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

Katsuki Kobayashi, *Student Member, IEEE*, and Naofumi Takagi, *Senior Member, IEEE*

***Abstract*—We propose a fast hardware algorithm for division in GF(2^m) based on the extended Euclid's algorithm. The algorithm requires only one iteration to perform the operations that correspond to the ones performed in two iterations of previously reported division algorithms. 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.**

***Index Terms*—Division, Euclid's algorithm, Galois field, hardware algorithm.**

I. Introduction

GALOIS field GF(2^m) has many applications, particularly in elliptic curve cryptography. To accelerate such applications, high-speed 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 brief, we propose a fast hardware algorithm for division in GF(2^m) suitable for sequential circuit implementation.

In general, one of the following three methods is employed for division in the Galois field: Fermat's little theorem [1], [2], the extended Euclid's algorithm [3]–[8], or a solution of a system of linear equations [9], [10]. 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 easily can be implemented and have lower area–time products [5], [10]. The algorithm to be proposed in this brief is based on the extended Euclid's algorithm.

The proposed algorithm requires only one iteration to perform the operations that correspond to the operations performed in two iterations of previously reported division algorithms based on the extended Euclid's algorithm. In a division algorithm based on the extended Euclid's algorithm, modular reductions are performed. The previously reported division algorithms sequentially perform two modular reductions in two successive iterations. On the other hand, the proposed algorithm performs them in parallel by changing the order of execution of the operations.

We have designed a sequential circuit that performs the operations in one iteration of the proposed algorithm in one clock cycle. The circuit has a latency of m clock cycles, which is almost half of that of the circuits proposed in [3] and [4]. The critical path delay of the circuit is larger by the delay of a two-input XOR gate compared to that of the circuit proposed in [4] and smaller by approximately the delay of a 2 : 1 multiplexer compared to that of the circuit proposed in [3] because of its parallelization of modular reductions.

This brief is organized as follows. In Section II, we explain arithmetic operations in GF(2^m), the division algorithm based on the extended Euclid's algorithm proposed by Guo and Wang [5], and its modification for developing the proposed algorithm. In Section III, we propose a fast hardware algorithm for division in GF(2^m) with parallelization of two modular reductions. In Section IV, we show a design of a circuit based on the proposed algorithm and estimate its computation time and area.

II. Preliminaries

A. Arithmetic Operations in GF(2^m)

Let

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

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

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1x + a_0 \tag{2}$$

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

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

$$B(x) \cdot B^{-1}(x) = 1. \tag{3}$$

Then, division "÷" in GF(2^m) is defined as

$$A(x) \div B(x) = A(x) \cdot B^{-1}(x). \tag{4}$$

Manuscript received September 30, 2008; revised March 4, 2009. First published July 10, 2009; current version published August 14, 2009. This work was supported in part by the VLSI Design and Education Center, The University of Tokyo, in collaboration with Synopsys, Inc., and Rohm, Inc. This paper was recommended by Associate Editor A. Y. Wu.

The authors are with the Department of Information Engineering, Graduate School of Information Science, Nagoya University, Nagoya 464-8603, Japan (e-mail: katsu@takagi.i.is.nagoya-u.ac.jp; ntakagi@takagi.i.is.nagoya-u.ac.jp).

Digital Object Identifier 10.1109/TCSII.2009.2024253

1549-7747/$26.00 © 2009 IEEE

B. Guo and Wang's Division Algorithm in GF(2^m)

Here, we describe the hardware algorithm for division in GF(2^m) proposed by Guo and Wang [5] and its modification. We will employ the modified algorithm for developing our new algorithm, which is suitable for sequential circuit implementation.

First, we describe Guo and Wang's algorithm developed for systolic circuit implementation. The feature of this algorithm is that there are two for-loops in it so that bidirectional shifts can be avoided when it is implemented 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.

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

```
$R(x) := B(x)$; $S(x) := G(x)$;
$U(x) := A(x)$; $V(x) := 0$;
$\delta := 0$;
for $i = 1$ to $2m$ do
  if $r_m = 0$ then
    $R(x) := R(x) \times x$;
    $U(x) := U(x) \times x \bmod G(x)$;
    $\delta := \delta + 1$;
  else
    $S(x) := (S(x) - s_m \wedge R(x)) \times x$;
    $V(x) := (V(x) - s_m \wedge U(x)) \times x \bmod G(x)$;
    if $\delta = 0$ then
      $\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} S(x) \\ R(x) \end{bmatrix}$;
      $\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} V(x) \\ U(x) \end{bmatrix}$;
      $\delta := \delta + 1$;
    else
      $\delta := \delta - 1$;
    end if
  end if
end for
for $i = 1$ to $m$ do
  $U(x) := U(x)/x \bmod G(x)$;
end for
output $U(x)$ as the result.
```

Note that in Algorithm GW, the modular operations "$U(x) \times x \bmod G(x)$" and "$U(x)/x \bmod G(x)$" can be calculated as

$$u_j := (u_{m-1} \wedge g_j) \oplus u_{j-1} \tag{5}$$

$$u_j := (u_0 \wedge g_{j+1}) \oplus u_{j+1} \tag{6}$$

for $0 \leq j < m$, respectively. Thus, both multiplying by x and dividing by x modulo $G(x)$ can be computed with m two-input AND gates and m two-input XOR gates. The computation time of each operation is $T_X + T_A$, where T_X and T_A mean the delay of a two-input XOR gate and a two-input AND gate, respectively.

Next, to make Algorithm GW suitable for sequential circuit implementation, we modify the algorithm so that the value of δ can be negative. By this modification, shift registers can be reduced as in [6]–[8] for sequential circuit implementation. The first for-loop of the following algorithm is the same as Yan and Sarwate's inversion algorithm [8], except that the latter does not perform polynomial reduction on $U(x)$.

The modified algorithm is given as follows, where the notation $SEL(flag, A(x), B(x))$ denotes

$$SEL\,(flag, A(x), B(x)) = \begin{cases} A(x), & \text{if } flag = 1 \\ B(x), & \text{otherwise} \end{cases} \tag{7}$$

and the notation $SGN(a)$ denotes

$$SGN(a) = \begin{cases} 1, & \text{if } a < 0 \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

$swap$ is a 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:   $\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \times x \bmod G(x) \\ SEL(swap, U(x), V(x)) \end{bmatrix}$;
8:   $\delta := (-1)^{swap}\delta - 1$;
9: end for
10: for $i = 1$ to $m$ do
11:   $V(x) := V(x)/x \bmod G(x)$;
12: end for
13: output $V(x)$ as the result.
```

Note that in the first for-loop of Algorithm MGW, the operation for updating $U(x)$ takes the maximum time because of modular reduction. This operation is represented as

$$u_j := u_{j-1} \oplus (r_m \wedge v_{j-1}) \oplus ((u_{m-1} \oplus (r_m \wedge v_{m-1})) \wedge g_j) \tag{9}$$

where u_j and v_j denote the jth coefficients of $U(x)$ and $V(x)$, respectively. Thus, if we implement the first for-loop of Algorithm MGW as a sequential circuit that performs the operation in one iteration of the algorithm in a cycle, the critical path delay of the circuit will be $2T_X + 2T_A$. This value is the same as that of the circuit proposed in [4] and smaller than that of the circuit proposed in [3], which are compared with a circuit based on the proposed algorithm in Section IV.

III. New Fast Algorithm for Division in GF(2^m)

In this section, we propose a fast hardware algorithm for division in GF(2^m). We employ Algorithm MGW for developing the algorithm to be proposed in this section.

First, we start with merging the second for-loop of Algorithm MGW into the 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

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \\ SEL\,(swap, U(x), V(x))\,/x \bmod G(x) \end{bmatrix}. \tag{10}$$

Thus, 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

$$\begin{bmatrix} U'(x) \\ V'(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \times x \bmod G(x) \\ SEL\,(swap, U(x), V(x)) \end{bmatrix} \tag{11}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) - r'_m \wedge V'(x)) \\ SEL\,(swap', U'(x), V'(x))\,/x \bmod G(x) \end{bmatrix}. \tag{12}$$

Here, $U'(x)$ 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

$$\begin{cases} r'_m := r_{m-1} \oplus (r_m \wedge s_{m-1}) \\ swap' := SGN\,((-1)^{swap}\delta - 1) \wedge r'_m. \end{cases} \tag{13}$$

Note that in (11) and (12), two modular reductions are sequentially performed.

Next, we modify the time of polynomial reduction in the above operations to parallelize the modular reductions. The reason why the modular reductions in (11) and (12) are sequentially performed is that the operation for updating $U(x)$ in (12) depends on $U'(x)$ in (11), which requires a modular reduction. Let us consider removing the modular reduction in (11). In that case, $U'(x)$ will be a polynomial with degree m or less and its constant term will be zero. Thus, to update $U(x)$ by (12), we need to perform a polynomial reduction of $U'(x)$. However, we do not need to perform a polynomial reduction of $V'(x)$ because it is already a polynomial with degree less than m. Similarly, to update $V(x)$ by (12), we need to perform a polynomial reduction of $V'(x)/x$. However, we do not need to perform a polynomial reduction of $U'(x)/x$ because its constant term is zero. Therefore, we can replace (11) and (12) with

$$\begin{bmatrix} U'(x) \\ V'(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \times x \\ SEL\,(swap, U(x), V(x)) \end{bmatrix} \tag{14}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) \bmod G(x)) - r'_m \wedge V'(x) \\ SEL\,(swap', U'(x), V'(x)/x \bmod G(x)) \end{bmatrix} \tag{15}$$

without any influence on the result. The above two operations are performed as

$$\begin{cases} u'_j := u_{j-1} \oplus (r_m \wedge v_{j-1}); \\ v'_j := SEL(swap, u_j, v_j) \\ u_j := u'_j \oplus (u'_m \wedge g_j) \oplus (r'_m \wedge v'_j)\,; \\ v_j := SEL\,(swap', u'_{j+1}, v'_{i+1} \oplus (v'_0 \wedge g_{i+1})) \end{cases} \tag{16}$$

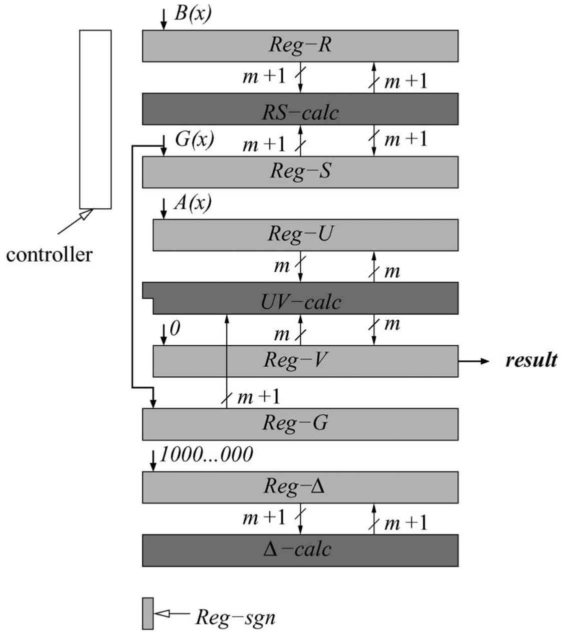

Fig. 1. Block diagram of the circuit based on Algorithm DEEA.

where u'_j, v'_j, and g_j denote the jth coefficients of $U'(x)$, $V'(x)$, and $G(x)$, respectively. The proposed hardware algorithm is given 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:  $\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) \bmod G(x)) - r'_m \wedge V(x)' \\ SEL(swap', U'(x), V'(x)/x \bmod G(x)) \end{bmatrix};$
12:  $\delta := (-1)^{swap'}\delta' - 1;$
13: end for
14: output $V(x)$ as the result.
```

IV. Circuit Based on the Proposed Algorithm

A. Circuit Design

We have designed a sequential circuit that performs the operations in one iteration of the proposed algorithm in a cycle. Fig. 1 shows a block diagram of the circuit. Fig. 2(a)–(d) shows the basic cells of the circuit. Reg-R, Reg-S, Reg-U, Reg-V, Reg-G, Reg-Δ, and Reg-sgn are registers for storing $R(x)$, $S(x)$, $U(x)$, $V(x)$, $G(x)$, $\Delta(= 2^{m-|\delta|})$, and the sign of δ, respectively. Fig. 2(e) shows the controller of the circuit. Note that to accelerate the circuit, we employ 1-hot counter for δ that

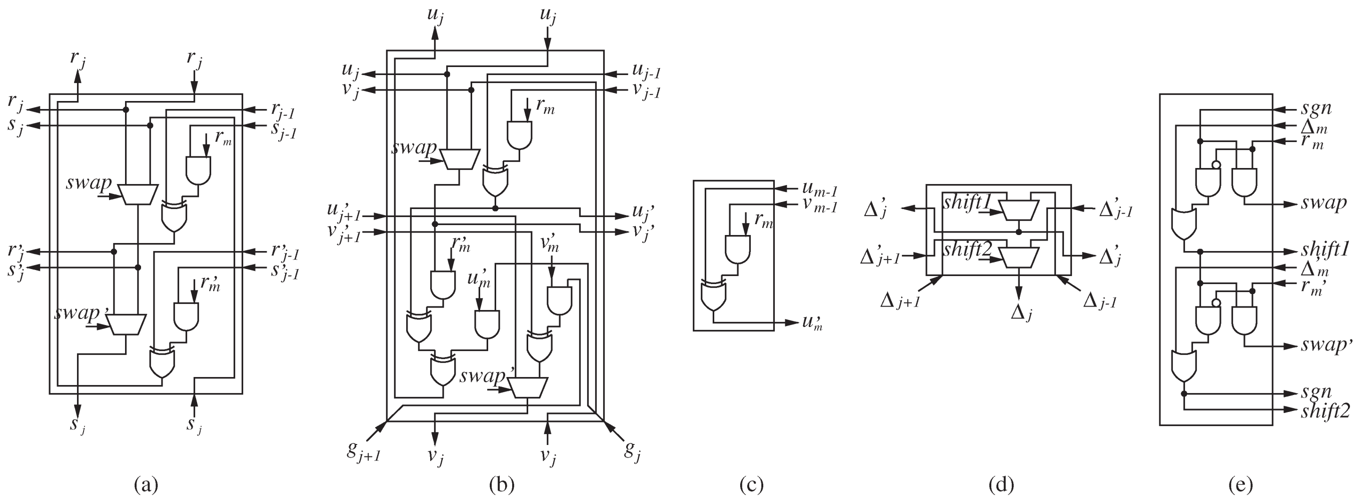

Fig. 2. (a) RS-cell. (b) UV-cell. (c) UV-cell2. (d) Δ-cell. (e) Controller.

consists of Reg-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

$$\begin{cases} r'_j := r_{j-1} \oplus (r_m \wedge s_{j-1}) \\ s'_j := SEL(swap, r_j, s_j) \\ r_j := r'_{j-1} \oplus \left(r'_m \wedge s'_{j-1}\right) \\ s_j := SEL\left(swap', r'_j, s'_j\right) \end{cases} \tag{17}$$

and consists of $(m+1)$ RS-cells. UV-calc is the part that updates the polynomials $U(x)$ and $V(x)$ according to (16) and consists of m UV-cells and one UV-cell2 at the far left of UV-calc. Δ-calc is the part that updates Δ and consists of $(m+1)\Delta$-cells.

The value of $|\delta|$ is updated as

$$|\delta| := \begin{cases} |\delta| + 1, & \text{if } \delta = 0 \text{ or } (\delta < 0 \text{ and } r_m = 0) \\ |\delta| - 1, & \text{otherwise} \end{cases} \tag{18}$$

and the sign of δ "sgn" is updated as

$$sgn := \begin{cases} 1, & \text{if } \delta = 0 \text{ or } (\delta < 0 \text{ and } r_m = 0) \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

Therefore, the control signals of the circuit, i.e., $swap$, $swap'$, $shift1$, and $shift2$, are computed as

$$\begin{cases} swap := sgn \wedge r_m \\ swap' := sgn' \wedge r'_m \\ sgn' := \Delta'_m \vee (sgn \wedge \overline{r_m}) \\ sgn := \Delta_m \vee \left(sgn' \wedge \overline{r'_m}\right) \\ shift1 := \Delta'_m \vee (sgn \wedge \overline{r_m}) \\ shift2 := \Delta_m \vee \left(sgn' \wedge \overline{r'_m}\right). \end{cases} \tag{20}$$

Note that the value of sgn' is the same as that of $shift1$, and the value of sgn in the next iteration is the same as that of $shift2$.

From (16)–(20), the critical path of a circuit based on the proposed algorithm will be the path from r_m to u_j via r'_m with a delay of $3T_X + 2T_A$. Since a circuit based on the proposed circuit has a bit-slice structure, we can perform place and route with relative ease.

B. Comparison With Previously Proposed Circuits

We have compared the circuit based on the proposed algorithm with two previously proposed division circuits designed as sequential circuits. Note that for fair comparison about computation time, we have employed a 1-hot counter as the counter of Brunner *et al.*'s division circuit, although the original circuit employs a binary counter. Table I shows a comparison of the circuits, where the number of transistors is based on the assumption that a two-input AND gate, a two-input OR gate, a two-input XOR gate, a 2 : 1 multiplexer, and a 1-bit latch consist of four, six, six, six, and eight transistors, respectively, as in [4].

The critical path delay of the circuit based on this algorithm is larger by the delay of a two-input XOR gate compared to that of the circuit proposed in [4]. The circuit based on the proposed algorithm has a latency of m clock cycles, which is almost half of that of the previously proposed circuits.

C. Logic Synthesis Results

We have described the circuit designed in Section IV-A with SystemVerilog. To evaluate the proposed algorithm, we have also described three circuits. One is the circuit proposed by Brunner *et al.* [3], another is the circuit proposed by Kim *et al.* [4], and the third is the circuit that performs the operations performed in two clock cycles of Kim *et al.*'s circuit in one clock cycle and denoted as "Kim *et al.*'s (duplicated)" in what follows.

We have synthesized them with Synopsys Design Compiler using the Rohm 0.18-μm CMOS standard cell library provided by the VLSI Design and Education Center, University of Tokyo. Table II shows the result of logic synthesis. The computation time is the product of the critical path delay and the latency.

The computation time of the circuit based on the proposed algorithm has been estimated to be over 35% smaller than that of the previously proposed circuits, and approximately 10%

TABLE I
COMPARISON OF THE CIRCUITS

	Brunner et al. [3] (with 1-hot counter)	Kim et al. [4]	Proposed
Latency [clock cycle]	$2m$	$2m-1$	m
Throughput [result / cycle]	$1/2m$	$1/(2m-1)$	$1/m$
Critical Path Delay	$2T_A+2T_X+2T_M$	$2T_A+2T_X$	$2T_A+3T_X$
Register Size [bit]	$6m+4$	$6m+3$	$6m+4$
# of Gates			
2-input AND gate	$3m+4$	$3m+5$	$6m+7$
2-input OR gate	1	1	2
2-input XOR gate	$3m+1$	$3m+1$	$6m+3$
2:1 MUX	$9m+5$	$3m$	$6m+4$
# of Transistors	$132m+90$	$96m+56$	$144m+114$
AT-product	$O(m^2)$	$O(m^2)$	$O(m^2)$

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 II
LOGIC SYNTHESIS RESULTS (0.18-μm CMOS TECHNOLOGY)

m	Circuit	Critical Path Delay [ns]	Area [mm^2]	Comp. Time [ns]	# of Cells
128	Kim et al.'s [4]	1.231	0.1172	313.9	3,445
	Kim et al.'s (duplicated)	1.834	0.1493	234.8	5,211
	Brunner et al.'s [3]	1.882	0.1425	481.8	5,555
	Proposed	1.582	0.1316	202.6	4,536
256	Kim et al.'s	1.289	0.2219	658.7	7,328
	Kim et al.'s (duplicated)	1.818	0.2881	465.4	9,527
	Brunner et al.'s	1.826	0.2692	934.9	8,774
	Proposed	1.576	0.2955	403.5	10,450
512	Kim et al.'s	1.343	0.4103	1,343.2	13,447
	Kim et al.'s (duplicated)	1.838	0.5411	941.1	17,196
	Brunner et al.'s	1.913	0.5358	1,958.9	18,434
	Proposed	1.612	0.5564	825.4	18,532

smaller than that of Kim *et al.*'s (duplicated). The area of the circuit based on the proposed algorithm has been estimated to be only 30% larger than that of Kim *et al.*'s circuit.

V. CONCLUSION

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 correspond to two iterations in previously reported division algorithms with parallel execution of modular reductions in one iteration.

We have designed a sequential circuit based on the proposed algorithm that performs one iteration of the algorithm in one clock cycle. The circuit has a latency of m clock cycles, which is almost half of that of the previously proposed circuits. It has almost the same critical path delay as the previously proposed circuits because of its parallelism. Therefore, it can compute division in GF(2^m) much faster than the previously proposed circuits. The area of the circuit based on the proposed algorithm has been estimated to be only 30% larger than that of the circuit proposed in [4] with logic synthesis.

We can employ a two-level 1-hot counter [7] for storing the value of $|\delta|$ to reduce the area of the circuit instead of a 1-hot counter. It consists of δ_h-bit and δ_l-bit 1-hot counters, where $m+1 \leq \delta_h \cdot \delta_l$, and $\delta_h \approx \delta_l \approx \sqrt{m}$. Thus, we can significantly reduce the register size and the number of $2:1$ multiplexer without a high cost for critical path delay.

ACKNOWLEDGMENT

The authors would like to thank Assoc. Prof. K. Takagi of Nagoya University for his valuable advice.

REFERENCES

[1] C. Wang and J. Guo, "New systolic arrays for $C + AB^2$, inversion, and division on 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 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 GF(2^m)," *IEEE Trans. Comput.*, vol. 42, no. 8, pp. 1010–1015, Aug. 1993.

[4] 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)," in *Proc. ICCSA*, 2003, pp. 855–864.

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

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

[7] Y. Watanabe, N. Takagi, and K. Takagi, "A VLSI algorithm for division in GF(2^m) based on extended binary GCD algorithm," *IEICE Trans. Fundam.*, vol. E85-A, no. 5, pp. 994–999, May 2002.

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

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

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