REVIEW PAPER: BIT-Serial Multiplier Techniques for Finite Fields

MR. HARDIK PUNAMCHAND SUTARIA

M.E [Electronics and Communication] Student, Department of Electronics and Communication, L.J.E.I.T, Ahmedabad, Gujarat

Sutariahardik_bvm@yahoo.co.in

Abstract-This Paper gives review of different types of Bit-serial multiplier techniques for Finite Fields. Based on this Review we recommended general method for Bit-serial multiplier. These multipliers are for the Finite Fields and they are represented by polynomial, dual and normal basis so, they are reviewed in this paper. Comparison of Berlekamp, Massey-Omura and Polynomial Basis Multipliers are done finally.

Keywords– Finite Fields, Polynomial Basis, Dual Basis, Normal Basis, Berlekamp Multiplier, Massey-omura Multiplier, Polynomial Basis Multiplier, Galois Fields.

I. INTRODUCTION

The most commonly implemented finite field operations are multiplication and addition. Multiplication is considered to be a degree of magnitude more complicated than addition and a large body of research has been carried out attempting to reduce the hardware and time complexities of multiplication. Finite field adders and multipliers can be classified according to whether they are bit-serial or bit-parallel, that is whether the m bits representing field elements are processed in series or in parallel. Whereas bit-serial multipliers generally require less hardware than bit-parallel multipliers, they also usually require m clock cycles to generate a product rather than one. Hence in time critical applications bit-parallel multipliers must be implemented, in spite of the increased hardware overheads.

A. What is Finite Fields?

Error control codes rely to a large extent on powerful and elegant algebraic structures called finite fields. A field is essentially a set of elements in which it is possible to add, subtract, multiply and divide field elements and always obtain another element within the set. A finite field is a field containing a finite number of elements. A well-known example of a field is the infinite field of real numbers.

B. Field Definition

The concept of a field is now more formally introduced. A field F is a non-empty set of elements with two operations usually called addition and multiplication, denoted ‘+’ and ‘*’ respectively. For F to be a field a number of conditions must hold [3,7]:

1. Closure: For every a, b in F, c = a + b; d = a * b;                                           (1)

   Where c, d ∈ F.

2. Associative: For every a, b, c in F, a + (b + c) = (a + b) + c and a * (b * c) = (a * b) * c. (2)

3. Identity: There exists an identity element ‘0’ for addition and ‘1’ for multiplication that satisfy 0 + a = a 0 + 0 = 0 and a * 1 = 1 * a = a. (3)

   For every a in F.

4. Inverse: If a is in F, there exist elements b and c in F such that a + b = 0, a * c = 1. (4)

   Element b is called the additive inverse, b = (-a), and element c is called the multiplicative inverse, c = a

   (a≠0).

5. Commutative: For every a, b in F, a + b = b + a, a * b = b * a. (5)

6. Distributive: For every a, b, c in F, (a + b) * c = a * c + b * c. (6)

   The existence of a multiplicative inverse a⁻¹ enables the use of division. This is because for a, b, c ∈ F, c = b/a is defined as c = b * a⁻¹. Similarly the existence of an additive inverse (-a) enables the use of subtraction. In this case for a, b, c ∈ F, c = b - a is defined as c = b + (-a).

   It can be shown that the set of integers {0, 1, 2, ..., p-1} where p is a prime, together with modulo p addition and multiplication forms a field [8]. Such a field is called the finite field of order p, or GF(p), in honour of Evariste Galois [13].

C. Polynomial, Dual and Normal Basis Representations

Definition 1 [4] A set of m linearly independent elements β = {β₀, β₁, ..., βₘ₋₁} of GF(2ᵐ) is called a basis for GF(2ᵐ).

Definition 2 [4] Let p(x) be the defining irreducible polynomial for GF(2ᵐ). Take α as a root of p(x), then A = {1, α, α², ..., αᵐ⁻¹} is the polynomial basis for GF(2ᵐ).

Definition 3 [4] Let {λᵢ} and {μᵢ} be bases for GF(2ᵐ), let f be a linear function from GF(2ᵐ) → GF(2), and β ∈ GF(2ᵐ), β≠0. Then {λᵢ} and {μᵢ} are dual to each other with respect to f and β if.
\[ f(\beta \lambda, \mu_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j. \end{cases} \]

(7)

In this case, \( \{ \lambda_i \} \) is the standard basis and \( \{ \mu_i \} \) is the dual basis.

**Definition 4** A normal basis for GF(2\(^m\)) is a basis of the form \( B = \{ \beta, \beta^2, \ldots, \beta^{2^{m-1}} \} \) where \( \beta \in GF(2^m) \). For every finite field there always exists at least one normal basis [8].

**II. TYPES OF BIT-SERIAL MULTIPLIERS**

Generally bit-serial multipliers are classified as the following.

**A. Berlekamp Multiplier**

The Berlekamp multiplier [1] uses two basis representations, the polynomial basis for the multiplier and the dual basis for the multiplicand and the product. Because it is normal practice to input all data in the same basis, this means some basis transformation circuits will be required. Even including the extra hardware for basis conversions, the Berlekamp multiplier is known to have the lowest hardware requirements of all available bit-serial multipliers [5].

Now let \( a, b, c \in GF(2^m) \) such that \( c = a \cdot b \) and represent \( b \) over the polynomial basis as \( b = \sum_{k=0}^{m-1} b_k \cdot \alpha^k \), let \( \{ \mu_0, \mu_1, \ldots, \mu_{m-1} \} \) be the dual basis to the polynomial basis for some \( f \) and \( \beta \). Hence \( a = \sum_{i=0}^{m-1} a_i \cdot \mu_i \) and \( c = \sum_{i=0}^{m-1} c_i \cdot \mu_i \) where these values of \( a \) and \( c \) are given by the following.

**Lemma 1** [4] Let \( \{ \mu_0, \mu_1, \ldots, \mu_{m-1} \} \) be the dual basis to the polynomial basis for GF(2\(^m\)) for some \( f \) and \( \beta \) and let \( a = \sum_{i=0}^{m-1} a_i \cdot \mu_i \) be the dual basis representation of \( a \in GF(2^m) \). Then \( a_i = f(\alpha \mu^i \beta) \) for \( (i=0,1, \ldots, m-1) \).

The multiplication \( c = a \cdot b \) can therefore be represented in the matrix form [4]

\[
\begin{bmatrix}
    a_0 & a_1 & \ldots & a_{m-1} \\
    a_1 & a_2 & \ldots & a_m \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{m-1} & a_m & \ldots & a_{2m-2}
\end{bmatrix}
\begin{bmatrix}
    b_0 \\
    b_1 \\
    \vdots \\
    b_{m-1}
\end{bmatrix}
= \begin{bmatrix}
    c_0 \\
    c_1 \\
    \vdots \\
    c_{m-1}
\end{bmatrix}
\]

(8)

Where \( a_i = f(\alpha \mu^i \beta) \) and \( c_i = f(\beta \alpha \mu^i \beta) \) \( (i=0,1, \ldots, m-1) \) are the dual basis coefficients of \( a \) and \( c \) respectively and \( a_i = f(\alpha \beta \mu^i) \) \( (i=m, m+1, \ldots, 2m-2) \). It can be shown [15] that

\[ a_{m+k} = f(\alpha \beta \mu^{m+k}) = \sum_{j=0}^{m-1} p_j \cdot a_{j+k} \]

\[ (k=0,1, \ldots, m-1) \]

(9)

Where \( p_j \) are the coefficients of \( p(x) \). These values of \( a_{m+k} \) can therefore be obtained from an \( m \)-stage linear feedback shift register (LFSR) where the feedback terms correspond to the \( p_j \) coefficients and the LFSR is initialised with the dual basis coefficients of \( a \). On clocking the LFSR \( a_{m+k} \) is generated, then on the next clock cycle \( a_{m+k+1} \) is produced, and so on. The \( m \) vector multiplications listed in equ(8) are then carried out by a structure comprising \( m \)-input AND gates and \((m-1)\) 2-input XOR gates. As an example, a Berlekamp multiplier for GF(2\(^4\)) is shown in Fig. 1 where \( p(x) = x^4 + x + 1 \).

**Figure 1** Bit-serial Berlekamp multiplier for GF(2\(^4\)). [4]

The registers in Fig. 1 are initialised by \( a_i = a_i \) and \( b_i = b_i \) for \((i=0,1,2,3)\). At this point the first product bit \( c_0 \) is available on the output line. The remaining values of \( c_1, c_2 \) and \( c_3 \) are obtained by clocking the register a further three times.

With the above scheme at least one basis conversion is required if both inputs and the output are to be represented over the same basis. This basis transformation is a linear transformation of the basis coefficients and can be implemented within the multiplier structure itself. However with GF(2\(^4\)) the dual basis is a permutation of the polynomial basis coefficients and so this conversion can be implemented by a simple reordering of the inputs.

**B. Massey-Omura Multiplier**

The Massey-Omura multiplier [9,14] operates entirely over the normal basis and so no basis

**Figure 1** Bit-serial Berlekamp multiplier for GF(2\(^4\)). [4]
converters are required. The idea behind the Massey-Omura multiplier is that if the Boolean function generating the first product bit has the inputs cyclically shifted, then this same function will also generate the second product bit. Furthermore with each subsequent cyclic shift a further product bit is generated. Hence instead of $m$ Boolean functions, one Boolean function is required to generate all $m$ product bits but with the inputs to this function shifted each clock cycle.

As an example, consider a Massey-Omura bit-serial multiplier for $\text{GF}(2^4)$. A circuit diagram for this multiplier is given in Fig. 2. The registers in Fig. 2 are initialised by $A_i = a_i$ and $B_i = b_i$ for ($i=0,1,2,3$). At this point the first product bit $c_0$ will be available on the output line. The remaining values of $c_1$, $c_2$ and $c_3$ are obtained by cyclically shifting the registers a further three times.

In general there is a result that states the defining Massey-Omura function for a $\text{GF}(2^m)$ multiplier requires at least $(2^m-1)$ 2-input AND gates and at least $(2^m-2)$ 2-input XOR gates [11]. In the case of the above example, it can be seen that the $\text{GF}(2^4)$ Massey-Omura multiplier has achieved this lower bound.

C. Polynomial Basis Multiplier

Polynomial basis multipliers operate entirely over the polynomial basis and require no basis converters. These multipliers are easily implemented, reasonably hardware efficient and the time taken to produce the result is the same as for Berlekamp or Massey-Omura multipliers. In truth however bit-serial polynomial basis multipliers are serial-in parallel-out multipliers. In some applications this results in an additional register being required and adds an extra m clock cycles to the computation time. This is the main reason why polynomial basis multipliers are frequently overlooked for use in codec design.

There are two different methods of operation for polynomial basis multipliers, least significant bit (LSB) first or most significant bit (MSB) first. Either of these approaches may be chosen and both modes are described below.

Option L - LSB first

In this option the LSB appears first on the multiplier input. Therefore denote this multiplier a Bit-Serial Polynomial Basis Multiplier option L (SPBML). This multiplier is described in detail in the literature [2], ([7], pp.163 -164), ([8], pp. 90-91)

Option M - MSB first

In this option the MSB appears first on the multiplier input. The Bit-Serial Polynomial Basis Multiplier option M (SPBMM) has been known for many years [6]([7], p.163) and more recently modified by Scott et al [12].

III. Comparison of Multipliers

In comparing all four multipliers directly, it is noted that they each take m clock cycles to generate a solution. Similarly they each require 2m flip-flops. In order to compare the hardware requirements of these four multipliers some notation is introduced. Let $Na$ equal the number of 2-input AND gates required by a multiplier and let $Nx$ equal the number of 2-input XOR gates required by a multiplier. Furthermore, let $Da$ and $Dx$ be the delays through a 2-input AND gate and XOR gate respectively. Let $H(pp)$ be the Hamming weight of the primitive polynomial chosen.
for $\text{GF}(2^m)$. The hardware requirements and delays of these multipliers are given in below.

Berlekamp multiplier
\[ Na = m; \quad Nx = m + H(pp) - 3 \]
\[ \text{Delay} = Da + \lceil \log_2(m - 1) \rceil \ast Dx. \]  

(10)

Standard basis multiplier option L
\[ Na = m \]
\[ Nx = m + H(pp) - 2 \]
\[ \text{Delay} = Da + Dx. \]  

(11)

Standard basis multiplier option M
\[ Na = m \]
\[ Nx = m + H(pp) - 2 \]
\[ \text{Delay} = Da + 2Dx. \]  

(12)

For Massey-Omura multipliers the number of gates cannot be explicitly specified. As a comparison, values of $Na$ and $Nx$ for all three types of multiplier are given in Table

<table>
<thead>
<tr>
<th>$m$</th>
<th>Massey Omura [33]</th>
<th>Berlekamp</th>
<th>SPBML/SPBMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Na</td>
<td>Nx</td>
<td>Na</td>
<td>Nx</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>19</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table The usage of gates for bit-serial Massey Omura, Berlekamp and standard basis multipliers

IV. CONCLUSION

The Massey-Omura multiplier circuit is relatively hardware inefficient (compared to the Berlekamp multiplier for example, [5,10]) and cannot be hardwired to carry out reduced complexity constant multiplication. Furthermore, the Massey-Omura multiplier cannot be easily extended for different values of $m$ given a particular choice of $m$.

The Berlekamp multiplier is known to have very low hardware requirements [5]. The Berlekamp multiplier can also be hardwired to allow for particularly efficient constant multiplication [1]. The disadvantage of this multiplier is that it operates over both the dual and the polynomial basis, and so basis converters may be required.

The bit-serial polynomial basis multipliers do not require basis converters, and are almost as hardware efficient as the Berlekamp multiplier. They do however have a different interface to the Berlekamp multiplier being serial-in-parallel-out. Hence the choice between a Berlekamp and a polynomial basis multiplier often depends on the circuit in which the multiplier is to be implemented.

In conclusion polynomial basis multipliers can be only serial-in-parallel-out, whereas dual and normal basis multipliers can be either parallel-in serial-out or serial-in-parallel-out.

REFERENCES