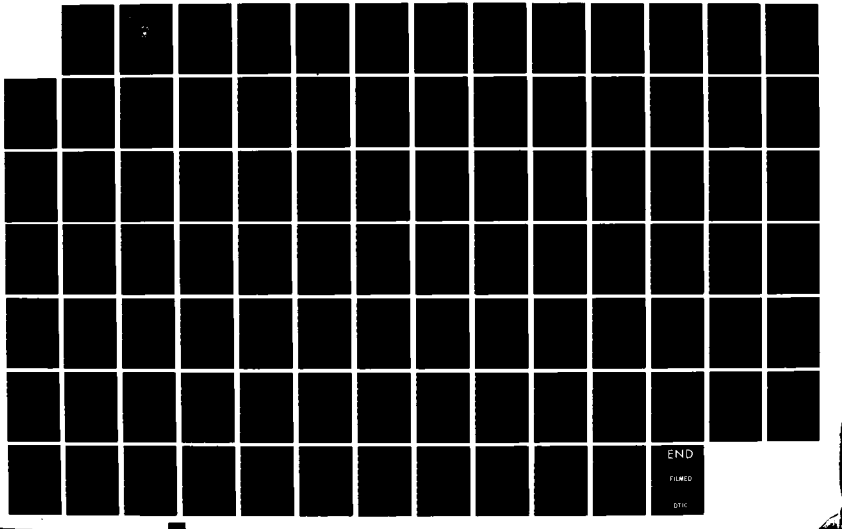
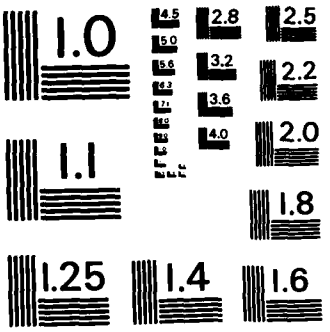
AD-A159 824 A SYSTOLIC ARRAY IMPLEMENTATION OF A REED-SOLOMON ENCODER AND DECODER(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S S MCKENZIE JUN 85 1/1

UNCLASSIFIED F/G 9/2 NL





NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A SYSTOLIC ARRAY IMPLEMENTATION OF A REED-SOLOMON ENCODER AND DECODER

by

Stephen Scott McKenzie

June 1985

Thesis Advisor: H. Fredricksen

Approved for public release; distribution is unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A159 824	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Systolic Array Implementation of a Reed-Solomon Encoder and Decoder		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen Scott McKenzie		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		12. REPORT DATE June 1985
		13. NUMBER OF PAGES 92
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Systolic Arrays, Finite Fields, Reed-Solomon Codes, RS Encoder, RS Decoder, Systolic Multiplier, Primitive Polynomial, Primitive Shift Register, VLSI, Pipelining, Parallelism

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A systolic array is a natural architecture for the implementation of a Reed-Solomon (RS) encoder and decoder. It possesses many of the properties desired for a special-purpose application: simple and regular design, concurrency, modular expansibility, fast response time, cost-effectiveness, and high reliability. As a result, it is very well suited for the simple and regular design essential for VLSI implementation.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

20 (Continued)

This thesis takes a modular approach to the design of a systolic array based RS encoder and decoder. Initially, the concept of systolic arrays is discussed followed by an introduction to finite field theory and Reed-Solomon codes. Then it is shown how RS codes can be encoded and decoded with primitive shift registers and implemented using a systolic architecture. In this way, the reader can gain valuable insight and comprehension into how these entities are coalesced together to produce the overall implementation.

Approved for public release; distribution is unlimited

A Systolic Array Implementation of a Reed-Solomon Encoder and Decoder

by

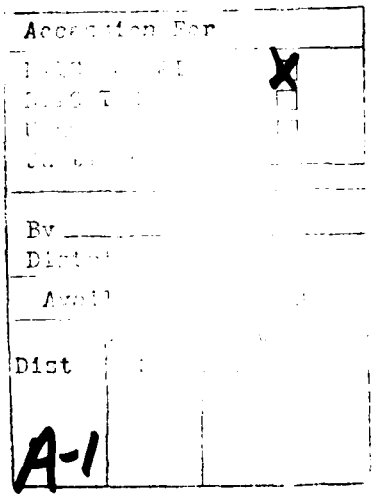
Stephen Scott McKenzie
Lieutenant, United States Navy
B.S., United States Naval Academy, 1979

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1985



Author: Stephen Scott McKenzie
Stephen Scott McKenzie

Approved by: Harold M. Fredricksen
Harold M. Fredricksen, Thesis Advisor

Alan A. Ross
Alan A. Ross, Second Reader

Bruce J. MacLennan
Bruce J. MacLennan, Chairman, Department of Computer Science

Kneale T. Marshall
Kneale T. Marshall, Dean of Information and Policy Sciences

ABSTRACT

A systolic array is a natural architecture for the implementation of a Reed-Solomon (RS) encoder and decoder. It possesses many of the properties desired for a special-purpose application: simple and regular design, concurrency, modular expansibility, fast response time, cost-effectiveness, and high reliability. As a result, it is very well suited for the simple and regular design essential for VLSI implementation.

This thesis takes a modular approach to the design of a systolic array based RS encoder and decoder. Initially, the concept of systolic arrays is discussed followed by an introduction to finite field theory and Reed-Solomon codes. Then it is shown how RS codes can be encoded and decoded with primitive shift registers and implemented using a systolic architecture. In this way, the reader can gain valuable insight and comprehension into how these entities are coalesced together to produce the overall implementation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

In this very volatile and technological age, it is imperative that communication links and computer memories transmit information reliably and quickly. However, in many cases this is virtually impossible because noise causes the received data to differ significantly from the original data. In order to rectify this situation error-correcting codes have been developed to enable a system to continually maintain a high degree of reliability despite the presence of noise. To accomplish the error correction, in addition to the data or information bits that are transmitted, some additional redundant-check bits or parity bits are also transmitted. In this way, although the noise may introduce some errors in either the transmitted data bits or the transmitted check bits, there are usually still enough uncorrupted bits available to the receiver to allow a sophisticated decoder to correct the errors. In fact, only a modest amount of redundancy is actually needed to ensure that the probability of the decoding error is negligibly small. [Ref. 1]

Nonetheless, unlike the encoders and decoders of the 1950's and 1960's which were constrained by digital hardware costs and virtually nonexistent chip technology, today's

encoders and decoders coupled with significant improvements in their associated algorithms have become, and will continue to be, increasingly attractive from an economic viewpoint.

One such class of error-correcting codes which is very popular in the communication circles and is paramount in this author's discussion of systolic array encoders and decoders are the Reed-Solomon (RS) codes. These codes can correct both random and burst errors over a communication channel, and as such are ideal for the very low error probabilities needed for reliable space communications. Still, the RS codes are only as effective as the complexity of the encoder that produces them and the decoder by which errors are corrected. The encoder complexity is directly proportional to the error-correcting capability of the code, the speed of the encoding process, and the interleaving level used, i.e., the number of original codewords which are multiplexed together to increase the immunity of codes to burst errors [Ref. 1]. In fact, for truly reliable space communications there is a bonafide need to use RS codes with a large error-correcting capability and an equally large interleaving level. As a result, one is especially interested in decreasing or minimizing the complexity of an RS encoder while simultaneously ensuring maximum performance and high reliability. Clearly, what is needed for this type of application is a special-purpose system which compliments

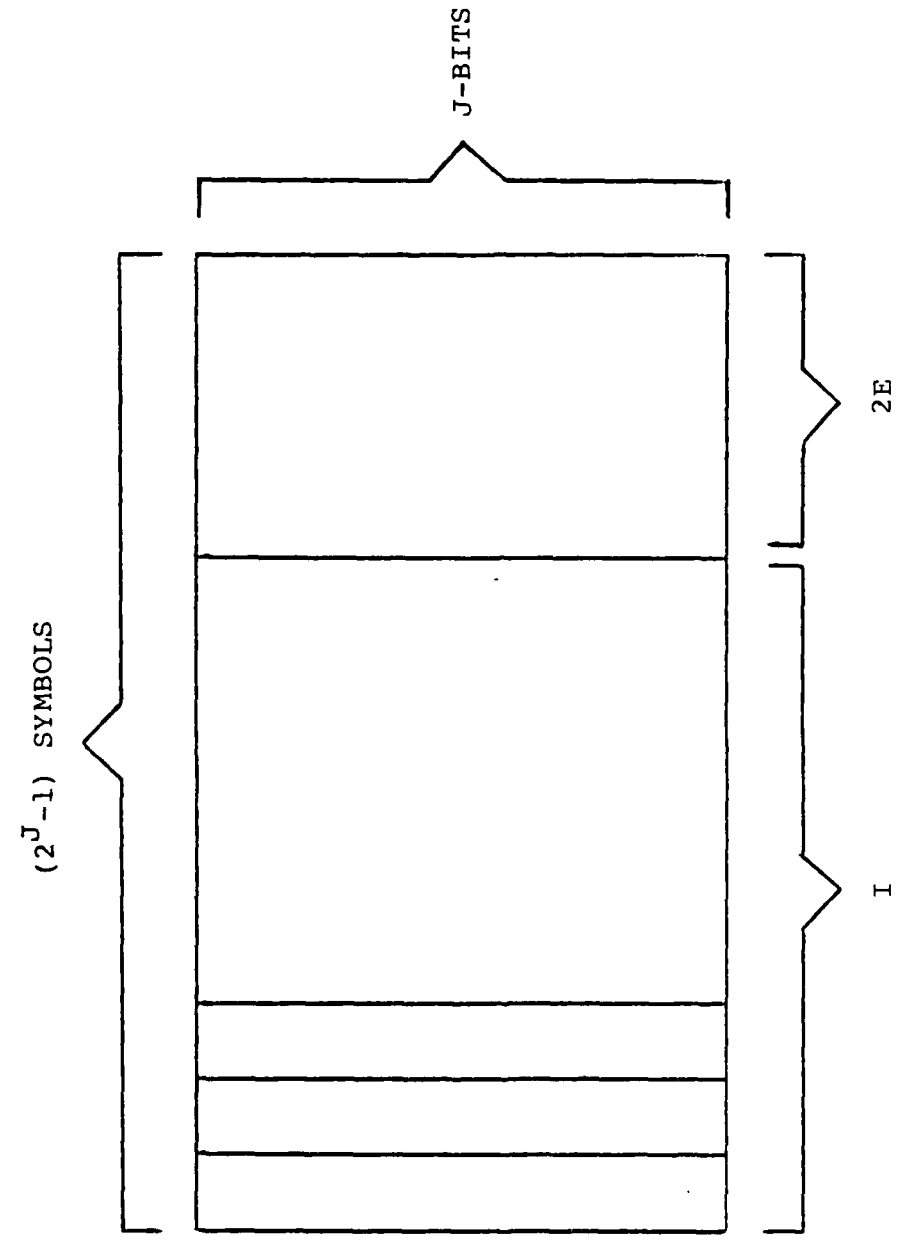


Figure 4.1 A Reed-Solomon Codeword

In this chapter we look at a generic construction and architecture of an RS encoder developed by Johl [Ref. 9] and use this design as a foundation for the subsequent discussion and implementation in the later chapters. This implementation utilizes a systolic architecture of identical cells arranged in a linear array, each executing a finite-field multiplication and addition in a pipelined manner; thereby, significantly increasing the throughput rate. Also, since the layout of the cell need only be done once and then replicated, it is extremely attractive for eventual VLSI implementation.

B. GENERIC ARCHITECTURE

The RS code is a block code which consists of symbols of more than one bit. When each symbol is J-bits wide, an RS codeword has (2^J-1) symbols. As depicted in Figure 4.1, an RS code can be designed to be capable of correcting E errors with each codeword consisting of I information symbols, together with 2E parity or check symbols. As an example, given the irreducible polynomial $1+\beta+\beta^4=0$ and its corresponding finite field as described in Table I we are able to establish an important foundation vital to the development of a generic RS encoder. This RS code consists of a total of 15-four bit symbols for each codeword. If this particular code should correct one error, it would need two parity symbols and therefore would contain thirteen information

IV. REED-SOLOMON CODES

A. BACKGROUND

Reed-Solomon (RS) codes are Bose-Chaudhuri-Hocquenghem (BCH) codes over GF(q) of length q-1. They are error-correcting codes which are used in many special-purpose applications ranging from deep-space communications and spread spectrum to digital audio disk systems and secure data transmissions [Ref. 7]. These codes can correct both random and burst errors over a communication channel and hence are ideal for the numerous, real-time, and reliable applications demanded by these applications. The complexity of RS encoders and decoders is proportional to the error-correcting capability of the code, the speed of the decoding, and the interleaving depth used [Ref. 8]. For truly reliable communications there is a very strong tendency to use RS codes with a large error-correcting capability and an equally large interleaving level. Hence, one is especially interested in minimizing the complexity of RS encoders and decoders for communications and other pertinent applications. Toward this end, there is a considerable interest in systolic array construction and eventual VLSI implementation of RS encoders and decoders which yield significant savings in size, weight, and power consumption while simultaneously providing high reliability.

TABLE I

REPRESENTATION OF $GF(2^4)$

FIELD ELEMENT	BETA POLYNOMIAL	4-TUPLE
$\beta^0=1$	1	1 0 0 0
$\beta^1=\beta$	β	0 1 0 0
$\beta^2=\beta^2$	β^2	0 0 1 0
$\beta^3=\beta^3$	β^3	0 0 0 1
$\beta^4=\beta+1$	$1 + \beta$	1 1 0 0
$\beta^5=\beta(\beta^4)$	$\beta + \beta^2$	0 1 1 0
$\beta^6=\beta(\beta^5)$	$\beta^2 + \beta^3$	0 0 1 1
$\beta^7=\beta(\beta^6)$	$1 + \beta + \beta^3$	1 1 0 1
$\beta^8=\beta(\beta^7)$	$1 + \beta^2$	1 0 1 0
$\beta^9=\beta(\beta^8)$	$\beta + \beta^3$	0 1 0 1
$\beta^{10}=\beta(\beta^9)$	$1 + \beta + \beta^2$	1 1 1 0
$\beta^{11}=\beta(\beta^{10})$	$\beta + \beta^2 + \beta^3$	0 1 1 1
$\beta^{12}=\beta(\beta^{11})$	$1 + \beta + \beta^2 + \beta^3$	1 1 1 1
$\beta^{13}=\beta(\beta^{12})$	$1 + \beta^2 + \beta^3$	1 0 1 1
$\beta^{14}=\beta(\beta^{13})$	$1 + \beta^3$	1 0 0 1

B. AN EXAMPLE OF THE CREATION OF A FIELD

Consider the Galois field $GF(2^4)$. It has 2^4 elements and may be constructed as the field of polynomials over GF(2) modulo the irreducible polynomial $1+x+x^4$. If we let β represent a root of $1+x+x^4$, then it is also a primitive element of the field. Field addition of the elements is bit-by-bit modulo 2 addition while multiplication of the elements is described using the primitivity of the element β. Thus, $\beta^i * \beta^j = \beta^{i+j}$ where i+j is reduced modulo 15, if necessary. For example, given the field elements β^{13} and β^9, two of the 15 nonzero field elements listed in Table I, we can easily demonstrate both operations: $\beta^{13} + \beta^9 = (\beta^3 + \beta^2 + 1) + (\beta^3 + \beta) = 2\beta^3 + \beta^2 + \beta + 1 = \beta^2 + \beta + 1 = \beta^{10}$ while $\beta^{13} * \beta^9 = \beta^{13+9} = \beta^{22} = \beta^{22-15} = \beta^7 = \beta^3 + \beta + 1$.

infinite field. The rational numbers, the real numbers, and the complex numbers are all examples of infinite fields. If the number of elements is finite we call the field a finite field [Ref. 5].

For any prime p and any positive integer m a Galois field denoted $GF(p^m)$ or GF(q) exists. We can construct a field containing p^m elements as an algebra of polynomials modulo an irreducible polynomial over GF(p) of degree m. Addition is bit-by-bit modulo p addition.

The multiplicative group of the nonzero field elements is cyclic, i.e., it is a group that consists of all the powers of one of its elements, β. Multiplication is defined as $\beta^i * \beta^j = \beta^{i+j}$ where i+j is computed modulo (p^m-1) and β is a generator of this group. A generator of this multiplicative group, called a primitive element, is a root of an irreducible polynomial over the prime field GF(p). This irreducible polynomial, called a primitive polynomial, is the minimal polynomial of the primitive element, i.e., the polynomial of least degree with the primitive element β as a root. Generally speaking, an irreducible polynomial is analogous to a prime number: it has no nontrivial factors. Lastly, the Galois fields that can be created by taking residue or equivalence classes of polynomials modulo an irreducible polynomial over GF(p) are said to be fields of characteristic p. Thus, $GF(p^m)$ is a field of characteristic p for each choice of positive integer m [Ref. 6].

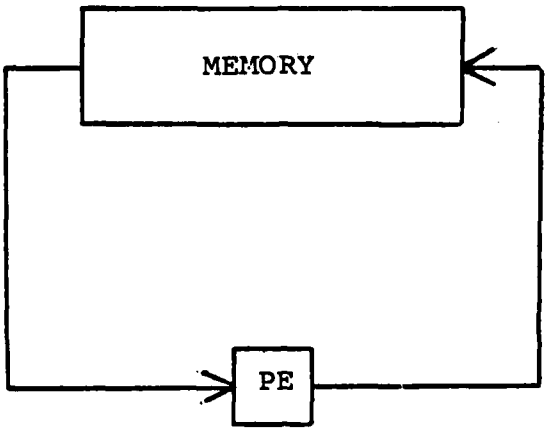
III. FINITE FIELD THEORY

A. BACKGROUND

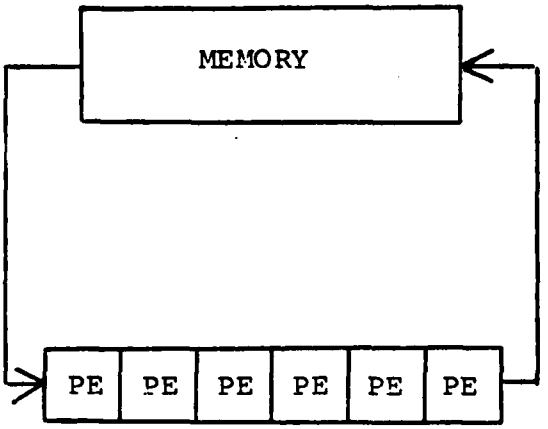
Finite or Galois fields (named after the nineteenth century French mathematician Évariste Galois) play many important and diverse roles in numerous applications ranging from digital signal processing to switching theory. However, in this thesis we are concerned with their use in the construction of Reed-Solomon error-correcting codes. We begin with a general analysis of the pertinent facts regarding finite fields. In the next chapter the necessary facts about Reed-Solomon codes are discussed.

A field is a set of elements, including 0 and 1, any pair of which may be added or multiplied (denoted by + and *, respectively) to give a unique result in the field. The addition and multiplication are associative and commutative, and multiplication distributes over addition in the usual way: u*(v+w)=u*v+u*w. Every field element u has a unique negative -u such that u+(-u)=0. Every nonzero field element u has a unique reciprocal field element 1/u, such that u*(1/u)=1. For every field element u, 0+u=u=1*u, and 0*u=0. Thus the numbers 0 and 1 are the additive and multiplicative identities, respectively. [Ref. 5]

The order of a field is the number of elements in the field. If the order is infinite, we call the field an



(a) THE CONVENTIONAL PROCESSOR



(b) A SYSTOLIC PROCESSOR ARRAY

Figure 2.2 The Concept of a Systolic Processor Array

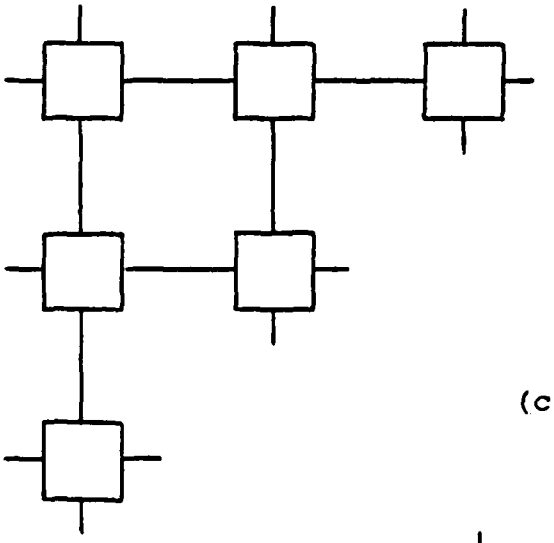
B. PRINCIPLE OF OPERATION

The basic principle of a systolic array is illustrated in Figure 2.2. As stated earlier, by replacing a single processing element (PE) with an array of processing elements, a higher computation throughput can be achieved without increasing the memory bandwidth.

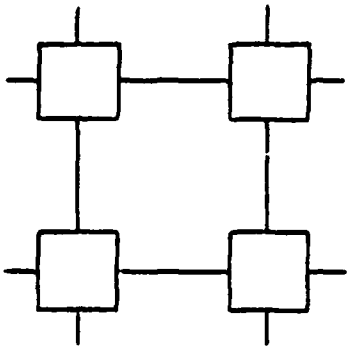
> Suppose each processing element in Figure 2.2 operates with a clock period of 100 nanoseconds (ns). The conventional memory-processor organization in Figure 2.2a has at most a performance of 5 million operations per second (MOPS). With the same clock rate, the systolic array processor will result in 30 MOPS performance. This gain in processing speed can also be justified with the fact that the number of pipeline stages has been increased six times in Figure 2.2b. Being able to use each input data item a number of times is just one of the many advantages of the systolic approach. Other advantages include modular expansibility, simple and regular data and control flows, use of simple and uniform cells, elimination of global broadcasting, limited fan-in and fast response time. [Ref. 3]

With the above criteria a systolic array is a natural architecture for the implementation of an RS encoder and decoder which will become apparent after our introduction of Reed-Solomon codes in Chapter IV.

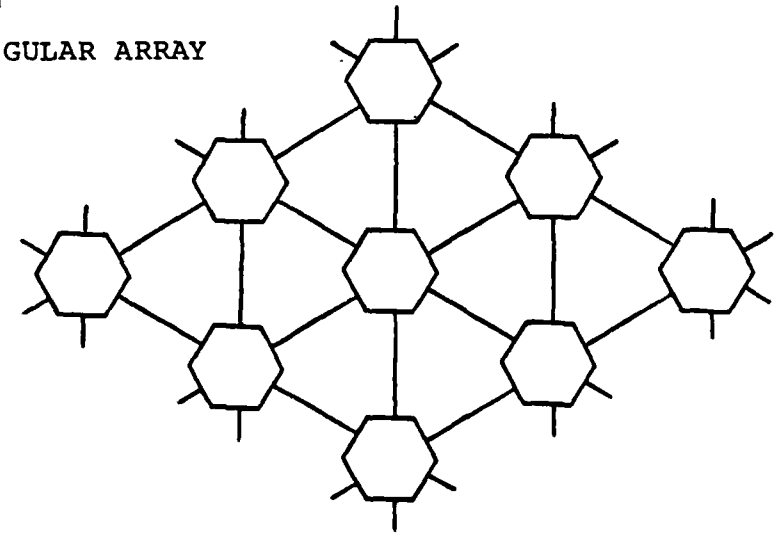
(a) ONE-DIMENSIONAL LINEAR ARRAY



(b) TRIANGULAR ARRAY



(c) TWO-DIMENSIONAL SQUARE ARRAY



(d) TWO-DIMENSIONAL HEXAGONAL ARRAY

Figure 2.1 Various Systolic Array Configurations

Clearly, what is required is a special-purpose design which employs simple and regular communication paths for multiprocessor structures in addition to pipelining as a general method for utilizing these structures. In short, systolic arrays provide a realistic model of computation which captures these concepts of pipelining, parallelism, and interconnection structures.

According to Kung and Leiserson [Ref. 2]:

> A systolic array is a collection of relatively simple processing units, usually of the same type, which are connected together by a simple communication network and that operate in parallel, as depicted in Figure 2.1. The performance advantage of a systolic array architecture is that it uses each datum retrieved from memory numerous times without having to store and retrieve intermediate results, thus allowing significant speedups relative to the memory bandwidth. Thus, a systolic system is a network of processors which rhythmically computes and passes data through the system. The analogy is to the rhythmic contraction of the heart which pulses blood through the circulatory system of the body. Each processor in a systolic network can be thought of as an element through which multiple streams of data are pumped. The regular beating of these parallel processors maintains a constant flow of data throughout the entire network. As data items are pumped through the network some constant-time computation is performed and, depending on the operation, updates of some of the items may occur. However, unlike the closed-loop circulatory system of the body, a systolic computing system usually has ports into which inputs flow, and ports from which the results of the computation are received. Thus, a systolic system can be viewed as a pipelined system--one in which input and output occur with every pulsation.

As a result, this makes it extremely attractive for a wide class of compute-bound computations where multiple operations are performed on each data item in a repetitive manner.

which are used repetitively with simple interfaces, tremendous savings can be achieved.

This is especially true for VLSI designs where a single chip usually comprises hundreds of thousands of identical components. Clearly, in order to overcome this complexity, simple and regular designs are essential. In fact, VLSI systems which are based on simple, regular layouts are very likely to be modular and therefore adjustable to various performance levels. Still, with the technological indication of a diminishing growth rate for component speed, any major improvement in computation speed must come from the concurrent use of many processing elements. [Ref. 3]

The degree of concurrency in a VLSI computing structure is largely determined by the underlying algorithm. Consequently, massive parallelism can be achieved if the algorithm is designed to exploit high degrees of pipelining and multiprocessing. For instance, when a large number of processing elements work simultaneously, coordination and communication become significant--especially with VLSI technology where routing costs dominate the power, time, and area required to implement a computation. Thus, the requirement is to design algorithms that support high degrees of concurrency, and at the same time to employ only simple, regular communication and control to ensure efficient implementation. [Ref. 4]

II. SYSTOLIC ARRAYS

A. BACKGROUND

It is clear today that developments in microelectronics have made a revolutionary impact on computer design [Ref. 2]. For example, integrated circuit technology has made a significant increase in the number and complexity of components that can now fit on a chip or a printed circuit board. In fact, with the component density presently doubling every one-to-two years, the notion of the million-transistor chip will soon be a reality [Ref. 3]. Commensurate with this major increase in chip density is the utilization of highly parallel computing structures which, almost by definition, implies a basic computational element repeated hundreds or thousands of times. This architectural style, which has structural properties suitable for VLSI implementation, reduces the design problem by several orders of magnitude. As a result, we are interested in high-performance parallel structures that can be implemented directly via very economical hardware devices [Ref. 2]. In other words, cost-effectiveness has always been, and will continue to be, a major concern in designing special-purpose VLSI systems; their cost must be low enough to justify their limited applicability. Furthermore, if a structure can truly be decomposed into a few types of building blocks

the forementioned attributes. Therefore, a systolic array is a natural architecture for the simple, regular, and cost-effective implementation of an RS encoder and decoder.

In an effort to assist the reader in simplicity and comprehension, this author has taken the pertinent information vital to the thesis and created a chapter for each. After systolic arrays are introduced in Chapter II the necessary fundamentals of finite fields for an understanding of Reed-Solomom codes is discussed in Chapters III and IV. In Chapter V a systolic array multiplier for finite fields is discussed and finally in Chapter VI the encoder and decoder for binary codes is described a well as the encoder and decoder for RS codes.

symbols. This representation is known as an RS (15,13) code, where the first integer depicts the total number of symbols in the codeword, and the second integer indicates the number of information symbols. It is the responsibility of the encoder to use the information symbols to generate the check or parity symbols for the codeword. The information symbols are treated as coefficients of a polynomial f(x),

$$f(x) = \sum_{i=1}^{2^J-1-2E} f_i x^{2^J-1-i}$$

where f_i is the i^{th} transmitted information symbol. The corresponding generator polynomial is known as g(x).

$$g(x) = \prod_{i=1}^{2E} (x+\beta^i)$$

Then, the 2E parity symbols are defined as the coefficients of the remainder of f(x)/g(x). Therefore, in the RS (15,13) code previously mentioned

$$g(x) = \prod_{i=1}^{2} (x+\beta^i)$$

$$= (x+\beta^1)(x+\beta^2)$$

$$= x^2+(\beta^1+\beta^2)x+\beta^3$$

$$= x^2+\beta^5 x+\beta^3$$

Furthermore, let us assume that the thirteen information symbols are β^6, β^1, β^8, β^2, β^4 β^5, β^{12}, β^7, β^9, β^{11}, β^{14}, β^3, β^{13}. Then,

$$f(x) = \sum_{i=1}^{13} f_i x^{15-i}$$

$$= f_1x^{14}+f_2x^{13}+f_3x^{12}f_4x^{11}+f_5x^{10}+f_6x^9+f_7x^8+f_8x^7+f_9x^6+f_{10}x^5 + f_{11}x^4+f_{12}x^3+f_{13}x^2$$

$$= \beta^6x^{14}+\beta^1x^{13}+\beta^8x^{12}+\beta^2x^{11}+\beta^4x^{10}+\beta^5x^9+\beta^{12}x^8+\beta^7x^7+\beta^9x^6+\beta^{11}x^5 + \beta^{14}x^4+\beta^3x^3+\beta^{13}x^2$$

Performing the required division, $f(x)/g(x)$

$$\begin{array}{r|l} & \beta^6x^{12}+\beta^6x^{11}+\beta^0x^{10}+\ldots+\beta^{14}x^2+\beta^9x+0 \\ \hline x^2+\beta^5x+\beta^3 & \beta^6x^{14}+\beta^1x^{13}+\beta^8x^{12}+\ldots+\beta^{14}x^4+\beta^3x^3+\beta^{13}x^2 \end{array}$$

$$\begin{array}{r} \beta^6x^{14}+\beta^{11}x^{13}+\beta^9x^{12} \\ \hline \beta^6x^{13}+\beta^{12}x^{12}+\beta^2x^{11} \\ \beta^6x^{13}+\beta^{11}x^{12}+\beta^9x^{11} \\ \hline \beta^0x^{12}+\beta^{11}x^{11}+\beta^4x^{10} \\ \beta^0x^{12}+\beta^5x^{11}+\beta^3x^{10} \\ \hline \vdots \\ \beta^{14}x^4+\beta^{14}x^3+\beta^{13}x^2 \\ \beta^{14}x^4+\beta^4x^3+\beta^2x^2 \\ \hline \beta^9x^3+\beta^{14}x^2+0x \\ \beta^9x^3+\beta^{14}x^2+\beta^{12}x \\ \hline \beta^{12}x \\ 0 \quad x \\ \hline \beta^{12}x+0 \end{array}$$

Hence, the remainder we seek is $\beta^{12}, 0$ and thus the corresponding 15-symbol codeword is $\beta^6\ \beta^1\ \beta^8\ \beta^2\ \beta^4\ \beta^5\ \beta^{12}\ \beta^7\ \beta^9\ \beta^{11}\ \beta^{14}\ \beta^3\ \beta^{13}\ \beta^{12} 0$ where the first thirteen symbols represent the information symbols and the last two symbols represent the parity symbols. [Ref. 9]

The architecture of the systolic implementation consists of a regular array of identical cells. Division is performed in a pipelined manner by simultaneously entering the highest order of terms of the f(x) and g(x) polynomials on the left most cell and generating the appropriate codeword on the far right, as depicted in Figure 4.2. In fact, a codeword can immediately follow the previous one without any interruption in the pipeline flow. Likewise, the control is also systolic. One control bit pipeline path will signal the start of a new codeword; another will signal the start of the division operation. Meanwhile, each cell of the array will hold one term of the quotient. As a result, if d represents the difference in degrees between two polynomials, then

$$d=[\deg f(x)-\deg g(x)]$$

and thus d+1 cells are required. For example,

$$\deg f(x) = 14$$

$$\underline{\deg g(x) = 2}$$

$$d = 12 \text{ (degree of quotient)}$$

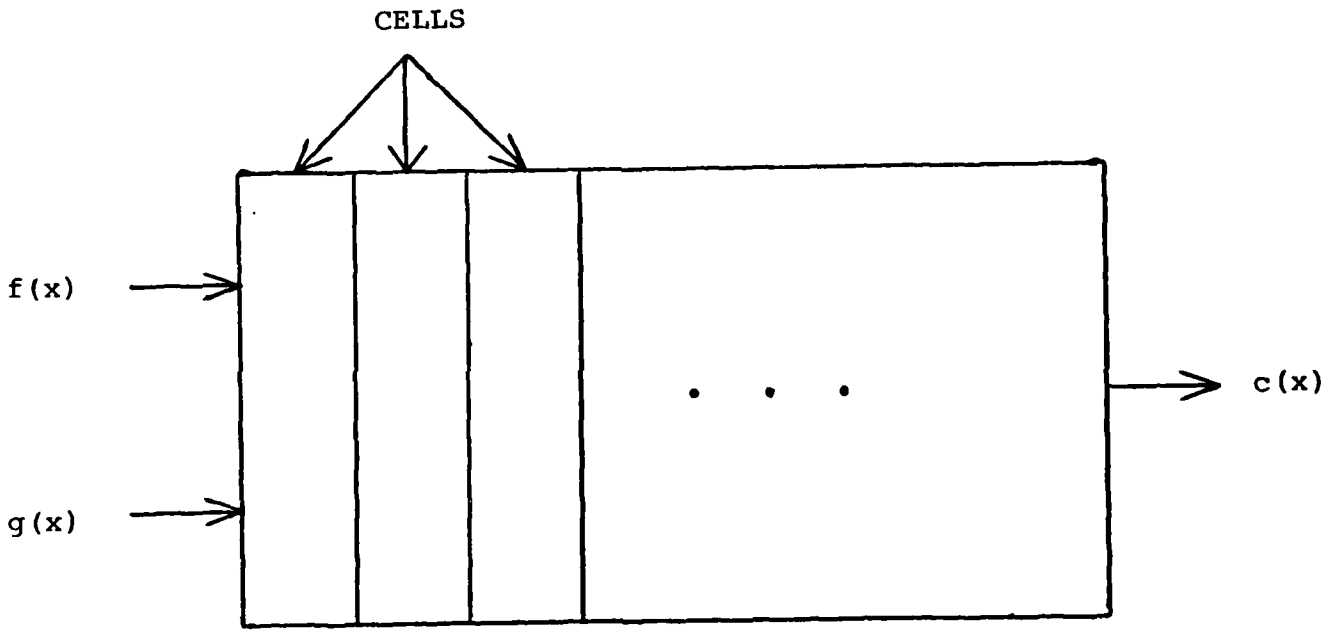


Figure 4.2 A Systolic Architecture

From our previous calculation, the quotient was ($\beta^6 x^{12} + \beta^6 x^{11} + \beta^0 x^{10} + \ldots + \beta^{14} x^2 + \beta^9 x + 0$). Since it consists of thirteen terms, thirteen cells would be needed. In general,

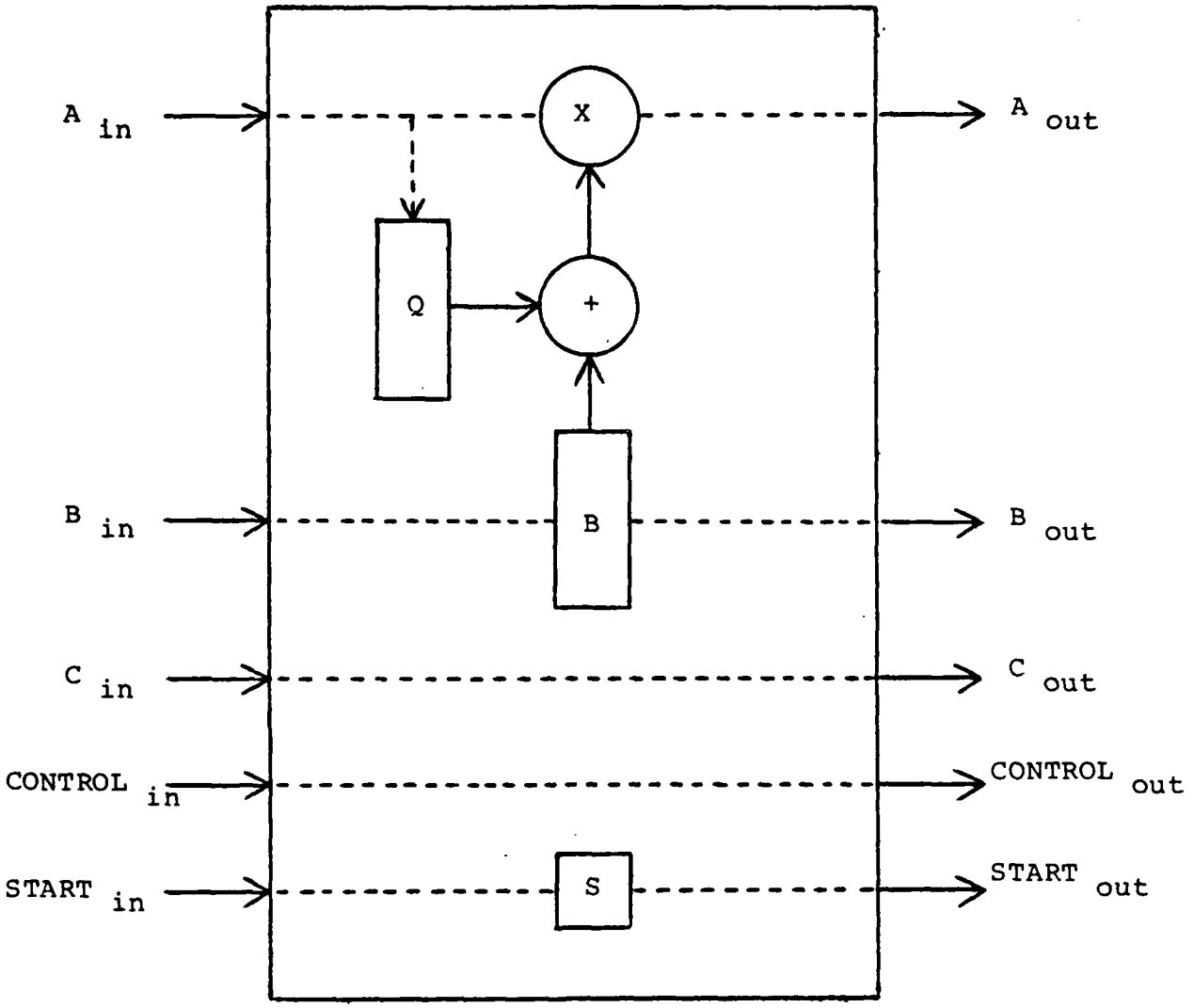
$$\deg f(x) = 2^J - 2$$

$$\underline{\deg g(x) = 2E}$$

$$d = 2^J - 2E - 2$$

and so the total number of cells required is d+1 or $2^J - 2E - 1$. [Ref. 9]

The operation of each cell is simple and regular. Essentially, it accomplishes one line of the normal division by initially determining the specific term of the quotient, multiplying by the divisor, subtracting the result from the dividend, and finally passing along the divisor and partial result to the next cell. More specifically, there are three J-bit data paths and two 1-bit control paths, as shown in Figure 4.3. The function of the C data path is to allow the information symbols to pass through the array unchanged while the other two data paths, A and B, are for the dividend and divisor, respectively. The register Q is set at the start of the division, and remains the same throughout the polynomial division of one block. The register B is used as a temporary storage device. While a control bit accompanies the first byte of information to signal the start of a new codeword a preceding start bit, one-half the rate of the control bit, initiates the division operation in



A: USED FOR DIVIDEND
B: USED FOR DIVISOR
C: USED FOR INFORMATION SYMBOLS
Q: DIVISION REGISTER
S: START REGISTER
CONTROL: USED TO START DIVISION
⊗: FINITE FIELD MULTIPLIER
⊕: FINITE FIELD ADDER

Figure 4.3 The Systolic Cell Structure

each cell. In short, the above architecture is simply a pipelined parallel processor which is composed of a systolic array of identical cells, each performing a finite-field multiplication and addition. Since the layout is simple and regular, it is easily replicated and economical to produce. [Ref. 9]

In Chapter VI the encoder and decoder for an RS code are described in greater detail with the encoding and decoding process carried out for a specific example.

V. IMPLEMENTATION THEORY

A. BACKGROUND

In this chapter we look at the theoretical concepts behind the systolic implementation of an encoder and decoder. We then apply these concepts to the actual implementation in the subsequent chapter. There, the binary case is initially presented because of its simple architecture and ease of understanding. It is then followed by the more intricate and complex Reed-Solomon case.

We also, in this chapter, discuss in-depth the design of a systolic array multiplier used in the RS encoder. Unlike the binary case which deals only with the elements 0 and 1 in the complete codeword, the Reed-Solomon codeword will contain symbols which lie in a larger field than GF(2). As a result, the systolic array multiplier is increasingly more detailed and complicated than in the binary design which simply uses a primitive binary shift register scheme.

B. PRIMITIVE BINARY SHIFT REGISTER DESIGN

A primitive binary shift register is a series of registers each capable of containing a zero or a one. The contents of the register all shift on a designated time signal via use of an external clock. The contents of the newest stage of the register is defined as a function of the

current contents of the register. Because these shift registers utilize this feedback property they are commonly referred to as feedback shift registers or primitive shift registers since the feedback is usually described by a primitive polynomial [Ref. 10]. For example, the diagram in Figure 5.1 describes a primitive shift register composed of four registers, labeled 1, x, x^2, x^3 and one modulo 2 adder situated between registers 1 and x. Each register is capable of storing one bit of binary information, i.e., a "1" or a "0". The all zero contents of the register is typically prohibited. This restriction is placed on the primitive shift register to ensure a change of state when a new clock signal is received. The register is allowed to step from state to state, therefore the length of a primitive cycle is independent of its initial state and is equal to 2^m-1. The primitive shift register of Figure 5.1 will move through 15 distinct binary patterns before repeating (see Table II). This primitive shift register is said to have a cycle length of 2^4-1 or 15. Moreover, since all nonzero patterns are included in the cycle, it is called a maximum-length cycle. In general, a primitive shift register composed of m stages will generate a maximum-length cycle of period 2^m-1. It is possible for each value of m to determine a primitive feedback function for the shift register so that a maximum-length shift register sequence of period 2^m-1 is generated.

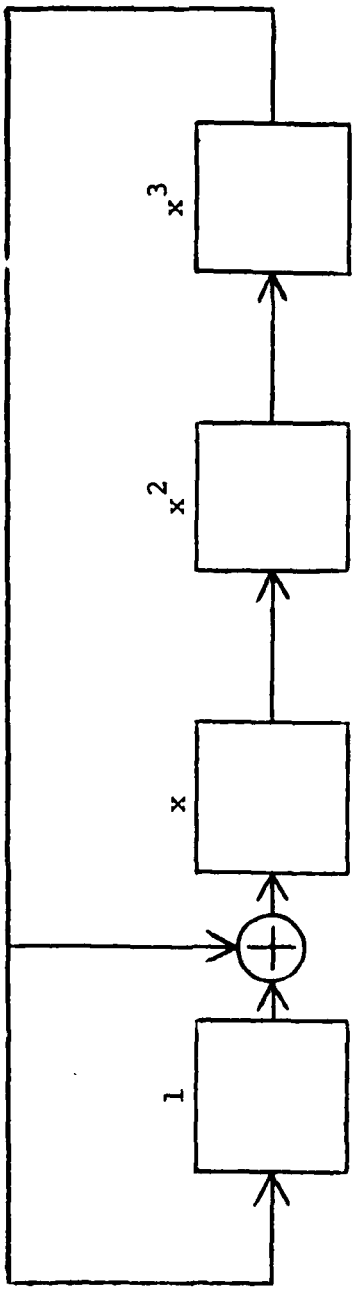


Figure 5.1 A 4-Stage Primitive Shift Register

TABLE II

REGISTER CONTENTS AFTER SUCCESSIVE CLOCK SIGNALS

TIME (t)	REGISTER CONTENTS	BETA POLYNOMIAL
t = 0	1 0 0 0	1
t = 1	0 1 0 0	β
t = 2	0 0 1 0	β^2
t = 3	0 0 0 1	β^3
t = 4	1 1 0 0	β^4
t = 5	0 1 1 0	β^5
t = 6	0 0 1 1	β^6
t = 7	1 1 0 1	β^7
t = 8	1 0 1 0	β^8
t = 9	0 1 0 1	β^9
t = 10	1 1 1 0	β^{10}
t = 11	0 1 1 1	β^{11}
t = 12	1 1 1 1	β^{12}
t = 13	1 0 1 1	β^{13}
t = 14	1 0 0 1	β^{14}
t = 15	1 0 0 0	β^{15}

Maximum-length cycles and maximum-length sequences have broad applications in data communication systems and computer simulation while primitive shift registers designed as division circuits have applications in coding theory [Ref. 10]. It is the objective of this chapter to utilize the concepts of the latter to propose an RS encoder and decoder.

In order to generate a maximum-length cycle or sequence we need to understand the necessary component connections given a primitive polynomial. That is, given an arbitrary primitive polynomial, how do we design the shift register? For the example of Figure 5.1, assume $p(x)=1+x+x^4$ is a primitive polynomial over GF(2). We can consider $GF(2^4)$ as an algebra of polynomials modulo $p(x)=1+x+x^4$ and design a register to produce a pattern cycle of length 2^4-1. Using four delay units (since we need a register unit for the coefficient of each term x^t with $0 \leq t \leq 3$) we need only decide how the primitive polynomial affects the feedback to know where to place the modulo 2 adder components and where to make the necessary circuit connections. The feedback is the coefficient of x^4, but in this polynomial algebra $x^4=1+x$. Thus, the feedback goes to the registers which contain the coefficients of the x^0 and x^1 terms. Making these connections and supplying the modulo 2 adder component where we have two inputs to the register, we arrive at the shift register given in Figure 5.1. Then each step of the

register is equivalent to multiplying the contents of the register by the primitive element β. Thus, the sequence of contents are the powers of β modulo $(1+\beta+\beta^4)$. In this way multiplication of the elements of the field is produced simply as described in Chapter III and the powers of β are as given as in Table I of Chapter III.

C. CODING THEORY

Suppose that we wish to transmit a sequence of binary digits across a noisy channel. If we send a one a one will probably be received and if we send a zero a zero will also probably be received. Occasionally, the channel noise will cause a transmitted one to be received as a zero or a transmitted zero to be received as a one. Although we are unable to prevent the channel from generating such errors, we can reduce their undesirable effects with the use of coding [Ref. 5]. The basic idea is simple. A set of k message digits which we wish to transmit is concatenated to r check digits. The entire block of n=k+r channel digits then forms the transmitted codeword. Assuming that the channel noise changes sufficiently few of these n transmitted channel digits, the redundancy afforded by r check digits provides the receiver with sufficient information to detect and correct the channel errors. Figure 5.2 illustrates the basic idea of the encoding process for an (n,k) encoder with n=15 and k=11. The codeword is constructed in such a way

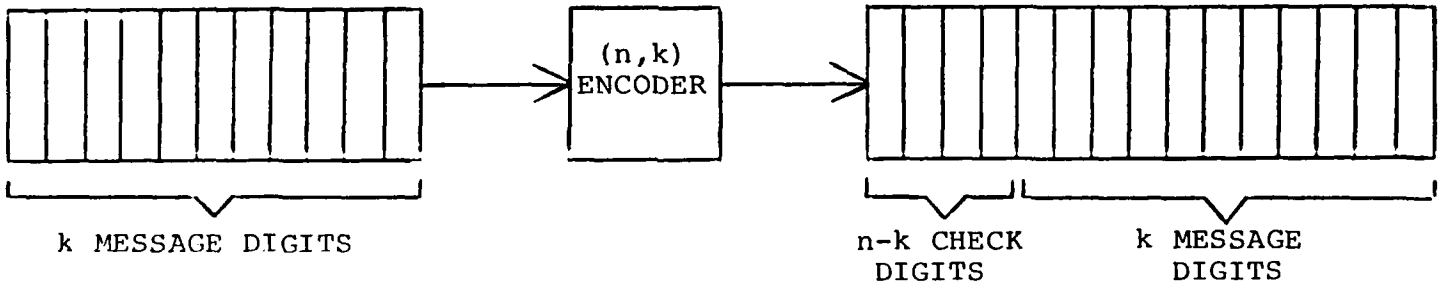


Figure 5.2 The Encoding Process

$r_i^*=0$, u_i retains its value. Two principle operations of the system are the following:

$$e_{i+1} \leftarrow\!\!- (g_i^* d_i) \oplus e_i^*$$

$$g_{i+1}^* \leftarrow\!\!- (u_i h_i^*) \oplus (g_i^* t_i^*)$$

where $0 \leq i \leq 3$, $\oplus$ denotes Exclusive-OR operation, i.e., modulo-2 addition, and the backwards arrow denotes the substitution operation.

A comparison of the procedure in Table V and the structure in Figures 5.3 and 5.4 yields the following facts: The signal u_i in L_i is equal to $a_3^{(i)}$ in $A\beta^i$. The signal g_i^* is equal to $a_n^{(i)}$ in $A\beta^i$ for some n. The signal e_i^* is equal to the partial sum AB+C.

The multiplier in Figure 5.3 can be generalized to the finite field $GF(2^m)$ by simply concatenating m identical cells. Furthermore, additional registers and control signals would be required if the b_i's are fed serially into the system in the same manner as the a_i's. [Ref. 7]

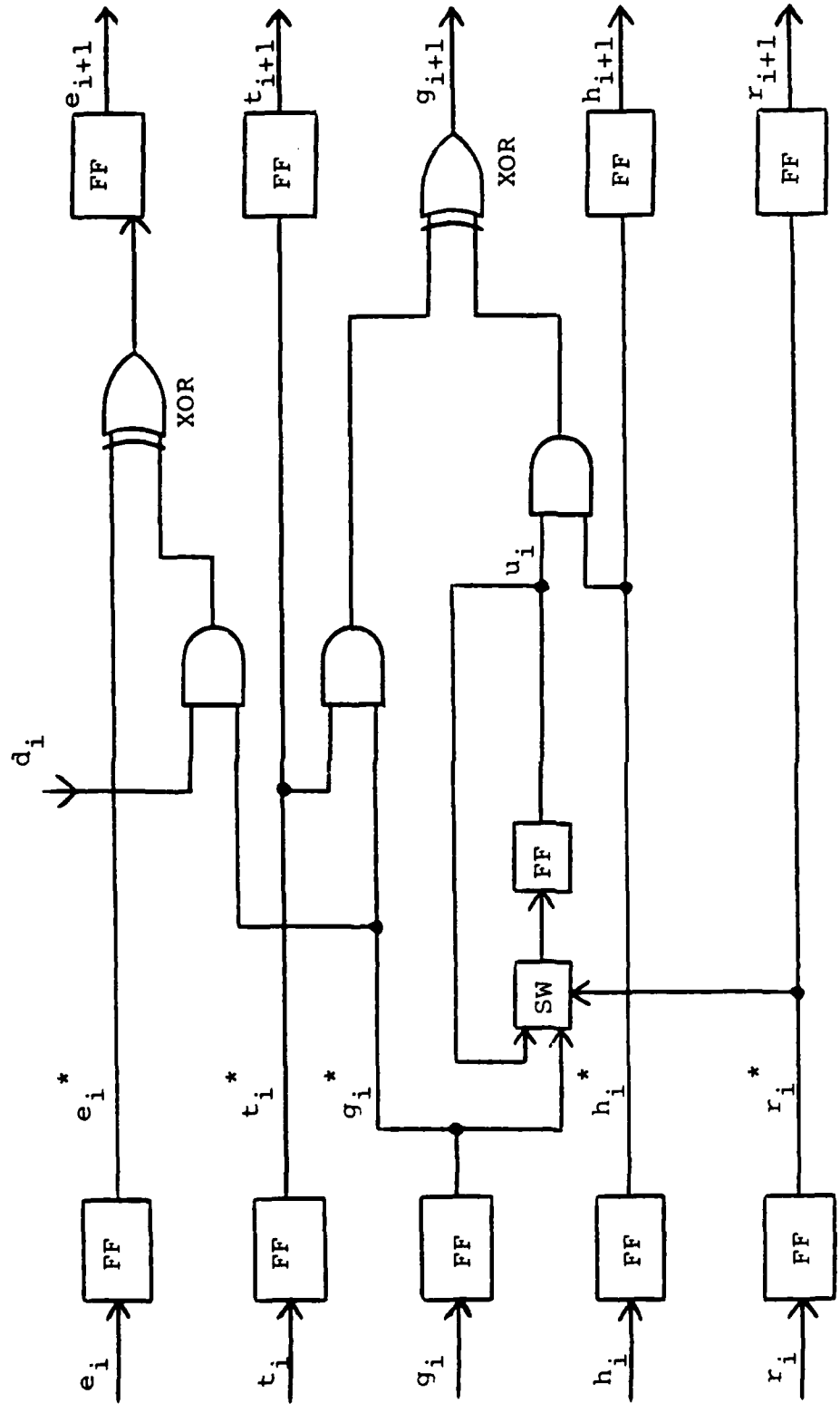


Figure 5.4 The Circuit of the Cell L_i

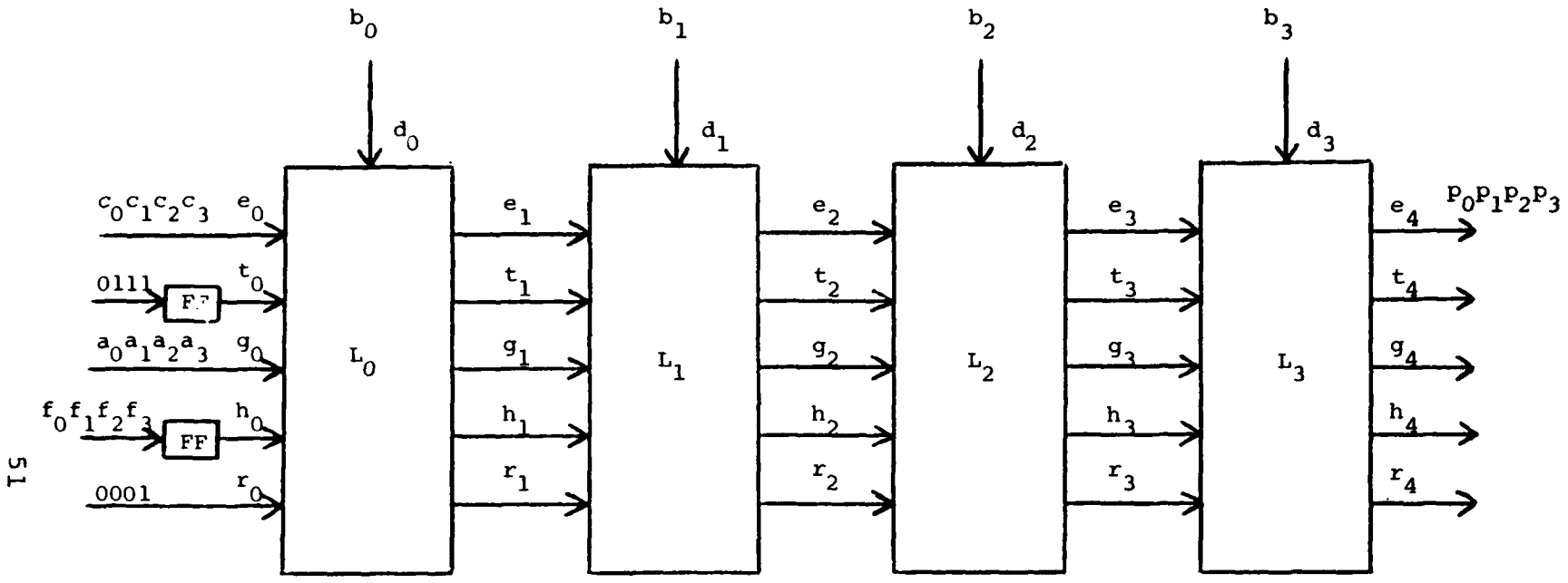


Figure 5.3 A Systolic Multiplier for the Finite Field GF(2^4)

TABLE V

COMPUTATION OF $P = AB + C$ IN $GF(2^4)$

STEP NUMBER	OPERATIONS	
1	$p_3^{(0)} = c_3$	$a_3^{(0)} = a_3$
2	$p_3^{(1)} = p_3^{(0)} + a_3^{(0)} b_0$, $p_2^{(0)} = c_2$	$a_2^{(0)} = a_2$
3	$p_2^{(1)} = p_2^{(0)} + a_2^{(0)} b_0$, $p_1^{(0)} = c_1$	$a_1^{(0)} = a_1$, $a_3^{(1)} = a_2^{(0)} + a_3^{(0)} f_3$
4	$p_3^{(2)} = p_3^{(0)} + a_3^{(1)} b_1$, $p_1^{(1)} = p_1^{(0)} + a_1^{(0)} b_0$, $p_0^{(0)} = c_0$	$a_0^{(0)} = a_0$, $a_2^{(1)} = a_1^{(0)} + a_3^{(0)} f_2$
5	$p_2^{(2)} = p_2^{(1)} + a_2^{(1)} b_1$, $p_0^{(1)} = p_0^{(0)} + a_0^{(0)} b_0$,	$a_1^{(1)} = a_0^{(0)} + a_3^{(0)} f_1$, $a_3^{(2)} = a_2^{(1)} + a_3^{(1)} f_3$
6	$p_3^{(3)} = p_3^{(2)} + a_3^{(2)} b_2$, $p_1^{(2)} = p_1^{(1)} + a_1^{(1)} b_1$,	$a_0^{(1)} = a_3^{(0)} f_0$, $a_2^{(2)} = a_1^{(1)} + a_3^{(1)} f_2$
7	$p_2^{(3)} = p_2^{(2)} + a_2^{(2)} b_2$, $p_0^{(2)} = p_0^{(1)} + a_0^{(1)} b_1$,	$a_1^{(2)} = a_0^{(1)} + a_3^{(1)} f_1$, $a_3^{(3)} = a_2^{(2)} + a_3^{(2)} f_3$
8	$p_3 = p_3^{(4)} = p_3^{(3)} + a_3^{(3)} b_3$, $p_1^{(3)} = p_1^{(2)} + a_1^{(2)} b_2$,	$a_0^{(2)} = a_3^{(1)} f_0$, $a_2^{(3)} = a_1^{(2)} + a_3^{(2)} f_2$
9	$p_2 = p_2^{(4)} = p_2^{(3)} + a_2^{(3)} b_3$, $p_0^{(3)} = p_0^{(2)} + a_0^{(2)} b_2$,	$a_1^{(3)} = a_0^{(2)} + a_3^{(2)} f_1$
10	$p_1 = p_1^{(4)} = p_1^{(3)} + a_1^{(3)} b_3$,	$a_0^{(3)} = a_3^{(2)} f_0$
11	$p_0 = p_0^{(4)} = p_0^{(3)} + a_0^{(3)} b^3$,	

From equation (3), we obtain

$$a_n(k) = a_{n-1}(k-1) + a_{m-1}(k-1)f_n \quad \text{for } 1 \leq n \leq m-1$$
$$a_0(k) = a_{m-1}(k-1)f_0 \tag{4}$$

Table V indicates the step-by-step procedure for computing P=AB+C in GF(2^4). In Table V $a_n(k)$, b_n, c_n, f_n, and p_n are the n-th bits of $A\beta^k$, B, C, F, and P, respectively, where F is the primitive polynomial and $p_n(i)$ is the partial sum of p_n.

Figure 5.3 depicts the systolic multiplier for our given finite field. The primitive polynomial is $F=f_3\beta^3+f_2\beta^2+f_1\beta+f_0$. Input d_n receives the bit b_n of B. The n-th bits c_n, a_n and f_n, of C, A, and F, respectively, are received serially at inputs e_0, g_0, and h_0. Two control signals, START (0001) and END (0111) are used in the design with inputs r_0 and t_0 receiving the signals, respectively.

Output e_4 serially transmits the n-th bit, p_n, of the result P out of the system. The order of the inputs and the outputs is also shown in Figure 5.3. The flip-flops (FF) associated with inputs t_0 and h_0 are used for the purpose of synchronization.

The circuit of cell L_i is shown in Figure 5.4. The operation of the flip-flops in the system is synchronized implicitly by a clock signal. When $r_i^*=1$, $u_i=g_i^*$ at the next time unit (through switch SW). Additionally, when

$$P = \sum_{k=0}^{m-1} (A\beta^k)b_k = \sum_{k=0}^{m-1} \left(\sum_{n=0}^{m-1} a_n^{(k)} \beta^n \right) b_k$$

$$= \sum_{n=0}^{m-1} \left(\sum_{n=0}^{m-1} a_n^{(k)} b_k \right) \beta^n \qquad (1)$$

where $a_n^{(k)}$ is the coefficient of β^n in $A\beta^k$, i.e., $A\beta^k = a_{m-1}^{(k)}\beta^{m-1}+\ldots+a_1^{(k)}\beta+a_0^{(k)}$ for $0 \leq k \leq m-1$. From equation (1) we obtain $p_n=a_n^{(0)}b_0+a_n^{(1)}b_1+\ldots+a_n^{(m-2)}b_{m-2}+a_n^{(m-1)}b_{m-1}$.

The computation of $A\beta^k$ can be performed recursively on k for $0 \leq k \leq m-1$. Initially for $k=0$, $A\beta^0=A$, i.e., $a_n^{(0)}=a_n$ for $0 \leq n \leq m-1$. For $1 \leq k \leq m-1$,

$$A\beta^k = (A\beta^{k-1})\beta = \sum_{n=0}^{m-1} a_n^{(k-1)} \beta^{n+1} \qquad (2)$$

$$= a_{m-1}^{(k-1)} \beta^m + \sum_{n=1}^{m-1} a_{n-1}^{(k-1)} \beta^n$$

Substituting $\beta^m=f_{m-1}\beta^{m-1}+\ldots+f_1\beta+f_0$ into equation (2), yields

$$A\beta^k = \sum_{n=1}^{m-1} (a_{n-1}^{(k-1)}+a_{m-1}^{(k-1)} f_n)\beta^n + a_{m-1}^{(k-1)} f_0 \qquad (3)$$

$GF(2^m)$ of 2^m elements, where A, B, and C are arbitrary elements of the field. The multiplier is a serial-in, serial-out, one-dimensional systolic array which requires m basic cells. To perform an isolated computation the multiplier requires 3m time units, however, the average time per computation is only m time units if a number of computations are carried out consecutively. Because the architecture is simple and regular and possesses the desirable properties of concurrency and modularity, it is well suited for VLSI implementation. [Ref. 7]

Consider the nonzero elements of $GF(2^m)$. They can be represented as the powers of β, a primitive element of the field as discussed in Chapter III. Since $F(\beta)=0$, $\beta^m=f_{m-1}\beta^{m-1}+\ldots+f_1\beta+f_0$, where the coefficients f_i are determined by the polynomial f(x) which β satisfies. Therefore an element of $GF(2^m)$ is of the form $a_{m-1}\beta^{m-1}+\ldots+a_1\beta+a_0$ where $a_i \in GF(2)$ for $0 \leq i \leq m-1$. In the following discussion, the polynomial representation is used to represent the finite field $GF(2^m)$.

Let $A=a_{m-1}\beta^{m-1}+\ldots+a_1\beta+a_0$ and $B=b_{m-1}\beta^{m-1}+\ldots+b_1\beta+b_0$ be two elements in $GF(2^m)$. Then $A+B=S_{m-1}\beta^{m-1}+\ldots+S_1\beta+S_0$, where $S_i=a_i+b_i \pmod 2$ for $0 \leq i \leq m-1$. Therefore addition in $GF(2^4)$ is realized easily by m independent Exclusive-OR gates.

Suppose $P=p_{m-1}\beta^{m-1}+\ldots+p_1\beta+p_0$ is the product of A and B, i.e., P=AB. Then P can be written as follows:

error patterns depends upon finding higher powers of β which belong to cyclotomic cosets for the smaller powers of β which belong to the code for the designed error correcting distance. The tables of cyclotomic cosets for $GF(2^5)$, $GF(2^6)$ show that β^9 belongs to β^5, β^{17} belongs to β^5 and β^{19} belongs to β^{13}, etc. See [Ref. 11] for further discussion of the error correcting capabilities of given error correcting codes.

E. SYSTOLIC ARRAY MULTIPLIER

As mentioned earlier in this chapter, the systolic array multiplier used in the generation of Reed-Solomon codewords is much more complex than in the binary case. In this section, we discuss the design of a systolic array multiplier developed by Yeh, Reed, and Truong [Ref. 7] to assist us in our implementation of an RS encoder.

According to [Ref. 7] several circuits have been proposed to realize multiplication in $GF(2^m)$. Unfortunately, these circuits are not suited for use in VLSI systems, due to irregular wire routing, complicated control problems, nonmodular structure and lack of concurrency. The systolic array multiplier of [Ref. 7] performs the multiplication in the field $GF(2^m)$ which overcomes some of these ... ited attributes.

The systolic architecture is developed for performing the product-sum computation, AB+C, in the finite field

TABLE IV

MINIMAL POLYNOMIALS OF ELEMENTS IN $GF(2^4)$

$$M^{(1)}(x) = M^{(2)}(x) = M^{(4)}(x) = M^{(8)}(x) = 1+x+x^4$$

$$M^{(3)}(x) = M^{(6)}(x) = M^{(12)}(x) = M^{(9)}(x) = 1+x+x^2+x^3+x^4$$

$$M^{(5)}(x) = M^{(10)}(x) = 1+x+x^2$$

$$M^{(7)}(x) = M^{(14)}(x) = M^{(13)}(x) = M^{(11)}(x) = 1+x^3+x^4$$

which is analogous to the generator polynomial g(x) in our generic architecture of the previous chapter. Moreover, by utilizing various techniques beyond the scope of this thesis, we may determine all the minimal polynomials of elements in $GF(2^4)$, as depicted in Table IV. Using this table we may construct all the Reed-Solomon codes of block length 15 which correct t or fewer channel errors. These codes have the following generator polynomials:

$$t=1 \quad g(x)=M^{(1)}(x)=1+x+x^4$$

$$t=2 \quad g(x)=M^{(1)}(x)*M^{(3)}(x)=1+x^4+x^6+x^7+x^8$$

$$t=3 \quad g(x)=M^{(1)}(x)*M^{(3)}(x)*M^{(5)}=1+x+x^2+x^4+x^5+x^8+x^{10}$$

Hence, the t-error correcting RS code of block length n is then the cyclic code whose generator polynomial is the product of the distinct minimal polynomials of β, β^2, β^3, ..., β^{2t-1}, β^{2t} [Ref. 5]. Of noteworthy interest is the fact that an RS code over $GF(2^5)$ which is designed to correct up to 4 errors is also able to correct 5 errors. This is because $M^{(9)}(x)$, the minimal polynomial of β^9, is identical to $M^{(5)}(x)$, the minimal polynomial of β^5. Similarly, the 6 error-correcting RS code is identical to the 7 error-correcting code just as the 8-to-14 error correcting codes of length 31 are all identical to the 15 error-correcting code. In a similar way, codes over $GF(2^6)$ and $GF(2^7)$ are sometimes able to correct more errors than they are designed to correct. The ability to correct these extra

TABLE III

CYCLOTOMIC COSETS

OVER GF(2^3)

$C_0 = \{0\}$

$C_1 = \{1,2,4\}$

$C_3 = \{3,6,5\}$

OVER GF(2^5)

$C_0 = \{0\}$

$C_1 = \{1,2,4,8,16\}$

$C_3 = \{3,6,12,24,17\}$

$C_5 = \{5,10,20,9,18\}$

$C_7 = \{7,14,28,25,19\}$

$C_{11} = \{11,22,13,26,21\}$

$C_{15} = \{15,30,29,27,23\}$

OVER GF(2^6)

$C_0 = \{0\}$

$C_1 = \{1,2,4,8,16,32\}$

$C_3 = \{3,6,12,24,48,33\}$

$C_5 = \{5,10,20,40,17,34\}$

$C_7 = \{7,14,28,56,49,35\}$

$C_9 = \{9,18,36\}$

$C_{11} = \{11,22,44,25,50,37\}$

$C_{13} = \{13,26,52,41,19,38\}$

$C_{15} = \{15,30,60,57,51,39\}$

$C_{21} = \{21,42\}$

$C_{23} = \{23,46,29,58,53,43\}$

$C_{27} = \{27,54,45\}$

$C_{31} = \{31,62,61,59,55,47\}$

$1+x+x^4$. Likewise β^3, β^6, β^{12}, $\beta^{24}=\beta^9$ also have the same minimal polynomial $1+x+x^2+x^3+x^4$. We see that the powers of β fall into disjoint sets, called cyclotomic cosets. In fact, all β^j which are elements of the same cyclotomic coset have the same minimal polynomial. The cyclotomic coset containing β^s consists of the following powers of β:

$$\{s,\ 2s,\ 2^2s,\ 2^3s,\ \ldots,\ 2^{m_s-1}s\}$$

where m_s is the smallest positive integer such that $2^{m_s}s \equiv s \pmod{2^m-1}$ [Ref. 11]. For example, the cyclotomic cosets over $GF(2^4)$ are:

$$C_0 = \{0\}$$
$$C_1 = \{1,2,4,8\}$$
$$C_3 = \{3,6,12,9\}$$
$$C_5 = \{5,10\}$$
$$C_7 = \{7,14,13,11\}$$

Other cyclotomic coset decompositions for various values of m are listed in Table III.

If we let $M^{(i)}(x)$ represent the minimal polynomial of $\beta^i \varepsilon GF(p^m)$, it follows that if i is in the cyclotomic coset C_s, then

$$M^{(i)}(x) = \prod_{j \varepsilon C_s} (x+\beta^j)$$

polynomials for the elements β, β^3, β^5, and β^7 are given in the following table.

Element	Minimal Polynomial
0	x
1	$1+x$
β	$1+x+x^4$
β^3	$1+x+x^2+x^3+x^4$
β_5	$1+x+x^2$
β^7	$1+x^3+x^4$

Furthermore, in $GF(2^m)$ β^i and β^{2i} have the same minimal polynomial. In general, if β^i is a root of a minimal polynomial then so is β^{pi} (where p is the characteristic of the ground field GF(p); in this case p=2). To illustrate, let us substitute the elements β and β^2 into our minimal polynomial $1+x+x^4$. Upon substitution of β we obtain $1+\beta+\beta^4$. Thus in $GF(2^4)$ $\beta^4=\beta+1$ and $M(\beta)=0$. Likewise, upon substituting β^2 for x in the same minimal polynomial we obtain $1+\beta^2+\beta^8$, which in $GF(2^4)$ is also zero as can be seen in Table I of Chapter III. Elements of the field with the same minimal polynomial are called conjugates. In the same way, the imaginary roots i and -i are referred to as conjugate complex numbers--they both have the same minimal polynomial x^2+1 over the reals [Ref. 11].

From our preceding discussion, it is clear that β, β^2, $(\beta^2)^2=\beta^4$, $(\beta^4)^2=\beta^8$ all have the same minimal polynomial

that the message digits appear at the far right. The error correcting capability of the generated code depends upon the number of check bits added. To illustrate, the binary code constructed using the encoder of Figure 5.2 is capable of correcting one error when, for example, $n=2^m-1$, $k=2^m-1-m$ for each integer $m > 2$, the so called Hamming single-error-correcting code.

D. MINIMAL POLYNOMIALS

In order for a code to correct every pattern of t or fewer channel errors, the codewords must be generated by a polynomial whose length is the product of at least t distinct minimal polynomials [Ref. 5]. Occasionally, extra error correcting capability is possessed by words of a code beyond the designed capacity of the code. To understand this situation and the general error correcting capacity of the code, it is necessary that we discuss some of the mathematical concepts and properties that comprise minimal polynomials before discussing the actual implementation.

A minimal polynomial for a primitive element β over GF(p) is the lowest degree irreducible monic (has leading coefficient 1) polynomial M(x) with coefficients from GF(p) such that $M(\beta)=0$ [Ref. 11]. For example, the Galois field $GF(2^4)$ is constructed using the primitive element β, the root of the irreducible polynomial $1+x+x^4$. Then the minimal

VI. IMPLEMENTATION

A. BINARY ENCODER

In this section we discuss the encoding process for a binary code and utilize a primitive shift register design to implement both a single-error-correcting binary encoder and a double-error-correcting binary encoder.

1. Encoding Process

As discussed in Chapter V, an (n,k) code can be generated with a polynomial of degree n-k. If the polynomial is primitive of degree r and $n=2^r-1$, the code can be encoded and decoded with primitive shift registers. Hence, we restrict our attention solely to the case of primitive polynomials.

We illustrate this procedure by generating the (15,11) binary code using the primitive polynomial $p(x)=1+x+x^4$. Here $n-k=4$, $r=4$, $n=2^4-1=15$, and $k=11$. The encoding process for the 11-bit message 10101010101 proceeds as in the example below.

Example of Encoding Process:

Message = 10101010101

1) Represent the message as a polynomial. $m(x)=1+x^2+x^4+x^6+x^8+x^{10}$

2) Multiply m(x) by x^{n-k} to shift the message digits to the far right. $x^4m(x)=x^4+x^6+x^8+x^{10}+x^{12}+x^{14}$

3) Calculate the remainder when $x^{n-k}m(x)$ is divided by $p(x)$. $r(x)=1+x+x^3$

4) Form the code polynomial as the sum $x^{n-k}m(x)+r(x)$, a multiple of $p(x)$. $c(x)=1+x+x^3+x^4+x^6+x^8+x^{10}+x^{12}+x^{14}$

Code Word = 110110101010101

Note that codewords in this code are formed as multiples of the primitive generating polynomial $p(x)$. As $p(x)$ is of degree r there are $n-r=k$ information symbols which can be chosen freely and then r check symbols are chosen so that the resulting codeword satisfies this criteria, namely that the codewords are multiples of the generator polynomial. In other words, the check digits are the coefficients of the remainder $r(x)$ upon division of $x^{n-k}m(x)$ by $p(x)$ as shown below.

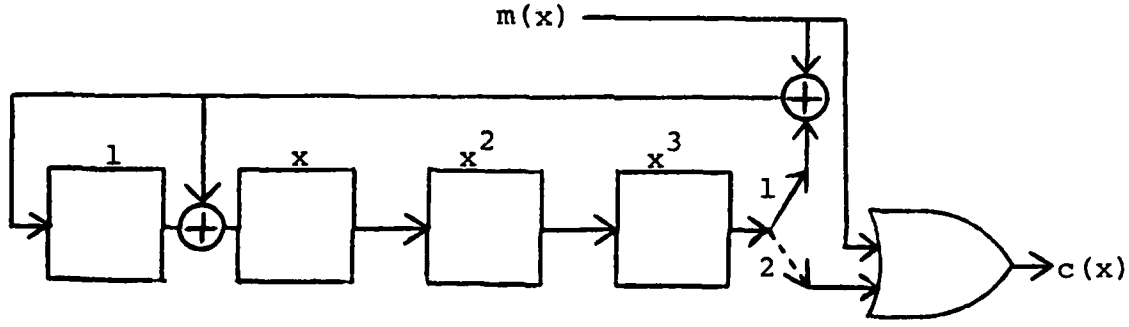
$$
\begin{array}{r|l}
 & x^{10}+x^8+x^7+x^5+x^4+x^3+1 \\
x^4+x+1 & x^{14}+x^{12}+x^{10}+x^8+x^6+x^4 \\
 & \underline{x^{14}+x^{11}+x^{10}} \\
 & \quad x^{12}+x^{11}+x^8 \\
 & \quad \underline{x^{12}+x^9+x^8} \\
 & \quad\quad x^{11}+x^9+x^6 \\
 & \quad\quad \underline{x^{11}+x^8+x^7} \\
 & \quad\quad\quad x^9+x^8+x^7+x^6+x^4 \\
 & \quad\quad\quad \underline{x^9+\quad\quad +x^6+x^5} \\
 & \quad\quad\quad\quad x^8+x^7+x^5+x^4 \\
 & \quad\quad\quad\quad \underline{x^8\quad +x^5+x^4} \\
 & \quad\quad\quad\quad\quad x^7 \\
 & \quad\quad\quad\quad\quad \underline{x^7+x^4+x^3} \\
 & \quad\quad\quad\quad\quad\quad x^4+x^3 \\
 & \quad\quad\quad\quad\quad\quad \underline{x^4+x+1} \\
 & \quad\quad\quad\quad\quad\quad\quad x^3+x+1
\end{array}
$$

2. Single-Error-Correcting Binary Encoder

By utilizing the previously discussed concepts, we may now describe the encoding process of the binary (15,11) code as implemented in a primitive shift register shown in Figure 6.1. By simply feeding in the message m(x) at the x^4-stage we are able to simulate the effect of multiplying m(x) by x^4. The switch remains in position 1 as m(x) is fed completely into the shift register. The shift register computes the remainder when $x^4m(x)$ is divided by p(x) as the shift register is in essence a division circuit. The register contents after the information bits have all been fed into the register is the remainder after division of the information polynomial by the generator polynomial p(x). In the example the remainder is 1101=$1+x+x^3$. The switch is then changed to position 2 to allow the check digits to follow the message digits producing the coded output 110110101010101 for the example given. [Ref. 10]

3. Double-Error-Correcting Binary Encoder

To design a double-error-correcting binary encoder to correct up to two errors, additional redundancy must be added. Since we are now concerned with correction of up to two errors the generator polynomial is the product of the two distinct minimal polynomials $M^{(1)}(x)$ and $M^{(3)}(x)$ as described in the previous chapter. Their product is the polynomial $1+x^4+x^6+x^7+x^8$. The implementation of the encoder is carried out in essentially the same manner as its



TIME (t)		REGISTER CONTENTS	OUTPUT
t = 0	INITIAL	0 0 0 0	--
t = 1		1 1 0 0	1
t = 2		0 1 1 0	0
t = 3		1 1 1 1	1
t = 4		1 0 1 1	0
t = 5		0 1 0 1	1
t = 6	SWITCH IN	1 1 1 0	0
t = 7	POSITION 1	1 0 1 1	1
t = 8		1 0 0 1	0
t = 9		0 1 0 0	1
t = 10		0 0 1 0	0
t = 11		1 1 0 1	1
t = 12		0 1 1 0	1
t = 13	SWITCH IN	0 0 1 1	0
t = 14	POSITION 2	0 0 0 1	1
t = 15		0 0 0 0	1

Figure 6.1 A Single-Error-Correcting Binary Encoder

single-error counterpart. The encoder is presented in Figure 6.2. Now n=15 and k=15-8=7 so that there are a smaller number of codewords (2^7) in this more powerful code. As the error correcting capability of the code increases, the number of information bits correspondingly decreases.

B. REED-SOLOMON ENCODER

In this section we draw upon the work of Liu [Ref. 8] and our acquired knowledge of finite field theory and Reed-Solomon codes to produce an RS encoder.

As discussed in Chapter IV, an RS codeword has (2^J-1) symbols each of which is J-bits wide. Of the (2^J-1) symbols there are (2^J-1-2E) information symbols and 2E parity-check symbols, where E is the number of symbol-errors the RS code is able to correct. If we treat the (2^J-1-2E) information symbols as the coefficients of the polynomial

$$f(x) = \sum_{i=1}^{2^J-1-2E} f_i x^{2^J-1-i} = f_1 x^{2^J-2} + f_2 x^{2^J-3} + \ldots + f_{2^J-1-2E} x^{2E}$$

then the 2E parity-check symbols can be obtained as the coefficients of the remainder of f(x)/g(x) where g(x) is the generator polynomial of the code. Usually, g(x) is defined as

$$g(x) = \prod_{i=1}^{2E} (x+\beta^i) = \sum_{j=0}^{2E} g_j x^j$$

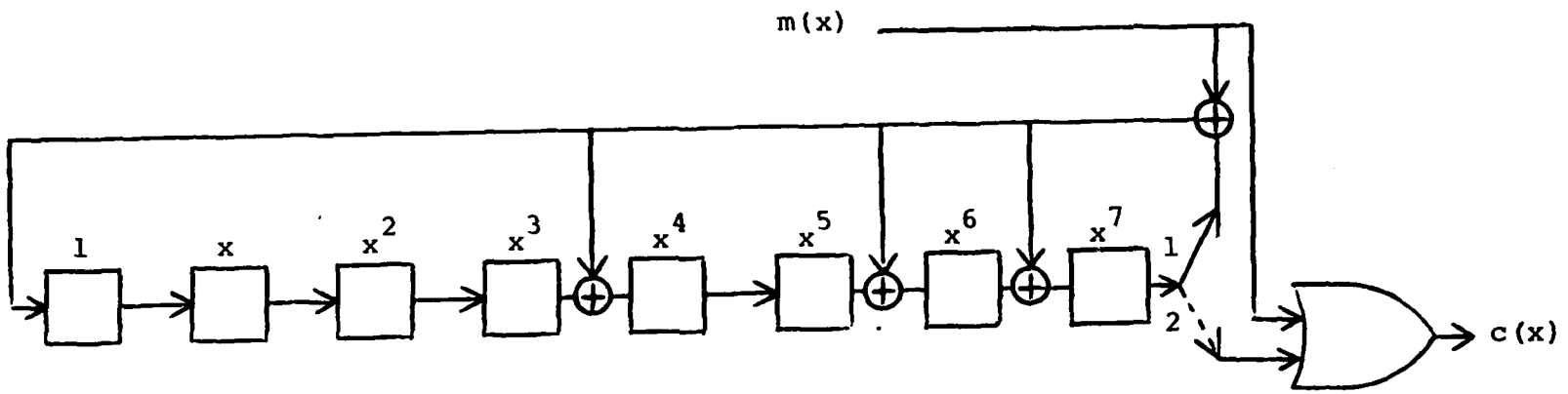


Figure 6.2 A Double-Error-Correcting Binary Encoder

where β is a primitive element of the Galois field $GF(2^J)$ and g_j's are the coefficients of g(x) with $g_{2E}=1$.

A diagram of the RS encoder which generates the remainder of f(x)/g(x) is given in Figure 6.3. It is composed of 2E systolic array multipliers, 2E "exclusive-or" adders, and 2E shift registers. The coefficients of the generator polynomial g(x) are fed into their respective systolic multipliers where the finite field multiplication A*B occurs, as discussed in Chapter V. Upon completion the partial product is "exclusive-or'ed" with the contents of C of the previous shift register and distributed down the line to the next shift register in a pipeline fashion. The switches are normally in the "ON" position until the last information symbol goes into the encoder. At this moment all the switches are turned to the "OFF" position and the encoder behaves like a long shift register. The output of the encoder is then taken from the output of the last shift register. [Ref. 8]

C. BINARY DECODER

In this section we discuss the decoding process and design a single-error-correcting binary decoder and a double-error-correcting binary decoder both of which can be used in conjunction with the binary encoders of Section A of this chapter.

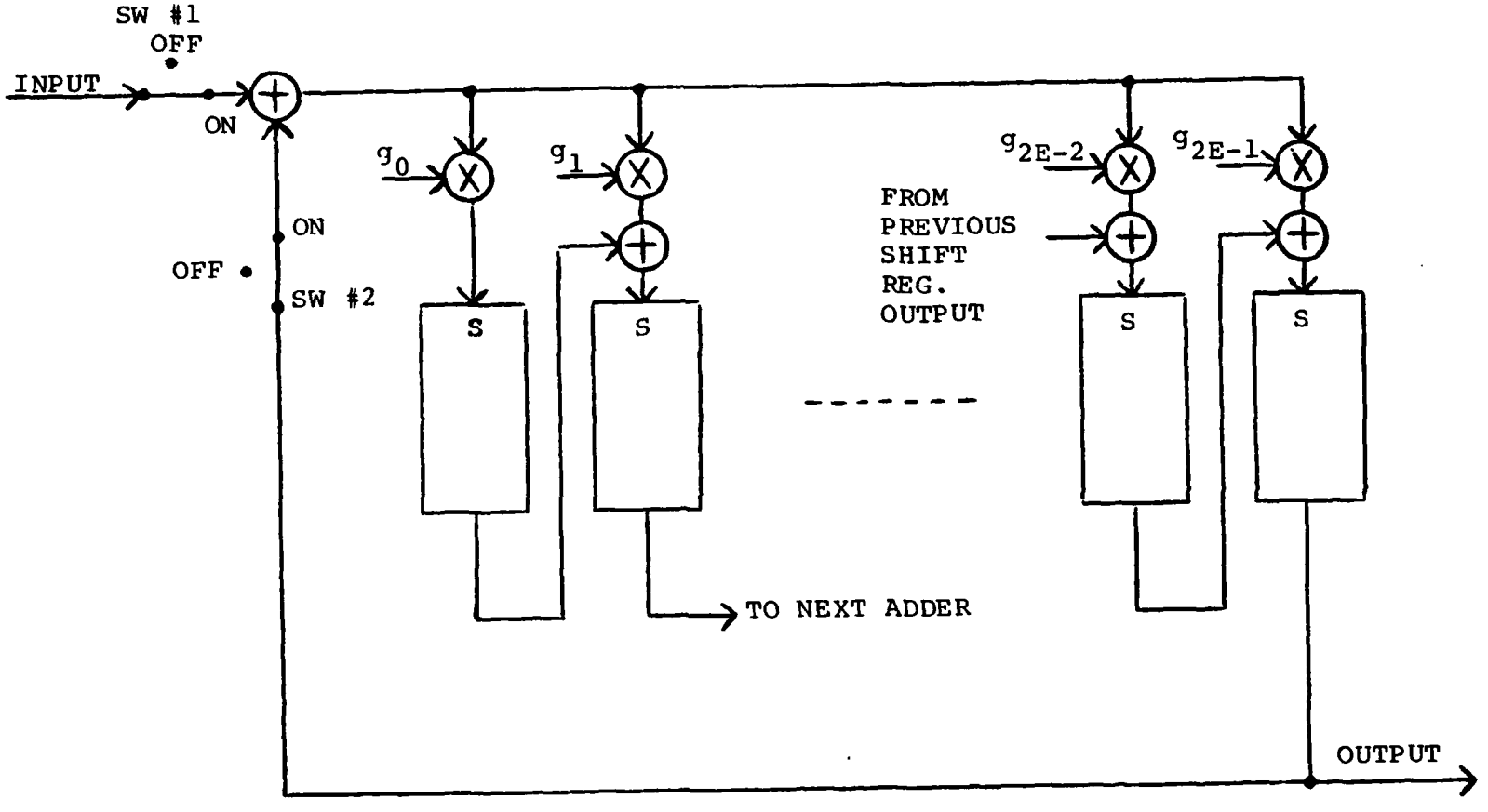


Figure 6.3 The RS Encoder

1. Decoding Process

The decoding process is, in general, much more complicated than the encoding process. Not only must we deal with the detection of errors but also with their correction. As a result, we must be able to design a decoder which simultaneously detects and corrects errors.

Error detection is usually much easier than error correction. Recall that a code polynomial is a multiple of the generating polynomial p(x). In other words, the received polynomial u(x) will be a code polynomial if and only if the remainder upon division of u(x) by p(x) is zero, i.e., $u(x) \equiv 0$ modulo p(x). An example is given in Table VI. The register contents after u(x) is fed completely into the detecting division register will contain u(x) modulo p(x). If any of the register contents are nonzero, u(x) is not a valid codeword. Thus the shift register acts as an error detector by performing a division of u(x) by p(x). In fact, the nonzero contents not only indicate that an error has occurred in transmission, but those contents also indicate the error pattern needed to correct the error and the location of the error in the transmitted codeword. [Ref. 10]

2. Single-Error-Correcting Binary Decoder

Because of the complexity of the decoding process, we will initially design an error detection register followed by its error correction counterpart and then

TABLE VI

VERIFICATION OF THE CODE POLYNOMIAL

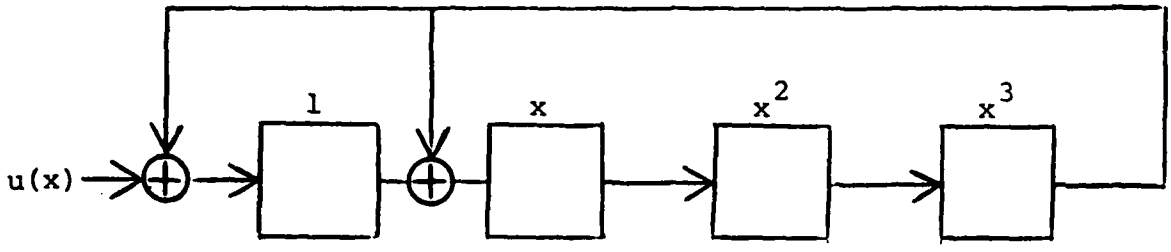
$$
\begin{array}{r|l}
 & x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{3}+1 \\
\hline
x^{4}+x+1 & x^{14}+x^{12}+x^{10}+x^{8}+x^{6}+x^{4}+x^{3}+x+1 \\
 & \underline{x^{14}+x^{11}+x^{8}} \\
 & \quad x^{12}+x^{11}+x^{8} \\
 & \quad \underline{x^{12}+x^{9}+x^{8}} \\
 & \quad\quad x^{11}+x^{9}+x^{6} \\
 & \quad\quad \underline{x^{11}+x^{8}+x^{7}} \\
 & \quad\quad\quad x^{9}+x^{8}+x^{7}+x^{6}+x^{4} \\
 & \quad\quad\quad \underline{x^{9}\qquad\quad +x^{6}+x^{5}} \\
 & \quad\quad\quad\quad x^{8}+x^{7}+x^{5}+x^{4}+x^{3} \\
 & \quad\quad\quad\quad \underline{x^{8}\quad\quad +x^{5}+x^{4}} \\
 & \quad\quad\quad\quad\quad x^{7}+x^{3}+x \\
 & \quad\quad\quad\quad\quad \underline{x^{7}+x^{4}+x^{3}} \\
 & \quad\quad\quad\quad\quad\quad x^{4}+x+1 \\
 & \quad\quad\quad\quad\quad\quad \underline{x^{4}+x+1} \\
 & \quad\quad\quad\quad\quad\quad\quad 0
\end{array}
$$

synthesize them together to implement the complete decoder. To begin, we utilize the error detection register of Figure 6.4. It is identical to the encoding register of Figure 6.1 except that the received codeword is input to the decoder at the left end of the register. If the received word is 110111101010101, then the nonzero contents 0110 after division indicate that an error has occurred in transmission. In order to correct the received word we need to know the error position.

The received word can be viewed as a polynomial $u(x)$ which can be written as the sum of the code polynomial $c(x)$ and an error polynomial $e(x)$, namely $u(x) = c(x) + e(x)$. The error polynomial $e(x)$ has ones in its error positions and zeros elsewhere, and addition is term by term modulo 2.

Since the codewords $c(x)$ are generated as multiples of the generator polynomial $g(x)$ and since β is a root of $g(x)$, the code polynomials evaluated at β are equal to zero, namely $c(\beta) = 0$. Thus $u(\beta) = c(\beta) + e(\beta) = e(\beta)$. Since we assume in this sub-section that only single errors have occurred in transmission we can also assume that if an error occurs then $e(x)$ is a power of x, say $e(x) = x^i$ for some i. Thus $u(\beta) = e(\beta) = \beta^i$.

In order to correct the error we need to compute $u(\beta)$ which is called the syndrome of the received word and then find the specific value i for which $u(\beta) = \beta^i$. The value i will indicate the error position. We need then only



TIME (t)	REGISTER CONTENTS
t = 0	0 0 0 0
t = 1	1 0 0 0
t = 2	0 1 0 0
t = 3	1 0 1 0
t = 4	0 1 0 1
t = 5	0 1 1 0
t = 6	0 0 1 1
t = 7	0 1 0 1
t = 8	1 1 1 0
t = 9	1 1 1 1
t = 10	0 0 1 1
t = 11	0 1 0 1
t = 12	0 1 1 0
t = 13	0 0 1 1
t = 14	0 1 0 1
t = 15	0 1 1 0 = ERROR

Figure 6.4 The Error Detection Register

set $c(x) = u(x) + x^i$ to obtain the code polynomial $c(x)$ a multiple of $p(x)$ which is "nearest" to the received polynomial $u(x)$. The primitive shift register facilitates this task because while it is computing $u(x)$ modulo $p(x)$ it also leaves the coefficients of $u(\beta) = \beta^i$ in the shift register. [Ref. 10]

For example, in Figure 6.5 the primitive shift register computes $u(x)=1+x+x^3+x^4+x^5+x^6+x^8+x^{10}+x^{12}+x^{14}$ modulo $p(x)=1+x+x^4$ and the syndrome is 0110 = $x+x^2$. Note from Table I of Chapter III that 0110 is the 4-digit representation of β^5. Hence the error in the received polynomial occurs in the position of x^5. Therefore, the code polynomial is $c(x)= u(x)+e(x)=(1+x+x^3+x^4+x^5+x^6+x^8+x^{10}+x^{12}+x^{14})+x^5 = 1+x+x^3+x^4+x^6+x^8+x^{10}+x^{12}+x^{14}$. The corrected codeword is 110110101010101 and the corrected information symbols are 10101010101. The same procedure is also illustrated in Table VII by the actual long division process.

We now examine the primitive shift register decoding process which performs the error correction. After the syndrome is computed by the primitive shift register division process, an additional primitive shift register of the same type can be used to correct the error without reference to a table of powers of the primitive element β. The correcting register shown in Figure 6.6 is basically the same primitive shift register used throughout this chapter with the exception that there are output lines leading from

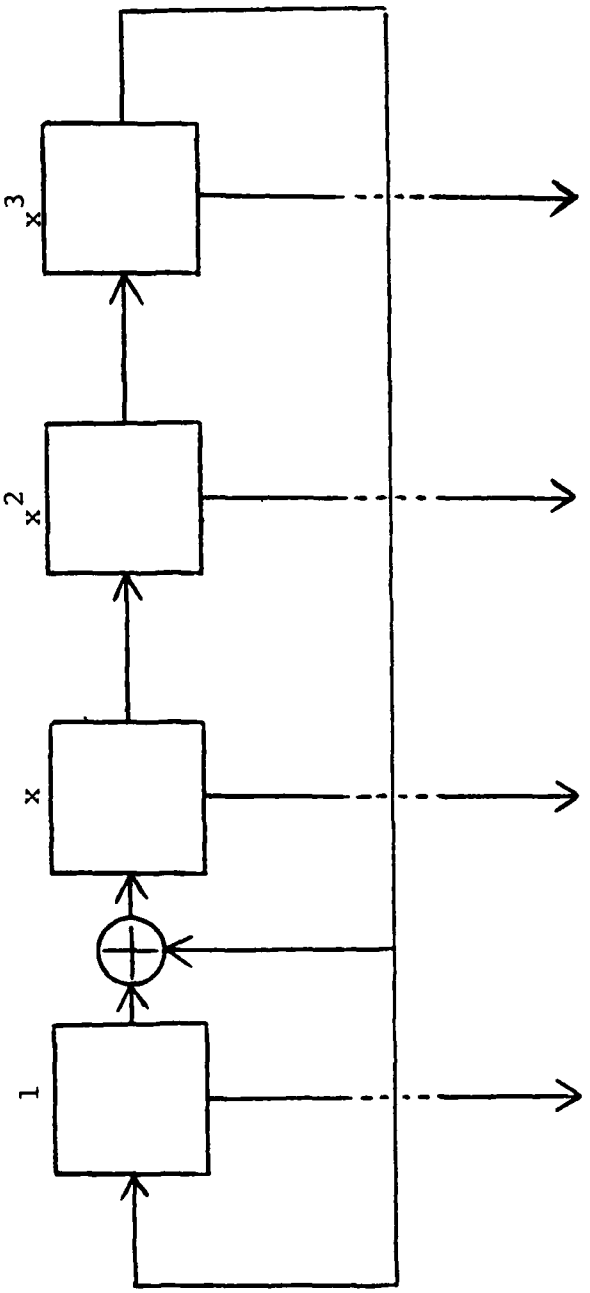


Figure 6.5 The Error Correction Register

[Ref. 12] will be presented in this section to obtain a repetitive and recursive technique which is suitable for systolic array development and eventual VLSI implementation.

Recall that the information symbols of an RS code are treated as the coefficients of the polynomial f(x). If we let

$$f(x) = f_0+f_1x+\ldots+f_{N-1}x^{N-1}$$

be the transmitted code vector (where N = codeword length), and let

$$r(x) = r_0+r_1x+\ldots+r_{N-1}x^{N-1}$$

be the received code vector over a noisy channel, then the error pattern added by the channel is

$$e(x) = r(x)-f(x) = e_0+e_1x+\ldots+e_{N-1}x^{N-1}.$$

The first step of the decoding procedure is to store the received code vector r_j into the buffer register and then compute the syndrome S_i using the equation

$$S_i = r(\beta^{k+1}) = \sum_{j=0}^{N-1} r_j \beta^{(k+i)j} \qquad (5)$$

where $0 \leq i \leq 2E-1$. Since $r_j=f_j+e_j$, equation (5) can be expressed as

compute the locations of the two errors. However, for example, suppose that the Central Galois Field Processor is able to compute the error locator in half the time required for n digits to be received from the channel. In that case, the buffer need only be capable of storing 3n/2 digits. After a complete word is received, the central processor computes its error location by the time the beginning of this word is ready to leave the buffer. The error locator is then fed into the Chien Searcher, and the central processor sits idle until the rest of the incoming word is received. See [Ref. 5] for details.

Although the above discussion pertains strictly to a binary decoder capable of correcting two errors, it can be generalized to correct t or fewer errors. By expanding the hardware in Stages I and III to accommodate the additional shift register size required by t distinct minimal polynomials, we are able to implement the decoder with approximately the same effectiveness. Likewise, the same procedure of utilizing the product of t distinct minimal polynomials would also be used in the design of a multiple-error-correction binary encoder.

D. REED-SOLOMON DECODER

As with any multiple-error detection and correction process, the decoding of RS codes is very complex. As a result, the known decoding procedures as discussed by Liu

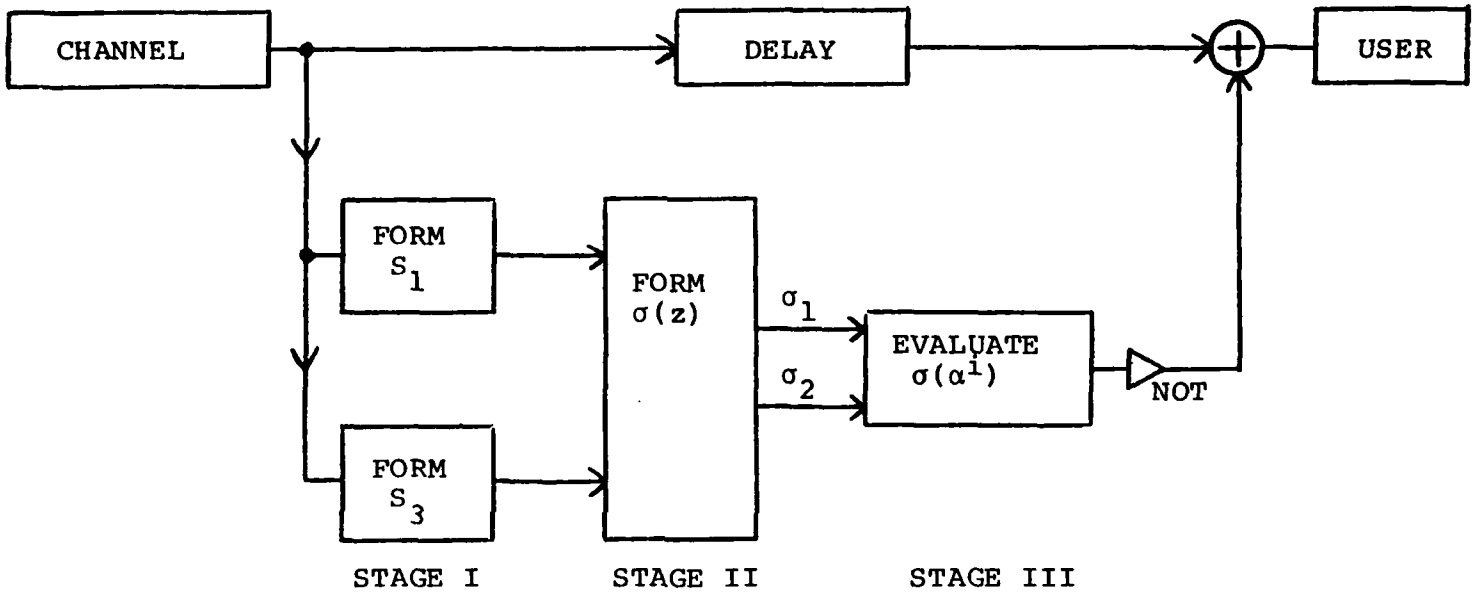


Figure 6.11 The Complete Double-Error-Correcting Binary Decoder

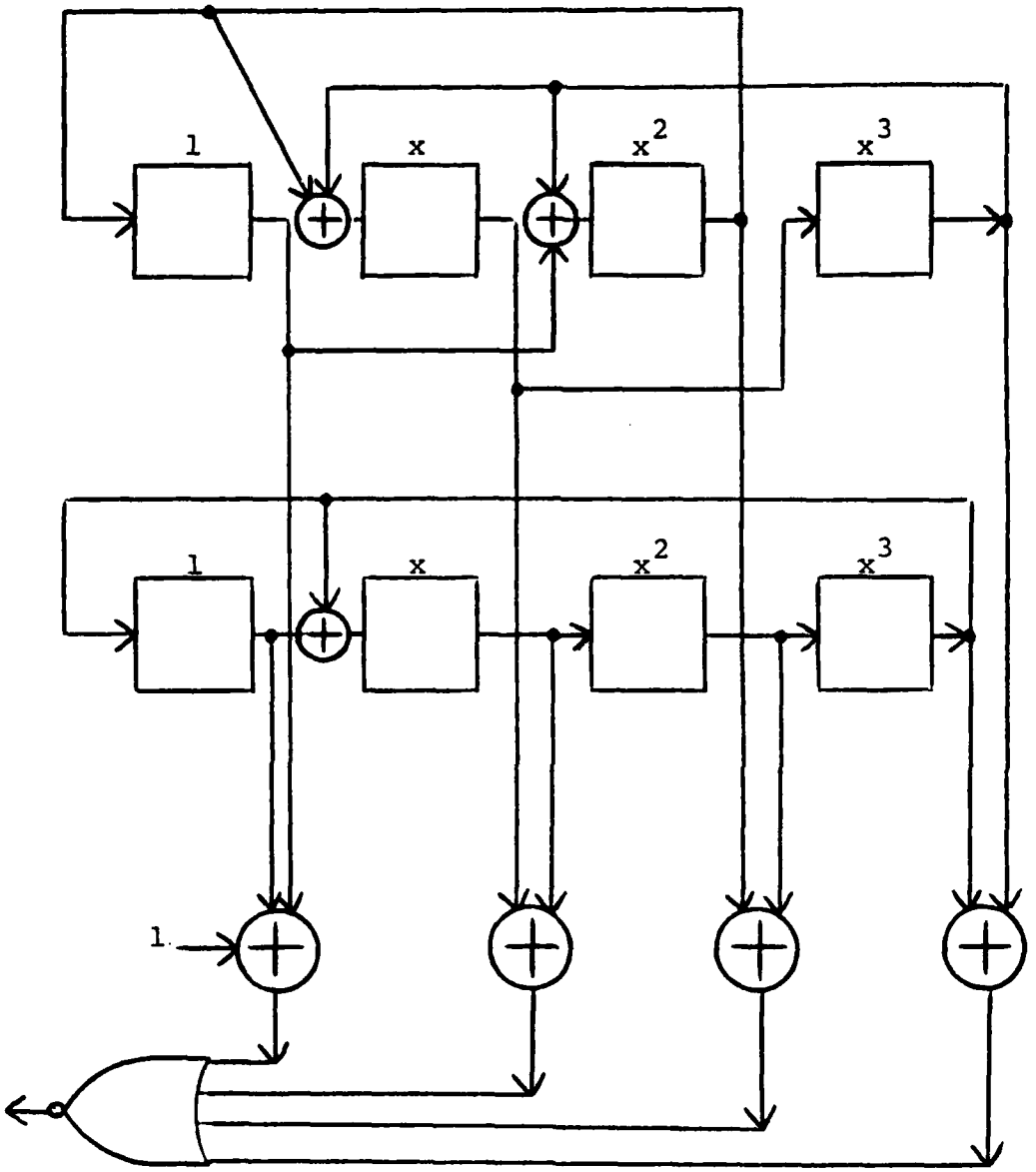


Figure 6.10 Stage III: The Chien Searcher

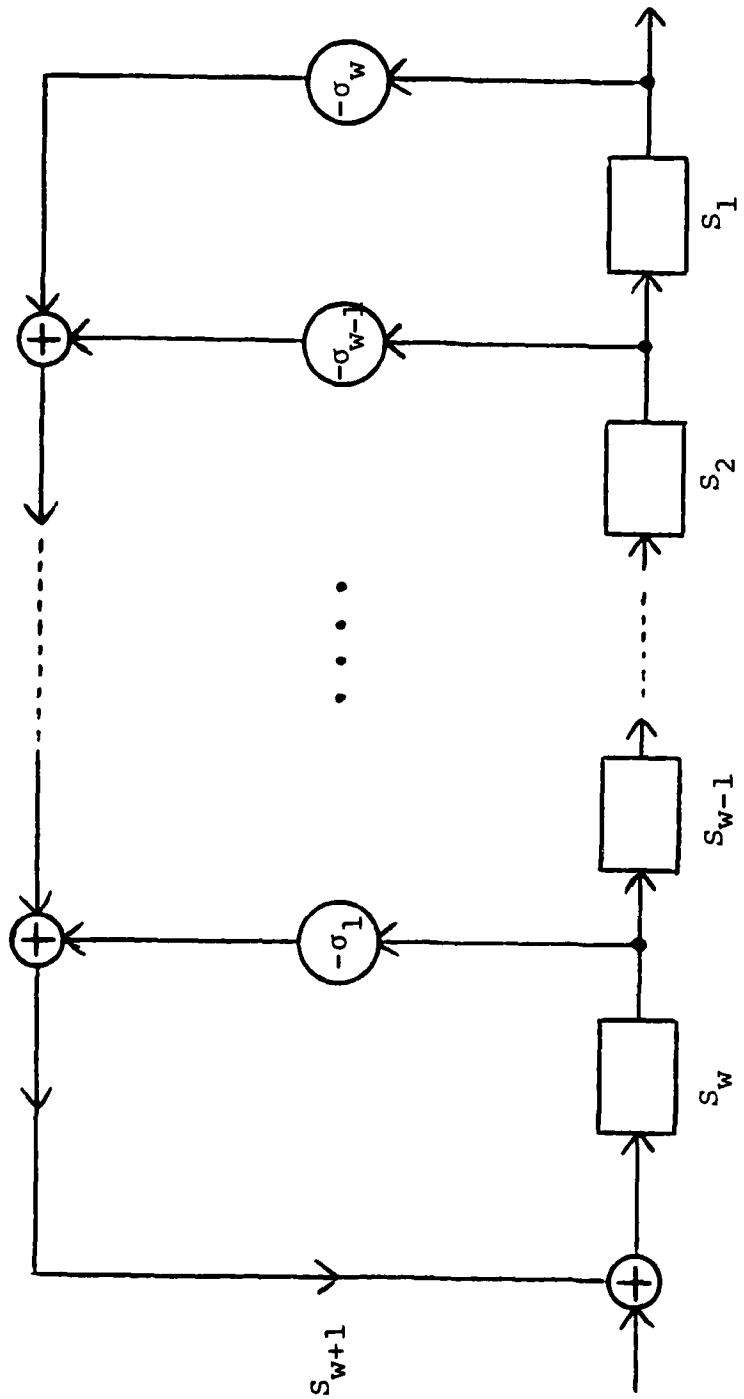


Figure 6.9 Stage II: The Central Galois Field Processor

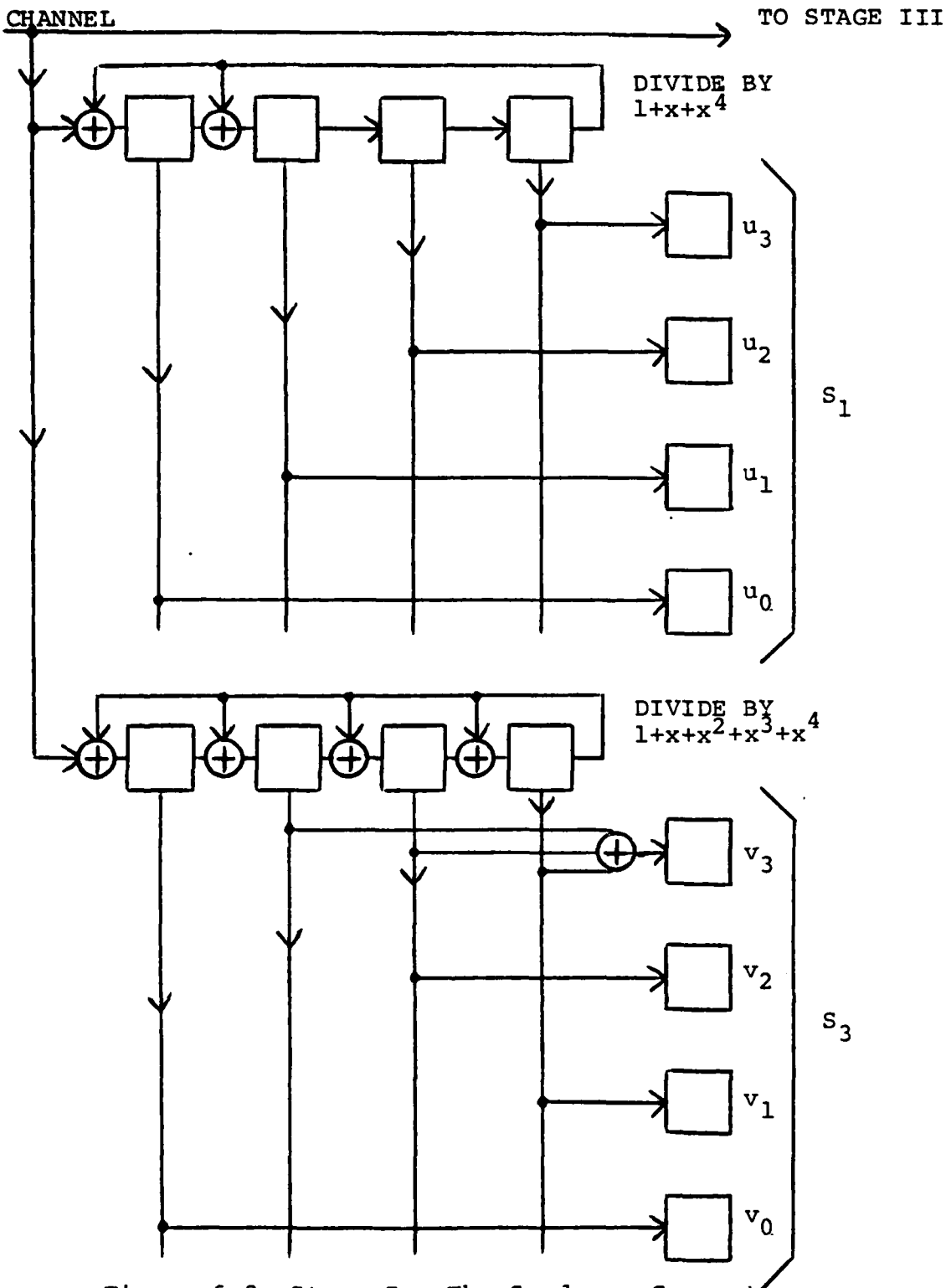


Figure 6.8 Stage I: The Syndrome Generator

$(1+x+x^4)(1+x+x^2+x^3+x^4)$, we are able to produce Stage I of the decoding process as illustrated by the division process in Figure 6.8. Similarly, we are also able to produce Stages II and III (Figures 6.9 and 6.10, respectively) along with a block diagram of the complete decoder in Figure 6.11.

The operation of the decoder is relatively straight forward as in the previous section. Utilizing a buffer capable of storing 2n digits, the Chien Searcher is in the process of computing $\sigma(z)$ in order to determine whether or not the next digit to leave the buffer should be corrected. The Syndrome Generator at the same time computes the syndrome of the received word while the Central Galois Field Processor finds the error-locator polynomial for the buffered word. Once the coefficients of the error-locator polynomial are read out of the Central Galois Field Processor and into the Chien Searcher, the syndrome or the nonzero remainder of the next block of received words is read back into the Central Galois Field Processor for continual operation. See [Ref. 5] for further details of the multiple error correction process.

If the Central Galois Field Processor operates so fast that it is able to compute the error location before all of the new received word arrives, then the buffer size may be reduced. In general, the buffer is made big enough to accommodate the expected worst case for the time to

To recapitulate the correction process, the detecting register computes the syndrome of the received word. As each digit of u(x) enters the detecting register it simultaneously enters the storage register. When the syndrome is determined, it is transferred to the correcting register for the error-correcting procedure just described.

3. Double-Error-Correcting Binary Decoder

To implement a double-error-correcting binary decoder we begin with a general analysis of the three stages that comprise decoding. The first stage is the Syndrome Generator stage. The syndrome is defined as the nonzero remainder of the received polynomial when it is divided by the given primitive shift register. The second stage or the Central Galois Field Processor finds the error locator polynomial $\sigma(z)$ (usually accomplished by using Berlekamp's iterative algorithm or Massey's linear feedback shift register synthesis algorithm). At this stage the polynomial is determined which defines the location of the errors that have occurred in transmission. Finally, the third stage or the Chien Searcher stage finds the roots of $\sigma(z)$ to determine which digits should be corrected. Note, in the binary code, correction is trivial when the location of the errors is determined, i.e., the bit in error need only be complemented. [Ref. 11]

Using our previous double-error-correcting generator polynomial $1+x^4+x^6+x^7+x^8$, which is the product of

Note from Table VIII that the incorrect digit u_5 leaves the storage register when the correcting register is in state 1000. If the detector is made to produce an output 1 when it detects 1000 and 0 otherwise, then u_5 will be properly corrected. In general, if the syndrome is β^i, then the error occurs in the coefficient of x^i, namely u_i, where the received polynomial has the form

$$u(x) = \sum_{i=0}^{n-1} u_i x^i$$

If j is such that $u_{n-j} = u_i$, then u_i leaves the storage register when the correcting register is in state β^{i+j} as shown below:

State	Output
β^{i+1}	u_{n-1}
β^{i+2}	u_{n-2}
.	.
.	.
.	.
β^{i+j}	$u_{n-j}=u_i$

Since $\beta^{i+j}=\beta^n=1$, the detector will correct the digit u_i and the received word will be corrected to the nearest code word after the decoder completes this process. [Ref. 10]

TABLE VIII

CORRECTION AND DECODING PROCESS

SHIFT	4-TUPLE	BETA POLYNOMIAL	DETECTOR OUTPUT	STORAGE OUTPUT	DECODER
0	0 1 1 0	$\beta^5 = \beta + \beta^2$	--	--	--
1	0 0 1 1	$\beta^6 = \beta^2 + \beta^3$	0	$u_{14} = 1$	1
2	1 1 0 1	$\beta^7 = 1 + \beta + \beta^3$	0	$u_{13} = 0$	0
3	1 0 1 0	$\beta^8 = 1 + \beta^2$	0	$u_{12} = 1$	1
4	0 1 0 1	$\beta^9 = \beta + \beta^3$	0	$u_{11} = 0$	0
5	1 1 1 0	$\beta^{10} = 1 + \beta + \beta^2$	0	$u_{10} = 1$	1
6	0 1 1 1	$\beta^{11} = \beta + \beta^2 + \beta^3$	0	$u_9 = 0$	0
7	1 1 1 1	$\beta^{12} = 1 + \beta + \beta^2 + \beta^3$	0	$u_8 = 1$	1
8	1 0 1 1	$\beta^{13} = 1 + \beta^2 + \beta^3$	0	$u_7 = 0$	0
9	1 0 0 1	$\beta^{14} = 1 + \beta^3$	0	$u_6 = 1$	1
10	1 0 0 0	$\beta^{15} = 1$	1	$u_5 = \underline{1}$	$\underline{0}$

NOTE: Register u_5 is corrected from its initial error state 1 to its corrected state 0.

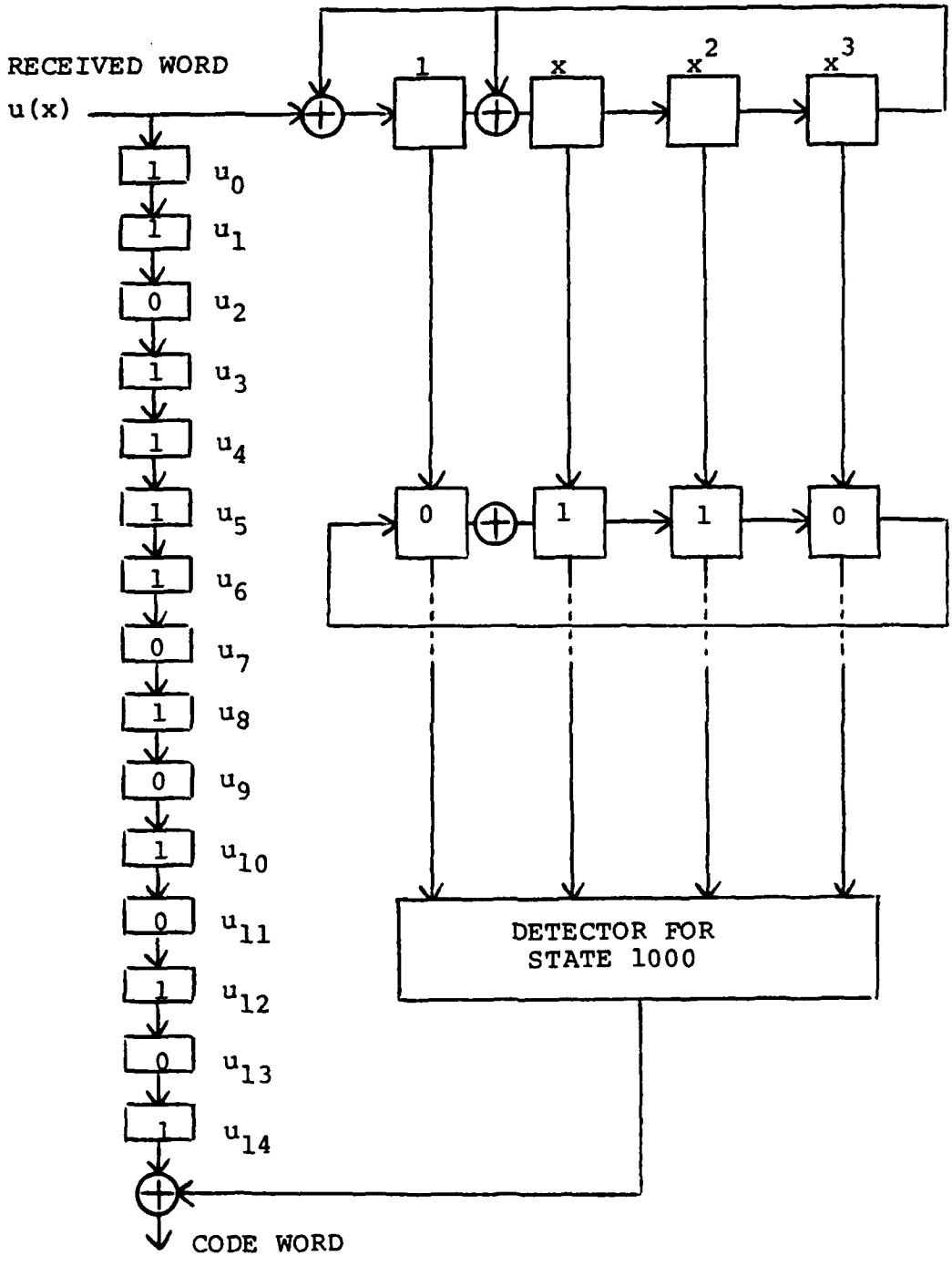


Figure 6.7 The Complete Single-Error-Correcting Binary Decoder

each of the four registers. If the correcting register is set initially at 0100, the 4-digit representation for the element β, then, as it shifts, the output is the same cycle as the 4-digit representation listing of the $\beta^i (i=1,2,3,\ldots,15)$ in Table I since a shift in the primitive shift register is the same as multiplication by β. No matter which state the register is set to initially the correcting register will output elements of that maximum-cycle in the same cycle order as long as the register continues to shift. If the register is set at β^i, it will be in state β^{i+j} after j shifts. [Ref. 10]

Figure 6.7 (the complete single-error-correcting binary decoder) shows the received word of our example, namely 110111101010101 whose polynomial form is $1+x+x^3+x^4+x^5+x^6+x^8+x^{10}+x^{12}+x^{14}$ in a storage register and the syndrome 0110 in the correcting register. From our previous discussion we know that the error occurs in u_5. Thus, if the detector register has output 1 as u_5 leaves the storage register and 0 otherwise, the word 110111101010101 will be corrected after fifteen shifts to read 110110101010101. We illustrate how the correcting register is used to accomplish this task by listing the new states of the correcting register, and the outputs from the storage register and decoder after each shift. The states are depicted in Table VIII.

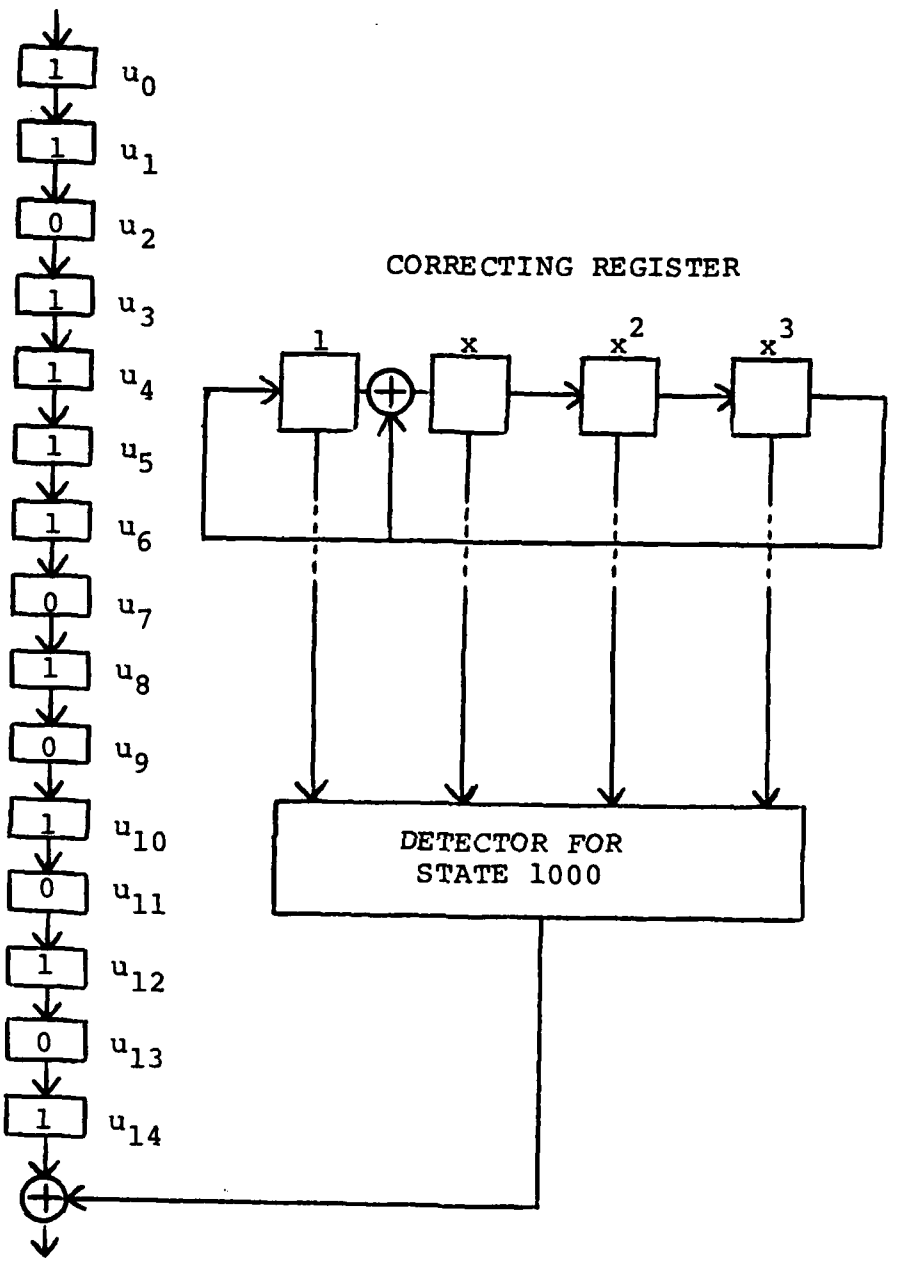


Figure 6.6 The Initial Single-Error-Correcting Binary Decoder

TABLE VII

SYNDROME CALCULATION USING LONG DIVISION

$$
\begin{array}{r|l}
 & x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{3}+x+1 \\
\hline
x^4+x+1 & x^{14}+x^{12}+x^{10}+x^{8}+x^{6}+x^{5}+x^{4}+x^{3}+x+1 \\
 & \underline{x^{14}+x^{11}+x^{10}} \\
 & \quad x^{12}+x^{11}+x^{8} \\
 & \quad \underline{x^{12}+x^{9}+x^{8}} \\
 & \quad\quad x^{11}+x^{9}+x^{6} \\
 & \quad\quad \underline{x^{11}+x^{8}+x^{7}} \\
 & \quad\quad\quad x^{9}+x^{8}+x^{7}+x^{6}+x^{5} \\
 & \quad\quad\quad \underline{x^{9}+x^{6}+x^{5}} \\
 & \quad\quad\quad\quad x^{8}+x^{7}+x^{4} \\
 & \quad\quad\quad\quad \underline{x^{8}+x^{5}+x^{4}} \\
 & \quad\quad\quad\quad\quad x^{7}+x^{5}+x^{3} \\
 & \quad\quad\quad\quad\quad \underline{x^{7}+x^{4}+x^{3}} \\
 & \quad\quad\quad\quad\quad\quad x^{5}+x^{4}+x \\
 & \quad\quad\quad\quad\quad\quad \underline{x^{5}+x^{2}+x} \\
 & \quad\quad\quad\quad\quad\quad\quad x^{4}+x^{2}+1 \\
 & \quad\quad\quad\quad\quad\quad\quad \underline{x^{4}+x+1} \\
 & \quad\quad\quad\quad\quad\quad\quad\quad x^{2}+x
\end{array}
$$

SYNDROME: $x+x^2 = 0110 = \beta^5$

1	β	β^2	β^3	β^4	β^5	β^6	β^7	β^8	β^9	β^{10}	β^{11}	β^{12}	β^{13}	β^{14}	
1	1	0	1	1	1	1	0	1	0	1	0	1	0	1	= ERROR CODEWORD
1	1	0	1	1	0	1	0	1	0	1	0	1	0	1	= CORRECTED CODEWORD

$$S_i = \sum_{j=0}^{N-1} (f_j + e_j)\beta^{(k+i)j}$$

$$= \sum_{j=0}^{N-1} f_j \beta^{(k+i)j} + \sum_{j=0}^{N-1} e_j \beta^{(k+i)j}$$

$$= F_{k+i} + E_{k+i} \qquad (6)$$

In the above equation

$$F_{k+i} = \sum_{j=0}^{N-1} f_j \beta^{(k+i)j} \qquad (7)$$

and

$$E_{k+i} = \sum_{j=0}^{N-1} e_j \beta^{(k+i)j} \qquad (8)$$

Note that in equation (8) E_{k+i} is the finite field transform of the e_j's.

The second step of the decoding procedure is to compute σ_ℓ for $1 \leq \ell \leq v$ (where v = number of errors) using the equation

$$S_i + \sum_{\ell=1}^{v} S_{i-\ell}\sigma_\ell = 0 \qquad \text{for } 0 \leq i \leq 2E-1$$

from the syndromes computed in the previous step. This can be accomplished using Berlekamp's iterative algorithm or Massey's linear feedback shift register (LFSR) synthesis algorithm. [Ref. 12]

Upon obtaining the σ_ℓ's, the third step of the decoding procedure is to use the recursive equation

$$E_{k+i} = \sum_{\ell=1}^{v} E_{k+i-\ell}\sigma_\ell = 0 \qquad \text{for } 2E \leq i \leq N-1$$

where

$$E_{k+i} = E_{k+i-N} \qquad \text{for } k+i \geq N$$

to compute the remaining E_{k+i} for $2E \leq i \leq N-1$.

After determining the transform of the error pattern E_{k+i} for $0 \leq k+i \leq N-1$, by equation (8), we can then apply the inverse transform to E_{k+i}, to obtain the error pattern e_j, i.e.,

$$e_j = (N)^{-1} \sum_{k+i=0}^{N-1} E_{k+i}\beta^{-(k+i)j}$$

for $j=0,1,2,\ldots,N-1$. Then the corrected codeword is obtained by subtracting the error pattern e_j from the stored code vector r_j in the buffer register.

In summary, the decoding of an RS code is composed of the following five steps:

1) Compute the syndrome S_i using the equation

$$S_i = r(\beta^{k+i}) = \sum_{j=0}^{N-1} r_j \beta^{(k+i)j}$$

2) Use Berlekamp's iterative algorithm or Massey's LFSR synthesis algorithm to determine the coefficients of the error locator polynomial σ_ℓ from the known $S_i = E_{k+i}$ for $i=0,1,2,\ldots,2E-1$.

3) Compute the remaining $E_{k+\ell}$ from the known S_i using the equation

$$E_{k+i} + \sum_{\ell=1}^{v} E_{k+i-\ell}\sigma_\ell = 0$$

for $2E < i < N-1$.

4) Compute the inverse transform

$$e_j = (N)^{-1} \sum_{k+i=0}^{N-1} E_{k+i}\beta^{-(k+i)j}$$

to obtain the error pattern, where $(N)^{-1}$ is the inverse of N.

5) Subtract the error pattern e_j from the received code vector r_j in the buffer memory to obtain the corrected codeword.

Note that in steps 1, 4, and 5, the processing time is proportional to N*J, while in steps 2 and 3 the

processing time is proportional to 2E*J and (N-2E)*J, respectively. Hence, a natural partition for pipeline processing is to divide the decoder system into three pipeline stages. Stage 1 is used to perform step 1, stage 2 is used to perform steps 2-4, and stage 3 is used to perform step 5. To obtain a uniform throughput of one decoded symbol per symbol clock cycle, each pipeline stage is required to complete its computations in N symbol clock cycles. As always, the throughput of the system is determined by the slowest stage in the pipeline. [Ref. 12]

The RS decoder architecture using the above pipeline decoding technique is shown in Figure 6.12. The timing chart of the decoder is shown in Figure 6.13. In both figures, note that the first 2E input symbols of the inverse transform, which are S_0, S_1, ..., S_{2E-1}, can be processed in parallel with the Berlekamp/Massey LFSR synthesis algorithm. The remaining N-2E input symbols of the inverse transform are obtained from the remaining transform. Each of these N-2E input symbols is processed by the inverse transform circuit immediately after its generation. In stage 3, the buffer memory is read out symbol-by-symbol and "Exclusive-OR'ed" with the output of the inverse transform. A triple-buffered memory is required to store the three active codewords in the pipeline. [Ref. 12]

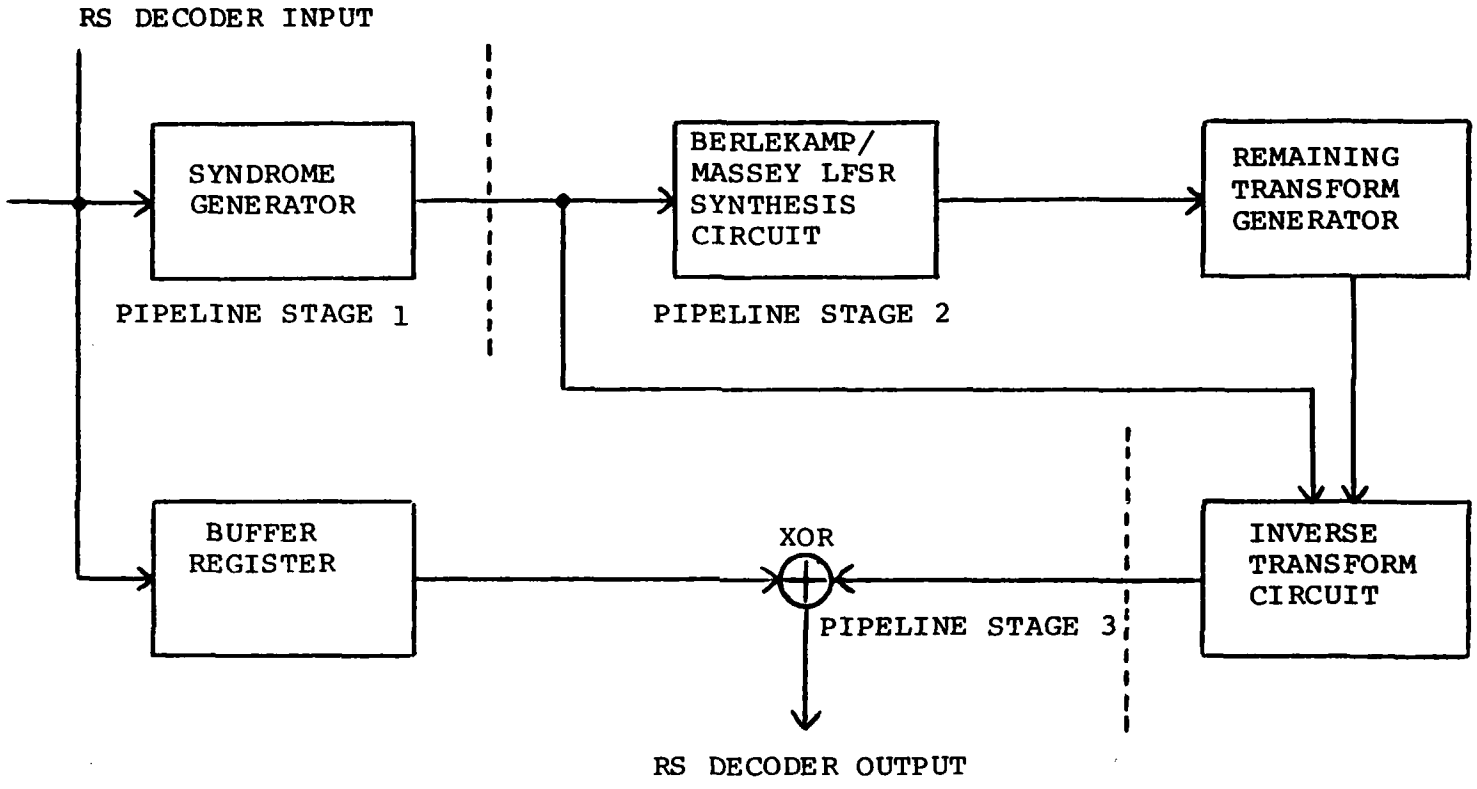


Figure 6.12 The RS Decoder Architecture

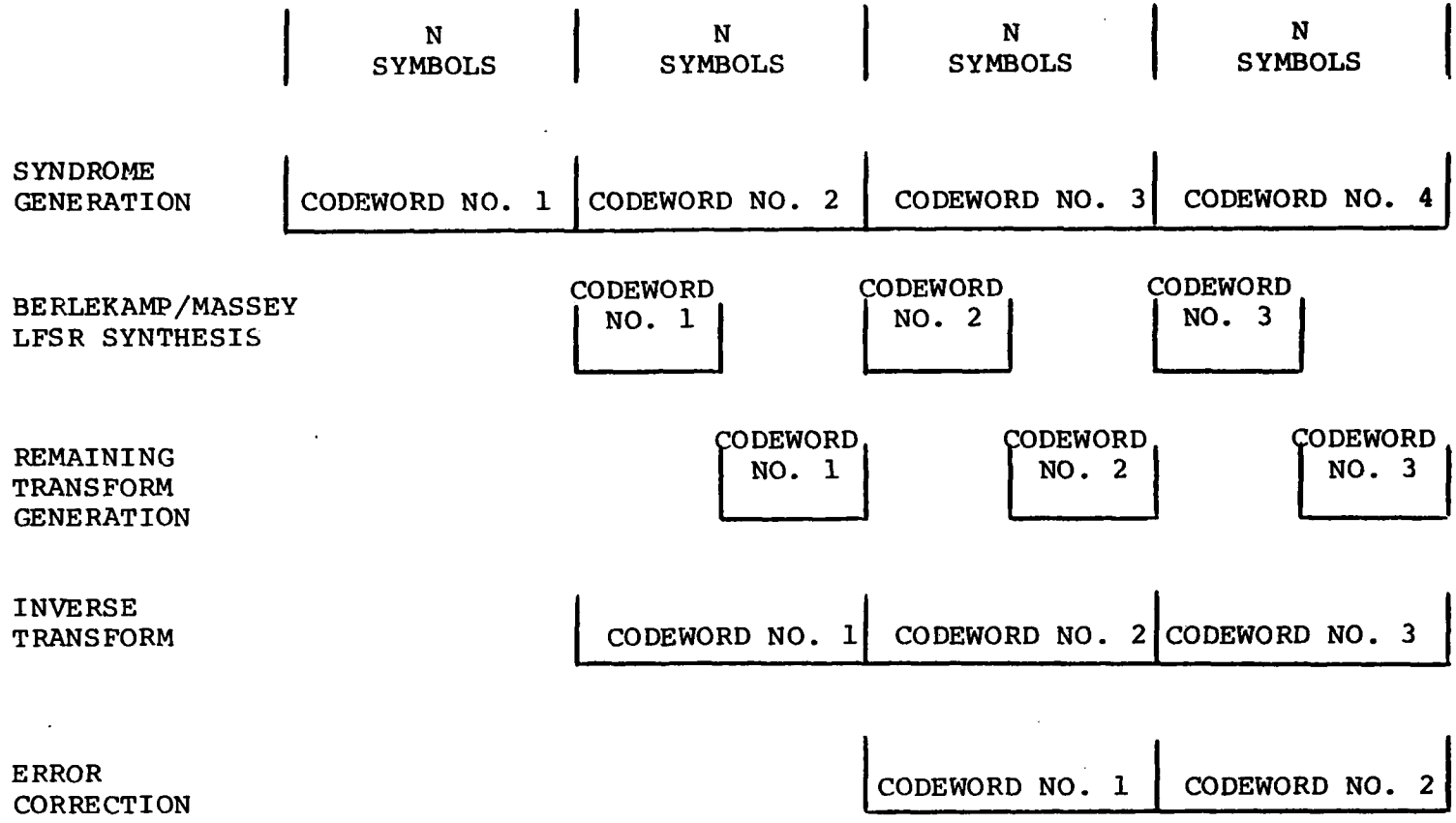


Figure 6.13 The RS Decoding Timing Chart

VII. CONCLUSION

In this thesis we have taken a modular approach to the systolic implementation of a Reed-Solomon encoder and decoder. By initially discussing the theory behind systolic arrays and finite fields, we have shown how they play an integral part in the overall implementation. The binary case is presented first because of its simple architecture and ease of understanding. It is then followed by a design of a systolic multiplier and an RS encoder and decoder.

The multiplier requires m basic cells for the finite field $GF(2^m)$. Because of its simple-control methodology, regular interconnection pattern, and modular structure it is highly suited for VLSI implementation. The encoder using the systolic multiplier offers the advantage of requiring less power, minimal size, and high reliability. The decoder being modular in design is also highly suited for a systolic architecture, thus the decoding speed can easily be increased by using a distributive processing scheme. In this way, several decoders can operate in parallel simultaneously, while each individual decoder can operate in a pipeline fashion.

The design of both the RS encoder and decoder is simple and regular. They can be constructed using a systolic array

of identical cells with every interconnection path occurring between adjacent cells. This makes implementation in VLSI extremely attractive since the layout of the cell need only be done once and then replicated.

It is hoped that with this thesis as a guide, an interested electrical engineering student could implement the encoder or decoder in hardware. By building the four cell-binary encoder first, the student would establish a firm foundation vital to the development of the more complicated RS encoder. This process could then be expanded to produce an encoder of eight or sixteen cells, or the more general case of 2^m.

LIST OF REFERENCES

1. Berlekamp, E. R., "Technology of Error-Correcting Codes," Proc. IEEE, Vol. 68, May 1980, pp. 567-593.

2. Kung, H. T. and Leiserson, C. E., "Systolic Arrays (for VLSI)," Sparse Matrix Proc. 1978, Society for Industrial and Applied Mathematics, 1979, pp. 256-282.

3. Hwang, K. and Briggs, F. A., Computer Architecture and Parallel Processing, McGraw-Hill, New York, 1984, pp. 768-770.

4. Kung, H. T., "Why Systolic Architectures?" Computer, Vol. 15, pp. 37-46, January 1982.

5. Berlekamp, E. R., Algebraic Coding Theory, McGraw-Hill, New York, 1968, pp. 87-88.

6. Peterson, W. W., Error-Correcting Codes, Cambridge, MA: MIT Press, 1961, pp. 97-100.

7. Yeh, C. S., Reed, I. S., and Truong, T. K., "A Systolic Multiplier for Finite Fields of GF(2^m)," IEEE Trans. Comput., Vol. C-33, pp. 357-360, April 1984.

8. Liu, K. Y., "Architecture for VLSI Design of Reed-Solomon Encoders," IEEE Trans. Comput., Vol. C-31, pp. 170-175, February 1982.

9. Johl, J. T., "VLSI Design for Reed-Solomon Encoder," IEEE Proc. of the 1984 Custom Integrated Circuits Conference, pp. 615-618, May 1984.

10. Fellin, J. A., "Primitive Shift Registers," Applications of Abstract Algebra and Finite Field Theory to Computer Design and Data Communications System, pp. 1-34, November 1981.

11. MacWilliams, F. J. and Sloane, N. J. A., The Theory of Error-Correcting Codes, North-Holland, New York, 1977, pp. 294-295.

12. Liu, K. Y., "Architecture for VLSI Design of Reed-Solomon Decoders," IEEE Trans. Comput., Vol. C-33, pp. 178-189, February 1984.

BIBLIOGRAPHY

Berlekamp, E. R., "Bit-Serial Reed-Solomon Encoders," IEEE Trans. Inform. Theory, Vol. IT-28, pp. 869-874, November 1982.

Blahut, R. E., "Fast Decoding Algorithms for Reed-Solomon Codes," Secure Digital Communications, G. Longo (ed.), Springer-Verlag Wien-New York, November 1983, pp. 281-316.

Brent, R. P. and Kung, H. T., "Systolic VLSI Arrays for Polynomial GCD Computation," CMU Technical Report, March 1982.

Bromley, K., Symanski, J. M., and Whitehouse, H. J., "Systolic Array Processor Developments," VLSI Systems and Computations, H. T. Kung, R. F. Sproull, and G. L. Steele, Jr., (eds.), Carnegie-Mellon University, Computer Science Press, October 1981, pp. 273-284.

National Aeronautics and Space Administration Report 32-1275, Error Correction for Deep Space Network Teletype Circuits, by H. M. Fredricksen, 1 June 1968.

Laws, B. A. and Rushforth, C. K., "A Cellular-Array Multiplier for GF(2^m), IEEE Trans. Comput., Vol. C-20, pp. 1573-1578, December 1981.

Mandelbaum, D., "On Decoding of Reed-Solomon Codes," IEEE Trans. Inform. Theory, Vol. IT-17, pp. 707-712, November 1971.

Michelson, A., "A Fast Transform in Some Galois Fields and an Application to Decoding Reed-Solomon Codes," Proc. IEEE Int. Symp. Inform. Theory, p. 49, October 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Curricular Officer Computer Technology Programs Code 37 Naval Postgraduate School Monterey, California 93943-5100	1
4.	Prof. Harold M. Fredricksen Code 53Fs Department of Mathematics Naval Postgraduate School Monterey, California 93943-5100	3
5.	LTCOL Alan A. Ross Code 52Rs Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
6.	LT Stephen S. McKenzie 2301-5th Avenue, #4LL New York, New York 10037	2

END

FILMED

11-85

DTIC