A Fast Algorithm for Multiplicative Inversion in $GF(2^m)$ Using Normal Basis

Naofumi Takagi, *Member*, *IEEE*, Jun-ichi Yoshiki, and Kazuyoshi Takagi, *Member*, *IEEE Computer Society*

Abstract—A fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis is proposed. It is an improvement on those proposed by Itoh and Tsujii and by Chang et al., which are based on Fermat's Theorem and require $O(\log m)$ multiplications. The number of multiplications is reduced by decomposing $m-1$ into several factors and a small remainder.

Index Terms—Finite field, finite field inversion, Fermat's theorem, normal basis.

1 Introduction

Finite or Galois field $GF(2^m)$ is used in many applications such as error-correcting codes and cryptography. In these applications, it is crucial to carry out operations, such as addition, multiplication, and multiplicative inversion, in $GF(2^m)$ fast. It is known that multiplicative inversion is much more time-consuming than addition and multiplication and several attempts have been made to carry out this operation fast.

Several algorithms have been proposed for multiplicative inversion in $GF(2^m)$. Some of them are based on Fermat's theorem and use normal basis [1], [2], [3], [4], [5], [6], [7]. Fermat's theorem implies that, for any nonzero element $\beta \in GF(2^m)$, $\beta^{-1} = \beta^{2^m-2}$ and, hence, multiplicative inversion can be carried out by exponentiation by 2^m-2. Wang et al. proposed an algorithm in which the exponentiation is carried out by iterative squarings and multiplications [1]. It requires $m-1$ squarings and $m-2$ multiplications. Since, in the normal basis representation, squaring of an element in $GF(2^m)$ is carried out by a simple cyclic shift and, hence, much faster than multiplication, then it is important to reduce the number of multiplications. Itoh and Tsujii reduced the number of required multiplications to $O(\log m)$ [2], [3]. Feng proposed a similar algorithm, which requires the same number of multiplications as Itoh and Tsujii's [4]. Chang et al. improved Itoh and Tsujii's algorithm and showed that the number of required multiplications can be further reduced for some *m*s by factorizing $m-1$ into two factors [5].

In this paper, we propose a new fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis. It is an improvement on the algorithm proposed by Chang et al. It further reduces the number of required multiplications for some *m*s by decomposing $m-1$ into several factors and a small remainder. It is applicable to some *m*s to which the algorithm by Chang et al. is not applicable. It also reduces the number of multiplications for some *m*s further than the algorithm by Chang et al. For example, when $m = 2^7 = 128$, it requires 10 multiplications, while the one by Itoh and Tsujii requires 12 and the one by Chang et al. is not applicable. When $m = 2^{10} = 1,024$, it requires 13 multiplications, while the one by Itoh and Tsujii requires 18 and the one by Chang et al. requires 14.

In the next section, normal basis and multiplicative inversion in $GF(2^m)$ using normal basis are summarized. We will propose a new fast algorithm in Section 3.

2 Multiplicative Inversion Using Normal Basis

For an $\alpha \in GF(2^m)$, $(\alpha^{2^0}, \alpha^{2^1}, \cdots, \alpha^{2^{m-1}})$ is called a normal basis of $GF(2^m)$ over $GF(2)$ if α^{2^0}, α^{2^1}, $\cdots$, and $\alpha^{2^{m-1}}$ are linearly independent [8]. There exists at least one normal basis for any m. Using a normal basis, any $\beta \in GF(2^m)$ is represented as a vector $(b_0, b_1, \cdots, b_{m-1})$, where $\beta = b_0\alpha^{2^0} + b_1\alpha^{2^1} + \cdots + b_{m-1}\alpha^{2^{m-1}}$ and $b_i \in \{0,1\}$ for $0 \leq i \leq m-1$.

For any β and $\gamma \in GF(2^m)$, $(\beta+\gamma)^2 = \beta^2 + \gamma^2$ holds because $2\beta\gamma = 0$. From Fermat's theorem, i.e., $\beta^{2^m-1} = 1$, $\beta^{2^m} = \beta$ holds. Therefore, when

$$\begin{aligned}\beta &= b_0\alpha^{2^0} + b_1\alpha^{2^1} + \cdots + b_{m-1}\alpha^{2^{m-1}},\\ \beta^2 &= b_0\alpha^{2^1} + b_1\alpha^{2^2} + \cdots + b_{m-1}\alpha^{2^m}\\ &= b_{m-1}\alpha^{2^0} + b_0\alpha^{2^1} + \cdots + b_{m-2}\alpha^{2^{m-1}}.\end{aligned}$$

Hence, in normal basis, when $\beta = (b_0, b_1, \cdots, b_{m-1})$, $\beta^2 = (b_{m-1}, b_0, \cdots, b_{m-2})$. In other words, squaring is carried out by a simple cyclic right shift. Note that powering by 2^i can be carried out by an $(i \bmod m)$-bit cyclic right shift.

Massey and Omura proposed an efficient algorithm for multiplication in $GF(2^m)$ using normal basis [9]. In the algorithm, the fact that squaring is carried out by a cyclic shift is used. Although the algorithm is efficient, multiplication is more time-consuming than squaring.

- *N. Takagi and K. Takagi are with the Department of Information Engineering, Nagoya University, Nagoya 464-8603, Japan. E-mail: ntakagi@nuie.nagoya-u.ac.jp.*
- *J.-i. Yoshiki is with Oki Electric Industry Co., Ltd., 10-3 Shibaura 4-chome, Minato-ku, Tokyo 108-8551, Japan.*

Manuscript received 17 July 1999; revised 20 June 2000; accepted 17 Jan. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 110264.

From Fermat's theorem, for any nonzero element $\beta \in GF(2^m)$, $\beta^{-1} = \beta^{2^m-2}$ holds. Therefore, multiplicative inversion can be carried out by computing β^{2^m-2}.

Since $2^m - 2 = 2^1 + 2^2 + \cdots + 2^{m-1}$,

$$\beta^{-1} = \beta^{2^m-2} = \beta^{2^1} \times \beta^{2^2} \times \cdots \times \beta^{2^{m-1}}.$$

Based on this fact, Wang et al. proposed an algorithm in which the exponentiation is carried out by iterative squarings and multiplications [1]. It requires $m-2$ multiplications as well as $m-1$ squarings.

As stated, using normal basis, squaring is carried out by a simple cyclic shift and, hence, much faster than multiplication. Therefore, it is important to reduce the number of multiplications for accelerating the exponentiation.

Itoh and Tsujii reduced the number of required multiplications to $O(\log m)$ [2], [3]. The algorithm proposed in [3] is based on the following fact: Let $m-1 = 2^{q-1} + m_{q-2}2^{q-2} + \cdots + m_1 2^1 + m_0 2^0$. Namely, $m-1$ is represented as a q-bit binary representation $[1m_{q-2}\cdots m_1 m_0]_2$. Then,

$$\begin{aligned}2^{m-1} - 1 &= (2^{2^{q-1}} - 1)2^{[m_{q-2}\cdots m_1 m_0]_2} + 2^{[m_{q-1}\cdots m_1 m_0]_2} - 1\\ &= (1 + 2^{2^{q-2}})(1 + 2^{2^{q-3}})\cdots\\ &\quad (1 + 2^{2^1})(1 + 2^{2^0})2^{[m_{q-2}\cdots m_1 m_0]_2} + 2^{[m_{q-2}\cdots m_1 m_0]_2} - 1,\end{aligned}$$

where $2^{[m_{q-2}\cdots m_1 m_0]_2}$ means $2^{m_{q-2}2^{q-2}+\cdots+m_1 2^1 + m_0 2^0}$. Furthermore,

$$\begin{aligned}2^{[m_{q-2}\cdots m_1 m_0]_2} - 1 &= m_{q-2}(2^{2^{q-2}} - 1)2^{[m_{q-3}\cdots m_1 m_0]_2}\\ &\quad + 2^{[m_{q-3}\cdots m_1 m_0]_2} - 1\\ &= m_{q-2}(1 + 2^{2^{q-3}})\cdots(1 + 2^{2^1})(1 + 2^{2^0})\\ &\quad 2^{[m_{q-3}\cdots m_1 m_0]_2} + 2^{[m_{q-3}\cdots m_1 m_0]_2} - 1.\end{aligned}$$

Therefore,

$$\begin{aligned}2^{m-1} - 1 &= ((1 + 2^{2^{q-2}})2^{m_{q-2}2^{q-2}} + m_{q-2})(1 + 2^{2^{q-3}})\cdots\\ &\quad (1 + 2^{2^1})(1 + 2^{2^0})2^{[m_{q-3}\cdots m_1 m_0]_2} + 2^{[m_{q-3}\cdots m_1 m_0]_2} - 1.\end{aligned}$$

Iterative application of this reduction yields

$$\begin{aligned}2^{m-1} - 1 &= (((\cdots(((1 + 2^{2^{q-2}})2^{m_{q-2}2^{q-2}} + m_{q-2})(1 + 2^{2^{q-3}})\\ &\quad 2^{m_{q-3}2^{q-3}} + m_{q-3})\cdots)(1 + 2^{2^2})2^{m_2 2^2} + m_2)\\ &\quad (1 + 2^{2^1})2^{m_1 2^1} + m_1)(1 + 2^{2^0})2^{m_0 2^0} + m_0.\end{aligned}$$

Therefore,

$$\begin{aligned}\beta^{-1} &= \beta^{2^m-2} = (\beta^{2^{m-1}-1})^2\\ &= ((((\cdots((\beta^{(1+2^{2^{q-2}})2^{m_{q-2}2^{q-2}}} \times \beta^{m_{q-2}})^{(1+2^{2^{q-3}})2^{m_{q-3}2^{q-3}}} \times \beta^{m_{q-3}})\\ &\quad \cdots)^{(1+2^{2^2})2^{m_2 2^2}} \times \beta^{m_2})^{(1+2^{2^1})2^{m_1 2^1}} \times \beta^{m_1})^{(1+2^{2^0})2^{m_0 2^0}} \times \beta^{m_0})^2.\end{aligned}$$

Hereafter, we call the algorithm proposed in [3] Algorithm[IT]. Algorithm[IT] requires $l(m-1) + w(m-1) - 2$ multiplications and $l(m-1) + w(m-1) - 1$ (multiple-bit) cyclic shifts, where $l(m-1) = q$ is the number of bits of the binary representation of $m-1$ and $w(m-1)$ is the number of 1s in the representation, i.e., the Hamming weight of the representation.

Feng proposed a similar algorithm, which requires the same number of multiplications and cyclic shifts as Algorithm[IT] [4].

Chang et al. improved Algorithm[IT] and showed that the number of required multiplications can be further reduced for some *m*s [5]. The algorithm proposed in [5] is based on the following fact: Let $m-1$ be factorized as $m-1 = s \times t$. Then,

$$\begin{aligned}2^m - 2 &= 2(2^{m-1} - 1) = 2(2^{st} - 1)\\ &= 2(2^s - 1)((2^s)^{t-1} + (2^s)^{t-2} + \cdots + (2^s)^1 + (2^s)^0)\\ &= (2^{s+1} - 2)((2^s)^{t-1} + (2^s)^{t-2} + \cdots + (2^s)^1 + (2^s)^0).\end{aligned}$$

Therefore, $\beta^{-1} = (\beta^{2^{s+1}-2})^{(2^s)^{t-1}+(2^s)^{t-2}+\cdots+(2^s)^1+(2^s)^0}$. $\beta^{2^{s+1}-2}$ can be calculated by Algorithm[IT] with replacing $m-1$ by s. Let t be represented by r-bit binary representation $[1n_{r-2}\cdots n_1 n_0]_2$. Then,

$$\begin{aligned}&(2^s)^{t-1} + (2^s)^{t-2} + \cdots + (2^s)^1 + (2^s)^0\\ &= (((\cdots(((1 + 2^{s2^{r-2}})2^{n_{r-2}s2^{r-2}} + n_{r-2})(1 + 2^{s2^{r-3}})2^{n_{r-3}s2^{r-3}}\\ &\quad + n_{r-3})\cdots)(1 + 2^{s2^2})2^{n_2 s2^2} + n_2)(1 + 2^{s2^1})2^{n_1 s2^1} + n_1)\\ &\quad (1 + 2^{s2^0})2^{n_0 s2^0} + n_0.\end{aligned}$$

Therefore, letting $\beta^{2^{s+1}-2}$ be γ,

$$\begin{aligned}\beta^{-1} &= ((\cdots((\gamma^{(1+2^{s2^{r-2}})2^{n_{r-2}s2^{r-2}}} \times \gamma^{n_{r-2}})^{(1+2^{s2^{r-3}})2^{n_{r-3}s2^{r-3}}} \times \gamma^{n_{r-3}})\\ &\quad \cdots \times \gamma^{n_2})^{(1+2^{s2^1})2^{n_1 s2^1}} \times \gamma^{n_1})^{(1+2^{s2^0})2^{n_0 s2^0}} \times \gamma^{n_0}.\end{aligned}$$

Hereafter, we call the algorithm proposed in [5] Algorithm[Chang]. Algorithm[Chang] requires $(l(s) + w(s) - 2) + (l(t) + w(t) - 2)$ multiplications and $(l(s) + w(s) - 1) + (l(t) + w(t) - 2)$ (multiple-bit) cyclic shifts. The number of multiplications is reduced for some *m*s compared to Algorithm[IT]. For example, when $m = 2^{10} = 1,024$, $m - 1 = 1,023$ can be factorized as 31×33 and the number of required multiplications is 14, while Algorithm[IT] requires 18. Note that the number of required multiplications depends on the way of factorization when $m-1$ contains more than two factors. For example, when $m = 1,024$, $m - 1 = 1,023$ can also be factorized as 11×93 and the number of required multiplications is 15 by this factorization. Note also that the number of required multiplications is not always reduced, even if $m-1$ can be factorized. For example, when $m = 962$, $m - 1 = 961$ can be factorized as 31×31 and the number of required multiplications becomes 16, while Algorithm[IT] requires 13.

3 New Algorithm

The algorithm proposed by Chang et al. is efficient, but it is not applicable to m such that $m-1$ is prime. We propose a new algorithm which is also applicable to such m.

Since

$$\begin{aligned}2^m - 2 &= 2^{m-1} + 2^{m-1} - 2 = 2^{m-1} + 2^{m-2} + \cdots + 2^{m-h}\\ &\quad + 2^{m-h} - 2,\end{aligned}$$

$$\beta^{-1} = \beta^{2^m-2} = \beta^{2^{m-1}} \times \beta^{2^{m-2}} \times \cdots \times \beta^{2^{m-h}} \times \beta^{2^{m-h}-2}.$$

We can calculate $\beta^{2^{m-i}}$ by i-bit cyclic left shift. Therefore, we can obtain β^{-1} from $\beta^{2^{m-h}-2}$ by h multiplications. We can calculate $\beta^{2^{m-h}-2}$ by Algorithm[IT] or Algorithm[Chang] by replacing m by $m-h$.

By this method, we can reduce the number of required multiplications for some ms. For example, when $m = 2^7 = 128$, we can reduce the number of multiplications to 10 by decomposing $m-1 = 127$ as $18 \times 7 + 1$, while Algorithm[IT] requires 12 multiplications and Algorithm[Chang] is not applicable. When $m = 254$, we can reduce the number of multiplications to 11 by decomposing $m-1 = 253$ as $84 \times 3 + 1$, while Algorithm[IT] requires 13 and Algorithm[Chang] requires 12 by factorization of 11×23.

Although it is not stated in [5], it is obvious that the principle used in Algorithm[Chang] can be iteratively applied when $m-1$ contains more than two factors. We can reduce the number of required multiplications for some ms by factorizing $m-1$ into more than two factors. For example, when $m = 2^8 = 256$, we can reduce the number of multiplications to 10 by factorizing $m-1 = 255$ as $3 \times 5 \times 17$. Note that Algorithm[IT] requires 14 multiplications and that Algorithm[Chang] requires 11 by factorization of 15×17 or 5×51 or 3×85. We can adopt this method when $m-h-1$ can be factorized into more than two factors.

Furthermore, the principle that decomposing $m-1$ into several factors and a small remainder h can be recursively applied to one of the factors of $m-h-1$. For example, when $m = 384$, we can reduce the number of multiplications to 13 by decomposing $m-1 = 383$ as $(38 \times 5 + 1) \times 2 + 1$. Note that Algorithm[IT] requires 15 multiplications and that Algorithm[Chang] is not applicable.

Based on the consideration, we propose a new algorithm as follows. In the following algorithm, function $Func$ is recursively called.

```
Algorithm[TYT]
function Func(β, t) /* calculating β^(2^(t+1) - 2) */
{
   if we do not decompose t then
         /* Assume t = [1 m_{q-2} ··· m_1 m_0]_2. */
   {
      γ := β;
      for i := q - 2 to 0 do
      {
         γ := γ × γ^(2^(2^i));
         if m_i = 1 then γ := γ^(2^(2^i)) × β;
      }
      γ := γ^2;
      return γ;
   }
   else
   {
      decompose t as t = ∏_{j=1}^{k} s_j + h;
      γ := Func(β, s_1);
      s := 1;
      for j := 2 to k do
         /* Assume s_j = [1 m^(j)_{q_j-2} ··· m^(j)_1 m^(j)_0]_2. */
      {
         δ := γ;
         s := s × s_{j-1};
         for i := q_j - 2 to 0 do
         {
            δ := δ × δ^(2^(s 2^i));
            if m^(j)_i = 1 then δ := δ^(2^(s 2^i)) × γ;
         }
         γ := δ;
      }
      for i := 1 to h do δ := δ × β^(2^(t+1-i));
      return δ;
   }
}
main
{
   β^(-1) := Func(β, m - 1);
}
```

When $m-1$ is decomposed as $m-1 = \prod_{j=1}^{k} s_j + h$ and s_1 is not decomposed, the number of required multiplications is $\sum_{j=1}^{k}(l(s_j) + w(s_j) - 2) + h$. This is because the number of required multiplications corresponding to factor s_j is $l(s_j) + w(s_j) - 2$. When the first factor s_1 is decomposed further, we can calculate the number of required multiplications by using this formula iteratively.

The number of required multiplications depends on the way of decomposition. There may exist several decompositions that minimize the number of multiplications. In such a case, it seems better to adopt the simplest decomposition, i.e., the one with the fewest components, in order to make the control of β^{-1} computation simpler. (We refer to the factors and the remainder(s) as components.) We call the decomposition that minimizes the number of required multiplications and consists of the fewest components "optimal decomposition." There may exist more than one optimal decomposition.

The following propositions are useful for finding the optimal decomposition(s) of $m-1$.

Proposition 1. *When $m-1 = 2^n$, the optimal decomposition is $m-1$ itself (nondecomposition) and the number of required multiplications is n.*

Proposition 2. *When $m-1 = 2^{n'}s + h$, where s is odd, the smallest number of required multiplications by a decomposition of $m-1$ as $\prod_{j=1}^{k} s_j + h$ (either s_1 is decomposed further or not) is $n' + h$ plus the number of required multiplications by the optimal decomposition of s.*

When $s_j = 2^{n'_j}s'_j$, the number of required multiplications corresponding to s_j and that corresponding to $2^{n'_j} \times s'_j$ are identical, i.e., $l(s'_j) + w(s'_j) - 2 + n'$. Therefore, the optimal decomposition of $m-1$ does not include a power of 2 as a factor unless it is in the form $S \times 2^{n'} + h$ and S is a decomposition of s with a nonzero remainder, where $m-1 = 2^{n'}s + h$.

When $m-1 = 2^n + c$ $(0 < c < 2^n)$, the decomposition of $m-1$ as $2^n + c$ does not decrease the number of required multiplications because the number of multiplications becomes $n + c$, that is, not less than

TABLE 1
Optimal Decomposition for $m = 2^n$ $(4 \le n \le 16)$

n	$m = 2^n$	$m-1$	optimal decomposition	#mul.	#mul. [IT]
4	16	15	5×3	5	6
5	32	31	$10 \times 3 + 1$	7	8
6	64	63	9×7	8	10
7	128	127	$18 \times 7 + 1$	10	12
8	256	255	$17 \times 5 \times 3$	10	14
9	512	511	73×7	12	16
10	1024	1023	$(10 \times 3 + 1) \times 33$	13	18
11	2048	2047	$(10 \times 3 + 1) \times 66 + 1$	15	20
12	4096	4095	$65 \times 9 \times 7$	15	22
13	8192	8191	$130 \times 9 \times 7 + 1$	17	24
14	16384	16383	$(18 \times 7 + 1) \times 129$	18	26
15	32768	32767	$(520 \times 9 + 1) \times 7$	19	28
16	65536	65535	$257 \times 17 \times 5 \times 3$	19	30

$l(m-1) + w(m-1) - 2 = n + w(c)$. Therefore, it is obvious that, in the optimal decomposition of $m-1$, the remainder h must be smaller than c and, hence,

$$l(m-h-1) = l(m-1) = n+1.$$

When $m-1$ is decomposed as $\prod_{j=1}^{k} s_j + h$ and s_1 is not decomposed further, the number of required multiplications is at least $\sum_{j=1}^{k} l(s_j) + h \ge l(m-1) + h$ because $w(s_j) \ge 2$. When the first factor s_1 is decomposed further, the number of required multiplications corresponding to the optimal decomposition of s_1 is at least $l(s_1)$ and, hence, the number of required multiplications by the optimal decomposition of $m-1$ is also at least $l(m-1) + h$. Therefore, we have the following propositions.

TABLE 2
Optimal Decomposition for $m = 32k$ $(k \le 31)$

m	$m-1$	optimal decomposition	#mul.	#mul. [IT]
32	31	$10 \times 3 + 1$	7	8
64	63	9×7	8	10
96	95	19×5	9	11
128	127	$18 \times 7 + 1$	10	12
160	159	53×3	10	12
192	191	$38 \times 5 + 1$	11	13
224	223	$74 \times 3 + 1$	11	13
256	255	$17 \times 5 \times 3$	10	14
288	287	41×7	11	13
320	319	29×11	12	14
352	351	$13 \times 9 \times 3$	11	14
384	383	$(38 \times 5 + 1) \times 2 + 1$	13	15
416	415	83×5	12	14
448	447	149×3	12	15
480	479	$(34 \times 7 + 1) \times 2 + 1$	13	15
512	511	73×7	12	16
544	543	$(36 \times 5 + 1) \times 3$	12	14
576	575	115×5	13	15
608	607	$202 \times 3 + 1$	13	15
640	639	213×3	13	16
672	671	$(20 \times 3 + 1) \times 11$	13	15
704	703	37×19	13	16
736	735	$49 \times 5 \times 3$	12	16
768	767	59×13	14	17
800	799	$266 \times 3 + 1$	13	15
832	831	277×3	13	16
864	863	$41 \times 21 + 2$	15	16
896	895	179×5	14	17
928	927	$(34 \times 3 + 1) \times 9$	13	16
960	959	137×7	13	17
992	991	$66 \times 5 \times 3 + 1$	13	17

TABLE 3
Optimal Decomposition with More than Two Components for $m \le 256$

m	$m-1$	optimal decomposition	#mul.	#mul. [IT]
32	31	$10 \times 3 + 1$	7	8
62	61	$20 \times 3 + 1$	8	9
63	62	$(10 \times 3 + 1) \times 2$	8	9
80	79	$26 \times 3 + 1$	9	10
94	93	$(10 \times 3 + 1) \times 3$	9	10
104	103	$34 \times 3 + 1$	9	10
110	109	$36 \times 3 + 1$	9	10
122	121	$40 \times 3 + 1$	9	10
123	122	$(20 \times 3 + 1) \times 2$	9	10
125	124	$(10 \times 3 + 1) \times 4$	9	10
128	127	$18 \times 7 + 1$	10	12
136	135	$9 \times 5 \times 3$	9	10
152	151	$50 \times 3 + 1$	10	11
156	155	$(10 \times 3 + 1) \times 5$	10	11
158	157	$52 \times 3 + 1$	10	11
159	158	$(26 \times 3 + 1) \times 2$	10	11
182	181	$36 \times 5 + 1$	10	11
184	183	$(20 \times 3 + 1) \times 3$	10	12
187	186	$(10 \times 3 + 1) \times 6$	10	11
192	191	$38 \times 5 + 1$	11	13
200	199	$66 \times 3 + 1$	10	11
207	206	$(34 \times 3 + 1) \times 2$	10	11
218	217	$72 \times 3 + 1$	10	11
219	218	$(36 \times 3 + 1) \times 2$	10	11
224	223	$74 \times 3 + 1$	11	13
236	235	$26 \times 9 + 1$	11	12
238	237	$(26 \times 3 + 1) \times 3$	11	12
240	239	$34 \times 7 + 1$	11	13
242	241	$80 \times 3 + 1$	10	11
243	242	$(40 \times 3 + 1) \times 2$	10	11
245	244	$(20 \times 3 + 1) \times 4$	10	11
249	248	$(10 \times 3 + 1) \times 8$	10	11
252	251	$50 \times 5 + 1$	11	13
254	253	$84 \times 3 + 1$	11	13
255	254	$(18 \times 7 + 1) \times 2$	11	13
256	255	$17 \times 5 \times 3$	10	14

Proposition 3. *In the optimal decomposition of $m-1$, the remainder h must be smaller than $w(m-1) - 2$.*

Proposition 4. *When $m - 1 = 2^n + 2^{n'}$, where $n > n'$, i.e., $w(m-1) = 2$, the optimal decomposition is $m-1$ itself and the number of required multiplications is $n+1$.*

When m is given, we can find the optimal decomposition(s) of $m-1$ by an exhaustive search with efficient pruning using the above propositions. Note that we can also use the above propositions for finding the optimal decomposition of the first factor s_1. Although it is an interesting problem, the problem of finding the optimal decomposition is not crucial because we have to solve this problem only once when we choose m.

In practical applications, m is frequently selected as a power of 2. When $m = 2^n$, $m - 1 = 2^n - 1$ and $l(m-1) = w(m-1) = n$. If we do not decompose $m-1$, Algorithm[IT] requires $2n-2$ multiplications. When n is even, $2^n - 1$ can be factorized as $(2^{n/2}+1) \times (2^{n/2}-1)$ and, when $n/2$ is even again, $2^{n/2}-1$ can be factorized further.

In such a case, we can greatly reduce the number of multiplications. On the other hand, it is known that $2^n - 1$ is prime for $n = 5, 7, 13, 19, \cdots$. When $2^n - 1$ is prime, Algorithm[Chang] is not applicable. In such a case, since n (> 2) is odd, we can always decompose $2^n - 1$ as $2(2^{(n-1)/2} + 1) \times (2^{(n-1)/2} - 1) + 1$ and can reduce the number of multiplications by our algorithm. Table 1 shows one of the optimal decompositions of $m - 1$ for $m = 2^n$ $(4 \leq n \leq 16)$.

In digital systems, m is often selected as a multiple of word size of the computer such as $32k$, where k is an integer. Table 2 shows one of the optimal decompositions of $m - 1$ for $m = 32k$ $(1 \leq k \leq 31)$.

Table 3 shows one of the optimal decompositions of $m - 1$ for every m (≤ 256) such that the optimal decomposition consists of more than two components. For such m, our algorithm requires fewer multiplications than Algorithm[IT] and than Algorithm[Chang].

4 Conclusion

We have proposed a new fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis. It is based on Fermat's Theorem. The number of required multiplications is reduced by decomposing $m - 1$ into several factors and a small remainder. We have shown the effectiveness of the proposed algorithm by showing optimal decompositions of $m - 1$ for practical ms.

The proposed algorithm can be easily modified for multiplicative inversion in $GF((2^n)^m)$ or in $GF(p^m)$, where p is an odd prime.

References

[1] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architecture for Computing Multiplications and Inverses in GF(2^m)," *IEEE Trans. Computers,* vol. 34, no. 8, pp. 709-716, Aug. 1985.

[2] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Basis," *Information and Computing,* vol. 78, pp. 171-177, 1988.

[3] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in Finite Fields Using Normal Basis," *IEICE Trans. (A).,* vol. J70-A, no. 11, pp. 1637-1645, Nov. 1989 (in Japanese).

[4] G.L. Feng, "A VLSI Architecture for Fast Inversion in $GF(2^m)$," *IEEE Trans. Computers,* vol. 38, no. 10, pp. 1383-1386. Oct. 1989.

[5] T. Chang, E. Lu, Y. Lee, Y. Leu, and H. Shyu, "Two Algorithms for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Basis," accepted by *Information Processing Letters.*

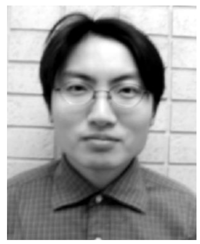
[6] T. Itoh and S. Tsujii, "Effective Recursive Algorithm for Computing Multiplicative Inverses in $GF(2^m)$," *Electronics Letters,* vol. 24, no. 6, pp. 334-335, Mar. 1988.

[7] Y. Asano, T. Itoh, and S. Tsujii, "Generalized Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$," *Electronics Letters,* vol. 25, no. 10, pp. 664-665, May 1989.

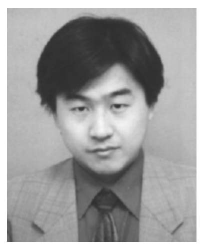
[8] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes.* New York: North-Holland, 1977.

[9] J.L. Massey and J.K. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," US Patent Application, submitted 1981.

Naofumi Takagi (S'82-M'84) received the BE, ME, and PhD degrees in information science from Kyoto University, Kyoto, Japan, in 1981, 1983, and 1988, respectively. He joined the Department of Information Science, Kyoto University, as an instructor in 1984 and was promoted to an associate professor in 1991. He moved to the Department of Information Engineering, Nagoya University, Nagoya, Japan, in 1994, where he has been a professor since 1998. His current interests include computer arithmetic, hardware algorithms, and logic design. He received the Japan IBM Science Award and the Sakai Memorial Award of the Information Processing Society of Japan in 1995. Dr. Takagi was an associate editor of the *IEEE Transactions on Computers* from 1996 to 2000. He is a member of the IEEE.



Jun-ichi Yoshiki received the BE degree in information engineering and ME degree in computing science and engineering from Nagoya University, Nagoya, Japan, in 1997 and 1999, respectively. While he was a student at the university, he researched algorithms for arithmetic operations in Galois fields. He is now with Oki Electric Industry Co., Ltd., Tokyo, Japan.



Kazuyoshi Takagi received the BE, ME, and DrEng degrees in information science from Kyoto University, Kyoto, Japan, in 1991, 1993, and 1999, respectively. From 1995 to 1999, he was a research associate at the Nara Institute of Science and Technology. Since 1999, he has been an assistant professor in the Department of Information Engineering, Nagoya University, Nagoya, Japan. His current interests include circuit complexity theory, VLSI design, and VLSI CAD algorithms. He is a member of the IEEE Computer Society.