Studies on Hardware-assisted Implementation of Arithmetic Operations in Galois Field GF(2^m)

Katsuki Kobayashi

ABSTRACT

Galois field $\mathrm{GF}(2^m)$ has many important applications, such as cryptography and error correcting codes. For high-speed implementation of these applications, efficient implementation of arithmetic operations in $\mathrm{GF}(2^m)$ is important. In this dissertation, three methods for efficient implementation of arithmetic operations in $\mathrm{GF}(2^m)$ are proposed. These methods are based on the idea of hardware assist that yields higher-speed and lower-power-consumption implementation.

Chapter 2 shows arithmetic operations in $\mathrm{GF}(2^m)$, the extended Euclid's algorithm, and a previously proposed hardware inversion algorithm as preliminaries. Chapter 3 proposes a software algorithm for inversion in $\mathrm{GF}(2^m)$ that is suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$. Among previously proposed instruction set extensions for cryptography, ones for elliptic curve cryptography (ECC) or advanced encryption standard (AES) include a polynomial multiply instruction on $\mathrm{GF}(2)$, because this instruction can accelerate multiplication in $\mathrm{GF}(2^m)$. The algorithm proposed in the chapter employs the matrix that represents the operations required by several contiguous iterations of the previously reported algorithm, and computes inversion fast through the matrix with a polynomial multiply instruction on $\mathrm{GF}(2)$. When the word size of the processor is 32 and m is 571, the proposed algorithm computes inversion with approximately half the number of polynomial multiply instructions on $\mathrm{GF}(2)$ and `XOR` instructions required by the previously reported algorithm on the average.

Chapter 4 proposes a fast hardware division algorithm in $\mathrm{GF}(2^m)$ with parallelization of modular reductions for fast division circuit. This algorithm requires only one iteration to perform the operations required by two iterations of a previously reported algorithm, and performs two modular reductions in parallel by changing the order of execution of the operations. A circuit based on the algorithm proposed in the chapter has almost the same critical path delay as previously reported circuits, nevertheless the number of clock cycles required by the circuit is almost half of that of previously reported circuits. The circuit is estimated to be over 35% faster than previously reported circuits with logic synthesis.

Chapter 5 proposes a hardware algorithm for a combined circuit of multiplication and inversion in $\mathrm{GF}(2^m)$. Although both multiplication and inversion are employed for ECC, realization of two circuits for them yields large area. Thus, for reduction of hardware of these circuits, the algorithm proposed in the chapter is developed by focusing on the similarities between conventional multiplication and inversion algorithms so that almost all hardware components of a circuit based on the algorithm can be shared by multiplication and inversion. The combined circuit is estimated to be over 15% smaller than the previously proposed combined circuits with logic synthesis.

Finally, Chap. 6 concludes that hardware-assist is a promising technique for efficient implementation of arithmetic operations in $\mathrm{GF}(2^m)$. In addition, by focusing on similarities and parallelism of algorithms, reduction and acceleration of them are possible. The knowledge obtained through the study should make hardware-assisted implementation of arithmetic operations in $\mathrm{GF}(2^m)$ as well as other operations necessary in important applications more efficiently.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUNDS

With the spread of the Internet and mobile devices, the technologies for transferring information safely and surely become much more important than ever before. Galois field has many applications for these technologies, such as public-key cryptography, symmetric key cryptography and error correcting codes. Galois field $\mathrm{GF}(2^m)$, i.e. an extension field of $\mathrm{GF}(2)$, is the most frequently employed among various types of Galois fields, because of its ease of implementation with computers [1].

Implementation of arithmetic operations in $\mathrm{GF}(2^m)$ is studied especially in elliptic curve cryptography (ECC), because $\mathrm{GF}(2^m)$ with large m is employed for it. In this study, we focus on the hardware-assisted implementation of arithmetic operations in $\mathrm{GF}(2^m)$ because it yields higher-speed and lower-power-consumption, although there are many studies about software implementation of them [2–7].

Hardware-assisted implementation can be classified into two types. One is adding small arithmetic units to a general purpose processor and extending its instruction set for efficient implementation. Although this way has been studied especially in the area of multime-

dia, recently it has also come to be studied in the area of cryptography such as AES and ECC [8–13]. Most of these studies reported that instruction set for operations that appear in cryptographic processing, especially for multiplication in $\mathrm{GF}(2^m)$, is useful for cryptography. The other type of implementation is designing dedicated circuits for arithmetic operations in $\mathrm{GF}(2^m)$. This way is common in multiplication [14–22], multiplicative inversion [20–30], and the combined arithmetic operations such as $A^2(x) \cdot B(x) + C(x)$ [31,32].

In this dissertation, we propose three methods for hardware-assisted implementation of arithmetic operations in $\mathrm{GF}(2^m)$. One is a fast software algorithm for inversion in $\mathrm{GF}(2^m)$ that is suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$. Another is a fast hardware algorithm for division in $\mathrm{GF}(2^m)$ that parallelizes the two modular reductions required by two iterations of previously reported algorithms by changing the order of execution of the operations. The other one is a hardware algorithm for a combined circuit of multiplication and inversion. By using this algorithm, we can design a combined circuit in which almost all hardware components are shared by multiplication and inversion.

Using the methods proposed in this dissertation, we can implement arithmetic operations in $\mathrm{GF}(2^m)$ with hardware assist efficiently.

1.2 OUTLINES OF THE DISSERTATION

In Chap. 2, we show arithmetic operations in Galois field $\mathrm{GF}(2^m)$ and a previously reported hardware algorithm for inversion in $\mathrm{GF}(2^m)$ which will be employed in Chap. 3 and 5.

In Chap. 3, we propose a fast software algorithm for inversion in $\mathrm{GF}(2^m)$ suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$. In recent years, instruction set extensions for cryptosystems are studied, and many researches about them reported a polynomial multiply instruction on $\mathrm{GF}(2)$ is useful for implementation of cryptography [8–13]. The proposed algorithm in the chapter can compute inversion in $\mathrm{GF}(2^m)$

fast by means of this instruction. In this algorithm, the operations required by several contiguous iterations of previously reported algorithm are represented as a matrix computed with only single-word operations. Then, these operations are performed at once through the matrix efficiently by means of a polynomial multiply instruction on $\mathrm{GF}(2^m)$. When the word size of the processor is 32 and m is 571, the proposed algorithm compute inversion with approximately half the number of polynomial multiply instructions on $\mathrm{GF}(2)$ and `XOR` instructions compared to the conventional algorithm evaluated in [4] on the average.

In Chap. 4, we propose a hardware algorithm for division in $\mathrm{GF}(2^m)$ to develop a fast division circuit. This algorithm 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 based on the extended Euclid's algorithm. Although two iterations of the previously proposed division algorithms perform two modular reductions sequentially, the algorithm proposed in the chapter performs them in parallel by changing the order of execution of the operations. Thus, a circuit based on the proposed algorithm has almost the same critical path delay as previously reported division circuits, while the number of clock cycles required by this circuit is almost half of that required by previously reported circuits.

In Chap. 5, we propose a hardware algorithm for a combined circuit for multiplication and inversion in $\mathrm{GF}(2^m)$ by focusing on the similarities between the conventional multiplication and inversion algorithms. In a combined circuit based on the algorithm proposed in the chapter, almost all hardware components of the circuit are shared by multiplication and inversion. Thus, the circuit is effective in area-restricted devices because it can be implemented with much smaller area than previously proposed circuits. Compared with previously proposed combined circuits for multiplication and division, the circuit has several advantages. Although the area complexity of the circuit proposed in [20] is $O(m^2)$, that of the circuit is $O(m)$. The circuits proposed in [21,22] are based on the Stein's binary

GCD algorithm for division and need to reverse the order of the coefficients of inputs and output polynomials for multiplication. Thus, the circuit proposed in [22] has extra area for such pre- and post-computation. By logic synthesis, the area of the proposed circuit is estimated to be approximately over 15% smaller than that of previously proposed combined multiplication and division circuits.

In Chap. 6, we conclude this dissertation.

CHAPTER 2

PRELIMINARIES

2.1 ARITHMETIC OPERATIONS IN $\mathrm{GF}(2^m)$

Let

$$G(x) = x^m + g_{m-1}x^{m-1} + \cdots + g_1 x + 1 \tag{2.1}$$

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

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0, \tag{2.2}$$

where $a_i \in \{0, 1\}$. This representation is called the polynomial representation of an element.

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 as

$$A(x) + B(x) = (a_{m-1} \oplus b_{m-1})\, x^{m-1} + \cdots + (a_1 \oplus b_1)\, x + (a_0 \oplus b_0)\,, \tag{2.3}$$

where the operation "$\oplus$" represents the XOR operation and b_i denotes the j-th coefficient of the polynomial $B(x)$. Multiplication "$\cdot$" in $\mathrm{GF}(2^m)$ is defined as a polynomial multiplication modulo $G(x)$ on $\mathrm{GF}(2)$ as

$$A(x) \cdot B(x) = A(x) \times B(x) \bmod G(x), \tag{2.4}$$

where the operation "$\times$" represents polynomial multiplication on $\mathrm{GF}(2)$. Multiplicative inverse $B^{-1}(x)$ of $B(x)$ in $\mathrm{GF}(2^m)$ is defined as the element that satisfies

$$B(x) \cdot B^{-1}(x) = 1. \tag{2.5}$$

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

$$A(x) \div B(x) = A(x) \cdot B^{-1}(x). \tag{2.6}$$

In this dissertation, an algorithm that receives three polynomials $A(x)$, $B(x)$, and $G(x)$, and outputs $A(x) \cdot B(x)$ in the field defined by $G(x)$ is called multiplication algorithm, where $A(x)$ and $B(x)$ are polynomial representations of two elements and $G(x)$ is the irreducible polynomial that defines the field. Similarly, an algorithm that receives two polynomials $B(x)$ and $G(x)$, and outputs $B^{-1}(x)$ in the field defined by $G(x)$ is called inversion algorithm. 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.

2.2 EXTENDED EUCLID'S ALGORITHM FOR POLYNOMIAL

The Euclid's algorithm for polynomial, which calculates the greatest common divisor (GCD) polynomial of two polynomials, $B(x)$ and $G(x)$, can be extended so that it can calculate

the two polynomials, $V(x)$ and $Z(x)$, that satisfy

$$GCD\left(B(x), G(x)\right) = V(x) \times B(x) + Z(x) \times G(x). \tag{2.7}$$

The extended Euclid's algorithm is as follows, where the notation $\deg(\cdot)$ denotes the degree of a polynomial. the notation $\lfloor R(x)/S(x) \rfloor$ denotes the quotient polynomial that satisfies

$$S(x) = \lfloor S(x)/R(x) \rfloor \times R(x) + rem(x) \qquad \left(\deg(rem(x)) < \deg(S(x))\right). \tag{2.8}$$

[Algorithm EEA]

(The Extended Euclid's Algorithm for Polynomial on $\mathrm{GF}(2)$*)*

Input: $B(x), G(x)$: $\deg(G(x)) \leq \deg(B(x))$

Output: $GCD\left(B(x), G(x)\right)$, and $V(x)$ and $Z(x)$ that satisfy Eq. (2.7).

1: $R(x) := B(x);\ U(x) := 1;\ W(x) := 0;$

2: $S(x) := G(x);\ V(x) := 0;\ Z(x) := 1;$

3: **while** $R(x) \neq 0$ **do**

4: $\quad Q(x) := \lfloor S(x)/R(x) \rfloor;$

5: $$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} S(x) - R(x) \times Q(x) \\ R(x) \end{bmatrix};$$

6: $$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} V(x) - U(x) \times Q(x) \\ U(x) \end{bmatrix};$$

7: $$\begin{bmatrix} W(x) \\ Z(x) \end{bmatrix} := \begin{bmatrix} Z(x) - W(x) \times Q(x) \\ W(x) \end{bmatrix};$$

8: **end while**

9: *output* $S(x), V(x)$, and $Z(x)$ *as the results.*

	$R(x)$	$S(x)$	$U(x)$	$V(x)$	$Q(x)$
—	$x^6 + x^4$	$x^7 + x^6 + x^3 + x + 1$	1	0	—
1	$x^5 + x^4 + x^3 + x + 1$	$x^6 + x^4$	$x + 1$	1	$x + 1$
2	$x^4 + x^3 + x^2 + 1$	$x^5 + x^4 + x^3 + x + 1$	$x + 1$	1	$x + 1$
3	1	$x^4 + x^3 + x^2 + 1$	$x^3 + x + 1$	x^2	x
4	0	1	—	$x^3 + x + 1$	—

Figure 2.1: Example of Inversion by Algorithm EEA ($m = 7$, $G(x) = x^7+x^6+x^3+x+1$, $B(x) = x^6 + x^4$)

□

This algorithm can compute inversion by choosing the input values $B(x)$ and $G(x)$ as a polynomial representation of an element and the irreducible polynomial with degree m that defines the field, respectively [1]. This is explained as

$$\begin{aligned} GCD(B(x), G(x)) &= V(x) \times B(x) + Z(x) \times G(x) \\ 1 &\equiv V(x) \times B(x) \pmod{G(x)} \\ B^{-1}(x) &\equiv V(x) \pmod{G(x)}. \end{aligned} \tag{2.9}$$

This algorithm can also compute division "$A(x) \div B(x)$" by replacing the initial value $U(x)$ by $A(x)$. Note that, in this algorithm, we can compute inversion or division without computing $W(x)$ or $Z(x)$ because they are computed independently of $S(x)$ and $V(x)$. In addition, in the inversion case, we can obtain correct element without polynomial reductions of $U(x)$ or $V(x)$.

Figure 2.1 shows an example of inversion by Algorithm EEA, where $m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $B(x) = x^6 + x^4$. As described earlier, this example omits the calculation of $W(x)$ and $Z(x)$. In this example, the inverse $B^{-1}(x)$ can be obtained as $V(x)$.

2.3 PREVIOUSLY PROPOSED HARDWARE INVERSION ALGORITHM

We explain the hardware inversion algorithm in $\mathrm{GF}(2^m)$ proposed by Yan and Sarwate [29], which is a typical algorithm based on the extended Euclid's algorithm and will be employed in Chap. 3 and 5. For VLSI implementation, this algorithm checks only the m-th coefficients of two polynomials in the calculation of GCD, thus the polynomials are multiplied by some power of x relative to the proper ones. The algorithm is as follows, where r_m denotes the m-th coefficients of $R(x)$, and δ is a variable for determining the time of swap of the polynomials.

[Algorithm YS]

(*Yan and Sarwate's Inversion Algorithm* [29])

1: $S(x) := G(x)$; $V(x) := 0$;
2: $R(x) := B(x) \times x$; $U(x) := 1$;
3: $\delta := -1$;
4: **for** $i = 1$ **to** $2m - 1$ **do**
5: **if** $r_m = 0$ **then**
6: $R(x) := x \times R(x)$;
7: $U(x) := x \times U(x)$;
8: **else**
9: **if** $\delta \geq 0$ **then**
10: $R(x) := x \times (R(x) - S(x))$;
11: $U(x) := x \times (U(x) - V(x))$;
12: **else**

13: $$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} x \times (R(x) - S(x)) \\ R(x) \end{bmatrix};$$

14: $$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} x \times (U(x) - V(x)) \\ U(x) \end{bmatrix};$$

15: $\delta := -\delta$;

16: **end if**

17: **end if**

18: $\delta := \delta - 1$;

19: **end for**

20: *output* $V(x) \div x^{m-1}$ *as the result.*

□

Note that, at the end of the k-th iteration in this algorithm, all u_j and v_j are 0 for j's outside the range $[\max(0, k - m), k]$, where u_j and v_j denote the j-th coefficients of the polynomials, $U(x)$ and $V(x)$, respectively. Thus, when we implement this algorithm as a sequential circuit, $(m + 1)$-bit width is sufficient for registers of $R(x)$ and $S(x)$ by employing cyclic shift. The variable δ is employed to determine the time of swap of the two polynomials, $R(x)$ and $S(x)$, for mutual division.

Figure 2.2 shows an example of inversion by Algorithm YS, where $m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $B(x) = x^6 + x^4$.

i	$R(x)$	$S(x)$	$U(x)$	$V(x)$	δ
	x^7+x^5	$x^7+x^6+x^3+x+1$	1	0	-1
1	$x^7+x^6+x^4+x^2+x$	x^7+x^5	x	1	0
2	$x^7+x^6+x^5+x^3+x^2$	x^7+x^5	x^2+x	1	-1
3	$x^7+x^4+x^3$	$x^7+x^6+x^5+x^3+x^2$	x^3+x^2+x	x^2+x	0
4	$x^7+x^6+x^5+x^3$	$x^7+x^6+x^5+x^3+x^2$	x^4	x^2+x	-1
5	x^3	$x^7+x^6+x^5+x^3$	$x^5+x^3+x^2$	x^4	0
6	x^4	$x^7+x^6+x^5+x^3$	$x^6+x^4+x^3$	x^4	-1
7	x^5	$x^7+x^6+x^5+x^3$	$x^7+x^5+x^4$	x^4	-2
8	x^6	$x^7+x^6+x^5+x^3$	$x^8+x^6+x^5$	x^4	-3
9	x^7	$x^7+x^6+x^5+x^3$	$x^9+x^7+x^6$	x^4	-4
10	$x^7+x^6+x^4$	x^7	$x^{10}+x^8+x^7+x^5$	$x^9+x^7+x^6$	3
11	x^7+x^5	x^7	$x^{11}+x^{10}+x^9+x^7+x^6$	$x^9+x^7+x^6$	2
12	x^6	x^7	$x^{12}+x^{11}$	$x^9+x^7+x^6$	1
13	x^7	x^7	$x^{13}+x^{12}$	$\underline{x^9+x^7+x^6}$	0
				$=(x^3+x+1)\times x^6$	
				$=B^{-1}(x)\times x^{m-1}$	

Figure 2.2: Example of Inversion by Algorithm YS ($m=7$, $G(x)=x^7+x^6+x^3+x+1$, $B(x)=x^6+x^4$)

CHAPTER 3

FAST SOFTWARE INVERSION ALGORITHM IN $\mathrm{GF}(2^m)$ SUITABLE FOR IMPLEMENTATION WITH A POLYNOMIAL MULTIPLY INSTRUCTION ON $\mathrm{GF}(2)$

3.1 INTRODUCTION

Among the basic arithmetic operations in $\mathrm{GF}(2^m)$, inversion/division takes the maximum time. Therefore, high-speed implementation for inversion in $\mathrm{GF}(2^m)$ is required to accelerate applications of $\mathrm{GF}(2^m)$ such as ECC. In this chapter, we propose a fast inversion algorithm in $\mathrm{GF}(2^m)$ which is suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$.

In recent years, several instruction set extensions for cryptography, such as DES, AES, and ECC, have been developed [8–13,33,34]. These instruction set extensions can increase performance of a processor with small cost. Among them, instruction set extensions for

AES or ECC include polynomial multiply instructions on $\mathrm{GF}(2)$ because this instruction can accelerate calculation of multiplication in $\mathrm{GF}(2^m)$. The proposed algorithm in this chapter can compute inversion in $\mathrm{GF}(2^m)$ fast by means of the instruction.

The algorithm proposed in this chapter is based on the extended Euclid's algorithm. In this algorithm, the operations required by several contiguous iterations of Yan and Sarwate's inversion algorithm, which is suitable for VLSI implementation and based on the extended Euclid's algorithm [29], are represented as a matrix computed with only single-word operations. Then, these operations are performed at once through the matrix efficiently by means of a polynomial multiply instruction on $\mathrm{GF}(2)$.

When the word size of the processor is 32 and m is 571, the proposed algorithm computes inversion with approximately half the number of polynomial multiply instructions on $\mathrm{GF}(2)$ and `XOR` instructions required by the conventional algorithm evaluated in [4] on the average.

This chapter is organized as follows. In the next section, we explain the software inversion algorithm evaluated in [4], and a polynomial multiply instruction for cryptography which is reported before and considered in this chapter. In Sect. 3.3, we propose a fast inversion algorithm in $\mathrm{GF}(2^m)$ which is based on the extended Euclid's algorithm and suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$. In Sect. 3.4, we evaluate the proposed algorithm by the numbers of instructions required for the proposed algorithm and the previously reported algorithm. In Sect. 3.5, we make several discussions. In Sect. 3.6, we summarize this chapter.

3.2 PRELIMINARIES

3.2.1 PREVIOUSLY REPORTED SOFTWARE INVERSION ALGORITHMS

Hankerson et al. reported that an inversion algorithm based on the extended Euclid's algorithm is faster than the other inversion algorithms for software implementation [4]. The algorithm evaluated in [4] is as follows.

[Algorithm EEIA]

(*Software Inversion Algorithm in* $\mathrm{GF}(2^m)$ [4])

1: $S(x) := G(x)$; $V(x) := 0$;
2: $R(x) := B(x)$; $U(x) := 1$;
3: **while** $\deg(R(x)) \neq 0$ **do**
4: $\quad \delta := \deg(S(x)) - \deg(R(x))$;
5: $\quad$ **if** $\delta < 0$ **then**
6: $\quad\quad \begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} S(x) \\ R(x) \end{bmatrix}$;
7: $\quad\quad \begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} V(x) \\ U(x) \end{bmatrix}$;
8: $\quad\quad \delta := -\delta$;
9: $\quad$ **end if**
10: $\quad S(x) := S(x) - x^{\delta} \times R(x)$;
11: $\quad V(x) := V(x) - x^{\delta} \times U(x)$;
12: **end while**
13: *output* $U(x)$ *as the result.*

□

$R(x)$	$S(x)$	$U(x)$	$V(x)$	δ
x^6+x^4	$x^7+x^6+x^3+x+1$	1	0	
x^6+x^4	$x^6+x^5+x^3+x+1$	1	x	1
x^6+x^4	$x^5+x^4+x^3+x+1$	1	$x+1$	0
$x^5+x^4+x^3+x+1$	x^5+x^2+x	$x+1$	x^2+x+1	-1
$x^5+x^4+x^3+x+1$	$x^4+x^3+x^2+1$	$x+1$	x^2	0
$x^4+x^3+x^2+1$	1	x^2	x^3+x+1	-1
1	x^3+x^2+1	$\underline{x^3+x+1}$	x^6+x^3+x+1	-4

Figure 3.1: Example of Inversion by Algorithm EEIA ($m=7$, $G(x)=x^7+x^6+x^3+x+1$, $B(x)=x^6+x^4$)

Figure 3.1 shows an example of inversion by Algorithm EEIA, where $m=7$, $G(x)=x^7+x^6+x^3+x+1$ and $B(x)=x^6+x^4$. The point of this algorithm is we calculate the difference of the degrees of two polynomials instead of calculating the quotient polynomial of them, because multi-word polynomial division takes long time.

3.2.2 Polynomial Multiply Instruction for Cryptography

Here, we consider a typical polynomial multiply instruction on $\mathrm{GF}(2)$. The effectiveness of the instruction was reported in [8–11]. We call it `MULGF2` instruction as in [8]. `MULGF2` instruction calculates the 2-word product from two 1-word operands.

A polynomial multiplier on $\mathrm{GF}(2)$ can be realized as an AND-array followed by an XOR-tree, i.e. "carry-free" version of an integer multiplier. Since there are no carry propagation, such multiplier is fast and small. In addition, we can also unify this multiplier into an integer multiplier with a little cost as described in [35, 36].

3.3 Fast Software Inversion Algorithm Suitable for Implementation with Polynomial Multiply Instruction on $\mathrm{GF}(2)$

In this section, we propose a fast software inversion algorithm suitable for implementation with polynomial multiply instruction on $\mathrm{GF}(2)$. We develop the algorithm from Algorithm YS shown in Sect. 2.3. If we implement Algorithm YS directly with software, it will be too slow because this algorithm needs two 1-bit multi-word shifts in each iteration. However, we can modify this algorithm so that it will be suitable for software implementation by means of a polynomial multiply instruction on $\mathrm{GF}(2)$. We employ the matrix that represents the operations required by several contiguous iterations of Algorithm YS. The algorithm processes multiple bits in each iteration through the matrix for fast computation with only single-word operations.

First, we describe the matrix employed in the proposed algorithm. In Algorithm YS, $R(x), S(x), U(x)$, and $V(x)$ are updated according to the values of the m-th coefficients of $R(x)$. Similarly, we can decide the operations for updating $R(x)$, $S(x)$, $U(x)$, and $V(x)$ in several contiguous iterations of Algorithm YS according to several coefficients of $R(x)$ and $S(x)$. Therefore, we represent the operations for updating $R(x)$, $S(x)$, $U(x)$, and $V(x)$ in several contiguous iterations of Algorithm YS as a matrix employed in [37] for Montgomery modular inversion case.

The operations required by contiguous k iterations of Algorithm YS can be represented

as

$$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := H(x) \times \begin{bmatrix} R(x) \\ S(x) \end{bmatrix} ; \tag{3.1}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := H(x) \times \begin{bmatrix} U(x) \\ V(x) \end{bmatrix} ; , \tag{3.2}$$

where $H(x)$ is the (2×2)-matrix computed from coefficients from $(m-k+1)$-th to m-th coefficients of $R(x)$ and $S(x)$, and the elements in $H(x)$ are polynomials on $\mathrm{GF}(2)$ with a degree k or less.

Next, we explain how to compute the matrix $H(x)$ by using an example, where $m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$, $B(x) = x^6 + x^4$, and $k = 3$. In this case, the initial values of variables are $R(x) = x^7 + x^5$, $S(x) = x^7 + x^6 + x^3 + x + 1$, $U(x) = x$, $V(x) = 0$, and $\delta = -1$. In the first iteration of Algorithm YS, the operations,

$$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} x \times (R(x) - S(x)) \\ R(x) \end{bmatrix} ; \tag{3.3}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} x \times (U(x) - V(x)) \\ U(x) \end{bmatrix} ; , \tag{3.4}$$

are performed. These operations can be represented in matrices as

$$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} R(x) \\ S(x) \end{bmatrix} ; \tag{3.5}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} U(x) \\ V(x) \end{bmatrix} ; . \tag{3.6}$$

By the above operations, we update the variables as $R(x) = x^7 + x^6 + x^4 + x^2 + x$, $S(x) = x^7 + x^5$, $U(x) = x$, $V(x) = 1$, and $\delta = 0$. Similarly, in the next iteration of Algorithm YS, the operations,

$$R(x) := x \times (R(x) - S(x)); \tag{3.7}$$

$$U(x) := x \times (U(x) - V(x)); , \tag{3.8}$$

are performed. These operations can be represented in matrices as

$$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} R(x) \\ S(x) \end{bmatrix}; \tag{3.9}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} U(x) \\ V(x) \end{bmatrix}; . \tag{3.10}$$

By the above operations, we update the variables as $R(x) = x^7 + x^6 + x^5 + x^3 + x^2$, $S(x) = x^7 + x^5$, $U(x) = x^2 + x$, $V(x) = 1$, and $\delta = -1$. Then, in the next iteration of Algorithm YS, the operations

$$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} x \times (R(x) - S(x)) \\ R(x) \end{bmatrix}; \tag{3.11}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} x \times (U(x) - V(x)) \\ U(x) \end{bmatrix}; , \tag{3.12}$$

are performed. These operations can be represented in matrices as

$$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} R(x) \\ S(x) \end{bmatrix}; \tag{3.13}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} U(x) \\ V(x) \end{bmatrix}; . \tag{3.14}$$

The operations required by the above three iterations of Algorithm YS can be performed at once by employing the matrix,

$$\begin{aligned} H(x) &= \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} x^3 + x^2 + x & x^3 \\ x^2 + x & x^2 \end{bmatrix}. \end{aligned} \tag{3.15}$$

For the word-level description of the proposed algorithm, we partition the polynomials, $R(x)$, $S(x)$, $U(x)$, and $V(x)$, into polynomials with a degree $(w-1)$ on GF(2), where w is the word size of the processor. We can represent $R(x)$ by polynomials with a degree $(w-1)$ as

$$\begin{aligned} R(x) &= R_{M-1}(x)x^{(M-1)w} + \cdots + R_1(x)x^w + R_0(x) \\ R_i(x) &= r_{iw+w-1}x^{w-1} + \cdots + r_{iw+1}x + r_{iw}, \end{aligned} \tag{3.16}$$

where $M = \lceil (m+1)/w \rceil$ and $r_j = 0$ for $j \geq m$ because the degree of the polynomial $R(x)$ is up to m. Similarly, we can represent $S(x)$ by M polynomials with a degree $(w-1)$ and the two polynomials,$U(x)$ and $V(x)$, by at most $2M$ polynomials with a degree $(w-1)$. The matrix $H(x)$ whose elements are polynomials on GF(2) with a degree less than w can

be computed from $R_{M-1}(x)$ and $S_{M-1}(x)$ by only single-word operations.

Equations (3.1) and (3.2) include (multi-word $\times$ single-word)-multiplication. This multiplication is computed with single-word multiplication as

$$\begin{aligned} & R(x) \times h(x) \\ &= (R_{M-1}(x)x^{(M-1)w} + \ldots + R_1(x)x^w + R_0(x)) \times h(x) \\ &= R_{M-1}(x) \times h(x)x^{(M-1)w} + R_{M-2}(x) \times h(x)x^{(M-2)w} \\ &\qquad + \ldots + R_1(x) \times h(x)x^w + R_0(x) \times h(x). \end{aligned} \tag{3.17}$$

Thus, this multiplication is computed with M `MULGF2` instructions and $(M-1)$ `XOR` instructions.

The proposed fast inversion algorithm in $\mathrm{GF}(2^m)$ suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$ is as follows.

[Algorithm MIA]

(*The Proposed Algorithm*)

1: $M := \lceil (m+1)/w \rceil$;
2: $deg\_r := \deg(B(x))$; $deg\_s := m$;
3: $S(x) := G(x) \times x^{Mw-m-1}$; $V(x) := 0$;
4: $R(x) := B(x) \times x^{Mw-deg\_r-1}$; $U(x) := x^{Mw-deg\_r}$;
5: **while** $deg\_r > 0$ **do**
6: $\quad C(x) := R_{M-1}(x)$;
7: $\quad D(x) := S_{M-1}(x)$;
8: $\quad$ **if** $C(x) = 0$ **then**
9: $\quad\quad R(x) := x^w \times R(x)$;
10: $\quad\quad U(x) := x^w \times U(x)$;

11: $deg\_r := deg\_r - w;$

12: **else**

13: $H(x) := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix};$

14: $j := 1;$

15: **while** $j < w$ **and** $deg\_r > 0$ **do**

16: $j := j + 1;$

17: **if** $c_{w-1} = 0$ **then**

18: $C(x) := x \times C(x);$

19: $H(x) := \begin{bmatrix} x & 0 \\ 0 & 1 \end{bmatrix} \times H(x);$

20: **else**

21: **if** $deg\_r < deg\_s$ **then**

22: $\begin{bmatrix} deg\_r \\ deg\_s \end{bmatrix} := \begin{bmatrix} deg\_s \\ deg\_r \end{bmatrix};$

23: $\begin{bmatrix} C(x) \\ D(x) \end{bmatrix} := \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} C(x) \\ D(x) \end{bmatrix};$

24: $H(x) := \begin{bmatrix} x & x \\ 1 & 0 \end{bmatrix} \times H(x);$

25: **else**

26: $C(x) := x \times (D(x) - C(x));$

27: $H(x) := \begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix} \times H(x);$

28: **end if**

29: $deg\_r := deg\_r - 1;$

30: **end if**

31: **end while**

32: $$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := H(x) \times \begin{bmatrix} R(x) \\ S(x) \end{bmatrix};$$

33: $$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := H(x) \times \begin{bmatrix} U(x) \\ V(x) \end{bmatrix};$$

34: **end if**

35: **end while**

36: *output* $\begin{cases} V(x)/x^{Mw} & \textit{if } deg\_s = 0 \\ U(x)/x^{Mw} & \textit{otherwise} \end{cases}$ *as the result.*

□

Note that, we initialize $U(x)$ to $x^{Mw-\deg(B(x))}$ instead of $1(= x^0)$. This modification leads the result to be $B^{-1}(x) \times x^{Mw}$. Thus, we can avoid a multi-word shift at the end of the algorithm. $U(x)$ and $V(x)$ are $(M+1)$ or less words because all u_j and v_j are 0 for j's outside the range $[\max(0, k-m), k]$ as mentioned before. Figure 3.2 shows an example of inversion by Algorithm MIA, where $m = 7$, $w = 4$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $B(x) = x^6 + x^4$.

3.4 EVALUATION OF THE PROPOSED ALGORITHM

We have evaluated the proposed algorithm by comparing the number of `MULGF2` and `XOR` instructions of the algorithm with that of Algorithm EEIA. We assume that `MULGF2` instruction has the same clock cycle latency as `XOR` instruction. We count the number of `MULGF2` and `XOR` instructions required by the following operations in Algorithm EEIA

$R(x)$	$S(x)$	$U(x)$	$V(x)$	$C(x)$	$D(x)$	deg_r	deg_s	$H(x)$
x^7+x^5	$x^7+x^6+x^3+x+1$	x^2	0	x^3+x	x^3+x^2	6	7	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$
				x^3+x^2	x^3+x	6	6	$\begin{bmatrix}x & x\\ 1 & 0\end{bmatrix}$
				x^3+x^2	x^3+x	5	6	$\begin{bmatrix}x^2+x & x^2\\ 1 & 0\end{bmatrix}$
				x^3+x^2	x^3+x^2	5	5	$\begin{bmatrix}x^3+x^2+x & x^3\\ x^2+x & x^2\end{bmatrix}$
$x^7+x^4+x^3$	$x^7+x^6+x^5+x^3+x^2$	$x^5+x^4+x^3$	x^4+x^3	x^3+1	x^3+x^2+x	5	5	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$
				x^3+x^2+x	x^3+x^2+x	4	5	$\begin{bmatrix}x & x\\ 0 & 1\end{bmatrix}$
				0	x^3+x^2+x	4	4	$\begin{bmatrix}x^2 & x^2+x\\ x & x\end{bmatrix}$
				0	x^3+x^2+x	3	4	$\begin{bmatrix}x^3 & x^3+x^2\\ x & x\end{bmatrix}$
x^4	$x^7+x^6+x^5+x^3$	$x^8+x^6+x^5$	x^6	1	x^3+x^2+x	3	4	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$
				x	x^3+x^2+x	2	4	$\begin{bmatrix}x & 0\\ 0 & 1\end{bmatrix}$
				x^2	x^3+x^2+x	1	4	$\begin{bmatrix}x^2 & 0\\ 0 & 1\end{bmatrix}$
				x^3	x^3+x^2+x	0	4	$\begin{bmatrix}x^3 & 0\\ 0 & 1\end{bmatrix}$
		$\underline{x^{11}+x^9+x^8}$	$= (x^3+x+1)\times x^8 = B^{-1}(x)\times x^{Mw}$					

Figure 3.2: Example of Inversion by Algorithm MIA ($m = 7$, $w = 4$, $G(x) = x^7+x^6+x^3+x+1$, $B(x) = x^6+x^4$)

because they need (multi-word $\times$ single-word)-multiplication.

$$R(x) := R(x) - x^j \times S(x); \tag{3.18}$$

$$U(x) := U(x) - x^j \times V(x); \tag{3.19}$$

Similarly, we count the number of `MULGF2` and `XOR` instructions in Eq. (3.1) and (3.2), in Algorithm MIA. In addition, we also count the number of `MULGF2` and `XOR` instructions in operations for calculating $H(x)$ which is in the form

$$H(x) := \begin{bmatrix} h_{00}(x) & h_{01}(x) \\ h_{10}(x) & h_{11}(x) \end{bmatrix} \times H(x);. \tag{3.20}$$

Table 3.1 shows the number of `MULGF2` and `XOR` instructions of the above operations. The figures are the average of inversion of $1,000$ random elements, and we employed NIST-recommended irreducible polynomials [38]. When $w = 32$ and $w = 16$, Algorithm MIA can compute inversion faster than Algorithm EEIA in almost all m on the average. Especially, when m and w are 571 and 32, respectively, the proposed algorithm performs inversion with approximately half the number of instructions required by Algorithm EEIA on the average. Table 3.1 also shows no advantage when either the number of words or the word size is small. In these cases, the cost of the computation for the matrix $H(x)$ is probably bigger than the advantage from performing the operations at once through the matrix.

3.5 DISCUSSION

In the proposed algorithm, we can compute the matrix $H(x)$ whose elements are polynomials on $\mathrm{GF}(2)$ with a degree w or less from the most significant words of $R(x)$ and $S(x)$.

Table 3.1: Comparison of Algorithm EEIA and Algorithm MIA with `MULGF2` instruction

m	w	M	Algorithm EEIA			Algorithm 4			$\frac{\text{Algorithm MIA}}{\text{Algorithm EEIA}}$ [%]
			#MULGF2	#XOR	Total	#MULGF2	#XOR	Total	
163	8	21	2551.990	5751.183	8303.173	4407.820	7424.949	11832.769	142.51
	16	11	1341.678	2900.299	4241.977	1713.488	2339.617	4053.105	95.55
	32	6	733.744	1475.552	2209.296	965.282	903.594	1868.876	84.59
	64	3	437.356	781.916	1219.272	746.164	490.992	1237.156	101.47
233	8	30	5171.630	11782.952	16954.582	8486.847	14853.280	23340.127	137.66
	16	15	2686.084	5938.826	8624.910	2929.674	4305.907	7235.581	83.89
	32	8	1433.177	3005.143	4438.320	1481.997	1496.978	2978.975	67.12
	64	4	811.507	1550.121	2361.628	1093.937	756.188	1850.125	78.34
283	8	36	7606.275	17408.760	25015.035	12091.379	21511.568	33602.947	134.33
	16	18	3929.496	8770.623	12700.119	4028.232	6163.972	10192.204	80.25
	32	9	2069.663	4420.616	6490.279	1893.504	2000.347	3893.851	60.00
	64	5	1150.530	2265.673	3416.203	1351.924	966.546	2318.470	67.87
409	8	52	15796.999	36401.858	52198.857	24322.554	44405.148	68727.702	131.67
	16	26	8090.381	18302.942	26393.323	7566.274	12382.162	19948.436	75.58
	32	13	4196.562	9195.477	13392.039	3163.804	3744.276	6908.080	51.58
	64	7	2262.231	4673.211	6935.442	2059.998	1606.902	3666.900	52.87
571	8	72	30726.338	71092.647	101818.985	45863.413	85143.720	131007.133	128.67
	16	36	15624.255	35652.145	51276.400	13541.819	23238.516	36780.335	71.73
	32	18	8019.095	17888.957	25908.052	5125.872	6648.122	11773.994	45.46
	64	9	4220.452	9012.505	13232.957	3045.612	2574.853	5620.465	42.47

Therefore, the algorithm can compute inversion faster by modifying a polynomial multiply instruction on $\mathrm{GF}(2)$ to compute $w \times (w+1)$-bit multiplication.

We can reduce the computation time of the matrix $H(x)$ by employing a look-up table. The table has $w \cdot 2^{2(w-1)}$ entries and each entry is $(4w + \lceil \log_2(w+1) \rceil)$-bit. For large w, we can compute the matrix $H(x)$ from a smaller table instead of direct table look-up. For example, for $m = 7$, $w = 8$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $B(x) = x^4 + x^2 + x$, we can look up the matrix $H_1(x)$ from a smaller table as

$$H_1(x) = \begin{bmatrix} x^3 + x^2 & x^2 \\ x & 0 \end{bmatrix}, \tag{3.21}$$

according to coefficients from a degree 4 to 7 of $C(x)$ and $D(x)$. Then, we can update $C(x)$ and $D(x)$ by employing $H_1(x)$, and look up the matrix $H_2(x)$ as

$$H_2(x) = \begin{bmatrix} x^2 + x & x^2 \\ x^3 + x^2 + x & x^3 \end{bmatrix}, \tag{3.22}$$

from the same table. Finally, we can compute $H(x)$ by $H_1(x)$ and $H_2(x)$ as

$$\begin{aligned} H(x) &= H_2(x) \times H_1(x) \\ &= \begin{bmatrix} x^5 & x^4 + x^3 \\ x^6 + x^4 + x^3 & x^5 + x^4 + x^3 \end{bmatrix}. \end{aligned} \tag{3.23}$$

This operation can be performed with only single-word instructions.

3.6 SUMMARY OF THE CHAPTER

We have proposed a fast inversion algorithm in $\mathrm{GF}(2^m)$ suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$. In the proposed algorithm, the operations required by several contiguous iterations of the VLSI algorithm proposed by Yan and Sarwate are represented as a matrix. The operations are performed at once through the matrix. When the word size of the processor is 32 and m is 571, the algorithm performs inversion with approximately half the number of polynomial multiply instructions on $\mathrm{GF}(2)$ and `XOR` instructions required by the conventional algorithm for software implementation on the average. We can also accelerate the proposed algorithm by employing a look-up table.

CHAPTER 4

FAST HARDWARE DIVISION ALGORITHM IN $\mathrm{GF}(2^m)$ WITH PARALLELIZATION OF MODULAR REDUCTIONS

4.1 INTRODUCTION

Galois field $\mathrm{GF}(2^m)$ has many applications, especially in elliptic curve cryptography (ECC). In order to accelerate such applications, high-speed implementation of arithmetic operations in $\mathrm{GF}(2^m)$ is required. Among basic arithmetic operations in $\mathrm{GF}(2^m)$, inversion/division takes the maximum time. In this chapter, we propose a fast hardware algorithm for division in $\mathrm{GF}(2^m)$ with parallelization of modular reductions, which requires only one iteration to perform the operations that require two iterations in previously reported division algorithms based on the extended Euclid's algorithm.

In general, one of the following three methods is employed for division in Galois field: the Fermat's little theorem [7,31], the extended Euclid's algorithm [23–30,39], or a solution of a system of linear equations [20, 40]. 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 [20, 25]. The algorithm to be proposed in this chapter is also based on the extended Euclid's algorithm, and accelerated with parallelization of the modular reductions, although effectiveness of parallelization of the modular reductions for multiplication has been showed in [18, 19].

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. In contrast with 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 the circuits proposed in [23,30] 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 than that of the circuit reported in [30], and smaller by approximately the delay of a $2{:}1$ multiplexer than that of the circuit reported in [23] because of its parallelism.

This chapter is organized as follows. In the next section, we explain previously reported hardware division algorithms. In Sect. 4.3, we propose a fast hardware division algorithm with parallelization of modular reductions. In Sect. 4.4, we design a division circuit based on the proposed algorithm. In Sect. 4.5, we estimate and evaluate the circuit by comparing complexity of the circuit and the area and the delay obtained with logic synthesis of it with those of previously reported circuits. In Sect. 4.6, we make several discussions. In Sect. 4.7, we summarize this chapter.

4.2 PRELIMINARIES

4.2.1 BRUNNER ET AL.'S HARDWARE DIVISION ALGORITHM

Here, we describe the division algorithm proposed by Brunner et al. [23], 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

$$f \wedge P(x) = \begin{cases} P(x) & (f = 1) \\ 0 & (f = 0) \end{cases}. \tag{4.1}$$

[Algorithm BCH]

(*Brunner et al.'s Division Algorithm [23]*)

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.*

□

4.2.2 GUO AND WANG'S HARDWARE DIVISION ALGORITHM

Here, we describe the division algorithm proposed by Guo and Wang [25] for description of the proposed algorithm. 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.*

□

4.3 Fast Hardware Division Algorithm with Parallelization of Modular Reductions

In this section, we propose a fast hardware algorithm for division in $\mathrm{GF}(2^m)$. First, we modify Algorithm GW to reduce the number of shift registers when implementing the algorithm as a sequential circuit. This modification is performed by allowing δ to be negative. This technique is described in [24, 27], The modified algorithm is as follows, where the notation $SEL(flag, S_1(x), S_2(x))$ denotes

$$SEL(flag, S_1(x), S_2(x)) = \begin{cases} S_1(x) & \textit{if } flag = 1 \\ S_2(x) & \textit{otherwise} \end{cases}, \tag{4.2}$$

and $SGN(a)$ denotes

$$SGN(a) = \begin{cases} 1 & \textit{if } a < 0 \\ 0 & \textit{otherwise} \end{cases}. \tag{4.3}$$

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: $\quad swap := SGN(\delta) \wedge r_m;$

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: $\quad \delta := (-1)^{swap}\delta - 1;$

9: **end for**

10: **for** $i = 1$ **to** m **do**

11: $\quad V(x) := V(x) \div x;$

12: **end for**

13: *output* $V(x)$ *as the result.*

□

Figure 4.1 shows an example of division by Algorithm MGW, where $m = 4$, $A(x) = x^2 + x$, $B(x) = x^3 + x$, and $G(x) = x^4 + x + 1$.

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

i	$R(x)$	$S(x)$	$U(x)$	$V(x)$	δ	
	x^3+x	x^4+x+1	x^2+x	0	0	1st for-loop
1	x^4+x^2	x^4+x+1	x^3+x^2	0	-1	
2	x^3+x^2+x	x^4+x^2	x^3+x+1	x^3+x^2	0	
3	$x^4+x^3+x^2$	x^4+x^2	x^2+1	x^3+x^2	-1	
4	x^4	$x^4+x^3+x^2$	1	x^2+1	0	
5	x^4+x^3	$x^4+x^3+x^2$	x^3	x^2+1	-1	
6	x^3	x^4+x^3	x^3+1	x^3	0	
7	x^4	x^4+x^3	1	x^3	-1	
8	x^4	x^4	1	1	0	
1				x^3+1		2nd for-loop
2				x^3+x^2+1		
3				x^3+x^2+x+1		
4				$\underline{x^3+x^2+x}$		

Figure 4.1: Example of Division by Algorithm MGW ($m = 4, A(x) = x^2 + x, B(x) = x^3 + x, G(x) = x^4 + x + 1$)

by replacing m arbitrary chosen operations out of the $2m$ operations in line 7 with

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \\ SEL(swap, U(x), V(x)) \div x \end{bmatrix}; . \tag{4.4}$$

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

$$\begin{bmatrix} U'(x) \\ V'(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \cdot x \\ SEL(swap, U(x), V(x)) \end{bmatrix}; \tag{4.5}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) - {r'}_m \wedge V'(x)) \\ SEL(swap', U'(x), V'(x)) \div x \end{bmatrix};. \tag{4.6}$$

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

$$\begin{cases} r'_m := r_{m-1} \oplus (r_m \wedge s_{m-1}) \\ swap' := SGN\left((-1)^{swap}\delta - 1\right) \wedge r'_m. \end{cases} \tag{4.7}$$

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

$$\begin{bmatrix} U'(x) \\ V'(x) \end{bmatrix} := \begin{bmatrix} (U(x) - r_m \wedge V(x)) \times x \\ SEL(swap, U(x), V(x)) \end{bmatrix}; \tag{4.8}$$

$$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} (U'(x) - {r'}_m \wedge V'(x)) \bmod G(x) \\ SEL(swap', U'(x), V'(x)) \div x \end{bmatrix};. \tag{4.9}$$

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

above expressions are performed as

$$\begin{cases} u'_j := u_{j-1} \oplus (r_m \wedge v_{j-1}) ; \\ v'_j := SEL\,(swap, u_j, v_j) \\ u_j := u'_j \oplus (u'_m \wedge g_j) \oplus \left(r'_m \wedge v'_j\right) ; \\ v_j := SEL\left(swap', u'_{j+1}, v'_{i+1} \oplus (v'_0 \wedge g_{i+1})\right) \end{cases} \tag{4.10}$$

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: $$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} (R'(x) - {r'}_m \wedge S'(x)) \times x \\ SEL\,(swap', R'(x), S'(x)) \end{bmatrix};$$

i	$R(x)$	$S(x)$	$U(x)$	$V(x)$	δ
	x^3+x	x^4+x+1	x^2+x	0	0
1	x^3+x^2+x	x^4+x^2	x^3+x^2	x^2+x	0
2	x^4	$x^4+x^3+x^2$	x^3+x^2+1	x^3+x^2	0
3	x^3	x^4+x^3	x^3+x^2+x	1	0
4	x^4	x^4	x^3+x^2+x	$\underline{x^3+x^2+x}$	0

i	$R'(x)$	$S'(x)$	$U'(x)$	$V'(x)$	δ'
1	x^4+x^2	x^4+x+1	x^3+x^2	0	-1
2	$x^4+x^3+x^2$	x^4+x^2	x^4+x^3	x^2+x	-1
3	x^4+x^3	$x^4+x^3+x^2$	x	x^3+x^2	-1
4	x^4	x^4+x^3	$x^4+x^3+x^2$	1	-1

Figure 4.2: Example of Division by Algorithm DEEA ($m = 4, A(x) = x^2 + x, B(x) = x^3 + x, G(x) = x^4 + x + 1$)

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

4.4 Design of a Circuit Based on the Proposed Algorithm

We have designed a sequential circuit that performs the operations in one iteration of the proposed algorithm in a cycle. Figure 4.3 shows a block diagram of the circuit. Figures

4.4–4.6 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 4.7 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

$$\begin{cases} r'_j := r_{j-1} \oplus (r_m \wedge s_{j-1})\,; \\ s'_j := SEL\,(swap, r_j, s_j)\,; \\ r_j := r'_{j-1} \oplus \left(r'_m \wedge s'_{j-1}\right); \\ s_j := SEL\left(swap', r'_j, s'_j\right); \end{cases} \tag{4.11}$$

and consists of $(m+1)$ ***RS-cell***s. ***UV-calc*** is the part that updates the polynomials $U(x)$ and $V(x)$ according to expressions (4.10) and consists of m ***UV-cell***s 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***-cell***s.

The control signals of the circuit, $swap$, $swap'$, $shift1$, and $shift2$, are computed as

$$\begin{cases} swap := sgn \wedge r_m \\ swap' := sgn' \wedge r'_m \\ shift1 := (\delta_m \wedge \overline{sgn}) \vee (sgn \wedge \overline{r_m}) \\ shift2 := \left(\delta'_m \wedge \overline{sgn'}\right) \vee \left(sgn' \wedge \overline{r_m}'\right). \end{cases} \tag{4.12}$$

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 $shift2$.

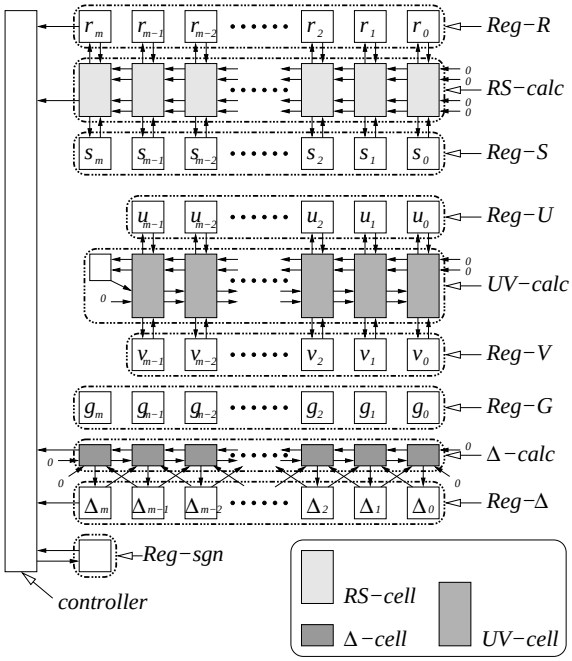

Figure 4.3: Block Diagram of the Circuit Based on Algorithm DEEA

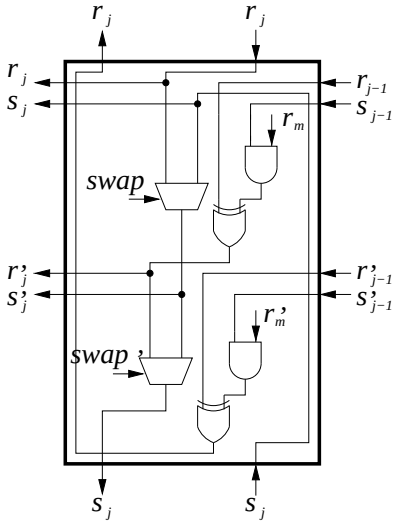

Figure 4.4: *RS-cell*

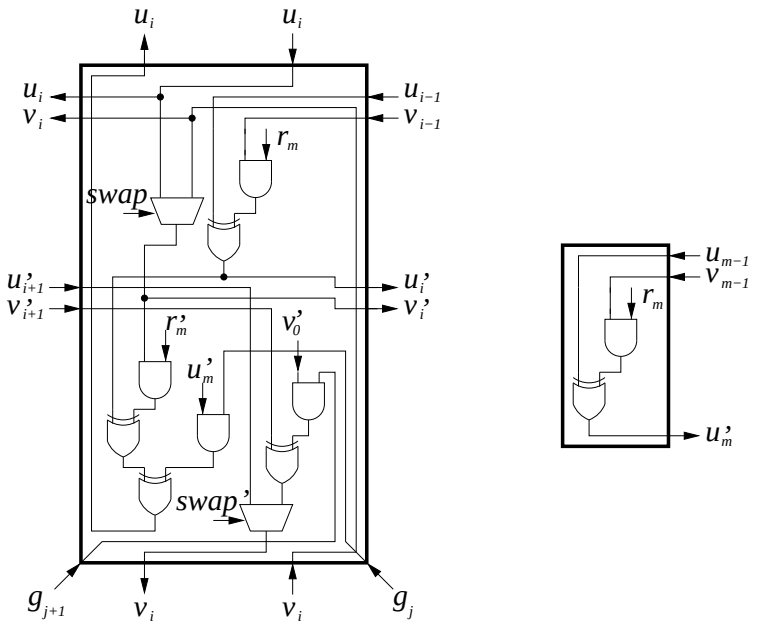

Figure 4.5: (a) *UV-cell* and (b) *UV-cell2*

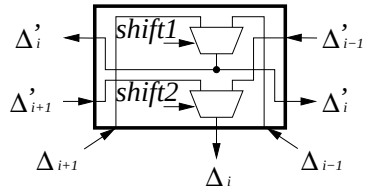

Figure 4.6: Δ-*cell*

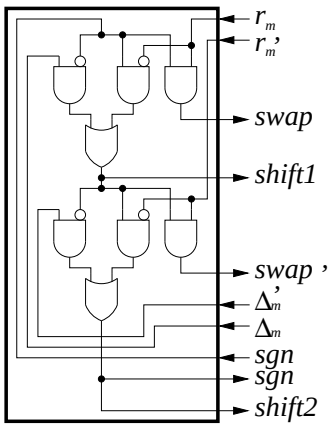

Figure 4.7: Controller

4.5 ESTIMATION AND EVALUATION OF THE CIRCUIT

We compare the circuit based on the proposed algorithm with two previously proposed circuits designed as sequential circuits. One is the circuit proposed by Brunner et al. described in Sect. 4.2.1. The other is proposed by Kim et al. [30] that has the same critical path delay as Guo and Wang's one, which is described in Sect. 4.2.2, with latency of $2m - 1$ cycles. Table 4.1 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 than that of the circuit reported in [30]. The circuit based on the proposed algorithm has m clock cycle latency, which is almost half of that of the previously proposed circuits.

We described the two circuits with Verilog-HDL for several m's, and synthesized them with Synopsys Design Compiler using Rohm 0.18 μm 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 [30], Table 4.2 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 [30]. The computation time of the proposed circuit is over 35% shorter than that of the circuit proposed in [30].

4.6 DISCUSSION

The circuit designed in the previous section employs 1-hot counter to store the value of δ because if we employ binary counter for it the counter can be the critical path. In order to reduce the area of the circuit, we can employ a two-level 1-hot counter [24] for storing the value of $|\delta|$. 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 reduce the register size and the number of 2:1 multiplexer

Table 4.1: Comparison of Circuits

	Brunner et al. [23]	Kim et al. [30]	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 4.2: Synthesis Results

m	Circuit	Area [mm^2]	Critical Path Delay [ns]	# of cycle	Comp. time [ns]
163	Kim et al. [30]	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

significantly. Since the critical path delay of the circuit is a little larger than that of Kim et al.'s one, we can apply such a modification to the circuit easier without a high cost about critical path delay.

4.7 SUMMARY OF THE CHAPTER

We have proposed a fast hardware algorithm for division in $\mathrm{GF}(2^m)$ in this chapter. This algorithm 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 the operations.

We have designed a circuit based on the proposed algorithm. The 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 [30].

CHAPTER 5

HARDWARE ALGORITHM FOR A COMBINED CIRCUIT OF MULTIPLICATION AND INVERSION IN $\mathrm{GF}(2^m)$

5.1 INTRODUCTION

Multiplication and inversion in $\mathrm{GF}(2^m)$ are employed for elliptic curve cryptography (ECC) which is one of the major public key cryptosystems. Since m is very large in ECC [1, 38], realization of circuits for both operations yields large area. Thus, the reduction of hardware of these circuits is important for area-restricted devices like portable ones.

In this chapter, we propose a combined circuit for multiplication and inversion in $\mathrm{GF}(2^m)$. In the circuit, multiplication and inversion are carried out with the MSB-first multiplication algorithm and the inversion algorithm proposed in [29] which is based on the extended Euclid's algorithm, respectively. In order to develop a combined circuit based

on these algorithms, we start with combining them by focusing on the similarities between them. Since almost all hardware components of the circuit are shared by multiplication and inversion, the circuit can be implemented with significantly smaller hardware than that necessary to implement both multiplication and inversion separately.

Compared with previously proposed combined circuits for multiplication and division, the circuit to be proposed has several advantages. Although the area complexity of [20] is $O(m^2)$, that of the circuit to be proposed is $O(m)$. The circuits proposed in [21, 22] are based on the Stein's binary GCD algorithm for division and need to reverse the order of the coefficients of inputs and output polynomials for multiplication. Thus, the circuit proposed in [22] has extra area for such pre- and post-computation, and [21] does not describe how to implement such computation. In contrast, the circuit proposed here does not need such computations.

We have synthesized the circuit to be proposed using 0.18 μm CMOS standard cell library, for several m's, and estimated its area and critical path delay. The area of the circuit and registers is significantly smaller than that of the previously proposed combined multiplication/division circuits [21, 22].

This chapter is organized as follows. In the next section, we explain a previously reported hardware multiplication algorithm. In Sect. 5.3, we propose a hardware algorithm for a combined circuit of multiplication and inversion. In Sect. 5.4, we design a combined circuit based on the proposed algorithm. In Sect. 5.5, we estimate and evaluate the circuit by comparing complexity of the circuit and the area and the delay obtained with logic synthesis of it with those of previously reported combined circuits. In Sect. 5.6, we make several discussions. In Sect. 5.7, we summarize this chapter.

5.2 PREVIOUSLY REPORTED HARDWARE MULTIPLICATION ALGORITHM

We combine the MSB-first multiplication algorithm with the inversion algorithm shown later, because they are suited to be combined. The MSB-first multiplication algorithm in $\mathrm{GF}(2^m)$ is as follows, where b_j denotes the j-th coefficient of the polynomial $B(x)$. The operation "$\times$" denotes polynomial multiplication on $\mathrm{GF}(2)$. The notation $\deg(\cdot)$ denotes the degree of a polynomial.

[Algorithm MSB]

(*The MSB-first Multiplication Algorithm in* $\mathrm{GF}(2^m)$)

Input: $A(x), B(x)$: $\deg(A(x)), \deg(B(x)) \leq m-1$

$G(x)$: $\deg(G(x)) = m$ and irreducible

Output: $A(x) \cdot B(x) (= A(x) \times B(x) \bmod G(x))$

1: $P(x) := 0$;
2: **for** $i := 1$ **to** m **do**
3: **if** $b_{m-1} = 0$ **then**
4: $P(x) := x \times P(x) \bmod G(x)$;
5: **else**
6: $P(x) := (x \times P(x) + A(x)) \bmod G(x)$;
7: **end if**
8: $B(x) := x \times B(x)$;
9: **end for**
10: *output* $P(x)$ *as the result*;

□

Table 5.1: Operations of Algorithm MSB in an iteration

b_{m-1}	$B(x)$	$P(x)$
0	$B(x) := x \times B(x);$	$P(x) := x \times P(x) \bmod G(x);$
1	$B(x) := x \times B(x);$	$P(x) := (x \times P(x) + A(x)) \bmod G(x);$ $\left(P(x) := x \times (P(x) + x^{-1} \times A(x)) \bmod G(x);\right)$

Table 5.2: Operations of Algorithm YS in an iteration

r_m	$\delta \geq 0$	$R(x), S(x)$
0	*don't care*	$R(x) := x \times R(x);$
1	*false*	$\begin{bmatrix} R(x) \\ S(x) \end{bmatrix} := \begin{bmatrix} x \times (R(x) + S(x)) \\ R(x) \end{bmatrix};$
	true	$R(x) := x \times (R(x) + S(x));$

r_m	$\delta \geq 0$	$U(x), V(x)$
0	*don't care*	$U(x) := x \times U(x) \bmod (x^{m+1} + 1);$
1	*false*	$\begin{bmatrix} U(x) \\ V(x) \end{bmatrix} := \begin{bmatrix} x \times (U(x) + V(x)) \bmod (x^{m+1} + 1) \\ U(x) \end{bmatrix};$
	true	$U(x) := x \times (U(x) + V(x)) \bmod (x^{m+1} + 1);$

Note that, in the execution of the algorithm, although the degree of $B(x)$ exceeds $(m-1)$, the variables b_k's are not referred to, for $k \geq m$. Therefore, when we implement the algorithm as a circuit, an m-bit register is sufficient to store $B(x)$.

5.3 Hardware Algorithm for a Combined Circuit of Multiplication and Inversion

In order to develop a combined circuit, we start with combining Algorithms MSB and YS by focusing on the similarities between them. Tables 5.1 and 5.2 show the operations in an iteration of Algorithms MSB and YS, respectively.

The polynomial $B(x)$ in Algorithm MSB and the polynomial $R(x)$ in Algorithm YS determine which operation is carried out in an iteration. Therefore, we consider merging

the two polynomials first. We represent the obtained polynomial as $BR(x)$. With Tables 5.1 and 5.2, we also merge the polynomials $P(x)$ and $x^{-1} \times A(x)$ in Algorithm MSB with the polynomials $U(x)$ and $V(x)$ in Algorithm YS, respectively. We represent the obtained polynomials as $PU(x)$ and $AV(x)$, respectively. Note that, for multiplication, we must modify the algorithm so that it satisfies $\delta \geq 0$ at all times.

Next, we consider merging the reduction polynomials of the two algorithms. The reduction polynomial of Algorithm MSB is the irreducible polynomial with degree m that defines the field. On the other hand, the reduction polynomial of Algorithm YS implemented as a circuit is the polynomial $x^{m+1} + 1$ whose degree is $m + 1$, because we employ cyclic left shift instead of logical left shift in an $(m + 1)$-bit register as described earlier. Since it is desirable for VLSI implementation that the degrees of the two reduction polynomials are identical, we employ $x \times G(x)$ instead of $G(x)$ as the reduction polynomial in Algorithm MSB. Along with this modification, we also modify the other polynomials, $PU(x)$, $BR(x)$, and $AV(x)$, as multiplied by x. This modification is simply performed by changing the initial values of them with the values multiplied by x.

Finally, we apply two modifications for efficiency. One is that we employ $(m - 1)$ and $x \times B(x) + 1$ as the initial values of δ and $BR(x)$, respectively, for multiplication. By this modification, $\delta \geq 0$ is always true in the execution of the algorithm, and after m iterations, the values of $BR(x)$, $PU(x)$, and δ will be 1, $(A(x) \cdot B(x)) \times x$, and -1, respectively. Therefore, in the $(m + 1)$-th iteration, the result $(A(x) \cdot B(x)) \times x$ will be transferred to the register for $AV(x)$ that stores the result of inversion. At the same time, we employ x^3 as the initial value of $PU(x)$ for inversion so that the result $B^{-1}(x) \times x$ will be stored in the register for $AV(x)$, namely both of the product and the inverse will appear in the same position in the register.

The other is that we also merge the two polynomials, $G(x)$ in Algorithm MSB and $S(x)$ in Algorithm YS, so that they can share register. It is because the polynomial $G(x)$ in

Algorithm MSB is not employed in Algorithm YS, and the polynomial $S(x)$ in Algorithm YS is not employed in Algorithm MSB. We represent the obtained polynomial as $GS(x)$.

A combined algorithm for multiplication and inversion in $\mathrm{GF}(2^m)$ is as follows, where br_j, pu_j, and gs_j denote the j-th coefficient of the polynomials, $BR(x)$, $PU(x)$, and $GS(x)$, respectively. $mode$ is a control signal for selection of multiplication or inversion.

[Algorithm MULINV]

(*A Combined Algorithm for Multiplication and Inversion in* $\mathrm{GF}(2^m)$)

Input: $A(x), B(x)$: $\deg(A(x)), \deg(B(x)) \leq m-1$

$G(x)$: $\deg(G(x)) = m > 3$ and irreducible

$$mode = \begin{cases} 0 & \textit{(inversion mode)} \\ 1 & \textit{(multiplication mode)} \end{cases}$$

Output: $B^{-1}(x)$ (*if* $mode = 0$)

$A(x) \cdot B(x)$ (*if* $mode = 1$)

1: **if** $mode = 0$ **then** /* *inversion* */
2: $BR(x) := x \times B(x);\ GS(x) := G(x);$
3: $PU(x) := x^3;\ AV(x) := 0;$
4: $\delta := -1;\ n := 2m-1;$
5: **else** /* *multiplication* */
6: $BR(x) := x \times B(x) + 1;\ GS(x) := G(x);$
7: $PU(x) := 0;\ AV(x) := A(x);$
8: $\delta := m-1;\ n := m+1;$
9: **end if**
10: **for** $i := 1$ **to** n **do**
11: **if** $br_m = 0$ **then**
12: $BR(x) := BR(x) \times x;$

13: $PU(x) := PU(x) \times x$;

14: **else**

15: **if** $\delta \geq 0$ **then**

16: $BR(x) := (BR(x) + \overline{mode} \times GS(x)) \times x$;

17: $PU(x) := (PU(x) + AV(x)) \times x$;

18: **else**

19: $$\begin{bmatrix} BR(x) \\ GS(x) \end{bmatrix} := \begin{bmatrix} x \times (BR(x) + GS(x)) \\ mode \times GS(x) + \overline{mode} \times BR(x) \end{bmatrix};$$

20: $$\begin{bmatrix} PU(x) \\ AV(x) \end{bmatrix} := \begin{bmatrix} x \times (PU(x) + AV(x)) \\ PU(x) \end{bmatrix};$$

21: $\delta := -\delta$;

22: **end if**

23: **end if**

24: **if** $pu_{m+1} = 1$ **then**

25: $PU(x) := PU(x)$
$+x^{m+1} + \sum_{k=1}^{m-1}(mode \wedge gs_k)x^{k+1} + \overline{mode}$;

26: **end if**

27: $\delta := \delta - 1$;

28: **end for**

29: *output* $AV(x)/x$ *as the result*;

□

Table 5.3: Operations of Algorithm MULINV in an iteration

br_m	$\delta \geq 0$	$BR(x), GS(x)$
0	*don't care*	$BR(x) := x \times BR(x);$
1	*false*	$\begin{bmatrix} BR(x) \\ GS(x) \end{bmatrix} := \begin{bmatrix} x \times (BR(x) + \overline{mode} \times GS(x)) \\ mode \times GS(x) + \overline{mode} \times BR(x) \end{bmatrix};$
	true	$BR(x) := x \times (BR(x) + \overline{mode} \times GS(x));$

br_m	$\delta \geq 0$	$PU(x), AV(x)$
0	*don't care*	$PU(x) := x \times PU(x) \bmod F(x);$
1	*false*	$\begin{bmatrix} PU(x) \\ AV(x) \end{bmatrix} := \begin{bmatrix} x \times (PU(x) + AV(x)) \bmod F(x) \\ PU(x) \end{bmatrix};$
	true	$PU(x) := x \times (PU(x) + AV(x)) \bmod F(x);$

$$F(x) = \begin{cases} x^{m+1} + 1 & mode = 0 \\ GS(x) \times x \,(= G(x) \times x) & mode = 1 \end{cases}$$

5.4 Design of a Circuit Based on the Proposed Algorithm

We have designed a sequential circuit based on Algorithm MULINV, which performs operations of an iteration of the algorithm in a cycle. Table 5.3 shows the operations in an iteration in Algorithm MULINV.

Figure 5.1 shows a block diagram of the proposed circuit. *Calc-1* and *Calc-2* consist of $(m+1)$ *Cell-I*s and $(m+1)$ *Cell-II*s, respectively. *Calc-1* updates $BR(x)$ and $GS(x)$, and *Calc-2* updates $BR(x)$ and $AV(x)$ in accordance with Table 5.3. *Reg-BR*, *Reg-GS*, *Reg-PU*, and *Reg-AV*, are the registers for storing $BR(x)$, $GS(x)$, $PU(x)$, and $AV(x)$, which have $(m+1)$-bit width, respectively. Figures 5.2–5.4 show design examples of basic cells (*Cell-I*, *Cell-II*), and the controller in Fig. 5.1, respectively. The control signals of the

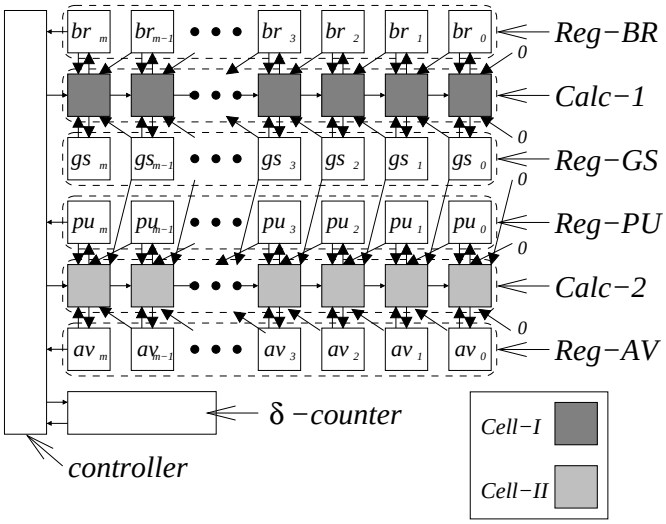

Figure 5.1: Block Diagram of the Proposed Circuit

circuit, $reduce\_A$, $reduce\_B$, $swap\_A$, $swap\_B$, and $reduction$, are computed as follows:

$$\begin{cases} reduce\_A = br_m \wedge \overline{mode} \\ reduce\_B = br_m \\ swap\_B = \begin{cases} 1 & (br_m = 1 \ \ \textit{and} \ \ \delta < 0) \\ 0 & (\textit{otherwise}) \end{cases} \\ swap\_A = swap\_B \wedge \overline{mode} \\ reduction = \begin{cases} (pu_m \oplus (av_m \wedge br_m)) \wedge \overline{mode} & (\textit{for } pu_0 \textit{ and } av_0) \\ (pu_m \oplus (av_m \wedge br_m)) \wedge mode & (\textit{for the others}) \end{cases} \end{cases} \tag{5.1}$$

5.5 ESTIMATION AND EVALUATION OF THE CIRCUIT

Table 5.4 shows the comparison of the combined circuits that have $O(m)$ area complexity. The proposed circuit can be implemented with small area, although it needs more m clock cycles for division than the others circuits. Note that gate count in the table includes only

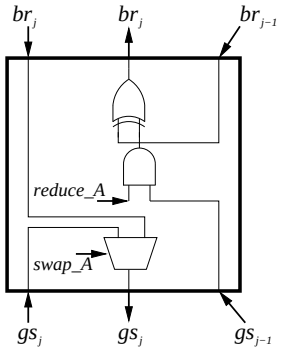

Figure 5.2: Cell-I

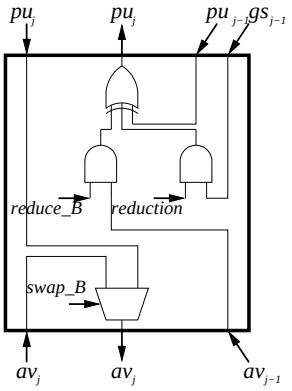

Figure 5.3: Cell-II

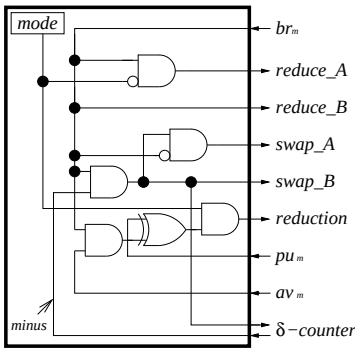

Figure 5.4: Controller of the Proposed Combined Circuit

Table 5.4: Comparison of Combined Circuits

	Kim et al. [21]	Lin et al. [22]	Proposed
# of Cycles per Mul.	m	m	$m+1$
# of Cycles per Inv.	$2m-1$	$2m-1$	$2m-1$
# of Cycles per Div.	$2m-1$	$2m-1$	$(3m)$
Register Width [bit]	$6m+2$	$5m$ $+\lceil\log_2(m+1)\rceil$	$4m+7$ $+\lceil\log_2(m+1)\rceil$
# of Gates for Basic Cells:			
2-input AND	$4m$	$6m+4$	$3m+3$
2-input XOR	$4m$	$3m$	$3m+3$
2:1 MUX	$4m$	$5m$	$2m+2$
Critical Path Delay	$2T_{\mathrm{A}}+2T_{\mathrm{X}}+2T_{\mathrm{M}}$	$2T_{\mathrm{A}}+2T_{\mathrm{X}}+T_{\mathrm{M}}$	$3T_{\mathrm{A}}+2T_{\mathrm{X}}$

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

gates for basic cells because the area of the whole circuit is almost occupied by basic cells and registers when m is large. Also note that we added m MUXs to the circuits proposed in [21, 22] to stop the operations after the circuit finished multiplication. Otherwise, the result of multiplication will change after m clock cycles.

We described circuits based on Algorithm MULINV with Verilog-HDL for several m's. For comparison, we also described circuits based on the MSB-first multiplication and the Yan and Sarwate's algorithm, and the previously proposed combined multiplication/division circuits in [21, 22]. Note that although the circuit in [21] needs to reverse the order of the coefficients of inputs and output polynomials, we did not include the cost of such pre- and post-computation in the estimation because they do not describe how to implement such computation.

We synthesized them with Synopsys Design Compiler using Rohm 0.18 μm CMOS standard cell library provided by VLSI Design and Education Center (VDEC), the University of Tokyo. Table 5.5 shows the synthesis results. We set 0 as area constraint and various values as critical path delay constraints, and synthesized them with Design Com-

Table 5.5: Synthesis Results

m	Circuit	Critical Path Delay [ns]	Calc. Time [ns] Mul.	Inv.	Div.	Area [mm^2]
163	Algorithm MSB+YS	1.034 / 1.172	168.5	308.9	—	0.2132
	Lin et al. [22]	1.282	209.0	416.8	416.8	0.1429
	Kim et al. [21]	1.314	214.2	427.1	427.1	0.1600
	Proposed	1.233	202.2	400.7	(602.7)	0.1238
233	Algorithm MSB+YS	1.099 / 1.189	256.1	552.9	—	0.2980
	Lin et al.	1.318	307.0	612.7	612.7	0.2007
	Kim et al.	1.303	303.6	605.9	605.9	0.2211
	Proposed	1.274	298.1	592.4	(890.4)	0.1710
283	Algorithm MSB+YS	1.046 / 1.197	296.0	676.3	—	0.3653
	Lin et al.	1.324	374.6	747.9	747.9	0.2406
	Kim et al.	1.373	388.6	775.7	775.7	0.2696
	Proposed	1.292	366.9	730.0	(1097)	0.2092
409	Algorithm MSB+YS	1.150 / 1.208	470.4	986.9	—	0.5234
	Lin et al.	1.354	553.8	1106	1106	0.3485
	Kim et al.	1.418	580.0	1159	1159	0.3880
	Proposed	1.326	543.7	1083	(1627)	0.2957
571	Algorithm MSB+YS	1.150 / 1.284	656.7	1464	—	0.7202
	Lin et al.	1.341	765.6	1530	1530	0.4872
	Kim et al.	1.433	818.2	1635	1635	0.5335
	Proposed	1.320	755.0	1506	(2261)	0.4169

piler's synthesis option "`-incremental_mapping`." The figures in the table are the best area-time product ones obtained with the synthesis. The area of the proposed circuit is approximately over 15% smaller than that of the combined multiplication/division circuits proposed in [21, 22].

5.6 DISCUSSION

Although we have considered the case of the general irreducible polynomial $G(x)$, we can simplify the combined circuit by employing the special formed irreducible polynomial as a multiplier. Especially, if $G(x)$ is all one polynomial (AOP), the reduction polynomial will be $(x^{m+1} + 1)$ as in [41]. Therefore, in such case, we can design a smaller combined circuit than a combined circuit with a general irreducible polynomial, because such circuit employes the same reduction polynomials for multiplication and inversion.

5.7 SUMMARY OF THE CHAPTER

We have proposed a combined circuit for multiplication and inversion in $\mathrm{GF}(2^m)$ in this chapter. In order to develop the combined circuit, we have combined the MSB-first multiplication algorithm and the Yan and Sarwate's inversion algorithm by focusing on the similarities between them. Almost all hardware components of a circuit based on the proposed algorithm are shared by multiplication and inversion.

The area of the proposed circuit has been estimated with logic synthesis using Rohm 0.18 μm CMOS standard cell library. The area of the circuit is significantly smaller than that necessary to implement both multiplication and inversion separately and that of the previously proposed combined multiplication/division circuits. Therefore, the proposed circuit is effective in area-restricted devices.

CHAPTER 6

CONCLUSION

In this dissertation, we proposed three methods for hardware-assisted implementation of arithmetic operations in Galois field $\mathrm{GF}(2^m)$.

In Chap. 3, we proposed a fast software algorithm for inversion in $\mathrm{GF}(2^m)$ that is based on the extended Euclid's algorithm and suitable for implementation with a polynomial multiply instruction on $\mathrm{GF}(2)$. This algorithm employes the matrix that represents the operations required by several contiguous iterations of the previously reported algorithm, and computes inversion with a polynomial multiply instruction on $\mathrm{GF}(2)$ through the matrix. When the word size of the processor is 32 and m is 571, the proposed algorithm computes inversion with approximately half the number of polynomial multiply instructions on $\mathrm{GF}(2)$ and `XOR` instructions required by the previously proposed algorithm.

In Chap. 4, we proposed a fast hardware algorithm for division in $\mathrm{GF}(2^m)$ for a fast dedicated circuit. This algorithm performs the operations required by two iterations of the previously reported algorithm in one iteration, and parallelizes the two modular reductions for fast calculation by changing the order of execution of the operations. By the parallelization of the modular reductions, the critical path delay of a circuit based on this algorithm is almost the same as that of the previously reported circuit, nevertheless the number of

clock cycles required by the circuit is almost half of that of the previously reported circuit. By logic synthesis, the computation time of the circuit has been estimated to be over 35% shorter than that of the previously reported circuit.

In Chap. 5, we proposed a hardware algorithm for a combined circuit of multiplication and inversion in $\mathrm{GF}(2^m)$. The proposed algorithm has been developed by focusing on the similarities between the MSB-first multiplication algorithm and the Yan and Sarwate's inversion algorithm. In a combined circuit based on this algorithm, almost all hardware components of a circuit based on the proposed algorithm are shared by multiplication and inversion. By logic synthesis, the area of the combined circuit is estimated to be over 15% smaller than that of the previously reported combined circuits. In addition, unlike the previously reported combined circuits, the combined circuit does not need any pre- and post-computation.

Thorough the study in this dissertation, we could show effectiveness of hardware-assisted implementation of arithmetic operations in $\mathrm{GF}(2^m)$ and obtain the knowledge about the effectiveness of designing software/hardware algorithms by focusing their similarities and parallelism for hardware-assisted implementation. We could show in Chap. 3 that a polynomial multiply instruction on $\mathrm{GF}(2)$ is useful for not only multiplication but also inversion in $\mathrm{GF}(2^m)$. We could also show in Chap. 4 that the method that parallelizes the modular reductions is a efficient technique for speeding up division in $\mathrm{GF}(2^m)$. We could also show in Chap. 5 that a combined circuit designed by focusing on the similarities between multiplication and inversion is useful for area-restricted devices because of this small area. Therefore, we can conclude this dissertation that hardware-assist is a promising technique for efficient implementation of arithmetic operations in $\mathrm{GF}(2^m)$. We could also conclude that hardware-assisted implementation of arithmetic operations in $\mathrm{GF}(2^m)$ can be more efficient by designing software/hardware algorithms carefully about their similarities and parallelism. Knowledge obtained through the study should make hardware-assisted im-

plementation more efficient for arithmetic operations in $\mathrm{GF}(2^m)$ as well as other operations necessary in important applications.

BIBLIOGRAPHY

[1] I. P. W. Group, *IEEE P1363/D13 (Draft Version 13):Standard Specifications for Public Key Cryptography*. IEEE P1363 Working Group., Nov. 1999. [Online]. Available: http://grouper.ieee.org/groups/1363/

[2] M. A. Hasan, "Look-up table-based large finite field multiplication in memory constrained cryptosystems," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 749–758, July 2000.

[3] J. López and R. Dahab, "High-speed software multiplication in $\mathbb{F}_{2^m}$," in *First International Conference in Cryptography in India — INDOCRYPT 2000*, May 2000, pp. 203–212.

[4] D. Hankerson, J. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *Proceeding of Cryptographic Hardware and Embedded Systems – CHES 2000*, Aug.17–18 2000, pp. 1–24.

[5] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast key exchange with elliptic curve systems," in *Advances in Cryptology – Crypto '95*, Aug.29–Sept.1 1995, pp. 43–56.

[6] Ç K Koç and T. Acar, "Montgomery multiplication in $\mathrm{GF}(2^k)$," in *Third Annual Workshop on Selected Areas in Cryptography*, Aug. 1996, pp. 95–106.

[7] 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.

[8] J. Großschädl and E. Saveş, "Instruction set extensions for fast arithmetic in finite fields $\mathrm{GF}(p)$ and $\mathrm{GF}(2^m)$," in *Proceeding of Cryptographic Hardware and Embedded Systems – CHES 2005*, Aug.29–Sept.1 2005, pp. 133–147.

[9] A. M. Fiskiran and R. B. Lee, "Evaluating instruction set extensions for fast arithmetic on binary finite fields," in *Proceeding of International Conference on Application-specific Systems, Architectures and Processors — ASAP 2004*, Sept.27-29 2004, pp. 125–136.

[10] S. Tillich and J. Großschädl, "Accelerating AES using instruction set extensions for elliptic curve cryptography," in *International Conference on Computational Science and Its Applications – ICCSA 2005*, 2005, pp. 665–674.

[11] H. Eberle, A. Wander, N. Gura, and S. Chang-Shantz, "Architectural extensions for elliptic curve cryptography over $\mathrm{GF}(2^m)$," *Sun Labs Open House*, 2004. [Online]. Available: http://research.sun.com/sunlabsday/docs.2004/Micro.pdf

[12] A. M. Fiskiran and R. B. Lee, "Pax: A datapath-scalable minimalist cryptographic processor for mobile devices," in *Embedded Cryptographic Hardware: Design & Security*, N. Nedjah and L. de Macedo Mourlle, Eds. Nova Science Publishers, Inc., 2005, ch. 2, pp. 19–34.

[13] S. Bartolini, I. Branovic, R. Giorgi, and E. Martinelli, "Effects of instruction set extensions on an embedded processor: a case study on elliptic curve cryptography over $\mathrm{GF}(2^m)$," *IEEE Trans. on Comput.*, vol. 57, no. 5, pp. 672–685, May 2008.

[14] C. H. Kim, C. P. Hong, and S. Kwon, "A digit-serial multiplier for finite field $GF(2^m)$," *IEEE Trans. VLSI Systems*, vol. 13, no. 4, pp. 476–483, Apr. 2005.

[15] B. Sunar and Ç. K. Koç, "Mastrovito multiplier for all trinomials," *IEEE Trans. on Comput.*, vol. 48, no. 5, pp. 522–527, May 1999.

[16] J. L. Imaña, J. M. Sánchez, and F. Tirado, "Bit-parallel finite field multipliers for irreducible trinomials," *IEEE Trans. on Comput.*, vol. 55, no. 5, pp. 520–533, May 2006.

[17] S. Kumar, T. Wollinger, and C. Paar, "Optimum digit serial $GF(2^m)$ multipliers for curve-based cryptography," *IEEE Trans. on Comput.*, vol. 55, no. 10, pp. 1306–1311, Oct. 2006.

[18] H. Fan and Y. Dai, "Fast bit-parallel $GF(2^n)$ multiplier for all trinomials," *IEEE Trans. on Comput.*, vol. 54, no. 4, pp. 485–490, 2005.

[19] A. Hariri and A. Reyhani-Masoleh, "Digit-serial structures for the shifted polynomial basis multiplication over binary extension fields," in *International Workshop on the Arithmetic of Finite Fields – WAIFI 08*, July 2008, pp. 103–116.

[20] 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.

[21] C. H. Kim, C. P. Hong, and S. Kwon, "A novel arithmetic unit over $GF(2^m)$ for low cost cryptographic applications." in *HPCC*, ser. Lecture Notes in Computer Science, L. T. Yang, O. F. Rana, B. D. Martino, and J. Dongarra, Eds., vol. 3726. Springer, 2005, pp. 524–534.

[22] W. C. Lin, J. H. Chen, M. D. Shieh, and C. M. Wu, "A combined multiplication/division algorithm for efficient design of ECC over $\mathrm{GF}(2^m)$," in *2007 IEEE Region 10 Conference — TENCON 2007*, Oct.30 –Nov.2 2007, pp. 1–4.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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, Mar. 2005.

[28] 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, Mar. 2004.

[29] 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.

[30] 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 *International Conference on Computational Science and Its Applications – ICCSA 2003*, 2003, pp. 855–864.

[31] 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.

[32] J. Guo and C. Wang, "A low-complexity power-sum circuit for $\mathrm{GF}(2^m)$ and its applications," *IEEE Trans. Circuits & Systems II*, vol. 47, no. 10, pp. 1091–1097, Oct. 2000.

[33] R. B. Lee, Z. Shi, and X. Yang, "Efficient permutation instructions for fast software cryptography," *j-IEEE-MICRO*, vol. 21, no. 6, pp. 56–69, Nov./Dec. 2001.

[34] Z. Shi, X. Yang, and R. B. Lee, "Arbitrary bit permutations in one or two cycles," in *International Conference on Application-Specific Systems, Architectures and Processors — ASAP 2003*, June 2003, pp. 237–247.

[35] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, pp. 449–460, Apr. 2003.

[36] J. Garcia and M. J. Schulte, "A combined 16-bit binary and dual Galois field multiplier," in *Signal Processing Systems, 2002. (SPIS '02). IEEE Workshop on*, Oct.16–18 2002, pp. 63–68.

[37] T. Kobayashi and H. Morita, "Fast modular inversion algorithm to match any operation unit," *IEICE Trans. Fund.*, vol. E82-A, no. 5, pp. 733–740, May 1999.

[38] National Institute of Standards and Technology, *FIPS PUB 186-2: Digital Signature Standard (DSS)*. pub-NIST, Jan. 2000. [Online]. Available: http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

[39] 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.

[40] 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.

[41] K.-Y. Chang, D. Hong, and H.-S. Cho, "Low complexity bit-parallel multiplier for $\mathrm{GF}(2^m)$ defined by all-one polynomials using redundant representation," *IEEE Transactions on Computers*, vol. 54, no. 12, pp. 1628–1630, Dec. 2005.

ACKNOWLEDGEMENT

The author would like to sincerely thank Prof. Naofumi Takagi of Nagoya Univ. for his valuable guidance, discussions, suggestions, comments, and technical supports throughout this study.

The author would also like to sincerely thank Prof. Tomio Hirata of Nagoya Univ. for his guidance and valuable comments in preparing this dissertation.

The author would also like to sincerely thank to Associate Prof. Kazuyoshi Takagi and Assistant Prof. Kazuhiro Nakamura of Nagoya Univ. for their valuable discussions and comments.

The author would also like to thank all reviewers of my journal papers and conference papers, and all audiences of my presentations at workshops and conferences for their valuable comments.

The author would also like to thank the other members of Takagi laboratory for their helpful discussions.

The author would also like to thank my parents for their supports.

This study is partially supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Rohm, Inc.

LIST OF PUBLICATIONS BY THE AUTHOR

JOURNAL PAPERS

1. K. Kobayashi, and N. Takagi, "A combined circuit for multiplication and inversion in $\mathrm{GF}(2^m)$," *IEEE Trans. on Circuits & Systems II*, vol. 55, no. 11, pp. 1144–1148, Nov. 2008.

INTERNATIONAL CONFERENCE

- K. Kobayashi, N. Takagi, and K. Takagi, "An algorithm for inversion in $\mathrm{GF}(2^m)$ suitable for implementation using a polynomial multiply instruction on $\mathrm{GF}(2)$," in *the 18th IEEE Symposium on Computer Arithmetic (ARITH–18)*, pp. 105–112, June 25–27, 2007.

- K. Kobayashi, and N. Takagi, "Design of a combined circuit for multiplication and inversion in $\mathrm{GF}(2^m)$," in *the 14th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI–2007)*, pp. 15–20, Oct. 15–16, 2007.

- K. Kobayashi, and N. Takagi, "Fast division circuit in $\mathrm{GF}(2^m)$ based on the extended Euclid's algorithm with parallelization of modular reductions," in *the 15th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI–2009)*, pp. 94–99, March 9–10, 2009.

TECHNICAL REPORTS

1. K. Kobayashi, N. Takagi, and K. Takagi, "An algorithm using look-up table based on extended Euclid's algorithm for computing inversion in $\mathrm{GF}(2^m)$," in *IEICE Tech. Rep.*, VLD2004–2, pp. 7–12, 2004 (in Japanese).

2. K. Kobayashi, and N. Takagi, "A combined circuit for multiplication and inversion in $\mathrm{GF}(2^m)$ based on the extended Euclid's algorithm," in *IEICE Tech. Rep.*, VLD2006–142, pp. 13–18, 2007 (in Japanese).

3. K. Kobayashi, and N. Takagi, "Hardware algorithm for division in $\mathrm{GF}(2^m)$ based on the extended Euclid's algorithm accelerated with parallelization of modular reductions," in *IEICE Tech. Rep.*, VLD2008–65, pp. 31–36, 2008.

CONVENTION RECORDS

1. K. Kobayashi, N. Takagi, and K. Takagi, "An algorithm for computing inversion in $\mathrm{GF}(2^m)$ based on extended Euclid's algorithm using look-up table," in *Proceedings of the 2003 IEICE General Conference*, D–1–3, March, 2004 (in Japanese).

2. K. Kobayashi, N. Takagi, and K. Takagi, "An algorithm for computing inversion in $\mathrm{GF}(2^m)$ based on extended Euclid's algorithm with reduced number of iterations," in *Proceedings of the 2004 IEICE General Conference*, A–3–2, March, 2005 (in Japanese).

3. K. Kobayashi, N. Takagi, and K. Takagi, "A hardware algorithm for multiplier/divider in $\mathrm{GF}(2^m)$," in *Proceedings of the 2006 IEICE Society Conference*, A–3–7, Sep., 2006 (in Japanese).

4. K. Kobayashi, and N. Takagi, "Accelerated circuit for inversion in $\mathrm{GF}(2^m)$ based on the extended Euclid's algorithm with parallelization of modular reductions," in *Proceedings of the 2008 IEICE Society Conference*, A–3–5, Sep., 2008 (in Japanese).

5. K. Kobayashi, and N. Takagi, "Fast Modified Montgomery Multiplier for $\mathrm{GF}(2^m)$ with Parallelization of Modular Reductions," in *Proceedings of the 2008 IEICE General Conference*, A–3–x, March, 2009 (in Japanese).