International Journal of Cyber-Security and Digital Forensics (IJCSDF) 1(2): 122-129
The Society of Digital Information and Wireless Communications, 2012 (ISSN: 2305-0012)

Finite Field Arithmetic Architecture Based on Cellular Array

Kee-Won Kim[*] and Jun-Cheol Jeon[**]
[*]Institute of Media Content, Dankook University
152, Jukjeon-ro, Suji-gu, Yongin, Gyeonggi-do, 448-701, Korea
[**]Dept. of Computer Engineering, Kumoh National Institute of Technology
61, Daehak-ro, Gumi, Gyeongbuk, 730-701, Korea
nirkim@gmail.com[*], jcjeon@kumoh.ac.kr(corresponding author)[**]

ABSTRACT

Recently, various finite field arithmetic structures are introduced for VLSI circuit implementation on cryptosystems and error correcting codes. In this study, we present an efficient finite field arithmetic architecture based on cellular semi-systolic array for Montgomery multiplication by choosing a proper Montgomery factor which is highly suitable for the design on parallel structures. Therefore, our architecture has reduced a time complexity by 50% compared to typical architecture.

KEYWORDS

cellular array, finite field, semi-systolic structure, Montgomery multiplication, arithmetic architecture

1 INTRODUCTION

Finite field arithmetic operations, especially for the binary field GF(2^m), have been widely used in the areas of data communication and network security applications such as error-correcting codes [1,2] and cryptosystems such as ECC(Elliptic Curve Cryptosystem) [3,4]. The finite field multiplication is the most frequently studied. This is because the time-consuming operations such as exponentiation, division, and multiplicative inversion can be decomposed into repeated multiplications. Thus, the fast multiplication architecture with low complexity is needed to design dedicated high-speed circuits.

Certainly, one of most interesting and useful advances in this realm has been the Montgomery multiplication algorithm, introduced by Montgomery [5] for fast modular integer multiplication. The multiplication was successfully adapted to finite field GF(2^m) by Koc and Acar [6]. They have proposed three Montgomery multiplication algorithms for bit-serial, digit-serial, and bit-parallel multiplication. They have chosen the Montgomery factor, $R=x^m$ for efficient implementation of the multiplication in hardware and software.

Wu [7] has chosen a new Montgomery factor and shown that choosing the middle term of the irreducible trinomial $G(\omega)= \omega^m+\omega^k+1$ as the Montgomery factor, i.e., $R=x^k$, results in more efficient bit-parallel architectures. In [8], MM is implemented using systolic arrays for all-one polynomials and trinomials. Chiu et al. [9] proposed semi-systolic array structure for MM which uses $R=x^m$. Hariri and Reyhani-Masoleh [10] proposed a number of bit-serial and bit-parallel Montgomery multipliers and showed that MM can accelerate the ECC scalar multiplication. Recently, in [11], they have considered concurrent error detection for MM over binary field.

Three different multipliers, namely the bit-serial, digit-serial, and bit-parallel multipliers, have been considered and the concurrent error detection scheme has been derived and implemented for each of them.

Chiou [12] used the recomputing with shifted operands (RESO) to provide a concurrent error detection method for polynomial basis multipliers using an irreducible all-one polynomial, which is a special case of a general polynomial. Lee et al. [13] described a concurrent error detection (CED) method for a polynomial multiplier with an irreducible general polynomial. Chiou et al. [9] also developed a Montgomery multiplier with concurrent error detection capability. Bayat-Sarmadi and Hasan [14] proposed semi-systolic multipliers for various bases, such as the polynomial, dual, type I and type II optimal normal bases. They have also presented semi-systolic multipliers with CED using RESO.

Recently, Huang et al. [15] proposed the semi-systolic polynomial basis multiplier over GF(2^m) to reduce both space and time complexities. Also they proposed the semi-systolic polynomial basis multipliers with concurrent error detection and correction capability. Various approaches adopt semi-systolic architectures to reduce the total number of latches and computation latency because of permitting the broadcast signals. However, almost existing polynomial multipliers suffer from several shortcomings, including large time and/or hardware overhead, and low performance.

In this paper, we consider the shortcomings that the typical architectures have, and propose a semi-systolic Montgomery multiplier with a new Montgomery factor. We show that an efficient multiplication architecture can be obtained by choosing a proper Montgomery factor, and reduces time complexity.

The remainder of this paper is organized as follows. Section 2 introduces Montgomery multiplication over finite fields. In Section 3, we propose a Montgomery multiplication architecture based on our algorithm which is highly optimized for hardware implementation. In Section 4, we analyze and compare our architecture with recent study. Finally, Section 5 gives our conclusion.

2 MONTGOMERY MULTIPLICATION ON FINITE FIELDS

GF(2^m) is a kind of finite field [16] that contains 2^m different elements. This finite field is an extension of GF(2) and any $A \in$ GF(2^m) can be represented as a polynomial of degree $m-1$ over GF(2), such as

$$A = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0,$$

where $a_i \in \{0,1\}$, $0 \le i \le m-1$.

Let x be a root of the polynomial, then the irreducible polynomial G is represented as a following equation.

$$G = g_m x^m + \cdots + g_1 x + g_0, \qquad (1)$$

where $g_i \in$ GF(2), $0 \le i \le m-1$.

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 $A = \alpha \cdot R \bmod G$ and $B = \beta \cdot R \bmod G$, where GCD($R$,$G$) =

1. Then, the Montgomery multiplication algorithm over GF(2^m) can be formulated as

$$P = A \cdot B \cdot R^{-1} \bmod G,$$

where R^{-1} is the inverse of R modulo G, and $R \cdot R^{-1} + G \cdot G' = 1$ [17]. Thus, by the definition of the Montgomery residue, the equation can be expressed as follows.

$$\begin{aligned} P &= (\alpha \cdot R) \cdot (\beta \cdot R) \cdot R^{-1} \bmod G \\ &= \gamma \cdot R \bmod G \end{aligned}$$

It means that P is the Montgomery residue of γ. This makes it possible to convert the operands to Montgomery residues once at the beginning, and then, do several consecutive multiplications/ squarings, and convert the final result to the original representation. The final conversion is a multiplication by R^{-1}, i.e., $\gamma = P \cdot R^{-1} \bmod G$. The polynomial R plays an important role in the complexity of the algorithm as we need to do modulo R multiplication and a final division by R.

3 PROPOSED ARCHITECTURE

This section describes the proposed Montgomery multiplication algorithm and architecture.

3.1 Proposed Algorithm

Based on the property of parallel architecture, we choose the Montgomery factor, $R = x^{\lfloor m/2 \rfloor}$. Then, the Montgomery multiplication over GF(2^m) can be formulated as

$$P = A \cdot B \cdot x^{-\lfloor m/2 \rfloor} \bmod G \qquad (2)$$

We know that x is a root of G and g_m and g_0 have always '1' over all irreducible polynomials. Thus, the equations can be rewritten as follows.

$$\begin{aligned} & x^m \bmod G \\ &= g_{m-1} x^{m-1} + \cdots + g_1 x + 1 \end{aligned} \qquad (3)$$

$$\begin{aligned} & x^{-1} \bmod G \\ &= x^{m-1} + g_{m-1} x^{m-2} + \cdots + g_1 \end{aligned} \qquad (4)$$

Meanwhile, (2) is represented by substituting A and B as follows.

$$\begin{aligned} & P \bmod G \\ &= [b_{\lfloor m/2 \rfloor - 1} A x^{-1} + b_{\lfloor m/2 \rfloor - 2} A x^{-2} + \cdots \\ &+ b_1 A x^{-\lfloor m/2 \rfloor + 1} + b_0 A x^{-\lfloor m/2 \rfloor} \\ &+ b_{\lfloor m/2 \rfloor} A + b_{\lfloor m/2 \rfloor + 1} A x + \cdots \\ &+ b_{m-2} A x^{\lceil m/2 \rceil - 2} + b_{m-1} A x^{\lceil m/2 \rceil - 1}] \end{aligned} \qquad (5)$$

Now, it expresses that P can be divided into two parts. One is based on the negative powers of x and the other is based on the positive powers of x. (5) can be denoted by $P = C + D$, where

$$\begin{aligned} C &= [b_{\lfloor m/2 \rfloor - 1} A x^{-1} + b_{\lfloor m/2 \rfloor - 2} A x^{-2} + \cdots + \\ &\quad b_1 A x^{-\lfloor m/2 \rfloor + 1} + b_0 A x^{-\lfloor m/2 \rfloor}] \bmod G, \\ D &= [b_{\lfloor m/2 \rfloor} A + b_{\lfloor m/2 \rfloor + 1} A x + \cdots + \\ &\quad b_{m-2} A x^{\lceil m/2 \rceil - 2} + b_{m-1} A x^{\lceil m/2 \rceil - 1}] \bmod G. \end{aligned}$$

Meanwhile, let $\overline{A}^{(i)}$ and $A^{(i)}$ be $Ax^{-i} \bmod G$ and $Ax^{i} \bmod G$, respectively. Then, based on (3) and (4), the equations can be expressed as

$$\begin{aligned}\overline{A}^{(i)} &= x^{-1}\overline{A}^{(i-1)} \bmod G \\ &= x^{-1}(\overline{a}_0^{(i-1)} + \overline{a}_1^{(i-1)}x + \cdots + \\ &\quad \overline{a}_{m-2}^{(i-1)}x^{m-2} + \overline{a}_{m-1}^{(i-1)}x^{m-1}) \bmod G \\ &= (\overline{a}_1^{(i-1)} + \overline{a}_0^{(i-1)}g_1) + \cdots \\ &\quad + (\overline{a}_{m-1}^{(i-1)} + \overline{a}_0^{(i-1)}g_{m-1})x^{m-2} + \overline{a}_0^{(i-1)}x^{m-1},\end{aligned}$$

$$\begin{aligned}A^{(i)} &= xA^{(i-1)} \bmod G \\ &= x(a_0^{(i-1)} + a_1^{(i-1)}x + \cdots + \\ &\quad a_{m-2}^{(i-1)}x^{m-2} + a_{m-1}^{(i-1)}x^{m-1}) \bmod G \\ &= a_{m-1}^{(i-1)} + (a_0^{(i-1)} + a_{m-1}^{(i-1)}g_1)x + \cdots \\ &\quad + (a_{m-2}^{(i-1)} + a_{m-1}^{(i-1)}g_{m-1})x^{m-1},\end{aligned}$$

where

$$\overline{a}_j^{(i)} = \begin{cases} \overline{a}_{j+1}^{(i-1)} + \overline{a}_0^{(i-1)}g_{j+1}, & 0 \le j \le m-2 \\ \overline{a}_0^{(i-1)}, & j = m-1, \end{cases} \quad (6)$$

$$a_j^{(i)} = \begin{cases} a_{j-1}^{(i-1)} + a_{m-1}^{(i-1)}g_j, & 1 \le j \le m-1 \\ a_{m-1}^{(i-1)}, & j = 0 \end{cases} \quad (7)$$

Also, using the formulae of $\overline{A}^{(i)}$ and $A^{(i)}$, the terms C and D are represented as follows.

$$\begin{aligned}&C \bmod G \\ &= [b_0 Ax^{-\lfloor m/2 \rfloor} + b_1 Ax^{-\lfloor m/2 \rfloor+1} + \cdots + \\ &\quad b_{\lfloor m/2 \rfloor-2}Ax^{-2} + b_{\lfloor m/2 \rfloor-1}Ax^{-1}] \\ &= z\overline{A}^{(0)} + b_{\lfloor m/2 \rfloor-1}\overline{A}^{(1)} + b_{\lfloor m/2 \rfloor-2}\overline{A}^{(2)} \\ &\quad + \cdots + b_1\overline{A}^{(\lfloor m/2 \rfloor-1)} + b_0\overline{A}^{(\lfloor m/2 \rfloor)}\end{aligned} \quad (8)$$

$$\begin{aligned}&D \bmod G \\ &= [b_{\lfloor m/2 \rfloor}A + b_{\lfloor m/2 \rfloor+1}Ax + \cdots + \\ &\quad b_{m-2}Ax^{\lceil m/2 \rceil-2} + b_{m-1}Ax^{\lceil m/2 \rceil-1}] \\ &= b_{\lfloor m/2 \rfloor}A^{(0)} + b_{\lfloor m/2 \rfloor+1}A^{(1)} + \cdots + \\ &\quad b_{m-2}A^{(\lceil m/2 \rceil-2)} + b_{m-1}A^{(\lceil m/2 \rceil-1)},\end{aligned} \quad (9)$$

where $z = 0$.

The coefficients of C and D are produced by summing the corresponding coefficients of each term in (8) and (9), respectively. It means that c_j and d_j, for $0 \le j \le m-1$ are represented as

$$\begin{aligned}c_j &= z\overline{a}_j^{(0)} + b_{\lfloor m/2 \rfloor-1}\overline{a}_j^{(1)} + b_{\lfloor m/2 \rfloor-2}\overline{a}_j^{(2)} \\ &\quad + \cdots + + b_1\overline{a}_j^{(\lfloor m/2 \rfloor-1)} + b_0\overline{a}_j^{(\lfloor m/2 \rfloor)}\end{aligned}$$

Algorithm 1. COM_C(A,B',G)

Input:

$A = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0)$,

$B' = (b_{\lfloor m/2 \rfloor-1}, b_{\lfloor m/2 \rfloor-2}, \cdots, b_1, b_0)$,

$G = (g_{m-1}, g_{m-2}, \cdots, g_1, g_0)$

Output:

$C = [b_0 Ax^{-\lfloor m/2 \rfloor} + b_1 Ax^{-\lfloor m/2 \rfloor+1} + \cdots + b_{\lfloor m/2 \rfloor-2}Ax^{-2} + b_{\lfloor m/2 \rfloor-1}Ax^{-1}] \bmod G$

$\overline{a}_j^{(0)} \leftarrow a_j$; $c_j^{(0)} \leftarrow 0$; $z = 0$;
for $i = 1$ to $\lfloor m/2 \rfloor + 1$ do
  for $j = 0$ to $m-1$ in parallel do
    if (j = 0) then /* $j = 0$ */
      $\overline{a}_{m-1}^{(i)} = \overline{a}_0^{(i-1)}$;
      $c_0^{(i)} = c_0^{(i-1)} + b_{\lfloor m/2 \rfloor-i+1}\overline{a}_0^{(i-1)}$
      (or $c_0^{(1)} = c_0^{(i)} + z\overline{a}_0^{(0)}$ if $i = 0$);
    else /* $j = 1,2,\cdots,m-2,m-1$ */
      $\overline{a}_{m-j-1}^{(i)} = \overline{a}_{m-j}^{(i-1)} + \overline{a}_0^{(i-1)}g_{m-j}$;
      $c_{m-j}^{(i)} = c_{m-j}^{(i-1)} + b_{\lfloor m/2 \rfloor-i+1}\overline{a}_{m-j}^{(i-1)}$
      (or $c_{m-j}^{(i)} = c_{m-j}^{(i-1)} + z\overline{a}_{m-j}^{(i-1)}$ if $i = 0$);
    end if
  end for
end for
return C

$$d_j = b_{\lfloor m/2 \rfloor} a_j^{(0)} + b_{\lfloor m/2 \rfloor+1} a_j^{(1)} + \cdots + b_{m-2} a_j^{(\lceil m/2 \rceil-2)} + b_{m-1} a_j^{(\lceil m/2 \rceil-1)}.$$

Now, we obtain the following recurrence equations from the above equations.

$$c_j^{(i)} = \begin{cases} c_j^{(i-1)} + z\bar{a}_j^{(i-1)}, & i = 1 \\ c_j^{(i-1)} + b_{\lfloor m/2 \rfloor - i+1} \bar{a}_j^{(i-1)}, & 1 < i \le \lfloor m/2 \rfloor + 1, \end{cases}$$

where $c_j^{(0)} = 0$ for $0 \le j \le m-1$ and $z = 0$, and

$d_j^{(i)} = d_j^{(i-1)} + b_{\lfloor m/2 \rfloor + i - 1} a_j^{(i-1)}$, $1 \le i \le \lceil m/2 \rceil$

where $d_j^{(0)} = 0$ for $0 \le j \le m-1$.

Algorithm 2. COM_D(A,B″,G)

Input:

$A = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0)$,

$B'' = (b_{\lfloor m/2 \rfloor}, b_{\lfloor m/2 \rfloor+1}, \cdots, b_{m-2}, b_{m-1})$,

$G = (g_{m-1}, g_{m-2}, \cdots, g_1, g_0)$

Output:

$D = [b_{\lfloor m/2 \rfloor} A + b_{\lfloor m/2 \rfloor+1} Ax + \cdots + b_{m-2} Ax^{\lceil m/2 \rceil-2} + b_{m-1} Ax^{\lceil m/2 \rceil-1}] \bmod G$

$a_j^{(0)} \leftarrow a_j$; $d_j^{(0)} \leftarrow 0$;
for $i = 1$ to $\lceil m/2 \rceil$ do
  for $j = 0$ to $m-1$ in parallel do
    if (j=0) then /* $j = 0$ */
      $a_0^{(i)} = a_{m-1}^{(i-1)}$;
      $d_{m-1}^{(i)} = d_{m-1}^{(i-1)} + b_{\lfloor m/2 \rfloor + i - 1} a_{m-1}^{(i-1)}$;
    else /* $j = 1, 2, \cdots, m-2, m-1$ */
      $a_j^{(i)} = a_{j-1}^{(i-1)} + a_{m-1}^{(i-1)} g_j$;
      $d_{j-1}^{(i)} = d_{j-1}^{(i-1)} + b_{\lfloor m/2 \rfloor + i - 1} a_{j-1}^{(i-1)}$;
    end if
  end for
end for
return D

As shown in Algorithm 1 and 2, the parallel computational algorithms for *C* and *D* are driven by the above equations. The proposed COM_C(A,B,G) and COM_D(A,B,G) algorithms can be executed simultaneously since there is no data dependency between computing *C* and *D*.

3.2 Proposed Multiplier

Based on the proposed algorithms, the hardware architecture of the proposed semi-systolic Montgomery multiplier is shown in Figure 1. The upper, lower, middle part of the array computes *C*, *D*, and *C*+*D*, respectively. Our architecture is composed of $\lfloor m/2 \rfloor + 1$ $\hat{U}_0^{(i)}$ cells, $(m-2) \times (\lfloor m/2 \rfloor + 1)$ $U_j^{(i)}$ cells, $\lceil m/2 \rceil$ $\hat{V}_0^{(i)}$ cells, $(m-2) \times \lceil m/2 \rceil$ $V_j^{(i)}$ cells, and one S cell.

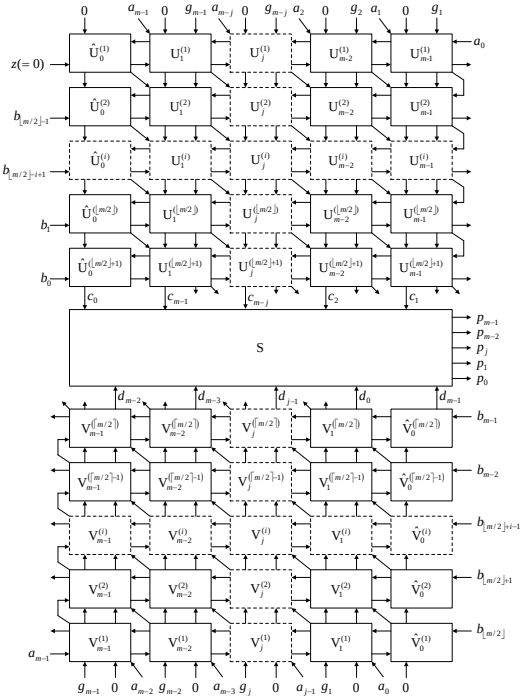

Figure 1. The proposed semi-systolic Montgomery multiplier over GF(2^m)

The detailed circuits of the cells in Figure 1 are depicted in Figure 2 thru Figure 4, and ⊕, ⊗, and D denote XOR gate, AND gate, and one-bit latch(flip-flop), respectively.

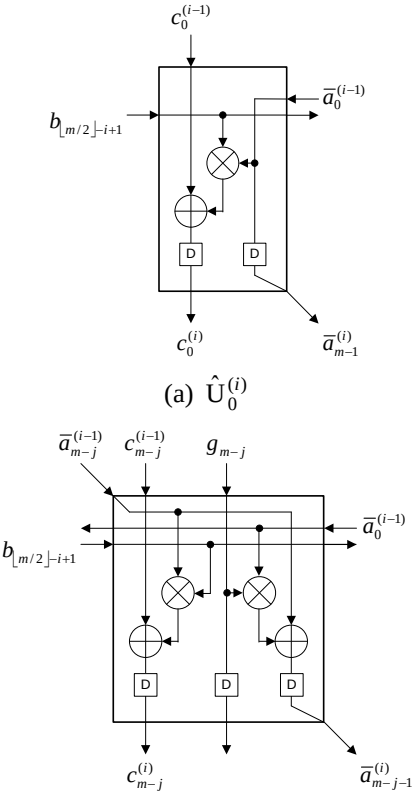

Figure 2. Circuit configuration of $\hat{U}_0^{(i)}$ and $U_j^{(i)}$ cell

The latency of the proposed semi-systolic multiplier requires $\lfloor m/2 \rfloor+1$ clock cycles. Each clock cycle takes the delay of one 2-input AND gate, one 2-input XOR gate, and one 1-bit latch. The space complexity of this multiplier requires $2m^2+m-1$ 2-input AND gates, $2m^2+2m-1$ 2-input XOR gates, and $3m^2+2m-1$(for odd m) or $3m^2+3m-1$(for even m) 1-bit latches.

Note that $U_j^{(i)}$ ($\hat{U}_0^{(i)}$) and $V_j^{(i)}$ ($\hat{V}_0^{(i)}$) cells in Figure 2 and 3 are functionally equivalent cells and the computations can be executed in parallel, and the computed results are added in S cell. In Figure 4, D^* denotes one bit latch when m is even, otherwise it is ignored.

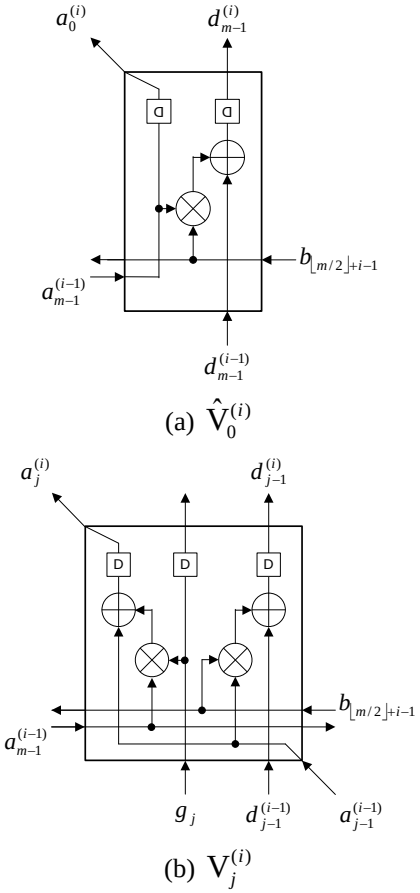

Figure 3. Circuit configuration of $\hat{V}_0^{(i)}$ and $V_j^{(i)}$ cell

4 COMPLEXITY ANALYSIS

In CMOS VLSI technology, each gate is composed of several transistors [18]. We adopt that $A_{\text{AND2}} = 6$, $A_{\text{XOR2}} = 6$, and $A_{\text{LATCH1}} = 8$, where $A_{\text{GATE}n}$ denotes transistor count of an n-input gate, respectively. Also, for a further comparison of time complexity, we adopt the practical integrated circuits in [19] and the following assumptions, as discussed in detail in [15], are made: $T_{\text{AND2}} = 7$, $T_{\text{XOR2}} = 12$, and $T_{\text{LATCH1}} = 13$,

where T_{GATEn} denotes the propagation delay of an *i*-input gate, respectively.

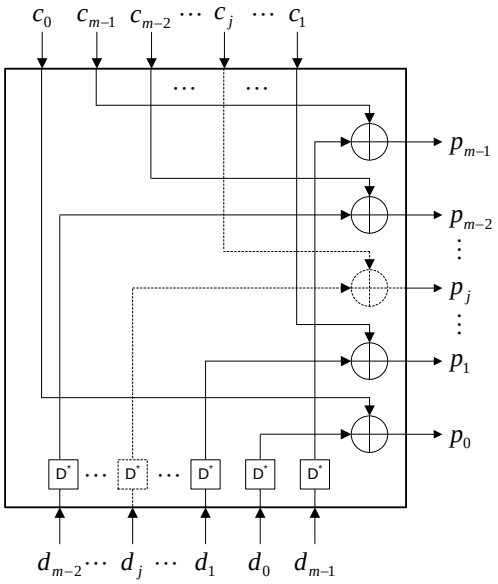

Figure 4. Circuit configuration of S cell

Table 1. Comparison of semi-systolic polynomial basis architectures

gate/delay	In [15]	Fig. 1 even *m*/odd *m*
Number of cells	m^2	$\hat{U}: \lfloor m/2 \rfloor + 1$ $U: (m-2)\times(\lfloor m/2 \rfloor + 1)$ $\hat{V}: \lceil m/2 \rceil$ $V: (m-2)\times\lceil m/2 \rceil$ $s: 1$
2-input AND	$2m^2$	$2m^2+m-1$
2-input XOR	$2m^2$	$2m^2+2m-1$
3-input XOR	0	0
one-bit latch	$3m^2$	$3m^2+3m-1 / 3m^2+2m-1$
Total transistor count	$48m^2$	$48m^2+42m-20 / 48m^2+34m-20$
Cell delay(ns)	32	32
Latency	m	$0.5m+1 / 0.5m+0.5$
Total delay(ns)	$32m$	$16m+32 / 16m+16$

A circuit comparison between the proposed multiplier and the related multiplier is given in Table 1. Although the proposed multiplier has nearly the same space complexity compared to Huang et al.[15], the time complexity is approximately reduced by 50%.

5 CONCLUSION

In this paper, we propose a cellular semi-systolic architecture for Montgomery multiplication over finite fields. We choose a novel Montgomery factor which is highly suitable for the design of parallel structures. We also divided our architecture into three parts, and computed two parts of them in parallel so that we reduced the time complexity by nearly 50% compared to the recent study in spite of maintaining similar space complexity. We expect that our architecture can be efficiently used for various applications, which demand high-speed computation, based on arithmetic operations.

6 ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2011-0014977).

7 REFERENCES

1. W. W. Peterson, and E. J. Weldon, Error-Correcting Codes, MIT Press, Cambridge (1972).
2. R. E. Blahut. Theory and Practice of Error Control Codes, Addison-Wesley, Reading (1983).
3. W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644-654 (1976).
4. B. Schneier, Applied Cryptography, John Wiley & Sons press, 2nd edition (1996).

5. P. Montgomery, "Modular Multiplication without Trial Division," Mathematics of Computation, vol. 44, no. 170, pp. 519–521 (1985).
6. C. Koc and T. Acar, "Montgomery Multiplication in GF(2^k)," Designs, Codes and Cryptography, vol. 14, no. 1, pp. 57–69 (1998).
7. H. Wu, "Montgomery Multiplier and Squarer for a Class of Finite Fields," IEEE Trans. Computers, vol. 51, no. 5, pp. 521-529 (2002).
8. C. Y. Lee, J. S. Horng, I. C. Jou and E. H. Lu, "Low-Complexity Bit-Parallel Systolic Montgomery Multipliers for Special Classes of GF(2^m)," IEEE Transactions on Computers, vol. 54, no. 9, pp. 1061–1070 (2005).
9. C. W. Chiou, C. Y. Lee, A. W. Deng and J. M. Lin, "Concurrent Error Detection in Montgomery Multiplication over GF(2^m)," IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences, vol. E89-A, no. 2, pp. 566-574, 2006.
10. A. Hariri and A. Reyhani-Masoleh, "Bit-Serial and Bit-Parallel Montgomery Multiplication and Squaring over GF(2^m)," IEEE Trans. Computers, vol. 58, no. 10, pp. 1332-1345 (2009).
11. A. Hariri and A. Reyhani-Masoleh, "Concurrent Error Detection in Montgomery Multiplication over Binary Extension Fields," IEEE Trans. Computers, vol. 60, no. 9, pp. 1341-1353 (2011).
12. C. W. Chiou, "Concurrent Error Detection in Array Multipliers for GF(2^m) Fields," IEE Electronics Letters, vol. 38, no. 14, pp. 688–689 (2002).
13. C. Y. Lee, C. W. Chiou, and J. M. Lin, "Concurrent Error Detection in a Polynomial Basis Multiplier over GF(2^m)," J. Electronic Testing: Theory and Applications, vol. 22, no. 2, pp. 143-150 (2006).
14. S. Bayat-Sarmadi and M.A. Hasan, "Concurrent Error Detection in Finite Field Arithmetic Operations Using Pipelined and Systolic Architectures," IEEE Trans. Computers, vol. 58, no. 11, pp. 1553-1567 (2009).
15. W. T. Huang, C. H. Chang, C. W. Chiou and F. H. Chou, "Concurrent error detection and correction in a polynomial basis multiplier over GF(2^m)," IET Information Security, vol. 4, no. 3, pp. 111-124 (2010).
16. R. Lidl and H. Niederreiter, Introduction to Finite Fields and Their Applications, Cambridge Univ. Press (1986).
17. J. C. Jeon and K. Y. Yoo, "Montgomery exponent architecture based on programmable cellular automata," Mathematics and Computers in Simulation, vol. 79, pp. 1189-1196 (2008).
18. N. Weste, K. Eshraghian, Principles of CMOS VLSI design: a system perspective, Addison-Wesley, Reading, MA (1985).
19. STMicroelectronics, Available at http://www.st.com/