Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Low-complexity multiplexer-based normal basis multiplier over GF(2^m)

Jenn-Shyong HORNG[†1], I-Chang JOU[1], Chiou-Yng LEE[2]

([1]*Institute of Engineering Science and Technology, National Kaohsiung First University of Science and Technology, Taiwan 811, Kaohsiung County*)

([2]*Department of Computer Information and Network Engineering, Lunghwa University of Science and Technology, Taiwan 333, Taoyuan County*)

[†]E-mail: shyong1@cht.com.tw

Received May 26, 2008; Revision accepted Nov. 18, 2008; Crosschecked Apr. 29, 2009

Abstract: We present a new normal basis multiplication scheme using a multiplexer-based algorithm. In this algorithm, the proposed multiplier processes in parallel and has a multiplexer-based structure that uses MUX and XOR gates instead of AND and XOR gates. We show that our multiplier for type-1 and type-2 normal bases saves about 8% and 16%, respectively, in space complexity as compared to existing normal basis multipliers. Finally, the proposed architecture has regular and modular configurations and is well suited to VLSI implementations.

Key words: Finite field multiplication, Normal basis, Gaussian normal basis, Elliptic curve cryptosystem
doi:10.1631/jzus.A0820398 **Document code:** A **CLC number:** TP309

INTRODUCTION

Efficient software algorithms and hardware implementations for arithmetic operations in a Galois field GF(2^m) are frequently required in coding theory, computer algebra, and elliptic curve cryptosystems (MacWilliams and Sloane, 1977; Lidl and Niederreiter, 1994). The success and efficiency of cryptographic applications such as encryption/decryption algorithms rely on efficient computations in very large finite fields. Multiplication is used heavily in cryptographic computations, so the design of an efficient multiplier is highly desirable for improving the operation of cryptographic schemes.

In a binary extension field, the field has 2^m elements and is conveniently represented as an m-dimensional vector space defined over GF(2). Any element can be represented as a linear combination of these basis vectors. There are three major basis representations: the polynomial basis (PB), the dual basis (DB), and the normal basis (NB). Finite field multiplications have previously used mostly AND-XOR architectures (Hasan *et al*., 1992; Koc and Sunar, 1998; Lee *et al*., 2001; Sunar and Koc, 2001; Elia and Leone, 2002; Reyhani-Masoleh and Hasan, 2002; Kim *et al*., 2006). These architectures have not, however, achieved effective reductions in space complexity. Multiplexer-based multipliers over GF(2^m) are actively being investigated to overcome this problem, for example, the all-one polynomial multiplier proposed by Byun and Kim (2003). A DB multiplier using a multiplexer-based algorithm was proposed by Lee *et al*.(2005), and a bit-parallel systolic multiplier over GF(2^m) using this algorithm was derived by Lee *et al*.(2006).

The major advantage of NB representation is that squaring of an element can be performed simply by cyclically shifting its binary form. The bit-serial NB multiplier for GF(2^m) was discovered by Massey and Omura (1986) and depends on the selection of a key function for multiplication. Various efficient bit-parallel and bit-serial architectures for normal basis multiplication over GF(2^m) have been developed (Ash *et al*., 1989; Mullin *et al*., 1989; Agnew *et al*., 1991;

Hasan *et al*., 1993; Reyhani-Masoleh and Hasan, 2003; 2005). From the IEEE Standard Specifications of Public Key Cryptography (IEEE Std. 1363, 2000), each NB element in GF(2^m) can also be represented in a Gaussian period of (*m*, *t*), sometimes called the Gaussian normal basis (GNB) of GF(2^m). The GNB is a special class of NB that has received considerable attention because of its efficient implementation of field multiplication (Reyhani-Masoleh, 2006). An NB is known to exist for every positive integer *m* for GF(2^m). However, a GNB exists only if *m* is not divisible by 8 (Hasan *et al*., 1993). Note that every GNB for GF(2^m) is dependent on an integer *t* and is referred to as a type-*t* GNB. Reyhani-Masoleh (2006) presented two vector-level software algorithms that essentially eliminate such bit-wise operations for the GNB of GF(2^m).

In this paper we consider a multiplexer-based algorithm to improve the organization of generalized NB multiplication over GF(2^m) to achieve reductions in space complexity. The proposed architecture leverages divide-and-conquer and uniform shift methods to reduce the hardware area occupied by the bit-parallel implementation significantly. The proposed architecture is mainly composed of multiplexers rather than the conventional AND and XOR gates. The proposed multiplier for type-1 and type-2 NBs has lower space complexity than other normal basis multipliers.

MATHEMATICAL BACKGROUND

In this section, we will briefly review normal basis representation and multiplication operation.

Normal basis representation

An NB is known to exist in the field GF(2^m) over GF(2) for all positive integers *m*. Any element in GF(2^m) can also be viewed as a vector space of dimension *m* over GF(2^m). The set $N_1=\{\alpha, \alpha^2, \cdots, \alpha^{2^{m-1}}\}$ is called an NB, and a normal element *A* in GF(2^m) can be represented by $A=a_0\alpha+a_1\alpha^2+\cdots+a_{m-1}\alpha^{2^{m-1}}=(a_0, a_1, \cdots, a_{m-1})$, where α is a normal element of GF(2^m) if the trace function of α equals unity, i.e., tr α=1.

Definition 1 Let γ be a primitive (*mt*+1)th root of unity in some extension field. A Gaussian period of type (*m*, *t*) over GF(2^m) is defined as

$$\alpha=\gamma+\gamma^{<2^m>_p}+\cdots+\gamma^{<2^{(t-1)m}>_p}, \tag{1}$$

where $p=mt+1$ is a prime and gcd(*tm*/*k*, *m*)=1, where k is the multiplicative order of '2 mod *p*'. The notation 'gcd( )' stands for the greatest common denominator. Throughout this paper, $<x>_i$ denotes the nonnegative residue of '*x* mod *i*'. Applying the characteristics of Eq.(1), $A=a_0\alpha+a_1\alpha^2+\ldots+a_{m-1}\alpha^{2^{m-1}}$ can also be represented as

$$\begin{aligned}A=&a_0(\gamma+\gamma^{<2^m>}+\cdots+\gamma^{<2^{(t-1)m}>})\\&+a_1(\gamma^2+\gamma^{<2^{2m+1}>}+\cdots+\gamma^{<2^{(t-1)m+1}>})\\&+\cdots\\&+a_{m-1}(\gamma^{<2^{m-1}>}+\gamma^{<2^{2m-1}>}+\gamma^{<2^{2m}>}+\gamma^{<2^{2m+1}>}\\&\qquad+\cdots+\gamma^{<2^{tm-1}>}).\end{aligned} \tag{2}$$

With this element representation, the normal basis N_1 can be transformed into the redundant basis $N_2=\{\gamma, \gamma^2, \ldots, \gamma^{p-1}\}$, as $\gamma^p=1$. The field element can be rewritten as

$$A=a_{F(1)}\gamma+a_{F(2)}\gamma^2+\cdots+a_{F(p-1)}\gamma^{p-1}, \tag{3}$$

where $F(2^iu^j \bmod p)=i$ ($0\le i\le m-1$, $0\le j<t$), and *u* is an integer of order '*t* mod *p*'. For example, if *m*=5 and *t*=2, we use $\alpha=\gamma+\gamma^{<2^m>_p}=\gamma+\gamma^{10}$ to generate the NB $\{\alpha, \alpha^2, \alpha^{2^2}, \alpha^{2^3}, \alpha^{2^4}\}$, which can be re-expressed by the redundant representation $\{\gamma, \gamma^2, \gamma^3, \gamma^4, \gamma^5, \gamma^6, \gamma^7, \gamma^8, \gamma^9, \gamma^{10}\}$. In this way, $A=(a_0, a_1, \ldots, a_{m-1})$ can also be represented by $A=a_0\gamma+a_1\gamma^2+a_2\gamma^4+a_3\gamma^3+a_4\gamma^5+a_4\gamma^6+a_2\gamma^7+a_3\gamma^8+a_1\gamma^9+a_0\gamma^{10}$, where the coefficients of *A* are duplicated in the *t*-term coefficients of the original NB element $A=(a_0, a_1, \ldots, a_{m-1})$ if the field element *A* presents a type-*t* NB of GF(2^m). Significantly, multiplication using this redundant representation can be performed easily in a circular convolution structure, as previously reported by Lee *et al*.(2001). Applying the function $F(2^iu^j \bmod p)=i$, the field element $A=(a_{F(1)}, a_{F(2)}, \ldots, a_{F(p-1)})$ can be

expressed as

$$A=(\underbrace{a_0,\cdots,a_0}_{t},\underbrace{a_1,\cdots,a_1}_{t},\underbrace{a_2,\cdots,a_2}_{t},\cdots,\underbrace{a_{m-1},\ \cdots,\ a_{m-1}}_{t}).$$

Therefore, it is easy to perform the basis conversion from the redundant basis to the NB. For the example m=5 and t=2, $F(1)$=0, $F(2)$=1, $F(3)$=3, $F(4)$=2, $F(5)$=4, $F(6)$=4, $F(7)$=2, $F(8)$=3, $F(9)$=1, and $F(10)$=0. This means that

$$\begin{aligned}A&=a_{F(1)}\gamma+a_{F(2)}\gamma^2+a_{F(3)}\gamma^3+a_{F(4)}\gamma^4+a_{F(5)}\gamma^5\\&\quad+a_{F(6)}\gamma^6+a_{F(7)}\gamma^7+a_{F(8)}\gamma^8+a_{F(9)}\gamma^9+a_{F(10)}\gamma^{10}\\&=a_0\gamma+a_1\gamma^2+a_2\gamma^4+a_3\gamma^3+a_4\gamma^5+a_4\gamma^6+a_2\gamma^7\\&\quad+a_3\gamma^8+a_1\gamma^9+a_0\gamma^{10}.\end{aligned}$$

Remark 1 Let $\alpha=\gamma+\gamma^{2^m}+\cdots+\gamma^{2^{m(t-1)}}$ generate NB of GF(2^m) with t being an odd number. m is then an even number.

Conventional normal basis multiplication

Let A, B be two normal basis elements of GF(2^m). By using the redundant basis representation, the product $C=AB=(c_0, c_1, \ldots, c_{m-1})$ can then be written as

$$c_0=F(A,B)=\sum_{j=1}^{p-2}a_{F(j)}b_{F(p-j)}. \tag{4}$$

In general, this means that $c_i=F(A^{2^i}, B^{2^i})$. Therefore, the product of A and B can be calculated by Algorithm 1.

Algorithm 1 Conventional GNB multiplication

Input: A, $B\in$GF(2^m) and $F(A, B)$ in Eq.(4).
Output: $C=AB$.

```
C=0;
For (i=0 to m−1)
   c_i=F(A, B);
   A=A<<1; B=B<<1;
Return C.
```

We use a 2-input gate to analyze the complexity of the multiplier. Assuming that C_N denotes the number of product terms a_ib_j in $F(A, B)$, Reyhani-Masoleh and Hasan (2005) showed that the complexity of the NB is $C_N\geq 2m-1$ and an optimal NB multiplier has $C_N=2m-1$. Implementing c_i requires C_N AND gates and C_N-1 XOR gates. Hence, type-1 and type-2 GNBs can achieve the minimal space complexity. However, there are problems for $t>2$ where the space complexity becomes large.

PROPOSED NB MULTIPLICATION ALGORITHM

Any element A, $B\in$GF(2^m) can be represented by $A=a_0\alpha+a_1\alpha^2+\cdots+a_{m-1}\alpha^{2^{m-1}}$ and $B=b_0\alpha+b_1\alpha^2+\cdots+b_{m-1}\alpha^{2^{m-1}}$, where a_j, $b_j\in$GF(2) (j=0, 1, …, $m-1$). For simplicity, we define the following algebraic operation:

Definition 2 Let $A=a_0\alpha+a_1\alpha^2+\cdots+a_{m-1}\alpha^{2^{m-1}}$ be a normal element in GF(2^m), defined as

$$A_i=a_i\alpha+a_{i+1}\alpha^2+\cdots+a_{m-1}\alpha^{2^{m-i-1}} \text{ for } 0\leq i\leq m-1.$$

With Definition 2, the field element A can be represented by extending the representation to

$$\begin{aligned}A&=a_0\alpha+a_1\alpha^2+a_2\alpha^{2^2}+a_3\alpha^{2^3}+a_4\alpha^{2^4}+\cdots+a_{m-1}\alpha^{2^{m-1}}\\&=a_0\alpha+(a_1\alpha+a_2\alpha^2+a_3\alpha^{2^2}+a_4\alpha^{2^3}+\cdots+a_{m-1}\alpha^{2^{m-2}})^2\\&=a_0\alpha+A_1^2.\end{aligned} \tag{5}$$

Similarly,

$$\begin{aligned}B&=b_0\alpha+b_1\alpha^2+b_2\alpha^{2^2}+b_3\alpha^{2^3}+b_4\alpha^{2^4}+\cdots+b_{m-1}\alpha^{2^{m-1}}\\&=b_0\alpha+B_1^2.\end{aligned}$$

Notably, $A_0=A$ and $B_0=B$. A_iB_i will be used as proven in Theorem 1 based on Eq.(5).

Theorem 1 According to Definition 1, A_iB_i can be calculated by

$$A_iB_i=\alpha Z_i+(A_{i+1}B_{i+1})^2, \tag{6}$$

where

$$Z_i=a_iB_i+b_iA_{i+1}^2. \tag{7}$$

Proof Applying Definition 2, we have

$$\begin{aligned}A_i&=a_i\alpha+a_{i+1}\alpha^2+\cdots+a_{m-1-i}\alpha^{2^{m-1-i}}\\&=a_i\alpha+(a_{i+1}\alpha+\cdots+a_{m-1-i}\alpha^{2^{m-2-i}})^2=a_i\alpha+A_{i+1}^2.\end{aligned}$$

Thus, B_i can also be represented by $B_i = b_i\alpha + B_{i+1}^2$, and the product of A_i and B_i is

$$A_iB_i = \alpha(a_iB_i + b_iA_{i+1}^2) + (A_{i+1}B_{i+1})^2 = \alpha Z_i + (A_{i+1}B_{i+1})^2.$$

Following Definition 1, let A and B be rewritten as $A = a_0\alpha + A_1^2$ and $B = b_0\alpha + B_1^2$, respectively. Then, based on Eq.(7), the product C of A and B applies the basic concept of divide-and-conquer to derive the following formula:

$$\begin{aligned} C = AB &= (a_0\alpha + A_1^2)(b_0\alpha + B_1^2) \\ &= a_0\alpha(b_0\alpha + B_1^2) + b_0\alpha A_1^2 + (A_1B_1)^2 \\ &= \alpha(a_0B_0 + b_0A_1^2) + (A_1B_1)^2 \\ &= \alpha Z_0 + (A_1B_1)^2 \\ &= \alpha Z_0 + (\alpha Z_1)^2 + (A_2B_2)^{2^2} \\ &= \cdots \\ &= \alpha Z_0 + (\alpha Z_1)^2 + (\alpha Z_2)^{2^2} + (\alpha Z_3)^{2^3} + \cdots + (\alpha Z_{m-1})^{2^{m-1}}. \end{aligned} \tag{8}$$

According to Eq.(8), the product C is included by m-term αZ_i computations and $(m-1)$-term squaring operations. From the normal basis representation, the squaring of an element in GF(2^m) is performed simply by cyclically shifting its binary form. Thus, the major operation of Eq.(8) relies on αZ_i computations. The following steps are derived for efficient αZ_i operation:

Step 1: Convert Z_i to the redundant basis form.

Let $Z_i = z_{i,0}\alpha + z_{i,1}\alpha^2 + \cdots + z_{i,m-1-i}\alpha^{2^{m-1-i}}$ can be represented by the type-t GNB element of GF(2^m). Applying the characteristics of Eq.(2), Z_i can also be re-expressed as

$$\bar{Z}_i = z_{i,F(0)} + z_{i,F(1)}\gamma + z_{i,F(2)}\gamma^2 + z_{i,F(3)}\gamma^3 + \cdots + z_{i,F(p-1)}\gamma^{p-1}, \tag{9}$$

where $z_{i,F(0)}=0$, and $p=mt+1$ is a prime number.

Step 2: Calculate $\alpha\bar{Z}_i$.

From Definition 1, α is represented by $\alpha = \gamma + \gamma^{<2^m>_p} + \cdots + \gamma^{<2^{(t-1)m}>_p}$. Then $\alpha\bar{Z}_i$ can be computed by $\alpha\bar{Z}_i = (\gamma + \gamma^{<2^m>_p} + \cdots + \gamma^{<2^{m(t-1)}>_p})\bar{Z}_i$. Since $\gamma^p=1$, we have

$$\begin{aligned} \gamma\bar{Z}_i &= z_{i,F(p-1)} + z_{i,F(0)}\gamma + z_{i,F(1)}\gamma^2 + z_{i,F(2)}\gamma^3 \\ &\quad + \cdots + z_{i,F(p-2)}\gamma^{p-1} = \bar{Z}_i^{(1)}. \end{aligned} \tag{10}$$

Note that $\bar{Z}_i^{(1)}$ denotes a right shift of $\bar{Z}_i$ by one position. Similarly, $\gamma^{<2^k>_p}\ \bar{Z}_i = \bar{Z}_i^{(<2^k>_p)}$, for $0\le k\le m-1$. Therefore, the product of α and $\bar{Z}_i$ is obtained from

$$\begin{aligned} \alpha\ \bar{Z}_i &= (\gamma + \gamma^{2^m} + \cdots + \gamma^{2^{m(t-1)}})\bar{Z}_i \\ &= \bar{Z}_i^{(1)} + \bar{Z}_i^{(<2^m>_p)} + \cdots + \bar{Z}_i^{(<2^{m(t-1)}>_p)} \\ &= \sum_{j=0}^{p-1} z_{i,F(j-1)}\gamma^j + \sum_{j=0}^{p-1} z_{i,F(j-<2^m>)}\gamma^j + \cdots + \sum_{j=0}^{p-1} z_{i,F(j-<2^{m(t-1)}>)}\gamma^j \\ &= \sum_{j=0}^{p-1} z'_{i,F(j)}\gamma^j, \end{aligned} \tag{11}$$

where $z'_{i,F(j)} = \sum_{k=0}^{t-1} z_{i,F(j-<2^{mk}>)}$.

Step 3: Basis conversion from the redundant basis to the normal basis.

According to Eq.(11), the NB representation of αZ_i can be derived from its redundant basis representation to give

$$\alpha Z_i = z'_{i,F(0)} + \sum_{j=0}^{m-1} z'_{i,F(2^j)}\alpha^{2^j}. \tag{12}$$

It is easy to obtain αZ_i using the above steps. The following example illustrates αZ_i computations to make this clearer.

Example 1 Let $A=(a_0, a_1, a_2, a_3, a_4)$ be the NB element of GF(2^5), and let $\alpha = \gamma + \gamma^{10}$ be used to generate the NB. Applying redundant representation, the field element A can be represented by

$$\begin{aligned} A = {} & a_0\gamma + a_1\gamma^2 + a_3\gamma^3 + a_2\gamma^4 + a_4\gamma^5 + a_4\gamma^6 \\ & + a_2\gamma^7 + a_3\gamma^8 + a_1\gamma^9 + a_0\gamma^{10}. \end{aligned}$$

Then, αA is obtained by

$$\alpha A = (\gamma + \gamma^{10})A = A^{(1)} + A^{(10)},$$

where

$$A^{(1)} = a_0 + a_0\gamma^2 + a_1\gamma^3 + a_3\gamma^4 + a_2\gamma^5 + a_4\gamma^6 + a_4\gamma^7 + a_2\gamma^8 + a_3\gamma^9 + a_1\gamma^{10},$$

$$A^{(10)} = a_0 + a_1\gamma + a_3\gamma^2 + a_2\gamma^3 + a_4\gamma^4 + a_4\gamma^5 + a_2\gamma^6 + a_3\gamma^7 + a_1\gamma^8 + a_0\gamma^9.$$

Therefore, from the redundant basis to the NB, we can obtain $\alpha A=(a_1, a_0+a_3, a_3+a_4, a_2+a_1, a_2+a_4)$. As mentioned above, the proposed NB multiplication scheme is shown in Algorithm 2.

Algorithm 2 Proposed NB multiplication
Input: $A=(a_0, a_1, \ldots, a_{m-1})$, $B=(b_0, b_1, \ldots, b_{m-1})$.
Output: $C=AB$.

Step 1: Initial step.
  C=0;
  A=A<<1;
Step 2: For (i=0 to m−1)
2.1 If (a_i==1 && b_i==1) then $Z_i=A^2+B$;
2.2 Else if (a_i==1 && b_i==0) then $Z_i=A^2$;
2.3 Else if (a_i==0 && b_i==1) then $Z_i=B$;
2.4 Else if (a_i==0 && b_i==0) then Z_i=0;
2.5 $Z_i=\alpha Z_i$;
2.6 $C=C+Z_i>>i$;
2.7 $A=A<<1$, $B=B<<1$;
Step 3: Return C.

In the four steps 2.1~2.4, $Z_i = a_iB_i + b_iA_{i+1}^2$ in Eq.(7) can be selected from any of 0, B_i, A_{i+1}^2, or $B_i + A_{i+1}^2$, depending on the values of a_i and b_i. This clearly shows that $Z_i = a_iB_i + b_iA_{i+1}^2$ requires actual computation only when a_i=b_i=1. After Z_i is assigned, step 2.5 uses Eq.(12) to compute αZ_i. In step 2.6, Z_i adds the previous result C to store the register C. After m clock cycles, the register C can be used to obtain the product $C=AB$ in Eq.(8).

HARDWARE IMPLEMENTATION AND COMPLEXITY ANALYSIS

In this section we consider Algorithm 2 to implement a multiplexer-based NB multiplier. Fig.1 shows the proposed NB multiplier structure including the addition unit, the MUX array, the $(\alpha Z_i)^{2^i}$ unit, and the summation unit. In the following we analyze the configuration of the four units.

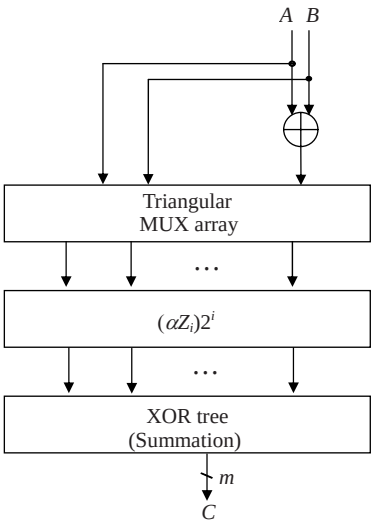

Fig.1 Proposed architecture of the NB multiplier

Structure of the addition unit and MUX array

In the ith computational loop of Algorithm 2, the four steps 2.1~2.4 for computing the function $Z_i = a_iB_i + b_iA_{i+1}^2$ can be performed using any of the alternatives in Table 1. This clearly shows that $Z_i = B_i + A_{i+1}^2$ requires actual computation only when a_i=b_i=1. Hence, the value of Z_i can be selected from 0, B_i, A_{i+1}^2, or $B_i + A_{i+1}^2$, depending on the values of a_i and b_i. In the next computational loop, Z_{i+1} can be determined from 0, B_{i+1}, A_{i+2}^2, or $B_{i+1}+ A_{i+2}^2$, depending on the values of a_{i+1} and b_{i+1}, where the three terms B_{i+1}, A_{i+2}^2, and $B_{i+1} + A_{i+2}^2$ are shifted from the three terms B_i, A_{i+1}^2, and $B_i + A_{i+1}^2$, respectively. From this shifting operation, the addition unit uses m–1 XOR gates to compute $B_0 + A_1^2$. After $B_0 + A_1^2$ is calculated in the addition unit, three data B_i, A_{i+1}^2, and $B_i + A_{i+1}^2$ from the top to input the MUX array (Fig.2) can be produced by the values Z_i ($0\le i\le m-1$). In Fig.2, the ith row MUX uses m–i MUXes to produce the value

$$Z_i = a_iB_i + b_iA_{i+1}^2 = (a_ib_i,\ a_ib_{i+1} + b_ia_{i+1},\ a_ib_{i+2} + b_ia_{i+2},\ \cdots,\ a_ib_{m-1} + b_ia_{m-1},\ \underbrace{0,\ \cdots,\ 0}_{i}).$$

Thus, the structure of the MUX array requires $m(m-1)/2$ MUX 4×1 gates. For example, if m=5, we have

Table 1 $Z_i = a_iB_i + b_iA_{i+1}^2$ function decision table

a_i	b_i	Z_i
0	0	0
0	1	A_{i+1}^2
1	0	B_i
1	1	$B_i + A_{i+1}^2$

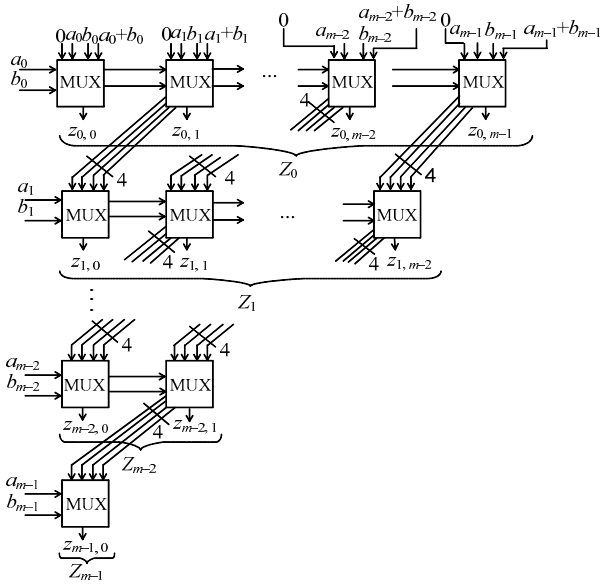

Fig.2 Triangular MUX array for computing Z_i

$$\begin{cases} Z_0 = (a_0b_0,\ a_0b_1 + b_0a_1,\ a_0b_2 + b_0a_2,\ a_0b_3 + b_0a_3, \\ \qquad a_0b_4 + b_0a_4), \\ Z_1 = (a_1b_1,\ a_1b_2 + b_1a_2,\ a_1b_3 + b_1a_3,\ a_1b_4 + b_1a_4,\ 0), \\ Z_2 = (a_2b_2,\ a_2b_3 + b_2a_3,\ a_2b_4 + b_2a_4,\ 0,\ 0), \\ Z_3 = (a_3b_3,\ a_3b_4 + b_3a_4,\ 0,\ 0,\ 0), \\ Z_4 = (a_4b_4,\ 0,\ 0,\ 0,\ 0). \end{cases}$$

Structure of $(\alpha Z_i)^{2^i}$ and summation units

After the values Z_i ($0 \le i \le m-1$) have been computed in the MUX array, the $(\alpha Z_i)^{2^i}$ unit is performed by $C = \alpha Z_0 + (\alpha Z_1)^2 + (\alpha Z_2)^{2^2} + (\alpha Z_3)^{2^3} + \cdots + (\alpha Z_{m-1})^{2^{m-1}}$ in Eq.(8). Let $Z_i = z_{i,0}\alpha + z_{i,1}\alpha^2 + \cdots + z_{i,m-1-i}\alpha^{2^{m-1-i}}$, and compute αZ_i based on Eq.(11) to obtain

$$\alpha Z_i = z'_{i,F(0)} + \sum_{j=0}^{m-1} z'_{i,F(2^j)}\alpha^{2^j},$$

where $z'_{i,F(2^j)} = \sum_{k=0}^{t-1} z_{i,F(<2^j-2^{mk}>_p)}$. Notably, if t is an even number, we have $z'_{i,F(0)} = 0$; otherwise, $z'_{i,F(0)} = z_{i,m/2}$. Then, $(\alpha Z_i)^{2^i}$ can be represented by

$$(\alpha Z_i)^{2^i} = z'_{i,F(0)} + \sum_{j=0}^{m-1} z'_{i,F(2^{<j-i>_m})}\alpha^{2^j}. \qquad (13)$$

According to Eq.(13), each coefficient of the product $C = c_0\alpha + c_1\alpha^2 + \ldots + c_{m-1}\alpha^{2^{m-1}}$ can be obtained by

$$c_j = \begin{cases} \sum_{i=0}^{m-1} z'_{i,F(2^{<j-i>_m})}, & \text{if } t \text{ is even}, \\ \sum_{i=0}^{m-1} z'_{i,F(2^{<j-i>_m})} + \sum_{i=0}^{m-1} z'_{i,F(0)}, & \text{if } t \text{ is odd}. \end{cases} \qquad (14)$$

In the following example, we use the result of αA in Example 1, i.e., $\alpha A = (a_1,\ a_0 + a_3,\ a_3 + a_4,\ a_2 + a_1,\ a_2 + a_4)$, to construct $(\alpha Z_i)^{2^i}$ and the summation unit in GF(2^5) with t=2.

Example 2 Assume that $A = a_0\alpha + a_1\alpha^2 + a_2\alpha^{2^2} + a_3\alpha^{2^3} + a_4\alpha^{2^4}$ and $B = b_0\alpha + b_1\alpha^2 + b_2\alpha^{2^2} + b_3\alpha^{2^3} + b_4\alpha^{2^4}$ are two elements in GF(2^5) with t=2. Let $C = c_0\alpha + c_1\alpha^2 + c_2\alpha^{2^2} + c_3\alpha^{2^3} + c_4\alpha^{2^4}$ denote the product of A and B. The product $C=AB$ is calculated as follows.

First, we use the Z_i function decision table (Table 1) and the triangular MUX array to obtain Z_i as in Eq.(15).

$$\begin{bmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \end{bmatrix} = \begin{bmatrix} a_0b_0 & a_0b_1 + a_1b_0 & a_0b_2 + a_2b_0 & a_0b_3 + a_3b_0 & a_0b_4 + a_4b_0 \\ a_1b_1 & a_1b_2 + a_2b_1 & a_1b_3 + a_3b_1 & a_1b_4 + a_4b_1 & 0 \\ a_2b_2 & a_2b_3 + a_3b_2 & a_2b_4 + a_4b_2 & 0 & 0 \\ a_3b_3 & a_3b_4 + a_4b_3 & 0 & 0 & 0 \\ a_4b_4 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha^2 \\ \alpha^{2^2} \\ \alpha^{2^3} \\ \alpha^{2^4} \end{bmatrix} = \begin{bmatrix} z_{0,0} & z_{0,1} & z_{0,2} & z_{0,3} & z_{0,4} \\ z_{1,0} & z_{1,1} & z_{1,2} & z_{1,3} & 0 \\ z_{2,0} & z_{2,1} & z_{2,2} & 0 & 0 \\ z_{3,0} & z_{3,1} & 0 & 0 & 0 \\ z_{4,0} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha^2 \\ \alpha^{2^2} \\ \alpha^{2^3} \\ \alpha^{2^4} \end{bmatrix}. \qquad (15)$$

We refer to the result of Example 1 for computing αA; αZ_i for $0\le i\le 4$ can be obtained as

$$\begin{bmatrix}\alpha Z_0\\ \alpha Z_1\\ \alpha Z_2\\ \alpha Z_3\\ \alpha Z_4\end{bmatrix}=\begin{bmatrix} z_{0,1} & z_{0,3}+z_{0,0} & z_{0,3}+z_{0,4} & z_{0,2}+z_{0,1} & z_{0,2}+z_{0,4}\\ z_{1,1} & z_{1,3}+z_{1,0} & z_{1,3} & z_{1,2}+z_{1,1} & z_{1,2}\\ z_{2,1} & z_{2,0} & 0 & z_{2,2}+z_{2,1} & z_{2,2}\\ z_{3,1} & z_{3,0} & 0 & z_{3,1} & 0\\ 0 & z_{4,0} & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \alpha^2\\ \alpha^{2^2}\\ \alpha^{2^3}\\ \alpha^{2^4}\end{bmatrix}. \tag{16}$$

Since $(\alpha Z_i)^{2^i}$ is αZ_i with shifted i positions, Eq.(16) can be translated to the following matrix:

$$\begin{bmatrix}\alpha Z_0\\ (\alpha Z_1)^2\\ (\alpha Z_2)^{2^2}\\ (\alpha Z_3)^{2^3}\\ (\alpha Z_4)^{2^4}\end{bmatrix}=\begin{bmatrix} z_{0,1} & z_{0,3}+z_{0,0} & z_{0,3}+z_{0,4} & z_{0,2}+z_{0,1} & z_{0,2}+z_{0,4}\\ z_{1,2} & z_{1,1} & z_{1,3}+z_{1,0} & z_{1,3} & z_{1,2}+z_{1,1}\\ z_{2,2}+z_{2,1} & z_{2,2} & z_{2,1} & z_{2,0} & 0\\ 0 & z_{3,1} & 0 & z_{3,1} & z_{3,0}\\ z_{4,0} & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \alpha^2\\ \alpha^{2^2}\\ \alpha^{2^3}\\ \alpha^{2^4}\end{bmatrix}=\begin{bmatrix} z'_{0,0} & z'_{0,1} & z'_{0,2} & z'_{0,3} & z'_{0,4}\\ z'_{1,0} & z'_{1,1} & z'_{1,2} & z'_{1,3} & z'_{1,4}\\ z'_{2,0} & z'_{2,1} & z'_{2,2} & z'_{2,3} & 0\\ 0 & z'_{3,1} & 0 & z'_{3,3} & z'_{3,4}\\ z'_{4,0} & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \alpha^2\\ \alpha^{2^2}\\ \alpha^{2^3}\\ \alpha^{2^4}\end{bmatrix}. \tag{17}$$

Thus, based on Eq.(8), the product $C=c_0\alpha+c_1\alpha^2+c_2\alpha^{2^2}+c_3\alpha^{2^3}+c_4\alpha^{2^4}$ can be obtained from

$$\begin{cases} c_0=z'_{0,0}+z'_{1,0}+z'_{2,0}+z'_{4,0},\\ c_1=z'_{0,1}+z'_{1,1}+z'_{2,1}+z'_{3,1},\\ c_2=z'_{0,2}+z'_{1,2}+z'_{2,2},\\ c_3=z'_{0,3}+z'_{1,3}+z'_{2,3}+z'_{3,3},\\ c_4=z'_{0,4}+z'_{1,4}+z'_{3,4}. \end{cases} \tag{18}$$

As mentioned above, the $(\alpha Z_i)^{2^i}$ unit is translated from Eq.(15) into Eq.(17), which requires seven XOR gates and one XOR gate delay. The summation unit uses an XOR-tree to implement Eq.(18).

Analysis of the time and space complexity

We analyzed our proposed multiplier in terms of time and space complexity, and compared it to other bit-parallel multipliers. The architecture includes the addition unit, the $(\alpha Z_i)^{2^i}$ unit, the MUX array, and the summation unit. The addition unit involves $B_0+A_1^2$ computations, which requires $m-1$ XOR gates. The MUX array shown in Fig.2 includes a total of $m(m-1)/2$ MUXes, each with four inputs and one output. For the type-2 NB multiplier (Table 2), the $(\alpha Z_i)^{2^i}$ unit requires $(m^2-m+8)/4$ XOR gates, and the summation unit requires $m(m-1)/2$ XOR gates. The total size complexity is $(3m^2-m+6)/4$ XOR gates and $m(m-1)/2$ MUXes. From Tables 2~5, the total delay is $2T_X+T_{MUX}+\lceil\log_2(m-1)\rceil T_X$. Moreover, for the type-1 NB (Table 3), the major operation of the $(\alpha Z_i)^{2^i}$ unit is to form the structure of Eq.(13), which is a simple permutation due to $\gamma^{m+1}=1$. Thus, the $(\alpha Z_i)^{2^i}$ unit is without hardware complexity. The summation unit demands $m(m-1)/2$ XOR gates. Therefore, the total size complexity is $(m^2+3m-2)/2$ XOR gates and $m(m-1)/2$ MUX.

Table 2 Analysis of the space and time complexity for the type-2 NB multiplier

Unit	Space complexity		Time delay
	#XOR	#MUX$_{4\times1}$	
Addition	$m-1$	0	T_X
MUX array	0	$m(m-1)/2$	T_{MUX}
$(\alpha Z_i)^{2^i}$	$(m^2-m+8)/4$	0	T_X
Summation	$m(m-1)/2$	0	$\lceil\log_2(m-1)\rceil T_X$
Total	$(3m^2-m+6)/4$	$m(m-1)/2$	$T_{MUX}+(2+\lceil\log_2(m-1)\rceil)T_X$

Table 3 Analysis of the space and time complexity for the proposed type-1 multiplier

Unit	Space complexity		Time delay
	#XOR	#MUX$_{4\times1}$	
Addition	$m-1$	0	T_X
MUX array	0	$m(m-1)/2$	T_{MUX}
$(\alpha Z_i)^{2^i}$	0	0	0
Summation	$(m^2+m)/2$	0	$(1+\lceil\log_2(m-1)\rceil)T_X$
Total	$(m^2+3m-2)/2$	$m(m-1)/2$	$T_{MUX}+(2+\lceil\log_2(m-1)\rceil)T_X$

For the normal basis of GF(2^m), Koc and Sunar (1998) showed that normal basis multiplication can use palindromic representation to construct a type-2 normal basis multiplier. Later, the best existing bit-parallel multipliers use a type-2 normal basis (Sunar and Koc, 2001; Elia and Leone, 2002; Reyhani-Masoleh and Hasan, 2002; Kim *et al*., 2006). The transistor count was based on standard CMOS VLSI, in which a 2-input XOR gate, a 2-input AND gate, a 4×1 MUX, and a 1-bit latch require 6, 6, 16, and 8 transistors, respectively (Pekmestzi, 1999). Tables 4 and 5 show a comparison of the total number of transistors and the time delay for various bit-parallel multipliers. Tables 6 and 7 show the space complexities of various bit-parallel multipliers for different m values. Table 6 shows that the proposed multiplier saves approximately 16% space complexity compared to various type-2 NB multipliers. Moreover, Table 7 indicates that for type-1 NB, our architecture can reduce space complexity by about 8% (Hasan *et al*., 1992; 1993; Koc and Sunar, 1998). Therefore, the proposed architecture requires very few transistors, but the latency is approximately equivalent to that of existing NB multipliers (Massey and Omura, 1986; Sunar and Koc, 2001; Reyhani-Masoleh and Hasan, 2002).

Table 4 Comparison of the total number of transistors and the time delay of various multipliers for a type-2 NB

Multiplier	Transistor count	Time delay
Sunar and Koc, 2001	$15m^2-9m$	$T_A+(1+\lceil\log_2(m-1)\rceil)T_X$
Reyhani-Masoleh and Hasan, 2002	$15m^2-9m$	$T_A+(1+\lceil\log_2(m-1)\rceil)T_X$
Elia and Leone, 2002	$15m^2-9m$	$T_A+(1+\lceil\log_2(m-1)\rceil)T_X$
Kim *et al*., 2006	$15m^2-9m$	$T_A+(1+\lceil\log_2(m-1)\rceil)T_X$
This study	$(25m^2-13m+12)/2$	$T_{MUX}+(2+\lceil\log_2(m-1)\rceil)T_X$

Table 5 Comparison of the total number of transistors and the time delay of various multipliers for a type-1 NB

Multiplier	Transistor count	Critical time delay
Hasan *et al*., 1993	$12m^2-6m$	$T_A+(1+\lceil\log_2 m\rceil)T_X$
Koc and Sunar, 1998	$12m^2-6m$	$T_A+(2+\lceil\log_2 m\rceil)T_X$
Wu, 1998	$12m^2+18m+6$	$T_A+\lceil\log_2(m+1)\rceil T_X$
This study	$11m^2+m-6$	$T_{MUX}+(2+\lceil\log_2(m-1)\rceil)T_X$

Table 6 Comparison of space complexity for various bit-parallel multipliers in GF(2^m) using a type-2 NB with different m values

Multiplier	Transistor count				
	m=173	194	233	239	293
Sunar and Koc, 2001	447378	562794	812238	854664	1285098
Reyhani-Masoleh and Hasan, 2002	447378	562794	812238	854664	1285098
Elia and Leone, 2002	447378	562794	812238	854664	1285098
Kim *et al*., 2006	447378	562794	812238	854664	1285098
Presented here	372994	469195	677104	712465	1071214
Space reduction	16.62%	16.63%	16.63%	16.63%	16.64%

Table 7 Comparison of space complexity for various bit-parallel multipliers in GF(2^m) using a type-1 NB with different m values

Multiplier	Transistor count				
	m=162	196	210	268	293
Hasan *et al*., 1993	314922	460986	529194	861882	1023162
Koc and Sunar, 1998	314922	460986	529194	861882	1023162
Wu, 1998	317850	464526	532986	866718	1028430
Presented here	288840	422766	485304	790326	938190
Space reduction*	8.28%	8.29%	8.29%	8.30%	8.30%

* Compared with the multiplier proposed by (Hasan *et al*., 1993; Koc and Sunar, 1998)

CONCLUSION

In this paper, we present a new NB multiplication algorithm based on divide-and-conquer and uniform shift methods to implement an efficient multiplexer-based architecture. The proposed structure is suitable for all types of NB multipliers.

Moreover, it was demonstrated that the proposed multiplier can achieve lower space complexity than existing multipliers, due to the reliance of conventional NB multipliers on AND and XOR gates. Compared to other type-2 NB multipliers (Sunar and Koc, 2001; Elia and Leone, 2002; Reyhani-Masoleh and Hasan, 2002; Kim *et al*., 2006), our proposed architecture using MUX and XOR saves about 16% space complexity; compared to various bit-parallel multipliers for type-1 NB (Hasan *et al*., 1992; 1993; Koc and Sunar, 1998) of GF(2^m), our proposed architecture saves about 8% space complexity. The multiplier architecture suggested here is well suited for elliptic-curve cryptography systems and will be especially useful in specialized applications such as smart cards, mobile phones, and other portable devices, in which space constraints are critical.

References

Agnew, G.B., Mullin, R.C., Onyszchuk, I.M., Vanstone, S.A., 1991. An implementation for a fast public-key cryptosystem. *J. Cryptol*., **3**:63-79. [doi:10.1007/BF00196789]

Ash, D.W., Blake, I.F., Vanstone, S.A., 1989. Low complexity normal basis. *Discr. Appl. Math*., **25**(3):191-210. [doi:10.1016/0166-218X(89)90001-2]

Byun, G.Y., Kim, H.S., 2003. Low-complexity multiplexer-based multiplier of GF(2^m). *IEICE Trans. Inf. Syst*., **E86-D**(12):2684-2690.

Elia, M., Leone, M., 2002. On the inherence space complexity of fast parallel multipliers for GF(2^m). *IEEE Trans. Comput.*, **51**(3):346-351. [doi:10.1109/12.990131]

Hasan, M.A., Wang, M., Bhargava, V.K., 1992. Modular construction of low complexity parallel multipliers for a class of finite fields GF(2^m). *IEEE Trans. Comput.*, **41**(8):962-971. [doi:10.1109/12.156539]

Hasan, M.A., Wang, M.Z., Bhargava, V.K., 1993. A modified Massey-Omura parallel multiplier for a class of finite fields. *IEEE Trans. Comput*., **42**(10):1278-1280. [doi:10.1109/12.257715]

IEEE Std. 1363, 2000. IEEE Standard Specifications for Public Key Cryptography. IEEE-SA Standards Board, IEEE/IEE Electronic Library.

Kim, C.H., Kim, Y., Ji, S.Y., Park, I.W., 2006. A New Parallel Multiplier for Type II Optimal Normal Basis. Int. Conf. on Computational Intelligence and Security, **2**:1315-1318. [doi:10.1109/ICCIAS.2006.295271]

Koc, C.K., Sunar, B., 1998. Low-complexity bit-parallel canonical and normal multipliers for a class of finite fields. *IEEE Trans. Comput*., **47**(3):353-356. [doi:10.1109/12.660172]

Lee, C.Y., Lu, E.H., Lee, J.Y., 2001. Bit-parallel systolic multipliers for GF(2^m) fields defined by all-one and equally-spaced polynomials. *IEEE Trans. Comput*., **50**(5):385-393. [doi:10.1109/12.926154]

Lee, C.Y., Chiou, C.W., Lin, J.M., 2005. Low-complexity bit-parallel dual basis multipliers using the modified Booth's algorithm. *Comput. Electr. Eng*., **31**(7):444-459. [doi:10.1016/j.compeleceng.2005.09.002]

Lee, C.Y., Chiu, Y.H., Chiou, C.W., 2006. New Bit-parallel Systolic Multiplier Over GF(2^m) Using the Modified Booth's Algorithm. IEEE Asia Pacific Conf. on Circuits and Systems, p.610-613. [doi:10.1109/APCCAS.2006.342062]

Lidl, R., Niederreiter, H., 1994. Introduction to Finite Fields and Their Applications. Cambridge University Press, New York.

MacWilliams, F.J., Sloane, N.J.A., 1977. The Theory of Error-correcting Codes. Amsterdam, North-Holland.

Massey, J.L., Omura, J.K., 1986. Computational Method and Apparatus for Finite Field Arithmetic. US Patent, No. 4587627.

Mullin, R.C., Onyszchuk, I.M., Vanstone, S.A., Wilson, R.M., 1989. Optimal normal basis in GF(p^n). *Discr. Appl. Math*., **22**(2):149-161. [doi:10.1016/0166-218X(88)90090-X]

Pekmestzi, K.Z., 1999. Multiplexer-based array multipliers. *IEEE Trans. Comput.*, **48**(1):15-23. [doi:10.1109/12.743408]

Reyhani-Masoleh, A., 2006. Efficient algorithms and architectures for field multiplication using Gaussian normal bases. *IEEE Trans. Comput.*, **55**(1):34-47. [doi:10.1109/TC.2006.10]

Reyhani-Masoleh, A., Hasan, M.A., 2002. A new construction of Massey-Omura parallel multiplier over GF(2^m). *IEEE Trans. Comput.*, **51**(5):511-520. [doi:10.1109/TC.2002.1004590]

Reyhani-Masoleh, A., Hasan, M.A., 2003. Fast normal basis multiplication using general purpose processors. *IEEE Trans. Comput.*, **52**(11):1379-1390. [doi:10.1109/TC.2003.1244936]

Reyhani-Masoleh, A., Hasan, M.A., 2005. Low complexity word-level sequential normal basis multiplier. *IEEE Trans. Comput.*, **54**(2):98-110. [doi:10.1109/TC.2005.29]

Sunar, B., Koc, C.K., 2001. An efficient optimal normal basis type II multiplier. *IEEE Trans. Comput.*, **50**(1):83-88. [doi:10.1109/12.902754]

Wu, H., 1998. Efficient Computations in Finite Fields with Cryptographic Significance. PhD Thesis, Waterloo University, Ontario.