Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 6 (6): 1362-1372, 2006
ISSN 1812-5654

Hardware Implementations of GF (2^m) Arithmetic Using Normal Basis

Turki F. Al-Somani and Alaaeldin Amin
Department of Computer Engineering, King Fahd University of Petroleum and Minerals,
Dhahran 31261, Saudi Arabia

Abstract: This study presents a survey of algorithms used in field arithmetic over GF (2^m) using normal basis and their hardware implementations. These include the following arithmetic field operations: addition, squaring, multiplication and inversion. This study shows that the type II Sunar-Koc multiplier is the best multiplier with a hardware complexity of m^2 AND gates + $\frac{3}{2}m(m-1)$ XOR gates and a time complexity of $T_A + (1+\lceil \log_2(m) \rceil)T_x$. The study also show that the Itoh-Tsujii inversion algorithm was the best inverter and it requires almost $\log_2(m-1)$ multiplications.

Key words: Normal Basis, GF (2^m) arithmetic, multiplication, inversion

INTRODUCTION

Efficient computations in finite fields and their architectures are important in many applications, including coding theory, computer algebra systems and public-key cryptosystems (e.g., elliptic curve cryptosystems (ECC) (Koblitz, 1987). Although all finite fields of the same cardinality are isomorphic, their arithmetic efficiency depends greatly on the choice of the basis use for field element representation. The most commonly used basis are polynomial basis (PB) and normal basis (NB) (Menezes, 1993; Menezes *et al*., 1993). Normal basis is more suitable for hardware implementations than polynomial basis since operations in normal basis representation are mainly comprised of rotation, shifting and exclusive-ORing which can be efficiently implemented in hardware (Lidl and Niederreiter, 1994). This study surveys GF (2^m) field arithmetic operations and algorithms.

A normal basis in GF (2^m) is a basis of the form $(\beta^{2^{m-1}}, \beta^{2^{m-2}}, ..., \beta^{2^1}, \beta^{2^0})$, where $\beta \in GF(2^m)$. In normal basis, an element $A \in GF(2^m)$ can be uniquely represented in the form $A = \sum_{i=0}^{m-1} \alpha_i \beta^{2^i}$ where, $\alpha_i \in \{0,1\}$. Arithmetic operations using normal basis in GF (2^m) are performed as follows:

Addition: Addition is performed by a simple bit-wise exclusive-OR (XOR) operation.

Squaring: Squaring is simply a rotate left operation. Thus, if $A = (a_{m-1}, a_{m-2}, ...a_1, a_0)$, then $A^2 = (a_{m-2}, a_{m-3}, ...a_0, a_{m-1})$.

Multiplication: $\forall$ A, B $\in$ GF (2^m), where

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$$

and

$$B = \sum_{i=0}^{m-1} b_i \beta^{2^i},$$

The product C = A * B, is given by:

$$C = A * B = \sum_{i=0}^{m-1} c_i \beta^{2^i}$$

Multiplication is defined in terms of a set of m multiplication matrices $\lambda^{(k)}$ (k = 0, 1,..., m-1).

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \lambda_{ij}^{(k)} a_i b_j \; \forall k = 0,1,...,m-1$$

where,

$$\lambda_{ij}^{(k)} \in \{0,1\}.$$

The number of non-zero elements in the λ matrix defines the complexity of the multiplication process and accordingly the complexity of its hardware implementation. This value is denoted as C_N and is equal to 2m-1 for optimal normal basis (ONB) (Mullin *et al*., 1989). An optimal normal basis is one with the minimum possible number of non-zero elements in the λ_{ij} matrix. Such basis typically lead to efficient hardware implementations since operations are mainly comprised of rotation, shifting and exclusive-OR operations.

Derivation of values of the λ matrix elements is dependent on the filed size m. There are two types of optimal normal basis that are refereed to as Type I and Type II (Mullin *et al*., 1989). An ONB of Type I exists a given field GF (2^m) if:

- m + 1 is a prime
- 2 is a primitive in GF (m + 1)

Corresponding Author: Turki F. Al-Somani, Department of Computer Engineering,
King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

On the other hand, a Type II optimal normal basis exists in GF (2^m) if:

- 2m + 1 is prime
- either 2 is a primitive in GF (2m + 1) or 2m + 1 ≡ 3 (mod 4) and 2 generates the quadratic residues in GF (2m + 1)

An ONB exists in GF (2^m) for 23% of all possible values of m (Mullin *et al.*, 1989). The $\lambda^{(k)}$ matrix can be constructed by a k-fold cyclic shift to $\lambda^{(0)}$ as follows:

$$\lambda_{ij}^{(k)} = \lambda_{i-k,j-k}^{(0)} \text{ for all } 0 \le i, j, k \le m-1$$

The $\lambda^{(0)}$ matrix is derived differently for the two types of ONB. For the Type I ONB, $\lambda_{ij}^{(0)} = 1$ iff i and j satisfy one of the following two congruencies (Rosing, 1999):

- $2^i + 2 \equiv 1 \bmod (m + 1)$
- $2^i + 2^j \equiv 0 \bmod (m + 1)$

For Type II ONB, $\lambda_{ij}^{(k)} = 1$ iff i and j satisfy one of the following four congruencies (Rosing, 1999):

- $2^i + 2^j \equiv 2^k \bmod (2m + 1)$
- $2^i + 2^j \equiv -2^k \bmod (2m + 1)$
- $2^i - 2^j \equiv 2^k \bmod (2m + 1)$
- $2^i - 2^j \equiv -2^k \bmod (2m + 1)$

Therefore, $\lambda_{ij}^{(0)} = 1$ iff i and j satisfy one of the following four congruencies:

$$2^i \pm 2^j \equiv \pm 1 \bmod (2m + 1)$$

Example 1: The λ matrix of Type I ONB over GF (2^4) is constructed as:

$$\lambda^{(0)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \quad \lambda^{(1)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix},$$

$$\lambda^{(2)} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \quad \lambda^{(3)} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

Example 2: The λ matrix of Type II ONB over GF (2^5) is constructed as:

$$\lambda^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}, \quad \lambda^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix},$$

$$\lambda^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad \lambda^{(3)} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix},$$

$$\lambda^{(4)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Inversion: Inverse of $\alpha \in$ GF (2^m), denoted as a^{-1}, is defined as follows.

$$aa^{-1} \equiv 1 \bmod 2^m$$

Most inversion algorithms are derived from Fermat's Little Theorem, where

$$a^{-1} = a^{2^m - 2}$$

for all $\alpha \neq 0$ in GF (2^m).

MULTIPLICATION

Multiplication is more complicated than addition and squaring operations in finite field arithmetic. An efficient multiplier is highly needed and is the key for efficient finite field computations. Finite filed multipliers using normal basis can be classified into two main categories: (1) λ-matrix based multipliers and (2) Conversion based multipliers.

λ-Matrix Based Multipliers: Massey and Omura (1986) proposed an efficient normal basis bit-serial multiplier over GF (2^m). The Massey-Omura multiplier requires only two m-bit cyclic shift registers and combinational logic. The combinational logic consists of a set of AND and XOR logic gates. The first implementation of the Massey-Omura multiplier was reported by Wang *et al.* (1985). The space complexity of the Massey-Omura multiplier is (2m - 1) AND gates + (2m - 2) XOR gates, while the time complexity is $T_A + (1 + \log_2 (m - 1)) T_X$, where T_A and T_X are the delay of one AND gate and one XOR gate, respectively. One advantage of

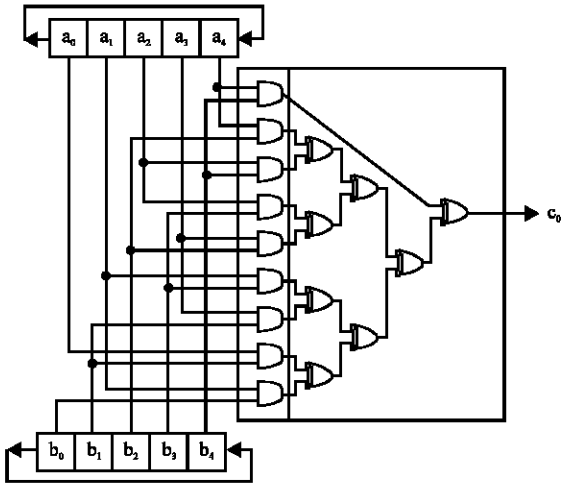

Fig. 1: GF (2^m) bit-serial Massey-Omura multiplier

the Massey-Omura multiplier is that it can be used with both types of the optimal normal basis (Type I and Type II). Another advantage is that it is a bit-serial multiplier and hence the same circuitry used to generate c_0 can be used to generate c_i (i = 1, 2,..., m - 1). The bit-parallel version of the Massey-Omura multiplier requires ($2m^2$ - m) AND gates + ($2m^2$ - 2m) XOR gates.

Example 3: Let A, B and C ∈ GF (2^5). Using the λ matrix in Example 2, the product C is found as:

$$c_0 = a_4b_4 + (a_0b_1 + a_1b_0) + (a_1b_3 + a_3b_1) + (a_2b_4 + a_4b_2) + (a_2b_3 + a_3b_2)$$
$$c_1 = a_0b_0 + (a_1b_2 + a_2b_1) + (a_2b_4 + a_4b_2) + (a_3b_0 + a_0b_3) + (a_3b_4 + a_4b_3)$$
$$c_2 = a_1b_1 + (a_2b_3 + a_3b_2) + (a_3b_0 + a_0b_3) + (a_4b_1 + a_1b_4) + (a_4b_0 + a_0b_4)$$
$$c_3 = a_2b_2 + (a_3b_4 + a_4b_3) + (a_4b_1 + a_1b_4) + (a_0b_2 + a_2b_0) + (a_0b_1 + a_1b_0)$$
$$c_4 = a_3b_3 + (a_4b_0 + a_0b_4) + (a_0b_2 + a_2b_0) + (a_1b_3 + a_3b_1) + (a_1b_2 + a_2b_1)$$

The corresponding GF (2^m) bit-serial Massey-Omura multiplier is shown in Fig. 1. c_1 is generated by simple 1-bit rotation of the content of the two registers in Fig. 1. Figure 1 shows the case where c_0 is generated. Other product bits (c_1, c_2,..., c_{m-1}) are generated in a similar manner.

Let A = B = (11000), ⇒ A × B = A^2 = $(11000)^2$ = (10001) = C. Now using the Massey-Omura multiplier we apply the above formulas to get:

$$c_0 = 1 + 0 + 0 + 0 + 0 = 1$$
$$c_1 = 0 + 0 + 0 + 0 + 0 = 0$$
$$c_2 = 0 + 0 + 0 + 0 + 0 = 0$$
$$c_3 = 0 + 0 + 0 + 0 + 0 = 0$$
$$c_4 = 1 + 0 + 0 + 0 + 0 = 1.$$

Hasan *et al.* (1993) proposed a modified version of the Massey-Omura parallel multiplier which works only with ONB Type I. The basic idea is to decompose the λ^{m-1} matrix as a sum of two matrices P and Q.

$$\lambda^{m-1} = P + Q \pmod 2$$

where the (i, j) entry of P is defined as follows:

$$\begin{cases} 1, & \text{if } i = (\frac{m}{2} + j); \\ 0 & \text{otherwise.} \end{cases}$$

The product c_{m-1-k} can be obtained as:

$$c_{m-1-k} = A\lambda^{(k)}B^t = APB^t + A^{(k)}QB^{(k)t}\hat{c} + \hat{c}_{m-1-k} \pmod 2$$

where $A^{(k)}$ and $B^{(k)}$ are the k-cyclic shifted vectors of A and B. Hasan *et al.* (1993) noticed that $\hat{c}$ is independent of k and is present in each c_{m-1-k}. Hence, it can be precomputed once at the beginning thus reducing the multiplier's hardware complexity. Compared to the Massey-Omura parallel multiplier, this multiplier requires only m^2 - 1 XOR gates and the same number of AND gates. However, the time complexity of this modified parallel multiplier is the same as the Massey-Omura parallel multiplier and the number of XOR gates is still O (m^2).

Example 4: Let A, B and C ∈ GF (2^4). Using the λ_3 matrix in Example 1, P and Q is found as:

$$\lambda^{(3)} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, Q = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Let A = B = (11000), ⇒ A × B = A^2 = $(11000)^2$ = (1001) = C.

1. Since $\hat{c} = APB^t$, we have:
$\hat{c}$ = [0 0 1 1] .P. $[0\ 0\ 1\ 1]^t$ = 0,
1. Since $\hat{c}_{m-1-k} = A^{(k)}\ QB^{(k)t}$, we have:

$$\hat{c}_3 = [0011] \cdot Q \cdot [0011]^t = 1 \Rightarrow c_3 = \hat{c}_3 + \hat{c} = 1 + 0 = 1$$
$$\hat{c}_2 = [1001] \cdot Q \cdot [1001]^t = 0 \Rightarrow c_2 = \hat{c}_2 + \hat{c} = 0 + 0 = 0$$
$$\hat{c}_1 = [1100] \cdot Q \cdot [1100]^t = 0 \Rightarrow c_1 = \hat{c}_1 + \hat{c} = 0 + 0 = 0$$
$$\hat{c}_0 = [0110] \cdot Q \cdot [0110]^t = 1 \Rightarrow c_0 = \hat{c}_0 + \hat{c} = 1 + 0 = 1$$

Alternatively, Gao and Sobelman (2000) noticed that the Massey-Omura bit-serial multiplier is constructed using an AND plane and an XOR plane. Gao and

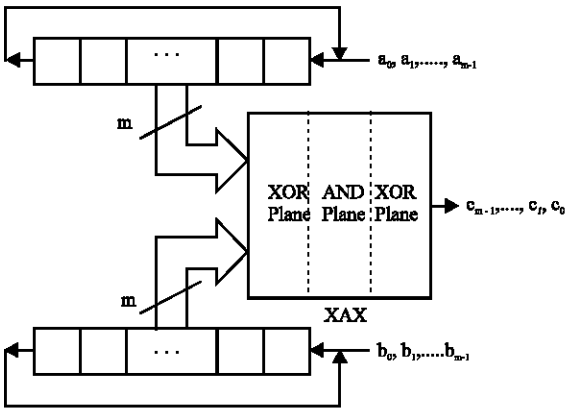

Fig. 2: XOR plane, AND and XOR planes of Gao and Sobelman multiplier

Sobelman (2000) suggested to rearrange these planes into three planes: XOR-plane, AND-plane and another XOR-plane as shown in Fig. 2. This is achieved by rearranging the multiplication equation as:

$$c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i \lambda_{ij}^{(k)} b_j = \sum_{i=0}^{m-1} a_i \left[\sum_{j=0}^{m-1} \lambda_{ij}^{(k)} b_j\right]$$

The Gao and Sobelman (2000) multiplier requires the same number of XOR gates and has the same time complexity as the Massey-Omura multiplier. The only improvement was reducing the number of AND gates to only m^2 AND gate compared to $(2m^2 - m)$ AND gates for the Massey-Omura multiplier.

Example 5: Let A, B and C ϵ GF (2^5). Using the λ matrix in Example 2 and Gao and Sobelman (2000) multiplier, the product C is found as:

$$\begin{aligned}
c_0 &= a_0b_1 + a_1(b_0 + b_3) + a_2(b_3 + b_4) + a_3(b_1 + b_2) + a_4(b_2 + b_4)\\
c_1 &= a_1b_2 + a_2(b_1 + b_4) + a_3(b_0 + b_4) + a_4(b_2 + b_3) + a_0(b_0 + b_3)\\
c_2 &= a_2b_3 + a_3(b_0 + b_2) + a_4(b_0 + b_1) + a_0(b_3 + b_4) + a_1(b_1 + b_4)\\
c_3 &= a_3b_4 + a_4(b_1 + b_3) + a_0(b_0 + b_1) + a_1(b_0 + b_4) + a_2(b_0 + b_2)\\
c_4 &= a_4b_0 + a_0(b_2 + b_4) + a_1(b_2 + b_3) + a_2(b_0 + b_1) + a_3(b_1 + b_3)
\end{aligned}$$

Let $A = B = (11000), \Rightarrow A \times B = A^2 = (11000)^2 = (10001) = C$, we have:

$$\begin{aligned}
c_0 &= 0+0+0+0+1 = 1\\
c_1 &= 0+0+1+1+0 = 0\\
c_2 &= 0+0+0+0+0 = 0\\
c_3 &= 1+1+0+0+0 = 0\\
c_4 &= 0+0+0+0+1 = 1.
\end{aligned}$$

Reyhani-Masoleh and Hasan (2004) presented another multiplier based on the same idea of rearranging the multiplier's XOR and AND planes (Gao and Sobelman, 2000) but with a different formulation. The product terms are reformulated as:

$$c_k = a_k b_k + \sum_{(r,s)\in\Phi_k} (a_r + a_s)(b_r + b_s), 0 \le k \le m-1.$$

where Φ_k contains the coordinates of 1's in the upper part of the λ_k matrix. The space complexity is (m^2) AND + $(3m^2\text{-}3m)$ XOR gates while the time complexity is $T_A + \lceil \log_2 (2m - 1) \rceil T_X$. This means that the Gao and Sobelman multiplier is more efficient than the reported multiplier by Reyhani-Masoleh and Hasan (2004).

Example 6: Let A, B and C ϵ GF (2^5). Using the λ matrix in Example 2 and Reyhani-Masoleh and Hasan multiplier, we have:

$$\begin{aligned}
\Phi_0 &= \{(0,1),(1,3),(2,3),(2,4)\}\\
\Phi_1 &= \{(0,3),(1,2),(2,4),(3,4)\}\\
\Phi_2 &= \{(0,3),(0,4),(1,4),(2,3)\}\\
\Phi_3 &= \{(0,1),(0,2),(1,4),(3,4)\}\\
\Phi_4 &= \{(0,2),(0,4),(1,2),(1,3)\}
\end{aligned}$$

Thus, the product C can be found as follows:

$$\begin{aligned}
c_0 &= a_0b_0 + (a_0 + a_1)(b_0 + b_1) + (a_1 + a_3)(b_1 + b_3)\\
&\quad +(a_2 + a_3)(b_2 + b_3) + (a_2 + a_4)(b_2 + b_4)\\
c_1 &= a_1b_1 + (a_0 + a_3)(b_0 + b_3) + (a_1 + a_2)(b_1 + b_2)\\
&\quad +(a_2 + a_4)(b_2 + b_4) + (a_3 + a_4)(b_3 + b_4)\\
c_2 &= a_2b_2 + (a_0 + a_3)(b_0 + b_3) + (a_0 + a_4)(b_0 + b_4)\\
&\quad +(a_1 + a_4)(b_1 + b_4) + (a_2 + a_3)(b_2 + b_3)\\
c_3 &= a_3b_3 + (a_0 + a_1)(b_0 + b_1) + (a_0 + a_2)(b_0 + b_2)\\
&\quad +(a_1 + a_4)(b_1 + b_4) + (a_3 + a_4)(b_3 + b_4)\\
c_4 &= a_4b_4 + (a_0 + a_2)(b_0 + b_2) + (a_0 + a_4)(b_0 + b_4)\\
&\quad +(a_1 + a_2)(b_1 + b_2) + (a_1 + a_3)(b_1 + b_3)
\end{aligned}$$

The corresponding GF (2^m) bit-serial Reyhani-Masoleh and Hasan multiplier is shown in Fig. 3. Let $A = B = (11000), \Rightarrow A \times B = A^2 = (11000)^2 = (10001) = C$, we have:

$$\begin{aligned}
c_0 &= 0+0+1+1+1 = 1\\
c_1 &= 0+1+0+1+0 = 0\\
c_2 &= 0+1+1+1+1 = 0\\
c_3 &= 1+0+0+1+0 = 0\\
c_4 &= 1+0+1+0+1 = 1.
\end{aligned}$$

Table 1: The λ-based multipliers and their space and time complexities

Multiplier	Space Complexity	Time Complexity	Type of ONB
(Massey and Omura, 1986)	$(2m^2 - m)$ AND + $(2m^2 - 2m)$ XOR	$T_A + (1 + \log_2 (m-1)) T_X$	I and II
(Hasan *et al*., 1993)	$(2m^2 - m)$ AND + $(m^2 - 1)$ XOR	$T_A + (1 + \log_2 (m-1)) T_X$	I
(Gao and Sobelman, 2000)	(m^2) AND + $(2m^2 - 2m)$ XOR	$T_A + (1 + \log_2 (m-1)) T_X$	I and II
(Reyhani-Masoleh and Hasan, 2004)	(m^2) AND + $(3m^2 - 3m)$ XOR	$T_A + \lceil \log_2 (2m-1 \rceil T_X$	I and II
(Reyhani-Masoleh and Hasan, 2002)	(m^2) AND + $(m^2 - 1)$ XOR	$T_A + (1 + \log_2 (m-1)) T_X$	I and II

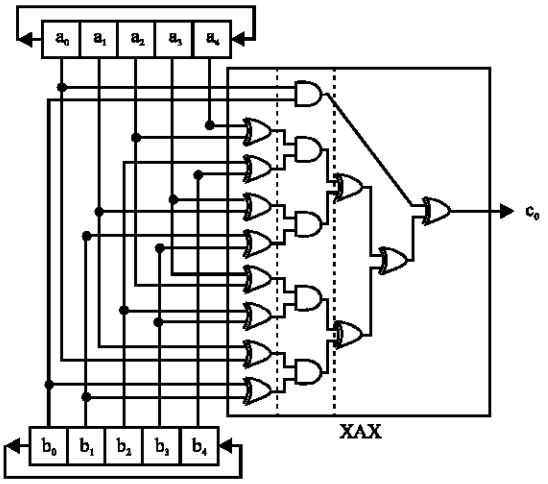

Fig. 3: GF (2^m) bit-serial Reyhani-Masoleh and Hasan multiplier

Reyhani-Masoleh and Hasan (2002) have also reported a parallel multiplier that takes advantage of the symmetry of the λ matrix and reduced redundancy in the λ matrix. This is achieved by rewriting the λ matrix as $\lambda = U + U^T + D$, where D is the diagonal matrix and U is the upper triangular matrix having zeros at diagonal entries. Thus, the multiplication product can be written as follows:

$$C = A \times U \times B^T + B \times U \times A^T + A \times D \times B^T$$

The U matrix is reformulated according to the value of m (even or odd). However, the proposed multiplier's space complexity is (m^2) AND + $(m^2 - 1)$ XOR and the time complexity is $T_A + (1 + \log_2 (m-1)) T_X$ which represents minor gain as compared to other multipliers. Table 1 compares the λ-based multipliers space and time complexities.

CONVERSION BASED MULTIPLIERS

Koc and Sunar (1998) proposed a new bit-parallel multiplier over GF (2^m). The multiplier was employed to perform Type I optimal normal basis multiplication by converting the two operands into canonical basis (a basis of the form $(\alpha^{m-1}, \ldots, \alpha^2, \alpha^1, 1)$, where $\alpha \in$ GF (2^m) is a root of the generating polynomial of degree m is called a canonical basis). After multiplication, the result is converted back to the normal basis. The proposed technique yields a slight improvement in the number of XOR gates as compared to the Massey-Omura multiplier. The number of required XOR gates is reduced down to m^2-1 as compared to $2m^2-m$ for the Massey-Omura multiplier, while its time complexity $(T_A + (2 + \log_2 (m-1)) T_X$ is slightly more than Massey-Omura's time complexity. The main advantage of this technique, however, is the opening of a new direction in multiplications based on basis conversion which was first explored by Sunar and Koc (2001).

Sunar and Koc (2001) reported a new Type II optimal normal basis multiplier. The main idea of the work was based on converting the two operands into their equivalent representation in another basis, perform the multiplication in that basis and convert the product back to the normal basis. The conversion step requires only a single clock cycle since it is nothing but a permutation of the normal basis (Sunar and Koe, 2001).

Using optimal normal basis Type II and assuming that p = 2m + 1 is prime, another new basis which is obtained by a simple permutation of the normal basis elements was presented. Let β be $\gamma + \gamma^{-1}$, where γ is the primitive pth root of unity, the new basis can be written in the form $\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2} + \gamma^3, + \gamma^{-z}, \ldots, \gamma^m + \gamma^{-m}$. Two elements $\hat{A}$ and $\hat{B} \in$ GF (2^m)can be represented in the new basis as:

$$\hat{A} = \sum_{i=1}^{m} \hat{a}_i (\gamma^i + \gamma^{-i}) = \sum_{i=1}^{m} \hat{a}_i \beta_i$$

and

$$\hat{B} = \sum_{i=1}^{m} \hat{b}_i (\gamma^i + \gamma^{-i}) = \sum_{i=1}^{m} \hat{b}_i \beta_i$$

The product $\hat{C} = \hat{A} \cdot \hat{B}$ is written as:

$$\hat{C} = (\sum_{i=1}^{m} \hat{a}_i (\gamma^i + \gamma^{-i}))(\sum_{j=1}^{m} \hat{b}_j (\gamma^j + \gamma^{-j}))$$

This product can transformed to the following form:

$$\hat{C} = \sum_{i=1}^{m} \sum_{j=1}^{m} \hat{a}_i \hat{b}_j (\gamma^{i-j} + \gamma^{-(i-j)}) + \sum_{i=1}^{m} \sum_{j=1}^{m} \hat{a}_i \hat{b}_j (\gamma^{i+j} + \gamma^{-(i+j)})$$
$$= \hat{C}_1 + \hat{C}_2$$

The term $\hat{C}_1$ has the property that the exponent (i-j) of γ is already within the proper range, i.e., $-m \le (i-j) \le m$ for all I, j $\in$ [1, m]. The term $\hat{C}_2$ should be ensured to be in the proper range. Hence, $\hat{C}_2$ is computed as follows:

$$\hat{C}_2 = \sum_{i=1}^{m}\sum_{j=1}^{m} \hat{a}_i\hat{b}_j(\gamma^{i+j} + \gamma^{-(i+j)})$$
$$= \sum_{i=1}^{m}\sum_{j=1}^{m-i} \hat{a}_i\hat{b}_j(\gamma^{i+j} + \gamma^{-(i+j)}) + \sum_{i=1}^{m}\sum_{j=m-i+1}^{m} \hat{a}_i\hat{b}_j(\gamma^{i+j} + \gamma^{-(i+j)})$$
$$= \hat{D}_1 + \hat{D}_2$$

The exponents of the basis elements $\gamma^{i+j} + \gamma^{-(i+j)}$ in $\hat{D}_1$ are guaranteed to be in the proper range $1 \le (I + j) \le m$ for I = 1, 2,..., m and j = 1, 2,..., m-i. If k = i + j, then product $\hat{a}_i\hat{b}_j$ contributes to the basis element β_k as i and j take these values. The basis elements of $\hat{D}_2$, however, are all out of range. Thus, the identity γ^{2m+1} is used to bring them to the proper range as:

$$\hat{D}_2 = \sum_{i=1}^{m}\sum_{j=m-i+1}^{m} \hat{a}_i\hat{b}_j(\gamma^{i+j} + \gamma^{-(i+j)})$$
$$= \sum_{i=1}^{m}\sum_{j=m-i+1}^{m-i} \hat{a}_i\hat{b}_j(\gamma^{2m+1-(i+j)} + \gamma^{-(2m+1-(i+j))})$$

Therefore, if $k = i + j > m$, β_k is replaced by β_{2m+1+k} and thus the final product can be found as:

$$\hat{C} = \hat{C}_1 + \hat{D}_1 + \hat{D}_2$$

The hardware complexity of this bit-parallel multiplier is m^2 AND gates + 3/2 (m^2 - m) XOR gates and the time complexity is $T_A + (1 + \lceil \log_2 m \rceil) T_X$. This represents an improvement of about 25 percent less XOR gates compared to the Massey-Omura multiplier but with slightly more delay. A major advantage of this method, however, is the fact that there is no need to store the λ matrix to perform multiplications which requires m^3 locations if all λ matrices are stored or m^2 if only $\lambda^{(0)}$ is stored. This is a major advantage in area constrained environments since we only need to store the permutation array which requires only $m\log_2(m)$ storage locations.

Example 7: Let A, B and C $\in$ GF (2^5). Since 2m + 1 = 2.5 + 1 = 11 and 2 is primitive in modulo 11, there exists an optimal basis of Type II for the field GF (2^5), which is of the form $(\beta^{2^0}, \beta^{2^1}, \beta^{2^2}, \beta^{2^3}, \beta^{2^4})$, where, Using the identity $\gamma^{11} = 1$, we find the converted form as follows:

- The first three exponents 1, 2 and 4 are in the proper range [1, 5].
- We have 16 = 15 (mod 11), which brings the exponent 16 to the proper range.
- In order to bring 8 to the range [1, m] = [1, 5], we use the identity $\gamma^8 = \gamma^{11-8} = \gamma^3$.

Basis →	β_1	β_2	β_3	β_4	β_5
C_1 →	$a_1b_2+a_2b_1$ $a_2b_3+a_3b_2$ $a_3b_4+a_4b_3$ $a_4b_5+a_5b_4$	$a_1b_3+a_3b_1$ $a_2b_4+a_4b_2$ $a_3b_5+a_5b_3$	$a_1b_4+a_4b_1$ $a_2b_5+a_5b_2$	$a_1b_5+a_5b_1$	
D_1 →		a_1b_1	a_1b_2 a_2b_1	a_1b_3 a_2b_2 a_3b_1	a_1b_4 a_2b_3 a_3b_2 a_4b_1
D_2 →	a_3b_5	a_4b_5 a_5b_4	a_1b_5 a_4b_4 a_5b_3	a_2b_5 a_3b_4 a_4b_3 a_5b_2	a_1b_5 a_2b_4 a_3b_3 a_4b_2 a_5b_1

Fig. 4: The construction of $\hat{C}_1, \hat{D}_1, \hat{D}_2$ and $\hat{C}$ in GF (2^5) in (Sunar and Koc, 2001)

Thus the converted form is (β_1, β_2, β_4, β_3, β_5). Figure 4 shows the construction of $\hat{C}_1, \hat{D}_1, \hat{D}_2$.

The product C can be found as:

$$\hat{c}_1 = (\hat{a}_1\hat{b}_2 + \hat{a}_2\hat{b}_1) + (\hat{a}_2\hat{b}_3 + \hat{a}_3\hat{b}_2) + (\hat{a}_3\hat{b}_4 + \hat{a}_4\hat{b}_3) + (\hat{a}_4\hat{b}_5 + \hat{a}_5\hat{b}_4) + \hat{a}_5\hat{b}_5$$
$$\hat{c}_2 = (\hat{a}_1\hat{b}_3 + \hat{a}_3\hat{b}_1) + (\hat{a}_2\hat{b}_4 + \hat{a}_4\hat{b}_2) + (\hat{a}_3\hat{b}_5 + \hat{a}_5\hat{b}_3) + \hat{a}_1\hat{b}_1 + \hat{a}_4\hat{b}_5 + \hat{a}_5\hat{b}_4$$
$$\hat{c}_3 = (\hat{a}_1\hat{b}_4 + \hat{a}_4\hat{b}_1) + (\hat{a}_2\hat{b}_5 + \hat{a}_5\hat{b}_2) + \hat{a}_1\hat{b}_2 + \hat{a}_2\hat{b}_1 + \hat{a}_3\hat{b}_5 + \hat{a}_4\hat{b}_4 + \hat{a}_5\hat{b}_3$$
$$\hat{c}_4 = (\hat{a}_1\hat{b}_5 + \hat{a}_5\hat{b}_1) + \hat{a}_1\hat{b}_3 + \hat{a}_2\hat{b}_2 + \hat{a}_3\hat{b}_1 + \hat{a}_2\hat{b}_5 + \hat{a}_3\hat{b}_4 + \hat{a}_4\hat{b}_3 + \hat{a}_5\hat{b}_2$$
$$\hat{c}_5 = \hat{a}_1\hat{b}_4 + \hat{a}_2\hat{b}_3 + \hat{a}_3\hat{b}_2 + \hat{a}_4\hat{b}_1 + \hat{a}_1\hat{b}_5 + \hat{a}_2\hat{b}_4 + \hat{a}_3\hat{b}_3 + \hat{a}_4\hat{b}_2 + \hat{a}_5\hat{b}_1$$

Let $A = B = (11000) \Rightarrow A \times B = A^2 = (11000)^2 = (10001) C$, $\hat{A} = \hat{B} = (10100)$, we have:

$$\hat{c}_1 = 0+0+0+0+1 = 1$$
$$\hat{c}_2 = 0+0+0+0+0+0 = 0$$
$$\hat{c}_3 = 0+0+0+0+1+0+1 = 0$$
$$\hat{c}_4 = 0+0+0+0+0+0+0+0 = 0$$
$$\hat{c}_5 = 0+0+0+0+0+0+1+0+0 = 1.$$

$\hat{C} = (10001)$ and the product C after conversion back to the normal basis is also (10001).

Alternatively, Wu *et al.* (2002) extended the work described by (Gao and Vanstone, 1995) and presented a new Type II multiplier. The basic idea is the same as that of (Sunar and Koc, 2001). Multiplication is performed by converting the two operands into another basis which is simply a permutation of the normal basis, perform the multiplication and convert the product back to the original normal basis. The difference is only in the multiplication part. The product can be found in the new basis as:

$$\hat{c}_j = \sum_{i=1}^{m} \hat{a}_i(\hat{b}_{s(j+i)} + \hat{b}_{s(j-i)}), j = 1,2,...,m.$$

where s (i) is defined as:

Table 2: The conversion based multipliers and their space and time complexities

Multiplier	Space complexity	Time Complexity	Type of ONB
(Koc and Sunar, 1998)	$(2m^2 - m)$ AND + $(m^2 - 1)$ XOR	$T_A + (2 + \log_2 (m-1)) T_X$	I
(Sunar and Koc, 2001)	m^2 AND + 3/2 $(m^2 - m)$ XOR	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	II
(Wu *et al.*, 2002)	m^2 AND + $(2m^2 - m)$ XOR	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	II

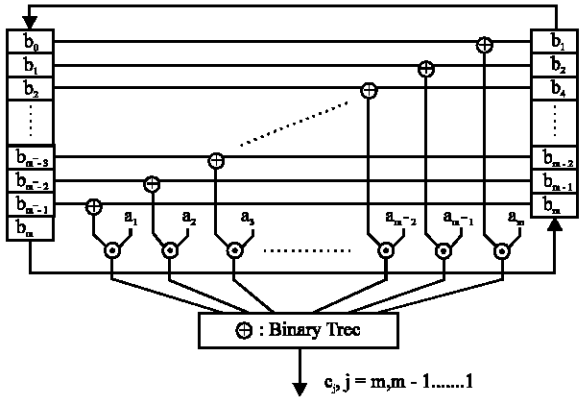

Fig. 5: The Wu *et al.*, (2002) multiplier

$$s(i) = \begin{cases} i \bmod 2m+1, \text{ if } 0 \le i \bmod 2m+1 \le m; \\ 2m+1-i \bmod 2m+1, \text{ otherwise;} \end{cases}$$

The hardware complexity of this multiplier is m^2 AND gates + $2m^2$ - m XOR gates which is worse than the Sunar and Koc multiplier (2001) as shown in Fig. 5. However, the time complexity is $T_A + (1 + \lceil \log_2 m \rceil) T_X$ which is the same as the multiplier reported by (Sunar and Koc, 2001).

Example 8: Let A, B and C ∈ GF (2^5). Using the same conversion procedure illustrated in Example 7, the product C can be found as:

$$\hat{c}_1 = \hat{a}_1(\hat{b}_2 + \hat{b}_0) + \hat{a}_2(\hat{b}_3 + \hat{b}_1) + \hat{a}_3(\hat{b}_4 + \hat{b}_2) + \hat{a}_4(\hat{b}_5 + \hat{b}_3) + \hat{a}_5(\hat{b}_5 + \hat{b}_4)$$
$$\hat{c}_2 = \hat{a}_1(\hat{b}_3 + \hat{b}_1) + \hat{a}_2(\hat{b}_4 + \hat{b}_0) + \hat{a}_3(\hat{b}_5 + \hat{b}_1) + \hat{a}_4(\hat{b}_5 + \hat{b}_2) + \hat{a}_5(\hat{b}_4 + \hat{b}_3)$$
$$\hat{c}_3 = \hat{a}_1(\hat{b}_4 + \hat{b}_2) + \hat{a}_2(\hat{b}_5 + \hat{b}_1) + \hat{a}_3(\hat{b}_5 + \hat{b}_0) + \hat{a}_4(\hat{b}_4 + \hat{b}_1) + \hat{a}_5(\hat{b}_3 + \hat{b}_2)$$
$$\hat{c}_4 = \hat{a}_1(\hat{b}_5 + \hat{b}_3) + \hat{a}_2(\hat{b}_5 + \hat{b}_2) + \hat{a}_3(\hat{b}_4 + \hat{b}_1) + \hat{a}_4(\hat{b}_3 + \hat{b}_0) + \hat{a}_5(\hat{b}_2 + \hat{b}_1)$$
$$\hat{c}_5 = \hat{a}_1(\hat{b}_5 + \hat{b}_4) + \hat{a}_2(\hat{b}_4 + \hat{b}_3) + \hat{a}_3(\hat{b}_3 + \hat{b}_2) + \hat{a}_4(\hat{b}_2 + \hat{b}_1) + \hat{a}_5(\hat{b}_1 + \hat{b}_0)$$

Let A = B = (11000), ⇒ A×B = A^2 = $(11000)^2$ = (10001) C, $\hat{A} = \hat{B}$ = (10100), we have:

$$\hat{c}_1 = 0+0+0+0+1 = 1$$
$$\hat{c}_2 = 0+0+0+1+1 = 0$$
$$\hat{c}_3 = 0+0+1+0+1 = 0$$
$$\hat{c}_4 = 0+0+0+0+0 = 0$$
$$\hat{c}_5 = 0+0+1+0+0 = 1.$$

$\hat{C}$ and the product C after conversion back to the normal basis is also (10001).

Table 2 shows the conversion based multipliers and their space and time complexities.

INVERSION

Inversion using normal basis consists of multiplications and cyclic shifts. Since cyclic shifts require almost negligible time, the number of multiplications is the key parameter for efficient inversion. In the following subsections we survey existing algorithms for inversion over GF (2^m) using normal basis. These inversion algorithms can be classified into three main categories: (1) Standard, (2) Exponent Decomposing and (3) Exponent Grouping inversion algorithms.

Standard inversion algorithm: This category contains only one algorithm, which is the first proposed normal basis inversion algorithm over GF (2^m) in Wang *et al.* (1985). The basic idea is derived from Fermat's Little Theorem where the inverse of a ∈ GF (2^m) is given by $a^{-1} = a^{2^m-2}$ since $2^m - 2 = \sum_{i=1}^{m-1} 2^i$, we can express a^{-1} as:

$$a^{-1} = (a^2).(a^{2^2})...(a^{2^{m-1}}) = a^{2^m-2}$$

Algorithm 1: Wang's *et al.* (1985) inversion algorithm.

Inputs: a
Output: a^{-1}

- $B = a^2$, C = 1 and k = 0.
- D = B × C and k = k + 1.
- if k = m-1, a^{-1} = D, Stop.
- if k < m-1, $B = B^2$ and C = D.
- Go back to 2.

This only requires multiplication and cyclic shift operations. The algorithm procedure for computing a^{-1} as suggested by Wang *et al.* (1985) is shown in Algorithm 1. It requires (m-2) multiplications + (m-1) cyclic shifts. The advantage of this method is its simplicity while its disadvantage is the large O (m) number of multiplications.

Example 9: A and A^{-1} C ∈ GF (2^5). The inverse of A is $A^{-1} = A^{2^5-2} = A^{30}$. By applying Wang's *et al.* (1985) inversion algorithm we have:

Step 1: $B = A^2$, $C = 1$, $k = 0$
Step 2: $D = B \times C = A^2 \times 1 = A^2$, $k = 1$
Step 4: $k = 1 < 4 \Rightarrow B = B^2 = (A^2)^2 = A^4$, $C = A^2$
Step 2: $D = B \times C = A^4 \times A^2 = A^6$, $k = 2$
Step 4: $k = 2 < 4 \Rightarrow B = B^2 = (A^4)^2 = A^8$, $C = A^6$
Step 2: $D = B \times C = A^8 \times A^6 = A^14$, $k = 3$
Step 4: $k = 3 < 4 \Rightarrow B = B^2 = (A^8)^2 = A^16$, $C = A^14$
Step 2: $D = B \times C = A^16 \times A^14 = A^{30}$, $k = 4$
Step 3: $k = 4 = 4 \Rightarrow A^{-1} = D = A^{30}$, STOP.

Exponent decomposing inversion algorithms: Since the number of multiplications is the dominant factor in determining the computation time of the inversion operation, several algorithms attempted to improve the inversion speed by decomposing the exponent to reduce the required number of multiplications and replace it with squaring operations which are much simpler compared to multiplication.

In 1988 Itoh and Tsujii (1988) proposed a GF (2^m) inversion algorithm derived from Fermat's Little Theorem using normal basis. The basic idea used was to decompose the exponent m-1 as follows:

$$a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2.$$

The exponent $2m^{-}1$ is further decomposed as follows:

1.If m is odd, then

$$(2^{m-1}-1) = (2^{\frac{m-1}{2}}-1)(2^{\frac{m-1}{2}}+1),$$

and

$$a^{2^{m-1}} = (a^{2^{\frac{m-1}{2}}-1})^{2^{\frac{m-1}{2}}+1}$$

2.If m is even, then

$$2^{m-1}-1 = 2(2^{m-2}-1)+1 = 2(2^{\frac{m-2}{2}}-1)(2^{\frac{m-2}{2}}+1)+1,$$

and

$$a^{2^{m-1}} = a^{2(2^{\frac{m-2}{2}}-1)(2^{\frac{m-2}{2}}+1)+1}$$

The proposed algorithm by (Itoh and Tsujii, 1988) is shown in Algorithm 2 and it requires $\log_2 (m-1)+v (m-1)-1$ multiplications, where v (x) is the number of 1's in the binary representation of x.

Algorithm 2: Itoh-Tsujii inversion algorithm.

Inputs: a
Output: $l = a^{-1}$
1. set $s \leftarrow \lfloor \log_2 (m-1) \rfloor -1$.
2. set $p \leftarrow a$.
3. for i = s down to 0 do
3.1. set r ← shift m-1 to right by s bit(s)
3.2. set $q \leftarrow p$
3.3. rotate q to left by $\lfloor r/2 \rfloor$ bit(s)
3.4. set $t \leftarrow p \times q$
3.5. if last bit of r = 1,
3.5.1. rotate t to left by 1 bit.
3.5.2. $p \leftarrow t \times a$
3.5. else
3.5.3. $p \leftarrow t$
3.6. $s \leftarrow s-1$
4. rotate p to left by 1 bit.
5. set $l \leftarrow p$.
6. return l.

Example 10: Let A and $A^{-1} \in$ GF (2^5). The inverse of A is $A^{-1} = A^{2^5-2} = A^{30}$. By applying Itoh and Tsujii inversion algorithm we have:

step 1. S = 2-1 = 1
step 2. P = A
step 3. i = 1
step 3.1. r = 2
step 3.2. $q = p = A$
step 3.3. $q = q^2 = A^2$
step 3.4. $t = p \times q = A \times A^2 = A^3$
step 3.5.3. $p = t = A^3$
step 3.6. s = s -1 = 1-1 = 0
step 3. i = 0
step 3.1. r = 4
step 3.2. $q = p = A^3$
step 3.3. $q = (q^2)^2 = ((A^3)^2)^2 = A^{12}$
step 3.4. $t = p \times q = A^3 \times A^{12} = A^{15}$
step 3.5.3. $p = t = A^{15}$
step 3.6. s =s -1 = 0-1 = -1
step 4. $P = P^2 = (A^{15})^2 = A^{30}$
step 5. $i = p = A^{30}$
step 6. return $l = A^{30}$

Feng (1989) has also proposed an inversion algorithm which requires the same time complexity as the Itoh and Tsujii inversion algorithm, i.e., O ($\log_2$ (m)). The inversion algorithm was also derived from Fermat's Little Theorem and is also based on exponent decomposition as the Itoh and Tsujii inversion algorithm. Feng defined m-1 as follows:

Let $m_q m_{q-1} \ldots m_1 m_0$ be the binary representation of (m-1), where $m_q = 1$ and m_i is 0 or 1 for i = 0 to q-1, i.e., $m-1 = m_q 2^q + m_{q-1} 2_{q-1} + \ldots + m_1 2^1 + m_0 2^0$. The inverse a^{-1} can be computed using Algorithm 2.

Algorithm 3: Feng's inversion algorithm.

Inputs: a
Output: a^{-1}
1. b = a
2. for i = q to 1 do
2.1. if $m_i = 1$, then $b = b^{2^{-2^i}}$.
2.2. $b = b \times b^{2^{-2^{i-1}}}$.
2.3. if $m_{i-1} = 1$, then $b = b \times a$.
3. $a^{-1} = (b)^{2^{m-m_0}}$, Stop.

Only step 2.2 and 2.3 in Algorithm 3 require multiplications. Step 2.1 and 3 need only cyclic shifts. Thus, the algorithm has (q + p) multiplications where $q = \lfloor \log_2 (m) \rfloor$ and, p = # 1's in the binary expression of (m-1). Accordingly, the proposed algorithm has the same complexity as the Itoh and Tsujii inversion algorithm. The difference is only that the Itoh and Tsujii algorithm performs squaring during each iteration, while Feng's algorithm computes the square roots each iteration and squares at the end by 2^{m-m0}.

Example 11: Let A and $A^{-1} \in GF(2^5)$. The inverse of A is $A^{-1} = A^{2^5-2} = A^{30}$. By applying Feng's inversion algorithm we have: q = 2, m-1 = 4, $m_2 = 1$, $m_1 = 0$, $m_0 = 0$ and p = 0,

step 1. b = A
step 2. i = 2
step 2.1. $m_2 = 1 \Rightarrow b = b^{2^{-2^2}} = A^{2^{-4}}$
step 2.2. $b = b \times b^{2^{2^{2-1}}} = A^{2^{-4}} \times A^{2^{-4^{2^2}}} = A^{2^{-4}(1+2^2)}$
step 2. i = 1
step 2.2. $b = b \times b^{2^{2^{1-1}}} = b \times b^2 = A^{2^{-4}(1+2^2)} \times A^{2^{-4}(1+2^2)(2)}$
$= A^{2^{-4}(1+2^2)(1+2)}$
step 3. $A^{-1} = (b)^{2^{5-0}} = A^{2^5 \times 2^{-4}(1+2^2)(1+2)}$
$= A^{2^1(1+2^2)(1+2)} = A^{30}$, STOP

Takagi *et al.* (2001) proposed another inversion algorithm which was also based on exponent decomposition. The main idea is as follows: Since

$$2^m - 2 = 2^{m-1} + 2^{m-1} - 2 = 2^{m-1} + 2^{m-2} .. + 2^{m-h} + 2^{m-h} - 2,$$
$$a^{-1} = a^{2^m-2} = a^{2^{m-1}} . a^{2^{m-2}} \ldots a^{2^{m-h}+2^{m-h}-2}$$

Hence, the algorithm can use the inversion algorithm by (Itoh and Tsujii, 1988) by replacing m by m - h. This method reduces the number of multiplications through performing an exhaustive search for an optimal value of h. The time complexity, however, is O ($\log_2$ (m)).

EXPONENT GROUPING INVERSION ALGORITHMS

Grouping exponent terms is another approach which attracted some researchers. However, the resulting speed is O (m) which is worse than the O ($\log_2$ (m)) of the Itoh and Tsujii inverter. The inversion algorithms in this category are based on the idea of grouping exponent terms as follows:

Since $a^{-1} = a^{2^m-2} = (a^2).(a^4)\ldots(a^{2^{m-1}})$, this allows grouping exponent terms in different ways. Fenn *et al.* (1996) proposed an inversion algorithm which requires m/2 multiplications. The basic idea was by dividing the exponent terms into two equal groups. The first group contains a^2, a^4,.., a^k, where, $k = 2^{(m-1)/2}$. The other group contains the remaining terms till a^{2^m}. By multiplying the first term in each group and repeated squaring by 2^{2^i}, the following formula can be applied to give the inverse within m/2 multiplications.

$$a^{-1} = \begin{cases} \prod_{i=1}^{(m-1)/2} [a \cdot a^{2^{(m-1)/2}}]^{2^i}, & m \text{ odd;} \\ a^{2^{m-1}} \cdot \prod_{i=1}^{(m-1)/2} [a \cdot a^{2^{(m-1)/2}}]^{2^i}, & m \text{ even.} \end{cases}$$

Example 12: Let A and $A^{-1} \in GF(2^5)$, m = 5 is odd. The inverse of A is $A^{-1} = A^{2^5-2} = A^{30}$. By applying Fenn's inversion algorithm we have:

$$A^{-1} = (AA^4)^2 \cdot (AA^4)^4 = A^{30}$$

Example 13: Let A and $A^{-1} \in GF(2^8)$, m = 8 is even. The inverse of A is $A^{-1} = A^{2^8-2} = A^{254}$. By applying Fenn's inversion algorithm we have:

$$A^{-1} = (AA^8)^2 \cdot (AA^8)^4 \cdot (AA^8)^8 \cdot A^{128} = A^{254}$$

A further improvement, however, have been reported by Jimenez Calvo and Torres (1997). The Jimenez Calvo and Torres (1997) inversion algorithm uses a fixed seed a^2. $a^4 = a^6$, which uses the same idea of grouping into two groups but in a different manner. This can be shown as:

$$a^{-1} = \begin{cases} \prod_{i=0}^{(m-3)/2} [a^6]^{2^{2i}}, & m \text{ odd;} \\ a^{2^{m-1}} \cdot \prod_{i=0}^{(m-4)/2} [a^6]^{2^{2i}}, & m \text{ even.} \end{cases}$$

Table 3: The inverters and their time complexities

Inverter	Class	Time Complexity (No. of Multiplications)
(Wang *et al*., 1985)	Standard	(m-2)
(Itoh and Tsujii, 1988)	Exponent Decomposing	$\log_2 (m-1) + v (m-1) -1$
(Feng, 1989)	Exponent Decomposing	$\lfloor \log2 (m) \rfloor + p$
(Fenn *et al*., 1996)	Exponent Grouping	m/2
(Jimenez Calvo and Torres, 1997)	Exponent Grouping	m/2
(Yen, 1997)	Exponent Grouping	m-1/p + (p-1)

Example 14: Let A and $A^{-1} \epsilon$ GF (2^8), m = 8 is even. The inverse of A is $A^{-1} = A^{2^8-2} = A^{254}$. By applying Jimenez Calvo and Torres inversion algorithm we have:

$$A^{-1} = A^{2^7} \cdot (A^6)^{2^0} \cdot (A^6)^{2^2} \cdot (A^6)^{2^4} = A^{254}$$

This method was generalized by choosing m' and b such that m = bm' + r, where r = [1,2]. This is basically the same way trying to group more terms in the seed at the beginning.

$$a^{-1} = \begin{cases} \prod_{i=0}^{b-1} [a^{2^{m'-1}-2}]^{2^{m'i}}, & m = am' + 1; \\ a^{2^{m-1}} \cdot \prod_{i=0}^{b-1} [a^{2^{m'-1}-2}]^{2^{m'i}}, & m = am' + 2. \end{cases}$$

This requires around m/m' multiplications but still with the same O (m) time complexity.

Example 15: Let A and $A^{-1} \epsilon$ GF (2^{16}), m = 16 is even. The inverse of A is $A^{-1} = A^{2^{16}-2} = A^{65536}$. By applying Jimenez Calvo and Torres generalized inversion algorithm we have:

Let b = 3, 5 and r = 1. First we calculate $A^{2^{m'+1}-2}$ as follows:

$$A^{2^{m'+1}-2} = A^{2^6-2} = A^{2^5} \cdot A^6 \cdot (A^6)^{2^2} = A^{62}.$$

Then applying the generalized equation we get:

$$A^{-1} = A^{62} \cdot (A^{62})^{2^5} \cdot (A^{62})^{2^{10}} = A^{2^{16}-2} = A^{65536}.$$

Yen (1997) proposed another approach for grouping terms which results in reduced number of clock cycles. Yen realized that the fundamental idea behind Fenn's et. al. inverter is to rearrange all terms into (m-1)/2 groups and to extract the common term $(a.a^{2^{(m-1)/2}})$. Accordingly, Yen redefined the common term to contain p terms and a^{-1} can be found as:

$$\prod_{i=1}^{k=(m-1)/p} (X)^{2^i}$$

where the common part X is precomputed as:

$$\prod_{j=0}^{p-1} a^{2^{j(m-1)/p}}$$

The best case requires k + (p - 1) multiplications while the worst case requires k + (p - 1) + (p - 2) = k + 2p - 3 multiplications and the optimal selection of p for m = pk + 1 is $\sqrt{m}$. The time complexity, however, is still O (m). Table 3 summarizes the inversion algorithms and their time complexities.

CONCLUSIONS

In this study we presented a survey on finite field arithmetic using normal basis over GF (2^m). Addition requires simple XOR operation while squaring requires only a cyclic shift. This is the most attractive property of using normal basis rather than using polynomial basis which is more suitable for software implementations.

We categorized normal basis multipliers into two main categories: (1) λ-matrix based multipliers and (2) Conversion based multipliers. The multipliers in (2) are more efficient since they don't require storing the λ-matrix. The comparisons shows that the Type II Sunar-Koc multiplier is the best multiplier with a hardware complexity of m^2 AND gates + 3/2m (m-1) XOR gates and a time complexity of $T_A + (1 + \lceil \log_2 m \rceil) T_X$.

We also categorized the inversion algorithms into three main categories: (1) Standard, (2) Exponent Decomposing and (3) Exponent Grouping. Inversion algorithms in category (3) performed better than the standard with time complexity O (m). The exponent decomposing inversion algorithm of Itoh and Tsujii, however, is found to be the best inverter requiring only $\log_2$ (m-1) multiplications.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support of King Abdul Aziz City for Science and Technology for the grant of the research No. AR-22-17. The authors would like also to acknowledge the support of King Fahd University of Petroleum and Minerals.

REFERENCES

Feng, G.L., 1989. A VLSI Architecture for Fast Inversion in GF (2^m). IEEE Trans. Computers, 38: 1383-1386.

Fenn, S.T.J., M. Benaissa and D. Taylor, 1996. Fast normal basis inversion in GF (2^m). Electronics Lett., 32: 17.

Gao, S. and S. Vanstone, 1995. On orders of optimal normal basis generators. Math. Computation, 64: 1227-1233.

Gao, L. and G.E. Sobelman, 2000. Fast normal basis inversion in GF (2^m). In Proceedings of 13th Annual IEEE International ASIC/SOC Conference, pp: 97-101.

Hasan, M.A., M.Z. Wang and V.K. Bhargava, 1993. A modified Massey-Omura parallel multiplier for a class of finite fields. IEEE Trans. Comput., 42: 1278-1280.

Itoh, T. and S. Tsujii, 1988. A fast algorithm for computing multiplicative inverses in GF (2^m) using normal basis. Inform. Comput., 78: 171-177.

Jimenez Calvo, I. and M. Torres, 1997. Complexity of the inversion in GF (2^m). Electronics Letters.

Koblitz, N., 1987. Elliptic curve cryptosystems. Mathematics of Computation, 48: 203-209.

Koc, C. K. and B. Sunar, 1998. Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields. IEEE Trans. Comput., 47: 353-356.

Lidl, R. and H. Niederreiter, 1994. Introduction to Finite Fields and Their Applications. Cambridge University Press, Cambridge, UK., Revised Edition.

Massey, J.L. and J.K. Omura, 1986. Computational method and apparatus for finite field arithmetic. US Patent No. 4: 587, 627.

Menezes, A.J., 1993. Elliptic Curve Public Key Cryptosystems. Kluwer Academic Publishers.

Menezes, A.J., I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone and T. Yaghoobian, 1993. Applications of Finite Fields. Kluwer Academic Publishers, Boston, MA.

Mullin, R.C., I.M. Onyszchuk, S.A. Vanstone and R.M. Wilson, 1988/89. Optimal normal bases in GF (2^m). Discrete Appl. Math., 22: 149-161.

Reyhani-Masoleh, A. and M.A. Hasan, 2002. A new construction of Massey-Omura parallel multiplier over GF (2^m). IEEE Trans. Comput., 51: 511-520.

Reyhani-Masoleh, A. and M.A. Hasan, 2004. Efficient digit-serial normal basis multipliers over binary extension fields. In ACM Trans. on Embedded Computing Systems, 3: 575-592.

Rosing, M., 1999. Implementing Elliptic Curve Cryptography. Manning Publications Company.

Sunar, B. and C.K. Koc, 2001. An efficient optimal normal basis Type II multiplier. IEEE Trans. Comput., 50: 83-88.

Takagi, N., J. Yoshiki and K. Takagi, 2001. A fast algorithm for multiplicative inversion in GF (2^m). Using normal basis. IEEE Transactions on Computers, 50: 394-398.

Wang, C.C., T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura and I.S. Reed, 1985. VLSI architectures for computing multiplications and inverses in GF (2^m). IEEE Trans. Comput., 34: 709-716.

Wu, H., A. Hasan, I. Blake and S. Gao, 2002. Finite field multiplier using redundant representation. IEEE Trans. Comput., 51: 1306-1316.

Yen, S., 1997. Improved normal basis inversion in GF (2^m). Electronics Lett., 33: 196-197.