Efficient low-complexity architectures for computing exponentiation

Dr. Chung-Hsin Liu

Chunghwa Telecom. Lab, Tao-Yuan, Taiwan, 326, R.O.C
E-mail: liu3@cht.com.tw

Abstract

In this investigation, we present cellular architecture for performing AB^2 multiplication in a class field $GF(2^m)$, where the definition by an irreducible polynomial for the field is an all-one polynomial (AOP). This multiplier is highly regular, modular, and thus suited to VLSI implementation. For finite field multiplication and exponentiation, we conclude that our proposed is more efficient as their basic cells have less computation time and circuit low-complexity. Furthermore, comparing the related cellular architecture reveal that our constructed multipliers are shorter than the conventional multipliers for per cell circuit complexity and computing delay time. In addition, based on pipeline architectures, we also produced to compute exponentiation.

Keywords: AOP, cellular architecture.

1 Introduction

Finite fields have many applications as algebraic constructions, such as coding theory [6,11] and cryptography. In general, the basic arithmetic operand over finite fields $GF(2^m)$ currently available require more computational time and more circuit complexity for addition, multiplication, and inversion implementation. As low-complexity and high-speed architectures have become increasingly attractive, finite field applications have increased. Hence, fast multiplication algorithms must be developed that have low circuit complexity. In VLSI architectures with concurrent, balanced with I/0, simple and regular designs have been widely implemented over finite fields. This investigation attempts to construct cellular architecture for low-complexity multipliers in a class of fields $GF(2^m)$.

An all-one polynomial (AOP) is utilized for irreducible polynomials to reduce the complexity of the field multiplication. Many architectures have been efficiently developed under various bases to construct low-complexity bit-parallel multiplication using irreducible AOPs. Itoh and Tsujii (1989)[2], in canonical basis, have been presented as a structure for a parallel multiplier over $GF(2^m)$. In order to improved the computation time, Koc and Sunor (1998) [4], based on normal basis, have designed low-complexity bit-parallel canonical basis multipliers that require m^2 AND gates and m^2-1 XOR gates. Wu and Hasan (1998) [5] recently presented low-complexity parallel multipliers using the weekly dual basis (WDB). However, the systems designed by previous studies are not suitable for implementing cellular architectures because they are not constructed with simple and regular cell structures.

In VLSI architectures, Laws in 1971[7] presented the first parallel-in-parallel-out multiplier of cellular-array architecture. This circuit requires 2m gate delays to compute multiplication. To reduce the computation time, Wang(1990) [9] modified the multiplier to contain two AND gates, one 3-input XOR gates, and seven latches to reduce the complexity of the circuit. Wei(1994) [3] also produced a power-sum circuit of systolic multipliers for computing AB^2+C, where A, B, and C are any element in $GF(2^m)$. In general, the computation time and circuit complexity need to be tradeoff against each other for example, the cellular multiplier has less the complexity of circuit and latency than the systolic-array multiplier.

Exponentiation can be implemented using read-only-memory (ROM) and consecutive multiplications. Several architectures for computing exponentiations over $GF(2^m)$ has been developed under the standard basis and the normal basis [9][10]. Although performing a simple arithmetic operation like addition is relatively straightforward, more complex operations such as multiplication and exponentiation are more difficult tasks to carry out efficiently. This is particularly true large number arithmetic is involved. In cryptography, many of the private and public-key algorithms which rely on computations in $GF(2^m)$ require large field sizes, some as high as $GF(2^{2000})$ [16], to achieve a high level of security. Hence, there is a need to develop effective algorithms for doing arithmetic operations in $GF(2^m)$ [9].

In this paper, based on an irreducible AOP, has been developed a novel bit-parallel multiplier of cellular architecture using our proposed inner-product multiplication algorithm. This multiplier is constructed by $(m+1)^2$ identical inner-product cells and 2m identical summation cells. Each inner-product cell consists of one 2-input AND gate and one 2-input XOR gate. Each of summation cell consist of one 2-input XOR gate. The time complexity of the presented multiplier only requires m+3 gates delay. Comparing the related cellular architectures reveals that our constructed multipliers are shorter than the conventional multipliers for computation time. In addition, we also produced computing exponentiation by using pipeline architectures with our proposed multiplier.

The rest of this paper is organized as follows. Section II briefly reviews the Itoh-Tsujii multiplier and definition all-one polynomial. A new bit-parallel multiplication algorithm is constructed in sections III. A new cellular bit-parallel AB^2 multiplier is presented in the Section IV. Section V proposed pipeline architectures for computing multiplicative exponentiation.

2 Background

In this Section important propertirs of AOP are outlined, the model used in this paper to describe the architecture is formulated, and the notations and definitions used are introduced.

Definition 1[2]: A polynomial $p(x)=p_0+p_1x+p_2x^2+...+p_mx^m$ over GF(2), if $p_0=p_1=...=p_m=1$, then the polynomial p(x) is called all one polynomial (AOP) of degree m.

Let $p(x)= 1+x+x^2+...+x^m$ over GF(2) is an irreducible AOP and α is a root of p(x), then any element in finite field $GF(2^m)$ can be represented the binary representation as $a = a_0 + a_1\alpha + a_2\alpha^2 +...+ a_{m-1}\alpha^{m-1}$, where $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$ is a canonical basis of $GF(2^m)$. An irreducible AOP with degree m have been suggested [2] for reducing the field multiplication complexity, where m+1 is a prime, i.e., $\alpha^{m+1}=1$. Any element in $GF(2^m)$ have an extended representation like $A = A_0 + A_1\alpha + A_2\alpha^2 +...+ A_m\alpha^m$, where "A" is denoted by the extended element. Namely, $A=A_0 + A_1\alpha+ A_2\alpha^2 +...+ A_m\alpha^m = a_0 + a_1\alpha + a_2\alpha^2 +...+ a_{m-1}\alpha^{m-1}$, where $A_i= a_i$, for i=0, 1, 2, ..., m-1, and $A_m =0$. Assume $\alpha^{m+1} +1$ takes as modulo polynomials, then the coefficients of extended element A multiplying by α, we can be obtained by a periodic shifting one bit to the right. For example, $A\alpha \pmod{\alpha^{m+1}+1}= A_m + A_0\alpha+ A_1\alpha^2 +...+ A_{m-1}\alpha^m$. This periodic shifting property is efficiently used to reduce the complexity of the field multiplication, and it is without performing modulo irreducible polynomials.

A common computation in an element A multiplied by α can be done as follows rule:

Let $A^{(1)}=A_m + A_0\alpha+ A_1\alpha^2+...+ A_{m-1}\alpha^m$,

(1)

where $A^{(1)}$ is called a periodic shift-right-by-one-bit, then

$A\alpha= A_0\alpha+A_{m-2}\alpha^2+\ldots+A_{m-1}\alpha^m+A_m\alpha^{m+1}$

$$A\alpha(\bmod\ \alpha^{m+1}+1)= A_m+A_0\alpha+A_1\alpha^2+\ldots+A_{m-1}\alpha^m =A^{(1)} \quad (2)$$

Similarly, the coefficients of element multiply by α^i can be derived cyclically operating periodic shift-right-by-i-bit operand, denoted by $A^{(i)}$. We have the recursive formula

$A^{(i)} = A_{<-i>}+A_{<1-i>}\alpha+A_{<2-i>}\alpha^2+\ldots+A_{<m-i>}\alpha^m$

$= A^{(i-1)}\alpha \pmod{\alpha^{m+1}+1}$ for $m \geq i \geq 1$ (3)

Where $<x>$ denotes x modulo m+1. On the other hand, from $\alpha^{m+1}=1$ can be obtained $\alpha^{m}=\alpha^{-1}$, hence, an element multiplied by α^{-1} can be done by the following rule:

Let $A^{(-1)} = A_1+A_2\alpha+A_3\alpha^2+\ldots+A_0\alpha^m \pmod{\alpha^{m+1}+1}$ (4)

be a periodic shifting-left-by-one-bit, then

$A\alpha^{-1} = A_0\alpha^{-1}+A_1+\ldots+A_{m-1}\alpha^{m-2}+A_m\alpha^{m-1}$

$A\alpha^{-1} \pmod{\alpha^{m+1}+1} = A_1+A_2\alpha+A_3\alpha^2+\ldots+A_0\alpha^m = A^{(-1)}$

(5)

Similarly, the coefficients of element multiplied by α^{-i} can be derived from cyclically shift-left-by-i-bit, denoted by $A^{(-i)}$. We have the recursive formula

$A^{(-i)} = A_{<i>}+A_{<1+i>}\alpha+A_{<2+i>}\alpha^2+\ldots+A_{<m+i>}\alpha^m$

$= A^{(-i+1)}\alpha \pmod{\alpha^{m+1}+1}$ for $m \geq i \geq 1$

(6)

The power-2 operation of elements in GF(2^m) can be extended representation as $C = A^2 = (A_0 + A_1 \alpha + A_2\alpha^2 + \ldots + A_m\alpha^m) = A_0 + A_1\alpha^2 + A_2\alpha^4 + \ldots + A_m\alpha^{2m} = C_0+C_1\alpha+C_2\alpha^2+\ldots+C_m\alpha^m$ (7)

Here $C_i = A_{i/2}$ (if i is even) or $A_{(i+m+1)/2}$ (if i is odd), hence the parallel squaring operation only needs to permute the coefficients of A, as Fig. 1 shows.

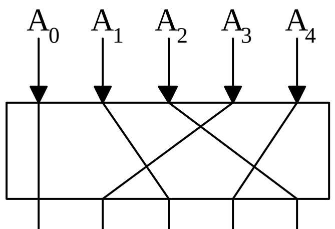


Fig.1: The parallel square unit over GF(2^4) based on AOP

A square element multiplied by α^2 can be done as following the rule:

Let $[A^2]^{(1)} = A_m+A_0\alpha^2+A_1\alpha^4+\ldots+A_{m-1}\alpha^{2m}$, (8)

where $[A^2]^{(1)}$ is called a period shifting-right-by-one-bit, then

$A^2\alpha^2 = A_0\alpha^2+A_1\alpha^4+A_2\alpha^6+\ldots+A_m\alpha^{2m+2}$

$A^2\alpha^2 \pmod{\alpha^{2m+2}+1} = A_m+A_0\alpha^2+A_1\alpha^4+\ldots+A_{m-1}\alpha^{2m} = [A^2]^{(1)}$ (9)

Similarly, the coefficients of square element multiplied by α^{2i} can be derived from cyclically shift-right-by-i-bit, denoted by $[A^2]^{(i)}$. We have the recursive formula

$$[A^2]^{(i)} = A_{-i \pmod{m+1}}+A_{1-i \pmod{m+1}}\alpha^2+A_{2-i \pmod{m+1}}\alpha^4+\ldots+A_{m-i \pmod{m+1}}\alpha^{2m}$$

$$= [A^2]^{(i-1)}\alpha^2 \pmod{\alpha^{2m+2}+1} \qquad \text{for } m\geq i \geq 1 \qquad (10)$$

From $\alpha^{m+1}=1$ can be obtained $\alpha^{2m}=\alpha^{-2}$, hence, a square element multiplied by α^{-2} can be done by the following rule:

Let $[A^2]^{(-1)} = A_1+A_2\alpha^2+\ldots+A_m\alpha^{2m-2}+A_0\alpha^{2m}$,

(11)

where $[A^2]^{(-1)}$ is called a period shifting-left-by-one-bit, then

$A^2\alpha^{-2} = A_0\alpha^{-2}+A_1+A_2\alpha^2+\ldots+A_m\alpha^{2m-2}$

$A^2\alpha^{-2} \pmod{\alpha^{2m+2}+1} = A_1+A_2\alpha^2+\ldots+A_m\alpha^{2m-2}+A_0\alpha^{2m} = [A^2]^{(-1)}$

(12)

Similarly, the coefficients of square element multiply by α^{-2i} can be derived from cyclically shift-left-by-i-bit, denoted by $[A^2]^{(-i)}$. We have the recursive formula

$$[A^2]^{(-i)} = A_{-i(\text{mod } m+1)}+A_{1-i(\text{mod } m+1)}\alpha^2+A_{2-i(\text{mod } m+1)}\alpha^4+\ldots+A_{m-i(\text{mod } m+1)}\alpha^{2m}$$

$$= [A^2]^{(-i+1)}\alpha^{-2} \pmod{\alpha^{2m+2}+1} \qquad \text{for } m\geq i \geq 1 \qquad (13)$$

Therefore, four results are obtained:

(a) An extended element multiplied by α^i is equal to a periodic shifting-right-by-i-bit operation.

(b) An extended element multiplied by α^{-i} is equivalent to a periodic shifting-left-by-i-bit operation.

(c) A square element multiplied by α^{2i} is equal to a periodic shifting-right-by-i-bit operation.

(d) A square element multiplied by α^{-2i} is equivalent to a periodic shifting-left-by-i-bit operation.

Based on above cyclic shifting operations, we will derive the multiplying AB^2 computation in next subsection.

3 New multiplication algorithm for computing AB^2

Let any $a,b\in GF(2^m)$ be given by

$a= a_0+a_1\alpha+a_2\alpha^2+\ldots+a_{m-1}\alpha^{m-1}$

$b=b_0+b_1\alpha+b_2\alpha^2+\ldots+b_{m-1}\alpha^{m-1}$

Where a_i, $b_i \in GF(2)$, $(1, \alpha, \alpha^2,\ldots, \alpha^{m-1})$ is a canonical basis. The product of a and b^2, in the canonical basis, is given by $c=ab^2$, where $c=c_0+c_1\alpha+c_2\alpha^2+\ldots+c_{m-1}\alpha^{m-1}$ which can be written as follows:

$$c = ab^2 = (AB^2 \pmod{\alpha^{m+1}+1}) \bmod p(\alpha)$$

$$= C \pmod{p(\alpha)} \qquad (14)$$

and

$A= A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m= a_0+a_1\alpha+a_2\alpha^2+\ldots+a_{m-1}\alpha^{m-1}$ (15)

$B= B_0+B_1\alpha+B_2\alpha^2+\ldots+B_m\alpha^m=b_0+b_1\alpha+b_2\alpha^2+\ldots+b_{m-1}\alpha^{m-1}$ (16)

$C= C_0+C_1\alpha+C_2\alpha^2+\ldots+C_m\alpha^m=AB^2 \pmod{\alpha^{m+1}+1}$

(17)

From (13), each coefficients of the element c can be executed by

$c_i = C_i+C_m$, for $0\leq i \leq m-1$ (18)

From (13), $c=ab^2$ is separated into two parts: the first part performs $C=AB^2$ modulo $\alpha^{m+1}+1$; and the second part performs modulo $p(\alpha)$.

Definition 2: The inner product of two extended elements in $GF(2^m)$ is defined as the sum of product for each term; thus, the inner product of two elements is give by

$A * B^2= A_0B_0+A_1B_1\alpha^3+A_2B_2\alpha^6+\ldots+A_mB_m\alpha^{3m}$. (19)

Definition 3: Let two extended elements be $A^{(-2i)}$ and $[B^2]^{(i)}$, respectively. Then i-th inner product is defined as $A^{(-2i)}*[B^2]^{(i)}$, thus, i-th inner product is depicted by

$A^{(-2i)} * [B^2]^{(i)}= (A_{<2i>}+A_{<1+2i>}\alpha+A_{<2+2i>}\alpha^2+\ldots+A_{<m+2i>}\alpha^m) *$

$(B_{<-i>}+B_{<1-i>}\alpha^2+\ldots+B_{<m-1- i>}\alpha^{2m-2}+B_{<m-i>}\alpha^{2m})$

$= A_{<2i>}B_{<-i>}+A_{<1+2i>}B_{<1-i>}\alpha^3+ \ldots+A_{<m+2i>}B_{<m-i>}\alpha^{3m}$

(20)

where $A^{(0)} * [B^2]^{(0)}= A * B^2$, $i = 0, 1, 2,\ldots,m$.

Theorem 1: Let $S^{(i)} = A^{(-2i)} * [B^2]^{(i)}$, where $i=0, 1, 2,\ldots,m$. Then, multiplication $C=AB^2$ can be represented as

$C=AB^2= (S^{(0)}+S^{(1)}+\ldots+S^{(m)}) \pmod{\alpha^{m+1}+1}$ (21)

Proof: The circular convolution method in discrete signal system[14] can efficiently multiply the two sequences via two N-point sequences that need N inner product operations. A square element is decomposed of m+1 recursive elements based on the configuration of circular convolution algorithm. Therefore, the product of $C=AB^2$ can be expressed accordingly:

$$
\begin{aligned}
C\ &= AB^2(\text{mod } \alpha^{m+1}+1)\\
&= (A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)(B_0+B_1\alpha^2+B_2\alpha^4+\ldots+B_m\alpha^{2m}) (\text{mod } \alpha^{m+1}+1)\\
&= (A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast(B_0+B_1\alpha^2+B_2\alpha^4+\ldots+B_m\alpha^{2m}) (\text{mod } \alpha^{m+1}+1)\\
&+(A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast(B_m\alpha^{2m}+B_0+B_1\alpha^2+\ldots+B_{m-1}\alpha^{2m-2}) (\text{mod } \alpha^{m+1}+1)\\
&+(A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast(B_{m-1}\alpha^{2m-2}+B_m\alpha^{2m}+B_0+\ldots+B_{m-2}\alpha^{2m-4}) (\text{mod } \alpha^{m+1}+1)\\
&+\ldots\\
&+(A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast(B_1\alpha^2+B_2\alpha^4+B_3\alpha^6+\ldots+B_0) (\text{mod } \alpha^{m+1}+1)\\
&= (A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast[B^2]^{(0)} (\text{mod } \alpha^{m+1}+1)\\
&+(A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast[B^2]^{(1)}\alpha^{-2} (\text{mod } \alpha^{m+1}+1)\\
&+(A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast[B^2]^{(2)}\alpha^{-4} (\text{mod } \alpha^{m+1}+1)\\
&+\ldots\\
&+(A_0+A_1\alpha+A_2\alpha^2+\ldots+A_m\alpha^m)\ast[B^2]^{(m)}\alpha^{-2m}(\text{mod } \alpha^{m+1}+1) \qquad (22)
\end{aligned}
$$

From (6), $A^{(-2i)} = \alpha^{-2i}A$, therefore, $C=AB^2$ (mod α^{m+1}+1) can be rewritten as

$$
\begin{aligned}
C\ &= AB^2 (\text{mod } \alpha^{m+1}+1)\\
&= (A^{(0)}\ast[B^2]^{(0)}+A^{(-2)}\ast[B^2]^{(1)}+A^{(-4)}\ast[B^2]^{(2)}+\ldots+A^{(-2m)}\ast[B^2]^{(m)}) (\text{mod } \alpha^{m+1}+1) \qquad (23)
\end{aligned}
$$

Let $S^{(i)} = A^{(-2i)}\ast[B^2]^{(i)}$, where i = 0,1,2,…,m, then the multiplication of $C = AB^2$ can be rewritten as

$$C\ = AB^2= (S^{(0)}+S^{(1)}+\ldots+S^{(m)})(\text{mod } \alpha^{m+1}+1)$$

Q.E.D.

The multiplication of $C=AB^2$ is very regular and simple since it multiplies two extended elements by the inner product and cyclic shifting as mentioned above. Operation (18) proves this multiplication algorithm clearly requires a m+1 inner product proceeding for performing $C=AB^2$ (mod α^{m+1}+1). Each inner product is proceeded by the multiplication of two elements, in which the element A must be a periodic shifting-left-by-two-bits and the square element must be a periodic shifting-right-by-one-bit. The results of each inner product are arranged as the formula in Equation (18). The ab^2 in GF(2^m) can be multiplied by Theorem 1 as a simple computing ab^2 algorithm:

Algorithm 1:

```
    A=a and B= b
    $C^{(0)}$= 0
```

```
$S^{(0)} = A^{(0)} * [B^2]^{(0)} \pmod{\alpha^{m+1+}1}$
For i=1 to m
{
    $C^{(i)} = (C^{(i-1)}+S^{(i-1)}) \pmod{\alpha^{m+1}+1}$        (24)
    $S^{(i)} = A^{(-2i)} * [B^2]^{(i)} \pmod{\alpha^{m+1+}1}$        (25)
}
$C = (C^{(m)}+S^{(m)}) \bmod \alpha^{m+1}+1$
    (26)
$c = C \bmod p(\alpha)$
```

The following example illustrates the bit-parallel multiplication in the case m=4.

Example 1: Here we verify the correctness of Theorem 1. Assume that m=4, $p(x)= 1 +x +x^2 +x^3 +x^4$ is an irreducible AOP over GF(2), and α is a root of p(x). For $\forall$ a, b $\in$ GF(2^4), the product $c=ab^2$, where

$a = a_0+a_1\alpha+a_2\alpha^2+a_3\alpha^3$, $a_i\in$ GF(2) for i=0,1,2,3

$b = b_0+b_1\alpha+b_2\alpha^2+b_3\alpha^3$, $b_i\in$ GF(2) for i=0,1,2,3

$c = c_0+c_1\alpha+c_2\alpha^2+c_3\alpha^3$, $c_i\in$ GF(2) for i=0,1,2,3

Here we define $C=AB^2 \pmod{\alpha^5+1}$, where

$A = A_0+A_1\alpha+A_2\alpha^2+A_3\alpha^3+A_4\alpha^4 = a_0+a_1\alpha+a_2\alpha^2+a_3\alpha^3$

$B^2= B_0+B_1\alpha^2+B_2\alpha^4+B_3\alpha^6+B_4\alpha^8 = b_0+b_1\alpha^2+b_2\alpha^4+b_3\alpha^6$

$C = C_0+C_1\alpha+C_2\alpha^2+C_3\alpha^3+C_4\alpha^4$

Then $A_4=B_4=0$, $A_i=a_i$, and $B_i = b_i$ for i=0,1,2,3. Hence, product $c=ab^2$ contains two part proceeding. First, multiplication $C=AB^2 \pmod{\alpha^5+1}$ is based on Theorem 1, and assume the coefficients of B^2 are cyclic shifting to right for each step operations; the coefficients of A are cyclic shifting to left-2-bits for each step operations. Second, the modulo unit is given by $c= C \bmod p(\alpha)$, where C is the result of multiplication $C=AB^2$. Therefore,

Step1:

$C^{(0)} = 0$

$S^{(0)} = A * B^2 \pmod{\alpha^5+1}$

$= (A_0+A_1\alpha+A_2\alpha^2+A_3\alpha^3+A_4\alpha^4)*(B_0+B_1\alpha^2+B_2\alpha^4+B_3\alpha^6+B_4\alpha^8) \pmod{\alpha^5+1}$

$= (A_0B_0+A_1B_1\alpha^3+A_2B_2\alpha^6+A_3B_3\alpha^9+A_4B_4\alpha^{12}) \pmod{\alpha^5+1}$

$= A_0B_0+A_1B_1\alpha^3+A_2B_2\alpha+A_3B_3\alpha^4+A_4B_4\alpha^2$

step 2: first cyclic shifting

$C^{(1)} = C^{(0)} + S^{(0)}$

$= A_0B_0+ A_1B_1\alpha^3+ A_2B_2\alpha+ A_3B_3\alpha^4+ A_4B_4\alpha^2$

$S^{(1)} = A^{(-2)}*[B^2]^{(1)} \pmod{\alpha^5+1}$

$= (A_2+A_3\alpha+A_4\alpha^2+A_0\alpha^3+A_1\alpha^4)*(B_4+B_0\alpha^2+B_1\alpha^4+B_2\alpha^6+B_3\alpha^8) \pmod{\alpha^5+1}$

$= (A_2B_4+A_3B_0\alpha^3+A_4B_1\alpha^6+A_0B_2\alpha^9+A_1B_3\alpha^{12}) \pmod{\alpha^5+1}$

$= A_2B_4+A_3B_0\alpha^3+A_4B_1\alpha+A_0B_2\alpha^4+A_1B_3\alpha^2$

step 3: 2nd cyclic shifting

$C^{(2)} = (C^{(1)} + S^{(1)}) \pmod{\alpha^5+1}$

$= (A_0B_0+A_2B_4)+(A_1B_1+A_3B_0)\alpha^3+(A_2B_2+A_4B_1)\alpha+(A_3B_3+A_0B_2)\alpha^4+(A_4B_4+A_1B_3)\alpha^2$

$S^{(2)} = A^{(-4)}*[B^2]^{(2)} \pmod{\alpha^5+1}$

$= (A_4+A_0\alpha+A_1\alpha^2+A_2\alpha^3+A_3\alpha^4)*(B_3+B_4\alpha^2+B_0\alpha^4+B_1\alpha^6+B_2\alpha^8) \pmod{\alpha^5+1}$

$= (A_4B_3+A_0B_4\alpha^3+A_1B_0\alpha^6+A_2B_1\alpha^9+A_3B_2\alpha^{12}) \pmod{\alpha^5+1}$

$= A_4B_3+A_0B_4\alpha^3+A_1B_0\alpha+A_2B_1\alpha^4+A_3B_2\alpha^2$

step 4: 3rd cyclic shifting

$C^{(3)} = (C^{(2)} + S^{(2)}) \pmod{\alpha^5+1}$

$= (A_0B_0+A_2B_4+A_4B_3)+(A_1B_1+A_3B_0+A_0B_4)\alpha^3+(A_4B_1+A_4B_1+A_1B_0)\alpha+(A_3B_3+A_0B_2+ A_2B_1)$
$\alpha^4+(A_4B_4+A_1B_3+ A_3B_2)\alpha^2$

$S^{(3)} = A^{(-6)}*[B^2]^{(3)} \pmod{\alpha^5+1}$

$= (A_1+A_2\alpha+A_3\alpha^2+A_4\alpha^3+A_0\alpha^4)*(B_2+B_3\alpha^2+B_4\alpha^4+B_0\alpha^6+B_1\alpha^8) \pmod{\alpha^5+1}$

$= (A_1B_2+A_2B_3\alpha^3+A_3B_4\alpha^6+A_4B_0\alpha^9+A_0B_1\alpha^{12}) \pmod{\alpha^5+1}$

$= A_1B_2+A_2B_3\alpha^3+A_3B_4\alpha+A_4B_0\alpha^4+A_0B_1\alpha^2$

step 5: 4th cyclic shifting

$C^{(4)} = (C^{(3)} + S^{(3)}) \pmod{\alpha^5+1}$

$= (A_0B_0+A_2B_4+A_4B_3+A_1B_2)+(A_1B_1+A_3B_0+A_0B_4+A_2B_3)\alpha^3+(A_4B_1+A_4B_1+A_1B_0+ A_3B_4)\alpha$
$+(A_3B_3+A_0B_2+ A_2B_1+ A_4B_0)\alpha^4+(A_4B_4+A_1B_3+ A_3B_2+ A_0B_1)\alpha^2$

$S^{(4)} = A^{(-8)}*[B^2]^{(4)} \pmod{\alpha^5+1}$

$= (A_3+A_4\alpha+A_0\alpha^2+A_1\alpha^3+A_2\alpha^4)*(B_1+B_2\alpha^2+B_3\alpha^4+B_4\alpha^6+B_0\alpha^8) \pmod{\alpha^5+1}$

$= (A_3B_1+A_4B_2\alpha^3+A_0B_3\alpha^6+A_1B_4\alpha^9+A_2B_0\alpha^{12}) \pmod{\alpha^5+1}$

$= A_3B_1+A_4B_2\alpha^3+A_0B_3\alpha+A_1B_4\alpha^4+A_2B_0\alpha^2$

step 6: Based on above operations, we can obtain the following coefficients of C:

$C = (C^{(4)} + S^{(4)}) \pmod{\alpha^5+1}$

$=(A_0B_0+A_2B_4+A_4B_3+A_1B_2+A_3B_1)+(A_1B_1+A_3B_0+A_0B_4+A_2B_3+A_4B_2)\alpha^3+$
$(A_4B_1+A_4B_1+A_1B_0+A_3B_4+A_0B_3)\alpha+(A_3B_3+A_0B_2+ A_2B_1+ A_4B_0+ A_1B_4)\alpha^4+$
$(A_4B_4+A_1B_3+ A_3B_2+ A_0B_1+ A_2B_0)\alpha^2$

$C_0 = A_0B_0+A_2B_4+A_4B_3+A_1B_2+A_3B_1$

$C_1= A_2B_2+A_4B_1+A_1B_0+A_3B_4+A_0B_3$

$C_2= A_4B_4+A_1B_3+A_3B_2+A_0B_1+A_2B_0$

$C_3 = A_1B_1+A_3B_0+A_0B_4+A_2B_3+A_4B_2$

$C_4= A_3B_3+A_0B_2+A_2B_1+A_4B_0+A_1B_4$

Finally, the element C modulo $p(\alpha)$ can obtain the coefficients of the element c using

$c_i = C_i+C_4$, for i=0, 1, 2, 3

From step 1 to 5, the output of each step is fixed location by $(\alpha^0, \alpha^3, \alpha, \alpha^4, \alpha^2)$ order arrangement. Therefore, our ideal proceeding, each step is very simplify and regular multiplying ab^2 using the properties of inner product and bi-directional cyclic shifting, and suited for implementing bit-parallel multiplication. As above mentions, in the next section we will present a cellular array for performing AB^2 computation.

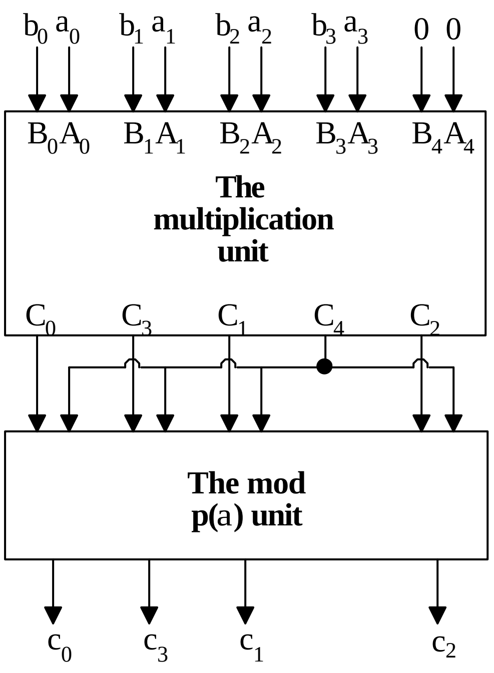
4 The novel bit-parallel cellular multipliers

This section uses Algorithm 1 to construct cellular multiplier for performing ab^2 multiplication, as shown Fig. 2. This circuit contains two parts: the multiplication and modulo $p(\alpha)$ units. The multiplication unit uses the properties of inner product and cyclic shift to perform $C=AB^2 \pmod{\alpha^{m+1}+1}$. Modulo $p(\alpha)$ unit is performs $c= C \pmod{\alpha^{m+1}+1}$, where the element C is the output of the multiplication unit. Our ideal multiplier has the following features:

(1) This multiplier is simple and regular multiplying operations, which uses the properties of cyclic shifting and inner product.

(2) This multiplier has the multiplication and modulo $p(\alpha)$ units.

(3) The time complexity of our proposed multiplier only requires m+3 2-input XOR gate delays.



b_0 a_0 b_1 a_1 b_2 a_2 b_3 a_3 0 0
B_0A_0 B_1A_1 B_2A_2 B_3A_3 B_4A_4
The multiplication unit
C_0 C_3 C_1 C_4 C_2
The mod p(a) unit
c_0 c_3 c_1 c_2

Fig. 2: The multiplier architecture for GF(2^4)

A. The multiplication unit:

Based on Algorithm 1, the multiplication unit is comprised of $(m+1)^2$ inner-product cells and m summation cells. The basic inner-product cell of the multiplication unit computes C_i as indicated in Equations (24) and (25). The basic summation cell of the multiplication unit is indicated in Equation (26). Fig. 3 shows a two dimension signal flow graph array of the multiplication unit. Each inner-product cell is contained of one 2-input AND gate and one 2-input XOR gate as depicted in Fig. 4. Each summation cell is composed of one 2-input XOR gate as depicted in Fig. 5. The whole multiplication unit desires $(m+1)^2$ 2-input AND gates and $(m+1)^2$ + m+1 2-input XOR gates. The overall computation delay of the multiplication unit demands m+1 2-input XOR gate delays, where let AND gate delay is shorter than XOR gate delay.

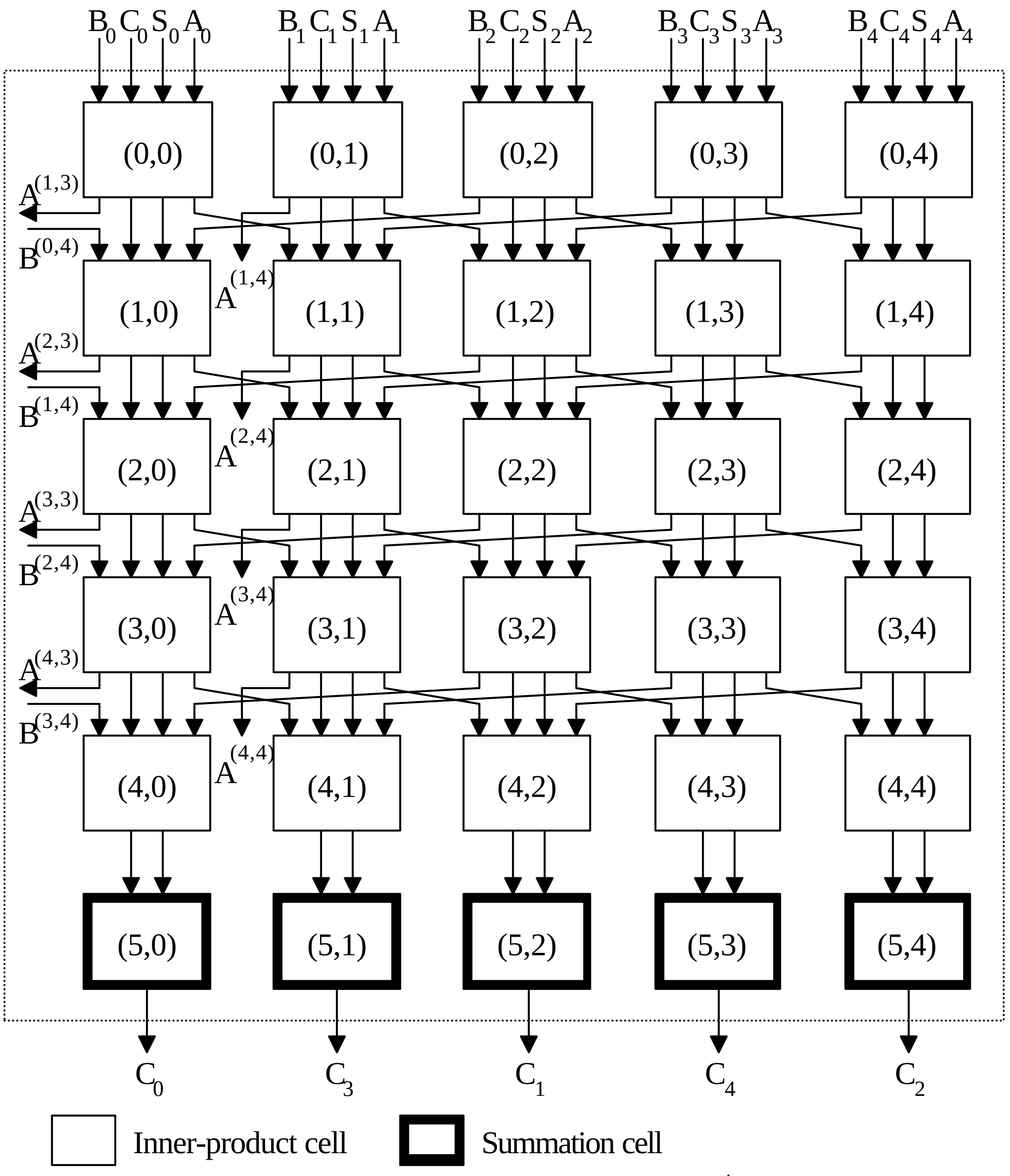


Fig. 3: The multiplication unit for GF(2^4)

Assuming that the inner-product cell illustrated in Fig.4 is located at the i-th low and j-th column of the multiplication unit, denote by (i,j)-cell, where 0≤i,j≤m. The coefficients of elements A and B enter the array from the top in parallel form. In the (i,j) inner-product cell, the AND gate is used to perform $B^{(i-1,j)}$ and $A^{(i-2,j+1)}$ computation, and the output is put in the temporary storage $S^{(i,j)}$; the XOR gate is used to execute the temporary storage $S^{(i-1,j)}$ and sum $C^{(i-1,j)}$ total, where $B^{(i-1,j)}$ is the output of B in (i-1,j)cell, $A^{(i-2,j+1)}$ is the output of A in (i-2,j+1)cell, and $C^{(i-1,j)}$ is the output of C in (i-1,j)cell.

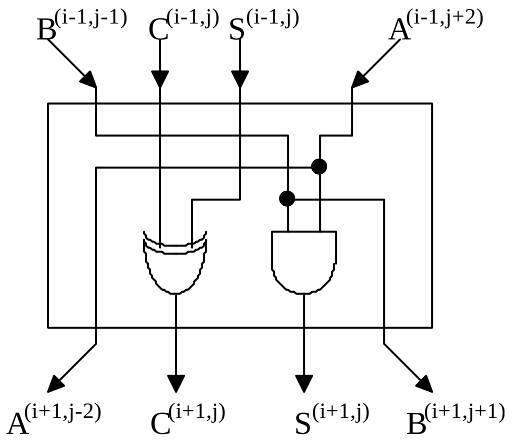


Fig. 4: The inner-product cell (i,j)

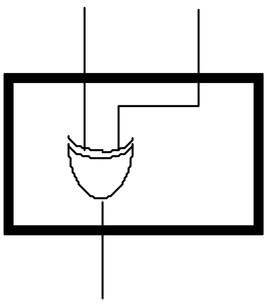


Fig 5: The summation cell (i)

B. The modulo p(a) unit:

Fig. 6 shows a two-dimension signal flow graph array of the modulo p(α) unit. Each summation cell includes one 2-input XOR gate, as depicted in Fig. 5. This unit is given by c=C (mod p(α)), where p(α) is an irreducible AOP, then $c_i = C_i + C_m$. Therefore, this unit wants m 2-input XOR gates. Based on the structures of the multiplication unit, the coefficients of C finally arrange by (C_0+ $C_1\alpha^3$+...+ $C_m\alpha^{3m}$) mod α^{m+1}+1 forms for example GF(2^4). Hence, the output of this multiplier is permuted by (c_0, c_1,c_3,c_2) arrangement.

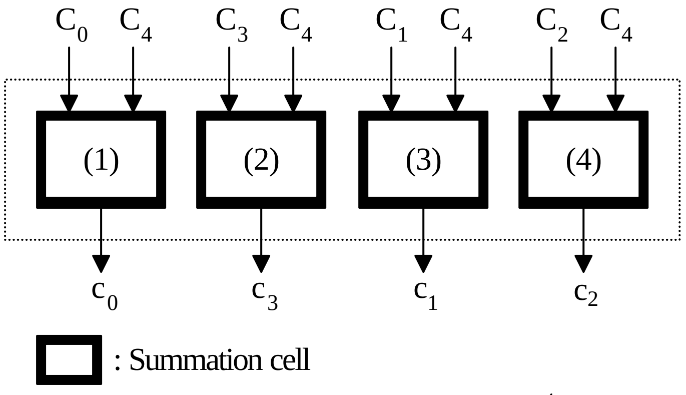


Fig. 6: The mod p(α) unit for GF(2^4)

C. Comparison

For the circuit complexity of the multiplication and mod p(α) units, the circuit complexity requires $(m+1)^2$ 2-input AND gates and $(m+1)^2 + 2m$ 2-input XOR gates; the computation time requires m+3 2-input XOR gate delays. The comparisons of circuit complexity and per cell computation time between the presented cellular multiplier, Law' s developed cellular multiplier [7], is listed in Table 1. The computation speed of our presented multiplier improved by per cell computation delay and complexity over the other multiplier. Our proposed architecture of circuit complexity are less than the other multiplier.

Table 1: Comparison of the related celluar-array multiplier computations

Multiplier / Item	Wei [8]	Law, et al.[7]	New proposed
Number of cells	m^2	m^2	Inner-product cell: $(m+1)^2$ summation cell: 2m+1
Circuit complexity per cell	3 2-input AND 1 2-input XOR 1 3-input XOR	1 3-input XOR 2 2-input AND	Inner-product cell: **1** 2-input AND **1** 2-input XOR summation cell: **1** 2-input XOR
Computation time per cell	$T_{AND}+T_{3XOR}$	$T_{AND}+T_{3XOR}$	$T_{AND}+T_{XOR}$
Computation delay	2m gates delay	2m gates delay	m+3 gates delay
Pre-computed polynomial	yes	yes	No

5 Pipeline architecture for exponentiation in GF(2^m)

Let β and S be elements of GF(2^m), where m+1 is a prime, then the exponentiation of β is defined [8]

$$S=\beta^N, \quad 0\le N\le 2^m-1 \qquad (27)$$

For any integer $N\le 2^m-1$, N can be expressed by

$$N = n_0+n_1 2+n_2 2^2+\ldots+n_{m-1}2^{m-1},\ n_i\in GF(2),\ i=0, 1, 2,\ldots, m-1 \qquad (28)$$

By means of polynomial form of representation, substrate Eq. (28) to Eq. (27) can obtain the exponentiation of β by means of a polynomial form as follows:

$$\begin{aligned} S &= \beta^N \\ &= \beta^{n_0+n_1 2+n_2 2^2+\ldots+n_m 2^{m-1}} \end{aligned}$$

$$= (\beta^{n_0})(\beta^{n_1})^2(\beta^{n_2})^{2^2}\ldots(\beta^{n_m})^{2^{m-1}} \qquad (29)$$

where if $n_i = 1$, then $\beta^{n_i} = \beta$, else $\beta^{n_i} = 1$

In order to reconstruct computing exponentiation for recursive architectures which is based on our proposed bit-parallel multipliers, hence, the exponentiation can be delineated powers form as follows:

$$\begin{aligned}\beta^N &= (\beta^{n_0})(\beta^{n_1})^2(\beta^{n_2})^{2^2}\ldots(\beta^{n_m})^{2^{m-1}}\\ &= \beta^{n_0}[\beta^{n_1}(\beta^{n_2})^2\ldots(\beta^{n_m})^{2^{m-2}}]^2\\ &= \beta^{n_0}[\beta^{n_1}[\beta^{n_2}\ldots(\beta^{n_m})^{2^{m-4}}]^2]^2\\ &= ¼\\ &= \beta^{n_0}[\beta^{n_1}[\beta^{n_2}\ldots[\beta^{n_{m-2}}(\beta^{n_{m-1}})^2]^2\ldots]^2 \qquad (30)\end{aligned}$$

Equation (30) is suitable for our proposed performing multiplier, because its possessed recursive function of polynomial form operations by the multiplication of AB^2. Based on our proposed multiplier, the exponentiation can be presented as follows algorithm:

Algorithm 2:

```
if $n_{m-1} = 1$ then $F = \beta$ else $F = 1$
for i = m-2 to 0
{
    if $n_i = 1$ then $E = \beta$ else $E = 1$
    $F = EF^2$
}
c= F mod $p(\alpha)$
```

According to algorithm 2, Fig. 7 shows computing exponentiation in GF(2^4). In Fig. 7 the MUX of proceeding is controlled selection n_i, i=0, 1, 2, ..., m-1, respectively. If n_i=1 then the output of the multiplier of input element is element β, else fixed value α^0=1. In order to make the signals arrive concurrently at each input of the multipliers. However, latches are required. In this architecture, the latch can be constructed by a combination of m+1 pieces of D-type flip-flops in parallel. In Fig.7 the D_i, if the multiplication unit uses by our proposed multiplier, then D_i= τ(m+3)i, where τ is one 2-input XOR gate delay time. The basis function of our proposed multiplier can be realized m-1 pipeline multipliers for computing exponentiation. Hence, the latency complexity of our proposed exponentiation requires m-1 clock cycles.

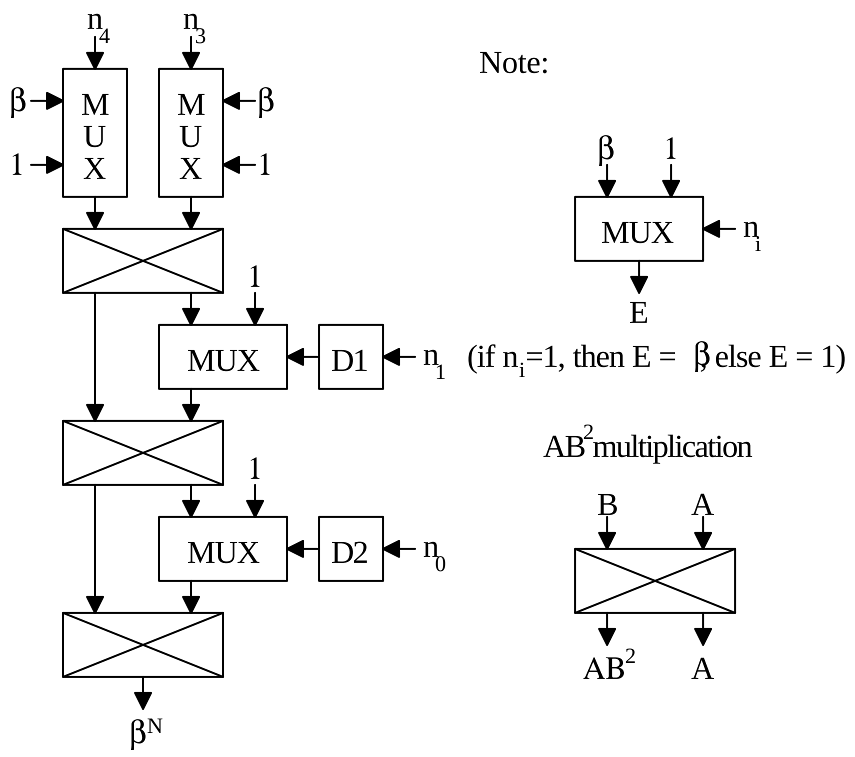


Fig. 7: Computing exponentiation for GF(2^4)

6 Conclusion

In this paper, we have explored circular convolution algorithms for computing AB^2 multiplication into low-complexity systolic architecture in a class field GF(2^m). These functions have the following properties:

(1) The new proposed structure possesses - both the multiplication and modulo p(α) units, where p(x) is an irreducible AOP.

(2) The multiplication unit uses the properties of the inner product and cyclic shifting to construct low-complexity cellular architecture.

(3) The computation time of the designed multiplier is less than the conventional cellular multipliers.

For finite field multiplication, we conclude that our proposed circular convolution algorithms are more efficient as their basic cells have less computation time. Comparison of the related cellular architecture reveal that our constructed cellular architecture is shorter than the conventional multipliers for per cell circuit complexity and computing time. In addition, comparison of parallel multiplier of GF(2^m) defined by an irreducible AOP of m is listed in Table 2. The complexity of our proposed multiplier is similar to the conventional low-complexity multipliers. Moreover, in complexity resembles presented low-complexity articles, our proposed multipliers are the only reality for bit-parallel cellular architectures (shown in table 2).

In addition, we also uses pipeline configuration to perform the exponentiations.

Table 2: Comparison of parallel multipliers of GF(2^m) defined an irreducible AOP of degree m

Multiplier	Structure	Based used
ITM[3]	Modular	Polynomial
HWBM[10]	Modular	Polynomial
M_MOM[11]	Modular	Normal
WDBM[5]	Modular	Weakly dual
New Proposed	Cellular	Polynomial

7 References

[1] M. Anwarul, Muzhong Wang, and Vijay K. Bhargava, " Modular construction of low complexity parallel multipliers for a class of finite fields GF(2^m)" IEEE trans. on computers vol. 41, no. 8, pp. 962~971, 1992.

[2] T. Itoh and S. Tsujii, "Structure of parallel multipliers for a class of fields GF(2^m)" Info. Comp. Vol. 83, pp. 21~40, 1989.

[3] Shyue-Win Wei, "A systolic power-sum circuit for GF(2^m)" IEEE trans. on computers vol. 43, no. 2, pp. 226~229, 1994.

[4] C.K. Koc and B. Sunar, " Low complexity bit-parallel canonical and normal basis multipliers for a class of finite fields" IEEE trans. on computers vol. 47, no. 3, pp. 353~356, 1998.

[5] H. Wu, M. A. Hasan, and Lan F. Blake, " New low-complexity bit-parallel finite field multipliers using weakly dual bases", IEEE trans. on computers vol. 47, no. 11, pp. 1223~1234, 1998.

[6] Y.R. Shayan, Tho Le-Ngoc, and V.K. Bhargava,"Binary-decision approach to fast chien search for software decoding of BCH codes", IEE ptoc. Vol. I34, Pt. F, No.6, pp. 629-632, 1987.

[7] B.A. Laws, Jr, and C.K. Rushforth,"A cellular-array multiplier for GF(2^m)", IEEE trans. on computers vol. C-20 , pp. 1573-1578,1971.

[8] Shyue-Win wei, "VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in GF(2^m)", IEEE trans. on circuits and systems-II, vol. 44, no. 10, pp.847-855, Oct.1997.

[9] C. C. Wang and D. Pei, " A VLSI design for computing exponentiations in GF(2^m) and its application to generate pseudorandom number sequences.", IEEE trans. on computers vol. C-39, pp.

258-262, 1990.

[10] P. A. Scott, S. J. Simmons, S. E. Tavares, and L. E. Peppard. "Architectures for exponentiation in GF(2^m).", IEEE J. Select. Areas commun., Vol.6, pp. 578-586, 1988.

[11] E. R. Berlekamp, Algebraic Coding Theory, revised ed. Laguna Hills, CA: Aegean Park, 1984.

[12] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Dentsh, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in GF(2^m)," IEEE trans. on computers vol. C-34 , pp.709-716, 1985.

[13] C. L. Wang and J. L. Lin, " A systolic architectures for computing inverses and divisions in finite fields GF(2^m)," IEEE trans. on computers vol. C-42 , pp.1010-1015, 1993.

[14] J. G. Proakis and D. G. Manolakis, "Digital signal processing principles algorithms, and applications ", Prentice-Hills, 1996.

[15] M.A. Hasan, M. Wang, and V.K. Bhargava, " A modified Massey-Omura multiplier for a class of finite fields", IEEE trans. on computers vol. C-41 , No.8, pp.962-972, Aug. 1992.