EFFICIENT COMPUTATIONS IN GALOIS FIELDS

by

Mohammed Anwarul Hasan
B. Sc., 1986 and M. Sc., 1988
Bangladesh University of Engineering & Technology, Dhaka

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

We accept this dissertation as conforming
to the required standard

Dr. Vijay K. Bhargava, Supervisor (Department of ECE)

Dr. Fayez El-Guibaly, Departmental Member (Department of ECE)

Dr. Kin F. Li, Departmental Member (Department of ECE)

Dr. Gholamali C. Shoja, Outside Member (Department of CS)

Dr. Ruediger Vahldieck, Graduate Advisor (Department of ECE)

Dr. T. Aaron Gulliver, External Member (Carleton University)

ABSTRACT

OF THE DISSERTATION

EFFICIENT COMPUTATIONS IN GALOIS FIELDS

by

Mohammed Anwarul Hasan

Supervisor: Professor Vijay K. Bhargava

In this dissertation some algorithms and related hardware structures for computing division and multiplication over finite or Galois fields are presented. The structures are regular, which is important for hardware realization, particularly for large finite fields.

The concept of *supporting* elements is introduced which leads to efficient algorithms for computing divisions and multiplications in finite fields. A relationship between systems of linear equations over GF(q) and division in GF(q^m) is established. Using this relationship, a division algorithm valid for any irreducible polynomial or any field basis is presented. It is also proved that if the elements are represented with respect to a canonical basis, then division over GF(q^m) can be performed by solving a discrete time Wiener-Hopf equation over GF(q).

A bit-serial systolic divider for finite fields of the form GF(2^m) is presented. The divider structure does not depend on the irreducible polynomial defining the field and requires no global data communications. Moreover, the time step duration is independent of the value of m, which is important for large finite fields.

By exploiting the structure of a Toeplitz matrix, a bit-serial multiplier applicable to any irreducible polynomial defining the field is presented. The multiplier is efficient in the sense that it requires, in general, less circuitry compared to equivalent existing multipliers.

Finite fields GF(2^m) generated by irreducible *all one polynomials* (AOP) and *equally space polynomials* (ESP) are considered. Algorithms and structures are presented for parallel computation of multiplications in these fields. It is shown

that if for a certain degree both an irreducible AOP and ESP exist, it is advantageous to use an ESP based parallel multiplier. Moreover, it is shown that parallel multipliers based on ESP can be obtained by using modules of a corresponding AOP based multiplier.

Finally, as an application of the efficient bit-serial multiplication algorithm, a Reed-Solomon encoder structure is presented. The structure features simple basis transformation circuitry and supports a variable code rate.

Examiners:

Dr. Vijay K. Bhargava, Supervisor (Department of ECE)

Dr. Fayez El-Guibaly, Departmental Member (Department of ECE)

Dr. Kin F. Li, Departmental Member (Department of ECE)

Dr. Gholamali C. Shoja, Outside Member (Department of CS)

Dr. Ruediger Valdieck, Graduate Advisor (Department of ECE)

Dr. T. Aaron Gulliver, External Member (Carleton University)

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AOP	All One Polynomial
BCH	Bose-Choudhuri-Hocquenghem
CM	Coefficient Matrix
DA	Division Algorithm 1
DTWHE	Discrete Time Wiener-Hopf Equations
ESP	Equally Spaced Polynomial
GF	Galois Field
GJE	Gauss-Jordan Elimination
GP	Generator Polynomial
ITM	Itoh-Tsujii Multiplier
LFSR	Linear Feedback Shift Register
MOM	Massey-Omura Multiplier
PISO	Parallel In Serial Out
ROM	Read Only Memory
RS	Reed-Solomon
SAFCM	Systolic Array for the Formation of the Coefficient Matrix
SASLE	Systolic Array for Solving Linear Equations
VLSI	Very Large Scale Integration

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Professor Vijay K. Bhargava, for his guidance and encouragement throughout my graduate studies at the University of Victoria. I would also like to thank Dr. Muzhong Wang for his helpful suggestions and comments on various aspects of my research work. I thank Professor Fayez El-Guibaly for his helpful discussions. Special thanks go to Professor Ian F. Blake for his encouragement. Thanks are also due to David Peterson for proof reading part of the dissertation.

I would like to acknowledge the scholarship awarded to me by the Canadian Commonwealth and Fellowship Committee, and the top-up financial support by the Canadian Institute for Telecommunications Research through a grant to Professor Vijay K. Bhargava.

To *my parents*

Chapter 1

Introduction

1.1 Motivation

A *finite field* is a set with a finite number of elements, where it is possible to add, subtract, multiply and divide (by nonzero elements) without leaving the set. Addition and multiplication must also satisfy commutative, associative and distributive laws.

The theory of finite fields is a branch of modern algebra. The origins of the subject can be traced back to the 17th and 18th centuries. During this period, eminent mathematicians such as Pierre de Fermat (1601-1665), Leonhard Euler (1707-1783), Joseph-Louis Lagrange (1736-1813) and Adrien-Marie Legendre (1752-1833) contributed to the structure theory of finite prime fields. The general theory of finite fields began with the work of Carl Friedrich Gauss (1777-1855) and Evariste Galois (1811-1832). With the recent emergence of discrete mathematics as an important applied discipline, finite fields have also become of interest to applied mathematicians. A finite field is also called a *Galois field* after the mathematician Evariste Galois. A Galois field with p elements is denoted by GF(p) [25].

Galois fields play an important role in error-control coding and cryptography. Error-control coding techniques are used for efficient and reliable digital data transmission and storage systems. Cryptographic techniques are used to provide security for many communication systems. Here we briefly illustrate two cases where

computations in Galois fields are involved.

Error-control coding: The origin of error-control coding lies with the work of Hamming [10]. During the last four decades, the theory of finite fields and the theory of polynomials over finite fields have been applied to the design of good codes and efficient decoding methods. BCH (Bose-Choudhuri-Hocquenghem) [17] codes and the related RS (Reed-Solomon) [34] codes are widely used codes. A number of efficient algorithms are available for encoding and decoding these codes. The following steps give a general outline of decoding algorithms for non-binary BCH codes [5].

1. Compute the *syndromes.*
2. Find the *error-location polynomial.*
3. Compute the *error-locations* and *error values.*

Computations in Galois fields are involved in all three steps. The syndrome computation requires multiplication, addition and subtraction operations, and in addition to these operations, steps 2 and 3 require Galois field inversions and divisions.

Cryptography: The design and breaking of systems for secret communication is the subject of cryptography. Such systems are called *cryptosystems.* The Diffie-Hellman scheme [8] is a well known key-exchange protocol for cryptosystems where Galois field computations are required. The basic idea behind this protocol is as follows. Let b be a fixed *primitive element* of GF(q). Suppose users A and B wish to communicate using a non-secure channel. They choose private numbers h and k, respectively where $2 \leq h, k \leq q-2$. A then sends b^h to B, while B transmits b^k to A. Both take b^{hk} as their common key, which can be computed by A as $(b^k)^h$ and by B as $(b^h)^k$.

In addition to coding and cryptography, finite fields have applications in switching theory [4], digital signal processing [35] and VLSI testing [9]. Among the different computational operations in finite fields, division and multiplication are widely

used in practical applications. Thus there is a need for good algorithms for such operations which can be easily realized in hardware.

1.2 Historical Perspective

The basic finite field operations are addition, subtraction, multiplication and inversion (more generally division). The complexity of the logic circuitry required to perform these operations depends on the particular representation of the field elements [43]. If the elements of GF(2^m) are represented as a power of a primitive element of the field, multiplication and inversion operations can be easily performed. However, addition and subtraction operations are difficult. In practice, each element of GF(2^m) is usually represented by an m-tuple whose elements can be considered as coefficients of a polynomial over GF(2) of degree less than m. With such a representation, addition and subtraction are very simple but multiplication and inversion operations constitute a formidable problem.

1.2.1 Multiplication

Let the m-tuple representations of two elements a and b of GF(2^m) be (a_0, a_1, $\cdots$, a_{m-1}) and (b_0, b_1, $\cdots$, b_{m-1}). Each of the m coordinates of the product is a linear combination of the m^2 binary products $a_k b_j$ ($0 \leq k, j \leq m-1$). Bartee and Schneider suggest a direct implementation of the multiplication by combinational logic [3]. They use a canonical basis to represent the elements of the field. Depending on the irreducible polynomial, the implementation requires as many as $m^3 - m$ two-input adders over GF(2) [5]. Even with a wise choice of the irreducible polynomial, the number of two-input modulo-2 adders tends to be so large that the method is quite expensive for large m. Subsequent approaches to the multiplication operation in GF(2^m) by Law and Rushforth [24], Yeh, Reed and Truong [46], Scott, Tavares and Peppard [36], and Zhang [47] are suitable for VLSI implemen-

tation. All these multipliers are based on a canonical basis representation of the field elements. More about VLSI architectures for different finite field multipliers can be found in [29].

In the last decade, two important contributions to multiplication in $GF(2^m)$ were made. One is the dual basis bit-serial multiplication algorithm by Berlekamp [6] and the other is the normal basis multiplication algorithm by Massey and Omura [28]. The representation of the field elements with respect to a normal basis is unconventional, but results in a very simple squaring operation. This is advantageous for the design of inversion and exponentiation circuitry. Multiplication using the Massey-Omura algorithm requires the same logic circuitry for all product coordinates. On the other hand, in Berlekamp's multiplication algorithm, one factor (the multiplicand) is represented with respect to a canonical basis and the other factor (the multiplier) with respect to the corresponding dual basis. The product is obtained with respect to the dual basis. The advantage of Berlekamp's bit-serial multiplier is that it requires minimum circuitry when the multiplicand is a constant.

The involvement of two bases in Berlekamp's bit-serial multiplication algorithm is not advantageous, especially when the multiplier is to be used as part of a larger circuit. In general, the canonical basis representation of both the multiplicand and multiplier are available. The product is also expected to be represented relative to the same basis. As a result, circuitry is required at the input to transform one of the factors (the multiplier) from the canonical basis to the dual basis and at the output to transform the product from the dual basis back to the canonical basis. Recent work by Morii, Kasahara and Whiting [31] shows that efficient bit-serial multiplication can be obtained if the irreducible polynomial is a trinomial. However, irreducible trinomials do not exist for all degrees. Wang and Blake [44] present a bit-serial multiplier featuring regular basis transformation circuitry at the input and output which works for all irreducible polynomials. However, this requires additional circuitry at both the input and output.

1.2.2 Division

The division of a field element γ by another field element β can be viewed as a multiplication of γ by β^{-1}. Thus division consists of a multiplication and an inversion. In general, computing the inverse of a finite field element is more complex than the multiplication of two field elements. For small values of m, the inverse can be obtained by a look-up table. The table is simply a ROM (Read Only Memory) containing all inverse elements which is addressed using the element to be inverted. Since the size of the ROM grows exponentially with m, this method is not attractive for large values of m.

Since Galois field elements can be represented by polynomials, inversion can be accomplished by Euclid's algorithm. This algorithm requires repetitive polynomial divisions and multiplications. A modular structured inverter based on Euclid's algorithm has been developed by Akari *et al.* [1]. The structure requires complicated control signals and the time complexity increases with the square of m. Moreover, the computation time is not the same for all elements of the field.

An inversion algorithm which has achieved recent prominence in the literature, especially from the viewpoint of implementation, is based on Fermat's theorem. It requires repetitive squaring and multiplying operations. When field elements are represented with respect to a normal basis, the squaring operation is simply a cycle shift of coordinates. However, multiplication using a normal basis requires circuitry which is highly dependent on the irreducible polynomial defining the field. The design of such circuitry poses a formidable task for large values of m.

Inversion in GF(2^m) can be computed by solving a system of linear equations over GF(2). Davida has shown that inversion can be performed by solving $2m-1$ linear equations in $2m-1$ unknowns [7]. A more efficient algorithm by Morii, Kasahara and Whiting [31] requires the solution of only m equations in m unknowns to compute a division directly. However, the formation of the system of equations is computationally intensive and there is no regular structure underlying

this division algorithm.

In light of the above discussion it is, therefore, advantageous to achieve the following goals:

- To develop efficient algorithms for computations of division and multiplication in Galois fields;
- To map the algorithms onto suitable structures.

This dissertation will address these problems.

1.3 Dissertation Outline

The dissertation is arranged as follows:

In Chapter 2, the mathematics of finite fields is discussed. Definitions and fundamental theorems on finite fields which relate to the subsequent chapters are presented.

In Chapter 3, the concept of supporting elements is formally presented. Then an algorithm for computing divisions in GF(q^m) based on supporting elements is derived. The division algorithm is general, in the sense that it is applicable for any basis representation of the field elements and for any irreducible polynomial defining the field. A relationship between discrete time Wiener-Hopf equations (DTWHE) and Galois field division is established. This leads to an efficient division algorithm for the canonical basis representation of the field elements.

Chapter 4 presents a bit-serial systolic divider for GF(2^m). An algorithm for the formation of a so-called coefficient matrix is given. This algorithm is mapped onto a one dimensional systolic array. Using Gauss-Jordan diagonalization over GF(2), an efficient two dimensional systolic array is developed. These two arrays are used to obtain a bit-serial systolic divider over GF(2^m).

In Chapter 5, the relationship between the DTWHE and Galois field division is exploited to develop a bit-serial multiplier. A relationship yielding the coefficients

of the DTWHE using linear feedback shift registers is derived.

Chapter 6 presents multiplication algorithms for a class of finite fields GF(2^m) generated by irreducible *all one polynomials* (AOP) and *equally spaced polynomials* (ESP). Structures for low complexity AOP and ESP based parallel multipliers are developed. It is also shown how a multiplier for a very large field can be constructed from the modules of an AOP based multiplier for a corresponding small field.

In Chapter 7 we present an application of the efficient bit-serial multiplier developed in Chapter 5. The complexity of a Reed-Solomon (RS) encoder depends on the finite field multiplier used. Using the bit-serial multiplication algorithm, an RS encoder is developed which has a low circuit complexity and supports a variable code rate.

Chapter 8 concludes the dissertation with a summary of results and suggestions for future research.

1.4 Research Contributions

The major contribution of this dissertation is the development of efficient algorithms for computing multiplication and division in finite fields. The attractiveness of the algorithms is that their realizations are, in general, area efficient and can be used for applications where fast computation is necessary.

Some specific contributions of the dissertation are as follows:

- Development of a general finite field division algorithm for any irreducible polynomial or any basis representation for the field.
- Establishment of a relationship between discrete time Wiener-Hopf equations and division over GF(q^m).
- Development of a bit-serial systolic divider for GF(2^m).
- Development of a bit-serial multiplication algorithm and structure for GF(q^m).

- Development of a low complexity parallel multiplier for a class of finite fields GF(2^m).
- Presentation of a method for constructing parallel multipliers for a very large finite field from the basic modules of a multiplier for the corresponding small field.
- Development of a structure for a variable error-correcting Reed-Solomon encoder.

Chapter 2

Mathematical Background

2.1 Introduction

This chapter gives some useful definitions, theorems and properties of finite fields. It covers only those topics which are relevant to the discussions of the subsequent chapters. Theorems and statements are given without any proof. Proofs and further details can be found in the literature, for example, [26], [5], [27] and [25].

2.2 Finite Fields

Let G be a set of elements. A *binary operation* $*$ on G is a rule that assigns to each pair of elements a and b a uniquely defined third element $c = a * b$ in G. When such a binary operation $*$ is defined on G, the latter is said to be *closed* under $*$. For example, let G be the set of all integers and let the binary operation on G be conventional addition '+'. For any two integers i and j in G, $i + j$ is a uniquely defined integer in G. Hence, the set of integers is closed under conventional addition.

A binary operation $*$ on G is said to be *associative* if, for any a, b and c in G,

$$a * (b * c) = (a * b) * c.$$

Definition 2.1 [26] A set G on which a binary operation $*$ is defined is called a *group* if the following conditions are satisfied:

(i) The binary operation $*$ is associative.

(ii) G contains an element e such that, for any $a \in G$,

$$a * e = e * a = a$$

This element e is called an identity element of G.

(iii) For any element $a \in G$, there exists another element $a' \in G$ such that

$$a * a' = a' * a = e$$

The element a' is called an inverse of a.

Theorem 2.1 [26] The identity element of a group is unique.

Theorem 2.2 [26] The inverse of a group element is unique.

A group G is said to be commutative if its binary operation $*$ also satisfies the following condition: For any a and b in G,

$$a * b = b * a.$$

The set of all integers is a commutative group under conventional addition. In this case, the integer 0 is the identity element and the integer $-i$ is the inverse of integer i. The set of all rational numbers excluding zero is a commutative group under conventional multiplication. The integer 1 is the identity element with respect to conventional multiplication, and the rational number b/a is the multiplicative inverse of a/b. The groups noted above contain an infinite number of elements. Groups with a finite number of elements do exist; for example, the set of two integers $\{0, 1\}$.

The group concepts are used to introduce a *field*, a formal definition of which is given below.

Definition 2.2 [26] Let F be a set of elements in which two binary operations, called addition '+' and multiplication '$\cdot$' are defined. The set F together with the two binary operations $+$ and $\cdot$ is a field if the following conditions are satisfied:

(i) F is commutative group under addition. The identity element with respect to addition is called the *zero* element or the additive identity of F and is denoted by 0.

(ii) The set of nonzero elements in F is a commutative group under multiplication. The identity element with respect to multiplication is called the *unit* element or the multiplicative identity of F and is denoted by 1.

(iii) Multiplication is *distributive* over addition; that is, for any three elements a, b and c in F,

$$a \cdot (b + c) = a \cdot b + a \cdot c.$$

The *order* of the field is defined as the number of elements in the field. A field with a finite number of elements is called a *finite field.* In a field, the additive inverse of an element a is denoted by $-a$, and the multiplicative inverse of a is denoted by a^{-1}, provided $a \neq 0$. Subtraction of a field element b from another field element a is defined as adding the additive inverse of b to a, i.e., $a - b \triangleq a + (-b)$. If b is a nonzero element, dividing a by b is defined as multiplying a by the multiplicative inverse of b, i.e., $a \div b \triangleq a \cdot b^{-1}$.

For example, the set of real numbers is a field under real number addition and multiplication. This field has an infinite number of elements. For p a prime number, the set $\{0, 1, \cdots, p-1\}$ is a field of order p under modulo-p addition and multiplication. Since this field is constructed from a prime p, it is called a prime field and is denoted by GF(p). For $p = 2$, we obtain the simple binary field GF(2).

Theorem 2.3 [5] For p prime and k a positive integer, there exists a unique finite field of order p^k. This field is called the Galois field of order p^k and is denoted by GF(p^k).

Definition 2.3 [5] The least positive integer c for which $\sum_{i=1}^{c} 1 = 0$ in a field is called the characteristic of the field.

If $\sum_{i=1}^{n} 1$ is nonzero for every integer n, then the field is said to have characteristic ∞.

Theorem 2.4 [5] The characteristic of any finite field is prime.

Theorem 2.5 [5] If $w_1, w_2, \cdots, w_k$ are elements in a field of characteristic p, then

$$\left(\sum_{i=1}^{k} w_i\right)^{p^n} = \left(\sum_{i=1}^{k} w_i^{p^n}\right) \quad \text{for all } n. \tag{2.1}$$

If a finite field contains an element α, then it must also contain the *powers* of α: $\alpha, \alpha^2, \alpha^3, \cdots$. The least positive integer for which $\alpha^n = 1$ is called the order of α. Then the following theorem is immediate.

Theorem 2.6 [5] If α has order n, then $\alpha^m = 1$ if and only if m is a multiple of n.

In a field of order p, a nonzero element α is said to be primitive if the order of α is $p-1$ [26].

Theorem 2.7 [5] A finite field of order p must contain a primitive field element whose order is $p-1$ and whose powers include all nonzero field elements.

Theorem 2.8 [5] Every element in a field of order p satisfies the equation $x^p - x = 0$.

2.3 Polynomials over Finite Fields

Let F be an arbitrary finite field. A polynomial over F in the indeterminate x is an expression of the form

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots$$

in which $a_i \in F$ for all i; but at most a finite number of the coefficients a_i are nonzero. The powers of the indeterminate are always integer. If $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ with $a_n \neq 0$, then n is called the degree of $A(x)$ and is denoted by $\deg(A(x))$. If the leading coefficient a_n is 1, then $A(x)$ is called a *monic* polynomial.

Definition 2.4 [26] The reciprocal of $A(x)$, denoted as $A^*(x)$, is defined by

$$A^*(x) = x^n A\left(x^{-1}\right) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n.$$

Definition 2.5 [26] A polynomial $A(x)$ is irreducible over F if $A(x)$ is only divisible by c or by $cA(x)$ where $c \in F$.

Definition 2.6 [27] The minimal polynomial $M(x)$ over GF(p), where p is prime, of $\beta \in \mathrm{GF}(p^m)$ is the lowest degree monic polynomial with coefficients from GF(p) such that $M(\beta) = 0$.

It can be show that $M(x)$ is unique and irreducible over GF(p). If $\beta \in \mathrm{GF}(p^m)$, then $\deg(M(x)) \leq m$.

Definition 2.7 [27] The minimal polynomial of a primitive element of $\mathrm{GF}(p^m)$ is called a primitive polynomial.

Irreducible polynomials are used to construct finite fields. The following theorems are related to irreducible polynomials.

Theorem 2.9 [25] For every finite field GF(q) and every positive integer m there exists an irreducible polynomial over GF(q) of degree m.

Theorem 2.10 [25] If $f(x)$ is an irreducible polynomial over GF(q) of degree m, then $f(x)$ has a root $\alpha \in \mathrm{GF}(q^m)$. Furthermore, all the roots of $f(x)$ are given by the m distinct elements $\alpha, \alpha^q, \cdots, \alpha^{q^{m-1}}$ of GF(q^m).

2.4 Bases and Field Element Representation

The field GF(p^m), where p is prime and m a positive integer, can be considered as a vector space of dimension m over GF(p). Any set of m linearly independent elements can be used as a basis for this vector space.

Definition 2.8 [27] The trace of $\beta \in \mathrm{GF}(p^m)$ is defined as follows:

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

The trace has the following important properties:

1. $\mathrm{Tr}(\beta + \gamma) = \mathrm{Tr}(\beta) + \mathrm{Tr}(\gamma)$, where β and γ are in GF(p^m).
2. $\mathrm{Tr}(\beta^p) = \mathrm{Tr}(\beta)^p = \mathrm{Tr}(\beta)$.
3. $\mathrm{Tr}(1) = m \pmod p$.

Definition 2.9 [25] Two bases $\{\lambda_0, \lambda_1, \cdots, \lambda_{m-1}\}$ and $\{\gamma_0, \gamma_1, \cdots, \gamma_{m-1}\}$ of GF(p^m) over GF(p) are called dual (or complementary) bases if for $0 \leq i, j \leq m-1$ we have

$$\mathrm{Tr}(\gamma_i \lambda_j) = \delta_{i,j}$$

where $\delta_{i,j}$ is Kronecker delta function which is 1 if $i = j$ and 0 otherwise.

Theorem 2.11 [27] Every basis has a dual basis.

Although the number of different bases of GF(p^m) over GF(p) is large [25], there are two special types of bases of practical importance. The first one is a canonical (or polynomial) basis $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$, made up of consecutive powers of a defining element α of GF(p^m) over GP(p). Another type of basis is a normal basis defined by a suitable element of GF(p^m).

Definition 2.10 [25] The set of elements of the form $\{\alpha, \alpha^p, \cdots, \alpha^{p^{m-1}}\}$, consisting of a suitable element of $\alpha \in$ GF(p^m) with respect to GF(p) is called a normal basis of GF(p^m) over GF(p).

Theorem 2.12 [25] For any finite field K and any extension F of K, there exists a normal basis of F over K.

The elements of a finite field GF(q) with $q = p^m$ elements, where p is the characteristic of GF(q), can be represented in three ways, viz., matrix, power and polynomial representations which are briefly discussed below.

The companion matrix of the monic polynomial $f(x) = a_0 + a_1x + \cdots + a_{m-1}x^{m-1} + x^m$ of degree m over a field is defined to be the $m \times m$ matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -a_{m-1} \end{bmatrix}.$$

It is well known that $\mathbf{A}$ satisfies $f(\mathbf{A}) = 0$, i.e., $a_0\mathbf{I} + a_1\mathbf{A} + a_2\mathbf{A}^2 + \cdots + \mathbf{A}^m = 0$ where $\mathbf{I}$ is the $m \times m$ identity matrix. As a result, if $\mathbf{A}$ is the companion matrix of an irreducible polynomial $f(x)$ over GF(p) then the polynomials in $\mathbf{A}$ over GF(p) of degree less than m yield a representation of the elements of GF(q) [25].

This matrix representation method is a very laborious way of describing the field. Computations involving the field elements are tedious as they require matrix operations.

The second possibility of representing the elements is by means of powers of a primitive element of the field. Since the order of a primitive element of GF(q) is $q-1$, all the nonzero elements of the field can be expressed as powers of the primitive element. The zero element is considered as the $-\infty$ power of the primitive element. With this representation, multiplication and division operations are simple, but addition and subtraction operations are not. Moreover, locating a primitive element is not always trivial [42].

The third method of representing the field elements is to express them as algebraic sums of m linearly independent elements. The set of m linearly independent elements forms a basis, e.g., a normal or a canonical basis. In a canonical basis of the form $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$, the element $\alpha \in \mathrm{GF}(p^m)$ is often taken to be a primitive element. However, if α is simply a root of the irreducible polynomial $f(x)$ over GF(p) of degree m, then every element of GF(p^m) can also be uniquely expressed as a polynomial in α over GF(p) of degree less than m [25]. This is a very convenient way to form a basis.

Chapter 3

Division Algorithms

3.1 Introduction

The well known methods to compute inverses are based either on Euclid's algorithm or Fermat's theorem. Inversion based on Euclid's algorithm requires polynomial divisions and multiplications; and inversion based on Fermat's theorem requires recursive squaring and multiplication operations over finite fields. In addition to these two methods, inversion of an element of GF(2^m) can also be performed by solving a set of simultaneous linear equations over GF(2). It has been shown that the inverse can be computed by solving $2m-1$ simultaneous linear equations in $2m-1$ unknowns over GF(2) [7]. However, a more efficient inversion method based on the solution of linear equations over GF(2) has recently been developed by Morii, Kasahara and Whiting [31].

In this chapter, two division algorithms over GF(q^m), where q is prime and m is a positive integer, are presented [12]. The algorithms use the so-called *supporting* elements. It is shown that when the field elements are represented as polynomials using any suitable basis, division over GF(q^m) can be performed by solving a system of m linear equations of a general form over GF(q); and for a canonical basis representation, a division can be performed by solving discrete time Wiener-Hopf equations (DTWHE) over GF(q) with $2m-1$ constants.

3.2 Supporting Elements

GF(q^m) is an extension field of GF(q) where q is a prime and m is a positive integer. The extension field has q^m elements. Let

$$g(z) = \sum_{i=0}^{m} g_i z^i$$

be an irreducible monic polynomial over GF(q) of degree m; $g(z)$ has a root α in GF(q^m). Then any element $a \in$ GF(q^m) can be represented as a polynomial of powers of α over GF(q) i.e., $a = a_0\alpha^{k_0} + a_1\alpha^{k_1} + a_2\alpha^{k_2} + \cdots + a_{m-1}\alpha^{k_{m-1}}$, where the coordinates $a_i \in$ GF(q) for $0 \le i \le m-1$ and $\{\alpha^{k_0}, \alpha^{k_1}, \cdots, \alpha^{k_{m-1}}\}$ is a basis of GF(q^m) over GF(q). The row vector a is denoted as

$$\mathbf{a} = [a_0, a_1, \cdots, a_{m-1}].$$

Define the set H as

$$H = \{\alpha^{k_i+k_j}\} \quad i,j = 0, 1, \cdots, m-1. \tag{3.1}$$

The elements of the set H are hereafter referred to as the *supporting* elements. The coordinates of these supporting elements are used in the following analyses. To distinguish these coordinates, they are denoted by superscripts as follows:

$$\alpha^n = \sum_{i=0}^{m-1} p_i^{[n]} \alpha^{k_i}. \tag{3.2}$$

Thus $p_i^{[n]}$ is the i-th coordinate of the supporting element α^n. We denote $\mathrm{p}_i^{[k_j]}$ as a column vector whose components are the ith coordinates of the supporting elements $\alpha^{k_0+k_j}, \alpha^{k_1+k_j}, \cdots, \alpha^{k_{m-1}+k_j}$; i.e.,

$$\mathrm{p}_i^{[k_j]} = \left[p_i^{[k_0+k_j]}, p_i^{[k_1+k_j]}, \cdots, p_i^{[k_{m-1}+k_j]}\right]^t. \tag{3.3}$$

3.3 A Generalized Division Algorithm

The conventional way to perform division c/a in a finite field is to first compute the multiplicative inverse of a and then multiply the inverse with c. The following theorem states that division in the finite field can be computed in an alternate way.

Theorem 3.1 Let $g(z)$ be an irreducible polynomial over GF(q) and a, b and $c \in$ GF(q^m). Let the elements be represented by a suitable basis of the form $\{\alpha^{k_0}, \alpha^{k_1}, \cdots, \alpha^{k_{m-1}}\}$. Then the division $b = c/a$, $a \neq 0$, in the finite field GF(q^m) can be performed by solving the following equations over GF(q)

$$\begin{bmatrix} \mathbf{a} \cdot \mathbf{p}_{m-1}^{[k_{m-1}]} & \mathbf{a} \cdot \mathbf{p}_{m-1}^{[k_{m-2}]} & \cdots & \mathbf{a} \cdot \mathbf{p}_{m-1}^{[k_0]} \\ \mathbf{a} \cdot \mathbf{p}_{m-2}^{[k_{m-1}]} & \mathbf{a} \cdot \mathbf{p}_{m-2}^{[k_{m-2}]} & \cdots & \mathbf{a} \cdot \mathbf{p}_{m-2}^{[k_0]} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a} \cdot \mathbf{p}_0^{[k_{m-1}]} & \mathbf{a} \cdot \mathbf{p}_0^{[k_{m-2}]} & \cdots & \mathbf{a} \cdot \mathbf{p}_0^{[k_0]} \end{bmatrix} \begin{bmatrix} b_{m-1} \\ b_{m-2} \\ \vdots \\ b_0 \end{bmatrix} = \begin{bmatrix} c_{m-1} \\ c_{m-2} \\ \vdots \\ c_0 \end{bmatrix} \tag{3.4}$$

where "$\mathbf{x} \cdot \mathbf{y}$" denotes the inner product of $\mathbf{x}$ and $\mathbf{y}$.

Proof: The polynomial representations of a, b and c are

$$a = \sum_{l=0}^{m-1} a_l \alpha^{k_l},$$

$$b = \sum_{j=0}^{m-1} b_j \alpha^{k_j}$$

and

$$c = \sum_{i=0}^{m-1} c_i \alpha^{k_i}$$

where the coordinates a_i, b_i, c_i are in GF(q) for $0 \leq i \leq m-1$. Then

$$\begin{aligned} c &= ab \\ &= \sum_{l=0}^{m-1} a_l \alpha^{k_l} \sum_{j=0}^{m-1} b_j \alpha^{k_j} \pmod{g(\alpha)} \\ &= \sum_{j=0}^{m-1} b_j \sum_{l=0}^{m-1} a_l \alpha^{k_l+k_j} \pmod{g(\alpha)}. \end{aligned}$$

Using (3.2) we can write

$$c = \sum_{j=0}^{m-1} b_j \sum_{l=0}^{m-1} a_l \sum_{i=0}^{m-1} p_i^{[k_l+k_j]} \alpha^{k_i}$$

$$\sum_{i=0}^{m-1} c_i \alpha^{k_i} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} b_j \sum_{l=0}^{m-1} a_l p_i^{[k_l+k_j]} \alpha^{k_i}.$$

Equating the coefficients of α on both sides of the above equation we obtain

$$c_i = \sum_{j=0}^{m-1} \left(\sum_{l=0}^{m-1} a_l p_i^{[k_l+k_j]} \right) b_j \qquad i = m-1,\ m-2,\ \cdots,\ 0 \tag{3.5}$$

which represents the system of m linear equations in b_0, b_1, $\cdots$, b_{m-1} of (3.4). Q.E.D.

From (3.4) we see that when the coordinates of a, $\mathbf{c}$ and the supporting elements are known, $b = c/a$ can be computed by solving the system of m linear equations in m unknowns over GF(q). For convenience of representation, denote (3.4) as $\mathbf{Ub} = \mathbf{c}$ where $\mathbf{U} \triangleq [u_{i,j}]_{i,j=0}^{m-1} = \left[\mathbf{a} \cdot \mathbf{p}_{m-1-i}^{[k_{m-1-j}]}\right]_{i,j=0}^{m-1}$, $\mathbf{b} = [b_{m-1-i}]_{i=0}^{m-1}$ and $\mathbf{c} = [c_{m-1-i}]_{i=0}^{m-1}$. The associated $m \times m$ matrix in (3.4) is referred to as the *coefficient matrix*. We now summarize the steps involved in the division algorithm as follows.

Division Algorithm 1:

Step 1) Form the coefficient matrix of (3.4).

Step 2) Solve Equation (3.4) for b.

Each element of the coefficient matrix requires m modulo-q multiplications and $m-1$ modulo-q additions resulting in $O(m^3)$ operation for the formation of the coefficient matrix. The essence of the second step of the above algorithm is the inversion of the coefficient matrix over GF(q). The computational complexity involved with the inversion of the $m \times m$ matrix of general form is $O(m^3)$. In the next section, we derive another division algorithm where the associated coefficient matrix is transformed to a Toeplitz matrix. The latter can be inverted by efficient algorithms, e.g., [39] and [40]. Below is an example using the above algorithm.

Example 3.1 Let the irreducible polynomial chosen for the field GF(2^3) be $g(z) = 1 + z^2 + z^3$. In this example we divide α^4 by α^2 over GF(2^3). The solution would be trivial if both the divisor and the dividend are given as powers of α in which case the division can be performed by simply subtracting the power of the divisor from that of the dividend. Unfortunately field elements are usually represented as polynomials of the powers of α using suitable bases. Here we consider two bases, namely the canonical basis $\{1, \alpha, \alpha^2\}$ and the normal basis $\{\alpha, \alpha^2, \alpha^4\}$. For the canonical basis representation

$$\alpha^4 = \sum_{i=0}^{2} c_i \alpha^i = 1 + \alpha + \alpha^2 \tag{3.6}$$

$$\alpha^2 = \sum_{i=0}^{2} a_i \alpha^i = \alpha^2 \tag{3.7}$$

and for the normal basis representation

$$\alpha^4 = \sum_{i=0}^{2} c_i \alpha^i = \alpha^4 \tag{3.8}$$

$$\alpha^2 = \sum_{i=0}^{2} a_i \alpha^i = \alpha^2 \tag{3.9}$$

We now follow Division Algorithm 1 step by step to compute the division.

Case I- Canonical basis representation.

Step 1) Here

$$H = \{1,\ \alpha,\ \alpha^2,\ \alpha^3,\ \alpha^4\}$$

and the coordinates of the supporting elements are obtained from the following.

$$\begin{aligned}
\alpha^0 &= p_0^{[0]} + p_1^{[0]}\alpha + p_2^{[0]}\alpha^2, \\
\alpha^1 &= p_0^{[1]} + p_1^{[1]}\alpha + p_2^{[1]}\alpha^2, \\
\alpha^2 &= p_0^{[2]} + p_1^{[2]}\alpha + p_2^{[2]}\alpha^2, \\
\alpha^3 &= 1 + \alpha^2 = p_0^{[3]} + p_1^{[3]}\alpha + p_2^{[3]}\alpha^2, \\
\alpha^4 &= 1 + \alpha + \alpha^2 = p_0^{[4]} + p_1^{[4]}\alpha + p_2^{[4]}\alpha^2.
\end{aligned}$$

Thus,

$$p_1^{[0]} = p_2^{[0]} = p_0^{[1]} = p_2^{[1]} = p_0^{[2]} = p_1^{[2]} = p_1^{[3]} = 0$$

and

$$p_0^{[0]} = p_1^{[1]} = p_2^{[2]} = p_0^{[3]} = p_2^{[3]} = p_0^{[4]} = p_1^{[4]} = p_2^{[4]} = 1.$$

For the canonical basis representation $k_i = i$. So with $m = 3$, U is

$$\mathrm{U} = \begin{bmatrix} \mathrm{a}\cdot\mathrm{p}_2^{[2]} & \mathrm{a}\cdot\mathrm{p}_2^{[1]} & \mathrm{a}\cdot\mathrm{p}_2^{[0]} \\ \mathrm{a}\cdot\mathrm{p}_1^{[2]} & \mathrm{a}\cdot\mathrm{p}_1^{[1]} & \mathrm{a}\cdot\mathrm{p}_1^{[0]} \\ \mathrm{a}\cdot\mathrm{p}_0^{[2]} & \mathrm{a}\cdot\mathrm{p}_0^{[1]} & \mathrm{a}\cdot\mathrm{p}_0^{[0]} \end{bmatrix}.$$

Substituting the values of the coordinates of the supporting elements we obtain

$$\mathbf{U} = \begin{bmatrix} a_0 + a_1 + a_2 & a_1 + a_2 & a_2 \\ a_2 & a_0 & a_1 \\ a_1 + a_2 & a_2 & a_0 \end{bmatrix}.$$

Step 2) Using the coordinates of the elements c and a from (3.6) and (3.7), we have

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Solving the system of three linear equations in three unknowns we obtain $b_0 = 0$, $b_1 = 0$ and $b_2 = 1$; so $b = \alpha^2$.

Case II- Normal basis representation.

Step 1) In this case

$$H = \{\alpha,\ \alpha^2,\ \alpha^3,\ \alpha^4,\ \alpha^5,\ \alpha^6\}$$

and

$$\begin{aligned}
\alpha^1 &= p_0^{[1]}\alpha + p_1^{[1]}\alpha^2 + p_2^{[1]}\alpha^4, \\
\alpha^2 &= p_0^{[2]}\alpha + p_1^{[2]}\alpha^2 + p_2^{[2]}\alpha^4, \\
\alpha^3 &= \alpha + \alpha^4 = p_0^{[3]}\alpha + p_1^{[3]}\alpha^2 + p_2^{[3]}\alpha^4, \\
\alpha^4 &= p_0^{[4]}\alpha + p_1^{[4]}\alpha^2 + p_2^{[4]}\alpha^4, \\
\alpha^5 &= \alpha^2 + \alpha^4 = p_0^{[5]}\alpha + p_1^{[5]}\alpha^2 + p_2^{[5]}\alpha^4, \\
\alpha^6 &= \alpha + \alpha^2 = p_0^{[6]}\alpha + p_1^{[6]}\alpha^2 + p_2^{[6]}\alpha^4,
\end{aligned}$$

which give

$$p_1^{[1]} = p_2^{[1]} = p_0^{[2]} = p_2^{[2]} = p_1^{[3]} = p_0^{[4]} = p_1^{[4]} = p_0^{[5]} = p_2^{[6]} = 0$$

and

$$p_0^{[1]} = p_1^{[2]} = p_0^{[3]} = p_2^{[3]} = p_2^{[4]} = p_1^{[5]} = p_2^{[5]} = p_0^{[6]} = p_1^{[6]} = 1.$$

For the normal basis representation in GF(2^3), $k_i = 2^i$; so we can write

$$\mathbf{U} = \begin{bmatrix} \mathbf{a} \cdot \mathbf{p}_2^{[2^2]} & \mathbf{a} \cdot \mathbf{p}_2^{[2^1]} & \mathbf{a} \cdot \mathbf{p}_2^{[2^0]} \\ \mathbf{a} \cdot \mathbf{p}_1^{[2^2]} & \mathbf{a} \cdot \mathbf{p}_1^{[2^1]} & \mathbf{a} \cdot \mathbf{p}_1^{[2^0]} \\ \mathbf{a} \cdot \mathbf{p}_0^{[2^2]} & \mathbf{a} \cdot \mathbf{p}_0^{[2^1]} & \mathbf{a} \cdot \mathbf{p}_0^{[2^0]} \end{bmatrix}.$$

For the finite field being considered here, $\alpha^8 = \alpha$. Using this relationship and substituting the values of the coordinates of the supporting elements we have

$$\mathbf{U} = \begin{bmatrix} a_0 & a_0 + a_1 & a_1 + a_2 \\ a_0 + a_1 & a_2 & a_0 + a_2 \\ a_1 + a_2 & a_0 + a_2 & a_1 \end{bmatrix}. \tag{3.10}$$

Step 2) Now using (3.4) and (3.10) and substituting the coordinates of the elements c and a from (3.8) and (3.9) we have the following system of linear equations

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

The solution of these equations gives $b_0 = 0$, $b_1 = 1$ and $b_2 = 0$ for the normal basis representation of b; consequently $b = \alpha^2$.

3.4 DTWHE and Division in Finite Fields

Definition 3.1 [39] The discrete time Wiener-Hopf equation (DTWHE) is defined as a system of m linear inhomogeneous equations with m unknowns x_i $(i = 0, 1, \cdots, m-1) \in \mathrm{GF}(q)$, $2m-1$ constant coefficients y_i $(i = 0, 1, \cdots, 2m-2) \in \mathrm{GF}(q)$ that are not all zero, and m constants z_i $(i = 0, 1, \cdots, m-1) \in \mathrm{GF}(q)$ such that

$$\begin{bmatrix} y_{m-1} & y_{m-2} & \cdots & y_0 \\ y_m & y_{m-1} & \cdots & y_1 \\ \vdots & \vdots & \ddots & \vdots \\ y_{2m-2} & y_{2m-3} & \cdots & y_{m-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{m-1} \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{m-1} \end{bmatrix}. \tag{3.11}$$

Equation (3.11) is referred to as the DTWHE of degree m over $\mathrm{GF}(q)$.

In our forthcoming analyses, the elements of $\mathrm{GF}(q^m)$ are represented with respect to the canonical basis $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$. In this section we show that if the elements of $\mathrm{GF}(q^m)$ are represented with respect to the canonical basis, then division over $\mathrm{GF}(q^m)$ can be performed by solving a DTWHE of degree m over $\mathrm{GF}(q)$. The motivation behind obtaining a system of linear equations in the form of a DTWHE is the lower computational complexity involved in solving a DTWHE [39].

Lemma 3.1 For the canonical basis representation of the elements of $\mathrm{GF}(q^m)$, i.e., $\alpha^k = \sum_{i=0}^{m-1} p_i^{[k]} \alpha^i$,

$$p_j^{[k+1]} = \begin{cases} -p_{m-1}^{[k]} g_j & (\mathrm{mod}\ q) \quad j = 0 \\ p_{j-1}^{[k]} - p_{m-1}^{[k]} g_j & (\mathrm{mod}\ q) \quad 1 \le j \le m-1 \end{cases} \tag{3.12}$$

where $g(z) = \sum_{i=0}^{m-1} g_i z^i + z^m$ is the irreducible monic polynomial over $\mathrm{GF}(q)$.

Proof:

$$\alpha^{k+1} = \sum_{i=0}^{m-1} p_i^{[k]} \alpha^{i+1}.$$

Substituting $j = i + 1$,

$$\alpha^{k+1} = \sum_{j=1}^{m} p_{j-1}^{[k]} \alpha^j = \sum_{j=1}^{m-1} p_{j-1}^{[k]} \alpha^j + p_{m-1}^{[k]} \alpha^m.$$

Since $g(\alpha) = 0$, $\alpha^m = -\sum_{j=0}^{m-1} g_j \alpha^j$, thus we have

$$\sum_{j=0}^{m-1} p_j^{[k+1]} \alpha^j = \sum_{j=1}^{m-1} p_{j-1}^{[k]} \alpha^j - p_{m-1}^{[k]} \sum_{j=0}^{m-1} g_j \alpha^j.$$

The coefficients of α^j $(0 \le j \le m-1)$ on both sides yield the proof.

Using (3.3), we can also write (3.12) in vector notation as follows.

$$\mathrm{p}_j^{[k+1]} = \begin{cases} -\mathrm{p}_{m-1}^{[k]} g_j & (\mathrm{mod}\ q) \quad j = 0 \\ \mathrm{p}_{j-1}^{[k]} - \mathrm{p}_{m-1}^{[k]} g_j & (\mathrm{mod}\ q) \quad 1 \le j \le m-1 \end{cases} \tag{3.13}$$

For the canonical basis representation of the elements of GF(q^m), Equation (3.4) can be written as

$$\begin{bmatrix} \mathrm{a} \cdot \mathrm{p}_{m-1}^{[m-1]} & \mathrm{a} \cdot \mathrm{p}_{m-1}^{[m-2]} & \cdots & \mathrm{a} \cdot \mathrm{p}_{m-1}^{[0]} \\ \mathrm{a} \cdot \mathrm{p}_{m-2}^{[m-1]} & \mathrm{a} \cdot \mathrm{p}_{m-2}^{[m-2]} & \cdots & \mathrm{a} \cdot \mathrm{p}_{m-2}^{[0]} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{a} \cdot \mathrm{p}_0^{[m-1]} & \mathrm{a} \cdot \mathrm{p}_0^{[m-2]} & \cdots & \mathrm{a} \cdot \mathrm{p}_0^{[0]} \end{bmatrix} \begin{bmatrix} b_{m-1} \\ b_{m-2} \\ \vdots \\ b_0 \end{bmatrix} = \begin{bmatrix} c_{m-1} \\ c_{m-2} \\ \vdots \\ c_0 \end{bmatrix} \tag{3.14}$$

over GF(q) with $\mathrm{U} \triangleq [u_{i,j}]_{i,j=0}^{m-1} = \left[\mathrm{a} \cdot \mathrm{p}_{m-1-i}^{[m-1-j]}\right]_{i,j=0}^{m-1}$. Let r_i denote the ith row of the matrix U. We now present the following theorem.

Theorem 3.2 Let r'_i denote the ith row of the new matrix, say U', obtained after the elementary row operations

$$\mathrm{r}'_i \;=\; \mathrm{r}_i - \sum_{k=1}^{i} \mathrm{r}'_{i-k} g_{m-k} \pmod q \quad (i = 1,\ 2,\ \cdots,\ m-1). \tag{3.15}$$

The above row operations transform (3.14) to the DTWHE

$$\begin{bmatrix} \tilde{a}_{m-1} & \tilde{a}_{m-2} & \cdots & \tilde{a}_0 \\ \tilde{a}_m & \tilde{a}_{m-1} & \cdots & \tilde{a}_1 \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{a}_{2m-2} & \tilde{a}_{2m-3} & \cdots & \tilde{a}_{m-1} \end{bmatrix} \begin{bmatrix} b_{m-1} \\ b_{m-2} \\ \vdots \\ b_0 \end{bmatrix} = \begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \vdots \\ \tilde{c}_{m-1} \end{bmatrix} \tag{3.16}$$

over GF(q) where

$$\tilde{a}_k \;=\; \mathrm{a} \cdot \mathrm{p}_{m-1}^{[k]} \pmod q \quad (k = 0,\ 1,\ \cdots,\ 2m-2) \tag{3.17}$$

and

$$\tilde{c}_i \;=\; \begin{cases} c_{m-1} & \text{if } i = 0 \\ c_{m-1-i} - \sum_{l=1}^{i} \tilde{c}_{i-l} g_{m-l} \pmod q & \text{if } i = 1,\ 2,\ \cdots,\ m-1 \end{cases}. \tag{3.18}$$

A proof of the theorem appears in Appendix A. Below is an example which demonstrates the elementary row operations of (3.15) giving a DTWHE.

Example 3.2 Let the irreducible polynomial chosen for the field GF(3^2) be $g(z) = 2 + z + z^2$. As in Example 3.1, the first step of Division Algorithm 1 results in

$$\begin{bmatrix} a_0 + 2a_1 & a_1 \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_0 \end{bmatrix}.$$

Then the elementary row operation (3.15) gives

$$\begin{bmatrix} a_0 + 2a_1 & a_1 \\ -a_0 - a_1 & a_0 + 2a_1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_0 - c_1 \end{bmatrix}$$

which is a DTWHE over GF(3) of degree 2.

Theorem 3.2 establishes a relationship between the DTWHE and division in finite fields. If the elements of GF(q^m) are represented with respect to the canonical basis, then a division over GF(q^m) can be performed by simply solving the DTWHE (3.16) over GF(q). The division algorithm is summarized below.

Division Algorithm 2:

Step 1) Construct the DTWHE (3.16).

Step 2) Solve the DTWHE.

Since there are only $2m-1$ elements in the associated coefficient matrix, the computational complexity of **Step 1** of the above algorithm is $O(m^2)$. As with Algorithm 1 in Section 3.3, the essence of **Step 2** is the inversion of an $m \times m$ coefficient matrix. However, in Algorithm 2, the matrix is a Toeplitz matrix and the computational complexity for its inversion is only $O(m \log^2 m)$ [39].

Algorithm 2 is similar to the approach of [31] in the sense that when the field elements are represented with respect to a canonical basis, both compute the division by solving DTWHEs of degree m. The advantage of Algorithm 2 is that it requires, for the construction of the DTWHE, the determination of the coordinates of only $2m-1$ supporting elements, whereas the approach of [31] requires the determination of $\mathrm{Tr}(\beta\alpha^i)$ $(i = 0,\ 1,\ \cdots,\ 3m-3)$, where $\beta \in \mathrm{GF}(q^m)$.

3.5 Conclusions

When the elements of the finite field GF(q^m) are represented by powers of α, the division of one element by another can be performed simply by subtracting the power of the divisor from that of the dividend. Conventionally, however, the elements are usually represented as a polynomial using a suitable basis. Division Algorithm 1 is general in the sense that it can be applied using any basis chosen for the field; it requires the solution of a system of m linear equations of the

general form over GF(q) to perform a division in GF(q^m). It has been shown that if the field elements are represented with respect to a canonical basis of the form $\{1, \alpha, \cdots, \alpha^{m-1}\}$, then a division can be performed with a lesser order of computational complexity by solving a discrete time Wiener-Hopf equation of degree m.

Chapter 4

Bit-Serial Systolic Divider for Finite Fields GF(2^m)

4.1 Introduction

Division Algorithm 1, presented in the previous chapter, consists of the formation of the coefficient matrix (CM) and the solution of a system of equations. The task of forming the CM and then solving the resulting system of m equations in m unknowns becomes more and more tedious as the value of m increases. Moreover, for a parallel-type divider, which is realized by combinational logic circuits, the final logic functions for the coordinates of the quotient b become quite lengthy. As a result, they cannot be easily implemented using combinational logic for large values of m. The realization of these types of dividers remains practical only for small values of m ($m \leq 5$). For typical cryptographic applications where the value of m is large, parallel processing using VLSI is an attractive approach.

In this chapter, an algorithm is presented for the formation of the CM. The algorithm is mapped onto a one dimensional systolic array. Then a two dimensional systolic array for the solution of the system of equations is developed to obtain a bit-serial systolic divider for GF(2^m) [11].

4.2 Formation of the Coefficient Matrix

Let $g(z)$ be an irreducible monic polynomial over GF(2) of degree m i.e.,

$$g(z) = \sum_{i=0}^{m-1} g_i z^i + z^m,$$

where $g_i \in$GF(2) for $i = 0, 1, \cdots, m-1$ and m is a nonzero positive integer. The polynomial $g(z)$ has a root $\alpha \in$ GF(2^m) and all elements of GF(2^m) can be represented with respect to the canonical basis $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$ [25]. Specifically if $a \in$GF(2^m), then there exist $a_i \in$ GF(2), $0 \le i \le m-1$, such that

$$a = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1}$$

where the terms a_i are the coordinates of a relative to the canonical basis.

The supporting elements are $\alpha^0, \alpha^1, \alpha^2, \cdots, \alpha^{2m-2}$ and $p_j^{[k]}$ is the j-th coordinate of the supporting element α^k, i.e.,

$$\alpha^k = \sum_{i=0}^{m-1} p_i^{[k]}\alpha^i, \quad 0 \le k \le 2m-2. \tag{4.1}$$

Let a, b and c be three elements in GF(2^m) such that $b = c/a$ and $a \neq 0$. Then for the division $b = c/a$ over GF(2^m), Equation (3.4) can be rewritten by rotating the coefficient matrix horizontally as shown below.

$$\begin{bmatrix} \sum_{l=0}^{m-1} a_l p_{m-1}^{[l]} & \sum_{l=0}^{m-1} a_l p_{m-1}^{[l+1]} & \cdots & \sum_{l=0}^{m-1} a_l p_{m-1}^{[l+m-1]} \\ \sum_{l=0}^{m-1} a_l p_{m-2}^{[l]} & \sum_{l=0}^{m-1} a_l p_{m-2}^{[l+1]} & \cdots & \sum_{l=0}^{m-1} a_l p_{m-2}^{[l+m-1]} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{l=0}^{m-1} a_l p_0^{[l]} & \sum_{l=0}^{m-1} a_l p_0^{[l+1]} & \cdots & \sum_{l=0}^{m-1} a_l p_0^{[l+m-1]} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{bmatrix} = \begin{bmatrix} c_{m-1} \\ c_{m-2} \\ \vdots \\ c_0 \end{bmatrix}, \tag{4.2}$$

which can be abbreviated as $\mathbf{Ab} = \mathbf{c}$ with

$$\mathbf{A} \triangleq [a_{i,j}]_{i,j=0}^{m-1} = \left[\sum_{l=0}^{m-1} a_l p_{m-1-i}^{[l+j]}\right]_{i,j=0}^{m-1}, \tag{4.3}$$

$$\mathbf{b} = [b_i]_{i=0}^{m-1} \quad .$$

and

$$\mathbf{c} = [c_{m-1-i}]_{i=0}^{m-1} \, .$$

The first step of the DA (i.e. Division Algorithm 1) is to form the coefficient matrix $\mathbf{A}$ from the coordinates of the divisor a and the supporting elements α^0, α^1, $\cdots$, α^{2m-2}. This requires m^3 arithmetic operations over GF(2). Using some memory elements can reduce computational load to $O(m^2)$.

From the definition of the supporting elements, we obtain

$$p_i^{[k]} = \delta_{i,k} \quad 0 \le i \le m-1, \;\; 0 \le k \le m-1, \tag{4.4}$$

where the Kronecker delta function $\delta_{i,k}$ is equal to 1 when $i = k$ and equal to 0 otherwise.

For GF(2^m), Equation (3.12) becomes

$$p_i^{[k]} = p_{m-1}^{[k-1]} g_i + (1 - \delta_{i,0})\, p_{i-1}^{[k-1]} \quad 0 \le i \le m-1, \;\; 0 \le k \le 2m-2. \tag{4.5}$$

Using (4.4), the elements of the 0th column of $\mathbf{A}$ satisfy

$$a_{i,0} = a_{m-1-i} \quad 0 \le i \le m-1. \tag{4.6}$$

For $1 \le j \le m-1$, substituting (4.5) in (4.3) yields

$$\begin{aligned} a_{i,j} &= \sum_{l=0}^{m-1} a_l p_{m-1-i}^{[l+j]} \\ &= \sum_{l=0}^{m-1} a_l \left\{ p_{m-1}^{[l+j-1]} g_{m-1-i} + (1 - \delta_{m-1-i,0})\, p_{m-1-(i+1)}^{[l+j-1]} \right\} \end{aligned}$$

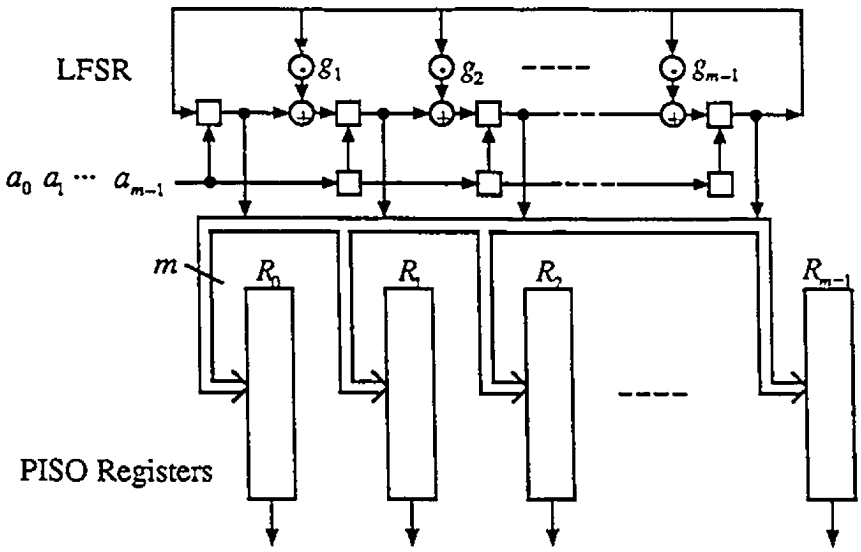


Figure 4.1: LFSR based structure for the formation of the CM.

$$= g_{m-1-i}a_{0,j-1} + (1 - \delta_{m-1-i,0})\, a_{i+1,j-1}$$

$$= \begin{cases} g_{m-1-i}a_{0,j-1} + a_{i+1,j-1} & i = 0,\ 1,\ \cdots,\ m-2 \\ a_{0,j-1} & i = m-1 \end{cases} \tag{4.7}$$

Equation (4.7) gives a recursive expression for the jth ($j = 1,\ 2,\ \cdots,\ m-1$) column of **A** in terms of its $(j-1)$th column. Using m memory elements, the elements of **A** can be obtained with $(m-1)^2$ multiplications and additions over GF(2). The 0th column is obtained directly from the coordinates of a as indicated in (4.6).

According to (4.7), the CM can be constructed using a linear feedback shift register (LFSR) which generates successive column elements in one time step. As described later in this section, the structure for Step 2 of the DA accepts one column element at every time step. As a result, the outputs of the LFSR must be stored in m PISO (parallel in serial out) registers, viz., R_0, $R_1 \cdots R_{m-1}$ as shown in Figure 4.1.

The above approach to the formation of **A** is simple; but it requires global data communications. The output of the LFSR is connected to the PISO registers by an

m bit data bus. In addition, the LFSR itself contains a feedback connection from one end to the other end. For large values of m (for example, consider the case of a cryptographic system with $m = 900$), these global data communications cause considerable difficulties in VLSI design [22], [23]. The design would be considerably simplified if the global data communications can be replaced by local, regular data communications. We now describe a structure for the formation of **A** which does not require global data communications.

Systolic array for the CM: A one dimensional systolic array for the formation of the CM (SAFCM) is shown in Figure 4.2(a). The array consists of $m-1$ basic rectangular processors marked from left to right as Q_1, Q_2, $\cdots$, Q_{m-1}. The coordinates of a are fed into Q_1 in a bit-serial fashion with a_{m-1} first. The output format of the array is shown in Figure 4.2(b). There is a delay of two time steps between any two adjacent columns of **A**.

The 0-th column of **A** is obtained directly as the coordinates of a enter Q_1. Columns 1 to $m-1$ are generated by Q_1, Q_2, $\cdots$, Q_{m-1} respectively. The output of processor Q_{j-1} $(1 \leq j \leq m-1)$ is fed into processor Q_j. The first output $a_{0,j-1}$ of Q_{j-1} is stored in the internal register r of Q_j. The coefficients g_0, g_1, $\cdots$, g_{m-1} of the irreducible polynomial $g(z)$ propagate through the array so that $a_{i,j} = a_{0,j-1}g_{m-1-i} + (1 - \delta_{m-1-i,0})\, a_{i+1,j-1}$ is formed in processor Q_j at time $i + 2j$. A control signal q is used to identify the beginning of the divisor. The same signal is also used by the processors to mark the point in time at which the internal register r updated. Figure 4.3(a) shows the operation of the processor, where the right hand side variables of the assignment statements are of time step t, and the left hand side variables are of time step $t + 1$. The corresponding circuit is given in Figure 4.3(b) where FF, AND, XOR and MUX denote flip flop, AND gate, XOR gate and multiplexer respectively.

The structure for the formation of the coefficient matrix as described above is a little more complex than the LFSR based structure. However, the systolic array

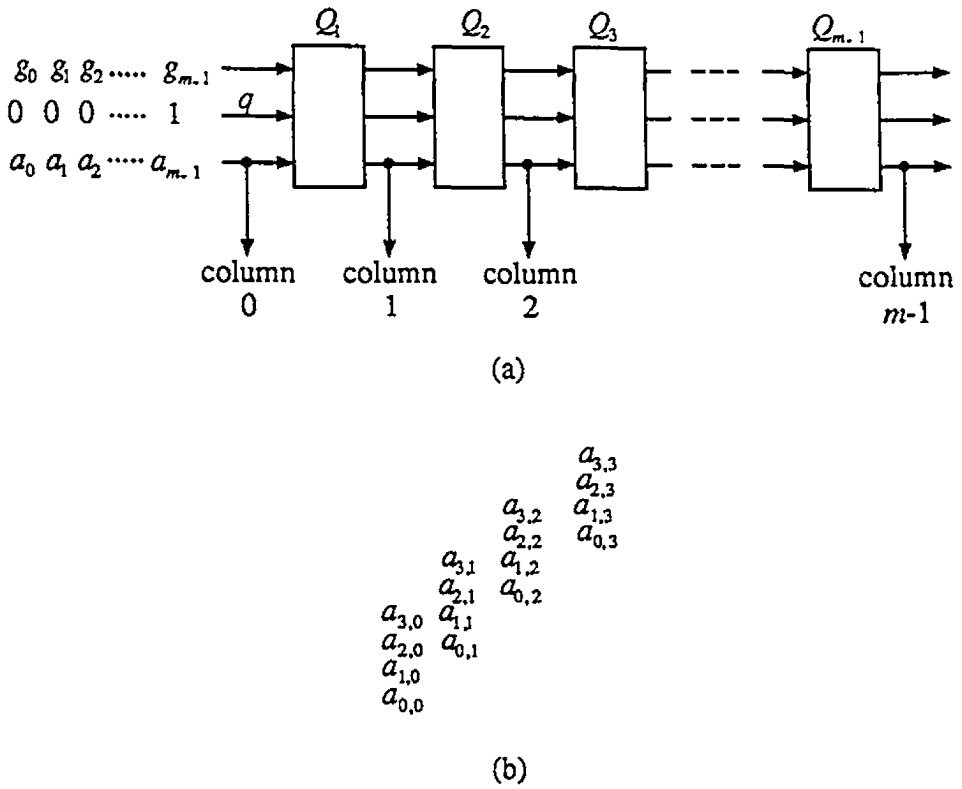
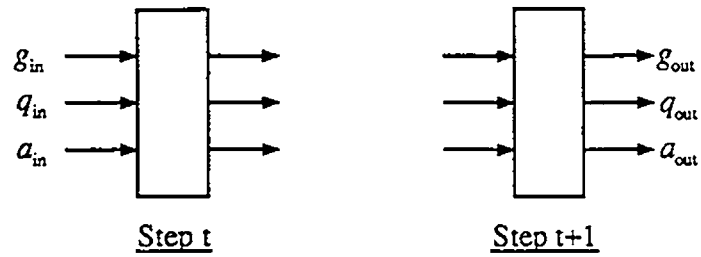
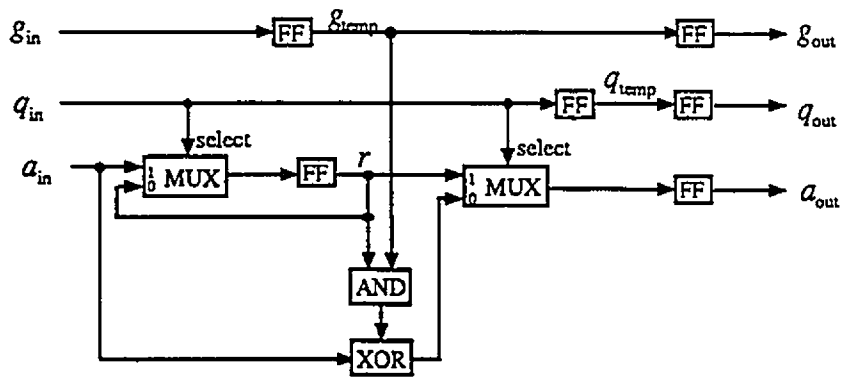


Figure 4.2: (a) Systolic array for the formation of the CM (SAFCM) and (b) Output format of the array ($m = 4$).



if $q_{in} = 1$ **then**
begin
$a_{out} := r$;
$r := a_{in}$;
end
else
begin
$a_{out} := g_{temp} r + a_{in}$;
end
$g_{out} := g_{temp}$; $g_{temp} := g_{in}$;
$q_{out} := q_{temp}$; $q_{temp} := q_{in}$;

(a)



(b)

Figure 4.3: Rectangular processor of the SAFCM: (a) operation and (b) circuit diagram.

based structure requires no global data communication or broadcast signal.

4.3 Solving the System of Equations

The following theorem is useful for obtaining a simplified structure for the second step of the DA.

Theorem 4.1 If a is a nonzero element of the field GF(2^m), then the determinant of the resulting CM is 1.

Proof: Since every nonzero element of the field GF(2^m) has an inverse, a solution of (4.2) exists and the CM is non-singular. Moreover, the elements of the CM are elements of GF(2), so the corresponding determinant is one. Q.E.D.

Pre-multiply the matrix **A** with the following elementary $m \times m$ matrix $\mathbf{P}_{i,j}$ $(i \neq j)$ over GF(2). The elements of $\mathbf{P}_{i,j}$ at (i,j), (i,i), (j,i) and (j,j) are h, $\overline{f}$, f and $\overline{f}$ respectively where $h = a_{i,j}$, $f = \overline{a}_{j,j}a_{i,j}$ and $\overline{f}$ denotes the complement of f. The remaining elements of $\mathbf{P}_{i,j}$ are 0 except for the other elements on the principal diagonal which are all 1. The structure of $\mathbf{P}_{i,j}$ with $i > j$ is shown below.

$$\mathbf{P}_{i,j} = \begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & 0 & \\ & & 1 & & & & & & \\ & & & \overline{f} & & & f & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & h & & & \overline{f} & & \\ & & & & & & & 1 & \\ & 0 & & & & & & \ddots & \\ & & & & & & & & 1 \end{bmatrix}_{m \times m} \tag{4.8}$$

The matrix (i.e., **A**) which $\mathbf{P}_{i,j}$ pre-multiplies is referred to as the operand matrix. Note that the elementary matrix $\mathbf{P}_{i,j}$ is uniquely determined by the two elements

h	f	Changes
0	0	no change
1	0	row i := row i + row j
1	1	rows i and j are interchanged

Table 4.1: Changes in the operand matrix after pre-multiplication.

at positions (i,j) and (j,j) of the operand matrix. Also the pre-multiplication changes only rows i and j of the operand matrix. There are three possible changes and these are listed in Table 4.1.

From the definition of $\mathbf{P}_{i,j}$,

$$\det(\mathbf{P}_{i,j}) = \overline{f}\,\overline{f} + hf = 1 + f + hf = 1 + \overline{h}f = 1 + \overline{a}_{i,j}\,\overline{a}_{j,j}a_{i,j} = 1 \qquad (4.9)$$

where all additions and multiplications are over GF(2).

Theorem 4.2 If $\mathbf{A}' = \mathbf{P}_{i,j}\mathbf{A}$ where $i \neq j$, then (i) all rows of $\mathbf{A}'$ except the j-th and i-th are same as those of $\mathbf{A}$, (ii) $a'_{i,j} = 0$, and (iii) $a'_{j,j} = 1$ if $a_{i,j} = 1$.

A simple proof of the theorem is given in the appendix.

Gauss-Jordan Elimination over GF(2): Define $\mathbf{U}_i = \mathbf{P}_{m-1,i}\mathbf{P}_{m-2,i}\cdots\mathbf{P}_{i+1,i}$ and $\mathbf{A}^{(0)} = \mathbf{U}_0\mathbf{A}$. Then according to Theorem 4.2, all elements below $a^{(0)}_{0,0}$ of the 0-th column of $\mathbf{A}^{(0)}$ are zero. Performing this annihilation process on columns 0 through $m-2$, we obtain

$$[\mathbf{T}, \mathbf{c}'] = \mathbf{U}_{m-2}\mathbf{U}_{m-3}\cdots\mathbf{U}_0\,[\mathbf{A}, \mathbf{c}]\,, \qquad (4.10)$$

where the $m \times m$ matrix $\mathbf{T}$ over GF(2) is upper triangular. This corresponds to Gaussian elimination with partial pivoting over GF(2). According to Theorem 4.1 and Equation (4.9), $\mathbf{T}$ is non-singular and its principal diagonal elements are all 1.

Similarly, defining $\mathbf{L}_i = \mathbf{P}_{i,m-1}\mathbf{P}_{i,m-2}\cdots\mathbf{P}_{i,i+1}$,

$$[\mathbf{I}, \mathbf{c}''] = \mathbf{L}_{m-2}\mathbf{L}_{m-3}\cdots\mathbf{L}_0\,[\mathbf{T}, \mathbf{c}'] \qquad (4.11)$$

where **I** is the $m \times m$ identity matrix. Then the column vector $\mathbf{c}''$ is the required solution i.e., $\mathbf{b} = \mathbf{c}''$.

Matrix **A** in (4.10) is upper-triangularized by annihilating the matrix elements column wise and **T** in (4.11) is diagonalized by annihilating the matrix elements row wise.

Example 4.1 Let $\alpha \in \mathrm{GF}(2^3)$ satisfy $g(\alpha) = 1 + \alpha + \alpha^3 = 0$. Let $a = \alpha^3$ and $c = \alpha^4$. Then (4.2) becomes

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}. \tag{4.12}$$

The diagonalization of the CM using (4.10) and (4.11) is shown in Table 4.2. The four elements of $\mathbf{P}_{i,j}$ which depend on h and f are underlined. The element of the operand matrix which is annihilated at the subsequent step is marked with an asterisk. The operand matrix of a row in the table is obtained by pre-multiplying the preceding operand matrix with the elementary matrix of the same row.

From Table 4.2 we obtain $\mathbf{b} = \mathbf{c}'' = [0\ 1\ 0]^t$ which is the binary representation of α, the required quotient.

On a sequential computer, the above Gauss-Jordan elimination (GJE) could be a time consuming operation, particularly for large values of m. But, systolic arrays for solving systems of linear equations (SASLE) by the GJE with partial pivoting are effective [16]. The array presented here is similar to the array of [16]; however, the processors of the array presented here perform relatively simpler operations and require fewer input and output connections. This simplification results from the fact that the divisor is a nonzero field element and the resulting CM is non-singular.

Systolic Array for Solving Linear Equations: The GJE algorithm is mapped onto the array as shown in Figure 4.4. The array has m rows. Row

h, f $\mathbf{P}_{i,j}$	Operand matrices
	$[\mathbf{A}, \mathbf{c}] = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1^* & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$
$h = 1,\ f = 1$ $\mathbf{P}_{1,0} = \begin{bmatrix} \underline{0} & \underline{1} & 0 \\ \underline{1} & \underline{0} & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1^* & 0 & 1 & 0 \end{bmatrix}$
$h = 1,\ f = 0$ $\mathbf{P}_{2,0} = \begin{bmatrix} \underline{1} & 0 & \underline{0} \\ 0 & 1 & 0 \\ \underline{1} & 0 & \underline{1} \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1^* & 0 & 1 \end{bmatrix}$
$h = 1,\ f = 0$ $\mathbf{P}_{2,1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \underline{1} & \underline{0} \\ 0 & \underline{1} & \underline{1} \end{bmatrix}$	$[\mathbf{T}, \mathbf{c}'] = \begin{bmatrix} 1 & 1^* & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
$h = 1,\ f = 0$ $\mathbf{P}_{0,1} = \begin{bmatrix} \underline{1} & \underline{1} & 0 \\ \underline{0} & \underline{1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0^* & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
$h = 0,\ f = 0$ $\mathbf{P}_{0,2} = \begin{bmatrix} \underline{1} & 0 & \underline{0} \\ 0 & 1 & 0 \\ \underline{0} & 0 & \underline{1} \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1^* & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
$h = 1,\ f = 0$ $\mathbf{P}_{1,2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \underline{1} & \underline{1} \\ 0 & \underline{0} & \underline{1} \end{bmatrix}$	$[\mathbf{I}, \mathbf{c}''] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Table 4.2: Diagonalization of the CM using GJE with partial pivoting.

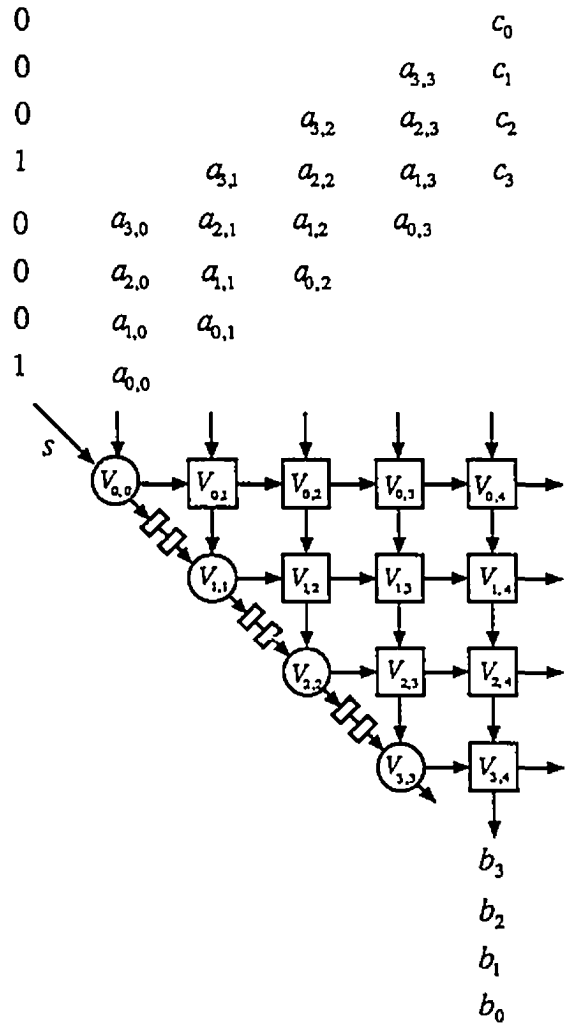


Figure 4.4: Systolic array for solving linear equations (SASLE).

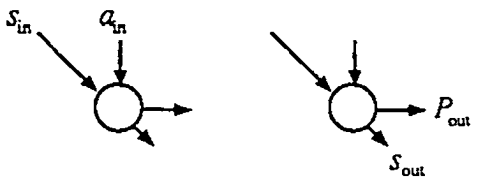
k has $m+1-k$ processors labeled $V_{k,k}$, $V_{k,k+1}$, $\cdots$, $V_{k,m}$ from left to right. The leftmost processor of each row is a circular processor. Between any two adjacent circular processors, there are two flip flops. All other processors are square processors. The start of the input of $[\mathbf{A}, \mathbf{c}]$ into the array is marked by the control signal s. The control signal is obtained by repeating the control signal for the SAFCM. The matrix [A, c] is fed into the array column by column. Column j of the matrix is input to processor $V_{0,j}$. The circular processor $V_{k,k}$ generates $\mathbf{P}_{i,k}$ $(i = k+1,\ k+2,\ \cdots, m-1,\ 0,\ 1,\ \cdots,\ k-1)$. The square processors of row i apply the operations corresponding to pre-multiplication by $\mathbf{P}_{i,k}$.

Before describing the operations of the processors, an informal description of how the array works is given. Row i of the matrix $[\mathbf{A}, \mathbf{c}]$ moves downward through rows 0, 1, $\cdots$, $i-1$ of the array and the matrix elements at positions $(i,0)$, $(i,1)$, $\cdots$, $(i,i-1)$ are annihilated. The other elements of row i reside in $V_{i,i}$, $V_{i,i+1}$, $\cdots$, $V_{i,m}$ for the next $m-1$ time steps and annihilate all elements which enter $V_{i,i}$. The annihilated elements are of positions $(i+1,i)$, $(i+2,i)$, $\cdots$, $(m-1,i)$, $(0,i)$, $\cdots$, $(i-1,i)$ respectively. Next, row i again starts moving downward. Now the elements at positions $(i,i+1)$, $(i,i+2)$, $\cdots$, $(i,m-1)$ are made zero in $V_{i+1,i+1}$, $V_{i+2,i+2}$, $\cdots$, $V_{m-1,m-1}$ respectively; and the element at (i,m) is the i th component of the vector b.

The annihilation processes above and below the principal diagonal are pipelined. The coordinates of b start emerging from the processor $V_{m-1,m}$ at time step $3m$. Subsequent coordinates emerge at a rate of one coordinate per time step, resulting in a computational time of $4m-1$ time steps.

We now briefly describe the operation of the two types of processors used in the array. Detailed operation and the corresponding circuit diagrams are given in Figures 4.5 and 4.6.

- Circular processors generate the elementary matrices and send them rightward to the neighboring square processors. Since an elementary matrix can

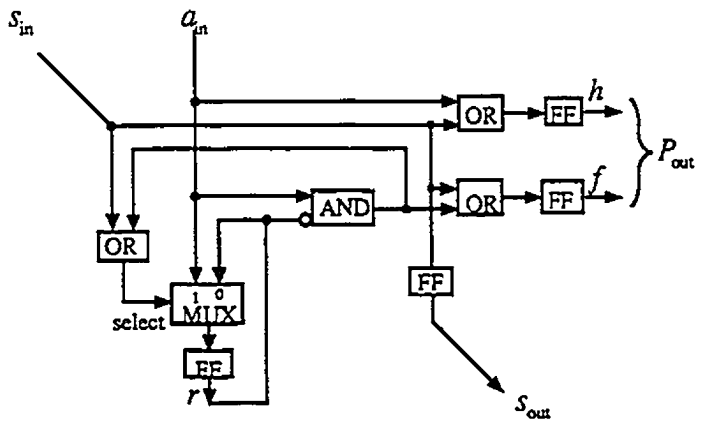


if $s_{in} = 1$ **then**
begin

$P_{out} := (1, 1)$; {i.e., $h = 1$; $f := 1$}
$r := a_{in}$;
end
else
case (r, a_{in}) **of**
$(0,0)$: $P_{out} := (0, 0)$;
$(0,1)$: $P_{out} := (1, 1)$;
$(1,0)$: $P_{out} := (0, 0)$;
$(1,1)$: $P_{out} := (1, 0)$;

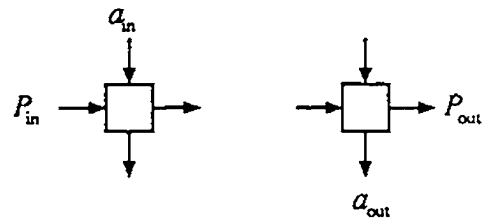
end
$s_{out} := s_{in}$;

(a)



(b)

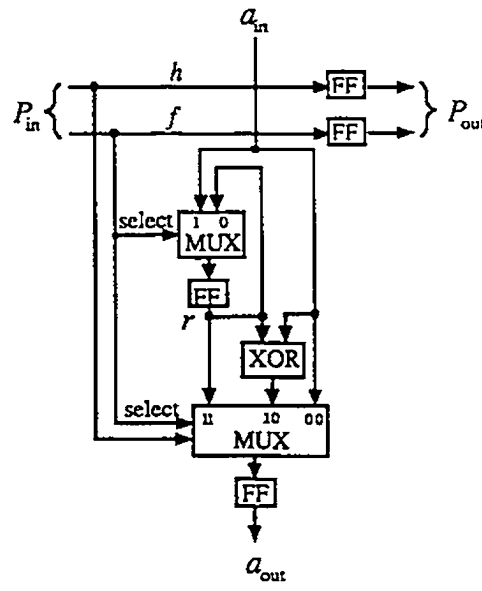
Figure 4.5: Circular processor of the SASLE: (a) operation and (b) circuit diagram.



Step t Step t+1

case (P_{in}) of
$(0,0)$: $a_{out} := a_{in}$;
$(1,0)$: $a_{out} := r + a_{in}$;
$(1,1)$: $a_{out} := r$; $r := a_{in}$;
$P_{out} := P_{in}$;

(a)



(b)

Figure 4.6: Square processor of the SASLE: (a) operation and (b) circuit diagram.

be specified by the values of h and f only, P_{out} in Figure 4.55(a) can be realized by two lines. The circular processor updates its internal register r if s_{in}=1 or if $r = 0$ and a_{in}=1.

- Depending upon P_{in}, the square processor performs one of the three operations listed in Table 4.1. It also sends the elementary matrices rightward to the neighboring square processor.

Compared to the square processor of [16], the square processor presented here performs relatively simpler operations (refer to Figure 6(a)) and requires two less input-output connections per processor. As the number of square processors increases with m^2, the use of these simple processors reduces the circuit complexity of the array in terms of number of gates and flip-flops.

4.4 Divider Structure

A complete bit-serial systolic divider over GF(2^m) is shown in Figure 4.7 where the SAFCM and SASLE are connected through some delay elements. The element D^i introduces a delay of i time steps. These delay elements are used to match the output and input of the SAFCM and SASLE respectively. The computational time to perform a division is $5m-1$ time steps and pipelined operation is possible.

The time step duration is essentially determined by the processor (rectangular, circular or square) which causes maximum time delay. This time delay is, however, independent of m. As a result, the time step duration does not increase with the increase in the values of m.

For the division operation to be valid over GF(2^m) requires that at least one of the coordinates of the divisor be nonzero. However, if it happens that all the input coordinates to the SAFCM are zero (intentionally or by some external noise), then the elements of **A**, which are the output of the SAFCM, are also zero. This particular case can be identified by providing a zero-detection circuit at the input

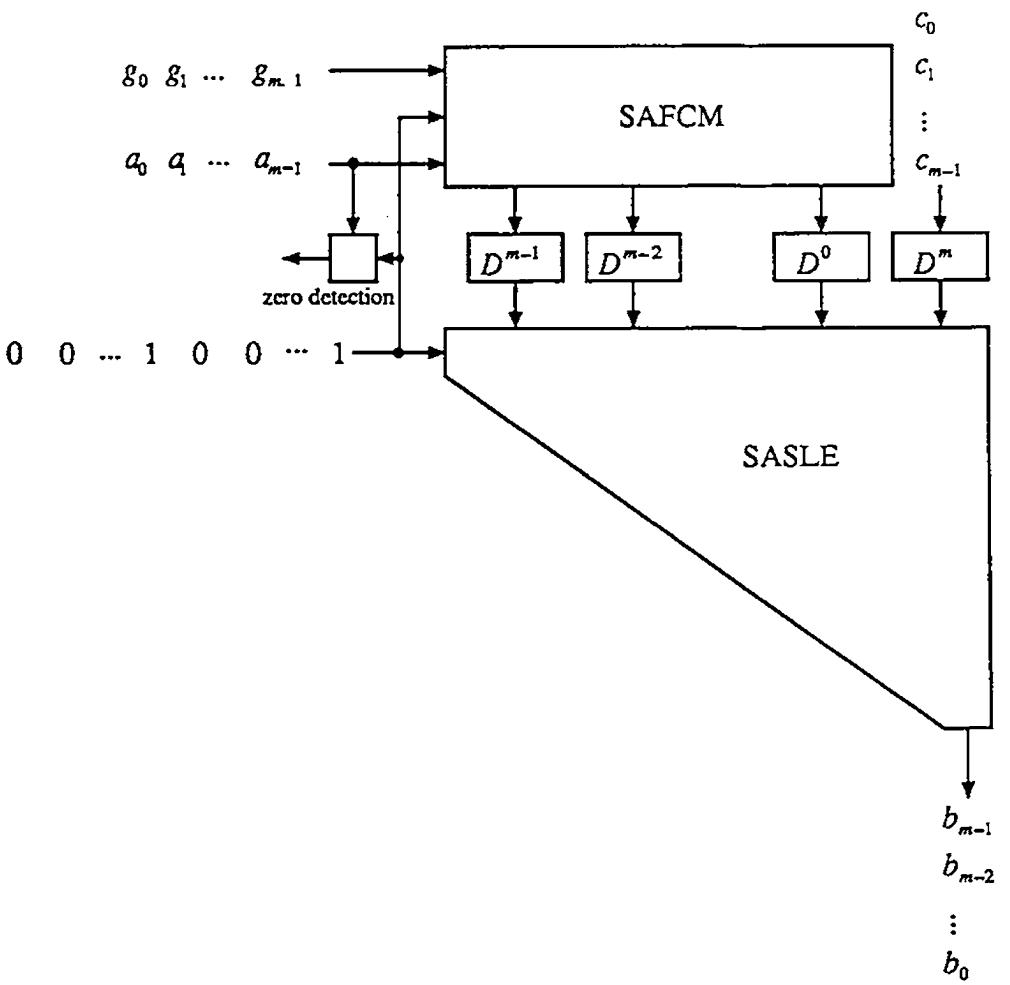


Figure 4.7: Structure for a bit-serial systolic divider.

as shown in Figure 4.7. Such a circuit may simply consist of an OR gate and a flip flop which is initialized by the control signal s.

The SAFCM and SASLE may be combined to obtain a single array. However, the main advantage of having two separate arrays is that the SAFCM can be used to obtain a bit-serial systolic multiplier for GF(2^m) [11]. In addition, the SASLE can be used to solve linear equations over GF(2).

4.5 Comparison

For comparison, we consider the modular inverters of [1] and [43]. Our bases for comparison are circuit and time complexities. So far we have measured the computational time in terms of the number of time steps. The time step duration is, however, different for the three structures to be compared below. As an alternative, consider the time complexity as the product of the computational time in numbers of time steps and the time step duration.

The inverter developed by Wang et al. [43] requires only $4m-1$ registers and m^2 AND gates. On average, for an arbitrary irreducible polynomial the circuit has $0.5m^3$ XOR gates [42], which results in $\lceil \log_2 0.5m^3 \rceil$ levels of XOR gates. New circuitry is required for the *product function* [43] if a different irreducible polynomial is chosen. The inverter also requires two control signals. For bit-serial input and output formats, the computation time is only $3m$ time steps, yielding a time complexity of $O(m \log m)$.

The number of registers for the inverter proposed by Akari et al. [1] is $7(m+1)$. Each cell of the inverter requires 21 switches. If the switches are realized by AND and OR operations, then the total number of AND gates, OR gates and XOR gates in the inverter are $45(m+1)$, $23(m+1)$ and $4(m+1)$, respectively. The structure is independent of the irreducible polynomial. However, the four control signals used to direct data to different registers are broadcast to all the basic processors. For bit-serial input and output operation the computational time varies from $4m+3$

Features	Circuit of Wang et al. [43]	Circuit of Akari et al. [1]	Circuit of this paper
Operation	Inversion	Inversion	Division
Number of registers	$O(m)$	$O(m)$	$O(m^2)$
Number of AND gates	$O(m^2)$	$O(m)$	$O(m^2)$
Number of XOR gates	$O(m^3)$	$O(m)$	$O(m^2)$
Number of control signals	Two	Four	One
Circuit independent of irreducible polynomial	No	Yes	Yes
Equal computation time for all elements	Yes	No	Yes
Global data communications	Yes	Yes	No
Time step independent of m	No	No	Yes
Time complexity	$O(m \log m)$	$O(m^2)$	$O(m)$

Table 4.3: Comparison of the circuits for inversion/division over GF(2^m).

to $5m+2$ time steps. One of the four control signals is generated by passing data through $m+1$ OR gates in a single time step. As a result, the time step duration increases linearly with m, resulting in a time complexity of $O(m^2)$.

The bit-serial systolic divider presented here requires $2.5m^2+11.5m-6$ registers, $4m^2+12m-5$ AND gates, $1.5m^2+7.5m-2$ OR gates and $0.5m^2+1.5m-1$ XOR gates. However, the structure is independent of the irreducible polynomial. There is no global data communication and only one easily generated control signal is required. The computational time is $5m-1$ time steps. The maximum time delay due to the gates occurs in the square processor and consists of the delays due to a XOR gate and a multiplexer. This delay is, however, independent of m yielding a time complexity of $O(m)$. We summarize the comparison in Table 4.3.

4.6 Conclusions

In this chapter, a bit-serial systolic structure has been presented for performing division over GF(2^m). The structure requires a simple control signal, and regular and local interconnections. As a result, it is well suited for VLSI systems. Moreover, the time step duration does not depend on m, hence the structure is more suitable for applications where large values of m are used.

Chapter 5

Bit-Serial Multiplication over $GF(q^m)$

5.1 Introduction

Berlekamp was the first to develop a bit-serial dual basis multiplication algorithm over GF(2^m) for the encoding of Reed-Solomon codes [6]. A block diagram for computing $c = ab$ over GF(2^m) using Berlekamp's bit-serial multiplication scheme is shown in Figure 5.1. When both the multiplicand and the multiplier are represented with respect to a common (also referred to as primal) basis, which is expected for most practical cases, the above multiplication scheme requires two basis transformations in addition to Berlekamp's bit-serial multiplication circuit. The latter requires only $2m$ shift registers, m AND gates and $m + W_H(g) - 3$ XOR gates, where $W_H(g)$ denotes Hamming weight of the irreducible polynomial $g(z)$ used to generate GF(2^m). However, the circuitry for the basis transformations are not always simple [38]. An explanation of Berlekamp's bit-serial multiplication scheme can be found in McEliece [30].

Berlekamp's algorithm is very efficient in the sense that it requires a minimum of circuitry. It has the additional advantage that multiplication by a fixed constant can be hardwired. However, the involvement of two different bases is not advantageous especially when the multiplier is to be used as a part of a larger device;

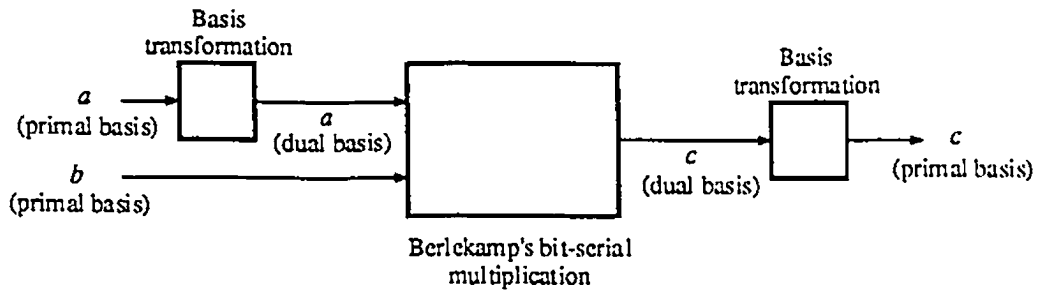


Figure 5.1: Involvement of dual basis in Berlekamp's bit-serial multiplication scheme.

because it would be necessary in general to enhance circuitry to change bases [30].

There are some cases for which the basis transformation is very easily accomplished, i.e., by a permutation of the coordinates. Such an easily accomplished basis change depends on the irreducible polynomial chosen. Morii, Kasahara and Whiting have shown in [31] that it is not necessary to use the dual basis for the realization of an efficient bit-serial multiplication when the irreducible polynomial is a trinomial. They have also shown that when the irreducible polynomial is of the form of $g(z) = z^m + z^{k+2} + z^{k+1} + z^k + 1$, $0 < k < m-2$, only a simple transformation of the bases is necessary to have an efficient bit-serial multiplication circuit.

Recently Wang and Blake [44] have proved that the element transformation into the dual basis can be performed by a simple permutation of the coefficients if and only if the irreducible polynomial is a trinomial. They have developed a bit-serial multiplication scheme which can be realized for all irreducible polynomials over GF(2^m). However, the requirement for some form of basis transformation circuit both at the input and at the output still exists. As a result, it is desirable to develop an algorithm where the number of gates and registers required for the basis transformation circuit can be minimized.

In this chapter, the relationship between the DTWHE and finite field division of Chapter 3 is used to develop a bit-serial multiplication algorithm which can be easily realized for all irreducible polynomials [12].

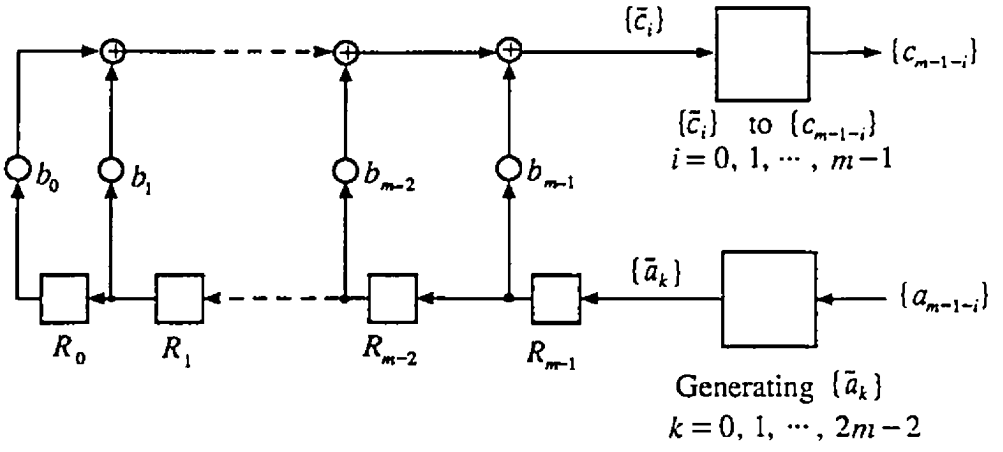


Figure 5.2: Conceptual diagram for bit-serial multiplication.

5.2 Bit-Serial Multiplication

If the coordinates of a and b are known, Equation (3.16) can be used to obtain a bit-serial multiplication circuit. The conceptual diagram for a bit-serial multiplier using (3.16) is shown in Figure 5.2. The inputs a_{m-1}, a_{m-2}, $\cdots$, a_0 generate $\tilde{a}_0$, $\tilde{a}_1$, $\cdots$, $\tilde{a}_{2m-2}$ which are sequentially shifted in to the registers R_0, R_1, $\cdots$, R_{m-1}. At each clock pulse the contents of these registers are multiplied by b_0, b_1, $\cdots$, b_{m-1}, respectively. This is equivalent to multiply one row vector of the Toeplitz matrix of (3.16) by the column vector $[b_{m-1}, b_{m-2}, \cdots, b_0]^t$ yielding the elements of the vector $[\tilde{c}_0, \tilde{c}_1, \cdots, \tilde{c}_{m-1}]^t$. The latter is then transformed to get the required results c_{m-1}, c_{m-2}, $\cdots$, c_0.

We now discuss shift register configurations to generate the constants $\tilde{a}_0$, $\tilde{a}_1$, $\cdots$, $\tilde{a}_{2m-2}$ and to transform $\tilde{\mathbf{c}} \triangleq [\tilde{c}_i]_{i=0}^{m-1}$ to $\mathbf{c} \triangleq [c_i]_{i=0}^{m-1}$.

5.2.1 LFSR Configuration for $\{\tilde{a}_0,\ \tilde{a}_1,\ \cdots,\ \tilde{a}_{2m-2}\}$

Since $g(z)$ is an irreducible monic polynomial over GF(q) and $\alpha \in$ GF(q^m) satisfies $g(\alpha) = 0$, we have

$$\alpha^m = -\sum_{i=0}^{m-1} g_i \alpha^i.$$

Again α^m is an element of the set H of supporting elements and we can write from (3.2)

$$\alpha^m = \sum_{i=0}^{m-1} p_i^{[m]} \alpha^i.$$

Thus

$$p_i^{[m]} = -g_i \quad i = 0,\ 1,\ \cdots,\ m-1.$$

Figure 5.3 is a well known configuration for α-multiplication over GF(q^m). In Figure 5.3 all the registers and connecting lines are assumed to function under q-valued logic. If the coordinates $x_0,\ x_1,\ \cdots,\ x_{m-1}$ of the element $x = \sum_{i=0}^{m-1} x_i \alpha^i$ are stored in the registers $B_0,\ B_1,\ \cdots,\ B_{m-1}$ respectively, then after one clock pulse the registers contain the coordinates of the product αx. Thus if the registers initially contain the coordinates of the supporting element α^{m-1}, then the contents of the last register B_{m-1} with successive clock pulses (up to $m-1$ pulses) can be expressed as follows:

$$p_{m-1}^{[m-1+j]} \triangleq d_j = \begin{cases} 1 & j = 0 \\ -\sum_{i=0}^{j-1} g_{m-j+i} p_{m-1}^{[m-1+i]} \pmod q & j = 1,\ 2,\ \cdots,\ m-1. \end{cases} \tag{5.1}$$

Now we present a Corollary of Theorem 3.2.

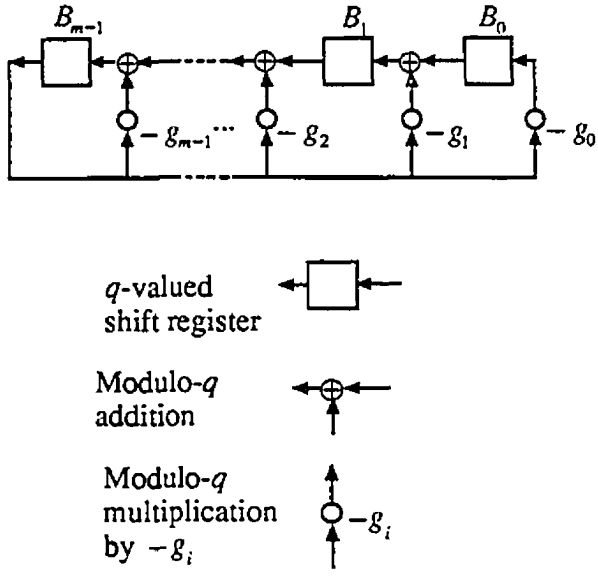


Figure 5.3: LFSR configuration for multiplication by α.

Corollary 5.1

$$\tilde{a}_l = \sum_{i=0}^{l} a_{m-1-l+i} d_i \quad (l = 0,\ 1,\ \cdots,\ m-1). \tag{5.2}$$

Proof: From (3.17),

$$\tilde{a}_k = \sum_{l=0}^{m-1} a_l p_{m-1}^{[l+k]}.$$

Substituting $l = m - 1 - k + i$, we obtain

$$\tilde{a}_k = \sum_{i=k-(m-1)}^{k} a_{m-1-k+i} p_{m-1}^{[m-1+i]}.$$

In the canonical basis, for $0 \leq j \leq m-1$

$$p_{m-1}^{[j]} = \begin{cases} 1 & \text{if } j = m-1 \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

So

$$\begin{aligned}\tilde{a}_k &= \sum_{i=0}^{k} a_{m-1-k+i} p_{m-1}^{[m-1+i]} \\ &= \sum_{i=0}^{k} a_{m-1-k+i} d_i \qquad \text{Q.E.D.}\end{aligned}$$

Equation (5.1), in conjunction with (3.12) and (5.2), is used to derive the following recursive relationship, presented as a Corollary of Theorem 3.2, which results in a LFSR (linear feedback shift register) configuration for the generation of $\tilde{a}_k$.

Corollary 5.2 The constant coefficients $\tilde{a}_k$ $(k = 0,\ 1,\ \cdots,\ 2m-2)$ of the DTWHE (3.16) are

$$\tilde{a}_k = \begin{cases} a_{m-1} & (k=0) \\ a_{m-1-k} - \sum_{l=0}^{k-1} \tilde{a}_l g_{m-k+l} \pmod q & (1 \le k \le m-1) \\ -\sum_{l=0}^{m-1} \tilde{a}_{k-1-l} g_{m-1-l} \pmod q & (m \le k \le 2m-2) \end{cases} . \tag{5.4}$$

Proof: With $k = 0$, substitute (5.3) in (3.17) to obtain $\tilde{a}_0 = a_{m-1}$. Now consider $\tilde{a}_k$ $(1 \le k \le m-1)$. With the help of (5.2) the R.H.S. of (5.4) can be written as

$$\text{R.H.S} = a_{m-1-k} - \sum_{l=0}^{k-1} \left(\sum_{i=0}^{l} a_{m-1-l+i} d_i \right) g_{m-k+l}. \tag{5.5}$$

Let

$$\begin{aligned} X &= \sum_{l=0}^{k-1} \left(\sum_{i=0}^{l} a_{m-1-l+i} d_i \right) g_{m-k+l} \\ &= \sum_{l=0}^{k-1} \left(a_{m-1-l} d_0 + a_{m-1-l+1} d_1 + a_{m-1-l+2} d_2 + \cdots + a_{m-1} d_l \right) g_{m-k+l} \end{aligned}$$

Using the fact that $a_j = 0$ when $j \geq m$ or $j < 0$, results in

$$\begin{aligned} X &= a_{m-1}d_0 g_{m-k} \\ &\quad + (a_{m-2}d_0 + a_{m-1}d_1)\, g_{m-k+1} \\ &\quad + (a_{m-3}d_0 + a_{m-2}d_1 + a_{m-1}d_2)\, g_{m-k+2} \\ &\quad \vdots \\ &\quad + (a_{m-k}d_0 + a_{m-k+1}d_1 + \cdots + a_{m-1}d_{k-1})\, g_{m-1} \\ &= a_{m-1}\,(g_{m-k}d_0 + g_{m-k+1}d_1 + \cdots + g_{m-1}d_{k-1}) \\ &\quad + a_{m-2}\,(g_{m-k+1}d_0 + g_{m-k+2}d_1 + \cdots + g_{m-1}d_{k-2}) \\ &\quad \vdots \\ &\quad + a_{m-k}\,(g_{m-1}d_0) \end{aligned} \tag{5.6}$$

Substituting (5.1) into (5.6)

$$\begin{aligned} X &= -a_{m-1}d_k - a_{m-2}d_{k-1} - a_{m-3}d_{k-2} - \cdots - a_{m-k}d_1 \\ &= -\sum_{l=1}^{k} a_{m-1-k+l}d_l \end{aligned}$$

Thus (5.5) becomes

$$\begin{aligned} \text{R.H.S.} &= a_{m-1-k} - X \\ &= \sum_{l=1}^{k} a_{m-1-k+l}d_l = \tilde{a}_k = \text{L.H.S.} \end{aligned}$$

This completes the proof.

Combining the three cases for $\tilde{a}_k$ viz., $k = 0$, $1 \leq k \leq m-1$ and $m \leq k \leq 2m-2$, $\tilde{a}_k$ $(k = 0, 1, \cdots, m-1)$ can be generated sequentially in the register R_{m-1} of Figure 5.4 provided that the LFSR initially contains zero and the input to the configuration is the sequence $\{a_{m-1}, a_{m-2}, \cdots, a_0, 0, 0, \cdots, 0\}$ of $2m-1$ elements.

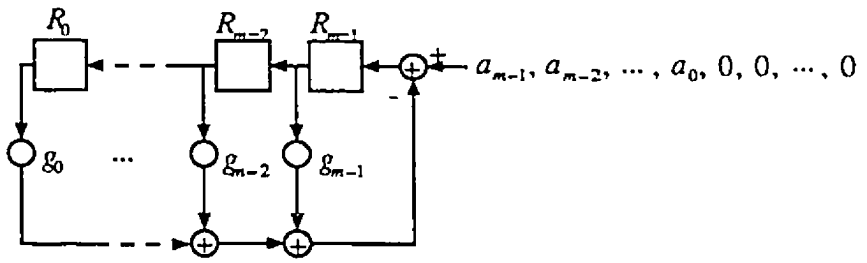


Figure 5.4: Generation of $\tilde{a}_k$.

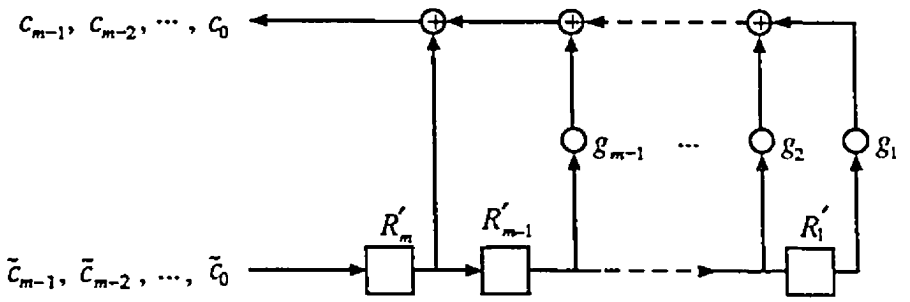


Figure 5.5: Transformation of $\tilde{\mathbf{c}}$ to $\mathbf{c}$

5.2.2 Transformation of $\tilde{\mathbf{c}}$ to $\mathbf{c}$

From (3.18), i.e.,

$$\tilde{c}_i = \begin{cases} c_{m-1} & \text{if } i = 0 \\ c_{m-1-i} - \sum_{l=1}^{i} \tilde{c}_{i-l} g_{m-l} \pmod q & \text{if } i = 1, 2, \cdots, m-1 \end{cases}$$

we have

$$c_{m-1-i} = \begin{cases} \tilde{c}_0 & \text{if } i = 0 \\ \sum_{l=0}^{i} \tilde{c}_{i-l} g_{m-l} \pmod q & \text{if } i = 1, 2, \cdots, m-1 \end{cases} . \tag{5.7}$$

A feed forward shift register configuration to transform $\tilde{\mathbf{c}}$ to $\mathbf{c}$ is shown in Figure 5.5. It is assumed that the logic is q-valued and the registers initially contain 0. The input is the sequence $\{\tilde{c}_0, \tilde{c}_1, \cdots, \tilde{c}_{m-1}\}$ and the corresponding output is $\{c_{m-1}, c_{m-2}, \cdots, c_0\}$.

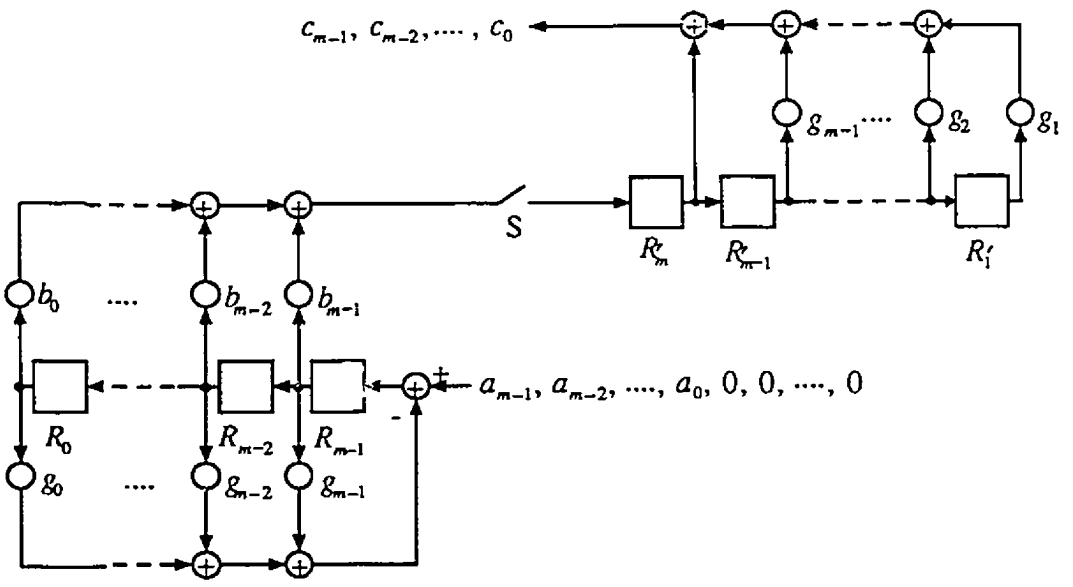


Figure 5.6: The bit-serial multiplication circuit.

Figure 5.6 shows the complete configuration for bit-serial multiplication. All registers are initially set to 0. The input to the circuit is the sequence $\{a_{m-1}, a_{m-2}, \cdots, a_0, 0, 0, \cdots, 0\}$ of $2m$ elements. As $a_{m-1}, a_{m-2}, \cdots, a_0$ enter the LFSR, $\tilde{a}_0, \tilde{a}_1, \cdots, \tilde{a}_{m-1}$ are loaded into $R_0, R_1, \cdots, R_{m-1}$ respectively; at which point the switch S closes. For the next m clock cycles, $c_{m-1}, c_{m-2}, \cdots, c_0$ are obtained at the output sequentially as shown in Figure 5.6.

If the irreducible monic polynomial is $g(z) = \sum_{i=0}^{m-1} g_i z^i + z^m$, $g_i \in \mathrm{GF}(q)$ with $g_k \neq 0$, where k $(0 < k < m)$ is the least nonzero positive integer, then only a $m - k + 1$ stage shift register is required for the feed forward shift register of Figure 5.6. Thus by choosing a suitable $g(z)$ with k as large as possible, the number of registers for the bit-serial multiplication circuit can be reduced. For example, both $1 + z + z^2 + z^3 + z^{31}$ and $1 + z^{28} + z^{29} + z^{30} + z^{31}$ are irreducible polynomials over GF(2) of weight five [32]. However, the former requires 31 stages of registers in the feed forward shift register while the latter requires only 4.

Components	Circuit of [44]	Circuit presented here
Number of shift registers	$5m$	$3m$
Number of 2-input AND gates	$2m$	m
Number of 2-input XOR gates	$2(m-1)+3[W_H(g)-2]$	$(m-1)+2[W_H(g)-2]+1$

Table 5.1: Comparison of number of gates and registers of two bit-serial multiplication circuits.

5.3 Comparison

If the irreducible polynomial is a trinomial or a pentanomial of the form of $g(z) = 1+z^k+z^{k+1}+z^{k+2}+z^m$, $(0 < k < m-2)$, then the bit-serial multiplication scheme of [31] for GF(2^m) requires a very simple basis transformation. However, irreducible trinomials and pentanomials do not exist for all m. A computer search for a suitable basis transformation as suggested in [31] is a case dependent approach. From this point of view the bit-serial multiplication scheme proposed here has the advantage of being applicable for all irreducible polynomials.

The bit-serial multiplication scheme presented here and that developed by Wang and Blake [44] require a multiplicand and multiplier in terms of a canonical basis, and both generate a product which is also in terms of a canonical basis. Functionally these two schemes are equivalent; however, a reduction in the number of gates and shift registers is obtained by using the scheme presented here. This is shown in Table 5.1 for GF(2^m). To determine the number of gates, it is assumed that if $g_i = 1$ $(i = 0, 1, \cdots, m-1)$, then the feedback (or feed forward) connection with weight g_i in Figure 5.6 exists and a XOR gate is used. The total number of clock cycles required for the proposed multiplier is $2m$ and the multiplier of [44] may need an additional m cycles to maintain the same ordering of the coordinates at the input and output.

5.4 Conclusions

The relationship between the finite field division and the discrete time Wiener-Hopf equation has lead to the development of a bit-serial multiplication scheme. The attractive feature of this multiplication scheme is that it can easily be realized for all irreducible polynomials and in many cases will require fewer gates and shift registers compared to other bit-serial multiplication algorithms.

Chapter 6

Parallel Multiplication for a Class of GF(2^m)

6.1 Introduction

The two important finite field operations- exponentiation and inversion can be performed by applying multiplication operations repeatedly. Hence it is important to develop fast multiplication algorithms which can be easily realized with low circuit complexity. When the size of the finite field is large, another issue which requires considerable attention is the modular structure of the multiplier.

A modular parallel structure has been presented in [43] using the Massey-Omura multiplication algorithm to multiply any two elements of GF(2^m) where each element of the field is represented with respect to a normal basis. Although in general the parallel Massey-Omura multiplier (MOM) requires $O(m^3)$ XOR and $O(m^2)$ AND gates (all gates are assumed to have only two inputs), a reduction in the number of gates can be obtained by using a suitable irreducible polynomial. If the irreducible polynomial is an *all one polynomial* (AOP), then only $2m^2 - 2m$ XOR and m^2 AND gates are required for the parallel multiplier [41]. A more general treatment of this topic is given in [2].

By representing field elements with respect to a canonical basis, Itoh and Tsujii have developed a structure for a parallel multiplier over GF(2^m) based on ir-

reducible AOPs [21]. They have also extended their multiplication algorithm for irreducible *equally spaced polynomials* (ESP). Their structure is modular and has a lower circuit complexity compared to the MOM.

In this chapter, the field elements are represented with respect to a canonical basis. Two structures for parallel multipliers over GF(2^m) based on an irreducible AOP of degree m and irreducible ESPs of degree $m(m+1)^i$, $i = 1, 2, \cdots$ are presented in [15], [14]. By increasing i, the degree of the ESP can be increased arbitrarily. It is shown how to construct a parallel multiplier for such large fields using the basic modules of the parallel multiplier based on the corresponding irreducible AOP of degree m.

We also develop a squaring algorithm for GF(2^m) based on an irreducible AOP. The realization of the squaring algorithm is very simple. Applying these algorithms for multiplication and squaring operations, a structure for computing inverses is also developed.

6.2 Itoh-Tsujii Parallel Multipliers

Definition 6.1 [41] A polynomial $f(z) = \sum_{i=0}^{m} f_i z^i$ over GF(2) is called an AOP of degree m if $f_i = 1$ for $i = 0, 1, \cdots, m$.

Examples of possible values of m for which an AOP of degree m is irreducible are 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, 100, 106, 130, 138, etc. [21].

Definition 6.2 [21] A polynomial $g(z) = \sum_{i=0}^{sm} g_i z^i = z^{sm} + z^{s(m-1)} + \cdots + z^s + 1 = f(z^s)$ over GF(2), where $f(z)$ is an AOP of degree m over GF(2), is called an *s-equally spaced polynomial* (s-ESP) of degree sm.

The coefficients of the ESP are

$$g_i = \sum_{k=0}^{m} \delta_{i,sk}. \tag{6.1}$$

Examples of possible values of n for which an ESP of degree n is irreducible are 6, 18, 20, 54, 100, 110, 156, 162, etc. [21].

Let $f(z) = \sum_{i=0}^{m} f_i z^i$ be an irreducible monic polynomial of degree m over GF(2) and α a root of $f(z)$. Any element of GF(2^m), say a, can be represented with respect to the canonical basis $\{\alpha^0, \alpha^1, \cdots, \alpha^{m-1}\}$, i.e., $a = a_0\alpha^0 + a_1\alpha^1 + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1}$, where the coordinates $a_i \in$ GF(2) for $0 \le i \le m-1$. With this representation, Itoh and Tsujii have developed structures for parallel multipliers based on AOP and ESP [21]. In this subsection we briefly review the configurations of Itoh-Tsujii multipliers (hereafter denoted by ITM).

ITM based on AOP: Let $f(z)$ be an irreducible AOP of degree m and α a root of $f(z)$. For any two elements $a = \sum_{i=0}^{m-1} a_i\alpha^i$ and $b = \sum_{i=0}^{m-1} b_i\alpha^i$ in GF(2^m), let $c = \sum_{i=0}^{m-1} c_i\alpha^i$ denote the product of a and b. If we represent

$$a = A_0 + A_1\alpha + \cdots + A_m\alpha^m$$

and

$$b = B_0 + B_1\alpha + \cdots + B_m\alpha^m,$$

then, by noting $\alpha^{m+1} = 1$, it is easy to show that

$$c = ab = C_0 + C_1\alpha + \cdots + C_m\alpha^m,$$

where

$$C_k = \sum_{i+j=k \pmod{m+1}} A_iB_j \pmod 2 \quad (0 \le k \le m). \tag{6.2}$$

The coefficients c_i are then given by

$$c_k = C_k + C_m \pmod 2 \quad (0 \le k \le m-1). \tag{6.3}$$

The basic module of the ITM computes C_k as indicated in (6.2). Each module contains $(m+1)$ AND gates and m XOR gates; and there are $m+1$ basic modules in the ITM. The transformation (6.3) requires m XOR gates. So in the ITM, there are $(m+1)^2$ AND gates and m^2+2m XOR gates.

ITM based on ESP: Let $g(z)$ be an irreducible s-ESP of degree $n = ms$ where $s = (m+1)^i$ for any positive integer i and let β be a root of $g(z)$. a and b are any two elements of GF(2^n) and $a = \sum_{i=0}^{n-1} a_i\beta^i$ and $b = \sum_{i=0}^{n-1} b_i\beta^i$. Let $c = \sum_{i=0}^{m-1} c_i\beta^i$ denote the product of a and b. Letting $r = s+n$, if we represent

$$a = A_0 + A_1\beta + \cdots + A_{r-1}\beta^{r-1}$$

and

$$b = B_0 + B_1\beta + \cdots + B_{r-1}\beta^{r-1},$$

by using $\beta^r = 1$ (see Theorem 6.2), we obtain

$$c = ab = C_0 + C_1\beta + \cdots + C_{r-1}\beta^{r-1},$$

where

$$C_k = \sum_{i+j=k \pmod r} A_iB_j \pmod 2 \quad (0 \le k \le r-1). \tag{6.4}$$

The coefficients c_i are given by

$$c_{i+js} = C_{i+js} + C_{i+n} \pmod 2 \quad (0 \le i \le s-1,\ \ 0 \le j \le m-1). \tag{6.5}$$

The basic function of the multiplier based on an ESP for GF(2^n) can be realized by r AND gates and $r-1$ XOR gates. The total number of AND gates is r^2 and the number of XOR gates is $r(r-1)+n$.

6.3 Proposed AOP Based Parallel Multiplier

6.3.1 Algorithm

Let $a = \sum_{i=0}^{m-1} a_i\alpha^i$ and $b = \sum_{i=0}^{m-1} b_i\alpha^i$ be any two elements in GF(2^m) and let $c = \sum_{i=0}^{m-1} c_i\alpha^i$ denote the product of a and b, where α is a root of an irreducible polynomial $f(z) = \sum_{i=0}^{m} f_i z_i$ of degree m over GF(2). Then we have (3.16) which is rewritten below:

$$\begin{bmatrix} \tilde{a}_{m-1} & \tilde{a}_{m-2} & \cdots & \tilde{a}_0 \\ \tilde{a}_m & \tilde{a}_{m-1} & \cdots & \tilde{a}_1 \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{a}_{2m-2} & \tilde{a}_{2m-3} & \cdots & \tilde{a}_{m-1} \end{bmatrix} \begin{bmatrix} b_{m-1} \\ b_{m-2} \\ \vdots \\ b_0 \end{bmatrix} = \begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \vdots \\ \tilde{c}_{m-1} \end{bmatrix} \tag{6.6}$$

where

$$\tilde{a}_k = \sum_{i=0}^{m-1} a_i\, p_{m-1}^{[i+k]} \qquad k = 0,\ 1,\ \cdots,\ 2m-2 \tag{6.7}$$

and

$$\tilde{c}_i = \begin{cases} c_{m-1} & \text{if } i = 0 \\ c_{m-1-i} + \sum_{l=1}^{i} \tilde{c}_{i-l} f_{m-l} & \text{if } i = 1,\ 2,\ \cdots,\ m-1. \end{cases} \tag{6.8}$$

According to (6.6), the corresponding multiplication algorithm has the following steps.

S1. Transformation of $a_0,\ a_1,\ \cdots,\ a_{m-1}$ to $\tilde{a}_0,\ \tilde{a}_1,\ \cdots,\ \tilde{a}_{2m-2}$ (Equation (6.7)).

S2. Matrix multiplication to obtain $\tilde{c}_0,\ \tilde{c}_1,\ \cdots,\ \tilde{c}_{m-1}$ (Equation (6.6)).

S3. Transformation of $\tilde{c}_0,\ \tilde{c}_1,\ \cdots,\ \tilde{c}_{m-1}$ to the desired coordinates $c_0,\ c_1,\ \cdots,\ c_{m-1}$ (Equation (6.8)).

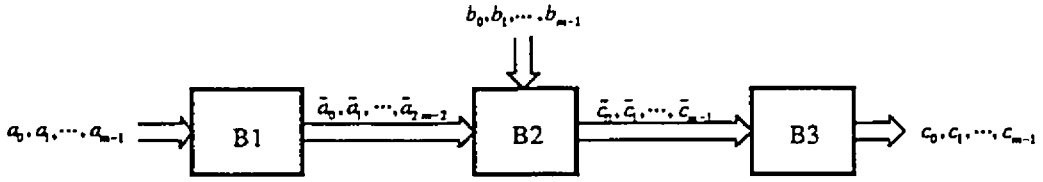


Figure 6.1: Block diagram of the realization of a parallel multiplication.

Figure 6.1 shows a general block diagram for the realization of a parallel multiplier using the above algorithm. Note that, if $f(z)$ in Eq. (6.6) is not irreducible, then the algorithm will perform multiplication over a polynomial ring modulo $f(z)$.

The $m \times m$ Toeplitz matrix in (6.6) has a maximum of $2m-1$ distinct entries. The complexity for the realization of **S1** (and consequently for the whole parallel multiplier), depends on the number of distinct entries in the Toeplitz matrix. By choosing a suitable irreducible polynomial $f(z)$ in Eq. (6.6), the number of distinct entries can be reduced. The focus of the following discussion is on the construction of a parallel multiplier based on an irreducible AOP using the above multiplication algorithm.

For an irreducible AOP $f(z) = \sum_{i=0}^{m} z^i$ over GF(2), a root α of $f(z)$ satisfies $\alpha^m = \sum_{i=0}^{m-1} \alpha^i$ and $\alpha^{m+1} = 1$. Hence, for any integer j, we have

$$p_i^{[j]} = p_i^{[j \bmod (m+1)]} \quad (i = 0,\ 1,\ \cdots,\ m-1) \tag{6.9}$$

where $p_i^{[j]}$ for $0 \leq i \leq m-1$ is the ith coordinate of α^j.

When $f(z)$ is an irreducible AOP, $f_i = 1$ for $i = 0,\ 1,\ \cdots,\ m-1$ and (3.12) becomes

$$p_i^{[k]} = \begin{cases} p_{m-1}^{[k-1]} & i = 0 \\ p_{m-1}^{[k-1]} + p_{i-1}^{[k-1]} & i = 1,\ 2,\ \cdots,\ m-1. \end{cases} \tag{6.10}$$

If (6.9) is substituted in (6.7), there are only $m+1$ distinct $\tilde{a}_k$ and they can be expressed in terms of the coordinates of a. Combining (4.4) and (6.10) for $0 \leq k \leq m$, the result is

$$p_{m-1}^{[k]} = \delta_{k,m-1} + \delta_{k,m}. \tag{6.11}$$

Using (6.9) and (6.11) in (6.7)

$$\tilde{a}_k = \begin{cases} a_{m-1} & k = 0 \\ a_{m-1-k} + a_{m-k} \pmod 2 & k = 1,\ 2,\ \cdots,\ m-1 \\ a_0 & k = m. \end{cases} \tag{6.12}$$

For $m+1 \leq k \leq 2m-2$, (6.9) results in

$$\tilde{a}_k = \tilde{a}_{k \bmod (m+1)}. \tag{6.13}$$

We therefore have

$$\begin{bmatrix} \tilde{a}_{m-1} & \tilde{a}_{m-2} & \tilde{a}_{m-3} & \cdots & \tilde{a}_1 & \tilde{a}_0 \\ \tilde{a}_m & \tilde{a}_{m-1} & \tilde{a}_{m-2} & \cdots & \tilde{a}_2 & \tilde{a}_1 \\ \tilde{a}_0 & \tilde{a}_m & \tilde{a}_{m-1} & \cdots & \tilde{a}_3 & \tilde{a}_2 \\ \tilde{a}_1 & \tilde{a}_0 & \tilde{a}_m & \cdots & \tilde{a}_4 & \tilde{a}_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \tilde{a}_{m-3} & \tilde{a}_{m-4} & \tilde{a}_{m-5} & \cdots & \tilde{a}_m & \tilde{a}_{m-1} \end{bmatrix} \begin{bmatrix} b_{m-1} \\ b_{m-2} \\ b_{m-3} \\ b_{m-4} \\ \vdots \\ b_0 \end{bmatrix} = \begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \\ \tilde{c}_3 \\ \vdots \\ \tilde{c}_{m-1} \end{bmatrix}$$

or, equivalently

$$\tilde{c}_i = \sum_{j=0}^{m-1} b_j \tilde{a}_{i+j \bmod (m+1)} \pmod 2 \quad i = 0,\ 1,\ \cdots,\ m-1. \tag{6.14}$$

For the AOP $f(z)$ of degree m, the transformation of $\tilde{c}_0, \tilde{c}_1, \cdots, \tilde{c}_{m-1}$ to $c_0, c_1, \cdots, c_{m-1}$ is obtained from (6.8) as

$$\begin{aligned} c_{m-1-i} &= \sum_{l=0}^{i} \tilde{c}_i \pmod 2, \quad i = 0,\ 1,\ \cdots,\ m-1 \\ &= \begin{cases} \tilde{c}_0 & i = 0 \\ \tilde{c}_i + c_{m-i} \pmod 2, & i = 1,\ 2,\ \cdots,\ m-1. \end{cases} \end{aligned} \tag{6.15}$$

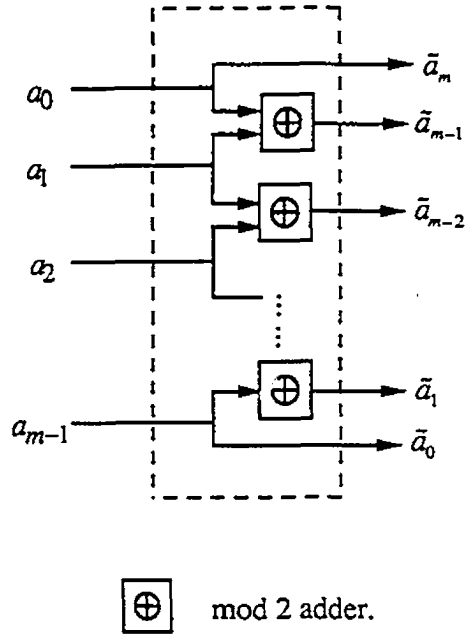


Figure 6.2: Transformation of $a_0, a_1, \cdots, a_{m-1}$ to $\tilde{a}_0, \tilde{a}_1, \cdots, \tilde{a}_m$ (Module P).

Note that (6.12) (along with (6.13)), (6.14) and (6.15) correspond to **S1**, **S2** and **S3**, respectively, for an irreducible AOP of degree m.

6.3.2 Structure, Complexity and Comparison

Denote the circuits which realize (6.12), (6.14) and (6.15) as modules P, Q and R, respectively. Details of these modules are shown in Figures 6.2, 6.3 and 6.4. Module P transforms $a_0, a_1, \cdots, a_{m-1}$ to $\tilde{a}_0, \tilde{a}_1, \cdots, \tilde{a}_m$, and requires $m-1$ XOR gates. There are m identical cells in module Q for matrix multiplication. The inputs to the i-th cell are two m-tuple vectors, viz., $\left[\tilde{a}_{(i+m-1)}, \tilde{a}_{(i+m-2)}, \cdots, \tilde{a}_{(i)}\right]$ and $[b_{m-1}, b_{m-2} \cdots, b_0]^t$, where (x) in the subscript is used to denote x mod $(m+1)$. The m-tuple vector $\left[\tilde{a}_{(i+m-1)}, \tilde{a}_{(i+m-2)}, \cdots, \tilde{a}_{(i)}\right]$ is tapped from the $m+1$ bit bus containing $\tilde{a}_m, \tilde{a}_{m-1}, \cdots, \tilde{a}_0$. The output of the ith cell is $\tilde{c}_i = \sum_{j=0}^{m-1} b_j \tilde{a}_{(i+j)}$. Each cell contains $m-1$ XOR gates and m AND gates to realize the mod 2 additions

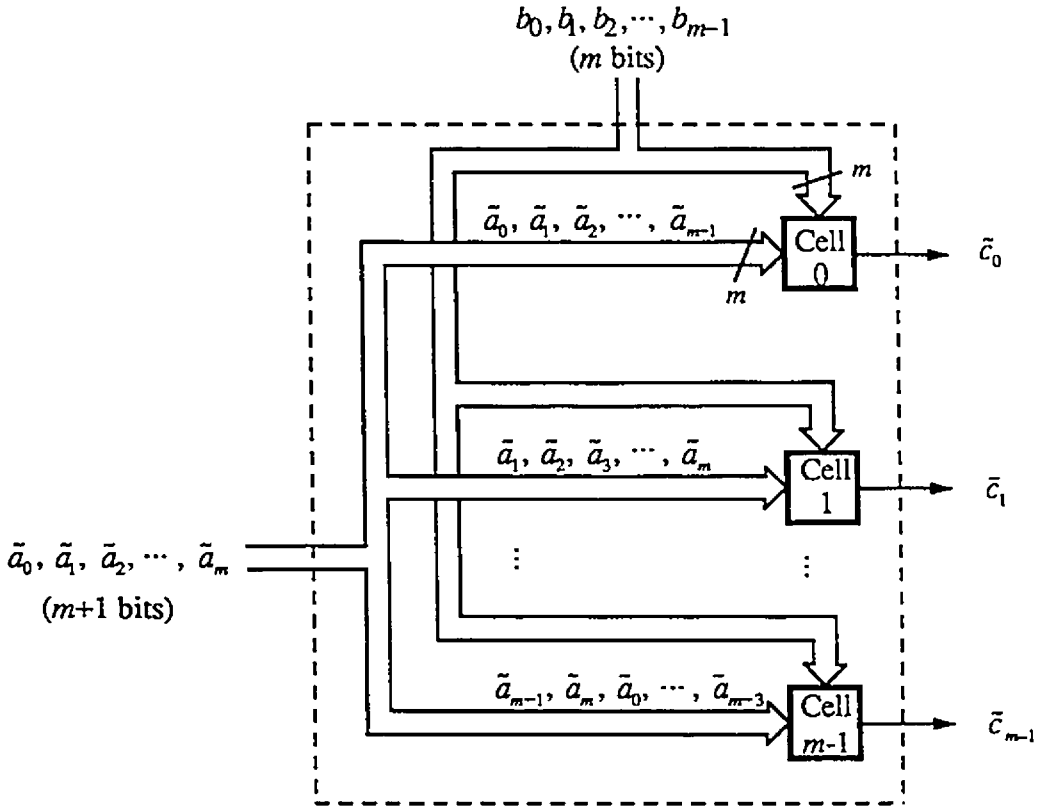


Figure 6.3: Matrix multiplication (Module Q).

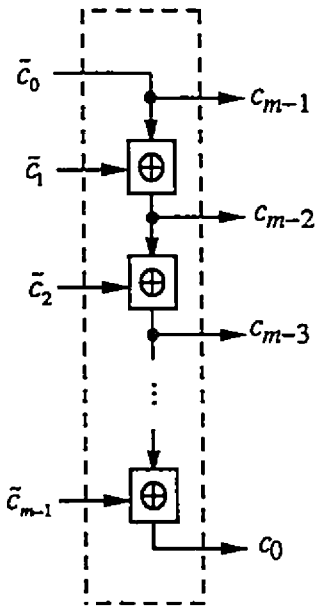


Figure 6.4: Transformation of $\tilde{c}_0, \tilde{c}_1, \cdots, \tilde{c}_{m-1}$ to $c_0, c_1, \cdots, c_{m-1}$ (Module R).

Multiplier description	Number of XOR gates	Number of AND gates	Time delays due to gates
MOM [43]	$2m^2 - 2m$	m^2	$D_A + (1+ \lceil \log_2(m-1) \rceil) D_X$
ITM [21]	$m^2 + 2m$	$m^2 + 2m + 1$	$D_A + \lceil \log_2 m + \log_2(m+2) \rceil D_X$
Presented here	$m^2 + m - 2$	m^2	$D_A + (m - 1 + \lceil \log_2(m-1) \rceil) D_X$

Table 6.1: Comparison of number of gates and time delays for three parallel multipliers based on the irreducible AOP of degree m.

and multiplications, and introduces a time delay of $D_A + \lceil \log_2(m-1) \rceil D_X$ where D_X and D_A denote the delay for an XOR gate and an AND gate respectively. Module R transforms $\tilde{c}_0, \tilde{c}_1, \cdots, \tilde{c}_{m-1}$ to $c_0, c_1, \cdots, c_{m-1}$ using $m-1$ XOR gates. Due to the dependency of c_i on c_{i+1}, the delay for this transformation is $(m-1)D_X$ and the total delay in the three modules is $D_A + (m + \lceil \log_2(m-1) \rceil) D_X$. The total number of XOR and AND gates are $m^2 + m - 2$ and m^2, respectively. The number of gates required is less than that of [43] or [21]. A comparison of the number of gates and approximate time delays for these modular structured parallel multipliers is shown in Table 6.1.

For an AOP based parallel multiplier, block B1 (refer to Figure 6.1) contains one P module, B2 contains one Q module and B3 contains one R module. Later it will be shown that an ESP based parallel multiplier can be constructed by using additional modules of the same type in each block.

It can be shown that, for arbitrary $f(z)$, the number of gates in the multiplier structure is $O(m^2)$, which leads to a partial solution to the problem proposed by Itoh and Tsujii in the last paragraph of [21].

6.4 Inversion

One of the main objectives in developing a parallel multiplier is to use it repeatedly for the computation of inverses and exponentiations in finite fields [43]. The parallel multiplier presented in the previous section has a lower circuit complexity, but a longer time delay compared to those in [43] and [21]. In this section we consider a structure for the fast computation of inverses. A similar approach can also be used to obtain a structure for computing exponentiations.

It is well known that for $a \in \mathrm{GF}(2^m)$, $a^{-1} = a^{2^m-2}$ and therefore [43]

$$a^{-1} = a^2 a^{2^2} \cdots a^{2^{m-1}}. \tag{6.16}$$

Thus, the computation of the inverse of a requires $m-1$ squaring operations and $m-2$ multiplications.

6.4.1 Squaring Algorithm

The multiplication algorithm presented in the previous section can be used to square an element of $\mathrm{GF}(2^m)$. However, squaring can be performed in a more efficient way as described below.

For $a = \sum_{i=0}^{m-1} a_i \alpha^i$, where α is a root of an irreducible polynomial $f(z)$ of degree m over GF(2), let $c = a^2$ then

$$c = \sum_{k=0}^{m-1} c_k \alpha^k = \sum_{i=0}^{m-1} a_i \alpha^{2i} = \sum_{i=0}^{m-1} a_i \sum_{k=0}^{m-1} p_k^{[2i]} \alpha^k.$$

Hence

$$c_k = \sum_{i=0}^{m-1} a_i p_k^{[2i]}.$$

If $f(z)$ is an irreducible AOP, (4.4) and (6.9) result in

$$p_k^{[2i]} = \begin{cases} \delta_{2i,m} + \delta_{2i,k} & \text{even } k \\ \delta_{2i,m} + \delta_{2i,k+m+1} & \text{odd } k. \end{cases}$$

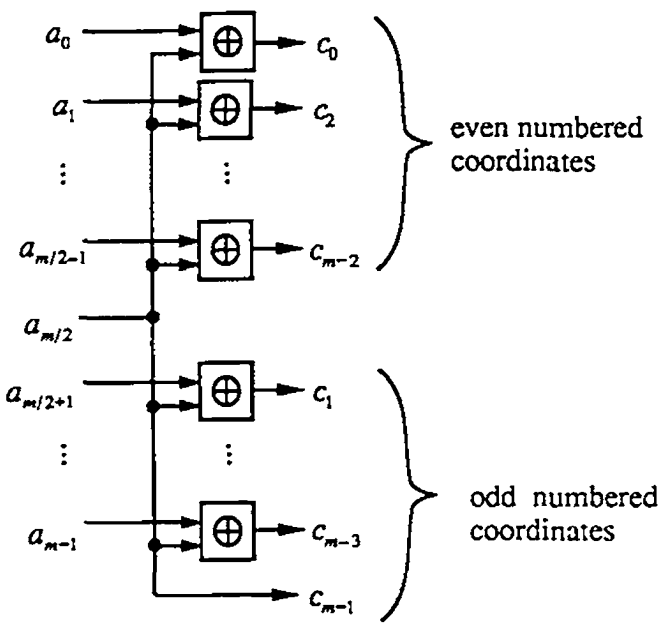


Figure 6.5: Configuration for the parallel squaring operation over GF(2^m) when the AOP of degree m is irreducible.

Consequently, $c_{m-1} = a_{\frac{m}{2}}$ and for $0 \leq k \leq m-2$,

$$c_k = \begin{cases} a_{\frac{m}{2}} + a_{\frac{k}{2}} \pmod 2, & \text{even } k \\ a_{\frac{m}{2}} + a_{\frac{k+m+1}{2}} \pmod 2, & \text{odd } k. \end{cases} \tag{6.17}$$

Figure 6.5 shows the configuration for the parallel squaring operation. It requires $m-1$ XOR gates and has a time delay of D_X.

6.4.2 Structure for Inversion

A block diagram for computing the inverse of a using (6.16) is shown in Figure 6.6. There are two m bit registers which are initially loaded with the m coordinates of elements a and $a^0 = 1$ as shown in the figure.

Referring to Figure 6.6, the time delay in the squaring loop is only D_X. However, the delay in the multiplication loop is much longer viz., $(m+\lceil \log_2(m-1) \rceil)D_X + D_A$. In this subsection, the structure for computing inversion is modified to minimize the time delay in the multiplication loop.

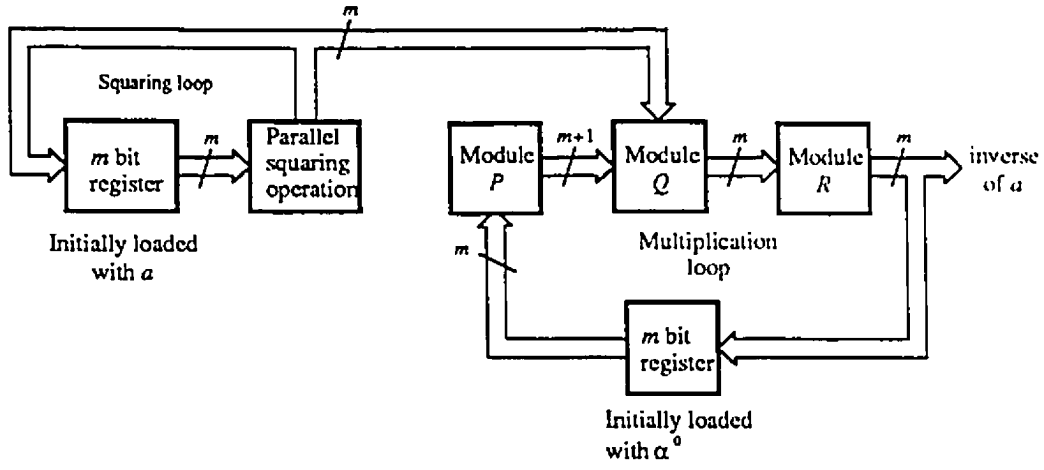


Figure 6.6: Block diagram for computing inverse.

First, rewrite (6.15) in the following matrix equation

$$\mathbf{c} = \mathbf{T}\tilde{\mathbf{c}} \tag{6.18}$$

where $\mathbf{c} = [0,\ c_0,\ c_1,\ \cdots,\ c_{m-1}]^t$, $\tilde{\mathbf{c}} = \left[\tilde{c}_0,\ \tilde{c}_1,\ \cdots,\ \tilde{c}_{m-1},\ \sum_{i=0}^{m-1} \tilde{c}_i\right]^t$ and

$$\mathbf{T} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

$\mathbf{T} \triangleq [t_{i,j}]_{i,j=0}^{m}$ is an $m+1$ by $m+1$ matrix over GF(2) where

$$t_{i,j} = \begin{cases} 1 & \text{if } i+j \le m \\ 0 & \text{otherwise.} \end{cases}$$

Similarly from (6.12),

$$\tilde{\mathbf{a}} = \tilde{\mathbf{T}}\mathbf{a} \tag{6.19}$$

where $\tilde{\mathbf{a}} = [\tilde{a}_0, \tilde{a}_1, \cdots, \tilde{a}_m]^t$, $\mathbf{a} = [0, a_0, a_1, \cdots, a_{m-1}]^t$ and

$$\tilde{\mathbf{T}} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

$\tilde{\mathbf{T}} \triangleq \left[\tilde{t}_{i,j}\right]_{i,j=0}^{m}$ is an $m+1$ by $m+1$ matrix over GF(2) where

$$\tilde{t}_{i,j} = \begin{cases} 1 & \text{if } i+j = m-1 \text{ or } m \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to show that

$$\tilde{\mathbf{T}}\mathbf{T} = \mathbf{I} \pmod 2, \tag{6.20}$$

where $\mathbf{I}$ is the $m+1$ by $m+1$ identity matrix.

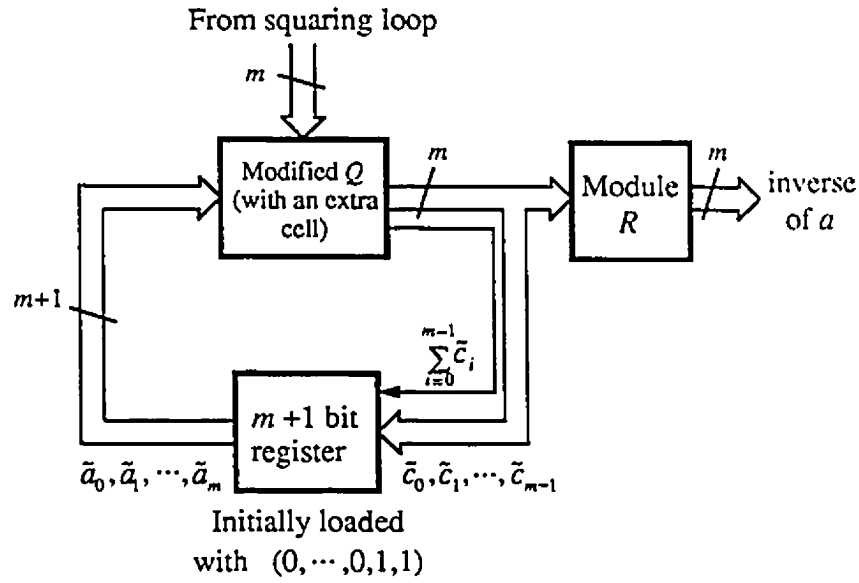
Now referring to the multiplication loop in Figure 6.6, the vectors $\mathbf{a}$, $\tilde{\mathbf{a}}$, $\tilde{\mathbf{c}}$ and $\mathbf{c}$ can be obtained from the outputs of the register and modules P, Q, R, respectively. Equating the output and the input of the register at the $(n+1)$th and the nth cycles, the result is

$$\mathbf{a}^{(n+1)} = \mathbf{c}^{(n)} = \mathbf{T}\tilde{\mathbf{c}}^{(n)}$$

where $\mathbf{x}^{(n)}$ is used to denote the vector $\mathbf{x}$ at the nth clock cycle. Using (6.19) and (6.20), the final result is

$$\tilde{\mathbf{a}}^{(n+1)} = \tilde{\mathbf{T}}\mathbf{T}\tilde{\mathbf{c}}^{(n)} = \tilde{\mathbf{c}}^{(n)}. \tag{6.21}$$

Equation (6.21) implies that the blocks for the transformation operations (modules P and R) as shown in the multiplication loop of Figure 6.6 can be bypassed. However, this requires the generation of the last element of vector $\tilde{\mathbf{c}}$ i.e., $\sum_{i=0}^{m-1} \tilde{c}_i$.



Loop delay: $\lceil \log_2(m-1) \rceil D_X + D_A$.

Figure 6.7: Block diagram of a faster multiplication loop for inverse computation.

The latter can be simply obtained using $m-1$ XOR gates to perform mod 2 additions of the outputs of module Q. The time delay in the multiplication loop is $2\lceil \log_2(m-1) \rceil D_X + D_A$. However, this delay can be approximately halved by generating the last element of $\tilde{\mathbf{c}}$ in parallel with the other elements of $\tilde{\mathbf{c}}$. As shown in Figure 6.7, instead of module Q, a modified version of module Q is used to generate all the elements in parallel. The modified Q is obtained by appending one cell (i.e., Cell m) in the original module Q (refer to Figure 6.3). The input vectors to this cell are $[b_{m-1}, \cdots, b_2, b_1, b_0]^t$ and $[\tilde{a}_{m-2}, \cdots, \tilde{a}_1, \tilde{a}_0, \tilde{a}_m]$ and the output is $\sum_{j=0}^{m-1} b_j \tilde{a}_{(j+m)}$, which is equal to $\sum_{i=0}^{m-1} \tilde{c}_i$ as shown below.

For $0 \le j \le m-2$, using (4.5) in (6.7) results in

$$\tilde{a}_{j+m} = \sum_{i=0}^{m-1} a_i p_{m-1}^{[i+j+m]} \pmod 2$$

$$= \sum_{i=0}^{m-1} a_i \left(p_{m-1}^{[i+j+m-1]} f_{m-1} + p_{m-2}^{[i+j+m-1]} \right) \pmod 2$$

$$= \tilde{a}_{j+m-1} f_{m-1} + \sum_{i=0}^{m-1} a_i p_{m-2}^{[i+j+m-1]} \pmod 2$$

$$= \tilde{a}_{j+m-1} f_{m-1} + \sum_{i=0}^{m-1} a_i \left(p_{m-1}^{[i+j+m-2]} f_{m-2} + p_{m-3}^{[i+j+m-2]} \right) \pmod 2$$

$$= \tilde{a}_{j+m-1} f_{m-1} + \tilde{a}_{j+m-2} f_{m-2} + \sum_{i=0}^{m-1} a_i p_{m-3}^{[i+j+m-2]} \pmod 2$$

$$\vdots$$

$$= \sum_{i=0}^{m-1} \tilde{a}_{j+i} f_i \pmod 2.$$

Then for an irreducible AOP,

$$\tilde{a}_{(j+m)} = \sum_{i=0}^{m-1} \tilde{a}_{(j+i)} \pmod 2, \quad 0 \le j \le m-2$$

where (x) in the subscript denotes x mod $(m+1)$. Recalling from (6.14) that $\tilde{c}_i = \sum_{j=0}^{m-1} b_j \tilde{a}_{(i+j)}$ $(i = 0, 1, \cdots, m-1)$, the final result is,

$$\sum_{i=0}^{m-1} \tilde{c}_i = \sum_{j=0}^{m-1} b_j \tilde{a}_{(j+m)} \pmod 2.$$

Since module P for the transformation of $a_0, a_1, \cdots, a_{m-1}$ to $\tilde{a}_0, \tilde{a}_1, \cdots, \tilde{a}_m$ has been eliminated from the multiplication loop in Figure 6.7, the register must be initially loaded with the appropriate vector. The m-tuple initial value of the register in the multiplication loop of Figure 6.6 is $\alpha^0 = (1, 0, 0, \cdots, 0)$. Thus the corresponding $(m+1)$ tuple vector a at the 0-th clock cycle is

$$\mathbf{a}^{(0)} = [0, 1, 0, \cdots, 0, 0]^t.$$

Inverter description	Number of Registers	Number of XOR gates	Number of AND gates	Time delays due to gates
Wang *et al.* [43]	$2m$	$2m^2 - 2m$	m^2	$(m-2)(D_A + (1+\lceil \log_2(m-1) \rceil) D_X)$
Using ITM [21]	$2(m+1)$	$m^2 + 3m$	$m^2 + 2m + 1$	$(m-2)(D_A + \lceil \log_2 m + \log_2(m+2) \rceil D_X)$
Presented here	$2m+1$	$m^2 + 2m - 3$	m^2	$(m-2)(D_A + (1+\lceil \log_2(m-1) \rceil) D_X) + D_X$

Table 6.2: Comparison of number of gates and time delays for three inverters.

Substituting $\mathbf{a}^{(0)}$ in (6.19), we obtain

$$\tilde{\mathbf{a}}^{(0)} = [0,\ 0,\ \cdots,\ 0,\ 1,\ 1]^t \tag{6.22}$$

which is the desired vector to be initially loaded into the $m+1$ bit register of the multiplication loop in Figure 6.7.

Table 6.2 gives a comparison of the inverter presented here with the inverter of [43] and the one which can be obtained using ITM [21]. These inverters are based on (6.16) and each requires $m-2$ multiplications. The number of multiplications can be reduced using the algorithms proposed by Itoh [19].

6.5 Proposed ESP Based Parallel Multiplier

Recently, irreducible ESPs have been given special attention in the literature [21], [20]. ESPs of arbitrarily high degree can readily be obtained from a *corresponding* irreducible AOP of a very small degree. Moreover, Itoh and Tsujii have shown that parallel multipliers based on irreducible ESPs have structural modularity [21]. As a result, irreducible ESPs are of practical importance. In this section, ESP based parallel multipliers are presented which have structural modularity as well as low circuit complexity.

6.5.1 Condition for an Irreducible ESP

In order to derive a condition for an ESP to be irreducible, the following theorem is useful.

Theorem 6.1 [45] Let $f(z)$ be an irreducible polynomial of degree m over GF(q). Let k be a prime, $k|(q^m - 1)$. Then $g(z) = f(z^{k^i})$, $i = 1, 2, \cdots$ is irreducible over GF(q) iff

$$\alpha^{(q^m-1)/k} \neq 1 \tag{6.23}$$

where $\alpha \in \mathrm{GF}(q^m)$ satisfies $f(\alpha) = 0$.

Corollary 6.1 follows immediately.

Corollary 6.1 [15] Let $f(z)$ be an irreducible AOP of degree m over GF(2). Then $g(z) = f(z^{(m+1)^i})$, $(i = 1, 2, \cdots)$ is irreducible over GF(2) iff $2^m \neq 1 \pmod{(m+1)^2}$.

Proof: Since $f(z)$ is an irreducible AOP, $m + 1$ is prime and the order of α is $m + 1$. So (6.23) implies that

$$\frac{2^m - 1}{m+1} \neq m + 1,$$

or,

$$2^m \neq 1 \pmod{(m+1)^2}.$$

A result similar to Corollary 6.1 has been found independently by Itoh [20].

6.5.2 Structure

We now apply the algorithm described in Section 6.3 to ESP based parallel multipliers. The following discussion is similar to that for the AOP based parallel multipliers. In this subsection, reference to an AOP and an ESP implies an irreducible AOP of degree m and an irreducible s-ESP of degree n ($n = ms = m(m+1)^i$), respectively.

Theorem 6.2 [15] If β satisfies $g(\beta) = 0$, then the order of β is $(m+1)^{i+1}$, where $g(z)$ is an $(m+1)^i$-ESP as specified in Corollary 6.1.

Proof: Since

$$\beta^{m(m+1)^i} + \beta^{(m-1)(m+1)^i} + \cdots + \beta^{(m+1)^i} + 1 = 0, \tag{6.24}$$

multiplying both sides of (6.24) by $\beta^{(m+1)^i}$ and then adding the result to (6.24) we obtain $\beta^{(m+1)^{i+1}} = 1$.

Suppose there exists a k such that $m(m+1)^i < k \leq (m+1)^{i+1}$ and $\beta^k = 1$. Then k must divide $(m+1)^{i+1}$. Since $(m+1)$ is prime, $k = (m+1)^{i+1}$ which completes the proof.

Using Theorem 6.2, we have for any integer j,

$$p_i^{[j]} = p_i^{[j \bmod r]} \quad i = 0,\ 1,\ \cdots,\ n-1, \tag{6.25}$$

where $r = (m+1)^{i+1}$ is the order of β. Corresponding to (6.11) for an AOP, we have the following equation for an ESP

$$p_{n-1}^{[k]} = \delta_{k,n-1} + \delta_{k,r-1} \quad 0 \leq k \leq r-1. \tag{6.26}$$

Equation (6.25) implies that there are only r distinct $\tilde{a}_k$ in (6.6) and they are obtained by using (6.25) and (6.26) in (6.7) as follows.

$$\tilde{a}_i = \begin{cases} a_{n-1-i} & 0 \leq i \leq s-1 \\ a_{n-1-i} + a_{r-1-i} \pmod 2 & s \leq i \leq n-1 \\ a_{r-1-i} & n \leq i \leq r-1. \end{cases} \tag{6.27}$$

Equations (6.6) and (6.8) can now be written as

$$\begin{bmatrix} \tilde{a}_{n-1} & \tilde{a}_{n-2} & \tilde{a}_{n-3} & \cdots & \tilde{a}_1 & \tilde{a}_0 \\ \tilde{a}_n & \tilde{a}_{n-1} & \tilde{a}_{n-2} & \cdots & \tilde{a}_2 & \tilde{a}_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{a}_{r-1} & \tilde{a}_{r-2} & \tilde{a}_{r-3} & \cdots & \tilde{a}_{s+1} & \tilde{a}_s \\ \tilde{a}_0 & \tilde{a}_{r-1} & \tilde{a}_{r-2} & \cdots & \tilde{a}_{s+2} & \tilde{a}_{s+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{a}_{n-s-2} & \tilde{a}_{n-s-3} & \tilde{a}_{n-s-4} & \cdots & \tilde{a}_n & \tilde{a}_{n-1} \end{bmatrix} \begin{bmatrix} b_{n-1} \\ b_{n-2} \\ b_{n-3} \\ b_{n-4} \\ \vdots \\ b_0 \end{bmatrix} = \begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \\ \tilde{c}_3 \\ \vdots \\ \tilde{c}_{n-1} \end{bmatrix} \tag{6.28}$$

or, equivalently

$$\tilde{c}_i = \sum_{j=0}^{n-1} b_j \tilde{a}_{i+j \pmod r} \pmod 2 \quad i = 0,\ 1,\ \cdots,\ n-1 \tag{6.29}$$

and

$$\begin{aligned} c_{n-1-i} &= \sum_{l=0}^{i} \tilde{c}_{i-l}\ g_{n-l} \pmod 2 \\ &= \sum_{l=0}^{i} \tilde{c}_{i-l} \left(\sum_{k=0}^{m} \delta_{n-l,sk} \right) \pmod 2 \end{aligned}$$

where (6.1) has been used to substitute the coefficients of the ESP. The quantity inside the parenthesis is nonzero when $l = js$ ($j = m - k$). So

$$\begin{aligned} c_{n-1-i} &= \sum_{j=0}^{\lfloor i/s \rfloor} \tilde{c}_{i-js} \pmod 2 \\ &= \begin{cases} \tilde{c}_i & 0 \le i \le s-1 \\ \tilde{c}_i + \tilde{c}_{i-s} \pmod 2 & s \le i \le 2s-1 \\ \vdots & \\ \tilde{c}_i + \tilde{c}_{i-s(m-1)} + \cdots + \tilde{c}_{i-s} \pmod 2 & s(m-1) \le i \le n-1 \end{cases} \\ &= \begin{cases} \tilde{c}_i & 0 \le i \le s-1 \\ \tilde{c}_i + c_{n-1-i-s} \pmod 2 & s \le i \le n-1. \end{cases} \end{aligned} \tag{6.30}$$

The structure of the s-ESP based parallel multiplier is then a straightforward realization of (6.27), (6.29) and (6.30). However, it is always advantageous to have a modular structure, especially for VLSI implementation of a parallel multiplier with a large value of n. Given the three basic modules (P, Q and R) of the AOP based parallel multiplier, if a corresponding ESP based parallel multiplier could be designed using the same three modules, it would reduce the design time.

In order to have a regular expansion of the three basic modules, modify (6.27), (6.29) and (6.30) for $0 \leq i \leq s-1$ as follows.

$$\tilde{a}_{i+sk} = \begin{cases} a_{n-1-i} & k = 0 \\ a_{n-1-i-sk} + a_{r-1-i-sk} \pmod 2 & k = 1,\ 2,\ \cdots,\ m-1 \\ a_{s-1-i} & k = m, \end{cases} \tag{6.31}$$

$$\begin{aligned} \tilde{c}_{i+sk} &= \sum_{j=0}^{s-1} \sum_{l=0}^{m-1} b_{j+sl}\ \tilde{a}_{i+j+sl+sk \pmod r} \pmod 2 \quad k = 0,\ 1,\ \cdots,\ m-1 \\ &= \sum_{j=0}^{s-1} \tilde{c}_{i,j}^{(k)} \pmod 2 \quad k = 0,\ 1,\ \cdots,\ m-1 \end{aligned} \tag{6.32}$$

where

$$\tilde{c}_{i,j}^{(k)} \equiv \sum_{l=0}^{m-1} b_{j+sl}\ \tilde{a}_{i+j+sl+sk \pmod r} \pmod 2 \quad k = 0,\ 1,\ \cdots,\ m-1, \tag{6.33}$$

$$c_{n-1-i-sk} = \begin{cases} \tilde{c}_i & k = 0 \\ \tilde{c}_{i+sk} + c_{n+s-1-i-sk} \pmod 2 & k = 1,\ 2,\ \cdots,\ m-1. \end{cases} \tag{6.34}$$

The similarities of (6.31), (6.33) and (6.34) with (6.12), (6.14) and (6.15), respectively, imply that an ESP based parallel multiplier can be constructed with modules P, Q and R. These are the same modules used for the construction of the corresponding AOP based parallel multiplier. For an ESP based multiplier, block B1 (refer to Figure 6.1) contains s copies of module P (P_i, $i = 0,\ 1,\ \cdots,\ s-1$). In

addition to s^2 copies of module Q ($Q_{i,j}$, $i,j = 0, 1, \cdots, s-1$), B2 contains $n(s-1)$ XOR gates. Block B3 contains s number of R type module (R_i, $i = 0, 1, \cdots, s-1$). Modules P_i, $Q_{i,j}$ and R_i realize $\tilde{a}_{i+sk}$ ($k = 0, 1, \cdots, m$), $\tilde{c}_{i,j}^{(k)}$ ($k = 0, 1, \cdots, m-1$) and $c_{n-1-i-sk}$ ($k = 0, 1, \cdots, m-1$) respectively. As a simple illustration, the parallel multiplier based on the irreducible 3-ESP $z^6 + z^3 + 1$ corresponding to the irreducible AOP $z^2 + z + 1$ is shown in Figure 6.8. Note that 3^2, 3^3, $\cdots$-ESP based parallel multipliers can be constructed using the same three modules.

6.5.3 Complexity and Comparison

In general, the proposed ESP based parallel multiplier which has modular structure requires $n^2 + n - 2s$ XOR gates and n^2 AND gates. When compared to the ITM multiplier based on the same ESP, a reduction of $s^2 + s + (2s-1)n$ XOR gates and $s^2 + 2ns$ AND gates is obtained by using the proposed scheme.

The time delay in the structure as a stand-alone parallel multiplier is about $(m + \log_2 n)D_X + D_A$. When the multiplier is to be used for computing inverses or exponentiations, the delay in the multiplication loop can be reduced to about $(\log_2 n)D_X + D_A$ as has been shown for the AOP case.

For a highly composite number n, an algorithm for parallel multiplication has been presented by Pincin where the existence of intermediate fields between GF(2^n) and GF(2) has been exploited [33]. The parallel multiplication can be realized with a time complexity of $O(\log n)$. However, the circuit complexity is relatively high. For example, when $n = m(m+1)^i$, the number of AND gates is $((m-1)^3 + m^2)(m^3 + (m+1)^2)^i$ [33]. The corresponding ESP based parallel multiplier presented here requires only $m^2(m+1)^{2i}$ AND gates; and the time delay of the ESP based multiplier can be reduced to $O(\log n)$ for computing inversion (or exponentiation) as described in the previous paragraph. Moreover, the ESP based multiplier provides a modular structure and a relatively lower communica-

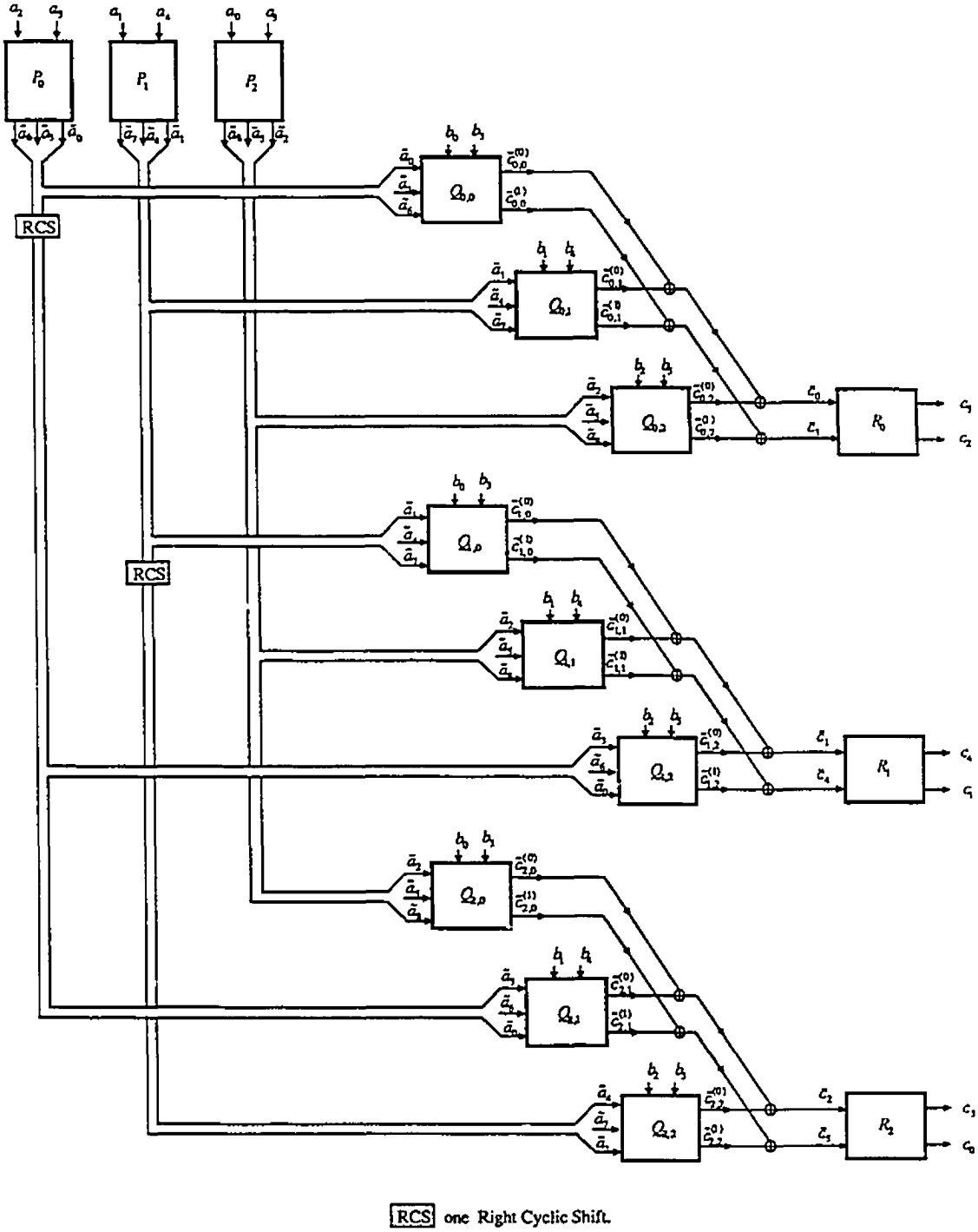


Figure 6.8: Structure for the 3-ESP based parallel multiplier.

tion complexity which are important for large finite fields.

It has been mentioned earlier that by choosing a suitable irreducible polynomial, the number of distinct entries in the Toeplitz matrix of (6.6) can be reduced. When the multiplication algorithm given in Section 6.3 is used, a reduction in the number of distinct entries does not always result in lower complexity parallel multiplier. There are certain n ($n = m(m+1)^i$) for which an irreducible AOP as well as an irreducible ESP of degree n exist. For the irreducible AOP of degree n, there are only $n+1$ distinct entries; and the AOP based parallel multiplier requires n^2 AND gates and n^2+n-2 XOR gates. On the other hand, for the irreducible ESP of degree n, there are $n+s$, ($s = (m+1)^i$), distinct entries and the ESP based parallel multiplier requires n^2 AND gates and n^2+n-2s XOR gates giving a reduction of $2(s-1)$ XOR gates over its AOP based counterpart.

An AOP of degree n is irreducible if and only if $n+1$ is prime and 2 is primitive mod $(n+1)$ [41]. Combining the conditions of the irreducibility of AOP and ESP, we can find n for which both irreducible AOP and ESP exist. Examples of such n are 18, 100, 162 and the corresponding spacings for the irreducible ESPs are 9, 25 and 81 respectively.

Thus it may be concluded that if, for a given n, an irreducible AOP as well as an irreducible ESP exist, from the complexity point of view, it is advantageous to use the ESP based parallel multiplier. Moreover, this particular ESP based multiplier can be constructed from the three basic modules of the parallel multiplier based on the irreducible AOP of degree m.

6.6 Conclusions

In this chapter, structures for parallel multiplication based on irreducible AOP and ESP have been developed. The structures are simple and modular which is important for VLSI design. The circuit complexity of the proposed parallel multipliers is less than that of the parallel multipliers of [43] or [21] of the same

classes of GF(2^m), and the reduction is significant for ESP based parallel multipliers. Furthermore, when repetitive multiplications are required, the structures can be modified to enable an implementation of fast multiplications and it therefore makes fast exponentiation and inversion possible. It has been shown that the three basic modules of an AOP based parallel multiplier of a small field can be used to construct all the corresponding ESP based parallel multipliers of larger fields. This expansion is regular and provides a convenient way to design parallel multipliers for very large finite fields. It has also been shown that if, for certain degrees, both irreducible AOP and irreducible ESP exist, it is advantageous to use the ESP based parallel multipliers.

Chapter 7

An Architecture for A Low Complexity Rate-Adaptive Reed-Solomon Encoder

7.1 Introduction

A Reed-Solomon (RS) code is a multiple-error-correcting code. The multiple-error-correcting capability of RS codes has been used in many practical applications. Examples of important practical applications include space communications, mobile communications, magnetic and optical recording systems.

An RS encoder which has a low circuit complexity has been developed by Hsu, *et al.* [18] using Berlekamp's bit serial multiplication algorithm [6]. However, the error correcting capability of their encoder is fixed. Moreover, the coefficients of the generator polynomial are represented with respect to a canonical basis, whereas the input data symbols as well as the output codeword symbols are represented with respect to the corresponding dual basis. When the encoder is to be part of a larger device, it is desirable that all terms be represented with respect to a common basis.

In this chapter, an RS encoder which overcomes the above two problems is presented [13]. The bit-serial multiplication algorithm developed in Chapter 5 is used to obtain the encoder.

Section 7.2 describes the encoding algorithm. Section 7.3 presents a so-called triangular basis and applies the multiplication algorithm of Chapter 5 to obtain a pipelined bit-serial constant multiplier. Section 7.4 describes a structure for a fixed-rate RS encoder using the multiplier. An efficient RS encoder with a variable code rate is presented in Section 7.5. Finally concluding remarks are made in Section 7.6.

7.2 Encoding Algorithm

An $(n,\ k)$ RS code with symbols from a finite field F can correct a maximum of $t = \lfloor (n-k)/2 \rfloor$ symbols in error. In our discussion, we consider $F = \mathrm{GF}(2^m)$, which is the most widely used finite field. Then the length of a codeword is $n = 2^m - 1$ symbols and each symbol can be represented by m binary digits. A codeword consists of k data symbols and $n - k = 2t$ parity check symbols. The *code rate* or *information rate* of the code is defined as k/n. This code has a minimum distance of $n - k + 1 = 2t + 1$ symbols. The generator polynomial $g(x)$ is defined as

$$g(x) \triangleq \sum_{i=0}^{2t} g_i x^i = \prod_{i=0}^{2t-1} \left(x - \gamma^{h+i}\right), \tag{7.1}$$

where γ is a primitive nth root of unity in $\mathrm{GF}(2^m)$ and h is an integer constant. Note that when the number of redundant symbols is decreased to obtain a higher code rate, the coefficients g_i of the generator polynomial change.

Let the sequence of k data symbols in $\mathrm{GF}(2^m)$ be $\{d_o,\ d_1,\ \cdots,\ d_{k-1}\}$. The data sequence can be represented by a polynomial as follows:

$$d(x) = d_0 + d_1 x + \cdots + d_{k-1} x^{k-1}. \tag{7.2}$$

Let the corresponding sequence of n codeword symbols be $\{c_0,\ c_1,\ \cdots,\ c_{n-1}\}$. The polynomial representation of the codeword is

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}, \quad c_i \in \mathrm{GF}(2^m). \tag{7.3}$$

In systematic form, a codeword is obtained by adding $n-k$ parity check symbols to the data symbols. Denote these symbols by $\{p_0, p_1, \cdots, p_{2t-1}\}$. Then

$$\{c_0, c_1, \cdots, c_{n-1}\} = \{p_0, p_1, \cdots, p_{2t-1}, d_0, d_1, \cdots, d_{k-1}\}.$$

In polynomial notation, we can write

$$c(x) = p(x) + x^{n-k}d(x), \tag{7.4}$$

where $p(x) = \sum_{i=0}^{2t-1} p_i x^i$ is a polynomial over GF(2^m) of degree $2t-1$ or less. The parity symbols p_i are chosen such that $c(x)$ is divisible by the generator polynomial $g(x)$, i.e.,

$$p(x) = R_{g(x)}\left[x^{n-k}d(x)\right], \tag{7.5}$$

where $R_{g(x)}[\cdot]$ denotes the remainder after division by $g(x)$.

Thus the encoding of the RS code in systematic form consists of the following three steps.

E1. Pre-multiply the data polynomial $d(x)$ by x^{n-k}.

E2. Obtain $p(x)$ as defined in (7.5).

E3. Combine $p(x)$ and $x^{n-k}d(x)$ to obtain the codeword $c(x)$ as indicated in (7.4).

The second step of the encoding algorithm requires multiplications in the field GF(2^m). The complexity of the RS encoder depends mainly on the associated multiplication circuitry. A standard architecture for an RS encoder is shown in Figure 7.1 where parallel-type multipliers are used for step E2. A parallel-type multiplier may take $O(m^2)$ two-input AND gates and XOR gates. However, the circuit complexity of a bit-serial multiplier is only $O(m)$ [6], [44].

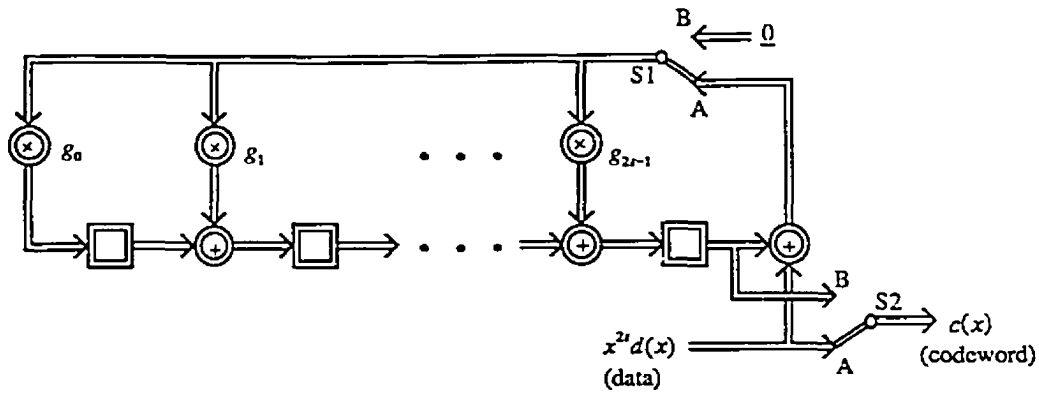


Figure 7.1: RS encoder with parallel multipliers of GF(2^m).

7.3 A Pipelined Bit-Serial Constant Multiplier

In this section, a particular set of independent elements are chosen to form a basis (a *triangular basis*). The involvement of the triangular basis in the multiplication algorithm of Chapter 5 is demonstrated. Then a structure for a pipelined bit-serial constant multiplier for GF(2^m) is presented.

7.3.1 Triangular Basis

Wang and Blake have used a transformation matrix of the following form [44]

$$\begin{bmatrix} f_1 & f_2 & \cdots & f_{m-1} & 1 \\ f_2 & f_3 & \cdots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix},$$

where f_i are the coefficients of the irreducible polynomial $f(x) = \sum_{i=0}^{m} f_i x^i$ defining the field GF(2^m). This leads to the following theorem.

Theorem 7.1 Let $\beta_j = \sum_{i=0}^{m-1-j} f_{i+j+1}\alpha^i$ where $f(x) = \sum_{i=0}^{m} f_i x^i$ is an irreducible polynomial of degree m over GF(2) and $f(\alpha) = 0$. Then $\{\beta_0, \beta_1, \cdots, \beta_{m-1}\}$ forms a basis of the field GF(2^m).

Proof: The elements $1, \alpha, \alpha^2, \cdots, \alpha^{m-1}$ are linearly independent and

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{m-1} \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & \cdots & f_{m-1} & 1 \\ f_2 & f_3 & \cdots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ \alpha \\ \vdots \\ \alpha^{m-1} \end{bmatrix} \tag{7.6}$$

where the $m \times m$ matrix is non-singular. Thus $\beta_0, \beta_1, \cdots, \beta_{m-1}$ are linearly independent and so form a basis. Q. E. D.

The basis $\{\beta_0, \beta_1, \cdots, \beta_{m-1}\}$ is referred to as a triangular basis. Any element $a \in$ GF(2^m) can be represented as

$$a = \sum_{i=0}^{m-1} \tilde{a}_i \beta_i, \tag{7.7}$$

where $\tilde{a}_i \in$ GF(2) for $0 \leq i \leq m-1$ are the triangular basis coordinates of a. The transformation of coordinates of a from the canonical basis to the triangular basis is given below.

$$\tilde{a}_i = a_{m-1-i} + (1 - \delta_{i,0}) \sum_{l=1}^{i} \tilde{a}_{i-l} f_{m-l} \pmod 2 \qquad 0 \leq i \leq m-1, \tag{7.8}$$

and from (7.8)

$$a_{m-1-i} = \sum_{l=0}^{i} \tilde{a}_{i-l} f_{m-l} \pmod 2 \; 0 \leq i \leq m-1. \tag{7.9}$$

Equation (7.9) correspond to transformations from the triangular basis to canonical basis. The involvement of the triangular basis in the multiplication algorithm of Chapter 5 is seen by comparing (7.8) and (7.9) with (3.17) and (3.18). Thus the basis transformations can be accomplished with the shift register configurations of Figures 5.4 and 5.5. These configurations are referred to here as recursive and non-recursive filters, respectively.

7.3.2 Multiplier Structure

The bit-serial multiplier (refer to Figure 5.6) does not provide a pipelined operation. To develop a structure for a pipelined bit-serial multiplier based on (3.16), (3.17) and (3.18) the multiplication algorithm is described as follows.

M1. Use Equation (7.8) to transform a_i to $\tilde{a}_i$ for $0 \leq i \leq m-1$.

M2. Generate the rows of the Toeplitz matrix in Equation (6.6).

M3. Perform the matrix-vector multiplication described in Equation (6.6) to obtain $\tilde{c}_i$ for $0 \leq i \leq m-1$.

M4. Use Equation (7.9) to transform $\tilde{c}_i$ to c_i for $0 \leq i \leq m-1$.

A pipelined bit-serial multiplier for a constant value of b using the above algorithm is shown in Figure 7.2 where the coordinate a_{m-1} enters the multiplier first (say, at the 0th clock cycle). As the other canonical basis coordinates of a enter the recursive filter, they are transformed to the triangular basis coordinates. At the mth clock cycle, the 0th row of the Toeplitz matrix is loaded into the LFSR and both the recursive and non-recursive filter registers are set to zero. During the clock cycles m to $2m-1$, the canonical basis coordinates of the product are obtained from the non-recursive filter and the coordinates of the multiplicand a for the next problem can enter the recursive filter, giving a pipelined operation. Thus the total computation time for a multiplication in GF(2^m) is $2m$ cycles with the

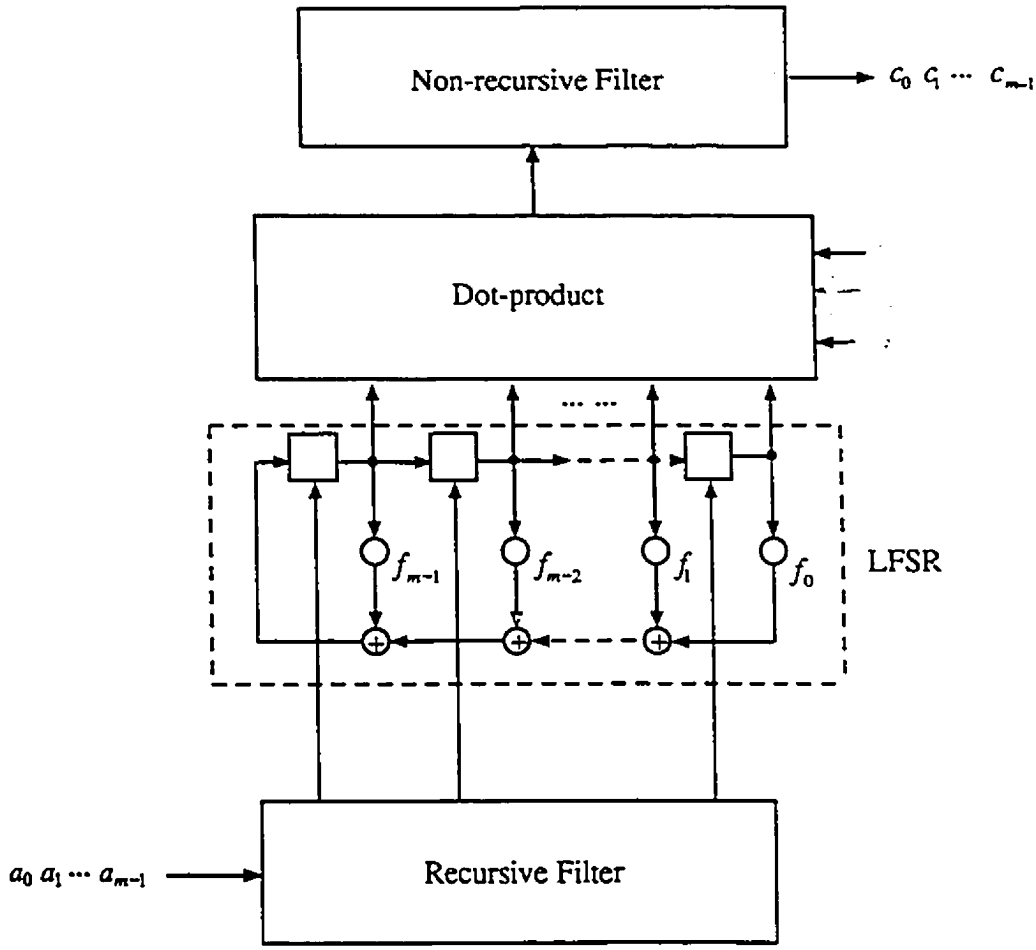


Figure 7.2: A pipelined bit-serial constant multiplier for GF(2^m).

first m cycles used for basis transformation. The computational delay is m clock cycles.

7.4 A Fixed-Rate RS Encoder

7.4.1 Structure

A structure for an RS encoder using the pipelined bit-serial constant multiplier is shown in Figure 7.3. The structure is divided into four main units.

1. Basis Transformation Unit: This consists of one recursive and one non-recursive filter. The recursive filter transforms the coordinates of the input data

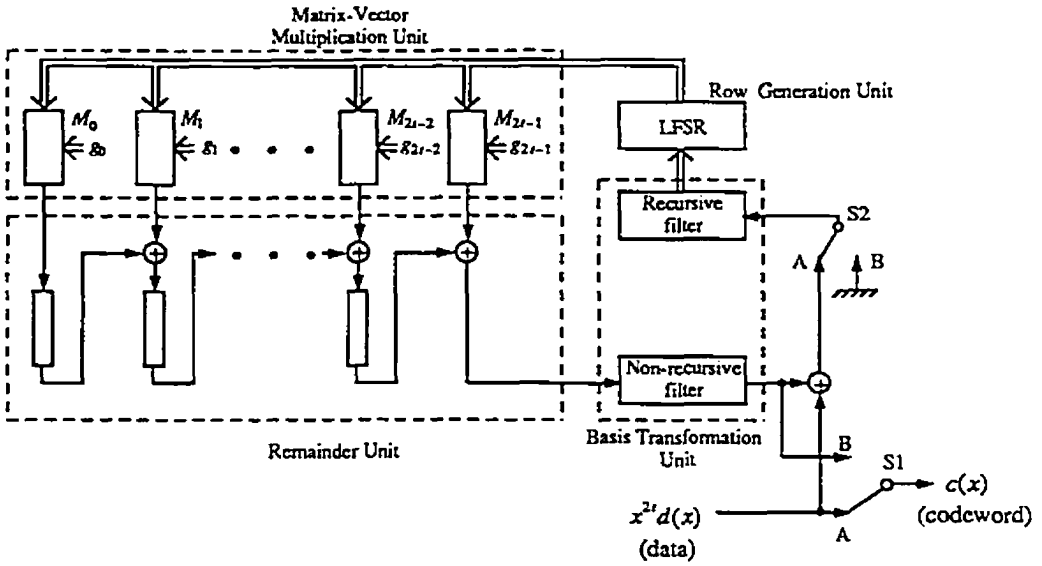


Figure 7.3: A fixed-rate RS encoder structure using a pipelined bit-serial constant multiplier.

symbols from the canonical basis to the triangular basis. The non-recursive filter transforms each outgoing codeword symbol back from the triangular basis to the canonical basis.

2. Row Generation Unit: This is simply an LFSR whose feedback connections are determined by the primitive polynomial $f(x)$. If the LFSR is initially loaded with row 0 of the Toeplitz matrix, then at the successive $m-1$ clock cycles, the contents of the LFSR are rows 1 to $m-1$, respectively.

3. Matrix-Vector Multiplication Unit: This unit consists of $2t$ identical modules $M_0, M_1, \cdots, M_{2t-1}$ each of which performs a dot product of two input vectors. The common input to all the modules is a row of the Toeplitz matrix. The other input to module M_i is the i-th coefficient g_i of the generator polynomial. For a fixed-rate encoder, the coefficients can be hardwired to the modules. The module outputs are the coordinates of the dot product with respect to the triangular basis.

4. Remainder Unit: Stores the coefficients of the remainder polynomial. The remainder coefficients are elements of GF(2^m) and are represented relative to the triangular basis. Modulo-2 additions are performed bit-by-bit in the triangular

basis.

In addition to these four units, the RS encoder requires circuitry to generate the necessary control signals. For example, this circuitry must generate a signal to load the LFSR and reset the registers of the two filters every m clock pulses. Another control signal is necessary to realize the operation of the two switches.

The data sequence $\{d_0, d_1, \cdots, d_{k-1}\}$ is shifted bit-by-bit into the encoder circuit and simultaneously into the communication channel with switches S1 and S2 at position A. Shifting the data into the circuit from the right hand side is equivalent to pre-multiplying $d(x)$ by x^{n-k}. As soon as the complete data sequence has entered the encoder, both switches are placed in position B. Over the next $m(n-k)$ clock cycles, the check symbols are transmitted into the channel.

7.4.2 Complexity and Comparison

As a measure of circuit complexity of the encoder, the number of registers and modulo-2 adders is considered. These adders can be realized by XOR gates.

The complexity of the Basis Transformation and the Row Generation units depends on the primitive polynomial $f(x)$ defining the field GF(2^m). If $f(x) = x^m + x^{k_n} + x^{k_{n-1}} + \cdots + x^{k_0} + 1$, where $m > k_n > k_{n-1} > \cdots > k_0 > 0$ and the Hamming weight of $f(x)$ is $W_H(f)$, then the Basis Transformation unit contains $2m - k_0 - 1$ registers and $2\{W_H(f) - 2\}$ adders.

In addition to $n-k-1$ length m registers, the Remainder unit contains modulo-2 adders. Let R_{ADD} denote the number adders required in the Remainder unit. It is obvious from Figure 7.3 that $R_{\text{ADD}} \leq n-k-1$. Any reduction in the number of adders is obtained only by choosing $g(x)$ such that it has one or more coefficients (expect g_0 and g_{n-k}) equal to 0. However, since $g(x)$ must itself be a codeword, it has Hamming weight $\geq n-k+1$. Hence all the coefficients $g_0, g_1, \cdots, g_{n-k}$ must be nonzero. Consequently, we must have $R_{\text{ADD}} = n-k-1$.

There are $2t$ identical modules in the Matrix-Vector Multiplication unit and

each module, in general, contains m modulo-2 multiplications and $m-1$ additions. For a fixed-rate code, the generator polynomial coefficients g_i are constants and can be hardwired, resulting in only $W_H(g_i)-1$ adders in module M_i $(0 \leq i \leq 2t-1)$ at the expense of similarity with other modules. A further reduction in the circuit complexity can be obtained by suitably choosing h and γ in (7.1). If $h = 2^{m-1} - t$, then the generator polynomial becomes

$$g(x) = x^{n-k} g(x^{-1})$$

i.e., the coefficients of the generator polynomial are symmetric [18]. As a result, for a fixed-rate encoder the total number of modulo-2 adders in the Matrix-Vector multiplication unit is $\sum_{i=0}^{t} (W_H(g_i) - 1)$.

A comparison of the fixed-rate encoder presented here with two other encoders [18], [37] is given below. The basis of comparison is circuit complexity in terms of number of registers, modulo-2 adders.

The VLSI architecture for the fixed-rate RS encoder developed by Hsu, *et al.* [18] has three main units viz., Remainder unit, Product unit and Quotient unit. The Remainder units of [18] and this paper have the same circuit complexity. The Product unit of [18] is equivalent to the Matrix-Vector Multiplication unit and the feedback circuitry of the Row Generation unit presented here. Since the complexity of the Product unit is not given in [18], we first determine its complexity.

We note that the Product unit generates the terms T_i $(0 \leq i \leq t)$ which are given as follows [18]:

$$\begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_t \end{bmatrix} = \begin{bmatrix} \mathrm{Tr}(zg_0) \\ \mathrm{Tr}(zg_1) \\ \vdots \\ \mathrm{Tr}(zg_t) \end{bmatrix} = \mathbf{P} \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{m-1} \end{bmatrix}, \tag{7.10}$$

where $\mathbf{P}$ is a binary transformation matrix, z_i $(0 \leq i \leq m-1)$ are the coordinates

of the input multiplicand with respect to the dual basis, and Tr is the trace function defined as $\mathrm{Tr}(x) = \sum_{i=0}^{m-1} x^{2^i}$. From (7.10),

$$T_k = \mathrm{Tr}(zg_k) = \mathrm{Tr}\left(z \sum_{i=0}^{m-1} g_{k,i}\alpha^i\right) \quad k = 0,\ 1,\ \cdots,\ t$$

where $g_{k,i}$ for $0 \leq i \leq m-1$ are the coordinates of the generator polynomial coefficient g_k with respect to the canonical basis. Using the properties of Tr we obtain

$$\begin{aligned} T_k &= \sum_{i=0}^{m-1} g_{k,i}\mathrm{Tr}(z\alpha^i) \\ &= \sum_{i=0}^{m-1} g_{k,i}z_i \\ &= \begin{bmatrix} g_{k,0} & g_{k,1} & \cdots & g_{k,m-1} \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{m-1} \end{bmatrix}. \end{aligned} \tag{7.11}$$

From (7.10) and (7.11), we finally have

$$\mathbf{P} = [g_{i,j}]_{i=0,j=0}^{i=t,j=m-1}, \tag{7.12}$$

i.e, the elements of the ith row of the binary transformation matrix P are the coordinates of g_i with respect to the canonical basis. Thus, the number of modulo-2 adders in the Product unit of [18] required to generate the terms $T_0,\ T_1,\ \cdots,\ T_t$ is $\sum_{i=0}^{t} (W_H(g_i) - 1)$, which is the same as that of the Matrix-Vector Multiplication unit presented here.

When compared to the encoder of [18], the encoder presented here requires an excess of $2\,(W_H(f) - 2)$ adders and $m - k_0$ registers where k_0 is given in $f(x) =$

$x^m + x^{k_n} + x^{k_{n-1}} + \cdots + x^{k_0} + 1$. This excess is due to the non-recursive filter and the feedback connection of the recursive filter in the Basis Transformation unit. The advantage of the encoder presented here is that all its terms (data, codeword and coefficients of the generating polynomial) are represented in terms of a common basis. It is thus more suitable for use in a larger device. It also avoids any time delay at the encoder input and output which arises due to basis transformation when the encoder of [18] used as a part of a larger device.

Recently, another RS encoder, which has a systolic structure and does not have any feedback path, has been presented by Seroussi [37]. The encoder consists of $2t + 1$ cells and each cell performs a parallel division operation in the finite field. The circuit complexity of the encoder using a parallel type divider in each cell would be as high as $O(m^3 t)$. From this point of view, the encoder presented here is more efficient. Furthermore, the encoder has a much shorter longest logic path through which a signal must pass in one clock cycle. This feature makes it possible to use a comparatively higher clock rate for the encoder.

7.5 A Rate-Adaptive RS Encoder

The encoder presented in the previous section provides a fixed code rate; in other words, the number of redundant symbols is fixed. Usually, a designer considers the worst possible channel conditions to determine the number of maximum redundant symbols, say $r_{\max}$ $(= 2t)$. However, in many applications the channel remains in its worst state for only a small fraction of the total time of use. As a result, it is desirable to make the number of redundant symbols variable so that a higher code rate is achieved during times when the channel is not in its worst state. In the following discussion, the issue of how the system detects the need for a rate change is not addressed, but an encoder architecture for which such a rate change can be efficiently implemented is presented.

As the number of redundant symbols varies from one code word to another,

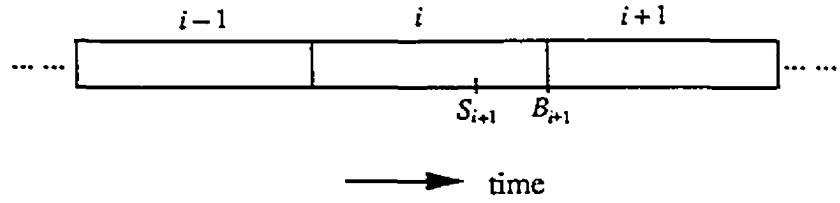


Figure 7.4: Timing sequence of operations of the rate-adaptive RS encoder.

the corresponding generator polynomial (GP) also changes. There are altogether $r_{\max}+1$ possible GPs. A straightforward approach to generate coefficients of all these GPs is to store them in ROMs and use the appropriate one according to the number of redundant symbols. This, however, requires a total memory size of $O(mr_{\max}^2)$ bits, distributed among $r_{\max}+1$ ROMs, each associated with one of the modules in the Matrix-Vector Multiplication unit of Figure 7.3. For large values of $r_{\max}$, the use of so many ROMs may not be advantageous. In the next subsection, we present a structure capable of generating all the GPs with a storage requirement of $O(mr_{\max})$ bits and $O(W_H(f))$ gates.

7.5.1 Recursive Generation of GPs

The timing sequence associated with GP generation and codeword transmission is depicted in Figure 7.4.

Here block i corresponds to the transmission of the codeword $\mathbf{c}_i$. (To distinguish from the previous notations, the time dependent codeword, data and GP are represented in bold face letters, and one index is used to specify the time slot to which they belong.) B_{i+1} denotes the beginning of the transmission of $\mathbf{c}_{i+1}$. To generate the next codeword $\mathbf{c}_{i+1}$, the corresponding GP $\mathbf{g}_{i+1}(x)$ must be made available before the time instant B_{i+1}. The generation of $\mathbf{g}_{i+1}(x)$ starts at S_{i+1}. Information about any increase or decrease (say Δr_i) in the redundancy of $\mathbf{c}_{i+1}$ relative to that of $\mathbf{c}_i$ must be made available before the time instant S_{i+1}. The time delay between S_{i+1} and B_{i+1} is determined by $r_{\max}$ and the maximum possible

value of Δr_i.

We initially concentrate on the case where the maximum difference between the number of redundant symbols of two successive codewords is unity, i.e.,

$$|\Delta r_i|_{\max} \triangleq |r_{i+1} - r_i|_{\max} = 1. \tag{7.13}$$

Here r_{i+1} and r_i ($i = 0, 1, 2, \cdots$) are the numbers of redundant symbols in codewords $\mathbf{c}_{i+1}$ and $\mathbf{c}_i$ respectively. For the sake of simplicity we take $h = 1$ and $\gamma = \alpha$ in (7.1), where α is a root of the primitive polynomial defining the field.

Let $\mathrm{g}_i(x) \triangleq \sum_{j=0}^{r_i} g_{i,j}x^j = \prod_{j=0}^{r_i}(x+\alpha^{j+1})$ denote the GP for the codeword $\mathbf{c}_i$. Then the GP $\mathrm{g}_{i+1}(x) \triangleq \sum_{j=0}^{r_{i+1}} g_{i+1,j}x^j$ for the next codeword $\mathbf{c}_{i+1}$ is given as follows:

$$\mathrm{g}_{i+1}(x) = \begin{cases} (x+\alpha^{r_i+1})\,\mathrm{g}_i(x) & \text{if } \Delta r_i = 1 \\ \mathrm{g}_i(x) & \text{if } \Delta r_i = 0 \\ (x+\alpha^{r_i})^{-1}\,\mathrm{g}_i(x) & \text{if } \Delta r_i = -1. \end{cases} \tag{7.14}$$

Considering the case $\Delta r_i = 1$, the coefficients of $\mathrm{g}_{i+1}(x)$ are

$$g_{i+1,j} = \begin{cases} g_{i,r_i} & j = r_{i+1} \\ g_{i,j-1} + g_{i,j}\alpha^{r_i+1} & r_{i+1} > j \geq 1 \\ g_{i,0}\alpha^{r_i+1} & j = 0. \end{cases} \tag{7.15}$$

For the case $\Delta r_i = -1$, we note that since $\mathrm{g}_i(x)$ has a root α^{r_i}, $(x+\alpha^{r_i})$ divides $\mathrm{g}_i(x)$. Performing this division operation and equating coefficients of like powers of x on both sides of (7.14) we obtain

$$g_{i+1,j} = \begin{cases} g_{i,r_i} & j = r_{i+1} \\ g_{i,j+1} + g_{i+1,j+1}\alpha^{r_i} & r_{i+1} > j \geq 0. \end{cases} \tag{7.16}$$

Figure 7.5 shows a flowchart for the generation of $\mathrm{g}_{i+1}(x)$ from $\mathrm{g}_i(x)$ using (7.15) and (7.16), where b is initially assigned α^{r_i}.

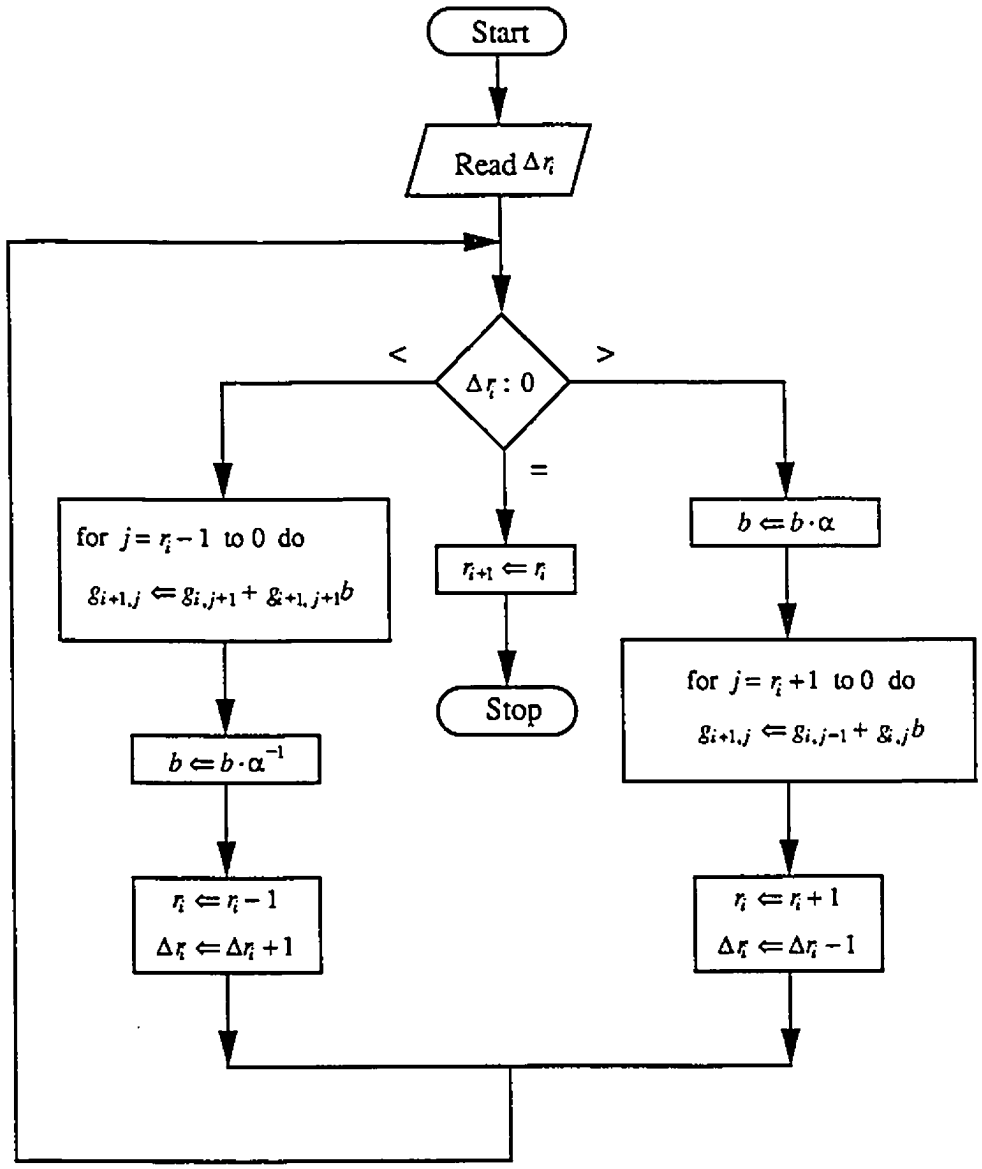


Figure 7.5: Flow diagram for the generation of $g_{i+1}(x)$ from $g_i(x)$.

7.5.2 Structure

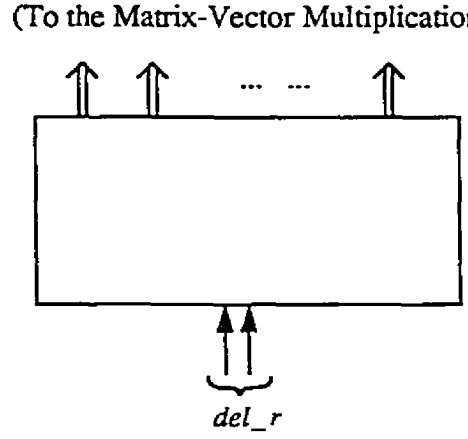
An overall block diagram for the variable GP generator is given in Figure 7.6(a). The input *del_r*, which is obtained from the receiver side, depends on the channel condition and may have one of the three values viz., {0, 0}, {1, 0} and {1, 1} corresponding to $\Delta r_i = 0,\ 1$, and -1 respectively. There are r_{max} output lines which are connected to the modules of the Matrix-Vector Multiplication unit of Figure 7.3, giving a rate-adaptive RS encoder. Each of the outputs is a symbol of GF(2^m). For the codeword c_{i+1}, the coefficient $g_{i+1,r_{i+1}-1}$ of the GP appears on the rightmost output line, $g_{i+1,r_{i+1}-2}$ on the adjacent line, and so on. The leftmost $r_{max} - r_{i+1}$ outputs are $0 \in$GF(2^m); if $r_{i+1} = 0$, then all the outputs are $0 \in$GF(2^m).

The structure for obtaining $g_{i+1}(x)$ from $g_i(x)$ is shown in Figure 7.6(b). It is divided into three main units- a Coefficient Unit, a pipelined bit-serial constant multiplier and a Root Generation Unit. The Coefficient unit stores the GP coefficients during the process of redundant symbol generation. Registers G_0, G_1, $\cdots$, $G_{r_{max}}$ are serial-in-parallel-out type. The output of $G_{r_{max}-j}$ is the coefficient g_{i,r_i-j} for $j = 0,\ 1,\ \cdots, r_i$ and $i = 0,\ 1,\ 2,\ \cdots$. At $i = 0$, the registers are initialized with the coefficients of $g_0(x)$. The root generator outputs one of the symbols of the set $\{\alpha^0,\ \alpha^1,\ \cdots,\ \alpha^{r_{max}}\}$. At $i = 0$, the root generator is set to α^{r_0}. It essentially consists of α and α^{-1} multiplication circuits whose structures are similar and require only one clock cycle to perform either of the multiplications [5].

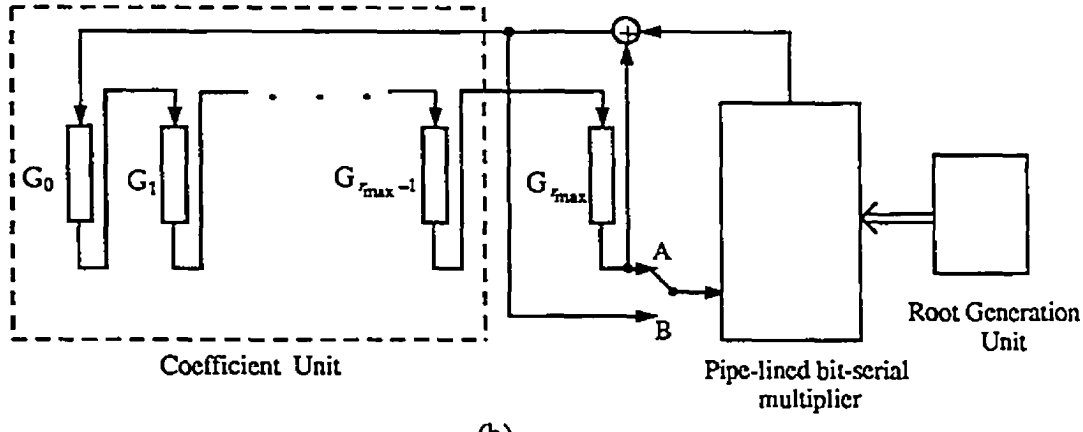
Depending on the value of Δr_i, the structure operates in the following three modes.

1. If $\Delta r_i = 1$, then at the instant S_{i+1} the switch is placed in position A and the output of the root generator is updated from α^{r_i} to $\alpha^{r_{i+1}} = \alpha^{r_i}\alpha$. From the next cycle, the contents of the coefficient registers are shifted $m(r_{max}+1)$ times to obtain $g_{i+1}(x) = (x + \alpha^{r_{i+1}})g_i(x)$.

2. If $\Delta r_i = 0$, the coefficient registers are not shifted and the output of the root



(a)



(b)

Figure 7.6: (a) An overall block diagram for the GP generator. (b) Structure for the generation of GPs for a rate-adaptive RS encoder.

generator is left unchanged.

3. If $\Delta r_i = -1$, then at the instant S_{i+1} the switch is placed in position B. After $m(r_{\max}+1)$ clock cycles, the coefficient registers contain $\mathrm{g}_{i+1}(x) = (x+\alpha^{r_i})^{-1}\mathrm{g}_i(x)$. Having been obtained the coefficients of the GP, the output of the root generator is then updated from α^{r_i} to $\alpha^{r_{i+1}} = \alpha^{r_i} \cdot \alpha^{-1}$.

In addition to a control unit, for an arbitrary primitive polynomial over GF(2) of degree m the rate-adaptive encoder (Figures 7.3 and 7.6 combined) requires $m(9+2r_{\max})-4$ one-bit registers, $m(9+2r_{\max})-(r_{\max}+8)$ two-input XOR gates and $m(9+r_{\max})-8$ two-input AND gates.

The time required to generate $\mathrm{g}_{i+1}(x)$ is $B_{i+1}-S_{i+1}$ which is the same for all i. To be able to start transmitting c_{i+1} at B_{i+1}, $B_{i+1}-S_{i+1}$ must be greater than or equal to $m(r_{\max}+1)+1$ clock cycles. This structure can be used for higher values of $|\Delta r_i|_{\max}$, say $L \leq r_{\max}$, in which case if Δr_i is greater (resp. less) than zero, then the structure operates in mode 1 (resp. 3) for L times. For $|\Delta r_i|_{\max} = L$,

$$B_{i+1} - S_{i+1} \geq L\{m(r_{\max}+1)+1\} \text{ clock cycles.} \tag{7.17}$$

7.6 Conclusions

In this chapter, a structure for a systematic rate-adaptive RS encoder over GF(2^m) has been developed using a pipelined bit-serial constant multiplier. The structure can be easily modified for RS codes with different values of m and t. Also, if a different primitive polynomial is chosen to define the field GF(2^m), this change can be readily incorporated without any pre-algebraic manipulation. Moreover, the encoder is very area efficient and has a low circuit complexity. Thus it is well suited for use in applications where silicon area is a prime concern.

Chapter 8

Summary, Conclusions and Suggestions for Future Research

8.1 Summary and Conclusions

Operations in the fields GF(2^m) are quite different from those in the field of integers. The elements of GF(2^m) can be represented by m binary digits. Such a representation allows simple addition and subtraction operations, but division and multiplication operations are necessarily complex.

In this dissertation, several important existing algorithms for computing multiplication and inversion in finite fields are briefly studied. The concept of *supporting* elements is then presented. Using the coordinates of the supporting elements, an algorithm for computing division in GF(q^m), where q is a prime and m is a nonzero positive integer, is developed. The algorithm solves m linear equations over GF(q) in m unknowns. The algorithm is general in the sense that it is applicable for any basis representation of the field and any irreducible polynomial defining the field.

Structures for division and inversion computations in GF(2^m) are considered. An algorithm is developed for the formation of the *coefficient matrix*. This algorithm is mapped onto a one dimensional systolic array. Using the Gauss-Jordan diagonalization algorithm over GF(2), an efficient two dimensional array is obtained. These two arrays are combined to yield a bit-serial systolic divider for

$GF(2^m)$ with a time complexity proportional to m. The divider uses three basic types of processing elements and requires no global data communication. This is suitable for applications where very large values of m (say, m=1000) are used.

For a canonical basis representation of elements of $GF(q^m)$, a relationship between a division and a discrete time Wiener-Hopf equation (DTWHE) over $GF(q)$ is derived. This relationship leads to an efficient bit-serial multiplication scheme which can be easily realized for all irreducible polynomials. Relationships are derived to generate the constant coefficients of the DTWHE by simple linear feedback shift registers.

Algorithms for computing inversion and multiplication in the class of finite field $GF(2^m)$ generated by irreducible *all one polynomials* (AOP) and *equally spaced polynomials* (ESP) are presented. Structures for parallel multipliers based on irreducible AOP and ESP are developed. The structures are simple and modular which is important for hardware realization. Relationships between an irreducible AOP and the corresponding irreducible ESP are exploited to construct ESP based multipliers for large fields by a regular expansion of the modules of the AOP based multiplier for a small field. Some features of the structures also enable fast squaring and multiplication algorithms, making fast exponentiation and inversion possible. It is shown that for a given degree if an irreducible AOP as well as an irreducible ESP exist, then from the complexity point of view, it is advantageous to use the ESP based parallel multiplier.

Finally, a low complexity Reed-Solomon encoder using a pipelined bit-serial constant multiplier has been presented. The coefficients of the generator polynomial, input data, and output codeword symbols are represented in terms of a common canonical basis. Moreover, the encoder supports variable code rates, allowing improved channel utilization.

8.2 Suggestions for Future Research

The followings are interesting topics which may be pursued for future research.

- Further use of the supporting elements, especially for the normal basis.
- Finding other bases where the coordinates of the supporting elements can be used to develop efficient algorithms for computations in finite fields.
- Development of modular structures for general parallel-type multipliers.
- Development of efficient dividers with area-time complexity less than $O(m^3)$.
- Algorithms for parallel inversion which will be well suited for hardware realization.

Appendix A

Proofs

Proof of Theorem 3.2

Denote (3.16) as $\mathbf{U'b} = \tilde{\mathbf{c}}$ with $\mathbf{U'} \triangleq \left[u'_{i,j}\right]_{i,j=0}^{m-1}$ and $\tilde{\mathbf{c}} \triangleq [\tilde{c}_i]_{i=0}^{m-1}$. The transformation from $\mathbf{c}$ of (3.14) to $\tilde{\mathbf{c}}$ follows directly from (3.15). It is required to show that

$$u'_{i,j} = \tilde{a}_{m-1+i-j} = \mathbf{a} \cdot \mathbf{p}_{m-1}^{[m-1+i-j]} \quad i,j = 0,\ 1,\ \cdots, m-1.$$

This will be shown in two parts, viz.,

P1) The 0th row and the 0th column of $\mathbf{U'}$ are

$$\{\tilde{a}_{m-1}\ \tilde{a}_{m-2}\ \cdots\ \tilde{a}_0\}$$

and

$$[\tilde{a}_{m-1}\ \tilde{a}_m\ \cdots\ \tilde{a}_{2m-2}]^t$$

respectively; and

P2) $\quad u'_{i,j} = u'_{i-1,j-1} \quad$ for $1 \le i,j \le m-1$.

In the rest of the analyses it is implicit that all operations other than those involving the indices are modulo q.

P1) As the row operation (3.15) does not change the 0th row of **U** in (3.14), using (3.17) we directly have the 0th row of **U'** as $\begin{bmatrix} \tilde{a}_{m-1} & \tilde{a}_{m-2} & \cdots & \tilde{a}_0 \end{bmatrix}$.

The row operations (3.15) leave the 0th column of **U** unchanged i.e.,

$$u'_{0,0} = u_{0,0} = \tilde{a}_{m-1}.$$

For $i = 1, 2, \cdots, m-1$, we have by induction

$$\begin{aligned} u'_{i,0} &= u_{i,0} - \sum_{k=1}^{i} u'_{i-k,0} g_{m-k} \\ &= \mathbf{a} \cdot \mathbf{p}_{m-1-i}^{[0]} - \sum_{k=1}^{i} \mathbf{a} \cdot \mathbf{p}_{m-1}^{[i-k]} g_{m-k} \\ &= \mathbf{a} \cdot \left(\mathbf{p}_{m-1-i}^{[0]} - \sum_{k=1}^{i} \mathbf{p}_{m-1}^{[i-k]} g_{m-k} \right). \end{aligned} \tag{A.1}$$

Using (3.12) repeatedly we obtain

$$\begin{aligned} u'_{i,0} &= \mathbf{a} \cdot \left(\mathbf{p}_{m-i}^{[1]} - \sum_{k=1}^{i-1} \mathbf{p}_{m-1}^{[i-k]} g_{m-k} \right) \\ &= \mathbf{a} \cdot \left(\mathbf{p}_{m-i+1}^{[2]} - \sum_{k=1}^{i-2} \mathbf{p}_{m-1}^{[i-k]} g_{m-k} \right) \\ &\vdots \\ &= \mathbf{a} \cdot \mathbf{p}_{m-1}^{[i]} = \tilde{a}_{m-1+i}. \end{aligned} \tag{A.2}$$

P2) Using (3.13), it is easy to verify that after the operations (3.15) on row 1, the latter becomes

$$u'_{1,j} = \tilde{a}_{m-j} = \mathbf{a} \cdot \mathbf{p}_{m-1}^{[m-j]} \quad j = 0, 1, \cdots, m-1.$$

Similarly, after completion of the operations (3.15) up to the $(i-1)$th row, for $1 \leq i, j \leq m-1$ we can write

$$\begin{aligned} u'_{i,j} &= u_{i,j} - \sum_{k=1}^{i} u'_{i-k,j} g_{m-k} \\ &= \mathbf{a} \cdot \mathbf{p}_{m-1-i}^{[m-1-j]} - \sum_{k=1}^{i} \bar{a}_{m-1+i-j-k} g_{m-k} \\ &= \mathbf{a} \cdot \left(\mathbf{p}_{m-1-i}^{[m-1-j]} - \sum_{k=1}^{i} \mathbf{p}_{m-1}^{[m-1+i-j-k]} g_{m-k} \right). \end{aligned} \tag{A.3}$$

Using (3.13) repeatedly we obtain

$$\begin{aligned} u'_{i,j} &= \mathbf{a} \cdot \left(\mathbf{p}_{m-i}^{[m-j]} - \sum_{k=1}^{i-1} \mathbf{p}_{m-1}^{[m-1+i-j-k]} g_{m-k} \right) \\ &= \mathbf{a} \cdot \left(\mathbf{p}_{m-i+1}^{[m-j+1]} - \sum_{k=1}^{i-2} \mathbf{p}_{m-1}^{[m-1+i-j-k]} g_{m-k} \right) \\ &\vdots \\ &= \mathbf{a} \cdot \mathbf{p}_{m-1}^{[m-1+(i-1)-(j-1)]} = u'_{i-1,j-1}. \end{aligned} \tag{A.4}$$

Proof of Theorem 4.2

Let $\mathbf{e}_l$ $(l = 0,\ 1, \cdots, m-1)$ denote the column vector whose l-th element is unity and whose other elements are zero. Then the elements of any row l, other than row j and i, of $\mathbf{A}'$ are

$$\begin{aligned} a'_{l,k} &= \mathbf{e}_l^t \mathbf{P}_{i,j} \mathbf{A} \mathbf{e}_k \quad (l \neq i, j;\ k = 0,\ 1,\ \cdots,\ m-1) \\ &= \mathbf{e}_l^t \mathbf{A} \mathbf{e}_k \\ &= a_{l,k} \end{aligned}$$

The elements of the i-th row of $\mathbf{A}'$ are $\mathbf{e}_i^t\mathbf{P}_{i,j}\mathbf{A}\mathbf{e}_k \quad (k = 0,\ 1,\ \cdots,\ m-1)$ i.e.,

$$\begin{aligned}
a'_{i,k} &= ha_{j,k} + \overline{f}a_{i,k} \quad (k = 0,\ 1, \cdots,\ m-1) \\
a'_{i,j} &= ha_{j,j} + \overline{f}a_{i,j} \\
&= a_{i,j}a_{j,j} + \overline{\overline{a}_{j,j}a_{i,j}}a_{i,j} \\
&= a_{i,j}a_{j,j} + (1 + \overline{a}_{j,j}a_{i,j})a_{i,j} \\
&= a_{i,j}a_{j,j} + a_{i,j} + (1 + a_{j,j})a_{i,j} = 0
\end{aligned}$$

Similarly the elements of the j-th row are $\mathbf{e}_j^t\mathbf{P}_{i,j}\mathbf{A}\mathbf{e}_k \quad (k = 0,\ 1,\ \cdots,\ m-1)$ i.e.,

$$\begin{aligned}
a'_{j,k} &= \overline{f}a_{j,k} + fa_{i,k} \quad (k = 0, 1, \cdots, j, \cdots, i, \cdots, m-1) \\
a'_{j,j} &= \overline{f}a_{j,j} + fa_{i,j} \\
&= \overline{\overline{a}_{j,j}a_{i,j}}a_{j,j} + \overline{a}_{j,j}a_{i,j}a_{i,j} \\
&= (1 + \overline{a}_{j,j}a_{i,j})a_{j,j} + \overline{a}_{j,j}a_{i,j} \\
&= a_{j,j} + \overline{a}_{j,j}a_{i,j} \\
&= \begin{cases} 1 & \text{if } a_{i,j} = 1 \\ a_{j,j} & \text{if } a_{i,j} = 0. \end{cases}
\end{aligned}$$

Bibliography

[1] K. Akari, 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.

[2] D. W. Ash, I. F. Blake, and S. A. Vanstone, "Low complexity normal bases," *Disc. Appl. Math.*, vol. 25, pp. 191–210, 1989.

[3] T. C. Bartee and D. I. Schneider, "Computation with finite fields," *Inform. and Comput*, vol. 6, pp. 79–98, Mar. 1963.

[4] B. Benjauthrit and I. S. Reed, "Galois switching functions and their applications," *IEEE Trans. Comput.*, vol. C-25, pp. 78–86, Jan. 1976.

[5] E. R. Berlekamp, *Algebraic Coding Theory.* New York: McGraw-Hill, 1968.

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

[7] G. I. Davida, "Inverse of elements of a Galois field," *Electron. Lett.*, vol. 8, pp. 518–520, Oct. 1972.

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

[9] T. A. Gulliver, M. Serra, and V. K. Bhargava, "The generation of primitive polynomials in GF(q) with independent roots and their applications for power

residue codes, VLSI testing and finite field multipliers using normal basis," *Int. J. Electronics*, vol. 71, no. 4, pp. 559–576, 1991.

[10] R. W. Hamming, "Error detecting and error correcting codes," *Bell Sys. Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

[11] M. A. Hasan and V. K. Bhargava, "Bit-serial systolic divider and multiplier for GF(2^m)." To appear in *IEEE Trans. Comput.*

[12] M. A. Hasan and V. K. Bhargava, "Division and bit-serial multiplication over GF(q^m)." To appear in *IEE Proceedings Part E.*

[13] M. A. Hasan and V. K. Bhargava, "A VLSI architecture for a low complexity rate-adaptive Reed-Solomon encoder." Accepted for presentation in the 16th Biennial Symposium on Communications, May 1992, Kingston, Ontario.

[14] M. A. Hasan and V. K. Bhargava, "Multiplication and inversion over a class of GF(2^m)," *Proc. IEEE Pacific Rim Conf.*, 1991.

[15] M. A. Hasan, M. Z. Wang, and V. K. Bhargava, "Modular construction of low complexity parallel multipliers for a class of finite fields GF(2^m)." To appear in *IEEE Trans. Comput.*

[16] B. Hochet, P. Quinton, and Y. Robert, "Systolic Gaussian elimination over GF(p) with partial pivoting," *IEEE Trans. Comput.*, vol. C-38, pp. 1321–1324, Sept. 1989.

[17] A. Hocquenghem, "Codes corecteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, 1959.

[18] I.-S. Hsu, I. S. Reed, T. K. Truong, K. Wang, C. S. Yeh, and L. J. Deutsch, "The VLSI implementation of a Reed-Solomon encoder using Berlekamp's bit-serial multiplier algorithm," *IEEE Trans. Comput.*, vol. C-33, pp. 906–911, Oct. 1984.

[19] T. Itoh, "A fast algorithm for computing multiplicative inverses in GF(2^m)," *Inform. and Comp.*, vol. 78, pp. 171–177, 1988.

[20] T. Itoh, "Characterization for a family of infinitely many irreducible equally spaced polynomials," *Inform. Process. Letters*, vol. 37, pp. 272–277, 1991.

[21] T. Itoh and S. Tsujii, "Structure of parallel multipliers for a class of fields GF(2^m)," *Inform. and Comp.*, vol. 83, pp. 21–40, 1989.

[22] H. T. Kung, "Why systolic architectures?," *IEEE Comput.*, vol. 15, no. 1, pp. 37–45, 1982.

[23] S. Kung, *VLSI Array Processors.* Englewood Cliffs, NJ: Prentice Hall, 1988.

[24] B. A. Law and C. K. Rushforth, "A cellular-array multiplier for GF(2^m)," *IEEE Trans. Comput.*, vol. C-20, pp. 1573–1578, Dec. 1971.

[25] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications.* Cambridge: Cambridge University Press, 1986.

[26] S. Lin and J. D.J. Castello, *Error Control Coding: Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1983.

[27] F. J. MacWilliams and N. J. Sloane, *The Theory of Error-correcting Codes.* New York: North Holland, 1977.

[28] J. L. Massey and J. K. Omura, "Apparatus for finite field computation," *US Patent Application*, pp. 21–40, 1984.

[29] E. D. Mastrovito, *VLSI Architectures for Computations in Galois Fields.* PhD thesis, Dept. Elect. Eng., Linköping University, Linköping, Sweden, 1991.

[30] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers.* Boston, MA: Kluwer Academic, 1987.

[31] 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.

[32] W. W. Peterson and E. J. Weldon, *Error Correction Codes*. Cambridge, MA: MIT Press, 1972.

[33] A. Pincin, "A new algorithm for multiplication in finite fields," *IEEE Trans. Comput.*, vol. C-38, pp. 1045–1049, 1989.

[34] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300–304, June 1960.

[35] I. S. Reed and T. K. Truong, "The use of finite fields to compute convolutions," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 208–213, Mar. 1975.

[36] P. A. Scott, S. E. Tavares, and L. E. Peppard, "A fast VLSI multiplier for GF(2^m)," *IEEE J. Select. Areas Commun.*, vol. SAC-4, pp. 62–66, Jan. 1986.

[37] G. Seroussi, "A systolic Reed-Solomon encoder," *IEEE Trans. Inform. Theory*, vol. IT-37, pp. 1217–1220, July 1991.

[38] D. R. Stinson, "On bit-serial multiplication and dual bases in GF(2^m)," *IEEE Trans. Inform. Theory*, vol. IT-37, pp. 1733–1736, Nov. 1991.

[39] 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, 1988.

[40] W. F. Trench, "An algorithm for the inversion of finite Toeplitz matrices," *J. Soc. Indus. Appl. Math.*, vol. 12, pp. 512–522, Sept. 1964.

[41] P. K. S. Wah and M. Z. Wang, "Realization and application of the Massey-Omura lock," *International Zurich Seminar*, 1984.

[42] C. C. Wang, *Exponentiation in Finite Fields.* PhD thesis, School Eng. Appl. Sci., Univ. Calif. Los Angeles, 1985.

[43] 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.

[44] M. Z. Wang and I. F. Blake, "Bit serial multiplication in finite fields," *SIAM J. Disc. Math.*, vol. 3, 1990.

[45] M. Z. Wang, I. F. Blake, and V. K. Bhargava, "Normal bases and irreducible polynomials in the finite field GF(2^{2^r})." Submitted to *Disc. Appl. Math.*, 1990.

[46] C.-S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite fields GF(2^m)," *IEEE Trans. Comput.*, vol. C-33, pp. 357–360, Apr. 1894.

[47] B. B. Zhou, "A new bit-serial systolic multiplier over GF(2^m)," *IEEE Trans. Comput.*, vol. C-37, pp. 749–751, June 1988.