Parallel Algorithm for Polynomial Basis Multiplier in GF(2^m) Fields

Che-Wun Chiou1* and Huey-Lin Jeng2

1Department of Computer Science and Information Engineering, Ching Yun University, Chung-Li, Taiwan 320, R.O.C.
2Information Planning Division, Information Technology Development Department, Hua Nan Commercial Bank, LTD, Taipei, Taiwan 100, R.O.C.

Abstract

Fast multiplication in a finite field GF(2^m) is a basis step in communications engineering applications, such as error-correcting codes or cryptograph algorithms. A new parallel algorithm on the polynomial basis bit-parallel multiplier is presented. This new parallel algorithm saves about 25% execution time while comparing with the conventional algorithms. The hardware version for the proposed parallel algorithm is also invented. The new hardware structure requires only the space complexity of O(m) while existing multipliers need the space complexity of O(m^2). The time complexity of the proposed multiplier takes only about half of the time complexity of the existing Lee’s multiplier.

Key Words: Cryptography, Finite Field Arithmetic, Parallel Algorithm, Multiplication, Array Multiplier

1. Introduction

The arithmetic operations in the finite fields GF(2^m) play an increasingly important role in error-correcting code [1], public-key cryptography [2], digital signal processing [3,4], switching theory [5], and pseudorandom number generator [6]. In GF(2^m), multiplication is a basic operation which is much slower than addition or squaring and other complex operations such as exponentiation and multiplicative inversion can be carried out by repeated multiplications. For this reason the realization of multiplication operations in finite fields has received wide attention, and numerous multiplication algorithms have already been proposed with the aim of achieving even smaller computing-time delays and space complexity [7–35].

Multipliers can be classified into bit serial and bit parallel architectures. Fast multiplication operations over finite fields are essential in a great variety of cryptosystems and error-correcting code applications. Bit-parallel multipliers tend to be faster, which makes them attractive for many applications. Hence, the proposed parallel multiplication algorithm is based on the bit-parallel multipliers. The performance of the multiplication operations is closely related to the representation of the field elements. There are three different popular basis representations for elements of GF(2^m), normal basis [7–12], dual basis [12–15], and polynomial (or standard, canonical) basis [16–35]. The original normal basis multiplication algorithm was invented by Massey and Omura [7]. The major advantage of the normal basis representation is that the squaring of an element in GF(2^m) is readily shown to be a simple cycle shift of its binary digits. Thus, the normal basis multipliers are very effective for performing inverse, squaring, and exponentiation operations. Recently, many efficient algorithms based on the Massey-Omura algorithm have been presented [8,9]. In general, the normal basis multipliers have a space complexity of O(m^2) and take the time complexity of O(log m). Berlekamp [13] has developed a bit-serial multiplication algorithm over GF(2^m) for encoding of Reed-Solomon codes. Berlekamp’s algorithm to multi-
ply two elements requires the representation of the multiplier by a canonical basis and the multiplicand by the corresponding dual basis and the product is obtained in the dual basis. Wu et al. [14] and Wu and Hasan [15] also adopted a weakly dual basis multiplier. The dual basis multiplier needs the least number of gates which in turn leads to the smallest area required for VLSI implementation. However, the former two types of finite field multipliers require basis conversion, while the polynomial basis multiplier does not. Hence, the polynomial basis multiplier is readily matched to any input or output system. The earliest parallel polynomial basis multiplier was suggested by Bartee and Schneider [16]. Recently, several bit-parallel polynomial basis multipliers have been proposed and are suitable for VLSI implementation by using all-one polynomials and equally-spaced polynomials [27–29], and composite fields [30,31]. The polynomial basis architecture has a lower design complexity and its size is easier to extend to meet various applications. Polynomial basis multipliers have a hardware complexity of $O(m^2)$ and a time complexity of $O(\log_2 m)$. However, due to XOR tree structure, most existing polynomial basis multipliers are not regular and then are not suited to VLSI implementation. To alleviate this problem, Lee [25] has adopted a new systolic array multiplier using trinomial, and its regular and modular architecture makes it attractive to VLSI implementation. The Lee’s polynomial basis multiplier requires a space complexity of $O(m^2)$ and a time complexity of $O(m)$. In this paper, a new parallel algorithm based on the polynomial basis bit-parallel multiplier using general irreducible polynomials is invented. The parallel algorithm saves about 25% execution time while comparing with existing conventional algorithms. A hardware version of the proposed parallel algorithm is also included. The new multiplier requires a space complexity of $O(m)$ and takes only half of time complexity of the Lee’s multiplier.

2. Preliminaries

It is assumed that the reader is familiar with the basic concepts of finite fields. The properties of finite fields are covered in detail in [1,2]. A finite field, $\text{GF}(2^m)$, is defined by an irreducible polynomial, $P(X)$, of degree $m$ and each element in the field can be represented as a polynomial of degree $(m-1)$. Let $A(X)$ and $B(X)$ be two elements in $\text{GF}(2^m)$ that each element is generated by $P(X)$, where

\[
A(X) = a_0 + a_1X^1 + a_2X^2 + \ldots + a_{m-1}X^{m-1},
\]

\[
B(X) = b_0 + b_1X^1 + b_2X^2 + \ldots + b_{m-1}X^{m-1},
\]

\[
P(X) = p_0 + p_1X^1 + p_2X^2 + \ldots + p_{m-1}X^{m-1} + X^m,
\]

where the coefficients $a_i$, $b_i$, and $p_i$ are the binary digits 0 and 1 and $p_0 = 1$ for $0 \leq j \leq m-1$, and $1 \leq i \leq m-1$. The product $A(X)B(X) \mod P(X)$ can be represented as

\[
C(X) = (A(X)B(X)) \mod P(X)
\]

\[
= c_0 + c_1X^1 + c_2X^2 + \ldots + c_{m-1}X^{m-1},
\]

where $c_j = 0$ or 1 for all $j$, $0 \leq j \leq m-1$. (1)

3. A Parallel Algorithm for Polynomial Basis Array Multiplier in $\text{GF}(2^m)$

We assume that $m$ is an even number in this paper. The results are easily extended to the case that $m$ is odd. Using Horner’s rule, the product $C(X) = A(X)B(X) \mod P(X)$ can be obtained by

\[
C(X) = (A(X)B(X)) \mod P(X)
\]

\[
= a_0B_0(X) + a_1X^1B_1(X) + a_2X^2B_2(X) + \ldots
+ a_{m-1}X^{m-1}B_{m-1}(X).
\]

(2)

We now let

\[
B_i(X) = X^iB_i(X).
\]

(3)

Eq. (2) becomes

\[
C(X) = a_0B_0(X) + a_1B_1(X) + a_2B_2(X) + \ldots
+ a_{m-1}B_{m-1}(X).
\]

(4)

All $B_i(X)$s can be partitioned into 2 groups, even group with $i$ being an even number and odd group with $i$ being an odd number. Thus, we may rearrange Eq. (4) to yield

\[
C(X) = (a_0B_0(X) + a_2B_2(X) + a_4B_4(X) + \ldots
+ a_{m-2}B_{m-2}(X)) + (a_1B_1(X) + a_3B_3(X)
+ a_5B_5(X) + \ldots + a_{m-1}B_{m-1}(X)).
\]

(5)

According to Eq. (3), the relation of $B_i(X)$ and $B_{i\pm2}(X)$ is given by

\[
B_i(X) = X^2B_{i\pm2}(X).
\]

(6)

Thus, using Eq. (6), $B_i(X)$ can be computed recursively as follows:
\[ B_i(X) = X^2B_{i-1}(X) + X^3B_{i-2}(X) + \ldots + X^{m-1}B_{i-m}(X) \]

(7)

For example, \( B_0(X) \) and \( B_1(X) \) are computed by

\[
B_0(X) = X^2(B_0(X)) = X^2(X^2B_0(X)) = X^2(X^2(B_0(X))) \]

\[
B_1(X) = X^2(B_1(X)) = X^2(X^2(B_0(X))) = X^2(B_0(X)) \]

If values of both \( B_0(X) \) and \( B_1(X) \) have been known, \( B_0(X) \) and \( B_1(X) \) can be concurrently computed. Let \( B_i(X) = b_{i,0}X^0 + b_{i,1}X^1 + \ldots + b_{i,m-1}X^{m-1} \), where \( b_{i,j} \) is a binary digit 0 or 1 for all \( i \) and \( j \), \( 0 \leq i \leq m-1 \) and \( 0 \leq j \leq m-1 \). Suppose that \( B_i(X) \) has been pre-computed by use of the coefficients of \( B_0(X) \) as

\[
B_i(X) = b_{i,0} + b_{i,1}X + b_{i,2}X^2 + \ldots + b_{i,m-1}X^{m-1} \]

\[
= X^iB_0(X) = X^i(b_{0,0} + b_{0,1}X + b_{0,2}X^2 + \ldots + b_{0,m-1}X^{m-1}) \]

\[
= (b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \cdot X^i = (b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \cdot X^i \]

\[
= (b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \cdot (1 + p_1X^1 + \ldots + p_{m-1}X^{m-1}) \]

\[
= (b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \cdot X^i \]

(8)

where the operation \( \oplus \) denotes the logical XOR operation. The mod-2 summation is equal to the logical XOR operation. Based on the well-known coefficients of \( B_0(X) \), the coefficients of \( B_1(X) \) are computed by

\[ b_{1,0} = b_{0,m-1}, \text{ and for all } j, 1 \leq j \leq m-1 \]

\[ b_{1,j} = b_{0,j} \oplus b_{0,j} \cdot p_j. \]

(9)

Similarly, if the coefficients of \( B_2(X) \) have been known, the coefficients of \( B_3(X) \) are given by

\[ B_i(X) = b_{i,0} + b_{i,1}X + b_{i,2}X^2 + b_{i,3}X^3 + \ldots + b_{i,m-1}X^{m-1} \]

\[
= X^iB_0(X) = X^i(b_{0,0} + b_{0,1}X + b_{0,2}X^2 + \ldots + b_{0,m-1}X^{m-1}) \]

\[
= X^i(b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \]

\[
= X^i(b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \cdot (1 + p_1X^1 + \ldots + p_{m-1}X^{m-1}) \]

\[
= X^i(b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \cdot X^i \]

\[
= (b_{0,0},X + b_{0,1}X^2 + \ldots + b_{0,m-1}X^{m-1}) \cdot X^i \]

(10)

The relations between the coefficients of \( B_1(X) \) and the coefficients of \( B_2(X) \) are shown as follows:

\[ b_{1,0} = b_{2,0} \oplus b_{2,1} \cdot p_{m-1}, \]

\[ b_{1,1} = b_{2,0} \oplus b_{2,1} \cdot p_{m-1} \cdot p_1, \text{ and for all } j, \]

\[ 2 \leq j \leq m-1, \]

\[ b_{1,j} = b_{2,0} \oplus b_{2,1} \cdot p_{j-1} + b_{2,2} \cdot p_{j-1} \cdot p_{m-1} \cdot p_j. \]

(11)

The coefficients of \( C(X) \) can be obtained by summing up the corresponding coefficients of \( B_i(X) \) for all \( i, 0 \leq i \leq m-1 \) by

\[ c_i = a_i \oplus b_{i,0} \oplus a_i b_{i,1} \oplus a_i b_{i,2} \oplus \ldots \oplus a_i b_{i,m-1}, \text{ for all } i, 0 \leq i \leq m-1. \]

As aforementioned, both \( B_0(X) \) and \( B_1(X) \) have firstly been obtained, and then \( B_2(X) \) and \( B_3(X) \) in Eq. (5) can be concurrently computed. As \( B_2(X) \) and \( B_3(X) \) have been computed, then both \( B_4(X) \) and \( B_5(X) \) can be computed in parallel, and so on. The traditional bit-parallel algorithm for \( C(X) = A(X) \cdot B(X) \mod P(X) \) is described in Algorithm-A and our proposed parallel algorithm based on Eq. (5), Eq. (8), and Eq. (10) is depicted in Algorithm-B.

Algorithm-A:

// Let \( A(X) = a_0 + a_1X + a_2X^2 + \ldots + a_{m-1}X^{m-1}, \)

\[ B(X) = b_0 + b_1X^1 + b_2X^2 + \ldots + b_{m-1}X^{m-1}, \]

// \( C(X) = c_0 + c_1X^1 + c_2X^2 + \ldots + c_{m-1}X^{m-1}, \)

\[ P(X) = p_0 + p_1X^1 + p_2X^2 + \ldots + p_{m-1}X^{m-1}, \]

// \( p_0 = 1, \) and \( C(X) = A(X) \cdot B(X) \mod P(X). \)
/* Three arrays A[0..m-1], B[0..m-1,0..m-1], and P[0..m-1] are initially loaded */
/* with following values: A[i] = a_i, B[0,i] = b_i, P[i] = p_i, and C[i] = 0 for all i, */
/* 0 ≤ i ≤ m-1. The final result C is obtained by */
/* c_i = C[i] for all i, 0 ≤ i ≤ m-1. */

begin
for i = 0 to m-1 do
    begin
        for j = 0 to m-1 do
            parallel begin
            parallel end;
        parallel begin
            if i = m-1 do
                parallel begin
                    for k = 1 to m-1 do
                        parallel begin
                            B[i+1,k] := B[i,k-1] XOR (B[i,m-1] AND P[k]);
                        parallel end
                    parallel end
                parallel end
            end;
        parallel end;
    end;
end;

Algorithm-B:
/* Let A(X) = a_0 + a_1 X^1 + a_2 X^2 + ... + a_{m-1} X^{m-1}, */
/* B(X) = b_0 + b_1 X^1 + b_2 X^2 + ... + b_{m-1} X^{m-1}, */
/* C(X) = c_0 + c_1 X^1 + c_2 X^2 + ... + c_{m-1} X^{m-1}, */
/* P(X) = p_0 + p_1 X^1 + p_2 X^2 + ... + p_{m-1} X^{m-1} + X^m, */
/* P_0 = 1, and C(X) = A(X)*B(X) mod P(X). */
/* Six arrays A[0..m-1], B[0..m-1,0..m-1], */
/* C[0..m-1], P[0..m-1], C0[0..m-1], */
/* and C1[0..m-1] are initially loaded with */
/* following values: */
/* A[i] = a_i, B[0,i] = b_i, P[i] = p_i, C[i] = 0, C0[i] = 0, */
/* and C1[i] = 0 for all i, 0 ≤ i ≤ m-1. */
/* The final result C is obtained as follows: */
/* c_i = C[i] = C0[i] XOR C1[i] for all i, 0 ≤ i ≤ m-1. */
begin
    parallel begin
        for k = 1 to m-1 do
            parallel begin
                B[1,k] := B[0,k-1] XOR (B[0,m-1] AND P[k]);
            parallel end;
        B[1,0] := B[0,m-1];
        parallel end;
    end;

for i = 0 to m-1 step 2 do
    begin
        for j = 0 to m-1 do
            parallel begin
                C0[j] := C0[j] XOR (A[i] AND B[i,j]);
                C1[j] := C1[j] XOR (A[i+1] AND B[i+1,j]);
            parallel end;
        parallel begin
            B[i+2,0] := B[i,m-2] XOR (B[i,m-1] AND P[m-1]);
            B[i+2,1] := B[i,m-1] XOR ((B[i,m-2] AND P[m-1]) AND P[1]);
            B[i+3,0] := B[i+1,m-2] XOR (B[i+1,m-1] AND P[m-1]);
            B[i+3,1] := B[i+1,m-1] XOR ((B[i+1,m-2] AND P[m-1]) AND P[1]);
        parallel end;
    parallel end;
end;

The traditional Algorithm-A takes an execution time of 2m logical-XOR operations and 2m logical-AND operations. The proposed Algorithm-B executes two operations concurrently, thus it takes an execution time of ((3m/2)+2) logical-XOR operations and ((3m/2)+1) logical-AND operations. Our parallel algorithm saves about 25% execution time while comparing with the traditional algorithm.

4. Hardware Structure

The proposed parallel algorithm is very suitable for VLSI implementation. A hardware version of the proposed parallel algorithm will be described in the follow-
(T) at the clock cycle t. Both shift registers S and T are shown in Figure 2.

Algorithm-C:

Step 1: R is initially loaded with the following values,
\[ R_i(0) = b_i \text{ for all } i, 0 \leq i \leq m-1. \]

Step 2: R will be shifted left one bit after one clock cycle and has the content \( R_i(1) \) for all \( i, 0 \leq i \leq m-1. \)

Step 3: H and K are initially loaded with the following values,
\[ H_i(0) = b_i \text{ for all } i, 0 \leq i \leq m-1, \text{ and} \]
\[ K_i(0) = R_i(1) \text{ for all } i, 0 \leq i \leq m-1. \]

Step 4: S and T are initially loaded as follows:
\[ S_i(0) = a_{2i}, \text{ for all } i, 0 \leq i \leq m/2-1, \text{ and} \]
\[ T_i(0) = a_{2i+1}, \text{ for all } i, 0 \leq i \leq m/2-1. \]

Step 5:

(1) As the next clock cycle has arrived, these shift registers and linear feedback shift registers are shifted one bit.

(2) The U and V cells perform the following functions,
\[ U_i(t+1) = H_i(t)S_0(t) \text{ for all } i, 0 \leq i \leq m-1, \]
\[ V_i(t+1) = K_i(t)T_0(t) \text{ for all } i, 0 \leq i \leq m-1. \]

Where \( U_i(t) \) and \( V_i(t) \) denote the contents of \( U_i \) and \( V_i \) at the clock cycle t, and their detail circuits are shown in Figure 3.

Step 6: Repeat Step 5 for each clock cycle and the results \( C_0 \) and \( C_1 \) are obtained after \( m/2-1 \) clock cycles.

Step 7: Using the XOR array, shown in Figure 4, to perform \( C := C_0 \oplus C_1. \)

Comparisons of various polynomial basis multipliers on the space and time complexity are listed in Table 1. The hardware structure of the proposed parallel algorithm requires a space complexity of \( O(m) \) and its time complexity is about half of the time complexity of the Lee’s multiplication algorithm in [25]. One interesting result is that the time complexity is drastically reduced to be one quarter of the time complexity of the Lee’s multiplication algorithm if \( p_{m-1} \) is equal to 0.

---

**Figure 1.** A linear feedback shift register for pre-computing \( B_1(X). \)
5. Conclusion

A new parallel algorithm based on the polynomial basis bit-parallel multiplier has been presented and the new parallel algorithm saves about 25% execution time.
while comparing with the conventional algorithms. A hardware version of the proposed parallel algorithm has also been proposed. The new hardware multiplier has the features of regularity, modularity, and concurrency and then is very suitable for VLSI implementation. The space complexity of the proposed hardware multiplier is $O(m)$ while the existing polynomial basis multipliers are $O(m^2)$. The time complexity of the proposed multiplier takes only about half of the time complexity of the existing Lee’s multiplier in [25].

### References


Manuscript Received: Mar. 11, 2007
Accepted: Aug. 2, 2007