A Comparison of VLSI Architecture of Finite Field Multipliers Using Dual, Normal or Standard Basis

I. S. Hsu, T. K. Truong, H. M. Shao, and L. J. Deutsch
Communications Systems Research Section

I. S. Reed
Department of Electrical Engineering
University of Southern California

Three different finite field multipliers are presented: (1) a dual basis multiplier due to Berlekamp, (2) the Massey-Omura normal basis multiplier, and (3) the Scott-Tavares-Peppard standard basis multiplier. These algorithms are chosen because each has its own distinct features which apply most suitably in different areas. Finally, they are implemented on silicon chips with NMOS technology so that the multiplier most desirable for VLSI implementations can readily be ascertained.

I. Introduction

The era of the VLSI digital signal processor has arrived and its impact is evident in many areas of technology. The trend is to put more and more elements on a single silicon chip in order to enhance the performance and reliability of the system.

Recently, finite field arithmetic has found widespread applications. Examples include cryptography, coding theory, and computer arithmetic. Among the finite field arithmetic operations, multiplication is the most complex and time consuming. Already, it is used in a variety of systems, e.g., the VLSI design of Reed-Solomon coder (Refs. 1 and 2). Hence, a small, high performance finite field multiplier is urgently needed. Such a multiplier can also be used as a building block for the design of many large systems which use finite field arithmetic.

In this article, three different finite field multipliers are compared for suitability of VLSI implementation. These include: (1) the dual basis multiplier due to Berlekamp (Ref. 3), (2) the Massey-Omura normal basis multiplier (Ref. 4), and (3) the Scott-Tavares-Peppard standard basis multiplier (Ref. 5). They are chosen for comparison because each has its own distinct features which make them suitable for specific applications.

Different basis representations of field elements are used in these three multipliers. The dual basis multiplier uses the dual basis representation for the multiplicand and standard basis for the multiplier. The product is again in dual basis representation. The Massey-Omura multiplier uses normal basis representations for both the multiplicand and multiplier. The Scott-Tavares-Peppard multiplier uses the standard basis representations for all field elements. The complexity of basis conversion

is heavily dependent on the choice of the primitive irreducible polynomial which generates the field. If the polynomial is chosen adequately, the basis conversion is a simple operation. The algorithms for performing the basis conversions are presented in Appendixes A and B.

It is concluded in this article that the dual basis multiplier needs the least number of gates, which in turn leads to the smallest area required for VLSI implementation. The Massey-Omura multiplier is very effective in performing operations such as finding inverse elements or in performing squaring or exponentiation of a finite field element. The standard basis multiplier does not require basis conversion; hence it is readily matched to any input or output systems. Also, due to its regularity and simplicity, the design and expansion to high order finite fields is easier to realize than in the dual or normal basis multipliers.

Examples of these three multipliers are given in this article for the purpose of illustration. Their 8-bit versions are implemented on a silicon chip. The chip layouts are also presented separately so that their differences can be distinguished without difficulty.

II. The Dual Basis Multiplication Algorithm

Recently, Berlekamp developed a bit-serial multiplier for use in the design of a Reed-Solomon encoder (Ref. 3). Hsu et al. (Ref. 1) used Berlekamp's multiplier to design an 8-bit single chip VLSI (255, 223) Reed-Solomon encoder which has proved to perform well. However, in that design, the multiplicand is a fixed finite field constant which is inconvenient if one desires to change the multiplicand.

In the following, Berlekamp's bit-serial multiplication algorithm is modified and generalized to allow both the multiplicand and multiplier to be variable. Thus, revision of Berlekamp's algorithm is called the dual basis multiplication algorithm in the rest of this article.

In order to understand the dual basis multiplication algorithm, some mathematical preliminaries are needed. Toward this end, the mathematical concepts of the "trace" and a "dual" basis are introduced. For more details and proofs see Refs. 2, 6, and 7.

Definition 1. The trace of an element β belonging to $GF(p^m)$, the Galois field, of p^m elements, is defined as follows:

$$\mathrm{Tr}(\beta) = \sum_{k=0}^{m-1} \beta^{p^k}$$

In particular, for $p = 2$,

$$\mathrm{Tr}(\beta) = \sum_{k=0}^{m-1} \beta^{2^k}$$

A fast algorithm for computing trace values of elements in $GF(2^m)$ is presented in Appendix C. The trace has the following properties which will not be proved here:

(1) $[\mathrm{Tr}(\beta)]^p = \beta^p + \beta^{p^2} + \cdots + \beta^{p^{m-1}} = \mathrm{Tr}(\beta)$

where $\beta \in GF(p^m)$. This implies that $\mathrm{Tr}(\beta) \in GF(p)$, i.e., the trace is in the ground field $GF(p)$.

(2) $\mathrm{Tr}(\beta + \gamma) = \mathrm{Tr}(\beta) + \mathrm{Tr}(\gamma)$, where $\beta, \gamma \in GF(p)$.

(3) $\mathrm{Tr}(c\beta) = c\mathrm{Tr}(\beta)$, where $c \in GF(p)$

(4) $\mathrm{Tr}(1) = m \pmod p$

Definition 2. A basis $\{u_k\}$ in $GF(p^m)$ is a set of m linearly independent elements in $GF(p^m)$.

Definition 3. Two bases $\{u_j\}$ and $\{\lambda_k\}$ are said to be the dual of one another if

$$\mathrm{Tr}(u_j\lambda_k) = \begin{cases} 1, \text{if } j = k \\ 0, \text{if } j \neq k \end{cases}$$

For convenience, the basis $\{u_j\}$ is sometimes called the original basis, and the basis $\{\lambda_k\}$ is called its dual basis, even though the concept of duality is symmetric.

Theorem. Every basis has a unique dual basis.

Proof. See Ref. 8.

Corollary 1. Let $\{u_j\}$ be a basis of $GF(p^m)$ and let $\{\lambda_k\}$ be its dual basis. Then a field element Z can be expressed in the dual basis $\{\lambda_k\}$ by the expansion

$$Z = \sum_{k=0}^{m-1} z_k\lambda_k$$

where

$$z_k = \mathrm{Tr}(Z \cdot u_k)$$

Proof. Let

$$Z = z_0\lambda_0 + z_1\lambda_1 + \cdots + z_{m-1}\lambda_{m-1}$$

Multiply both sides by u_k and take the trace. Then by Definition 3 and the property of trace:

$$\mathrm{Tr}(Z \cdot u_k) = \mathrm{Tr} \sum_{i=0}^{m-1} z_i (\lambda_i u_k) = z_k$$

The following corollary is an immediate consequence of Corollary 1.

Corollary 2. Let $\{u_j\}$ be a basis of $GF(p^m)$ and let $\{\lambda_k\}$ be its dual basis. The product $W = ZG$ of two fixed elements in $GF(p^m)$ can be expressed in the dual basis by the expansion

$$W = \sum_{k=0}^{m-1} \mathrm{Tr}(Wu_k) \cdot \lambda_k = \sum_{k=0}^{m-1} \mathrm{Tr}(ZGu_k) \lambda_k$$

where $\mathrm{Tr}(Wu_k)$ is the kth coefficient of the dual basis for the product of two field elements.

These two corollaries provide a theoretical basis for the dual basis finite field multiplier. In the following section, a detailed example is developed to illustrate the dual basis bit-serial multiplication algorithm.

III. An Example of the Dual Basis Multiplication Algorithm

The example is given in $GF(2^4)$ for purposes of illustration; the extension to more general cases is obvious.

Let α be a root of the primative irreducible polynomial $f(x) = x^4 + x + 1$ over $GF(2)$. Then α satisfies the equation $\alpha^4 + \alpha + 1 = 0$. It is also true that $\alpha^{15} = 1$. Let the standard basis be $\{1, \alpha, \alpha^2, \alpha^3\}$ and its dual basis be $\{\lambda_0, \lambda_1, \lambda_2, \lambda_3\}$. Then, by Definition 3,

$$\mathrm{Tr}(1 \cdot \lambda_0) = 1$$

$$\mathrm{Tr}(\alpha \cdot \lambda_1) = 1$$

$$\mathrm{Tr}(\alpha^2 \cdot \lambda_2) = 1$$

and

$$\mathrm{Tr}(\alpha^3 \cdot \lambda_3) = 1$$

Let

$$Z = \sum_{k=0}^{3} z_k \lambda_k$$

be represented in dual basis. Also let

$$G = \sum_{k=0}^{3} g_k u_k$$

be represented in standard basis. Then, by Corollary 1,

$$z_k = \mathrm{Tr}(Z\alpha^k)$$

Furthermore, let $W = ZG$ be the product of two elements Z and G. If W is represented in dual basis, then,

$$W = \sum_{k=0}^{3} \omega_k \cdot \lambda_k$$

where by Corollary 1,

$$\omega_k = \mathrm{Tr}(W\lambda_k) = \mathrm{Tr}(ZG \cdot \lambda_k)$$

If one defines

$$T^{(k)}(W) = \mathrm{Tr}(ZG \cdot \lambda_k)$$

then

$$\begin{aligned} T^{(0)}(W) &= \mathrm{Tr}(ZG\alpha^0) = \mathrm{Tr}(ZG) \\ &= \mathrm{Tr}(Z(g_0 \cdot \alpha^0 + g_1 \cdot \alpha + g_2 \cdot \alpha^2 + g_3 \cdot \alpha^3)) \\ &= g_0 z_0 + g_1 z_1 + g_2 z_2 + g_3 z_3 \end{aligned} \quad (1)$$

From the definition of $T^{(k)}(W)$, one obtains

$$T^{(k)}(W) = T^{(k-1)}(\alpha ZG \cdot \alpha^{k-1})$$

Therefore, if ZG is replaced by $\alpha \cdot ZG$ in $T^{(k-1)}$, i.e., Z by $\alpha \cdot Z$, $T^{(k)}(W)$ can be obtained from $T^{(k-1)}(W)$. Let

$$Y = \alpha Z = y_0\lambda_0 + y_1\lambda_1 + y_2\lambda_2 + y_3\lambda_3$$

where

$$y_m = \mathrm{Tr}(Y \cdot \alpha^m) = \mathrm{Tr}(Z \cdot \alpha^{m+1})$$

for each m. Then $T^{(k)}$ is obtained from $T^{(k-1)}$ by replacing

$$z_0 \text{ by } y_0 = \mathrm{Tr}(Z\alpha) = z_1$$

$$z_1 \text{ by } y_1 = \mathrm{Tr}(Z\alpha^2) = z_2$$

$$z_2 \text{ by } y_2 = \mathrm{Tr}(Z\alpha^3) = z_3$$

and

$$z_3 \text{ by } y_3 = \mathrm{Tr}(Z\alpha^4) = z_0 + z_1$$

To reiterate,

$$W = ZG = \sum_{k=0}^{3} T^{(k)}(W)\lambda_k$$

can be computed as follows:

(1) Initially for $k = 0$, compute

$$T^{(0)}(W) \text{ by Eq. (1).}$$

(2) For $k = 1, 2, 3$, compute $T^{(k)}(W)$ by

$$T^{(k-1)}(W) = T^{(k-1)}(YG)$$

and

$$Y = \alpha Z = y_0\lambda_0 + y_1\lambda_1 + y_2\lambda_2 + y_3\lambda_3$$

with

$$y_0 = z_1, y_1 = z_2, y_2 = z_3, \text{ and } y_3 = z_0 + z_1 = Z_f$$

where $Z_f = z_0 + z_1$ is the feedback term of the algorithm.

The above algorithm illustrates the dual basis multiplication algorithm. The extension to an 8-bit multiplier is obvious and will not be included here. The primitive irreducible polynomial in the 8-bit design is chosen to be

$$f(x) = x^8 + x^4 + x^3 + x^2 + 1$$

In this case, the feedback term is

$$Z_f = z_4 + z_3 + z_2 + z_0$$

Figure 1 shows the logic diagram of an 8-bit dual basis multiplier. The architecture is composed of four blocks. In Fig. 1, the Z-serial-to-parallel unit performs the serial-to-parallel operation of the input element with dual basis representation. Once all 8 bits of this element are stored in the bottom register, the cyclic operation starts and one bit is fed back from the feedback logic circuitry. The G-serial-to-parallel unit also performs the serial-to-parallel operation of the input standard basis element. Once all 8 bits of this element are stored in the bottom register, they are latched bitwise so that no further operations are performed on this element as required by the algorithm.

Next, the output bits of these two units are fed into the AND-generation unit. The output consists of the bitwise AND-ed terms. These AND-ed terms again are fed into the XOR-array unit which performs the addition of AND-ed terms. This is needed since the addition of two elements in $GF(2)$ is just an exclusive-OR (XOR) operation. The terms included in this XOR-array are as shown in the following:

$$z_0g_0 + z_1g_1 + z_2g_2 + z_3g_3 + z_4g_4 + z_5g_5 + z_6g_6 + z_7g_7$$

The product is then obtained from the output of this XOR-array bit by bit. Figure 2 shows the layout of this dual basis multiplier.

IV. VLSI Architecture for the Massey-Omura Multiplier

Recently, Massey and Omura developed a multiplier which obtains the product of two elements in the finite field $GF(2^m)$. In this invention, they utilize a normal basis of form $\{\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{m-1}}\}$ to represent each element in the field, where α is the root of an irreducible polynomial of degree m over $GF(2)$. In this basis, each element in the field $GF(2^m)$ can be represented by m binary digits.

Using the normal basis representation, the squaring of an element in $GF(2^m)$ is readily shown to be a simple cyclic shift of its binary digits (Ref. 4). Multiplication of two elements with a normal basis representation requires the same logic circuitry for every product digit. Adjacent product digit circuits differ only in their inputs which are cyclically shifted versions of one another (Ref. 4).

The conventional method for finding an inverse element in a finite field uses either table look-up or Euclid's algorithm. These methods are not easily realized in a VLSI circuit. However, by using a Massey-Omura multiplier, a recursive, pipeline, inversion circuit can be developed (Ref. 4). The details of the Massey-Omura multiplier algorithm are not discussed further here. For a more detailed discussion, see Ref. 4.

The function f as described in (Ref. 4) is chosen to be the following:

$$f(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7; b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) =$$

$$a_5b_0 + a_6b_0 + a_3b_1 + a_5b_1 + a_4b_2 + a_5b_2 + a_6b_2 + a_7b_2$$

$$+ a_1b_3 + a_4b_3 + a_2b_4 + a_3b_4 + a_0b_5 + a_1b_1 + a_2b_5$$

$$+ a_6b_5 + a_0b_6 + a_2b_6 + a_5b_6 + a_6b_6 + a_2b_7$$

where the primitive irreducible polynomial of this finite field is

$$g(x) = x^8 + x^7 + x^6 + x^1 + 1$$

There are a variety of different possible expressions for the function f; however, the above one was chosen for the purpose of illustration.[1] Since each term in the above expression represents a conducting line in the AND portion of a PLA (programmable logic array), the fewer the number of terms there are, the smaller the area needed to be used for VLSI implementation.

Figure 3 shows the block diagram of an 8-bit finite field multiplier using the Massey-Omura normal basis algorithm. The architecture of this chip is identical to that of the dual basis multiplier. The differences between these two multipliers are the following:

(1) The number of terms in the expression of the Massey-Omura multiplier is twenty-one, while in the dual basis multiplier it is only eleven. This means a substantial amount of area is saved in the dual basis multiplier over the normal basis multiplier.

(2) Both the input serial-to-parallel units are identical in the Massey-Omura multiplier and no feedback is needed. On the other hand, in the dual basis multiplier, the register storing the element with standard basis representation does not need to be cyclically shifted. This field element remains latched in the same position.

Figure 4 shows the layout for the 8-bit Massey-Omura multiplier.

V. VLSI Architecture for the Standard Basis Multiplier

The Scott-Tavares-Peppard multiplication algorithm is serial-in, serial-out, and pipeline in architecture. This algorithm performs multiplication in $GF(2^m)$ with order $0(m)$ in both computation time and implementation area, but requires $m + 1$ time units between the first-in and first-out of computation. Due to the regularity of this architecture, the expansion to higher order finite fields needs only replicas of a basic cell. Furthermore, the irreducible primitive polynomial which generates the finite field can be changed. This feature makes it more convenient in use. This algorithm performs the finite field multiplication with elements represented in standard basis. As a consequence no basis conversion is needed. This multiplier can be used for applications such as crypotography where m is large. The algorithm is advantageous because of its efficient implementation time and high throughputs. The detailed algorithm will not be discussed further here. For more details, see Ref. 5.

Figure 5 shows the logic diagram of an 8-bit standard basis multiplier by Scott, Tavares, and Peppard. Inputs to this chip are A and B, the two elements to be multiplied, and the irreducible primitive polynomial F. These are fed into the chip serially. The output is the product element P, which is shifted out bit-by-bit.

In Fig. 5, A and F are shifted into their respective registers serially bit-by-bit. Here A is the multiplicand and F is the primitive irreducible polynomial that generates the finite field. The multiplier is denoted by B and the product by P. The register P_i contains the immediate product. Two control signals are required. One is derived from the most significant bit (MSB) of P, and the other from the state of b_i, which is latched with a flip-flop. The left shift is performed by loading the output of cell CELL-i into the product register of cell CELL-i + 1. Once the multiplication is completed, the most significant bits of the product register are transferred to the output shift register and shifted out serially.

The circuit diagram of the ith cell CELL-i is shown in Fig. 6. Since the ground field is $GF(2)$, additions are performed by exclusive-OR (XOR) gates. Pass transistors are used to control the data flow. If a "0" is to be added, the input line to the XOR gate is grounded; otherwise A and/or F are passed. The output of the XOR gate is directed to the product register of the next stage so adding and shifting is done in one clock cycle. Figure 7 shows the layout of this 8-bit standard basis finite field multiplier.

VI. Concluding Remarks

Three finite field multipliers are compared here. They are dual basis multiplier, normal basis multiplier, and standard basis multiplier. The dual basis multiplier occupies the smallest amount of chip area in VLSI implementation if the basis conversion is not included. Furthermore, since the dual basis multi-

[1] Wang, C. C., "Computer Simulation of Finite Field Multiplications Based on Massey-Omura's Normal Basis Representation of Field Elements," private communication, 1985.

plier performs multiplication by taking the inner product of two elements and then feeds back the sum of certain bits of one element, it is expected that as the order of field goes higher, the dual basis multiplier will outperform the others. The normal basis multiplier is very effective in performing operations such as finding the inverse element or in performing squaring or exponentiation of a finite field element. But the area grows dramatically as the order of field goes up. Also, the f function described in Ref. 4 is to be searched again by computer as the field is changed, and it is usually very time consuming. The standard basis does not require basis conversion; hence it is readily matched to any input or output system. Also, due to its regularity and simplicity, the design and expansion to high order finite fields are easier to realize than the dual or normal basis multipliers. The irreducible primitive polynomial of the field is changeable in standard basis multiplier. This distinct feature makes it more useful in certain aspects.

References

1. Hsu, I. S., Reed, I. S., Truong, T. K., Wang, K., Yeh, C. S., and Deutsch, L. J., "The VLSI Implementation of a Reed-Solomon Encoder Using Berlekamp's Bit-Serial Multiplier Algorithm," *IEEE Trans. on Computers*, Vol. C-33, No. 10, pp. 906–911, Oct. 1984.
2. Shao, H. M., Truong, T. K., Deutsch, L. J., Yuen, J. H., and Reed, I. S., "A VLSI Design of a Pipeline Reed-Solomon Decoder," *IEEE Trans. on Computers*, Vol. C-34, No. 5, pp. 393–403, May 1985.
3. Berlekamp, E. R., "Bit-Serial Reed-Solomon Encoders," *IEEE Trans. Inform. Theory*, Vol. IT-28, No. 6, pp. 869–874, Nov. 1982.
4. Wang, C. C., Truong, T. K., Shao, H. M., Deutsch, L. J., Omura, J. K., and Reed, I. S., "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Trans. on Computers*, Vol. C-34, No. 8, pp. 709–717, Aug. 1985.
5. Scott, P. A., Tarvares, S. E., and Peppard, L. E., "A Fast Multiplier for $GF(2^m)$," submitted to *IEEE Trans. on Computers*, 1985.
6. MacWilliams, F. J., and Sloane, N. J. A., *The Theory of Error-Correcting Codes*, North-Holland Publishing Company, New York, N.Y., 1978.
7. Perlman, M., and Lee, J. J., "A Comparison of Conventional Reed-Solomon Encoders and Berlekamp's Architecture," NASA Tech. Brief, No. 3610-81-119, Jet Propulsion Laboratory, Pasadena, Calif., July 10, 1981.
8. Hsu, I. S., "New VLSI Architectures for Coding and Digital Signal Processing," Ph.D. Dissertation, Electrical Engineering Dept., University of Southern California, Los Angeles, Calif., 1985.

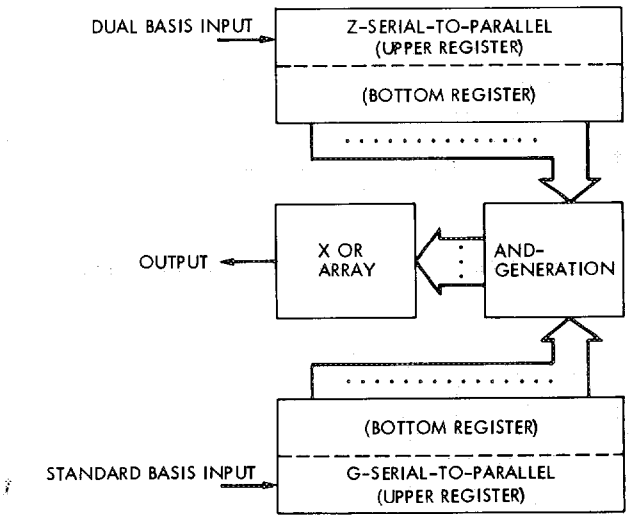

Fig. 1. Logic diagram of an 8-bit dual basis finite field multiplier

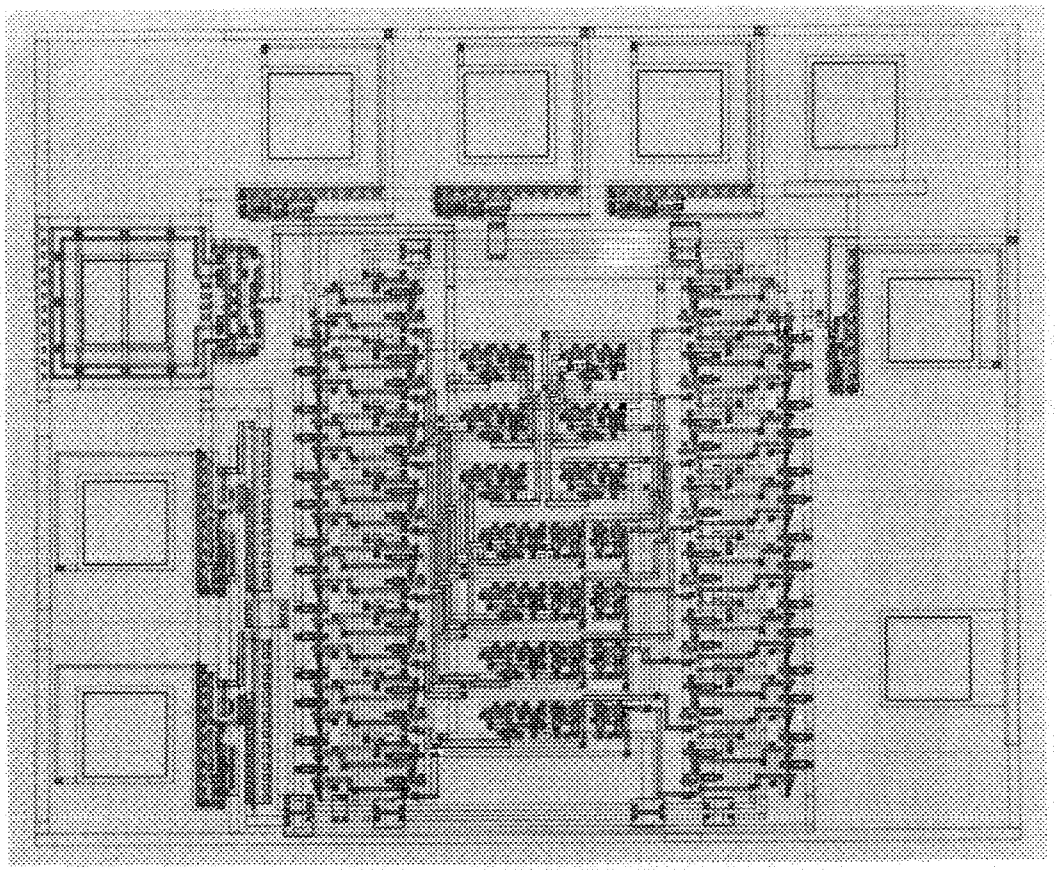

Fig. 2. Layout of an 8-bit dual basis finite field multiplier

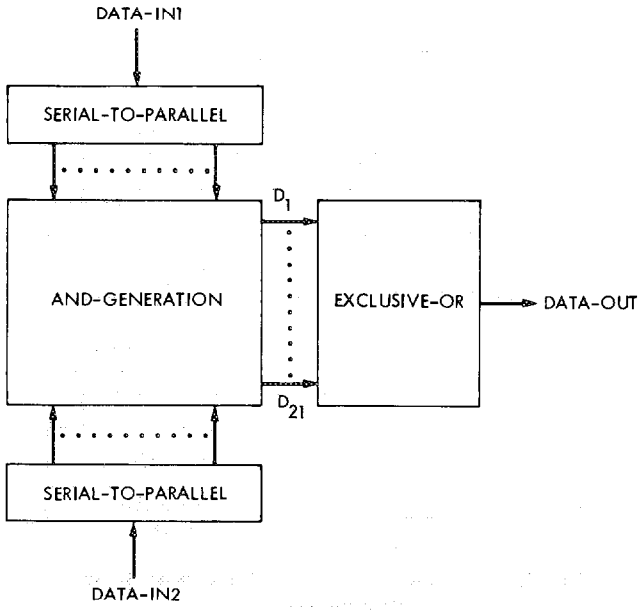

Fig. 3. Block diagram of an 8-bit finite field multiplier using Massey-Omura's normal basis algorithm

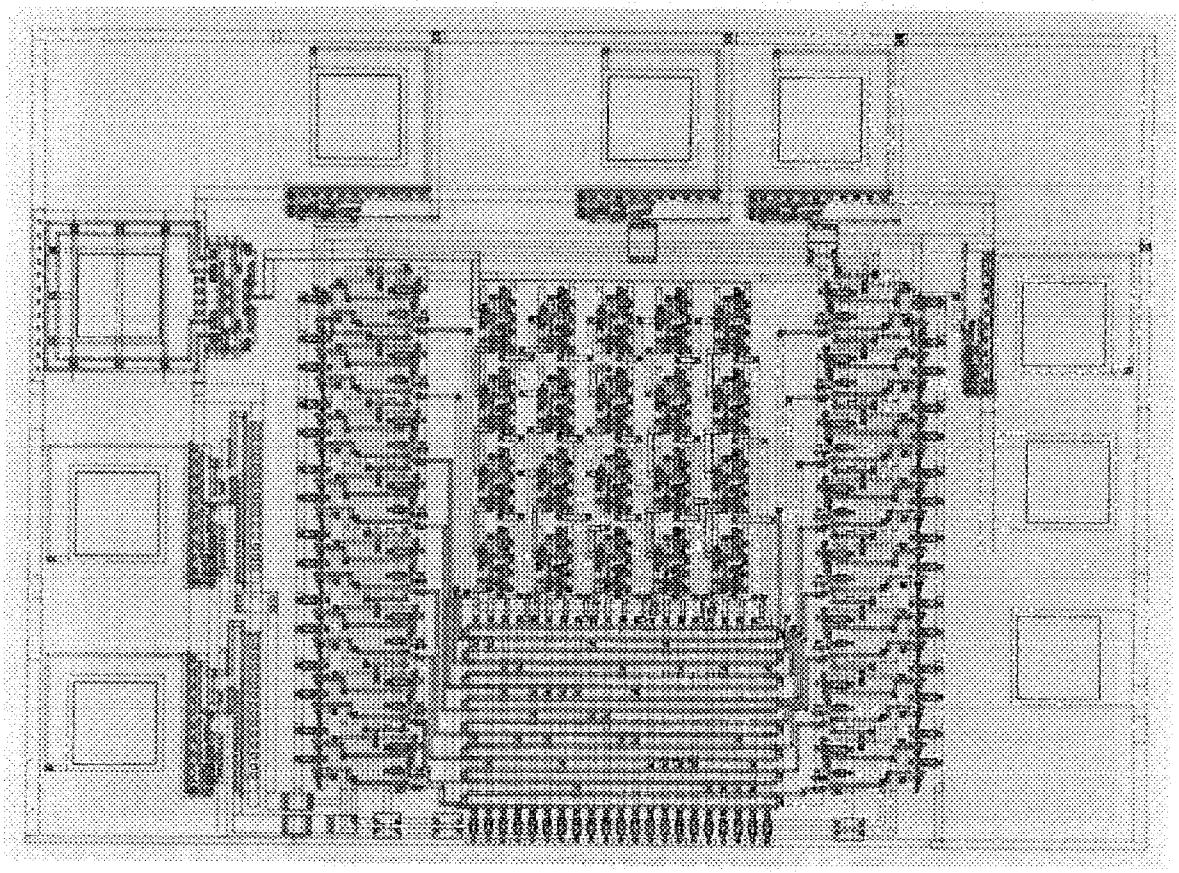

Fig. 4. Layout of an 8-bit Massey-Omura finite field multiplier

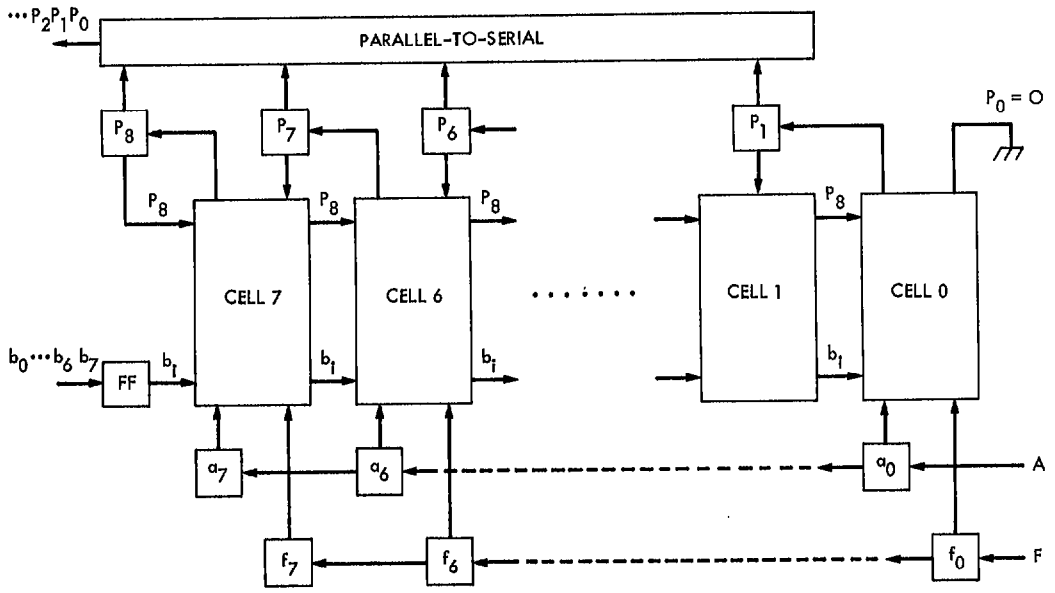

Fig. 5. Logic diagram of an 8-bit standard basis finite multiplier

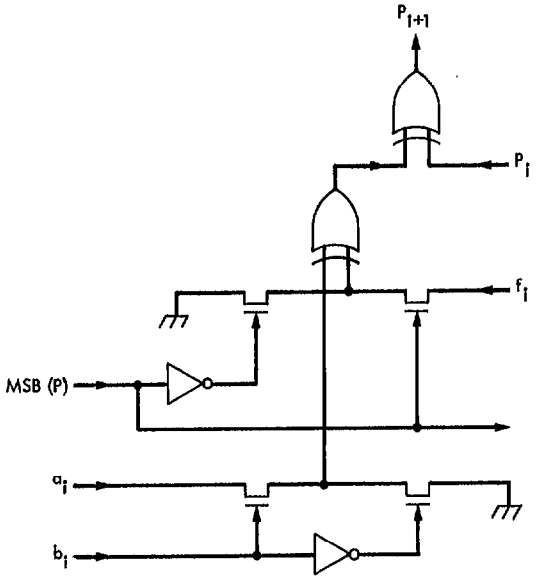

Fig. 6. Circuit diagram of the *i*th cell as shown in Fig. 5

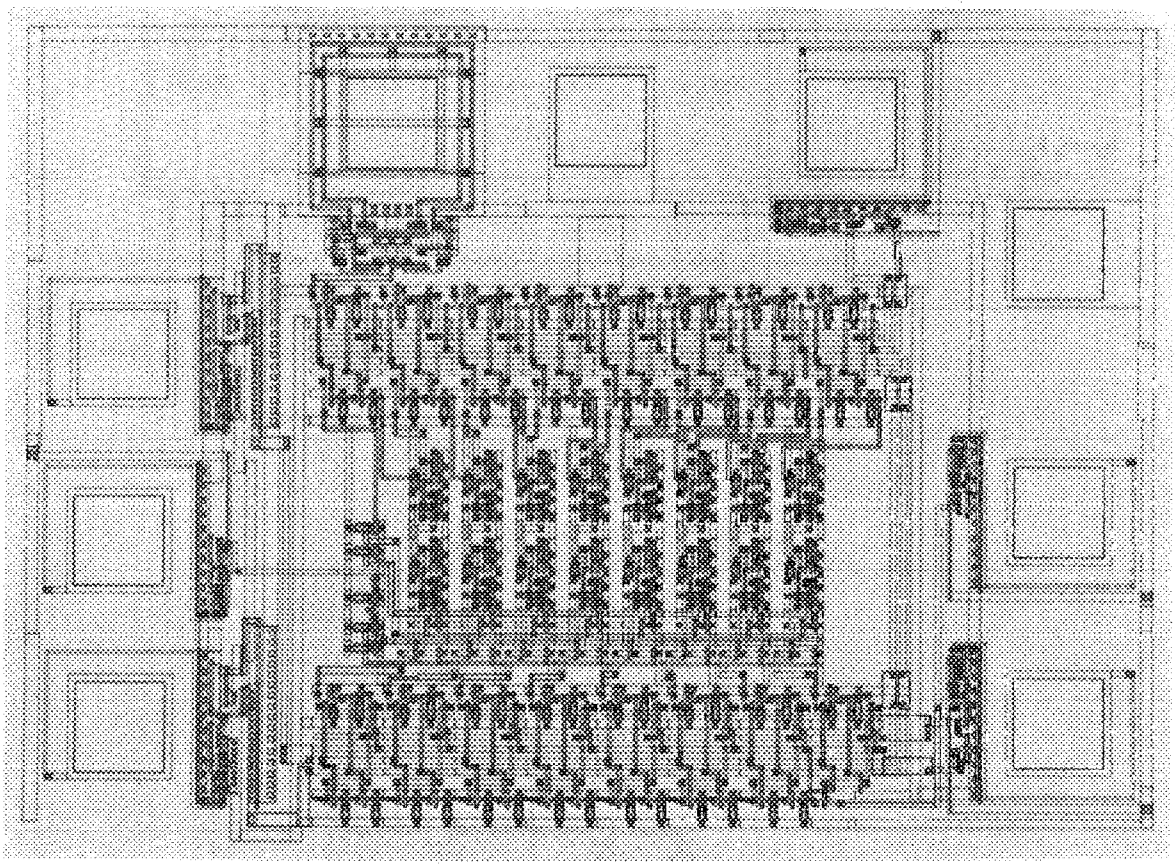

Fig. 7. Layout of an 8-bit standard basis finite field multiplier

Appendix A

A Method for Converting an Element in Standard Basis to Dual Basis

In this appendix, a method for converting an element represented in standard basis to its counterpart in dual basis is described by example. First, let the irreducible primitive polynomial in $GF(2^8)$ be

$$f(X) = x^8 + x^4 + x^3 + x^2 + 1$$

Then, from the definition of trace, one obtains

$$\mathrm{Tr}(1) = 0, \mathrm{Tr}(\alpha) = 0, \mathrm{Tr}(\alpha^2) = 0, \mathrm{Tr}(\alpha^3) = 0,$$

$$\mathrm{Tr}(\alpha^4) = 0, \mathrm{Tr}(\alpha^5) = 1, \mathrm{Tr}(\alpha^6) = 0, \mathrm{Tr}(\alpha^7) = 0$$

where α satisfies the equation $x^8 + x^4 + x^3 + x^2 + 1 = 0$.

An element Z in standard basis is written as

$$Z = \sum_{k=0}^{7} z_k \alpha^k$$

In dual basis, it is represented as

$$Z = \sum_{k=0}^{7} z'_k \lambda_k$$

where

$$\begin{aligned} z'_k &= \mathrm{Tr}(Z\alpha^k) \\ &= \mathrm{Tr}((z_0\alpha^0 + z_1\alpha^1 + z_2\alpha^2 + z_3\alpha^3 \\ &\quad + z_4\alpha^4 + z_5\alpha^5 + z_6\alpha^6 + z_7\alpha^7)\alpha^k) \\ &= z_0 \mathrm{Tr}(\alpha^k) + z_1 \mathrm{Tr}(\alpha^{k+1}) + z_2 \mathrm{Tr}(\alpha^{k+2}) \\ &\quad + z_3 \mathrm{Tr}(\alpha^{k+3}) + z_4 \mathrm{Tr}(\alpha^{k+4}) + z_5 \mathrm{Tr}(\alpha^{k+5}) \\ &\quad + z_6 \mathrm{Tr}(\alpha^{k+6}) + z_7 \mathrm{Tr}(\alpha^{k+7}) \end{aligned}$$

Therefore, once $\mathrm{Tr}(\alpha^k)$, for $0 \leqslant k \leqslant 14$, are known, the basis conversion from standard to dual can be completed.

Appendix B

A Method for Converting an Element in Dual Basis to Standard Basis

This appendix describes a method for converting an element represented in dual basis to standard basis. Again, let an element Z in dual basis be written as

$$Z = \sum_{k=0}^{7} z'_k \lambda_k$$

In standard basis let it be represented as

$$Z = \sum_{k=0}^{7} z_k \alpha^k$$

From the definition of the trace, one obtains

$$\begin{aligned} z_k &= \mathrm{Tr}(Z\lambda_k) \\ &= \mathrm{Tr}((z'_0\lambda_0 + z'_1\lambda_1 + z'_2\lambda_2 + z'_3\lambda_3 \\ &\quad + z'_4\lambda_4 + z'_5\lambda_5 + z'_6\lambda_6 + z'_7\lambda_7)\lambda_k) \\ &= z'_0 \mathrm{Tr}(\lambda_0\lambda_k) + z'_1 \mathrm{Tr}(\lambda_1\lambda_k) + z'_2 \mathrm{Tr}(\lambda_2\lambda_k) \\ &\quad + z'_3 \mathrm{Tr}(\lambda_3\lambda_k) + z'_4 \mathrm{Tr}(\lambda_4\lambda_k) + z'_5 \mathrm{Tr}(\lambda_5\lambda_k) \\ &\quad + z'_6 \mathrm{Tr}(\lambda_6\lambda_k) + z'_7 \mathrm{Tr}(\lambda_7\lambda_k) \end{aligned}$$

Hence, if the dual basis is determined and the trace values of the above calculated, the basis conversion from dual basis to standard basis can be completed.

Appendix C

Fast Algorithm for Calculating Trace Values of Elements in $GF(2^m)$

In this appendix, a fast algorithm for calculating the trace values of elements in finite field $GF(2^m)$ is described. From the definition of the trace one has, for $\beta, \beta^2 \in GF(2^m)$:

$$\mathrm{Tr}(\beta^2) = \sum_{k=0}^{m-1} (\beta^2)^{2^k} = \beta^2 + \beta^{2^2} + \cdots + \beta^{2^m} = \beta + \beta^2 + \cdots + \beta^{2^{m-1}} = \mathrm{Tr}(\beta)$$

Hence, if $\mathrm{Tr}(\beta)$ is obtained, then $\mathrm{Tr}(\beta^2)$ can also be obtained without calculation.

Since every element in $GF(2^m)$ can be represented by the elements which compose the basis, i.e., for $GF(2^m)$, and the basis is $\{\alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7\}$, then β can be written as

$$\beta = \beta_0\alpha^0 + \beta_1\alpha^1 + \beta_2\alpha^2 + \beta_3\alpha^3 + \beta_4\alpha^4 + \beta_5\alpha^5 + \beta_6\alpha^6 + \beta_7\alpha^7$$

From the properties of the trace, one has

$$\mathrm{Tr}(\beta) = \beta_0\,\mathrm{Tr}(\alpha^0) + \beta_1\,\mathrm{Tr}(\alpha^1) + \beta_2\,\mathrm{Tr}(\alpha^2) + \beta_3\,\mathrm{Tr}(\alpha^3) + \beta_4\,\mathrm{Tr}(\alpha^4) + \beta_5\,\mathrm{Tr}(\alpha^5) + \beta_6\,\mathrm{Tr}(\alpha^6) + \beta_7\,\mathrm{Tr}(\alpha^7)$$

Hence it is only necessary to calculate trace values of $\alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7$, the rest can be obtained easily once it is represented by the basis elements.