Double-Basis Multiplicative Inversion over GF(2^m) †

M. A. Hasan

Abstract

Inversion over Galois fields is much more difficult than the corresponding multiplication. In this article, efficient computation of inverses in GF(2^m) is considered by solving a set of linear equations over the ground field GF(2). The proposed algorithm uses two separate bases for the representation of its input and output elements, and has low computational complexity. The algorithm is also suitable for hardware implementation using VLSI technologies.

Index Terms: Computer arithmetic, Euclid's algorithm, Galois (or finite) fields, multiplicative inverses, canonical (or polynomial) basis and triangular basis.

I. INTRODUCTION

In order to provide digital signatures or message authentication, many digital communications systems are becoming increasingly equipped with some form of cryptosystems. Many of these cryptosystems require computation in Galois (finite) fields. In a public-key cryptosystem, two parties wishing to communicate using a non-secure channel do not have to meet– they do not even have to have established any kind of previous communication. Using the Diffie-Hellman key exchange protocol, the two parties can set a common key [2]. The implementation of this key exchange protocol using elliptic curve cryptosystems requires inversion in Galois fields [3]. Galois field inversion is also needed in many error-control coding schemes used for reliable data transmission and storage systems. For example, certain encoding and decoding schemes of Reed-Solomon codes require Galois field inversions [4].

In the recent past, considerable efforts have been made to develop efficient schemes for the inversion in Galois fields, *e.g.*, [5, 6, 7]. These schemes are mainly based on either Fermat's theorem or Euclid's algorithm or the solution of a set of equations. The Fermat theorem based scheme can be used for any *basis* representation of the field elements [5]. However, a *normal* basis representation, where squaring is very simple, seems to be the most suitable one since the underlying

†Appeared in *IEEE Trans. Computers,* pp. 960-970, September 1998. A preliminary version of this paper was presented at the 8th Canadian Conference on Electrical and Computer Engineering, Montreal, PQ., Canada [1] The author is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. His email address is ahasan@ece.uwaterloo.ca.

scheme requires repeated squaring operations. The Euclid algorithm based scheme has a low space complexity. So far, however, it has been used only for a *canonical* basis representation [6]. The concept of computing inversion in an extended field by solving a set of equations in the corresponding ground field can be applied to any basis representation. Unfortunately, its hardware implementation requires too much space [7].

In this article, the inversion over the field GF(2^m) is considered. An efficient inversion algorithm, which relies on the solution of a set of linear equations over the ground field GF(2), is presented. The proposed algorithm uses two separate bases for the representation of the input and output, but has low computational complexity. It is also suitable for hardware implementation.

The remainder of the article is organized as follows. In Section II, the relationship between inversion in GF(2^m) and linear equations over GF(2) is briefly reviewed, and the computational complexity for the formation of the equations for the double-basis inversion is formulated. In Section III, an efficient double-basis inversion algorithm is developed. Section IV presents an inverter structure based on a centralized control. Its operation, control and comparison to other circuits are given in Section V. Finally, some concluding remarks are made at end of the article.

II. INVERSION AND LINEAR EQUATIONS

This section discusses an approach to compute inverses using a system of linear equations [8] and lays the foundation of obtaining the main results of this article. The discussion is rather general and does not consider any special cases for which efficient algorithms can be used (see, for example, [9] and the references therein).

A. Single-Basis Inversion

The finite field GF(2^m) is an extension field of GF(2) and has 2^m elements. Let

$$F(z) = \sum_{i=0}^{m} f_i z^i, \quad f_i \in \{0, 1\}$$

be an irreducible polynomial of degree m over GF(2). Let $F(z)$ has a root ω in GF(2^m). Then $\Omega = \{1,\ \omega,\ \ldots,\ \omega^{m-1}\}$ is a *canonical* basis and all the elements of GF(2^m) can be represented with respect to this basis, *i.e.*, for $\alpha \in$GF(2^m) one can write

$$\alpha = \sum_{i=0}^{m-1} a_{\Omega i}\omega^i = \begin{bmatrix} 1, & \omega, & \ldots, & \omega^{m-1} \end{bmatrix} \begin{bmatrix} a_{\Omega 0} \\ a_{\Omega 1} \\ \vdots \\ a_{\Omega(m-1)} \end{bmatrix},$$

where $a_{\Omega i}$'s are the coordinates of α with respect to Ω. Similarly, for $0 \leq i \leq m-1$, $b_{\Omega i}$'s denote the coordinates of β.

Let $\alpha \neq 0$ and $1/\alpha = \beta$. Then

$$\begin{aligned} 1 &= \left(\sum_{j=0}^{m-1} b_{\Omega j}\omega^j\right)\left(\sum_{l=0}^{m-1} a_{\Omega l}\omega^l\right) \mod F(\omega) \\ &= \sum_{j=0}^{m-1} b_{\Omega j} \sum_{l=0}^{m-1} a_{\Omega l}\omega^{j+l} \mod F(\omega) \\ &= \sum_{j=0}^{m-1} b_{\Omega j} \sum_{l=0}^{m-1} a_{\Omega l} \sum_{i=0}^{m-1} p_{\Omega i}^{[j+l]}\omega^i, \end{aligned} \tag{1}$$

where $p_{\Omega i}^{[j]}$ is the ith coordinate of ω^j. Equating the coefficients of w^i (for $i = 0,\ 1,\ \ldots,\ m-1$) on both sides of (1), one obtains

$$\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,m-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m-1,0} & A_{m-1,1} & \cdots & A_{m-1,m-1} \end{bmatrix} \begin{bmatrix} b_{\Omega 0} \\ b_{\Omega 1} \\ \vdots \\ b_{\Omega(m-1)} \end{bmatrix}, \tag{2}$$

where $A_{i,j} = \sum_{l=0}^{m-1} a_{\Omega l} p_{\Omega i}^{[l+j]}$. Following [8], (2) can be written as

$$\underline{1}_\Omega = \mathbf{A}(\underline{\alpha}_\Omega)\underline{\beta}_\Omega, \tag{3}$$

where $\underline{1}_\Omega$, $\underline{\alpha}_\Omega$ and $\underline{\beta}_\Omega$ are the column vectors corresponding to the Ω basis representations of 1, α and β, respectively, and $\mathbf{A}(\underline{\alpha}_\Omega) = \left[\underline{\alpha}_\Omega,\ \underline{\alpha\omega}_\Omega,\ \ldots,\ \underline{\alpha\omega}_\Omega^{m-1}\right]$ is an $m \times m$ matrix over GF(2). Since $A_{i,j} = \sum_{l=0}^{m-1} a_{\Omega l} p_{\Omega i}^{[l+j]}$, the matrix in (3) can be uniquely defined by the coordinates of α and the coefficients of $F(z)$, and it has been shown in [7] that with m bits of memory the formation of the matrix requires $O(m^2)$ arithmetic operations over GF(2). Then the inverse β is obtained by solving the system of linear equations (2), which requires $O(m^3)$ arithmetic operations over GF(2).

B. Double-Basis Inversion

To avoid any basis change, the elements of GF(2^m) are conventionally represented with respect to a single basis, such as the canonical basis discussed above. However, it has been shown that the use of two suitable bases in one single device/system can lead to efficient realization of the multiplication operation [10, 11, 12]. In the following discussion, both the canonical basis and its *triangular* basis will be simultaneously used in the inversion operation.

The set $\Lambda = \{\lambda_0,\ \lambda_1,\ \cdots,\ \lambda_{m-1}\}$ of m elements is called the triangular basis of Ω if

$$\lambda_i = \sum_{j=0}^{m-1-i} f_{i+j+1}\omega^j, \quad 0 \le i \le m-1, \tag{4}$$

where f_i's are the coefficients of the irreducible polynomial $F(z)$ [13]. For any irreducible $F(z)$ of degree m over GF(2)

$$\lambda_0 = \omega^{-1}, \tag{5}$$
$$\lambda_{m-1} = 1. \tag{6}$$

Let $a_{\Lambda i}$ denote the ith coordinate of α with respect to Λ, and $\underline{\alpha}_\Lambda$ denote the vector of the coordinates of α represented with respect to Λ, i.e.,

$$\underline{\alpha}_\Lambda = \left[a_{\Lambda 0},\ a_{\Lambda 1},\ \cdots,\ a_{\Lambda(m-1)}\right]^T.$$

Since $\alpha = \sum_{i=0}^{m-1} a_{\Lambda i}\lambda_i = \sum_{j=0}^{m-1} a_{\Omega j}\omega_j$, one can change the Ω basis coordinates to the corresponding Λ basis coordinates using the following relationship [13]:

$$a_{\Lambda j} = \begin{cases} a_{\Omega(m-1-j)}, & j = 0, \\ a_{\Omega(m-1-j)} + \sum_{i=0}^{j-1} f_{m-j+i}a_{\Lambda i}, & 1 \le j \le m-1, \end{cases} \tag{7}$$

which can be expressed as

$$\underline{\alpha}_\Lambda = \mathbf{T}\underline{\alpha}_\Omega, \tag{8}$$

where

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & t_1 \\ 0 & 0 & 0 & \cdots & 1 & t_1 & t_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 1 & t_1 & \cdots & t_{m-4} & t_{m-3} & t_{m-2} \\ 1 & t_1 & t_2 & \cdots & t_{m-3} & t_{m-2} & t_{m-1} \end{bmatrix}, \tag{9}$$

and $t_j = \sum_{i=0}^{j-1} f_{m-j+i} t_i$, for $0 \leq i \leq m-1$, and $t_0 = 1$. Pre-multiplying both sides of (3) with $\mathbf{T}$, we have

$$\underline{1}_\Lambda \;=\; \mathbf{H}(\underline{\alpha}_\Lambda)\underline{\beta}_\Omega \tag{10}$$

where $\underline{1}_\Lambda$ is the Λ basis representation of the multiplicative unity of GF(2^m) and

$$\mathbf{H}(\underline{\alpha}_\Lambda) = \left[\underline{\alpha}_\Lambda \;\; \underline{\alpha\omega}_\Lambda \;\; \cdots \;\; \underline{\alpha\omega}_\Lambda^{m-1}\right]. \tag{11}$$

$\mathbf{H}(\underline{\alpha}_\Lambda)$ is a Hankel matrix, and the computational complexity for its formation, as stated in the following lemma, is equal to that of $\mathbf{A}(\underline{\alpha}_\Omega)$ in (3). However, the main advantage of using (10) to compute inverses is that (10) can be solved for $b_{\Omega i}$'s with $O(m^2)$ GF(2) arithmetic operations [14], as opposed to $O(m^3)$ required by (3). Efficient algorithms for solving (10) are discussed in the next section.

Lemma 1 The formation of $\mathbf{H}$ requires $O(m^2)$ arithmetic operations over GF(2).

Proof:

$$\alpha \;=\; \sum_{i=0}^{m-1} a_{\Lambda i}\lambda_i = \sum_{i=0}^{m-1} a_{\Lambda i} \sum_{j=0}^{m-1-i} f_{i+j+1}\omega^j. \tag{12}$$

$$\begin{aligned}
\alpha\omega \;&=\; \sum_{i=0}^{m-1} a_{\Lambda i} \sum_{j=0}^{m-1-i} f_{i+j+1}\omega^{j+1} \\
&=\; \sum_{i=0}^{m-1} a_{\Lambda i} \sum_{j=1}^{m-i} f_{i+j}\omega^{j} \\
&=\; \sum_{i=0}^{m-1} a_{\Lambda i} \left(\sum_{j=0}^{m-i} f_{(i-1)+j+1}\omega^j + f_i \right).
\end{aligned} \tag{13}$$

Using $F(\omega) = 0$, (4) and (6), one can write (13) as follows:

$$\alpha\omega \;=\; \sum_{i=1}^{m-1} a_{\Lambda i}\lambda_{i-1} + \sum_{i=0}^{m-1} a_{\Lambda i} f_i \lambda_{m-1}. \tag{14}$$

Equations (12) and (14) relate the coordinates of α and $\alpha\omega$. The result can be generalized to obtain

$$\underline{\alpha\omega}_{\Lambda j}^{i+1} = \begin{cases} \underline{\alpha\omega}_{\Lambda(j+1)}^{i} & 0 \leq j \leq m-2, \\ \sum_{l=0}^{m-1} \underline{\alpha\omega}_{\Lambda l}^{i} f_l & j = m-1. \end{cases} \tag{15}$$

Thus, $\underline{\alpha\omega}_{\Lambda}^{i+1}$ can be obtained from $\underline{\alpha\omega}_{\Lambda}^{i}$ with only m multiplications and $m-1$ additions over GF(2) resulting in only $O(m^2)$ operations for the formation of $\mathbf{H}$. Q.E.D.

III. EFFICIENT DOUBLE-BASIS INVERSION

In this section, an area efficient algorithm suitable for hardware implementation to solve (10) for $b_{\Omega i}$'s is presented. As an intermediate result, an inversion algorithm (referred to as Algorithm 1) is derived first from Sugiyama's work [14] which gives us a solution, if it exists, even when a principal sub-matrix of the matrix in (10) is singular. We then present the area efficient inversion algorithm (referred to as Algorithm 2) which is derived from Algorithm 1 by applying *GCD-preserving transformations*. The latter transforms a pair of polynomials into another pair with the property that a GCD (greatest common divisor) of the first pair is also a GCD of the transformed pair [15].

A. Less Complex Inversion Algorithm

The Hankel matrix in (10) has $2m-1$ constant coefficients. Let these coefficients be h_i, $0 \le i \le 2m-2$. Then the matrix entry at (i,j) is h_{i+j}, and equation (10) becomes

$$\begin{bmatrix} h_0 & h_1 & \cdots & h_{m-2} & h_{m-1} \\ h_1 & h_2 & \cdots & h_{m-1} & h_m \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ h_{m-2} & h_{m-1} & \cdots & h_{2m-4} & h_{2m-3} \\ h_{m-1} & h_m & \cdots & h_{2m-3} & h_{2m-2} \end{bmatrix} \begin{bmatrix} b_{\Omega 0} \\ b_{\Omega 1} \\ \vdots \\ b_{\Omega(m-2)} \\ b_{\Omega(m-1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{16}$$

Let $H(z)$ and $B_{\Omega}(z)$ denote the two polynomials whose coefficients are the entries of the corresponding matrix and vectors in (16). Then

$$\begin{aligned} 0 \le &\quad \deg H(z) &&\le 2m-2, \\ 0 \le &\quad \deg B_{\Omega}(z) &&\le m-1. \end{aligned}$$

Also, let $\lfloor U(z)/V(z) \rfloor$ denote the quotient polynomial when $U(z)$ is divided by $V(z)$. Then we can state the following algorithm which requires $O(m^2)$ arithmetic operations over GF(2):

Algorithm 1 (Less Complex Double-Basis Inversion Algorithm)

Step 1

$$\begin{aligned} &R^{(-1)}(z) = z^{2m-1}, && R^{(0)}(z) = z^{2m-2}H(z^{-1}), \\ &U^{(-1)}(z) = 0, && U^{(0)}(z) = 1, \\ &S^{(0)}(z) = z^{m-1} && L^{(0)}(z) = 0, \\ &i = 0. \end{aligned}$$

Step 2 While deg $R^{(i)}(z) \geq m-1$, do

{

$$
\begin{aligned}
i &= i+1 \\
Q^{(i)}(z) &= \left\lfloor R^{(i-2)}(z)/R^{(i-1)}(z) \right\rfloor \\
R^{(i)}(z) &= R^{(i-2)}(z) - Q^{(i)}(z)R^{(i-1)}(z) \\
U^{(i)}(z) &= U^{(i-2)}(z) - Q^{(i)}(z)U^{(i-1)}(z) \\
D^{(i)}(z) &= \left\lfloor S^{(i-1)}(z)/R^{(i-1)}(z) \right\rfloor \\
S^{(i)}(z) &= S^{(i-1)}(z) - D^{(i)}(z)R^{(i-1)}(z) \\
L^{(i)}(z) &= L^{(i-1)}(z) - D^{(i)}(z)U^{(i-1)}(z).
\end{aligned}
$$

}

Step 3 $B_\Omega(z) = L^{(i)}(z)$. □

Sketch of proof: Since every nonzero element in GF(2^m) has an inverse and the matrix in (16) is non-singular, a unique solution for the $b_{\Omega i}$'s exists. Noting that $R^{(0)}(z)$ is the *reciprocal* polynomial of $H(z)$, the remainder of the proof follows from [14]. □

Instead of Algorithm 1, the Sugiyama algorithm [14, p. 400] could be used to compute inverses in GF(2^m). While the Sugiyama algorithm checks the degrees of four polynomials to stop the iterations, Algorithm 1 checks the degree of only one polynomial. This simplification is advantageous especially for a hardware implementation.

B. Area Efficient Inversion Algorithm

Algorithm 1 keeps track of eight polynomials. Furthermore, its iteration loop requires polynomial divisions. For a polynomial division, one needs to locate the leading nonzero term of each *partial remainder* polynomial, and its automation in hardware is not simple. Below we present an area efficient algorithm which overcomes these problems.

Note that like the computation of the GCD using Euclid's algorithm, Algorithm 1 proceeds the same way to reduce the degree of $R^{(i-2)}$ to yield $R^{(i)}$ in the ith iteration. Thus the GCD-preserving transformation can be applied to obtain $R^{(i)}$ in several sub-iterations. For this purpose, instead of the full quotient polynomial $Q^{(i)}(z)$, only one nonzero term of it, starting from the highest degree, can be used. Then we have the following algorithm for computing inversion over GF(2^m):

Algorithm 2 (Area Efficient Double-Basis Inversion Algorithm)

Step 1

$$R^{(-1)}(z) = z^{2m-1}, \quad R^{(0)}(z) = z^{2m-2}H(z^{-1}),$$
$$U^{(-1)}(z) = 0, \qquad U^{(0)}(z) = 1,$$
$$i = 0.$$

Step 2 While $\deg R^{(i)}(z) > m-1$, do
{

$$\begin{aligned} i &= i+1 \\ d &= \deg R^{(i-2)}(z) - \deg R^{(i-1)}(z) \\ R^{(i)}(z) &= R^{(i-2)}(z) - z^d R^{(i-1)}(z) \\ U^{(i)}(z) &= U^{(i-2)}(z) - z^d U^{(i-1)}(z) \end{aligned}$$

If $\deg R^{(i)}(z) \geq \deg R^{(i-1)}(z)$ then swap $R^{(i)}(z)$ and $R^{(i-1)}(z)$, and $U^{(i)}(z)$ and $U^{(i-1)}(z)$.
}

Step 3 $B_\Omega(z) = U^{(i)}(z)$. □

Proof: Let Algorithm 1 terminate with $i = e$. Thus,

$$\begin{aligned} \deg R^{(e-1)}(z) &\geq m-1, \\ \deg R^{(e)}(z) &< m-1. \end{aligned}$$

The value of e is unique since $\deg R^{(i)}(z)$ is monotonously decreasing as i is increasing.

For $1 \leq i \leq e-1$, we have $D^{(i)}(z) = 0$, which, in turn, means $S^{(i)}(z) = z^{m-1}$ and $L^{(i)}(z) = 0$. Let us assume that $\deg R^{(e-1)}(z) > m-1$, then $B_\Omega(z) = L^{(e)} = 0$ which, however, cannot be true since $B_\Omega(z)$ is the polynomial representation of the inverse of α, and every nonzero element has a unique inverse which is also nonzero. Thus $\deg R^{(e-1)}(z) = m-1$ which means $D^{(e)}(z) = 1$ and $B_\Omega(z) = L^{(e)} = U^{(e)}$. Q.E.D.

Example 1 Consider the field GF(2^4) generated by the irreducible polynomial $z^4 + z + 1$. Let $\alpha = \omega^4$ whose 4-tuple representation w.r.t. the triangular basis $\{\omega^{14}, \omega^2, \omega, 1\}$ is (0, 0, 1, 1). Using the definition of the Hankel matrix of (11), we have

$$\mathbf{H}(\underline{\alpha}_\Lambda) = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

which yields $H(z) = z^5 + z^3 + z^2$. The step-by-step operations of Algorithm 2 applied to the calculation of the inverse of α are given below:

Step 1

$$R^{(-1)}(z) = z^7, \quad R^{(0)}(z) = z^4 + z^3 + z,$$
$$U^{(-1)}(z) = 0, \quad U^{(0)}(z) = 1,$$
$$i = 0.$$

Step 2 Computations of this step are performed by iterations. Before each iteration, the degree of $R^{(i)}(z)$ is checked. As long as deg $R^{(i)}(z) > m - 1$, (*i.e.*, 3) the iteration continues. The updated polynomials in each iteration are given in Table 1.

i	Updated Polynomials	
1	$d = 3$	$R^{(0)}(z) = z^6 + z^4$
	$R^{(1)}(z) = z^6 + z^4$	$R^{(1)}(z) = z^4 + z^3 + z$
	$U^{(1)}(z) = z^3$	$U^{(0)}(z) = z^3$
	Since deg $R^{(1)}(z) >$ deg $R^{(0)}(z)$, swap.	$U^{(1)}(z) = 1$
2	$d = 2$	$R^{(1)}(z) = z^5 + z^4 + z^3$
	$R^{(2)}(z) = z^5 + z^4 + z^3$	$R^{(2)}(z) = z^4 + z^3 + z$
	$U^{(2)}(z) = z^3 + z^2$	$U^{(1)}(z) = z^3 + z^2$
	Since deg $R^{(2)}(z) >$ deg $R^{(1)}(z)$, swap.	$U^{(2)}(z) = 1$
3	$d = 1$	
	$R^{(3)}(z) = z^3 + z^2$	
	$U^{(3)}(z) = z^3 + z^2 + z$	
	Since deg $R^{(3)}(z) <$ deg $R^{(2)}(z)$, do not swap.	

Table 1: Computations of Step 2 in Example 1.

Step 3 $B_\Omega(z) = U^{(3)}(z) = z^3 + z^2 + z$, which is the polynomial representation of the Ω basis coordinates of the inverse of α. □

The main advantage of Algorithm 2 is that its iteration process in Step 2 requires only four polynomials which is half of that of Algorithm 1. As a result, its implementation would require less

storage space. Additionally, there are no polynomial divisions in the algorithm; there are even no multiplications, since $z^d R^{(i-1)}(z)$ corresponds to a d-times shifting of the coefficients of $R^{(i-1)}(z)$.

IV. INVERTER STRUCTURE

Because of the simple shift and add operations, Algorithm 2 appears to be suitable for hardware implementation. In Step 1 of the algorithm, the only part that requires computation is $R^{(0)}(z)$. In Step 2, a number of polynomials are updated in each iteration by shift and add operations. Possible hardware structures of these two steps are presented below.

A. Generation of $R^{(0)}(z)$

Since $R^{(0)}(z) = z^{2m-2}H(z^{-1})$, using (11), (15) and (16) one can write

$$h_i = \begin{cases} a_{\Lambda i} & 0 \le i \le m-1, \\ \sum_{j=0}^{m-1} h_{i-m+j} f_j & m \le i \le 2m-2. \end{cases} \tag{17}$$

Thus, $R^{(0)}(z)$ can be realized using a Galois type linear feedback shift register (LFSR) with $F(z)$ as its *feedback* polynomial and $S(z)$ as the *seed* polynomial, which is given by

$$\begin{aligned} S(z) &= \sum_{j=0}^{m-1} a_{\Lambda j} z^j = \sum_{j=0}^{m-1} h_j z^j \\ &= H(z) \bmod z^m. \end{aligned}$$

Given the triangular basis coordinates $a_{\Lambda j}$, $j = 0, 1 \cdots, m-1$, the above LFSR based approach requires $m-1$ clock cycles to compute $R^{(0)}(z)$. In order to reduce the computation time, one may attempt to compute $R^{(0)}(z)$ using combinational logic circuits. Such circuits will require $(m-1)(W_F - 2)$ two-input XOR gates where W_F is the Hamming weight of the field defining polynomial $F(z)$. Thus, it is advantageous to use a low weight $F(z)$, such as, a trinomial, if it exists. A trinomial of the form $z^m + z + 1$ yields

$$h_i = a_{\Lambda,i-m} + a_{\Lambda,i-m+1} \qquad m \le i \le 2m-2,$$

which, unlike (17), does not have any recursion, and gives the shortest critical logic path of only one XOR gate. This is illustrated in Fig. 1 using $F(z)$ of Example 1.

B. Polynomial Updating

Hardware realization of the polynomial updating which occurs in Step 2 of Algorithm 2 can be achieved by a number of ways. The overall inverter structure using a centralized control unit is shown in Fig. 2. The datapath portion of the structure consists of a number of components whose functions and operations are given below. The control unit of the structure depends on the overall inversion operation and is given in the next section.

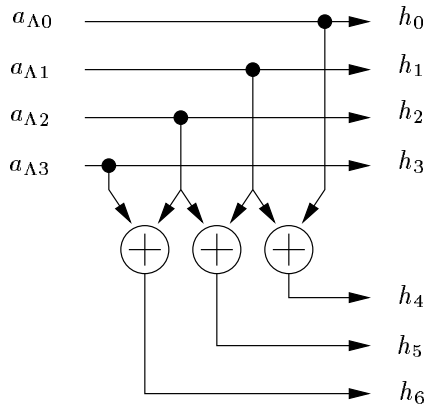

Figure 1: Generation of coefficients $h_0,\ h_1,\ \cdots,\ h_6$ when $F(z) = z^4 + z + 1$.

- Registers Φ and Ψ: The purpose of these two registers is to hold the coefficients of the polynomials of Step 2 of Algorithm 2. Each register consists of $3m$ storage cells labeled as ϕ_i and ψ_i, $0 \leq i \leq 3m-1$. These cells can be considered to be simple D flip-flops clocked by Ck_Φ and Ck_Ψ, respectively.

  Initially, the upper and lower portions of register Ψ are loaded with the coefficients of $R^{(0)}(z)$ and $U^{(0)}(z)$, respectively. Since $R^{(0)}(z) = z^{2m-2}H(z)$ and $U^{(0)}(z) = 1$, this loading operation results in

$$\psi_i = \begin{cases} 0 & i = 3m-1, \\ h_{3m-2-i} & m \leq i \leq 3m-2, \\ 0 & 1 \leq i \leq m-1, \\ 1 & i = 0, \end{cases} \tag{18}$$

  which implies that a suitable mechanism, such as multiplexing, is needed to initialize ψ_i with h_{3m-2-i}, for $m \leq i \leq 3m-2$.

  On the other hand, register Φ is initialized with the coefficients of $R^{(-1)}(z) = z^{2m-1}$ and $U^{(-1)}(z) = 0$, respectively, *i.e.*,

$$\phi_i = \begin{cases} 1 & i = 3m-1, \\ 0 & 0 \leq i \leq 3m-2. \end{cases} \tag{19}$$

  which can be accomplished simply by clearing all but the top cell which is set to 1.

  When a clock pulse is applied through Ck_Φ (resp. Ck_Ψ), contents of Φ (resp. Ψ) are shifted up by one cell. A sequence of d such consecutive shifts to one of these registers is equivalent to multiplying the corresponding register polynomials with z^d (as required in Step 2). Also, note that each shift operation pushes out the highest order coefficient of the polynomial stored

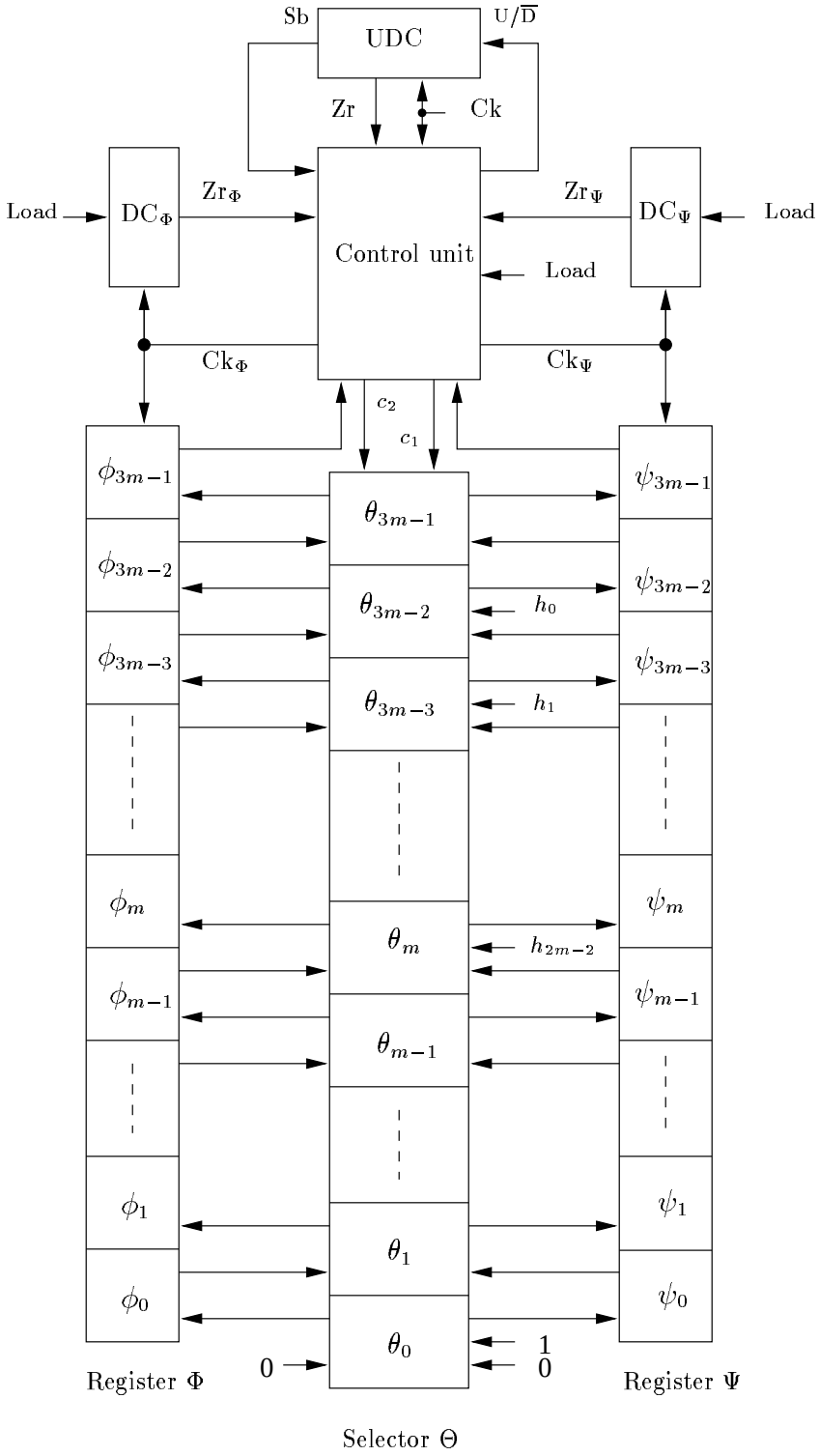

Figure 2: Structure for polynomial updating.

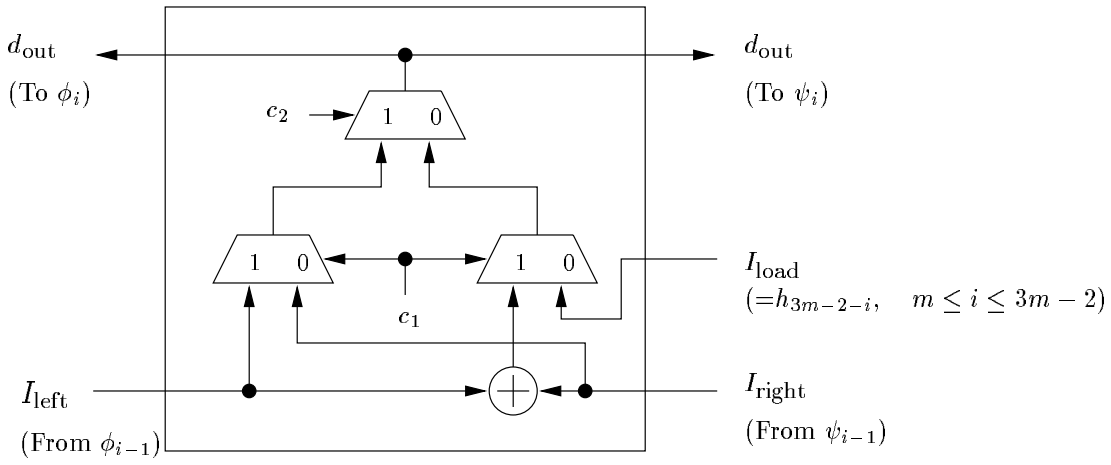

Figure 3: Details of cell θ_i.

in the upper portion of the register resulting in a reduction of the degree of the polynomial by one. Since the iterations of Step 2 do stop when deg $R^{(i)}(z)$ is no longer greater than $m-1$, the degrees of the polynomials stored in the upper portion of Φ and Ψ can go from $2m-1$ down to $m-1$, which implies that the iterations will stop when a maximum of $(2m-1)-(m-1)=m$ clock pulses are applied to one of these registers.

- Selector Θ: It provides appropriate inputs to registers Φ and Ψ, and is divided into $3m$ cells labeled as θ_i, $0 \le i \le 3m-1$. Each cell has three data inputs (I_{load}, I_{left} and I_{right}) and one data output (d_{out}). Each cell also has two control inputs c_1 and c_2. The output d_{out} is related to the inputs as follows:

$$d_{\text{out}} = \begin{cases} I_{\text{load}} & \text{if } (c_2, c_1) = (0,0), \\ I_{\text{left}} + I_{\text{right}} & \text{if } (c_2, c_1) = (0,1), \\ I_{\text{right}} & \text{if } (c_2, c_1) = (1,0), \\ I_{\text{left}} & \text{if } (c_2, c_1) = (1,1), \end{cases} \tag{20}$$

implying that a 4:1 multiplexor and an XOR gate are needed for each cell. If the 4:1 multiplexor is substituted with three 2:1 multiplexors, as shown in Fig. 3, then cell θ_i, for $i = 1, 2, \cdots, m-1$, and $3m-1$, can be constructed with only two 2:1 multiplexors, since in this type of cells, I_{load} is fixed and can be incorporated simply by clearing/pre-setting the corresponding flip-flop.

- Down counters: They are labeled as DC_Φ and DC_Ψ. The purpose of DC_Φ (resp. DC_Ψ) is to indicate the number of clock pulses applied to Φ (resp. Ψ). Since a maximum of m clock pulses are applied to Φ and Ψ (and hence to these two counters), the latter can be initialized with m and the iteration process of Step 2 will stop when either DC_Φ or DC_Ψ reaches zero. This would require both down counters to be $\lceil \log_2(m+1) \rceil$ bit long with zero detection signals (Zr_Φ and Zr_Ψ) as shown in Fig. 2.

- Up-down counters: The $\lceil \log_2 m \rceil + 1$ bit up-down counter, labeled as UDC, is to indicate the differences between the number of clock pulses applied to DC_Φ and DC_Ψ. When DC_Φ (resp. DC_Ψ) gets a pulse, the value of UDC is increased (resp. decreased) by 1. The increment/decrement is determined by the control signal U/$\overline{\text{D}}$ generated by the control unit. The UDC is clocked by the system clock denoted as Ck in Fig. 2. The Sb signal is the most significant bit of the UDC and is used to indicate whether the contents of the counter is positive or negative. Signal Zr is equal to 1 if UDC=0, and 0 otherwise. The UDC is initialized by clearing its flip-flops indicating UDC=$DC_\Phi - DC_\Psi = 0$ at the beginning.

V. OVERALL OPERATION AND RELATED ISSUES

A. Overall Inversion Operation

An informal description of the computation of an inverse using the structure of Fig. 2 is now given. The coefficients of $R^{(0)}(z)$ (which, in turn, are equal to h_i's, $0 \leq i \leq 2m-2$) are assumed to be generated using combinational logic circuits as explained in the previous section. When Load=1, counters DC_Φ and DC_Ψ are loaded with m. At the same time, register Ψ is loaded with $R^{(0)}(z)$ and $U^{(0)}(z)$, and register Φ is loaded with $R^{(-1)}(z)$ and $U^{(-1)}(z)$ (refer to (18) and (19)). Then, in each clock cycle, if ψ_{3m-1} (resp. ϕ_{3m-1}) is zero[1], the contents of register Ψ (resp. Φ) are shifted one position up by applying a clock pulse through Ck_Ψ (resp. Ck_Φ). The latter also reduces the value of DC_Ψ (resp. DC_Φ) by 1. The change in DC_Ψ (resp. DC_Φ) is reflected on the UDC by decreasing (resp. increasing) its value by 1.

If in a particular clock cycle, both ϕ_{3m-1} and ψ_{3m-1} are equal to 1, the contents of Φ and Ψ are added (say, $\Gamma = \Phi + \Psi$) using the XORs in the cells of Θ. Whether Γ will be written into Φ or Ψ depends on the values of DC_Φ and DC_Ψ. If $DC_\Phi > DC_\Psi$ (resp. $DC_\Psi > DC_\Phi$), Γ is written to Φ (resp. Ψ). When $DC_\Phi = DC_\Psi$, in which case UDC=0 and hence Zr=1, the contents of the two down counters in the previous clock cycle are compared, which could be accomplished by looking at one more signal, namely Sb$'$ which is the most significant bit of UDC in the previous cycle. Also, since $\phi_{3m-1} + \psi_{3m-1} = 0$, Γ is written (to Φ or Ψ) one cell up resulting in a saving one clock cycle.

When one of the down counters reaches zero, *i.e.*, either Zr_Φ or Zr_Ψ becomes 1, the iterations of Step 2 stop, and additional clock cycles do not affect the registers and the counters. Zr_Φ=1 (resp. Zr_Ψ=1) implies that the contents of Φ (resp. Ψ) have been shifted m times, and the required $U^{(i)}(z)$ polynomial coefficients can be found in the register cells numbered from m to $2m-1$. These coefficients which represent the inverse of the input element α, are also available at the d_{out} lines of θ_i, $i = m+1, m+2, \cdots, 2m$.

The maximum number of clock cycles needed for the iterations of Step 2 is $2m-1$. Since deg $R^{(0)}(z) \leq 2m-2$, the coefficients of $R^{(0)}(z)$ can be loaded into Ψ one cell up. Then the resulting maximum number of clock cycles would be $2m-2$. (This requires DC_Ψ to be initialized with $m-1$ rather than m.) Assuming that it takes two clock cycles for clearing and initializing all flip-flops of the structure, a throughput of $2m$ clock cycles per inversion can be obtained. The clock cycle period is independent of the circuits to generate $R^{(0)}(z)$ and, hence, the value of m.

[1] It can be shown that both ϕ_{3m-1} and ψ_{3m-1} do not become 0 in the same clock cycle.

B. Control Unit

The control unit which is a part of the structure presented in Fig. 2 and generates the signals required to coordinate the above inversion operation, is now given. The inputs to the control unit are signals Load, Ck, Sb, Zr, Zr_Φ and Zr_Ψ. The output signals are c_1, c_2, Ck_Φ and Ck_Ψ. While signals c_1 and c_2 are the select inputs to the multiplexors of Θ, signals Ck_Φ and Ck_Ψ are simply the clock inputs to the flip-flops of Φ and Ψ, respectively. Internally, the control unit generates another signal denoted as Sb$'$ which is simply the value of Sb in the previous clock cycle.

Signals c_1 and c_2 determine the data to be written into the selected register (Φ or Ψ). Based on (20), the generation of these two signals can be formulated as follows:

$$c_1 = \begin{cases} 0 & \text{if Load=1 or } \psi_{3m-1} = 0, \\ 1 & \text{otherwise.} \end{cases}$$

$$c_2 = \begin{cases} 0 & \text{if Load=1 or } (\phi_{3m-1}, \psi_{3m-1}) = (1,1), \\ 1 & \text{otherwise.} \end{cases}$$

Register Φ (resp. Ψ) is updated with the data available at its input by applying the system clock (*i.e.*, Ck) through Ck_Φ (resp. Ck_Ψ). The system clock is applied to Φ and Ψ as long as neither DC_Φ nor DC_Ψ reach zero. Based on the inversion operation given above, one can see that Ψ is clocked for the following three operations— (i) to load Ψ with h_i's when Load=1, (ii) to shift contents of Ψ one cell up when ψ_{3m-1}=0, and (iii) to add-and-shift (*i.e.*, $\psi_{i+1} \leftarrow \psi_i + \phi_i$) when both ψ_{3m-1} and ϕ_{3m-1} are equal to 1, and $\mathrm{DC}_\Psi > \mathrm{DC}_\Phi$ in the current or previous clock cycle. Except for the loading of h_i's, the updating conditions for Φ are similar to those of Ψ. Logic equations for the gated clock signals CK_Ψ and CK_Φ to update Ψ and Φ are given below.

$$\mathrm{Ck}_\Psi = \begin{cases} \mathrm{Ck} & \text{if } \overline{\mathrm{Zr}}_\Phi \overline{\mathrm{Zr}}_\Psi (\mathrm{Load} \vee c_2\overline{c}_1 \vee (\overline{c}_2 c_1 (\overline{\mathrm{Sb}} \vee \overline{\mathrm{Sb}}'\mathrm{Zr}))) \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathrm{Ck}_\Phi = \begin{cases} \mathrm{Ck} & \text{if } \overline{\mathrm{Zr}}_\Phi \overline{\mathrm{Zr}}_\Psi (c_2 c_1 \vee (\overline{c}_2 c_1 (\mathrm{Sb}\overline{\mathrm{Zr}} \vee \mathrm{Sb}'\mathrm{Zr}))) \\ 0 & \text{otherwise.} \end{cases}$$

where $\overline{x}$ is the complement of x and $\vee$ indicates a logical OR operation. The complete control unit which will generate the above signals can be implemented with a number of logic gates and one flip-flop.

C. Comparison

The inverter presented above will be compared with some other similar inverters, which use the canonical basis to represent either the input or the output. The inverter structures of [16] and [6]

are similar to the proposed one in the sense that they rely on Euclid's algorithm. The inverter of [16] has better area and time complexities than those of [6]. However, both use the canonical basis for their inputs as well as outputs; consequently, if the inputs are given w.r.t. the triangular basis, as it has been the case for the inverter presented in this article, then additional circuits are needed to change the basis of the inputs. The structure of [17], like the proposed inverter structure, uses two separate bases for representing the inputs and outputs; it can directly compute a division in GF(2^m), but relies on expensive Gauss-Jordan elimination method to solve its system of linear equations. A comparison of these structures on the basis of gate count, time complexity and other important VLSI design related issues is given in Table 2.

From the above table one can see that the proposed inverter has space and time complexities which are comparable to those of [16]. The main advantage of the proposed inverter seems to be in applications, *e.g.*, [18], where the input to the inverter is represented w.r.t. the triangular or dual basis, since the proposed inverter can be used without any basis change. Also, compared to the inverters of [6] and [17], the proposed inverter has less area and area-time product complexities.

D. Comments

Clock cycle distribution

With the rapid expansion of the wireless communications systems, the power dissipated by battery operated portable devices has become an important issue. If the proposed inverter is to be implemented in VLSI technologies, the average number of clock cycles that would be applied to Φ and Ψ for an inversion operation could be useful to estimate/reduce the dynamic power dissipation in the structure. Earlier, we have seen that the maximum number of clock cycles these two registers can receive to realize Step 2 of Algorithm 2 is $t_{\max} = 2m-2$. It can also be seen that the corresponding minimum number of clock cycles is $t_{\min} = m$ which occurs when the element to be inverted is equal to the multiplicative identity of GF(2^m). Our exhaustive computations of inversion operations over fields with various values of m suggest that the number of clock cycles is not evenly distributed in the range $[t_{\min}, t_{\max}]$. The actual probability distribution functions of the number of clock cycles for $m = 7,\ 8,\ 9$, and 10 are shown in Fig. 4. The corresponding average values are 11.05511811, 13.03137255, 15.01761252 and 17.00977517, respectively, and it is conjectured that the average value asymptotically approaches $2m - 3$.

Distributed control based structure

For Step 2 of Algorithm 2, the bit serial systolic structure of [15] can also be used. The structure consists of a number of identical processing cells (PEs) each of which decides the operations it needs to perform on the incoming data. Since

$$\begin{aligned} \deg R^{(-1)}(z) &= 2m-1, \\ \deg R^{(0)}(z) &\leq 2m-2, \\ \deg R^{(i-2)}(z) - \deg R^{(i)}(z) &\geq 1, \quad \forall i, \end{aligned}$$

the structure would have $2m-2$ PEs. Both the time and space complexities of such bit serial realization are linearly proportional to m. For a higher throughput, a bit parallel pipeline realization can be considered where an inverse is obtained in every time step. Such a structure would consist of a maximum of $2m-2$ PEs each of which is capable of performing the computations of one

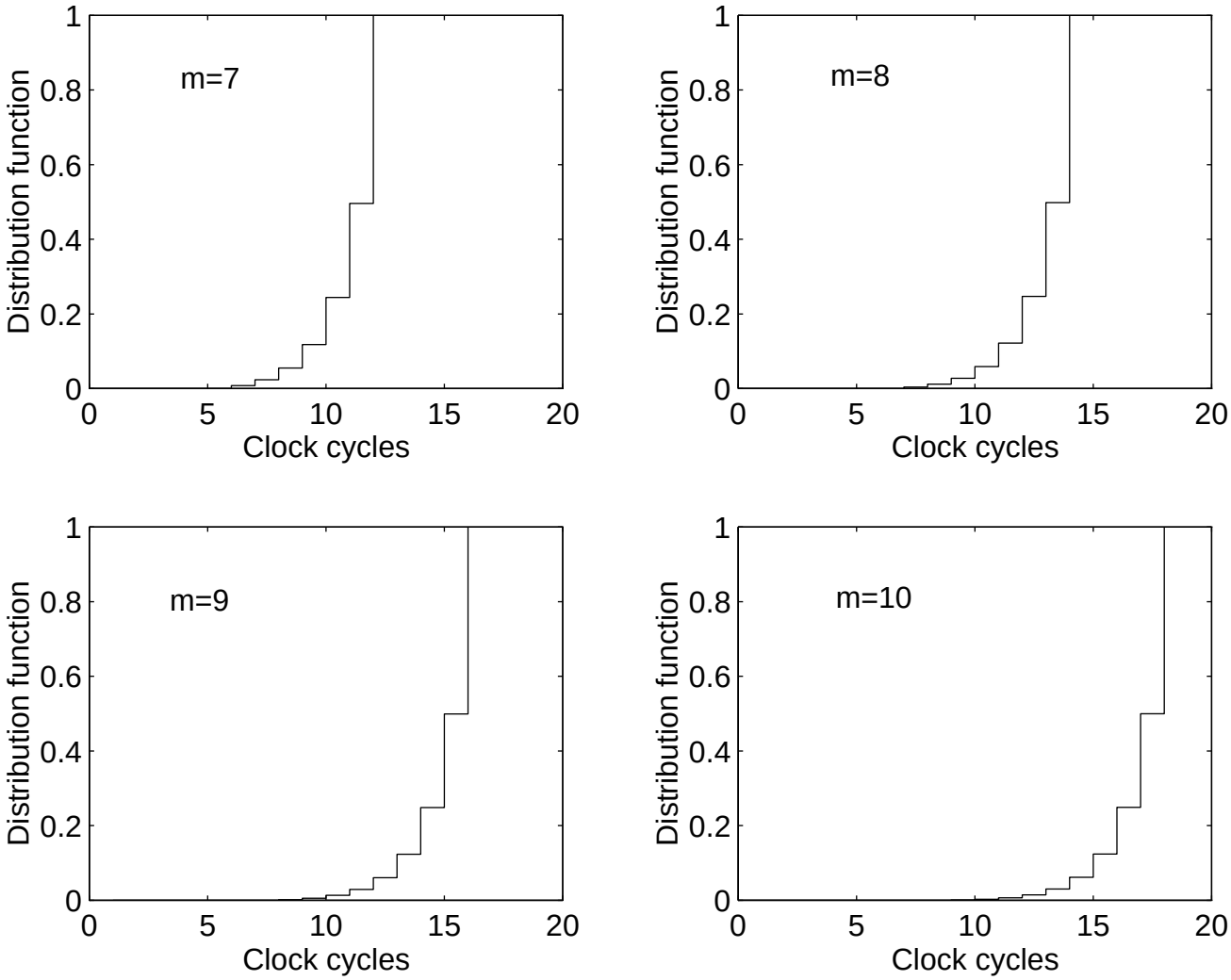

Figure 4: Probability distribution functions of the number of clock cycles for the iterations of Algorithm 2.

iteration of Step 2. The space complexity of this kind of structure is $O(m^2)$. If the chip's real-estate of a VLSI implementation is of prime concern, one can search for suitable $F(z)$'s whose use would reduce the maximum number of PEs needed. In this regard, Table 3 shows the maximum number of PEs using different primitive polynomials over GF(2). These polynomials[2] correspond to some primitive polynomials of degrees 5 to 10 most of which are listed in [19]. In the table, the polynomials are given in the octal notation, for example, 103 in the top row denotes the polynomial $z^6 + z + 1$ of degree 6.

It can be seen that the minimum number of PEs listed in Table 3 is given by

$$N_{\min} = \begin{cases} \frac{3}{2}(m-1) & m \text{ odd}, \\ \frac{3}{2}m - 1 & m \text{ even}, \end{cases} \tag{21}$$

for $5 \leq m \leq 10$. Equation (21) implies that compared to the approach of not optimizing the structure for the field generating polynomial to implement Step 2 which requires $2m - 2$ PEs, the use of the polynomials marked with underlines in Table 3 provides considerable space saving which could be up to 25% when m is odd and $\frac{m-2}{m-1} \times 25\%$ for m being even.

VI. CONCLUDING REMARKS

In this article, we have presented an algorithm and its hardware realization for computing inverses in GF(2^m) where the input is represented w.r.t. the triangular basis and the output w.r.t. the canonical basis. When the input is represented w.r.t. the *dual* basis, the proposed algorithm could also be used, which leads to a possible partial solution to the issue raised in the last paragraph of [11]. As the use of two bases has been suggested for the finite field multiplication operation, where the product is obtained w.r.t. the triangular or dual basis, the proposed inversion algorithm and its structure can be used without any basis change.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada through a Research Grant awarded to the author. The author thanks Mr. H. Wu for providing an electronic copy of binary primitive polynomials of degree up to 20. The author also thanks the reviewers for their useful comments and Dr. Ian Blake for his encouragement to pursue this topic.

References

[1] M. A. Hasan, "Double-basis inversion in GF(2^m)," *Proceedings of the 1995 Canadian Conf. on Elec. and Comp. Eng.*, pp. 229–232, 1995.

[2] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, 1976.

[3] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An Implementation of Elliptic Curve Cryptosystems over $F_{2^{155}}$," *IEEE J. on Selected Areas in Communications*, vol. 11, pp. 804–813, June 1993.

[2] Polynomials with degrees between 5 and 10 are frequently used in error control coding schemes.

[4] M. A. Hasan, V. K. Bhargava, and T. Le-Ngoc, "Algorithms and Architectures for the Design of a VLSI Reed-Solomon Codec," in *Reed-Solomon Codes and Their Applications* (S. B. Wicker and V. K. Bhargava, eds.), ch. 4, pp. 60–107, IEEE Press, June 1994.

[5] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI Architecture for Computing Multiplications and Inverses in GF(2^m)," *IEEE Trans. Comput.*, vol. C-34, pp. 709–717, Aug. 1985.

[6] K. Araki, I. Fujita, and M. Morisue, "Fast Inverter over Finite Field Based on Euclid's Algorithm," *Trans. IEICE*, vol. E 72, pp. 1230–1234, Nov. 1989.

[7] M. A. Hasan and V. K. Bhargava, "Bit-Serial Systolic Divider and Multiplier for GF(2^m)," *IEEE Trans. Comput.*, vol. 41, pp. 972–980, Aug. 1992.

[8] M. A. Hasan and V. K. Bhargava, "Division and Bit-Serial Multiplication over GF(q^m)," *IEE Proc. Part E*, vol. 139, pp. 230–236, May 1992.

[9] C. Paar, "Some Remarks on Efficient Inversion in Finite Fields," *Proc. IEEE International Symposium on Information Theory*, p. 58, 1995.

[10] E. R. Berlekamp, "Bit-Serial Reed-Solomon Encoder," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 869–874, Nov. 1982.

[11] M. Morii, M. Kasahara, and D. L. Whiting, "Efficient Bit-Serial Multiplication and the Discrete-Time Wiener-Hopf Equations over Finite Fields," *IEEE Trans. Inform. Theory*, vol. IT-35, pp. 1177–1183, Nov. 1989.

[12] M. Z. Wang and I. F. Blake, "Bit Serial Multiplication in Finite Fields," *SIAM J. Disc. Math.*, vol. 3, pp. 140–148, Feb. 1990.

[13] M. A. Hasan and V. K. Bhargava, "Architecture for a Low Complexity Rate-Adaptive Reed-Solomon Encoder," *IEEE Trans. Comput.*, pp. 938–942, June 1995. IEEE Trans. Comput.

[14] Y. Sugiyama, "An Algorithm for Solving Discrete-Time Wiener-Hopf Equations Based on Euclid's Algorithm," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 394–409, 1986.

[15] R. P. Brent and H. T. Kung, "Systolic VLSI arrays for polynomial GCD computation," *IEEE Trans. Comput.*, vol. 33, no. 8, pp. 731–736, 1984.

[16] H. Brunner, A. Curiger, and M. Hofstetter, "On Computing Multiplicative Inverses in GF(2^m)," *IEEE Trans. Comput.*, vol. 42, pp. 1010–1015, Aug. 1993.

[17] S. T. J. Fenn, M. Benessia, and D. Taylor, "GF(2^m) Multiplication and Division Over the Dual Basis," *IEEE Trans. Computers*, vol. 45, pp. 319–327, Mar. 1996.

[18] M. A. Hasan, "Division-and-Accumulation over GF(2^m)," *IEEE Trans. Comput.*, vol. 46, pp. 705–708, June 1997.

[19] R. E. Ziemer and R. L. Peterson, *Digital Communications and Spread Spectrum Systems.* Macmillan, NY, 1985.

Components/ Features	Fenn *et al.* [17]	Brunner *et al.* [16]	Araki *et al.* [6]	Proposed here
Flip-flops	$O(m^2)$	$4m$	$7(m+1)$	$6m$
AND/OR gates	$O(m^2)$	$3m$	$5(m+1)$	-
XOR gates	$O(m^2)$	$3m$	$4(m+1)$	$3m + (m-1)(W_F - 2)$
MUX	$O(m^2)$	$8m$	$21(m+1)$	$8m$
Other circuits	-	1 counter + 1 small control	-	3 counters + 1 small control
$F(z)$ dependency	-	$z.U$ & U/z	-	$h_m, h_{m+1}, \cdots, h_{2m-2}$
Global control signals	-	4	4	2
Clock cycles per inversion	m	$2m$	$2m+3$ to $3m+2$	$2m$
Input-output delay (in clock cycles)	$5m-1$	ditto	ditto	ditto
Clock period	$O(1)$	$O(1)$	$O(m)$	$O(1)$
AT-Product	$O(m^3)$	$O(m^2)$	$O(m^3)$	$O(m^2)$

Table 2: Comparison of inverter structures for GF(2^m) generated by a field-defining polynomial $F(z)$ of Hamming weight W_F. The calculation of the input-output delay assumes parallel-in and parallel-out data. Clock and clear signals for the flip-flops are excluded from the count of global signals.

Degree	Polynomial	Max. PEs	Degree	Polynomial	Max. PEs
5	45	8	6	103	8
5	75	6	6	147	8
5	67	6	6	155	8
7	211	10	8	435	11
7	217	10	8	551	12
7	235	10	8	747	12
7	367	10	8	453	13
7	277	9	8	545	12
7	325	11	8	537	11
7	203	10	8	703	12
7	313	10	8	543	11
9	1021	13	10	2011	14
9	1131	14	10	2415	14
9	1461	13	10	3771	14
9	1423	13	10	2157	15
9	1055	13	10	3515	16
9	1167	16	10	2773	14
9	1541	13	10	2033	15
9	1333	14	10	2443	14
9	1725	12	10	2461	14
9	1751	14	10	3023	15
9	1743	15	10	3543	17
9	1617	13	10	2745	15
9	1553	14	10	2431	14
9	1157	14	10	3177	16

Table 3: Maximum number of PEs using different primitive polynomials for the distributed control based bit parallel inverter structure.