$$u_0 = u_{m-1}g_0 = u_{m-1} \tag{7}$$

$$u_j = u_{j-1} + u_{m-1}g_j, \quad 1 \leq j \leq m - 1. \tag{8}$$

2.4 Elliptic Curve Scalar Multiplication

The ECC scheme requires the point or scalar multiplication, which represents repeated addition of the same point defined as follows:

$$Q = kP = \underbrace{P + P + \dots + P}_k \tag{9}$$

where P denotes a point on an elliptic curve E and k is a random integer with $1 \leq k \leq \text{order}(P) \approx 2^m - 1$. The well-known double-and-add algorithm [1] below, also referred to as the binary method, is commonly used to compute kP .

Double-and-add algorithm, DAA.

Initialization:	$Q = O, k = (k_{m-1}, k_{m-2}, \dots, k_0); k_i \in \{0, 1\}$
Result:	$Q = kP$
Algorithm:	<ol style="list-style-type: none"> 1. for $i = m - 1: 0$ { <li style="padding-left: 20px;">2. $Q = Q + Q$ <li style="padding-left: 20px;">3. if $k_i = 1$ { <li style="padding-left: 40px;">4. $Q = Q + P$ <li style="padding-left: 20px;">5. } <li style="padding-left: 20px;">6. } 7. return Q

In Algorithm *DAA*, the initial value Q is assigned an infinite point denoted by the symbol O , and $m = \lceil \log_2 k \rceil$ denotes the number of bits of k . Since the point addition is executed only when k_i is 1; therefore computing a scalar multiplication requires $0.5m$ (m) point additions and m (m) point doubles in the average (worst) case. To reduce the non-zero coefficients of k , the non-adjacent form (NAF) is commonly applied to achieve an expected value of $m/3$ [22]. The NAF representation of k means that all its nonzero digits, e.g. 1 or -1 , are never adjacent to each other. Note that this work is independent of string encoding methods such as NAF method for making a fair comparison with related work. The full addition formulas for point addition ($Q + P$) and point double ($Q + Q$) in affine coordinates are defined as follows.

Assume that elliptic curve E is defined as $E: y^2 + xy = x^3 + ax^2 + b$ where a and b are elements in $\text{GF}(2^m)$ with $b \neq 0$ and $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are two points on the curve. When $Q \neq -P$ or equivalently $(x_2, y_2) \neq (x_1, x_1 + y_1)$, the full addition $P + Q = (x_3, y_3)$ can then be derived as given below:

Point Addition ($P \neq Q$)	Point Double ($P = Q$)
$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$	$\lambda = x_1 + \frac{y_1}{x_1}$
$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$	$x_3 = \lambda^2 + \lambda + a$
$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$	$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$

Since the field division and field multiplication are much more complicated than the field addition, the field addition could be ignored in complexity analysis. From the full addition formulas above, we observe that the full addition requires one division to obtain λ and two multiplications to compute λ^2 and $\lambda(x_1 + x_3)$ for x_3 and y_3 , respectively. Due to the data dependency existing in the full addition formulas, a possible schedule of point addition/double operations is drawn in Fig. 1. In this figure, the abbreviations *div* and *mul* denote the field division and multiplication operations, respectively. The numbers on top are to emulate the advance of time expressed in terms of the multiplication time assuming that the division and multiplication converge in $2m$ and m iterations as stated previously. The resulting hardware utilization would be only 50% if we adopt two distinct division and multiplication components to implement the ECC systems. In addition, completing one point addition/double takes $4m$ iterations. The required number of iterations for the scalar multiplication is thus about $6m^2$ ($= 0.5m \times 4m + m \times 4m$) in average case and $8m^2$ ($= m \times 4m + m \times 4m$) in worst case.

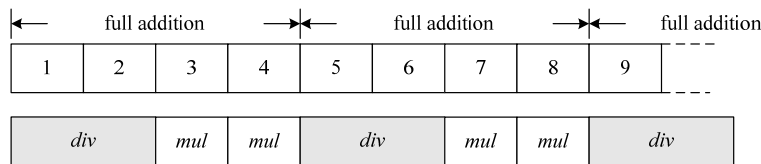


Fig. 1. Schedule of point multiplication operations.

3. OUR COMBINED MULTIPLICATION AND DIVISION ALGORITHM

This section addresses how to modify the traditional multiplication algorithm (MA_1) so that the multiplication operation can be effectively embedded into extended Stein’s division algorithm (DA). From our modification, a combined multiplier/divider (CMD) design can then be achieved with negligible speed degradation when used to design area-efficient ECC systems. Note that our CMD design cannot be extended to prime field because this work is based on the extended Stein’s algorithm [14], which can only operate over binary extension field.

3.1 Modified Multiplication Algorithm

From Algorithms DA and MA_1 , we observe that the DA includes the basic operations of MA_1 and the main difference between them is the opposite shift direction. To overcome the problem of different shift direction, we apply the concept of *reciprocal polynomial* to Algorithm MA_1 and derive a variant of the multiplication algorithm de-

noted as *MA_2*. In finite fields, the reciprocal polynomial $w^*(x)$ of a polynomial $w(x)$ of degree n is defined as $w^*(x) \equiv x^n \times w(1/x)$ [21]. Let $w(x) = w_{m-1}x^{m-1} + \dots + w_1x + w_0$ and $w^*(x) \equiv x^{m-1}w(x^{-1}) = w_0x^{m-1} + w_1x^{m-2} + \dots + w_{m-1}$. We summarize the analogy between $w(x)$ and $w^*(x)$ in Table 1.

Table 1. Analogy between $w(x)$ and $w^*(x)$.

$w(x)$	$w^*(x)$
$w = (w_{m-1}, \dots, w_1, w_0)$	$w^* = (w_0, w_1, \dots, w_{m-1})$
$(w \cdot x) = (w_{m-2}, \dots, w_1, w_0, 0)$	$(w^*/x) = (0, w_0, w_1, \dots, w_{m-2})$
$(w \cdot x)_g = (w_{m-2} + w_{m-1}g_{m-1}, \dots, w_0 + w_{m-1}g_1, w_{m-1})$	$(w^*/x)_{g^*} = (w_{m-1}, w_0 + w_{m-1}g_1, \dots, w_{m-2} + w_{m-1}g_{m-1})$

Note that $g^*(x) = x^m g(x^{-1}) = g_0x^m + g_1x^{m-1} + \dots + g_{m-1}x + 1$.

Based on Table 1, our multiplication algorithm, Algorithm *MA_2*, is presented below. To match the division algorithm *DA*, we move $u(x) \leftarrow (u(x)/x)_p$ to the end of each iteration; thereby the initial value of $v(x)$ becomes $(a^*(x) \cdot x)_p$. To reduce the design complexity, we set $v(x)$ to be an $(m + 1)$ -bit vector; thus the initial value of $v(x)$ can be simplified as $a^*(x) \cdot x$. Since Algorithm *MA_2* has the same shift direction as Algorithm *DA*, we can easily merge the arithmetic operations on variables r and u in two different algorithms by the following rules: If r_0 is 1, $r(x) = r(x)/x$ and $u(x) = ((u(x) + v(x))/x)_p$; otherwise $r(x) = r(x)/x$ and $u(x) = (u(x)/x)_p$.

Multiplication algorithm, *MA_2*.

Initialization:	$(r(x), v(x), u(x), p(x)) \leftarrow (b^*(x), (a^*(x) \cdot x)_{g^*}, 0, g^*(x))$
Result:	$u^*(x) = (a(x) \cdot b(x))_g$
Algorithm:	<ol style="list-style-type: none"> 1. for $k = 1: m$ { 2. if $r_0 = 1$ { 3. $u(x) \leftarrow u(x) + v(x)$ 4. } 5. $r(x) \leftarrow r(x)/x$ 6. $u(x) \leftarrow (u(x)/x)_p$ 7. }

3.2 The Developed *CMD* Algorithm

We merge Algorithms *DA* and *MA_2* to yield a combined multiplication/division algorithm expressed as follows.

Note that the result of multiplication, $u^*(x) = (a(x) \cdot b(x))_g$, is obtained after m iterations and the division, $v(x) = (a(x)/b(x))_g$, is accomplished in $2m - 1$ iterations. Since the division and multiplication take a different number of iterations, a new variable ρ is introduced to specify the iteration. A few comments are given for the multiplication operation in Algorithm *CMD*; (1) Since the condition $\delta \geq 0$ holds at each iteration, the multiplication terminates when $\delta = 0$; (2) The variable $s(x)$ remains zero ($s(x) = 0$) at each iteration; thus $s(x)$ does not affect the variable $r(x)$; (3) The value of $v(x)$ is fixed during the

Combined multiplication/division algorithm, *CMD*.

Initialization:	$mul: (r(x), s(x), u(x), v(x), p(x), \delta, \rho) \leftarrow (b^*(x), 0, 0, (a^*(x) \cdot x)_{g^*}, g^*(x), m, m)$ $div: (r(x), s(x), u(x), v(x), p(x), \delta, \rho) \leftarrow (b(x), g(x), a(x), 0, g(x), -1, 2m - 1)$
Result:	$mul: u^*(x) = (a(x) \cdot b(x))_g$ $div: v(x) = (a(x)/b(x))_g$
Algorithm:	<ol style="list-style-type: none"> 1. for $k = 1: \rho$ { 2. if $r_0 = 1$ { 3. if $\delta < 0$ { 4. $(r(x), s(x), u(x), v(x)) \leftarrow (r(x) + s(x), r(x), u(x) + v(x), u(x))$ 5. $\delta \leftarrow -\delta$ 6. } 7. else { 8. $(r(x), u(x)) \leftarrow (r(x) + s(x), u(x) + v(x))$ 9. } 10. } 11. $(r(x), u(x)) \leftarrow (r(x)/x, (u(x)/x)_p)$ 12. $\delta \leftarrow \delta - 1$ 13. }

Note: *mul* and *div* denote multiplication and division, respectively.

multiplication process. Note that except for different initial values required for performing either multiplication or division, the main parts of algorithms *CMD* and *DA* are the same. This implies that the multiplier can be combined with the divider using little multiplexing for operand selection. In summary, when the algorithm performs multiplication, the required initial values are given as follows: (1) r and p , respectively, become $b^* = (b_0, b_1, \dots, b_{m-1})$ and $g^* = (g_0, g_1, \dots, g_m)$; (2) $v(x)$ is initialized to $a^*(x) \cdot x$, i.e., $v = (a_0, a_1, \dots, a_{m-1}, 0)$; and (3) δ is set to m since there is no need to exchange the variables during the multiplication process.

3.3 Examples

Consider the modular division and multiplication over $\text{GF}(2^4)$. Let $g(x) = x^4 + x + 1$, $a(x) = x^3 + x^2 + 1$, and $b(x) = x^2 + x + 1$, i.e., $g = (10011)$, $a = (1101)$ and $b = (0111)$. Then we have $g^* = (11001)$, $b^* = (1110)$, and $a^* \cdot x = (10110)$ because $a^* = (1011)$. As shown in Table 2, Algorithm *CMD* takes seven steps to find the solution $v = (a/b)_g = (1000)$, corresponding to $v(x) = x^3$, and four steps to yield $u^*(x) = (a \cdot b)_g = (0101) = x^2 + 1$ since $u(x) = (1010)$. Note that $k = 0$ presents the initial values.

4. CMD DESIGN AND COMPLEXITY EVALUATION**4.1 Circuit Design**

Since the developed *CMD* algorithm stems from extended Stein's algorithm, the resulting circuit design can be derived from that based on Stein's algorithm. The remaining task is to deal with the different initial values required for performing either multiplication

Table 2. Iterations of algorithm CMD.

<i>k</i>	δ	<i>r</i>	<i>s</i>	<i>u</i>	<i>v</i>
Division					
0	-1	0111	1 0011	1101	00000
1	0	1010	0 0111	1111	01101
2	-1	0101	0 0111	1110	01101
3	0	0001	0 0101	1000	01110
4	-1	0010	0 0101	0011	01110
5	-2	0001	0 0101	1000	01110
6	1	0010	0 0001	0011	01000
7	0	0001	0 0001	1000	01000
Multiplication					
0	4	1110	0 0000	0000	10110
1	3	0111	0 0000	0000	10110
2	2	0011	0 0000	1011	10110
3	1	0001	0 0000	0010	10110
4	0	0000	0 0000	1010	10110

or division. Let $g(x)$ be the irreducible polynomial over GF(2^m) and $a(x), b(x), f(x), e(x)$ be four elements in GF(2^m). To obtain $(a(x)/b(x))_g$ and $(f(x)e(x))_g$, the initial inputs can be represented as follows:

$$r(x) = Ctrl5 \cdot b(x) + \overline{Ctrl5} \cdot e^*(x), \tag{10}$$

$$s(x) = Ctrl5 \cdot g(x), \tag{11}$$

$$u(x) = Ctrl5 \cdot a(x), \tag{12}$$

$$v(x) = \overline{Ctrl5} \cdot (f^*(x) \cdot x), \tag{13}$$

$$p(x) = Ctrl5 \cdot g(x) + \overline{Ctrl5} \cdot g^*(x), \tag{14}$$

where the control signal $Ctrl5 = 0$ (1) denotes the multiplication (division).

As examples, the CMD architectures of obtaining $c(x) = (f(x)e(x))_g$ and $d(x) = (a(x)/b(x))_g$ in GF(2^m) with $m = 5$ and $m = 6$ are depicted in Figs. 2 (a) and (b), respectively. According to these figures, the CMD circuit consists of $m + 1$ cells, including one G_A -cell, $\lfloor (m - 1)/2 \rfloor$ G_B -cells, $\lceil (m - 1)/2 \rceil$ H_B -cells, and one H_C -cell. In Fig. 2 (b), the G_B -cell for $j = m/2$ is removed since $m - m/2 = m/2$. Thus we just keep the H_B -cell for $m/2$. The symbol x in Fig. 2 (b), which denotes a don't-care signal, exists because the port of the H_B -cell is originally used to bypass a signal from the output of G_B -cell. Figs. 3-6 describe the implementation details of the four basic cells ($G_A, G_B, H_B,$ and H_C), where the small black rectangular represents a one-cycle delay element (flip-flop) for storing intermediate results to be used in the next iteration. In these figures, c_i^o and d_i^o represent the i th output of multiplication and division, respectively. The index j is used to denote the j th cell and the index n is defined as the value of $m - j$. To facilitate the bit stream reverse for implementing the reciprocal polynomial, the j th and $(m - j)$ th cells are allocated in the same

column. For convenience, the G_B -cell is used to denote the design of the j th cell for $j \leq \lfloor (m-1)/2 \rfloor$ and H_B -cell for $j > \lfloor (m-1)/2 \rfloor$. Except the bypass signals, the design of G_B -cell and H_B -cell is identical so that the architecture is regular and suitable for VLSI design.

In essence, the functions of A -cell (Fig. 3 (b)), B -cell (Fig. 4 (b) and Fig. 5 (b)), and C -cell (Fig. 6 (b)) are the same as our previous work [14], except for the gray-colored multiplexers (mux) used to select the input data. Specifically, the control signal $Ctrl1$ is used to control the gray-colored multiplexers for receiving input data ($Ctrl1 = 0$) in the first cycle, or iteration, and for latching intermediate results ($Ctrl1 = 1$) in the remaining cycles. The CMD array in Fig. 2 can output one multiplication (or division) result per m (or $2m - 1$) clock cycles. In the following, we describe the base cell designs of our CMD architecture.

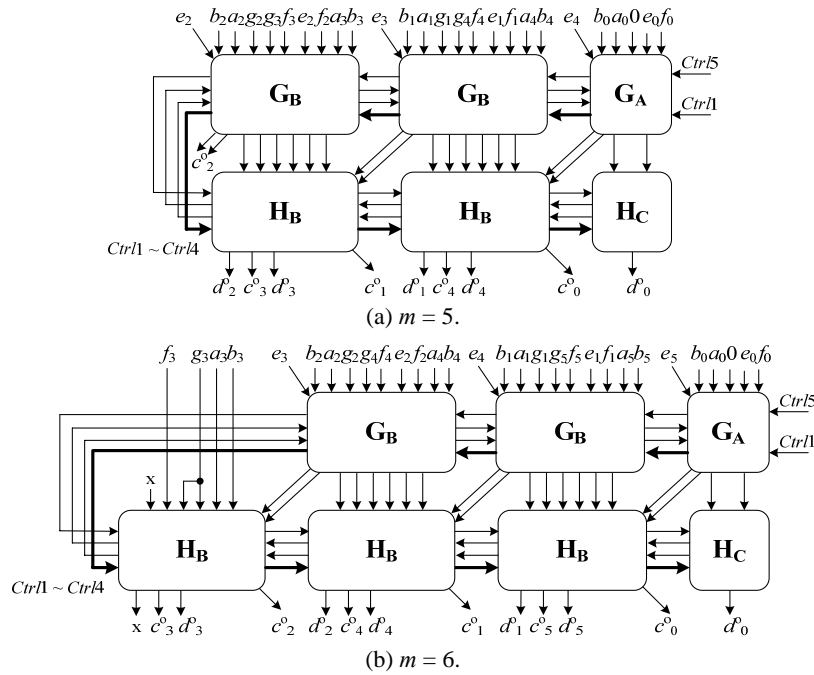


Fig. 2. Dependence graph of Algorithm CMD.

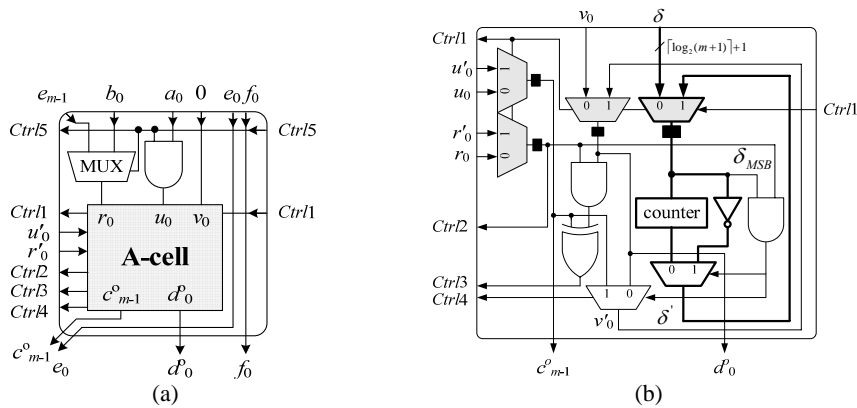


Fig. 3. (a) G_A -cell; (b) A -cell.

1. *A-cell*: As depicted in Fig. 3 (b), *A-cell* is responsible for updating the LSB of $v(v_0)$ and δ . Moreover, *A-cell* generates the controlling signals: $Ctrl2 \equiv r_0$, $Ctrl3 \equiv u_0 + r_0 \cdot v_0$, and $Ctrl4 \equiv (r_0 = 1) \& (\delta < 0)$ in each iteration. From algorithm *CMD*, we have the following observations. First, if r_0 is 1, $r(x) = (r(x) + s(x))/x$ and $u(x) = ((u(x) + v(x))/x)_p$; otherwise $r(x) = r(x)/x$ and $u(x) = (u(x)/x)_p$. Second, if $r_0 = 1$ and $\delta < 0$, then $s(x) = r(x)$, $v(x) = u(x)$, and $\delta = -\delta - 1$; otherwise $s(x) = s(x)$, $v(x) = v(x)$ and $\delta = \delta - 1$. $Ctrl3$ is the basic signal for computing $(u(x)/x)_p$ in Eqs. (3) and (4). We have $v'_0 = \overline{ctrl4} \cdot v_0 + ctrl4 \cdot u_0$. *A-cell* does not compute the LSBs of $s(x)$ and $r(x)$ since s_0 is always 1 and r_0 is shifted out. The 2's complement counter δ , which is embedded into *A-cell, has $(\lceil \log_2(m+1) \rceil + 1)$ bits and is updated as follows.*

$$\delta \leftarrow \overline{Ctrl4} \cdot (\delta - 1) + Ctrl4 \cdot (-\delta - 1). \quad (15)$$

When $Ctrl4 = 1$, $-\delta - 1 = (\overline{\delta} + 1) - 1 = \overline{\delta}$. Thus, $\delta' = \overline{Ctrl4} \cdot (\delta - 1) + Ctrl4 \cdot \overline{\delta}$. Note that determining the sign of δ is equivalent to checking MSB of δ , *i.e.*, its sign bit. Thus $Ctrl4 \equiv (r_0 = 1) \& (\delta_{MSB} = 1)$.

2. *B-cell and C-cell*: According to Fig. 4 (b) and Fig. 5 (b), the *B-cell* is designed to perform the necessary arithmetic operations on variables u' , v' , r' , and s' based on the control signals generated by *A-cell*. Taking into account the bit stream reverse for implementing the reciprocal polynomial, the design of *B-cell* in Fig. 5 (b) is a mirror image of that in Fig. 4 (b). The following equations can represent the updating of $r(x)$, $s(x)$, $u(x)$, $v(x)$.

$$(r, u) \leftarrow ((r + Ctrl2 \cdot s) / x, ((u + Ctrl2 \cdot v) / x)_p), \quad (16)$$

$$(s, v) \leftarrow (\overline{Ctrl4} \cdot s + Ctrl4 \cdot r, \overline{Ctrl4} \cdot v + Ctrl4 \cdot u), \quad (17)$$

in Eq. (16), the value of $r(x)$ is shifted right: $r'_{j-1} = r_j + Ctrl2 \cdot s_j$ and $u(x)$ is updated based on $Ctrl3$ and Eqs. (3) and (4): $u'_{j-1} = (u_j + Ctrl2 \cdot v_j) + Ctrl3 \cdot p_j$ for $1 \leq j \leq m-1$. The *C-cell*, depicted in Fig. 6 (b), is a simplified *B-cell* that computes the most significant bits $u'_{m-1} = Ctrl3 + Ctrl2 \cdot v_m$, $r'_{m-1} = Ctrl2 \cdot s_m$, and $s'_m = \overline{Ctrl4} \cdot s_m$.

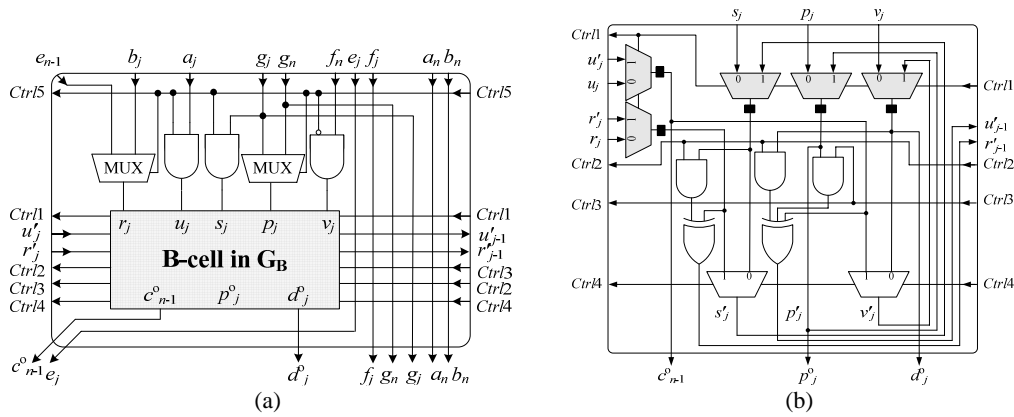
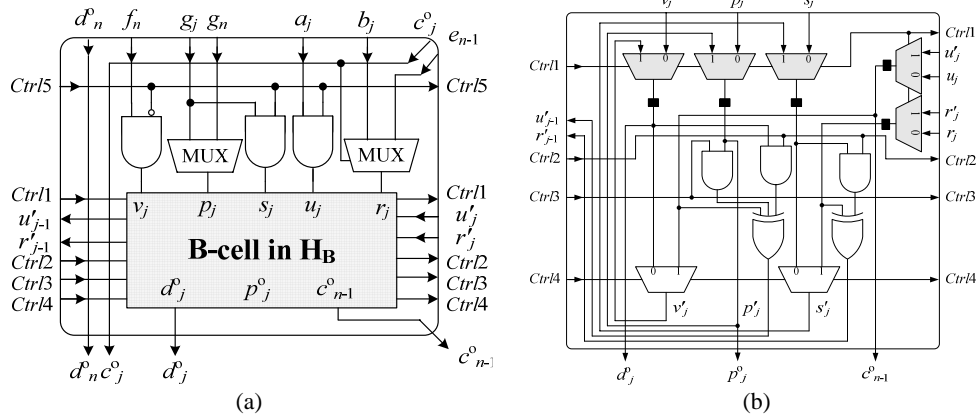
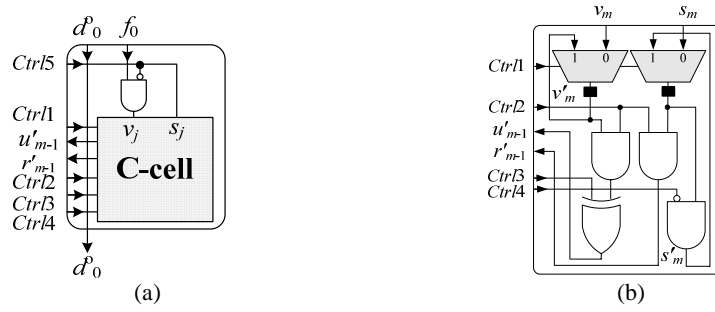


Fig. 4. (a) G_B -cell; (b) *B-cell* in G_B .

Fig. 5. (a) H_B -cell; (b) B -cell in H_B .Fig. 6. (a) H_C -cell; (b) C -cell.

4.2 Complexity Analysis of CMD Circuit

Compared with the divider design in [14], the hardware overhead of our *CMD* design contains $2m$ 2-input multiplexers and $3m$ 2-input AND gates. That is, the hardware overhead is estimated as $3m \times A_{AND2} + 2m \times A_{MUX2}$, where the notations A_{AND2} and A_{MUX2} denote the area of 2-input AND gate and 2-input multiplexer, respectively. From [14], when m is large enough, the area of divider is approximated to $A_{div} = 3m \times A_{AND2} + 3m \times A_{XOR2} + 7m \times A_{MUX2} + 5m \times A_{FF}$, where A_{FF} denotes the area of flip-flop, and we use two 2-input XOR gates to represent one 3-input XOR gate. From [17], the area of multiplier is $A_{mul} = 2m \times A_{AND2} + 2m \times A_{XOR2} + 3m \times A_{MUX2} + 3m \times A_{FF}$. Define the area reduction ratio (ARR) of our *CMD* design as

$$\frac{(A_{div} + A_{mul}) - (A_{div} + A_{over})}{A_{div} + A_{mul}} = \frac{A_{mul} - A_{over}}{A_{div} + A_{mul}}, \quad (18)$$

where A_{over} denotes the resulting hardware overhead of our *CMD* design. By the UMC $0.18 \mu\text{m}$ technology, we have $A_{AND2} \approx 13.30 \mu\text{m}^2$, $A_{XOR2} \approx 26.61 \mu\text{m}^2$, $A_{MUX2} \approx 29.94 \mu\text{m}^2$, and $A_{FF} \approx 79.83 \mu\text{m}^2$. Using the information, we compute $A_{div} \approx 728.4m \mu\text{m}^2$, $A_{mul} \approx 409.1m \mu\text{m}^2$ and $A_{over} \approx 99.8m \mu\text{m}^2$; therefore the resulting area reduction ratio is about 27.2%.

5. COMPLEXITY ANALYSIS FOR DIFFERENT ECC APPLICATIONS

5.1 Area Complexity of ECC Systems Using CMD Design

One of the applications of our *CMD* algorithm is used to compute points in ECC over GF(2^m) using affine coordinates. Because of the inherent data dependency of the field arithmetic in ECC systems using affine coordinates, the division and multiplication are sequentially performed, and it is well suited to implement both operations in the same hardware. In this way, we can increase the hardware utilization as well as decrease the hardware requirement.

Because the field division and multiplication are much more complicated than the field addition, the field addition is ignored in our complexity analysis. Furthermore, as described in section 2.4, we need one divider, one multiplier and six registers to accomplish the scalar multiplication for ECC systems using affine coordinates. By Eq. (18), the ARR of ECC systems employing our *CMD* design can be defined as

$$\frac{A_{ECC2} - A_{ECC1}}{A_{ECC2}} = \frac{A_{mul} - A_{over}}{A_{ECC1} + (A_{mul} - A_{over})}, \quad (19)$$

where A_{ECC1} and A_{ECC2} denote the area of ECC systems using our *CMD* design and that employing distinct multiplier and divider, respectively. Our sample design takes $A_{ECC1} \approx 1928.1m \mu m^2$ based on the UMC 0.18 μm technology; therefore the resulting area reduction ratio is about 13.8%.

5.2 Time Complexity of ECC Systems Using CMD Design

As described in [22], assuming $m = 163$ and $b = 1$ for the Koblitz elliptic curve, the estimated computation time of executing the scalar multiplication kP in two different coordinates is listed in Table 3. In this table, (1) *Add* and *Double*, respectively, denote the total number of required point addition and point doubling; (2) *mul*, *div*, *inv* and *sqr*, respectively, represent the number of multiplication, division, inversion and squaring operations used to accomplish the scalar multiplication; (3) α is the ratio of critical path delay of our *CMD* design to that of the selected multiplier; and (4) N_F is defined as the equivalent number of multiplications required to carry out the scalar multiplication as explained below.

Table 3. Computation time comparison in two different coordinates.

	α	Point Operation		Field Operation				N_F	Reduction ratio
		<i>Add</i>	<i>Double</i>	<i>mul</i>	<i>div</i>	<i>inv</i>	<i>sqr</i>		
Projective	1	82	162	1304	0	1	1220	2847	–
Affine	1.50 ⁽¹⁾	82	162	244	244	0	244	1464	48.6%
	2.04 ⁽²⁾	82	162	244	244	0	244	1991	30.1%

(1) The time ratio computed using the polynomial basis MSB-first multiplier.

(2) The time ratio computed using the polynomial basis LSB-first multiplier.

From Figs. 3-6, the critical path delay of our *CMD* design is approximated to $T_{CMD} = 2T_{AND2} + 3T_{XOR2} + T_{MUX2}$, which is about 0.849 ns based on UMC 0.18 μm technology. As for the multipliers, the critical path delays of the polynomial basis MSB-first [17] and LSB-first [18] multipliers, respectively, are estimated as $T_{Mul-MSB} = 2T_{XOR2} + T_{AND2} + T_{MUX2}$ and $T_{Mul-LSB} = T_{XOR2} + T_{AND2} + T_{MUX2}$, which are about 0.567 ns and 0.416 ns based on the same technology. Thus, the time ratio α is approximated to 1.50 ($= T_{CMD}/T_{mul-MSB} = 0.849/0.567$) and 2.04 ($= T_{CMD}/T_{mul-LSB} = 0.849/0.416$) for two different types of chosen multipliers.

In our analysis, we replace division with multiplication and use $\alpha = 1$ for ECC systems using projective coordinates assuming that the inversion of a field element is found by using Fermat's theorem that requires $m - 2$ multiplications and $m - 1$ squaring. It is further assumed that multiplication and squaring have the same time complexity because they are generally accomplished by a single multiplier [17, 18], although the squaring can be optimized for some specific polynomials like a trinomial [23]. Note that our analysis can be easily extended to deal with the cases which distinguish the squaring from the multiplication.

Remind that the number of iterations of division is twice than that of multiplication. Thus, the values of N_F can be estimated as $1 \times (mul + sqr + (m - 2 + m - 1) \times inv) = 2847$ in projective coordinates and $\alpha \times (mul + sqr + 2 \times div) = 1464$ (1991.04) in affine coordinates using the MSB-first (LSB-first) multiplier. From Table 3, we can see that the ECC system using affine coordinates and employing our *CMD* design can save up to $((2847 - 1464)/2847) \times 100\% = 48.6\%$ of computation time compared with that using projective coordinates. Note that trade-offs between different types of multiplier and divider designs may yield different results in both coordinates.

6. IMPLEMENTATION RESULTS AND COMPARISONS

6.1 Implementation Results

An ECC system over $GF(2^{191})$ using the developed *CMD* design has been successfully realized by the Verilog language and synthesized by Synopsys Design-CompilerTM based on the UMC 0.18 μm library. The system, verified by IEEE *std.* 1363 golden patterns, can operate higher than 200 MHz with gate counts of about 36.8k. Note that the gate counts are computed in terms of the equivalent number of 2-input NAND gates. As described in section 2.4, the required number of iterations for the scalar multiplication is approximated to $8m^2$ in the worst case and $6m^2$ in the average case.

6.2 Comparison with Other Low-Cost *CMD* Design

In Table 4, we compare the area complexity of our *CMD* design with that of the low-cost finite field arithmetic design in Kim's work [19] for $m = 64, 128, 160, \text{ and } 192$. Kim combined the variant Euclidean's algorithm for division and LSB-first multiplication algorithm. In fact, the divider based on variant Euclidean's algorithm involves two different shift directions for $(u(x)/x)_g$ and $(u(x)x)_g$ [15]. Thus the two algorithms can be easily combined since the divider has the datapath that a multiplier requires. However, it implies the

divider designed based on variant Euclidean's algorithm has more complex datapath and control than that based on Stein's algorithm. Therefore, the latter has both area and time advantages [14, 24]. As mentioned above, we merge the multiplier into the divider based on extended Stein's algorithm with little area overhead for operand selection and almost no performance degradation. Thus, up to 12.7% of area reduction can be achieved by using our *CMD* design as seen from Table 4. Note that because Kim's results were synthesized with a CMOS 0.35 μm library and ours with UMC 0.18 μm library, we adopt the gate counts to make a fair comparison.

Table 4. Comparison of the total gate counts for *CMD* designs.

m	Kim <i>et al.</i> [19] (R_1)	Ours (R_2)	Reduction ratio ($R_1 - R_2$)/ R_1
64	6,011	5,248	12.7%
128	11,638	10,245	12%
160	14,434	12,697	12%
192	16,847	15,174	9.9%

6.3 Comparison with Other ECC Designs in FPGA

In this subsection, we show the experimental results of our ECC design in different FPGA devices and make a comparison with existing ECC designs over GF(2^m) [8, 11, 20]. The ECC systems in [8, 11] achieve high throughput at the price of high area overhead because the set of ONB (optimized normal basis) multipliers is executed in parallel. Park [20] implemented an ECC processor using the modified Booth algorithm and polynomial basis for an embedded medical image system that demands fast operation and low hardware resource. Table 5 compiles the comparisons between our design and the three works when $m = 113$ or 163. In this table, (1) the area is computed in terms of the number of occupied slices – a unit used in Xilinx Virtex series; (2) the time complexity is the time required to accomplish a scalar multiplication; (3) the “efficiency” is defined as the executed scalar multiplications per second per slice, *i.e.* efficiency $\equiv 1000/AT$, where $AT = \text{Area (slices)} \times \text{Time (ms)}$; and (4) the AT reduction ratio is expressed as $(AT_1 - AT_2)/AT_1$, where AT_1 and AT_2 denote the AT complexity of other's work and ours, respectively.

In [8], the authors introduced a parallelism factor k to take different multiplier designs into consideration. A k -way multiplier means that the number of cycles required for accomplishing a multiplication is reduced to $\lfloor m/k \rfloor + 2$. As shown in [8], the performance improves as we increase the parallelism factor k and then quickly saturates with $k = 14$. For the case of $k = 14$, our work still has 24.6% improvement in AT complexity. Finally, the symbol FM defined in [11] represents the number of employed field multipliers. Note that Cheung [11] reported the best AT complexity of their ECC without providing related time and area information for their designs. Thus, we only adopted the best AT complexity of [11] to compare with ours. From the comparisons below, the ECC system using the proposed *CMD* circuit yields the best performance.

Table 5. Comparison between our ECC design and the related work.

m	FPGA devices	Work	Area (slices)	Time (ms)	Efficiency	AT reduction
113	XCV1000	[8]	1,410 ($k = 1$)	4.3	0.16	73.7%
			2,515 ($k = 14$)	0.84	0.47	24.6%
			8,753 ($k = 113$)	0.75	0.15	75.7%
		Ours	1,810	0.88	0.63	–
	XC2V6000	[11]	N/A	N/A	0.77 ($FM = 1$)	23.9%
			N/A	N/A	0.71 ($FM = 2$)	29.8%
			N/A	N/A	0.70 ($FM = 4$)	30.8%
Ours		1,647	0.6	1.01	–	
163	XCV1000E	[20]	3,600	3.05	0.09	42.2%
		Ours	2,490	2.55	0.16	–

7. CONCLUSION

In this paper, we presented a combined multiplication/division (*CMD*) algorithm based on extended Stein's algorithm. From our development, an efficient *CMD* module can then be achieved with little multiplexing for operand selection. The resulting circuit design features modularity, regularity, and concurrency so that it is well suited for VLSI implementation. In fact, our *CMD* design not only is area-efficient but also can be operated in very high clock rate. Thus, when applying the developed *CMD* design to ECC systems using affine coordinates, we can achieve both the area and speed advantages. The area reduction comes from the improved hardware utilization in the target ECC in which the multiplication and division operations are sequentially executed. Experimental results demonstrated the effectiveness of our development in comparison with the previous work. Finally, our evaluation results also show that the ECC systems using affine coordinates and our *CMD* design may outperform those using projective coordinates.

REFERENCES

1. IEEE Std-1363-2000: Standard specifications for public key cryptography, 2000.
2. ISO/IEC 14888-3, Information technology – Security techniques – Digital signatures with appendix – Part 3: Certificate based-mechanisms, 1998.
3. ISO/IEC 15946-4, Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 4: Digital signatures giving message recovery, 2004.
4. ANSI X9.38, Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm, 1999.
5. National Institute of Standards and Technology, Digital Signature Standard, FIPS Publication 186-2, 2000.
6. Certicom Corporation, “The basics of ECC 2006,” http://www.certicom.com/index.php?action=res,ecc_faq.
7. M. Bednara, M. Daldrup, J. Gathen, J. Shokrollahi, and J. Teich, “Reconfigurable

- implementation of elliptic curve crypto algorithms,” in *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, 2002, pp. 157-164.
8. P. H. W. Leong and I. K. H. Leung, “A microcoded elliptic curve processor using FPGA technology,” *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 10, 2002, pp. 550-559.
 9. M. Morales-Sandoval and C. Feregrino-Urbe, “On the hardware design of an elliptic curve cryptosystem,” in *Proceedings of the 5th IEEE Mexican International Conference*, 2004, pp. 64-70.
 10. Y. Eslami, A. Sheikholeslami, P. G. Gulak, S. Masui, and K. Mukaida, “An area-efficient universal cryptography processor for smart cards,” *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 14, 2006, pp. 43-56.
 11. R. C. C. Cheung, N. J. Telle, W. Luk, and P. Y. K. Cheung, “Customizable elliptic curve cryptosystems,” *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 13, 2005, pp. 1048-1059.
 12. N. A. Saqib, F. Rodriguez-Henriquez, and A. Diaz-Perez, “A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2^m),” in *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium*, 2004, pp. 141-148.
 13. S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, Inc., NJ, 1995.
 14. C. H. Wu, C. M. Wu, M. D. Shieh, and Y. T. Hwang, “High-speed, low-complexity systolic designs of novel iterative division algorithms in GF(2^m),” *IEEE Transactions on Computers*, Vol. 53, 2004, pp. 375-380.
 15. H. Brunner, A. Curiger, and M. Hofstetter, “On computing multiplicative inverses in GF(2^m),” *IEEE Transactions on Computers*, Vol. 42, 1993, pp. 1010-1015.
 16. M. D. Shieh, J. H. Chen, and C. M. Wu, “High-speed design of montgomery inverse algorithm over GF(2^m),” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 89, 2006, pp. 559-565.
 17. C. L. Wang and J. L. Lin, “Systolic array implementation of multipliers for finite fields GF(2^m),” *IEEE Transactions on Circuits and Systems*, Vol. 38, 1991, pp. 796-800.
 18. S. K. Jain, L. Song, and K. K. Parhi, “Efficient semisystolic architectures for finite-field arithmetic,” *IEEE Transactions on Very Large Scale Integral Systems*, Vol. 6, 1998, pp. 101-113.
 19. J. H. Kim and D. H. Lee, “A compact finite field processor over GF(2^m) for elliptic curve cryptography,” in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2002, pp. II-340-II-343.
 20. J. Park, J. T. Hwang, and Y. C. Kim, “FPGA and ASIC implementation of ECC processor for security on medical embedded system,” in *Proceedings of the 3rd IEEE International Conference on Information Technology and Applications*, 2005, pp. 547-551.
 21. W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, MIT Press, Cambridge, MA, 1972.
 22. E. Al-Daoud, R. Mahmood, M. Rushdan, and A. Kilicman, “A new addition formula for elliptic curves over GF(2ⁿ),” *IEEE Transactions on Computers*, Vol. 51, 2002, pp. 972-975.

23. H. Wu, "Bit-parallel finite field multiplier and squarer using polynomial basis," *IEEE Transactions on Computers*, Vol. 51, 2002, pp. 750-758.
24. C. H. Kim and C. P. Hong, "High-speed division architecture for $GF(2^m)$," *IEE Electronic Letters*, Vol. 38, 2002, pp. 835-836.



Ming-Der Shieh (謝明得) received the B.S. degree in Electrical Engineering from National Cheng Kung University, in 1984, the M.S. degree in Electronic Engineering from National Chiao Tung University, Taiwan, in 1986, and the Ph.D. degree in Electrical Engineering from Michigan State University, East Lansing, in 1993. From 1988 to 1989, he was an engineer at United Microelectronic Corporation, Taiwan. From 1993 to 2002, he was with the faculty of Department of Electronic Engineering, National Yunlin University of Science and Technology (NYUST). He received the teaching award from NYUST in 1998 and was the department chairman from 1999 to 2002. Since 2002, he has been with the Department of Electrical Engineering, National Cheng Kung University, where he is currently an Associate Professor. His research interests include VLSI design and testing, VLSI for signal processing, digital communication, and computer-aided design.



Jun-Hong Chen (陳俊宏) received the B.S. and M.S. degrees in Electronic Engineering from National Yunlin University of Science and Technology, Taiwan, in 2001 and 2003 respectively. He is pursuing his Ph.D. degree in National Cheng Kung University, Taiwan, since 2003. His primary research areas were VLSI implementation in digital signal processing architectures, data security, digital communication and reconfigurable architectures design.



Wen-Ching Lin (林文景) received the B.S. and M.S. degrees in Electrical Engineering from National Cheng Kung University, Taiwan, in 2005 and 2007 respectively. He is pursuing his Ph.D. degree in National Cheng Kung University, Taiwan, since 2007. His primary research areas were VLSI design and architecture, cryptography and finite field theory.



Chien-Ming Wu (吳建明) obtained his B.S. and M.S. degrees, both in Electronic Engineering, from National Yunlin University of Science and Technology, Taiwan, in 1997 and 1999, respectively. He received the Ph.D. degree from the Graduate School of Engineering Science and Technology at National Yunlin University of Science and Technology, Taiwan, in 2003. He is currently an associate researcher at National Chip Implementation Center (CIC), Taiwan. His research interests include VLSI design in communication, coding theory, and digital signal processing.