EURASIP Journal on Applied Signal Processing 2002:9, 954–960

Low-Complexity Versatile Finite Field Multiplier in Normal Basis

Hua Li
Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta, Canada T1K 3M4
Email: huali@cs.uleth.ca

Chang Nian Zhang
Department of Computer Science, TRLabs, University of Regina, Regina, SK, Canada S4S 0A2
Email: zhang@cs.uregina.ca

Received 6 August 2001 and in revised form 30 August 2002

A low-complexity VLSI array of versatile multiplier in normal basis over GF(2^n) is presented. The finite field parameters can be changed according to the user's requirement and make the multiplier reusable in different applications. It increases the flexibility to use the same multiplier for different applications and reduces the user's cost. The proposed multiplier has a regular structure and is very suitable for high speed VLSI implementation. In addition, the pipeline versatile multiplier can be modified to a low-cost architecture which is feasible in embedded systems and restricted computing environments.

Keywords and phrases: finite field multiplication, Massey-Omura multiplier, normal basis, VLSI, encryption.

1. INTRODUCTION

The finite fields GF(2^n) of characteristic 2 are of great interest for cryptosystems and digital signal processing. The addition operation in GF(2^n) is fast and inexpensive as it can be realized with n bitwise XOR operations. The multiplication operation is costly in terms of gate number and time delay. There have been three main kinds of basis representations of the field elements in GF(2^n): standard (canonical, polynomial) basis, dual basis, and normal basis. Different basis representation multipliers have their own benefits and tradeoffs. The dual basis multiplier [1] needs the least number of gates which leads to the smallest area required for VLSI implementation [2]. The normal basis multiplier, for example, Massey-Omura multiplier [3], is very effective in performing squaring, exponentiation, and inversion operation. The standard basis multiplier [4, 5, 6, 7] is easier to extend to high-order finite fields than the dual or normal basis multipliers.

Most of the proposed finite field multipliers operate over a fixed field. In other words, a new multiplier is needed if there is a change in the field parameters such as the irreducible polynomial defining the representation of the field elements. This makes the multiplier not reusable. There are few versatile multipliers [4, 6, 8, 9] reported and all based on canonical basis. In this paper, we present a new VLSI array of versatile pipeline multiplier based on the normal basis representation. In normal basis, the squaring is a cost-free cyclic shift operation and the inversion (the most complicated operation among the important finite field arithmetic operations) can be effectively computed by Fermat's theorem which requires recursive squaring and multiplication [10, 11]. Three main advantages accrue from the proposed pipelined versatile multiplier. First, the finite field parameters can be changed according to the application environments. It increases the flexibility to use the same multiplier for different applications. Secondly, the structure of the multiplier can be easily extended to higher-order finite fields. Thirdly, the basic architecture of the proposed multiplier can be modified to a low-cost multiplier which is very suitable for both embedded systems and wireless devices with restricted hardware resources. Moreover, the structure of the multiplier has the properties of modularity, simplicity, regular interconnection, and is easy for VLSI implementation. The proposed versatile multiplier can be efficiently used in public-key cryptosystems, such as elliptic curve cryptography; and the digital signal processing, for example, the Reed-Solomon encoder/decoder.

The outline of the remainder of the paper is as follows. In Section 2, we briefly review the normal basis representation and Massey-Omura multiplier. Section 3 contains the derivation of the pipeline versatile normal basis multiplier in GF(2^n) and comparison with previous works. Section 4 concludes with the improved result and a description of areas of applications.

2. MULTIPLICATION ON GF(2^n)

It has been proved that there always exists a normal basis [12] for a given finite field GF(2^n) which is the form of

$$N = \left\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{n-1}}\right\}, \tag{1}$$

where β is a root of the irreducible polynomial $P(x)$ of degree n over GF(2) and n elements of the set are linearly independent.

We say that β generates the normal basis N, or β is a normal element of GF(2^n). Every element $a \in \mathrm{GF}(2^n)$ can be represented by $a = \sum_{i=0}^{n-1} a_i \beta^{2^i}$, where $a_i \in \{0, 1\}$.

The following properties [10] of a finite field GF(2^n) are useful in the applications.

(1) Squaring is a linear operation, that is, given any two elements a and b in GF(2^n),

$$(a + b)^2 = a^2 + b^2. \tag{2}$$

(2) For any element $a \in \mathrm{GF}(2^n)$,

$$a^{2^n} = a. \tag{3}$$

(3) For any element $a \in \mathrm{GF}(2^n)$,

$$1 = a + a^2 + a^4 + \cdots + a^{2^{n-1}}. \tag{4}$$

This implies that the normal basis representation of 1 is $(1, 1, \ldots, 1)$.

(4) Squaring an element a in the normal basis representation is a cyclic shift operation, that is,

$$\begin{aligned} a^2 &= \sum_{i=0}^{n-1} a_i \beta^{2^{i+1}} \\ &= \sum_{i=0}^{n-1} a_{i-1} \beta^{2^i} \\ &= (a_{n-1}, a_0, \ldots, a_{n-2}) \end{aligned} \tag{5}$$

with indices reduced modulo n.

Let a and b be two arbitrary elements in GF(2^n) in a normal basis representation and $c = a \cdot b$ be the product of a and b. We denote $a = \sum_{i=0}^{n-1} a_i \beta^{2^i}$ as a vector $a = (a_0, a_1, \ldots, a_{n-1})$, $b = \sum_{i=0}^{n-1} b_i \beta^{2^i}$ as a vector $b = (b_0, b_1, \ldots, b_{n-1})$, and $c = \sum_{i=0}^{n-1} c_i \beta^{2^i}$ as a vector $c = (c_0, c_1, \ldots, c_{n-1})$, then the last term c_{n-1} of c is a logic function of the components of a and b, that is,

$$c_{n-1} = f(a_0, a_1, \ldots, a_{n-1}; b_0, b_1, \ldots, b_{n-1}). \tag{6}$$

Since squaring in normal representation is a cyclic shift of the element, we have $c^2 = a^2 \cdot b^2$ or equivalently

$$\begin{aligned} &(c_{n-1}, c_0, c_1, \ldots, c_{n-2}) \\ &\quad = (a_{n-1}, a_0, a_1, \ldots, a_{n-2}) \cdot (b_{n-1}, b_0, b_1, \ldots, b_{n-2}). \end{aligned} \tag{7}$$

Hence, the last component c_{n-2} of c^2 can be obtained by the same function f operating on the components of a^2 and b^2. That is,

$$c_{n-2} = f(a_{n-1}, a_0, a_1, \ldots, a_{n-2}; b_{n-1}, b_0, b_1, \ldots, b_{n-2}). \tag{8}$$

By squaring c repeatedly, we get

$$\begin{aligned} c_{n-1} &= f(a_0, a_1, \ldots, a_{n-1}; b_0, b_1, \ldots, b_{n-1}), \\ c_{n-2} &= f(a_{n-1}, a_0, a_1, \ldots, a_{n-2}; b_{n-1}, b_0, b_1, \ldots, b_{n-2}), \\ &\vdots \\ c_0 &= f(a_1, a_2, \ldots, a_{n-1}, a_0; b_1, b_2, \ldots, b_{n-1}, b_0). \end{aligned} \tag{9}$$

Equations 9 define the Massey-Omura multiplier in normal basis representation [10]. In Massey-Omura multiplier, the same logic function f for computing the last component of c_{n-1} of the product c can be used to get the remaining components $c_{n-2}, c_{n-3}, \ldots, c_0$ of the product sequentially. In parallel architecture, we can use n identical logic function f for calculating all components of the product simultaneously.

3. A PIPELINE ARCHITECTURE FOR THE SERIAL VERSATILE NORMAL BASIS MULTIPLIER

In this section, we derive a pipeline architecture to implement the versatile normal basis multiplier. Let c be the product of a and b,

$$c = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \beta^{2^i} \beta^{2^j}. \tag{10}$$

In the normal basis, we have

$$\beta^{2^i} \beta^{2^j} = \sum_{k=0}^{n-1} \lambda_{ij}^{(k)} \beta^{2^k}, \quad \lambda_{ij}^{(k)} \in \mathrm{GF}(2). \tag{11}$$

Thus, we can get

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij}^{(k)} a_i b_j, \quad 0 \le k \le n-1. \tag{12}$$

From the above analysis, we see that the important issue for building a versatile normal basis multiplier is to get the value of $\lambda_{i,j}^{(k)}$ for different irreducible polynomials. The $n \times n$ matrices $\lambda^{(k)}$ $(0 \le k \le n-1)$ whose elements is $\lambda_{i,j}^{(k)}$ $(0 \le i, j \le n-1)$ can be obtained if we know the transformation between the elements of the canonical basis and the elements of the normal basis, that is, the normal basis representation of the elements of the canonical basis.

In the following, we define the multiplication table of the normal basis and use the basis element transformation formula to get the values of the multiplication table, and then obtain the $n \times n$ matrices $\lambda^{(k)}$. Finally, we illustrate the approach to build the versatile pipeline normal basis multiplier.

Definition 1. Let $N = \{\beta, \beta^2, \ldots, \beta^{2^{n-1}}\}$ be a normal basis in GF(2^n), then for any i, j $(0 \le i, j \le n-1)$, $\beta^{2^i}\beta^{2^j}$ is a linear combination of $\beta, \beta^2, \ldots, \beta^{2^{n-1}}$ with coefficients in GF(2). In particular,

$$\beta \begin{bmatrix} \beta \\ \beta^2 \\ \vdots \\ \beta^{2^{n-1}} \end{bmatrix} = T \begin{bmatrix} \beta \\ \beta^2 \\ \vdots \\ \beta^{2^{n-1}} \end{bmatrix}, \tag{13}$$

where T is an $n \times n$ matrix over GF(2). We call T the *multiplication table* of the normal basis N. The number of nonzero entries in T is called the complexity of the normal basis N, denoted by C_N.

There always exists the multiplication table T and the matrix $\lambda^{(k)}$ for a given irreducible polynomial which defines the normal basis in GF(2^n) [12]. After the multiplication table T is obtained, the matrix $\lambda^{(k)}$ can be calculated according to (12). An example is shown below.

Example 1. Let the irreducible polynomial be $P_1(x) = x^5 + x^4 + x^2 + x + 1$ and β be a root of the polynomial, then the canonical basis is $\{1, \beta, \beta^2, \beta^3, \beta^4\}$ and the normal basis is $\{\beta, \beta^2, \beta^4, \beta^8, \beta^{16}\}$. We can get the following normal basis representation for the elements of the canonical basis:

$$\begin{aligned} 1 &= \beta + \beta^2 + \beta^4 + \beta^8 + \beta^{16}, \\ \beta &= \beta, \qquad \beta^2 = \beta^2, \\ \beta^3 &= \beta + \beta^8, \qquad \beta^4 = \beta^4. \end{aligned} \tag{14}$$

The appendix illustrates how to obtain the normal basis representation of β^3.

Thus the element β^i $(i > 5)$ can be reduced to the representation of canonical basis and converted to the corresponding representation of normal basis by the base element transformation formula (14). For instance,

$$\begin{aligned} \beta^{17} &= 1 + \beta^2 + \beta^3 \\ &= 1 + \beta^2 + (\beta + \beta^8) \\ &= \beta^{16} + \beta^4. \end{aligned} \tag{15}$$

Then we can get the multiplication table T for given $P_1(x)$ which is

$$\begin{aligned} T &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}, \\ \beta \begin{bmatrix} \beta \\ \beta^2 \\ \beta^4 \\ \beta^8 \\ \beta^{16} \end{bmatrix} &= T \begin{bmatrix} \beta \\ \beta^2 \\ \beta^4 \\ \beta^8 \\ \beta^{16} \end{bmatrix}. \end{aligned} \tag{16}$$

The product of a and b is

$$\begin{aligned} c &= ab \\ &= c_0\beta + c_1\beta^2 + c_2\beta^4 + c_3\beta^8 + c_4\beta^{16} \\ &= (a_0\beta + a_1\beta^2 + a_2\beta^4 + a_3\beta^8 + a_4\beta^{16}) \\ &\quad \times (b_0\beta + b_1\beta^2 + b_2\beta^4 + b_3\beta^8 + b_4\beta^{16}) \\ &= a_0b_0\beta^2 + a_0b_1\beta^3 + a_0b_2\beta^5 + a_0b_3\beta^9 + a_0b_4\beta^{17} \\ &\quad + a_1b_0\beta^3 + a_1b_1\beta^4 + a_1b_2\beta^6 + a_1b_3\beta^{10} + a_1b_4\beta^{18} \\ &\quad + a_2b_0\beta^5 + a_2b_1\beta^6 + a_2b_2\beta^8 + a_2b_3\beta^{12} + a_2b_4\beta^{20} \\ &\quad + a_3b_0\beta^9 + a_3b_1\beta^{10} + a_3b_2\beta^{12} + a_3b_3\beta^{16} + a_3b_4\beta^{24} \\ &\quad + a_4b_0\beta^{17} + a_4b_1\beta^{18} + a_4b_2\beta^{20} + a_4b_3\beta^{24} + a_4b_4\beta^{32}. \end{aligned} \tag{17}$$

As $\beta^6 = (\beta^3)^2$, $\beta^{10} = (\beta^5)^2$, $\beta^{18} = (\beta^9)^2$, $\beta^{12} = (\beta^6)^2$, $\beta^{20} = (\beta^5)^4$, $\beta^{24} = (\beta^3)^8$, $\beta^{32} = \beta$, we can easily obtain these elements' normal basis representation by cost-free cyclic shift operation on the row of the multiplication table T and get the matrix $\lambda^{(4)}$ which leads to the function f to compute the coefficient of c_4

$$\lambda^{(4)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{18}$$

It can be readily seen that the matrices $\lambda^{(k)}$ $(0 \le k \le n-1)$ are symmetric.

From the matrix $\lambda^{(4)}$, we can get the following logic function to compute the most significant bit of the product of ab in GF(2^5) defined on the irreducible polynomial $P_1(x)$

$$\begin{aligned} c_4 &= \sum_{i=0}^{4} \sum_{j=0}^{4} \lambda_{ij}^{(4)} a_i b_j \\ &= a_0b_2 + a_2b_0 + a_0b_4 + a_4b_0 + a_1b_2 \\ &\quad + a_2b_1 + a_1b_3 + a_3b_1 + a_3b_3. \end{aligned} \tag{19}$$

In the normal basis representation, the logic function $f = (a_0, a_1, \ldots, a_{n-1}; b_0, b_1, \ldots, b_{n-1})$ which is used to get the most significant bit (c_{n-1}) of the product can also be used to get the remaining bits $(c_{n-2}, c_{n-3}, \ldots, c_0)$ of the product, except we cyclically shift the input of the function [10]. Thus, we may choose one matrix from the matrices $\lambda^{(k)}$ $(0 \le k \le n-1)$ and input the values of upper triangle of the symmetric matrix for doing the multiplication.

A VLSI array architecture to implement the versatile GF(2^n) normal basis multiplier is proposed and illustrated in Figures 1 and 2. The basic cells in the structure are 3-input

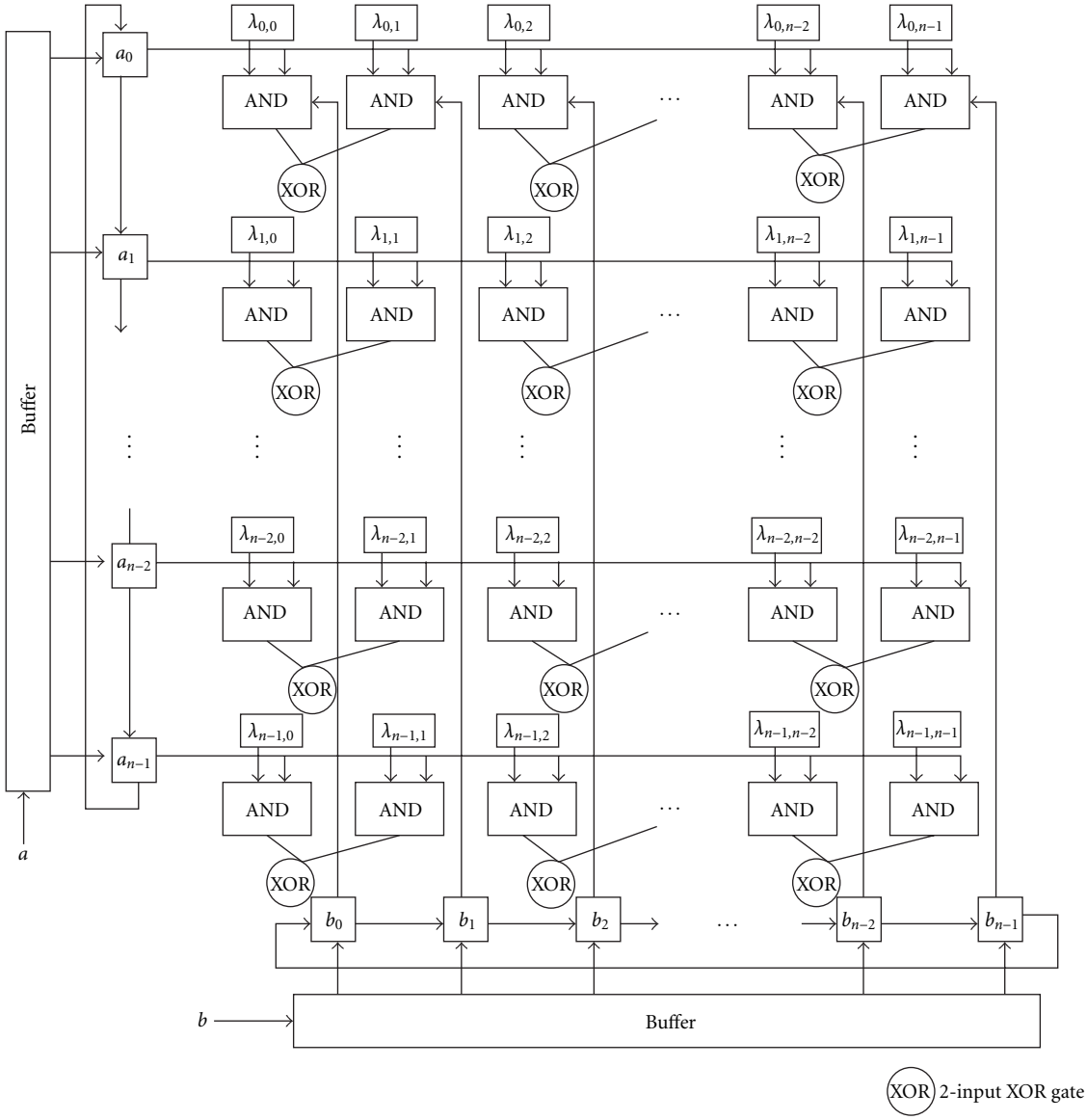

Figure 1: The logic circuit of the AND gate plane in the versatile multiplier.

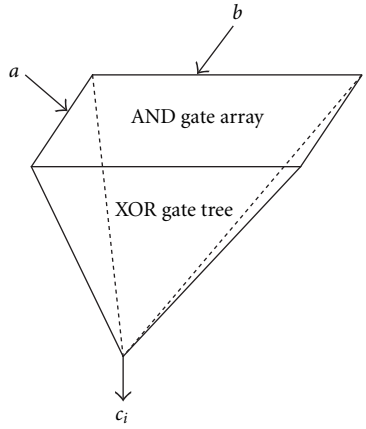

Figure 2: The architecture of the serial versatile normal basis GF(2^n) multiplier.

AND gates and 2-input XOR gates. We use the 3-input AND gates to compute $a_i b_j \lambda_{i,j}^{(n-1)}$ in the X-Y dimension, and compute the sum of $a_i b_j \lambda_{i,j}^{(n-1)}$ by a binary tree structure of 2-input XOR gates in the Z dimension. The architecture requires n^2 3-input AND gates and $n^2 - 1$ 2-input XOR gates, the time delay for generating one bit of the product is $T_{\text{AND}_3} + 2(\lceil \log_2 n \rceil) T_{\text{XOR}}$, where T_{AND_3} is the time delay of a 3-input AND gate and T_{XOR} is the time delay of a 2-input XOR gate. We can get all bits of the product by cyclically shifting the input coefficients of a and b. As the irreducible polynomial is not changed frequently as the multiplicands, we can store the elements of the matrix $\lambda^{(n-1)}$ in the registers once the irreducible polynomial has been decided.

The algorithm for this multiplication can be described as follows.

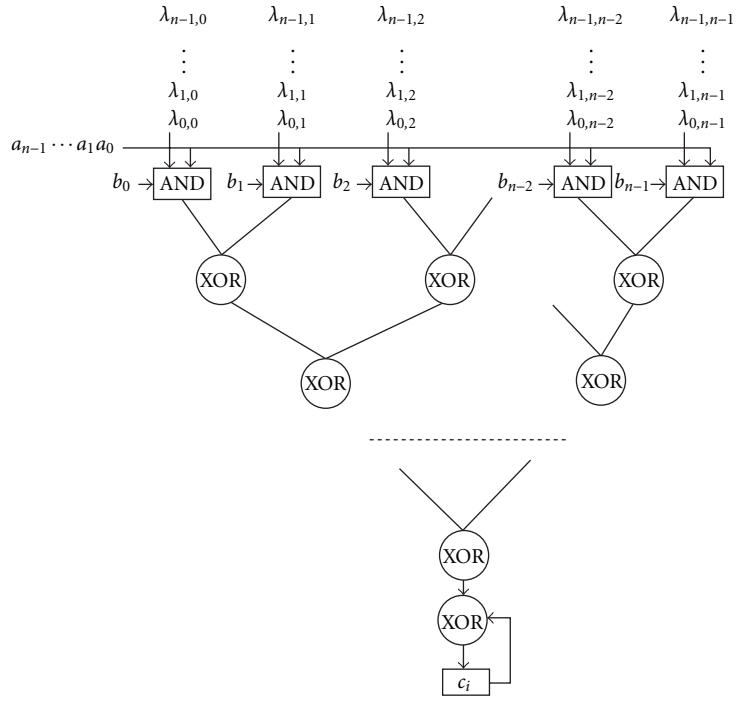

Figure 3: A low-cost architecture of the serial versatile normal basis GF(2^n) multiplier.

Algorithm 1 (versatile normal basis multiplication in GF(2^n)).

Input: Coefficients of a, b, and the matrix of $\lambda^{(n-1)}$.
Output: $c = ab$.
Begin
load matrix $\lambda^{(n-1)}$.
for $k = n - 1$ to 0 do
begin
$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \lambda_{ij}^{(n-1)}$;
cyclic shift the coefficients of a and b;
end;
End.

The proposed architecture can be implemented by a pipeline structure. In the first n clock cycles, the coefficients of a and b are fed sequentially into the buffers. In the following n clock cycles, we will get the result of the product by cyclically shifting the registers which store the original coefficients of a and b. In the meantime, the next two multiplicands can be fed into the buffers during these clock cycles and we can compute the second product immediately just after we finish the first one.

In the restricted computing environment, we can iterate using one level components of the proposed multiplier (Figure 2) to obtain a low-cost serial architecture as illustrated in Figure 3 to implement the same computation. It can be described by the following algorithm.

Algorithm 2 (low-cost serial versatile normal basis multiplication in GF(2^n)).
Input: Coefficients of a, b, and the matrix of $\lambda^{(n-1)}$.
Output: $c = ab$.
Begin
for $k = n - 1$ to 0 do
begin
$c_k^0 = 0$;
for $i = 0$ to $n - 1$
$c_k^{i+1} = c_k^i + \sum_{j=0}^{n-1} a_i b_j \lambda_{ij}^{(n-1)}$;
cyclic shift the coefficients of a and b;
end;
End.

The low-cost versatile normal basis multiplier in GF(2^n) requires n 3-input AND gates and n 2-input XOR gates. The time delay for generating one bit of the product is $n(T_{\text{AND}_3} + (\lceil \log_2 n \rceil + 1)T_{\text{XOR}})$.

The proposed versatile normal basis multipliers have modular structures, regular interconnections which are suitable for high speed or restricted space of VLSI implementations. Table 1 lists the comparison of space and time complexity between our new multipliers and previous works. The input ports of the proposed versatile multiplier are almost the same as the nonversatile multiplier, since the finite field parameters can be configured into the multiplier by the input ports of multiplicands (a and b) through a one-bit control signal at the configuration time. The finite field parameters do not need reconfiguration during the running time of the multiplier, until the application environments are changed. Thus the hardware cost can be greatly reduced compared to the nonversatile multiplier where a new multiplier has to be redesigned and implemented when the finite field parameters are required to be changed.

Table 1: Comparison of versatile multipliers with nonversatile multipliers in GF(2^n).

Multiplier	Type	# XOR Gates	# AND Gates	Time Delay
Wang-MOM [10]	Nonversatile	$2n-2$	$2n-1$	$n(T_{\text{AND}} + (\lceil \log_2 n \rceil + 1)T_{\text{XOR}})$
Li-CVM [9] (canonical basis)	Versatile	$2n^2$	$2n^2$	$n(T_{\text{AND}} + 2T_{\text{XOR}})$
Prop. multiplier (Figure 2)	Versatile	n^2-1	n^2 (3-input)	$n(T_{\text{AND}_3} + 2\lceil \log_2 n \rceil T_{\text{XOR}})$
Prop. low-cost multiplier (Figure 3)	Versatile	n	n (3-input)	$n^2(T_{\text{AND}_3} + (\lceil \log_2 n \rceil + 1)T_{\text{XOR}})$

Moreover, the proposed architecture in GF(2^n) can be easily expanded to the finite field of GF(2^{2n}). The one solution is to use two basic GF(2^n) architecture to implement the multiplication in GF(2^{2n}) and another alternative solution is to do the GF(2^{2n}) multiplication serially by using only one basic GF(2^n) architecture.

4. CONCLUSION

In this paper, the architectures for finite field multiplication based on normal basis have been proposed. The architectures require simple control signals and have regular local interconnections. As a consequence, they are very suitable for VLSI implementation. The versatile property of this VLSI array modular multiplier increases the application range and the same multiplier can be applied for different application environments, such as elliptic curve cryptosystems and Reed-Solomon encoder/decoder. The proposed multiplier can be easily extended to high order of n for more security. Moreover, the structures can be modified to make fast exponentiation and inversion. Also note that we can make a low-cost and space efficient serial multiplier which is feasible in the restricted computing environments and embedded systems.

APPENDIX

Let the irreducible polynomial be $P_1(x) = x^5 + x^4 + x^2 + x + 1$ and let β be a root of the polynomial. We show the procedures of computing the multiplication table T and the matrix $\lambda^{(4)}$.

As β is a root of the $P_1(x)$,

$$\beta^5 = \beta^4 + \beta^2 + \beta + 1, \tag{A.1}$$

$$\begin{aligned}\beta^6 &= \beta^5\beta \\ &= \beta^5 + \beta^3 + \beta^2 + \beta \\ &= \beta^4 + \beta^2 + \beta + 1 + \beta^3 + \beta^2 + \beta \\ &= \beta^4 + \beta^3 + 1.\end{aligned} \tag{A.2}$$

We multiply β^2 to both sides of (A.2), and get

$$\beta^8 = \beta^6 + \beta^5 + \beta^2. \tag{A.3}$$

From (A.3),

$$\beta^6 = \beta^8 + \beta^5 + \beta^2. \tag{A.4}$$

As

$$1 = \beta^{16} + \beta^8 + \beta^4 + \beta^2 + \beta. \tag{A.5}$$

Substitute (A.5) into (A.1),

$$\begin{aligned}\beta^5 &= \beta^4 + \beta^2 + \beta + \beta^{16} + \beta^8 + \beta^4 + \beta^2 + \beta \\ &= \beta^{16} + \beta^8.\end{aligned} \tag{A.6}$$

Substitute (A.6) into (A.4),

$$\begin{aligned}\beta^6 &= \beta^8 + \beta^5 + \beta^2 \\ &= \beta^8 + \beta^{16} + \beta^8 + \beta^2 \\ &= \beta^{16} + \beta^2.\end{aligned} \tag{A.7}$$

From (A.2), we get

$$\beta^3 = \beta^6 + \beta^4 + 1. \tag{A.8}$$

Substitute (A.7) and (A.5) into (A.8),

$$\begin{aligned}\beta^3 &= \beta^{16} + \beta^2 + \beta^4 + \beta^{16} + \beta^8 + \beta^4 + \beta^2 + \beta \\ &= \beta^8 + \beta.\end{aligned} \tag{A.9}$$

REFERENCES

[1] E. R. Berlekamp, "Bit-serial Reed-Solomon encoders," *IEEE Transactions on Information Theory*, vol. 28, no. 6, pp. 869–874, 1982.

[2] I. S. Hsu, T. K. Truong, L. J. Deutsch, and I. S. Reed, "A comparison of VLSI architecture of finite field multipliers using dual, normal, or standard bases," *IEEE Trans. on Computers*, vol. 37, no. 6, pp. 735–739, 1988.

[3] J. L. Massey and J. K. Omura, "Computational method and apparatus for finite field arithmetic," U.S. Patent application, 1981.

[4] B. A. Laws Jr. and C. K. Rushforth, "A cellular-array multiplier for $GF(2^m)$," *IEEE Trans. on Computers*, vol. 20, no. 12, pp. 1573–1578, 1971.

[5] P. A. Scott, S. E. Tarvares, and L. E. Peppard, "A fast VLSI multiplier for $GF(2^m)$," *IEEE Journal on Selected Areas in Communications*, vol. 4, pp. 62–66, January 1986.

[6] L. Song and K. Parhi, "Low-energy digit-serial/parallel finite field multipliers," *Journal of VLSI Signal Processing*, vol. 19, no. 2, pp. 149–166, 1998.

[7] S. K. Jain, L. Song, and K. K. Parhi, "Efficient semisystolic architectures for finite-field arithmetic," *IEEE Trans. on VLSI Systems*, vol. 6, no. 1, pp. 101–113, 1998.

[8] M. A. Hasan and A. G. Wassal, "VLSI algorithms, architectures and implementation of a versatile $GF(2^m)$ processor," *IEEE Trans. on Computers*, vol. 49, no. 10, pp. 1064–1073, 2000.

[9] H. Li and C. N. Zhang, "Efficient cellular automata based versatile modular multiplier for $GF(2^m)$," to appear in *Journal of Information Science and Engineering*.
[10] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in $GF(2^m)$," *IEEE Trans. on Computers*, vol. 34, no. 8, pp. 709–716, 1985.
[11] G. Feng, "A VLSI architecture for fast inversion in $GF(2^m)$," *IEEE Trans. on Computers*, vol. 38, no. 10, pp. 1383–1386, 1989.
[12] A. J. Menezes, *Applications of Finite Fields*, Kluwer Academic Publishers, Boston, Mass, USA, 1993.

Hua Li received his B.E. and M.S. degrees from Beijing Polytechnic University and Peking University. He is a Ph.D. candidate in the Department of Computer Science, University of Regina. Currently, he works as an assistant professor at Department of Mathematics and Computer Science, University of Lethbridge, Canada. His research interests include parallel systems, reconfigurable computing, fault-tolerant, VLSI design, and information and network security. He is a member of IEEE.

Chang Nian Zhang received his B.S. degree in applied mathematics from University of Science Technology, China, and the Ph.D. degree in computer science and engineering from Southern Methodist University. In 1998, he joined Concordia University as a research assistant professor in Department of Computer Science. Since 1990, he has been with University of Regina, Canada, in Department of Computer Science. Currently he is a full professor and leads a research group in parallel processing, data security, and neural networks.