社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－137（6）
2008 / 11 / 17

Hardware Algorithm for Division in GF(2^m) Based on the Extended Euclid's Algorithm Accelerated with Parallelization of Modular Reductions

Katsuki KOBAYASHI† and Naofumi TAKAGI†

† Dept. of Information Engineering, Graduate School of Information Science, Nagoya Univ.,
C3-1 (631), Furo-cho, Chikusa-ku, Nagoya 464-8603 Japan.
E-mail: †{katsu,ntakagi}@takagi.i.is.nagoya-u.ac.jp

Abstract We propose a fast hardware algorithm for division in GF(2^m). It is based on the extended Euclid's algorithm and requires only one iteration to perform the operations that require two iterations of previously reported division algorithms based on the extended Euclid's algorithm. Since the algorithm performs modular reductions in parallel by changing the order of execution of the operations, a circuit based on this algorithm has almost the same critical path delay as the previously proposed ones. The circuit computes division in m clock cycles, whereas the previously proposed circuits take $2m - 1$ or more clock cycles. By logic synthesis, the computation time of the circuit is estimated to over 35% shorter than that of a previously proposed circuit.

Key words Galois field, Division, Euclid's algorithm

1. Introduction

Galois field GF(2^m) has many applications, especially in elliptic curve cryptography (ECC). In order to accelerate such applications, rapid implementation of arithmetic operations in GF(2^m) is required. Among basic arithmetic operations in GF(2^m), division takes the maximum time. In this report, we propose a fast hardware algorithm for division in GF(2^m) with parallelization of modular reductions.

In general, one of the following three methods is employed for division in Galois field: the Fermat's little theorem [1], [2], the extended Euclid's algorithm [3]～[11], or a solution of a system of linear equations [12], [13]. When m is large, division algorithms based on the extended Euclid's algorithm are the most efficient way to implement circuits, because circuits based on them can be implemented easily and have lower AT-product [4], [13]. The algorithm to be proposed in this report is also based on the extended Euclid's algorithm.

The proposed algorithm requires only one iteration to perform the operations that require two iterations in previously reported division algorithms based on the extended Euclid's algorithm. Division algorithms based on the extended Euclid's algorithm perform modular reductions. Although two iterations of the previously proposed division algorithms perform two modular reductions sequentially, the proposed algorithm performs them in parallel by changing the order of execution of the operations.

We have designed a circuit based on this algorithm, which performs the operations in one iteration of the algorithm in one clock cycle. The latency of the circuit is m clock cycles, which is almost half of that of the circuits proposed in [3], [10] that have architecture similar to our circuit. The critical path delay of the circuit is larger by the delay of a 2-input XOR gate compared to that of the circuit reported in [10], and smaller by approximately the delay of a 2:1 multiplexer compared to that of the circuit reported in [3] because of its parallelism.

We have synthesized the circuit using 0.18μm CMOS standard cell library and estimated its area and computation time. The computation time of the proposed circuit is estimated to over 35% shorter than that of the circuit proposed in [10].

This report is organized as follows. In the next section, we explain arithmetic operations in GF(2^m) and the previously proposed division algorithms based on the extended Euclid's algorithm. In section 3., we propose a fast hardware algorithm for division in GF(2^m) with parallelization of two modular reductions. In section 4., we show a design of a circuit based on this algorithm and estimate the area and the computation time of the circuit.

2. Preliminaries

2.1 Arithmetic Operations in GF(2^m)

Let

$$G(x) = x^m + g_{m-1}x^{m-1} + \cdots + g_1x + 1$$

be an irreducible polynomial on GF(2), where $g_i \in \{0, 1\}$. Then, we can represent an arbitrary element in GF(2^m) defined by $G(x)$

as

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1x + a_0,$$

where $a_i \in \{0, 1\}$.

Addition and subtraction in $\mathrm{GF}(2^m)$ are defined as polynomial addition and subtraction on $\mathrm{GF}(2)$, respectively. Thus, both are computed with bitwise exclusive-OR operation. Multiplication "$\cdot$" in $\mathrm{GF}(2^m)$ is defined as a polynomial multiplication modulo $G(x)$ on $\mathrm{GF}(2)$. Multiplicative inverse $B^{-1}(x)$ of $B(x)$ in $\mathrm{GF}(2^m)$ is defined as the element that satisfies

$$B(x) \cdot B^{-1}(x) = 1.$$

Then, division "$\div$" in $\mathrm{GF}(2^m)$ is defined as

$$A(x) \div B(x) = A(x) \cdot B^{-1}(x).$$

In this report, an algorithm that receives three polynomials, $A(x)$, $B(x)$, and $G(x)$, and outputs $A(x) \div B(x)$ in the field defined by $G(x)$ is called division algorithm. $A(x)$ and $B(x)$ are polynomials on $\mathrm{GF}(2)$ with a degree less than m, and $G(x)$ is the irreducible polynomial on $\mathrm{GF}(2)$ with a degree m that defines the field.

2.2 Hardware Algorithm for Division in $\mathrm{GF}(2^m)$ Based on the Extended Euclid's Algorithm

Here, we describe the division algorithm proposed by Brunner et al. [3], which is a typical algorithm based on the extended Euclid's algorithm. The algorithm of Brunner et al. is as follows, where r_m and s_m denote the m-th coefficients of $R(x)$ and $S(x)$, respectively. The notation $f \wedge P(x)$ denotes

$$f \wedge P(x) = \begin{cases} P(x) & (f = 1) \\ 0 & (f = 0) \end{cases},$$

and the operation "$\times$" represents polynomial multiplication on $\mathrm{GF}(2)$. δ is a variable for determining the time of swap of the two polynomials, $R(x)$ and $S(x)$, for mutual division.

[Algorithm BCH]
(Brunner et al.'s Division Algorithm [3])

```
1:  $R(x) := B(x);\ S(x) := G(x);$
2:  $U(x) := A(x);\ V(x) := 0;$
3:  $\delta := 0;$
4:  for $i = 1$ to $2m$ do
5:    if $r_m = 0$ then
6:      $R(x) := R(x) \times x;$
7:      $U(x) := U(x) \cdot x;$
8:      $\delta := \delta + 1;$
9:    else
10:     $S(x) := (S(x) - s_m \wedge R(x)) \times x;$
11:     $V(x) := V(x) - U(x);$
12:     if $\delta = 0$ then
13:       $\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} S(x) \\ R(x) \end{bmatrix};$
14:       $\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} V(x) \\ U(x) \end{bmatrix};$
15:       $U(x) := U(x) \cdot x;$
16:       $\delta := 1;$
17:     else
18:       $U(x) := U(x) \div x;$
19:       $\delta := \delta - 1;$
20:     end if
21:   end if
22: end for
23: output $U(x)$ as the result.
```

□

3. Fast Hardware Algorithm for Division in $\mathrm{GF}(2^m)$

We propose a fast hardware algorithm for division in $\mathrm{GF}(2^m)$. First, we describe the division algorithm proposed by Guo and Wang [4]. Guo and Wang's algorithm is a modified version of Algorithm BCH, and has been developed for systolic architecture. The feature of this algorithm is that there are two for-loops in the algorithm so that we can avoid bidirectional shifts when we implement it as a circuit. This algorithm computes $(A(x) \div B(x)) \cdot x^m$ in the first for-loop, and computes $A(x) \div B(x)$ by dividing the result of the first for-loop by x^m in the second for-loop. Thus, the critical path delay of the circuit is smaller than that of the circuit based on Algorithm BCH, although its latency is $3m$ clock cycles. Guo and Wang's algorithm is as follows.

[Algorithm GW]
(Guo and Wang's Division Algorithm)

```
1:  $R(x) := B(x);\ S(x) := G(x);$
2:  $U(x) := A(x);\ V(x) := 0;$
3:  $\delta := 0;$
4:  for $i = 1$ to $2m$ do
5:    if $r_m = 0$ then
6:      $R(x) := R(x) \times x;$
7:      $U(x) := U(x) \cdot x;$
8:      $\delta := \delta + 1;$
9:    else
10:     $S(x) := (S(x) - s_m \wedge R(x)) \times x;$
11:     $V(x) := (V(x) - s_m \wedge U(x)) \cdot x;$
12:     if $\delta = 0$ then
13:       $\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} S(x) \\ R(x) \end{bmatrix};$
14:       $\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} V(x) \\ U(x) \end{bmatrix};$
```

```
15:           $\delta := \delta + 1$;
16:        else
17:           $\delta := \delta - 1$;
18:        end if
19:     end if
20: end for
21: for $i = 1$ to $m$ do
22:     $U(x) := U(x) \div x$;
23: end for
24: output $U(x)$ as the result.
```

□

In Algorithm GW, by allowing δ to be negative, shift registers can be reduced as [6], [9], when implementing the algorithm as a sequential circuit. The modified algorithm is as follows, where the notation $SEL(flag, A(x), B(x))$ denotes

$$SEL(flag, A(x), B(x)) = \begin{cases} A(x) & \textit{if flag} = 1 \\ B(x) & \textit{otherwise} \end{cases},$$

and $SGN(a)$ denotes

$$SGN(a) = \begin{cases} 1 & \textit{if } a < 0 \\ 0 & \textit{otherwise} \end{cases}.$$

Note that *swap* is the variable employed as a flag for deciding whether the algorithm assigns $R(x)$ to $S(x)$.

[Algorithm MGW]

(Modified Version of Guo and Wang's Division Algorithm)

```
1: $R(x) := B(x)$;  $S(x) := G(x)$;
2: $U(x) := A(x)$;  $V(x) := 0$;
3: $\delta := 0$;
4: for $i = 1$ to $2m$ do
5:     $swap := SGN(\delta) \wedge r_m$;
6:     $\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} (R(x) - r_m \wedge S(x)) \times x \\ SEL(swap, R(x), S(x)) \end{bmatrix}$;
7:     $\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \cdot x \\ SEL(swap, U(x), V(x)) \end{bmatrix}$;
8:     $\delta := (-1)^{swap}\delta - 1$;
9: end for
10: for $i = 1$ to $m$ do
11:     $V(x) := V(x) \div x$;
12: end for
13: output $V(x)$ as the result.
```

□

Figure 1 shows an example of division by Algorithm MGW, where $m = 4$, $A(x) = x^2 + x$, $B(x) = x^3 + x$, and $G(x) = x^4 + x + 1$.

i	R	S	U	V	δ	
	01010	10011	0110	0000	0	
1	10100	10011	1100	0000	−1	
2	01110	10100	1011	1100	0	
3	11100	10100	0101	1100	−1	
4	10000	11100	0001	0101	0	1st for-loop
5	11000	11100	1000	0101	−1	
6	01000	11000	1001	1000	0	
7	10000	11000	0001	1000	−1	
8	10000	10000	0001	0001	0	
1				1001		
2				1101		2nd for-loop
3				1111		
4				$\underline{1110}$		

Fig. 1 Example of Division by Algorithm MGW ($m = 4, A(x) = x^2 + x, B(x) = x^3 + x, G(x) = x^4 + x + 1$)

Next, we describe the algorithm to be proposed using Algorithm MGW. We start with merging the second for-loop of Algorithm MGW into its first for-loop. Since the operation in line 7 of Algorithm MGW is performed exactly $2m$ times, we can perform this merger by replacing m arbitrary chosen operations out of the $2m$ operations in line 7 with

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \\ SEL(swap, U(x), V(x)) \div x \end{bmatrix};.$$

For this purpose, we modify the algorithm so that it performs the operations of two iterations in one iteration of the first for-loop, and replace one of the two operations that update $U(x)$ and $V(x)$ with the above expression.

By the above modification, the operations for $U(x)$ and $V(x)$ in one iteration of the merged algorithm can be represented as

$$\begin{bmatrix} U'(x) \\ V'(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \cdot x \\ SEL(swap, U(x), V(x)) \end{bmatrix};$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) - r'_m \wedge V'(x)) \\ SEL(swap', U'(x), V'(x)) \div x \end{bmatrix};.$$

Note that, in the above operations, two modular reductions are performed sequentially, where $U'(x)$ and $V'(x)$ are intermediate variables for $U(x)$ and $V(x)$, respectively, and r'_m and $swap'$ are obtained from the result of the first operation as

$$r'_m := r_{m-1} \oplus (r_m \wedge s_{m-1})$$
$$swap' := SGN\left((-1)^{swap}\delta - 1\right) \wedge r'_m.$$

Finally, we modify the timing of polynomial reduction in the above operations as

$$\begin{bmatrix} U'(x) \\ V'(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \times x \\ SEL(swap, U(x), V(x)) \end{bmatrix};$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) - r'_m \wedge V'(x)) \bmod G(x) \\ SEL(swap', U'(x), V'(x)) \div x \end{bmatrix};.$$

i	R'	S'	U'	V'	δ'	R	S	U	V	δ
						01010	10011	0110	0000	0
1	10100	10011	01100	0000	-1	01110	10100	1100	0110	0
2	11100	10100	11000	0110	-1	10000	11100	1101	1100	0
3	11000	11100	00010	1100	-1	01000	11000	1110	0001	0
4	10000	11000	11100	0001	-1	10000	10000	1110	$\underline{1110}$	0

Fig. 2 Example of Division by Algorithm DEEA ($m = 4, A(x) = x^2 + x, B(x) = x^3 + x, G(x) = x^4 + x + 1$)

so that the two polynomial reductions are performed in parallel. Note that, since the constant term of $U'(x)$ and m-th coefficient of $V'(x)$ will always be zero, modular reductions of the above expressions are performed as

$$\begin{aligned} u'_j &:= u_{j-1} \oplus (r_m \wedge v_{j-1})\,; \\ v'_j &:= SEL\,(swap, u_j, v_j) \\ u_j &:= u'_j \oplus \left(u'_m \wedge g_j\right) \oplus \left(r'_m \wedge v'_j\right)\,; \\ v_j &:= SEL\left(swap', u'_{j+1}, v'_{i+1} \oplus \left(v'_0 \wedge g_{i+1}\right)\right) \end{aligned} \tag{1}$$

where u'_j, v'_j, and g_j denote the j-th coefficients of $U'(x)$, $V'(x)$, and $G(x)$, respectively. The proposed hardware algorithm is as follows.

[Algorithm DEEA]

(Proposed Division Algorithm)

```
1:  $R(x) := B(x);\ S(x) := G(x);$
2:  $U(x) := A(x);\ V(x) := 0;$
3:  $\delta := 0;$
4:  for $i = 1$ to $m$ do
5:      $swap := SGN(\delta) \wedge r_m;$
6:      $\begin{bmatrix} R'(x) \\ S'(x) \end{bmatrix} := \begin{bmatrix} (R(x) - r_m \wedge S(x)) \times x \\ SEL\,(swap, R(x), S(x)) \end{bmatrix};$
7:      $\begin{bmatrix} U'(x) \\ V'(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \times x \\ SEL\,(swap, U(x), V(x)) \end{bmatrix};$
8:      $\delta' := (-1)^{swap}\delta - 1;$
9:      $swap' := SGN(\delta') \wedge r'_m;$
10:     $\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} (R'(x) - r'_m \wedge S'(x)) \times x \\ SEL\,(swap', R'(x), S'(x)) \end{bmatrix};$
11:     $\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) - r'_m \wedge V'(x)) \bmod G(x) \\ SEL\,(swap', U'(x), V'(x)) \div x \end{bmatrix};$
12:     $\delta := (-1)^{swap'}\delta' - 1;$
13: end for
14: output $V(x)$ as the result.
```

□

Figure 2 shows an example of division by the proposed algorithm, where $m = 4$, $A(x) = x^2 + x$, $B(x) = x^3 + x$, and $G(x) = x^4 + x + 1$.

4. Circuit Based on the Proposed Algorithm and Its Evaluation

We have designed a sequential circuit that performs the operations in one iteration of the proposed algorithm in a cycle. Figure 3 shows a block diagram of the circuit. Figures 4–6 show basic cells in the circuit. *Reg-R*, *Reg-S*, *Reg-U*, *Reg-V*, *Reg-G*, *Reg-Δ*, and *Reg-sgn* are registers for storing $R(x)$, $S(x)$, $U(x)$, $V(x)$, $G(x)$, $2^{m-|\delta|}$, and the sign of δ, respectively. Figure 7 shows the controller of the circuit. Note that, in order to accelerate the circuit, we employ 1-hot counter for δ that consists of *Reg-Δ*, which holds $\Delta = 2^{m-|\delta|}$ instead of δ, and *Reg-sgn*, which holds 1 if δ is negative. *RS-calc* is the part that updates the polynomials $R(x)$ and $S(x)$ as

$$\begin{aligned} r'_j &:= r_{j-1} \oplus (r_m \wedge s_{j-1})\,; \\ s'_j &:= SEL\,(swap, r_j, s_j)\,; \\ r_j &:= r'_{j-1} \oplus \left(r'_m \wedge s'_{j-1}\right)\,; \\ s_j &:= SEL\left(swap', r'_j, s'_j\right)\,; \end{aligned}$$

and consists of $(m+1)$ *RS-cells*. *UV-calc* is the part that updates the polynomials $U(x)$ and $V(x)$ according to expressions (1) and consists of m *UV-cells* and one *UV-cell2* at the extreme left of the figure. *Δ-calc* is the part that updates Δ and consists of $(m+1)$ *Δ-cells*.

The control signals of the circuit, *swap*, *swap′*, *shift*1, and *shift*2, are computed as

$$\begin{aligned} swap &:= sgn \wedge r_m \\ swap' &:= sgn' \wedge r'_m \\ shift1 &:= (\delta_m \wedge \overline{sgn}) \vee (sgn \wedge \overline{r_m}) \\ shift2 &:= \left(\delta'_m \wedge \overline{sgn'}\right) \vee \left(sgn' \wedge \overline{r'_m}\right). \end{aligned}$$

The value of sgn' is the same as that of *shift*1, and the value of sgn in the next iteration is the same as that of *shift*2.

We compared the circuit based on the proposed algorithm with two previously proposed circuits designed as sequential circuits. Table 1 shows a comparison of the circuits. The critical path delay of the circuit based on this algorithm is larger by the delay of a 2-input XOR gate compared to that of the circuit reported in [10]. The circuit based on the proposed algorithm has m clock cycle latency, which is almost half of that of the previously proposed circuits.

Table 1 Comparison of the Circuits

	Brunner et al. [3]	Kim et al. [10]	Proposed
Latency [clock cycle]	$2m$	$2m-1$	m
Critical Path Delay	$2T_A + 2T_X + 2T_M$	$2T_A + 2T_X$	$2T_A + 3T_X$
Register Size [bit]	$4m + 2 + \lceil \log_2(m+1) \rceil$	$5m+2$	$5m+4$
# of Gates for Basic Cells			
2-input AND gate	$3m+1$	$3m$	$6m+3$
2-input XOR gate	$3m+1$	$3m$	$6m+3$
2:1 MUX	$8m+1$	$3m$	$6m+4$

T_A: the delay of a 2-input AND gate
T_X: the delay of a 2-input XOR gate
T_M: the delay of a 2:1 MUX

Table 2 Estimations of the Circuits

m	Circuit	Area [mm^2]	Critical path delay [ns]	# of cycle	Comp. time [ns]
163	Kim et al. [10]	0.1150	1.30	325	422.5
	Proposed	0.1576	1.67	163	272.2
233	Kim et al.	0.1648	1.33	465	618.5
	Proposed	0.2266	1.67	233	389.1
283	Kim et al.	0.2000	1.34	565	757.1
	Proposed	0.2707	1.71	283	483.9
409	Kim et al.	0.2869	1.38	816	1127.5
	Proposed	0.3782	1.78	409	728.0
571	Kim et al.	0.3936	1.42	1141	1620.2
	Proposed	0.5585	1.73	571	987.8

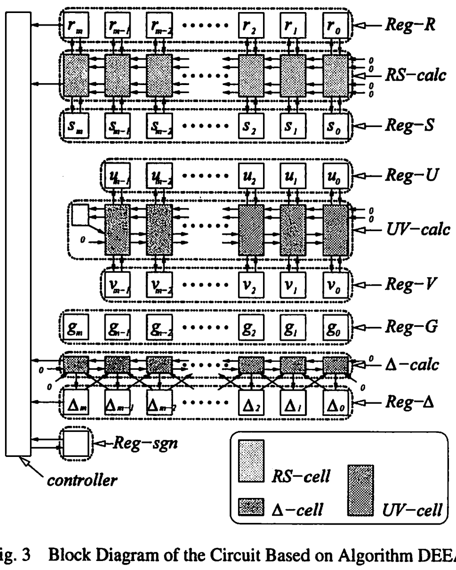

Fig. 3 Block Diagram of the Circuit Based on Algorithm DEEA

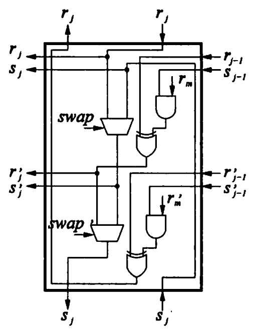

Fig. 4 *RS-cell*

We synthesized the two circuits with Synopsys Design Compiler using Rohm 0.18μm CMOS standard cell library provided by VLSI Design and Education Center (VDEC), the University of Tokyo. One is the circuit described in this section and the other is the circuit proposed in [10], Table 2 shows the synthesis results. We set 0 as area constraint and various values as critical path delay constraints. The figures in the table are the best AT-product ones obtained with the synthesis. The area of the proposed circuit is 40% or less larger than that of the circuit proposed in [10]. The computation time of the proposed circuit is over 35% shorter than that of the circuit proposed in [10].

5. Concluding Remarks

We have proposed a fast hardware algorithm for division in GF(2^m). It is based on the extended Euclid's algorithm, and requires only one iteration to perform the operations that require two iterations in previously reported division algorithms with parallel execution of modular reductions.

We have designed a circuit based on the proposed algorithm. The

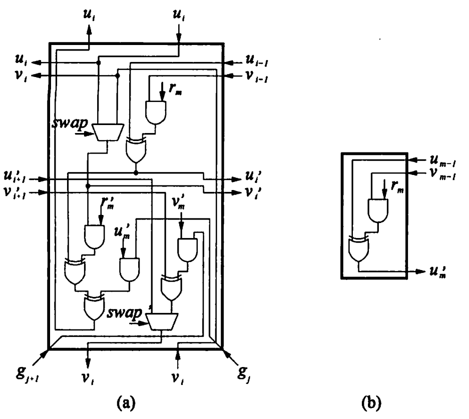

Fig. 5 *UV-cell* (a) and *UV-cell2* (b)

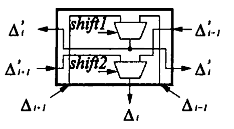

Fig. 6 Δ-*cell*

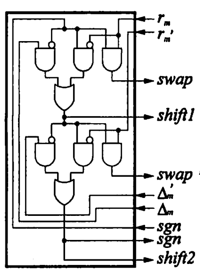

Fig. 7 Controller

circuit has m clock cycle latency, which is almost half of that of the previously proposed circuits. It has almost the same critical path delay as previously proposed circuits because of its parallelism. Therefore, it can compute division in $\mathrm{GF}(2^m)$ much faster than the previously proposed circuits. The computation time of the proposed circuit is over 35% shorter than that of the circuit proposed in [10].

In order to reduce the area of the circuit, we can employ a two-level 1-hot counter [9] for storing the value of $|\delta|$. It consists of δ_h-bit and δ_l-bit 1-hot counters, where $m + 1 \leqq \delta_h \cdot \delta_l$ and $\delta_h \approx \delta_l \approx \sqrt{m}$. Thus, we can reduce the register size and the number of 2:1 multiplexer significantly without a high cost for critical path delay.

Acknowledgment

The authors would like to thank Associate Prof. Kazuyoshi Takagi of Nagoya University for his valuable advice. This work is partially supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Rohm, Inc.

References

[1] C. Wang and J. Guo, "New systolic arrays for $C + AB^2$, inversion, and division on $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.49, no.10, pp.1120–1125, Oct. 2000.

[2] N. Takagi, J. Yoshiki, and K. Takagi, "A fast algorithm for multiplicative inversion in $\mathrm{GF}(2^m)$ using normal basis," IEEE Trans. Comput., vol.50, no.5, pp.394–398, May 2001.

[3] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.42, no.8, pp.1010–1015, Aug. 1993.

[4] J. Guo and C. Wang, "Systolic array implementation of Euclid's algorithm for inversion and division in $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.47, no.10, pp.1161–1167, Oct. 1998.

[5] C. Huang and C. Wu, "High-speed easily testable Galois-field inverter," IEEE Trans. Circuits & Systems II, vol.48, no.9, pp.909–918, Sept. 2000.

[6] A.K. Daneshbeh and M.A. Hasan, "A class of unidirectional bit serial systolic architectures for multiplicative inversion and division over $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.54, no.2, pp.370–380, March 2005.

[7] K. Araki, I. Fujita, and M. Morisue, "Fast inverters over finite field based on Euclid's algorithm," Trans. IEICE, vol.E–72, no.11, pp.1230–1234, Nov. 1989.

[8] Z. Yan and D.V. Sarwate, "New systolic architectures for inversion and division in $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.52, no.11, pp.1514–1519, Nov. 2003.

[9] Y. Watanabe, N. Takagi, and K. Takagi, "A VLSI algorithm for division in $\mathrm{GF}(2^m)$ based on extended binary GCD algorithm," IEICE Trans. Fund., vol.E85–A, no.5, pp.994–999, May 2002.

[10] C.H. Kim, S. Kwon, J.J. Kim, and C.P. Hong, "A compact and fast division architecture for a finite field $GF(2^m)$," International Conference on Computational Science and Its Applications – ICCSA 2003, pp.855–864, 2003.

[11] 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 $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.53, no.3, pp.375–380, March 2004.

[12] C. Wang and J. Lin, "A systolic architecture for computing inverses and divisions in finite fields $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.42, no.9, pp.1141–1146, Sept. 1993.

[13] M.A. Hasan and V.K. Bhargava, "Bit-serial systolic divider and multiplier for finite fields $\mathrm{GF}(2^m)$," IEEE Trans. Comput., vol.41, no.8, pp.972–980, Aug. 1992.