ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.14, No.12, 1–10

Efficient unified semi-systolic arrays for multiplication and squaring over $GF(2^m)$

Kee-Won Kim[1] **and Jae-Dong Lee**[2a)]

[1] *Department of Applied Computer Engineering, Dankook University, Yongin, Korea*

[2] *Department of Software Science, Dankook University, Yongin, Korea*

a) *letsdoit@dankook.ac.kr, Corresponding Author*

Abstract: In this paper, we propose a unified algorithm to concurrently perform multiplication and squaring over $GF(2^m)$ using the bipartite modular multiplication method and deriving common operations. Also we design efficient unified semi-systolic arrays from our proposed algorithm for fast exponentiation. The proposed arrays can be used as a core circuit for various applications. Also our architectures are well suited to VLSI implementation as well.

Keywords: finite field, Montgomery multiplication, squaring, systolic array, cryptography

Classification: Integrated circuits

References

[1] R. E. Blahut: *Theory and Practice of Error Control Codes* (Addison-Wesley, Reading, MA, 1983).

[2] A. J. Menezes, *et al.*: *Handbook of Applied Cryptography* (CRC Press, Boca Raton, FL, 1996).

[3] C. W. Chiou, *et al.*: "Concurrent error detection in Montgomery multiplication over $GF(2^m)$," IEICE Trans. Fundamentals **E89-A** (2006) 566 (DOI: 10.1093/ietfec/e89-a.2.566).

[4] W. T. Huang, *et al.*: "Concurrent error detection and correction in a polynomial basis multiplier over $GF(2^m)$," IET Inf. Secur. **4** (2010) 111 (DOI: 10.1049/iet-ifs.2009.0160).

[5] K. W. Kim and J. C. Jeon: "Polynomial basis multiplier using cellular systolic architecture," J. Inst. Electron. Telecommun. Eng. **60** (2014) 194 (DOI: 10.1080/03772063.2014.914699).

[6] S. H. Choi and K. J. Lee: "Efficient systolic modular multiplier/squarer for fast exponentiation over $GF(2^m)$," IEICE Electron. Express **12** (2015) 20150222 (DOI: 10.1587/elex.12.20150222).

[7] K. W. Kim and J. C. Jeon: "A semi-systolic Montgomery multiplier over $GF(2^m)$," IEICE Electron. Express **12** (2015) 20150769 (DOI: 10.1587/elex.12.20150769).

[8] P. Montgomery: "Modular multiplication without trial division," Math. Comput. **44** (1985) 519 (DOI: 10.1090/S0025-5718-1985-0777282-X).

[9] C. Koc and T. Acar: "Montgomery multiplication in $GF(2^k)$," Des. Codes Cryptogr. **14** (1998) 57 (DOI: 10.1023/A:1008208521515).

[10] M. E. Kaihara and N. Takagi: "Bipartite modular multiplication," CHES **3659** (2005) 201 (DOI: 10.1007/11545262_15).

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

[11] M. E. Kaihara and N. Takagi: "Bipartite modular multiplication method," IEEE Trans. Comput. **57** (2008) 157 (DOI: 10.1109/TC.2007.70793).
[12] P. A. Scott, *et al.*: "Architectures for exponentiation in $GF(2^m)$," IEEE J. Sel. Areas Commun. **6** (1988) 578 (DOI: 10.1109/49.1927).
[13] K. J. Lee and K. Y. Yoo: "Linear systolic multiplier/squarer for fast exponentiation," Inf. Process. Lett. **76** (2000) 105 (DOI: 10.1016/S0020-0190(00)00131-9).
[14] J. C. Ha and S. J. Moon: "A common-multiplicand method to the Montgomery algorithm for speeding up exponentiation," Inf. Process. Lett. **66** (1998) 105 (DOI: 10.1016/S0020-0190(98)00031-3).
[15] D. E. Knuth: *The Art of Computer Programming, Seminumerical Algorithms*, vol. II (Addison-Wesley, MA, 1997).

1 Introduction

The arithmetic operations over finite field $GF(2^m)$ are very important for many practical applications in error-correcting codes and cryptography [1, 2]. The multiplication is one of the basic arithmetic operations over finite fields. Modular exponentiation and inversion can be performed using a sequence of multiplications. Although many multiplier over $GF(2^m)$ have been proposed [3, 4, 5, 6, 7], their high space and time complexities are major limitations in various applications.

The Montgomery multiplication algorithm has been proposed for fast modular integer multiplication [8]. The Montgomery multiplication was successfully adapted to $GF(2^m)$ in [9]. An approach based on Montgomery multiplication which allows one to split the operand into two parts, which can be processed in parallel, is called a bipartite modular multiplication and is introduced in [10, 11]. They focused on application in the integer and mentioned the usability of their method in the finite field. Thereafter, Kim and Jeon adapted their bipartite technique to the modular multiplication in the finite fields for reducing the latency and proposed multipliers over $GF(2^m)$ [5, 7].

The modular exponentiation is an essential part of cryptographic algorithms. It is typically performed using a series of modular squaring and multiplication based on the binary method. There are two modular exponentiation schemes: LSB (least significant bit)-first and MSB (most significant bit)-first modular exponentiation, where LSB and MSB refer to the LSB and MSB of the exponent. For fast LSB-first modular exponentiation, modular squaring and multiplication can be performed simultaneously [12, 13, 14]. Using the common operations from multiplication and squaring, Choi and Lee [6] proposed the combined systolic multiplier/squarer that computes modular squaring and multiplication concurrently instead of computing them separately for the LSB-first exponentiation.

In this paper, we propose a unified algorithm to concurrently perform multiplication and squaring over $GF(2^m)$ using the bipartite modular multiplication method and deriving common operations. Then we design efficient unified semi-systolic arrays to concurrently compute multiplication and squaring over $GF(2^m)$ from our proposed algorithm for fast exponentiation.

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

2 Preliminaries

2.1 Montgomery multiplication

$GF(2^m)$ is a kind of finite field that contains 2^m different elements. This finite field is an extension of $GF(2)$ and any element over $GF(2^m)$ can be represented as a polynomial of degree $m-1$ over $GF(2)$. Let x be a root of the polynomial, then the irreducible polynomial G is represented as $G = \sum_{j=0}^{m} g_j x^j$, where $g_j \in GF(2)$.

Let α and β be two elements of $GF(2^m)$, then we define $\gamma = \alpha \cdot \beta \bmod G$. Also, let A and B be two Montgomery residues, then they are defined as $A = \alpha \cdot F \bmod G = \sum_{j=0}^{m-1} a_j x^j$ and $B = \beta \cdot F \bmod G = \sum_{j=0}^{m-1} b_j x^j$, where a Montgomery factor, F and an irreducible polynomial, G are relatively prime, and $gcd(F, G) = 1$. Then, the Montgomery multiplication algorithm over $GF(2^m)$ can be formulated as $P = A \cdot B \cdot F^{-1} \bmod G$, where F^{-1} is the inverse of F modulo G. Then, P can be expressed as $P = (\alpha \cdot F) \cdot (\beta \cdot F) \cdot F^{-1} \bmod G = \alpha \cdot \beta \cdot F \bmod G$.

2.2 Bipartite modular multiplication

Kaihara and Takagi first have proposed a bipartite modular multiplication using Montgomery algorithm and then they have extended their method [10, 11]. It splits the operand multiplier into two parts that can be processed simultaneously to increasing the calculation speed. We briefly review the main idea of their method.

Let the modulus M be an n-digit integer, where the radix of each digit is $r = 2^t$ and let a Montgomery radix $F = r^k$ where $0 \le k \le n$. Consider the multiplier Y to be split into two parts Y_H and Y_L so that $Y = Y_H R + Y_L$. Then, the Montgomery multiplication modulo M of the integers X and Y can be computed as follows:

$$\begin{aligned} X * Y &= XYF^{-1} \bmod M \\ &= X(Y_H F + Y_L)F^{-1} \bmod M \\ &= ((XY_H \bmod M) + (XY_L F^{-1} \bmod M)) \bmod M. \end{aligned}$$

The left term of the last equation, $XY_H \bmod M$, can be calculated using the classical modular multiplication that processes the upper part of the split multiplier Y_H. The right term, $XY_L F^{-1} \bmod M$, can be calculated using the Montgomery algorithm that processes the lower part of the split multiplier Y_L. Both calculations can be processed simultaneously. Since the split operands Y_H and Y_L are shorter in length than Y, the calculations $XY_H \bmod M$ and $XY_L F^{-1} \bmod M$ are performed faster than $XYF^{-1} \bmod M$.

They have focused on application in the integer and mentioned the usability of their method in the finite field. Thereafter, Kim and Jeon [5, 7] adapted their bipartite technique to the modular multiplication in the finite fields for reducing the latency and proposed multipliers over $GF(2^m)$.

2.3 Modular exponentiation

The exponentiation is a crucial part of modern cryptographic algorithms. The most commonly used algorithms for exponentiation are the binary methods (also called square-and-multiply methods) [15]. Its basic idea is to compute modular exponentiation by using the binary expression of exponent E and the exponentiation operation is broken into a series of squaring and multiplication operations. This

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

algorithm has the left-to-right (MSB) method and right-to-left method (LSB). The right-to-left method can be used to compute modular squaring and modular multiplication concurrently. The right-to-left binary square and multiply algorithm is represented as Algorithm 1 which computes the modular exponentiation starting from the LSB of the exponent and proceeding to the left.

Algorithm 1. LSB binary modular exponentiation algorithm in $GF(2^m)$
Input: M, $E = \sum_{i=0}^{m-1} e_i 2^i$ (where $e^i \in \{0, 1\}$), G
Output: $C = M^E \bmod G$
Step 1. $C = 1$;
Step 2. $S = M$;
Step 3. for $i = 0$ to $m - 1$ do {
Step 4. if $(e_i = 1)$ then $C = C \times S \bmod G$;
Step 5. $S = S \times S \bmod G$;
Step 6. }

Note that modular squaring and multiplication can be performed simultaneously in order to improve speed of exponentiation [6, 12, 13, 14]. Using the common-multiplicand method [13, 14], Choi and Lee [6] proposed the combined systolic multiplier/squarer that computes modular squaring and multiplication concurrently instead of computing them separately for the LSB-first exponentiation.

3 Proposed unified multiplication and squaring

In this section, we propose a unified algorithm to concurrently perform multiplication and squaring over $GF(2^m)$. We adopt the bipartite modular multiplication concept [10, 11, 5, 7] to decrease the latency required for calculating multiplication and squaring over finite fields. Using common-multiplicand method [6, 13, 14], we also decrease the space complexity by deriving common operations from bipartite parts.

Now, we will derive a bipartite algorithm for performing multiplication and squaring over $GF(2^m)$ in parallel. It is well known that $x^m \bmod G = \sum_{j=0}^{m-1} g_j x^j$ and $x^{-1} \bmod G = x^{m-1} + \sum_{j=1}^{m-1} g_j x^{j-1}$. Let $k = \lfloor m/2 \rfloor$ and $l = \lceil m/2 \rceil$. For deriving an efficient parallel architecture, we choose the Montgomery factor, $F = x^{\lfloor m/2 \rfloor} = x^k$ as selected in [5, 7]. Then, we can derive the following formula for the Montgomery multiplication P and squaring S over $GF(2^m)$.

$$\begin{aligned} P &= A \cdot B \cdot F^{-1} \bmod G = A \cdot B \cdot x^{-k} \bmod G \\ &= [b_0 A x^{-k} + b_1 A x^{-k+1} + \cdots + b_{k-2} A x^{-2} + b_{k-1} A x^{-1} \\ &\quad + b_k A + b_{k+1} A x^1 + \cdots + b_{m-2} A x^{-k+m-2} + b_{m-1} A x^{-k+m-1}] \bmod G \\ &= \sum_{i=1}^{k} b_{k-i} A x^{-i} \bmod G + \sum_{i=0}^{l-1} b_{k+i} A x^i \bmod G. \end{aligned} \tag{1}$$

$$\begin{aligned} S &= A \cdot A \cdot F^{-1} \bmod G = A \cdot A \cdot x^{-k} \bmod G \\ &= [a_0 A x^{-k} + a_1 A x^{-k+1} + \cdots + a_{k-2} A x^{-2} + a_{k-1} A x^{-1} \\ &\quad + a_k A + a_{k+1} A x^1 + \cdots + a_{m-2} A x^{-k+m-2} + a_{m-1} A x^{-k+m-1}] \bmod G \end{aligned}$$

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

$$= \sum_{i=1}^{k} a_{k-i} A x^{-i} \bmod G + \sum_{i=0}^{l-1} a_{k+i} A x^{i} \bmod G. \tag{2}$$

As seen in (1) and (2), the multiplication result P and squaring result S can be divided into two parts, respectively. One is based on the negative powers of x and the other is based on the positive powers of x. P can be denoted by $P = Q + R$, where $Q = \sum_{i=1}^{k} b_{k-i} A x^{-i} \bmod G$ and $R = \sum_{i=0}^{l-1} b_{k+i} A x^{i} \bmod G$. Similarly, S can be denoted by $S = T + U$, where $T = \sum_{i=1}^{k} a_{k-i} A x^{-i} \bmod G$ and $U = \sum_{i=0}^{l-1} a_{k+i} A x^{i} \bmod G$.

If Montgomery multiplication and squaring are executed concurrently, the components for the common operations can be shared and used only once for Montgomery multiplication and squaring in order to reduce the area complexity. We can derive the common operations, $Ax^{-i} \bmod G$ and $Ax^{i} \bmod G$, in each bipartite equation from (1) and (2).

For the derived common operations, we define $\bar{A}^{(i)} = Ax^{-i} \bmod G$ $(1 \le i \le k)$ and $A^{(i)} = Ax^{i} \bmod G$ $(0 \le i \le l-1)$. Then the equations can be expressed as $\bar{A}^{(i)} = \sum_{j=0}^{m-1} \bar{a}_j^{(i)} x^j$ and $A^{(i)} = \sum_{j=0}^{m-1} a_j^{(i)} x^j$, where $\bar{A}^{(0)} = A^{(0)} = A$. By using $x^m \bmod G = \sum_{j=0}^{m-1} g_j x^j$ and $x^{-1} \bmod G = x^{m-1} + \sum_{j=1}^{m-1} g_j x^{j-1}$, $\bar{A}^{(i)}$ and $A^{(i)}$ are rewritten as

$$\bar{A}^{(i)} = \bar{A}^{(i-1)} x^{-1} \bmod G = \sum_{j=0}^{m-1} (\bar{a}_{j+1}^{(i-1)} + \bar{a}_0^{(i-1)} g_{j+1}) x^j \tag{3}$$

$$A^{(i)} = A^{(i-1)} x \bmod G = \sum_{j=0}^{m-1} (a_{j-1}^{(i-1)} + a_{m-1}^{(i-1)} g_j) x^j. \tag{4}$$

Also, using the formulas of $\bar{A}^{(i)}$ and $A^{(i)}$, the terms Q, R, T, and U are represented by the following equations. For deriving the identical structure, we add $z\bar{A}^{(0)}$ to Q and T, where $z = 0$.

$$Q = \sum_{i=1}^{k} b_{k-i} A x^{-i} \bmod G = z\bar{A}^{(0)} + \sum_{i=1}^{k} b_{k-i} \bar{A}^{(i)}, \tag{5}$$

$$R = \sum_{i=0}^{l-1} b_{k+i} A x^{i} \bmod G = \sum_{i=0}^{l-1} b_{k+i} A^{(i)}. \tag{6}$$

$$T = \sum_{i=1}^{k} a_{k-i} A x^{-i} \bmod G = z\bar{A}^{(0)} + \sum_{i=0}^{k} a_{k-i} \bar{A}^{(i)}, \tag{7}$$

$$U = \sum_{i=0}^{l-1} a_{k+i} A x^{i} \bmod G = \sum_{i=0}^{l-1} a_{k+i} A^{(i)}. \tag{8}$$

From (5) to (8), the recurrence equations of Q, R, T, and U can be formulated as

$$Q^{(i)} = \begin{cases} Q^{(i-1)} + z\bar{A}^{(i-1)}, & \text{for } i = 1 \\ Q^{(i-1)} + b_{k-i+1} \bar{A}^{(i-1)}, & \text{for } 2 \le i \le k+1, \end{cases} \tag{9}$$

$$R^{(i)} = R^{(i-1)} + b_{k+i-1} A^{(i-1)}, \quad \text{for } 1 \le i \le l, \tag{10}$$

$$T^{(i)} = \begin{cases} T^{(i-1)} + z\bar{A}^{(i-1)}, & \text{for } i = 1 \\ T^{(i-1)} + a_{k-i+1} \bar{A}^{(i-1)}, & \text{for } 2 \le i \le k+1, \end{cases} \tag{11}$$

$$U^{(i)} = U^{(i-1)} + a_{k+i-1} A^{(i-1)}, \quad \text{for } 1 \le i \le l, \tag{12}$$

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

where $Q^{(0)} = R^{(0)} = T^{(0)} = U^{(0)} = 0$ and $Q^{(i)} = \sum_{j=0}^{m-1} q_j^{(i)} x^j$, $R^{(i)} = \sum_{j=0}^{m-1} r_j^{(i)} x^j$, $T^{(i)} = \sum_{j=0}^{m-1} t_j^{(i)} x^j$, and $U^{(i)} = \sum_{j=0}^{m-1} u_j^{(i)} x^j$ are ith intermediate results.

The equations {(3),(9),(11)} and {(4),(10),(12)} can be simultaneously executed because there are no data dependency between computations of $\{\bar{A}^{(i)}, Q^{(i)}, T^{(i)}\}$ and $\{A^{(i)}, R^{(i)}, U^{(i)}\}$. Therefore, the results of multiplication and squaring are represented as follows:

$$P = Q^{(k+1)} + R^{(l)} \tag{13}$$

and

$$S = T^{(k+1)} + U^{(l)}. \tag{14}$$

Then, the coefficients of $Q^{(i)}$, $R^{(i)}$, $T^{(i)}$, and $U^{(0)}$ can be computed as follows:

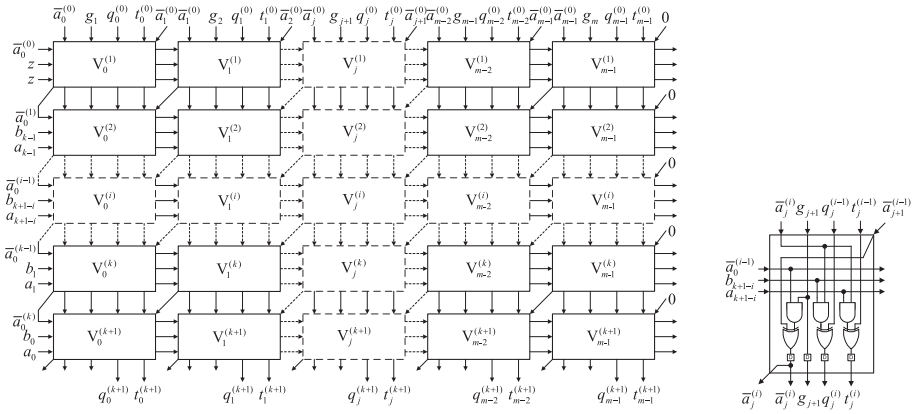
$$q_j^{(i)} = \begin{cases} q_j^{(i-1)} + z\bar{a}_j^{(i-1)}, & \text{for } i = 1 \\ q_j^{(i-1)} + b_{k-i+1}\bar{a}_j^{(i-1)}, & \text{for } 2 \le i \le k+1, \end{cases} \tag{15}$$

$$r_j^{(i)} = r_j^{(i-1)} + b_{k+i-1} a_j^{(i-1)}, \quad \text{for } 1 \le i \le l, \tag{16}$$

$$t_j^{(i)} = \begin{cases} t_j^{(i-1)} + z\bar{a}_j^{(i-1)}, & \text{for } i = 1 \\ t_j^{(i-1)} + a_{k-i+1}\bar{a}_j^{(i-1)}, & \text{for } 2 \le i \le k+1, \end{cases} \tag{17}$$

$$u_j^{(i)} = u_j^{(i-1)} + a_{k+i-1} a_j^{(i-1)}, \quad \text{for } 1 \le i \le l, \tag{18}$$

where $q_j^{(0)} = r_j^{(0)} = t_j^{(0)} = u_j^{(0)} = 0$ and $0 \le j \le m-1$.



(a) (b)

Fig. 1. (a) The semi-systolic array for computing $\{\bar{A}, Q, T\}$ (b) $\mathrm{V}_j^{(i)}$ cell

4 Proposed systolic arrays for unified multiplication and squaring

Based on the formulation used in the previous section, we present two efficient semi-systolic arrays to compute unified multiplication and squaring in this section.

The semi-systolic arrays for computing $\{\bar{A}, Q, T\}$ and $\{A, R, U\}$ is presented in Fig. 1(a) and Fig. 2(a), which are composed of $m \times (k+1)$ $\mathrm{V}_j^{(i)}$ and $m \times l$ $\mathrm{W}_j^{(i)}$ cells, respectively. The circuits of cells in semi-systolic arrays are illustrated in Fig. 1(b) and Fig. 2(b), where the boxed D denotes 1-bit latch (flip-flop). Each $\mathrm{V}_j^{(i)}$ cell employs three 2-input AND gates, three 2-input XOR gates, and four 1-bit

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

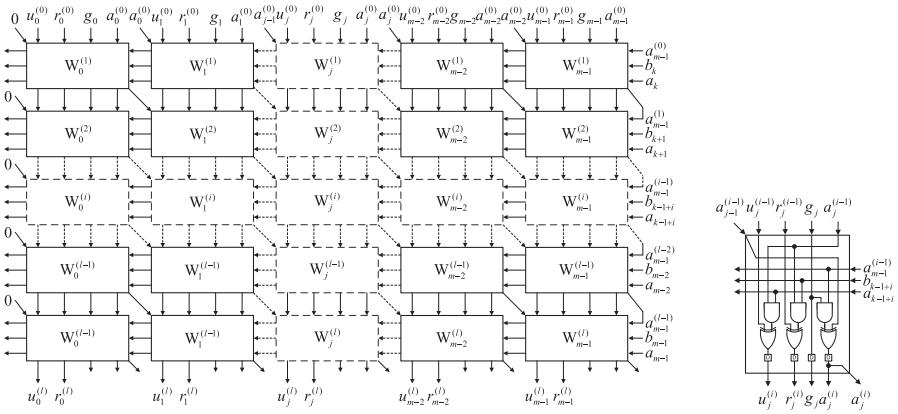


Fig. 2. (a) The semi-systolic array for computing $\{A, R, U\}$ (b) $W_j^{(i)}$ cell

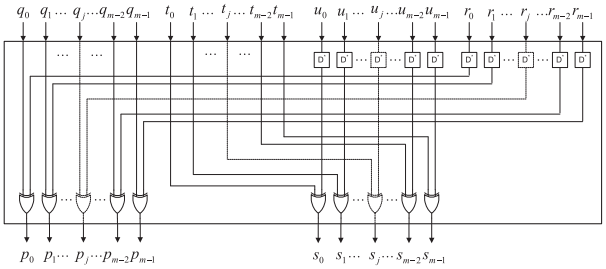


Fig. 3. The module for computing $Q + R$ and $T + U$

latches in order to compute $\bar{a}_j^{(i)}$, $q_j^{(i)}$, and $t_j^{(i)}$ in (3), (15) and (17). Similarly, each $W_j^{(i)}$ cell employs three 2-input AND gates, three 2-input XOR gates, and four 1-bit latches in order to compute $a_j^{(i)}$, $r_j^{(i)}$, and $u_j^{(i)}$ in (4), (16) and (18).

Fig. 3 shows a module for computing $Q^{(k+1)} + R^{(l)}$ and $T^{(k+1)} + U^{(l)}$ which includes $2m$ 2-input XOR gates and $2m$ boxed D* components, where the boxed D* denotes 1-bit latch only if m is even, otherwise it is ignored. The modules for computation of $\{\bar{A}, Q, T\}$ and $\{A, R, U\}$ takes $k + 1$ and l clock cycles, respectively. If m is odd, $k + 1 = l$. Otherwise, $k = l$. Therefore, if m is even, 1-bit latches are required at input lines of $R^{(l)}$ and $U^{(l)}$ before the computation of $Q^{(k+1)} + R^{(l)}$ and $T^{(k+1)} + U^{(l)}$ in order to synchronize them.

The efficient unified architecture for Montgomery multiplication and squaring over $GF(2^m)$ is depicted in Fig. 4. The latency of the proposed architecture requires $0.5m + 2$ clock cycles. Each clock cycle takes delays of one 2-input AND gate, one 2-input XOR gate, and one 1-bit latch. The space complexity of this architecture requires $3m^2 + 3m$ 2-input AND gates, $3m^2 + 5m$ 2-input XOR gates, and $4m^2 + 4m$ 1-bit latches.

The $V_j^{(i)}$ cell calculates equations (3), (15), and (17) and the $W_j^{(i)}$ cell calculates equations (4), (16), and (18). As we can see from Fig. 1(b) and Fig. 2(b), the $V_j^{(i)}$ cell and the $W_j^{(i)}$ cell perform identical function with different inputs. Therefore, one systolic array in Fig. 5 can calculate equations (3), (15), and (17) at the first round and equations (4), (16), and (18) at the second round, respectively. The Fig. 6 shows the module to compute equations (13) and (14). The Fig. 7 shows the

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

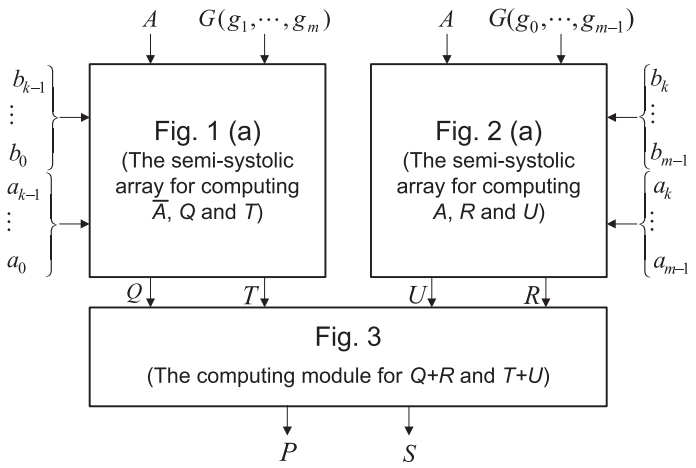


Fig. 4. The proposed multiplier/squarer over $GF(2^m)$

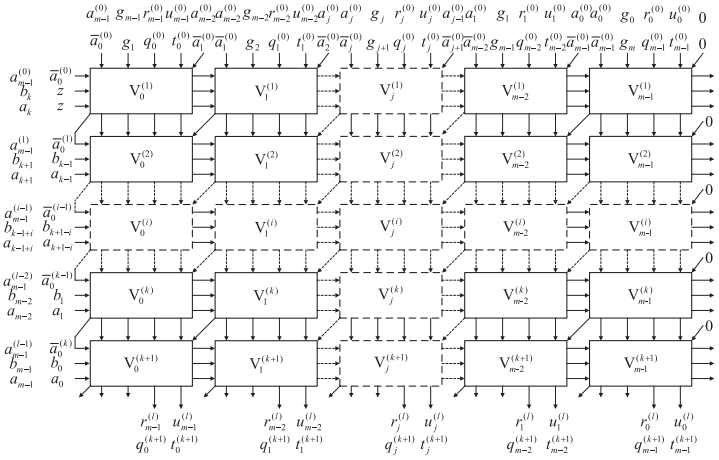


Fig. 5. The semi-systolic array for computing $\{\bar{A}, Q, T\}$ and $\{A, R, U\}$

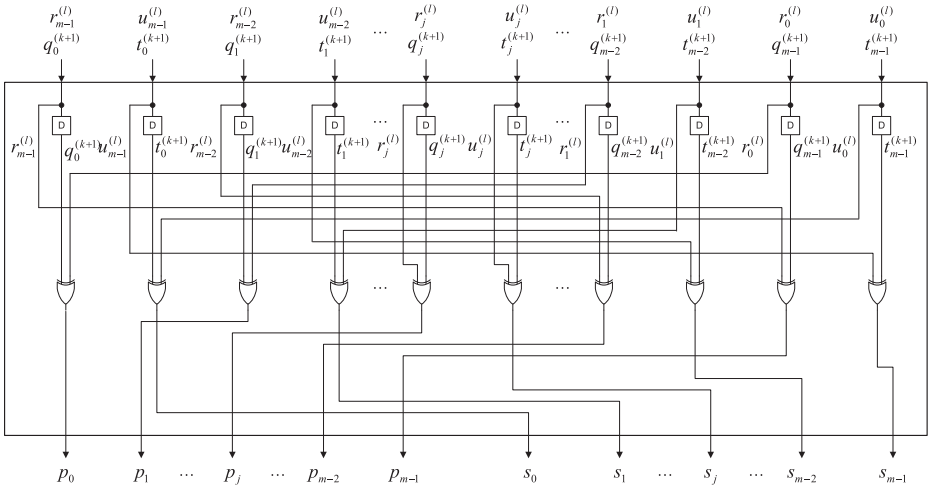


Fig. 6. The module for computing $Q + R$ and $T + U$

unified architecture for Montgomery multiplication and squaring over $GF(2^m)$, where m is odd. When m is even, it has similar structure. The space complexity of Fig. 7 is reduced by about half compared to Fig. 4.

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

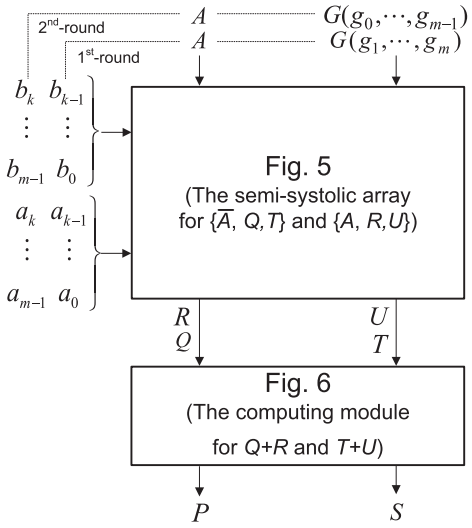


Fig. 7. The proposed area-efficient multiplier/squarer over $GF(2^m)$

5 Complexity analysis and conclusion

As mentioned in subsection 2.3, Montgomery multiplication and squaring can be performed in parallel in order to perform fast modular exponentiation. It can be implemented by the unified multiplier/squarer or by two multipliers. Recently, Choi and Lee [6] proposed the combined systolic array for performing multiplication and squaring in parallel. Kim and Jeon [5] proposed a semi-systolic multipliers using bipartite method. Also they proposed an area-efficient semi-systolic multiplier using bipartite method in [7]. A circuit comparison between the proposed and the related multipliers is given in Table I.

For a comparison of the time and area complexity, we utilize the "SAMSUNG STD 150 0.13m 1.2V CMOS Standard Cell Library". Based on this library, we estimated the time and area complexities of the proposed and the related multipliers. As discussed in detail in [6], we adopt that $A_{AND2} = 6.68$, $T_{AND2} = 0.094$ ns, $A_{XOR2} = 12.00$, $T_{XOR2} = 0.167$ ns, $A_{LATCH1} = 16.00$, and $T_{LATCH1} = 0.157$ ns, where A_{GATEn} denotes transistor count of an n-input gate and T_{GATEn} denotes the propagation delay of an n-input gate.

All multipliers in Table I has the same cell delay of $T_{AND2} + T_{XOR2} + T_{LATCH1}$. But the multiplier of Choi and Lee [6] has the latency of $3m$ clock cycles and the latency of the others is about $0.5m$ clock cycles. Compared with the multiplier of Choi and Lee, the proposed multipliers in Fig. 4 and Fig. 7 can reduce the space complexity by 44.4% and 72.1% and the AT complexity by 90.7% and 95.3%, respectively. The proposed multipliers in Fig. 4 and Fig. 7 have about 40.6% greater space complexity and 40.8% greater AT complexity, compared with the multipliers in [5] and [7], respectively. But our proposed multipliers can perform multiplication and squaring in parallel for fast modular exponentiation.

In this paper, a semi-systolic architecture for Montgomery multiplication/square for fast modular exponentiation over finite fields has been presented. We induced an efficient algorithm which is highly suitable for the design of parallel pipelined structures. We expect that our architecture can be efficiently used for

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017

Table I. Comparison of the systolic arrays of multiplication and squaring

Multipliers	Choi-Lee [6]	Kim-Jeon [5]	Kim-Jeon [7]	Fig. 4	Fig. 7
Array type	systolic	semi-systolic	semi-systolic	semi-systolic	semi-systolic
Function	AB and A^2	AB	AB	AB and A^2	AB and A^2
Bipartite	X	O	O	O	O
Throughput	1	1	1/2	1	1/2
Area complexity					
AND_2	$3m^2$	$2m^2 + 2m$	$m^2 + 0.5m$ -0.5	$3m^2 + 3m$	$1.5m^2 + 1.5m$
XOR_2	$3m^2$	$2m^2 + 3m$	$m^2 + 1.5m$ -0.5	$3m^2 + 5m$	$1.5m^2 + 3.5m$
Latch	$10m^2$	$3m^2 + 3m$	$1.5m^2 + 2m$ -0.5	$4m^2 + 4m$	$2m^2 + 4m$
Total transistors	$216.04m^2$	$85.36m^2$ $+97.36m$	$42.68m^2$ $+53.34m$ $-17.34m$	$120.04m^2$ $+144.04m$	$60.02m^2$ $+116.02m$
Time complexity					
Cell delay	0.418	0.418	0.418	0.418	0.418
Latency	$3m$	$0.5m + 0.5$	$0.5m + 1.5$	$0.5m + 2$	$0.5m + 2.5$
Total delay	$1.254m$	$0.209m + 0.209$	$0.209m + 0.627$	$0.209m + 0.836$	$0.209m + 1.045$
Area-Time complexity					
AT complexity	$270.91m^3$	$17.84m^3$ $+38.19m$	$8.92m^3$ $+37.90m^2$ $+29.82m$ -10.87	$25.09m^3$ $+130.46m^2$ $+120.42m$	$12.54m^3$ $+86.97m^2$ $+121.24m$

various applications including crypto coprocessor design, which demand high-speed computation, for security purposes.

Acknowledgments

The present research was conducted by the research fund of Dankook University in 2015.

DOI: 10.1587/elex.14.20170458
Received May 1, 2017
Accepted May 16, 2017
Publicized June 1, 2017
Copyedited June 25, 2017