A Low Complexity Design of Reed Solomon Code Algorithm for Advanced RAID System

Article *in* IEEE Transactions on Consumer Electronics · June 2007

DOI: 10.1109/TCE.2007.381684 · Source: IEEE Xplore

Sy-Yen Kuo
National Taiwan University
555 PUBLICATIONS **9,117** CITATIONS

A Low Complexity Design of Reed Solomon Code Algorithm for Advanced RAID System

Min-An Song, Sy-Yen Kuo, *Fellow*, IEEE and I-Feng Lan

**Abstract — ***This paper presents a new low complexity design of Reed Solomon model, which is the key technology for RAID system advanced codec. The real-time constraint of codec leads to a heavy computational bottleneck on today's data storage devices. To overcome this problem, design analysis and optimization of Reed Solomon code are addressed at the algorithmic level. First, the dominant calculation of spreading function is replaced with small look-up tables. Second, a method which greatly reduces the number of multipliers in performing the multiplication operations of the codec process is proposed to reduce the RAID (Redundant Array of Inexpensive Disks) system codec complexity while maintaining the quality. Moreover, our algorithm can be expanded to correct multiple failed disks. Therefore, the failed data in disks can be recovered and system still can work as usual without broken. This technique is different from the traditional methods. The proposed design could be implemented in a real-time Reed Solomon codec RAID system with reduction of hardware complexity by 28%. A hybrid strategy considering both data layout and load of storage nodes has a 13-25% higher I/O performance than conventional RS RAID.*

Index Terms —Reed Solomon codes, Reduced Static-Checksum, Reliability, RAID.

I. INTRODUCTION

Due to the rapid development of internet and multimedia data bases, the need for a big storage system becomes more and more important. The advanced storage system must provide rapid data processing ability and stable storage environment in order to avoid data loss. Presently, take hard disk with IDE interface and 7200 RPM as an example, size of hard-disk in mainstream market is usually 120 GB ~ 200 GB per drive. The size is big enough for personal computer. However, the size is too small for a server used as a database. No matter in capacity or reliability, it is deficient. In order to overcome the problems about limitations of single disk capacity and data processing speed and reliability of data storage, there are many distributed systems and fault-tolerant methods being proposed. RAID system is mostly appreciated.

Min-An Song is with the National Taiwan University, Department of Electrical Engineering. (phone: 886-2-23635251; fax: 886-2-23671909; e-mail: dennis@lion.ee.ntu.edu.tw).

Sy-Yen Kuo is with the National Taiwan University, Department of Electrical Engineering, also with Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan.

I-Feng Lan is with the Department of Electronic Engineering, Lee-Ming Institute of Technology (e-mail: iflan@venus.seed.net.tw).

Contributed Paper
Original manuscript received November 16, 2006
Revised manuscript received December 14, 2006

Because hard disks are produced by magnetic materials, the problems about wear out and life time limitation is coming up. Hence, data error or data failure will happen. Besides, using parallel processing algorithm to speed up data processing will also increase number of hard disks. Hence, error probability will become high. Therefore, how to use coding techniques to enhance reliability of storage system is an important issue. Presently, the greater part of RAID [1-4] techniques produced by industries mostly are simple parity check and binary array codes (like Even Odd code [5]), which can be implemented by very simple operations. But algorithms used at present usually can just solve at most two erasures. However, data amount of server and number of hard disk become larger, and system efficiency need also becomes greater, such that low erasure-tolerance will no longer be suitable. Under conditions of large amount of data and large number of hard disks, disk failures will happen easily. Hence, how to develop algorithms with high reliability and efficiency in order to implement in rapid and large RAID system becomes an important issue.

So far, only IBM brought up a coding algorithm of RAID 6, which uses an array code with two dimensions called Even Odd code. We propose here an algorithm based on Reed Solomon code to implement in advanced RAID system. The proposed algorithm can fit the needs for coding algorithms of RAID 6 at present. Besides, the advantages of the proposed algorithm based on Reed Solomon code are many. First, the number of data drives is flexible and the capability of fault-tolerance is flexible. Second, all computations can be completed by XOR. Furthermore, the proposed algorithm RS codec has been implemented by using about 28% less number of gates than the one based on conventional architectures.

The major purpose of this thesis is to apply Reed Solomon code to advanced RAID system in order to provide high system efficiency and fault-tolerant capability. Because the theory of Reed Solomon Code [6-10] has been developed over forty years, related coding theory has been mature. Hence, it would be good time for us to make practical applications to the Advanced RAID 6 System.

The rest of this paper is organized as follows. In Section II, the analyses of Reed Solomon algorithms and the dominant RAID 6 system are given. It shows the optimization of codec is necessary and critical. In Section III, our optimization methods are proposed to reduce the hardware complexity of RAID system. Experiments and simulation results are given in Section IV. Finally, in Section V, we summarize our major contributions.

II. Previous Works on RS Code and RAID 6

A. Encoding of RS code

The simplest coding method is that we can multiply message polynomial by generator polynomial directly to get the code polynomial. However, we can't get any relationship between the codeword and the original message directly after coding. Hence, for the purpose of getting information about the original messages in order to make reading speed faster, we will use systematic coding. Message polynomial is known as follows,

$$m(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}, \tag{1}$$

We multiply the message polynomial by x^{n-k} , where n is the length of codeword, and k is the length of message. Then divided by generator polynomial, we can get

$$x^{n-k} m(x) = a(x)g(x) + b(x), \tag{2}$$

Then

$$b(x) + x^{n-k} m(x) = a(x)g(x), \tag{3}$$

where

$$b(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-k-1} x^{n-k-1}, \tag{4}$$

Eq. (4) is the parity (checksum) symbols (polynomial) that we need. After expansion, we can get

$$a(x)g(x) = b(x) + x^{n-k} m(x)$$

$$= \underbrace{b_0 + b_1 x + \cdots + b_{n-k-1} x^{n-k-1}}_{\text{parity}} + \underbrace{m_0 x^{n-k} + m_1 x^{n-k+1} \cdots + m_{k-1} x^{n-1}}_{\text{message}} \tag{5}$$

From a(x), g(x), we can know that the original codeword is not changed. The changed part is the mapping method. Through such steps, we can get the codeword with information about the original message. Generally speaking, the coding method of systematic coding is shown in Figure 1 is just

$$b(x) = x^{n-k} m(x) \bmod g(x). \tag{6}$$

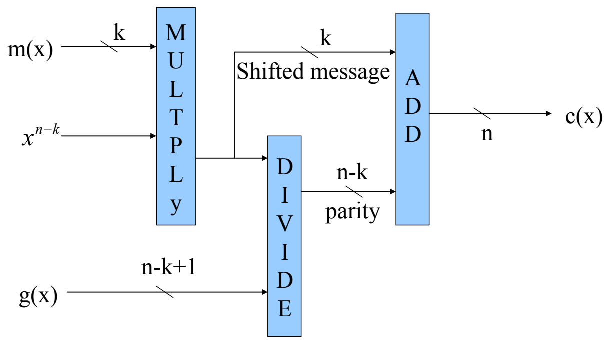


Fig. 1. Systematic coding

Because the fault model constructed on RAID belongs to erasures, the number of consecutive powers of α of generator polynomial represents the maximum number of disk failures that can be tolerable directly. For the example as shown above, if we construct the code with the generator polynomial with order four, there are four disk failures can be recovered.

B. Decoding of RS code

Decoding algorithm used in the industrial divides RS decoding process into four blocks, including Syndrome Calculation (SC), Key Equation Solver (KES), Chien Search (CS), and Forney Algorithm (FA). It is deserved to be mentioned that Key Equation is to solve the coefficients of ELP and Chien Search is to solve ELP equations.

However, the two parts are most complex in decoding process. Fortunately, these two parts can almost be omitted in erasures-only model. The decoding process is shown in Figure 2.

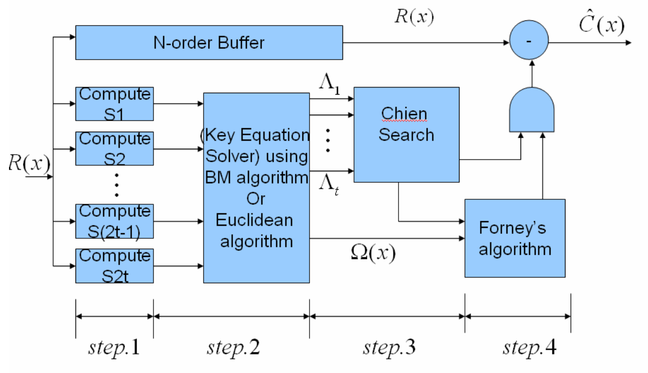


Fig. 2. Decoding flow of Reed Solomon Code

The advantage of dividing decoding process into four blocks is that the decoding process can be done as pipeline. Because KES and CS are not key points, we omit them and introduce FA immediately. Forney Algorithm was proposed by Forney. Forney Algorithm would be called a formula rather than an algorithm. Forney derived a new formula, which is called error-evaluator polynomial, by key equations in KES , CS and ELP. Formulas are listed without details. However, details about the derivative process can be found in [11-13].

Define error locator polynomial (ELP):

$$\Lambda(x) = \prod_{l=1}^{t} (1 - X_l x) = \Lambda_t x^t + \Lambda_{t-1} x^{t-1} + \ldots + \Lambda_1 x + \Lambda_0, \tag{7}$$

Construct syndrome Polynomial:

$$S(x) = 1 + \sum_{i=1}^{2t} S_i x^i, \tag{8}$$

Construct error locator polynomial (ELP) $\Lambda(x)$. Then, Error evaluator polynomial can be solved by Key Equation:

$$\Omega(x) = S(x)\Lambda(x) \mod x^{2t+1}, \tag{9}$$

where $\Omega(x)$ is called error-evaluator polynomial.

Forney formula:

$$Y_l = \frac{\Omega(X_l^{-1})}{\Lambda'(X_l^{-1})} \Big| X_l = \alpha^l, \tag{10}$$

where $\Lambda'(X_l^{-1})$ is the differential of $\Lambda(X_l^{-1})$, and ℓ is error position. Hence, error magnitude can be solved directly from this formula.

C. Construction of RAID6 system

Because only some companies (like IBM) own most techniques and intellectual property rights of RAID 6, many companies providing RAID system construction service want to find other usable algorithms to substitute for RAID 6. Hence, these companies need not to pay for the right and can save cost. The proposed new algorithm of RS code in this thesis, can also implement in RAID 6.

RAID 6 as shown in Figure 3 is essentially an extension of RAID 5 which allows for additional fault tolerance by using a second independent distributed parity scheme (dual parity). Data is striped on a block level across a set of drives, just like in RAID 5, and a second set of parity is calculated and written across all the drives; RAID 6 provides for an extremely high data fault tolerance and can sustain multiple simultaneous drive failures.

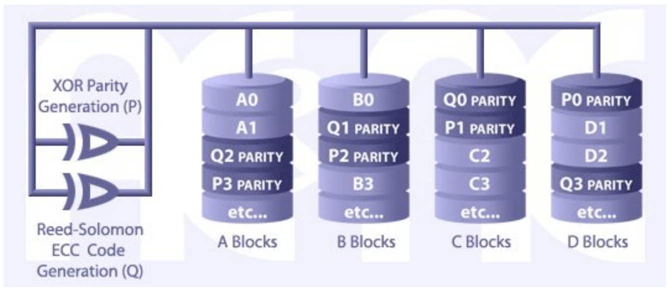


Fig. 3. RAID 6: Independent Data Disks with Two Independent Distributed Parity Schemes.

A mapping, for most RS encoders, from the message space in *k*-dimension to the code space in *n*-dimension is the most general approach of encoding. The encoder also usually generates systematic codes, namely, message bits of a symbol could be presented explicitly in its corresponding codeword.

$$c(x) = b(x) + m(x)x^{n-k}, \tag{11}$$

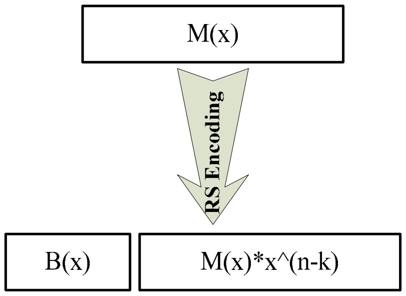


Fig. 4. Encode m(x) by RS Encoder

It shows a result after applying the systematic coding method, where *cw*(x), *b*(x), and *m*(x) are codeword, checksum, and message, respectively. Let a single disk with *K* bytes (8*K* bits) be separated into *L* (=8*K*/*W*) parts, where *W* is the codeword length in the RS code [11]. Figure 5 illustrates an example mapping. If *W*=4, there will be 4 checksum drives and 11 data drives in this system, i.e. b(x)={C1, C2, C3, C4} and m(x)={D1 , D2 , D3 , ……. , D10 , D11}.

Checksum				Data Drives			
$C_{1,1}$	$C_{2,1}$	$C_{3,1}$	$C_{4,1}$	$D_{1,1}$	…………..	$D_{10,1}$	$D_{11,1}$
$C_{1,2}$	$C_{2,2}$	$C_{3,2}$	$C_{4,2}$	$D_{1,2}$	…………..	$D_{10,2}$	$D_{11,2}$
. .	. .	. .	. .	. .	. .	. .	. .
$C_{1,L}$	$C_{2,L}$	$C_{3,L}$	$C_{4,L}$	$D_{1,L}$	…………..	$D_{10,L}$	$D_{11,L}$

Fig. 5 The RS- RAID structure.

Let's look at some basic properties of RS code first. Not like general linear block codes such as hamming code or even parity code, the basic unit (or symbol) of RS code is non-binary. Their symbol size may be $\{0 \sim p^m\}$, where *p* is a prime and *m*≧ 1.

What follows are the parameters of an (*n*, *k*) RS code

$$\begin{aligned} n &= 2^m - 1 \\ k &= 2^m - 1 - 2t, \\ n - k &= 2t \\ d_{\min} &= 2t + 1 \end{aligned} \tag{12}$$

where *t* is the tolerable number of errors of the codes, and $d_{\min}$ is the minimum Hamming distance. Apparently, RS code is also some kind of MDS (Maximum-distance-separable) codes because of its minimum distance.

$$d_{\min} = 2t + 1 = n - k - 1, \tag{13}$$

In order to construct an RS based coding environment, the following steps should be taken in the beginning stage.

Step 1: Choose the value of m, defined from *GF*(2*m*) and the maximum number of disks in our system cannot be more than 2*m* -1. Meanwhile, the basic elements of *GF*(2*m*) consist of the set: {0, *αz* }, where Z = 0 ~ *m*-2.

Step 2: Determine the number of failed disks, *R*, that the system can tolerate and recover. Then we use *R* to represent the number of roots in the generator polynomial g(x). For instance, if the system wants to recover up to 4 disk failures that may occur simultaneously, its generator polynomial must have 4 roots.

Recently, researchers have pointed out that the RS based RAID systems can recover multiple erasures while other schemes, such as the Even Odd code, can only recover at most two erasure disks at one time.

We have shown several basic concepts of RS code in RAID 6 system. Here, we begin explaining more details in its implementation and system optimization as well. If our system needs to tolerate up to two drives that fails simultaneously, the generator polynomial of the RS code could be designed as follows:

$$\begin{aligned} g(x) &= (x - \alpha^0)(x - \alpha^1) \\ &= x^2 - (1 + \alpha)x + \alpha \\ &= x^2 + \alpha^4 x + \alpha \end{aligned}, \tag{14}$$

We use the systematic coding method to encode codeword from original messages.

III. SYSTEM OPTIMIZATION AND RS CODEC DESIGN

The coding procedure in our scheme not only follows the rules of mapping to systematic codes, but also builds a look-up table with an aspect of the constant multiplier.

This algorithm, however, must obtain outcomes after all symbols are sent into registers. It is not useful and efficient to construct a RAID system with lots of disk reads and writes. For our specific purposes of reads and writes, we use constant multipliers to build another systematic-code encoder.

The RS code is a class of linear block codes, so its computation must satisfy a linear property, that is, we can treat each data symbol (drive) independently. In other words, any change in a data drive would affect the checksum symbols (drives) independently. Here, we deduce the linear property of our RS RAID model using constant multipliers as follows:

$$\begin{aligned} b(x) &= x^{n-k} m(x) \bmod g(x) \\ &= x^{n-k}(m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}) \bmod g(x) \\ &= (x^{n-k} m_0 \bmod g(x)) + (x^{n-k+1} m_1 \bmod g(x)) + \cdots + (x^{n-1} m_{k-1} \bmod g(x)) \\ &= m_0 (x^{n-k} \bmod g(x)) + m_1 (x^{n-k+1} \bmod g(x)) + \cdots + m_{k-1} (x^{n-1} \bmod g(x)) \end{aligned} \quad (15)$$

For that reason, the effect of each data symbol (drive) could be computed separately to see how it works on the checksum symbols first. On the other hand, this property also allows engineers to apply parallel processing skills in order to improve the coding performance in our scheme. Then the complete checksum symbols must be computed by accumulating the effects from all independent drives.

Algorithm for Building Checksum Symbols (Encoding Procedure):

Step 1: First, multiply (or shift) the message polynomial m(x) by x^{n-k}.

Step 2: Construct the static-checksum table. Computing the term, $[m_i x^{n-k} \bmod g(x)]$, where each m_i is equal to the multiplicative identity 1 in *GF*(x), we would know what the effect is in each location (drive).

Step 3: By using the table built in Step 2, the checksum symbols b(x) can be obtained by multiplying all the values of the static-checksum table by the practical value of m(x). It can be represented as

$$m_0\left(x^{n-k} \bmod g(x)\right) + m_1(x^{n-k+1} \bmod g(x)) + \cdots, \quad (16)$$

We can easily apply the constant multiplier to execute all the computations after constructing the static-checksum table, because all the values of the constant table from Step 2 are fixed.

A. Reduced Static-Checksum Table Approach

The encoding process is still very crucial due to many XOR gates operating at the same time, even after constructing the previous look-up table. In the case of *GF*(2^4), for example, applying a constant multiplier with only one variable can build a table, called the constant-multiplier-coefficient table (abbreviated as *CMC* table), as shown in Table 2, where $A = a_1 + a_2\alpha + a_3\alpha^2 + a_4\alpha^3$ is the variable with 4 coefficients $a_1 \sim a_4$, and $a_1' \sim a_4'$ are coefficients of A' which is generated after being multiplied by α^z, where *z* = 0 ~ 14. Here, we take the Galois Field of 2^4, i.e. *GF*(2^4), and take p(x)= 1+x + x^4 to be the primitive polynomial which is used to construct all elements of the *GF*(2^4). Thereby, the elements correlated with symbols are shown in Table 1.

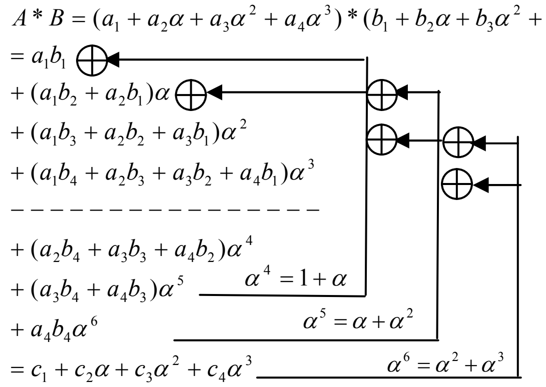
Two common multipliers will be introduced. One is called general multiplier, while the other proposed by us is called constant multiplier. The general multiplier can perform a multiplication by two variables. It has the advantage of generalization. The general multiplier could have variations through choosing different bases of elements.

The simplest and most common general multiplier is introduced as follows. Assume that there are two variables, *A* and *B*, and they could be presented as:

$$A = a_1 + a_2\alpha + a_3\alpha^2 + a_4\alpha^3 \quad , \quad B = b_1 + b_2\alpha + b_3\alpha^2 + b_4\alpha^3 \quad ,$$

respectively. Then a computation of *A*B=C* can be processed as:

$$\begin{aligned} A * B &= (a_1 + a_2\alpha + a_3\alpha^2 + a_4\alpha^3) * (b_1 + b_2\alpha + b_3\alpha^2 + b_4\alpha^3) \\ &= a_1 b_1 \\ &+ (a_1 b_2 + a_2 b_1)\alpha \\ &+ (a_1 b_3 + a_2 b_2 + a_3 b_1)\alpha^2 \\ &+ (a_1 b_4 + a_2 b_3 + a_3 b_2 + a_4 b_1)\alpha^3 \\ &\text{-----------------} \\ &+ (a_2 b_4 + a_3 b_3 + a_4 b_2)\alpha^4 \qquad \alpha^4 = 1 + \alpha \\ &+ (a_3 b_4 + a_4 b_3)\alpha^5 \qquad \alpha^5 = \alpha + \alpha^2 \\ &+ a_4 b_4 \alpha^6 \qquad \alpha^6 = \alpha^2 + \alpha^3 \\ &= c_1 + c_2\alpha + c_3\alpha^2 + c_4\alpha^3 \end{aligned} \quad (17)$$



where $C = c_1 + c_2\alpha + c_3\alpha^2 + c_4\alpha^3$, and $\alpha^4 \sim \alpha^6$ could be replaced with the equation: $\alpha^4 = 1 + \alpha$ formed by the rules of *GF*(2^4). The following is a circuit diagram for a general multiplier in *GF*(2^4). A multiplier of *GF*(2^4) is made of 16 AND-gates and 8 XOR-gates as well. It is not the best choice in terms of cost in constructing this special circuit to achieve the operation performance. Therefore, for the system in our approach, we use the constant multiplier instead of the general multiplier.

TABLE 1. ELEMENTS IN GF(2^4).

Symbols	Standard Basis	Physical Data
0	0	0000
1	1	1000
α	α	0100
α^2	α^2	0010
α^3	α^3	0001
α^4	$1+\alpha$	1100
α^5	$\alpha+\alpha^2$	0110
α^6	$\alpha^2+\alpha^3$	0011
α^7	$1+\alpha \quad +\alpha^3$	1101
α^8	$1 \quad +\alpha^2$	1010
α^9	$\alpha \quad +\alpha^3$	0101
α^{10}	$1+\alpha+\alpha^2$	1110
α^{11}	$\alpha+\alpha^2+\alpha^3$	0111
α^{12}	$1+\alpha+\alpha^2+\alpha^3$	1111
α^{13}	$1 \quad +\alpha^2+\alpha^3$	1011
α^{14}	$1 \quad +\alpha^3$	1001

B. The constant-multiplier-coefficient table.

A constant multiplier with only one variable can be easily understood. The only variable in this multiplier would be appropriately multiplied by one of the constant terms, from α^1 to α^{2^m-2}. Let us take a look at the simplest example: $A*\alpha^1$ in ($GF(2^4)$).

$$\begin{aligned} A*\alpha &= (a_1 + a_2\alpha + a_3\alpha^2 + a_4\alpha^3)*\alpha \\ &= a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + a_4\alpha^4 \\ &= a_4 + (a_1 + a_4)\alpha + a_2\alpha^2 + a_3\alpha^3 \end{aligned}, \quad \text{where } (\alpha^4 = 1 + \alpha), \tag{18}$$

From the above, it uses only an XOR gate to perform this computation. Similarly, a constant multiplier doesn't need to use any AND gates but only a few XOR gates to obtain the result cost effectively and correctly. Figure 6 is a basic logic diagram for a constant multiplier.

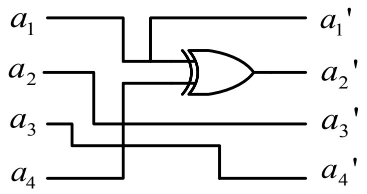


Fig. 6. Logic diagram of $A*\alpha^1$.

Thereby, we can get a complete table of a constant multiplier: A*$\alpha^2 \sim \alpha^{2^m-2}$ in $(GF(2^4))$ as shown in Table 2.

TABLE 2. THE CONSTANT-MULTIPLIER-COEFFICIENT TABLE.

A'	a_1'	a_2'	a_3'	a_4'
A*1	a_1	a_2	a_3	a_4
A*α	a_4	a_1+a_4	a_2	a_3
A*α^2	a_3	a_3+a_4	a_1+a_4	a_2
A*α^3	a_2	a_2+a_3	a_3+a_4	a_1+a_4
A*α^4	a_1+a_4	$a_1+a_2+a_4$	a_2+a_3	a_3+a_4
A*α^5	a_3+a_4	a_1+a_3	$a_1+a_2+a_4$	a_2+a_3
A*α^6	a_2+a_3	a_2+a_4	a_1+a_3	$a_1+a_2+a_4$
A*α^7	$a_1+a_2+a_4$	$a_1+a_3+a_4$	a_2+a_4	a_1+a_3
A*α^8	a_1+a_3	$a_2+a_3+a_4$	$a_1+a_3+a_4$	a_2+a_4
A*α^9	a_2+a_4	$a_1+a_2+a_3+a_4$	$a_2+a_3+a_4$	$a_1+a_3+a_4$
A*α^{10}	$a_1+a_3+a_4$	$a_1+a_2+a_3$	$a_1+a_2+a_3+a_4$	$a_2+a_3+a_4$
A*α^{11}	$a_2+a_3+a_4$	a_1+a_2	$a_1+a_2+a_3$	$a_1+a_2+a_3+a_4$
A*α^{12}	$a_1+a_2+a_3+a_4$	a_1	a_1+a_2	$a_1+a_2+a_3$
A*α^{13}	$a_1+a_2+a_3$	a_4	a_1	a_1+a_2
A*α^{14}	a_1+a_2	a_3	a_4	a_1

Now, if we take a generator polynomial:

$$\begin{aligned} g(x) &= (x-\alpha^0)(x-\alpha^1) \\ &= x^2 - (1+\alpha)x + \alpha \\ &= x^2 + \alpha^4 x + \alpha \end{aligned}, \tag{19}$$

with a capability to tolerate up to two erasures, the checksums b(x)=C$_2$x+C$_1$ could be shown as Table 3. In order to obtain the last column, indicated as the number of XOR operations after reduction in Table 3, our approach consists of the following steps:

Step 1: For each location of the static-checksum table, two values of checksums, C$_1$ and C$_2$, are marked. And then in the *CMC* table, i.e. Table 2, each marked value could be represented as 4 parts of a single row.

Step 2: Comparing each part of the two rows, there might be some common terms in both rows which we marked in Step 1. If so, we could reduce half of these common terms until there is no more common term between the two rows.

Step 3: Finally, the value of the last column in Table 3 can be accumulated by the rest of XOR operations in each part of the two marked rows in Table 2.

TABLE 3. THE REDUCED STATIC-CHECKSUM TABLE WITH M(X) = 1.

Location	$m(x)x^{n-k}$	C$_1$	C$_2$	The number of XOR operations	The number of XOR operations after reduction
D$_1$	x^2	α	α^4	6	5
D$_2$	x^3	α^5	α^{10}	14	9
D$_3$	x^4	α^{11}	α^{12}	14	9
D$_4$	x^5	α^{13}	α^6	8	6
D$_5$	x^6	α^7	α^9	14	9
D$_6$	x^7	α^{10}	α^5	14	9
D$_7$	x^8	α^6	α^{13}	8	6
D$_8$	x^9	α^{14}	α^3	4	4
D$_9$	x^{10}	α^4	α	6	5
D$_{10}$	x^{11}	α^2	α^8	8	6
D$_{11}$	x^{12}	α^9	α^7	14	9
D$_{12}$	x^{13}	α^8	α^2	8	6
D$_{13}$	x^{14}	α^3	α^{14}	4	4

Example 1: The reduction of location D2 in the Static-Checksum Table.

Step 1: C$_1$=α^5 and C$_1$=α^{10} and therefore, we mark the rows A*α^5 and A*α^{10}.

Step 2: By comparing the following two marked rows in Table 4.

TABLE 4. COMPARING TWO MARKED ROWS.

A'	a_1'	a_2'	a_3'	a_4'
A*α^5	a_3+a_4	a_1+a_3	$a_1+a_2+a_4$	a_2+a_3
A*α^{10}	$a_1+a_3+a_4$	$a_1+a_2+a_3$	$a_1+a_2+a_3+a_4$	$a_2+a_3+a_4$

as we can see, (a_3+a_4), (a_1+a_3), $(a_1+a_2+a_4)$, and (a_2+a_3) are all the common terms between the two rows. Hence, after applying this approach, the total number of XOR operations can be reduced by 5.

Step 3: The number of required XOR operations after Step 2 is 14-5 = 9.

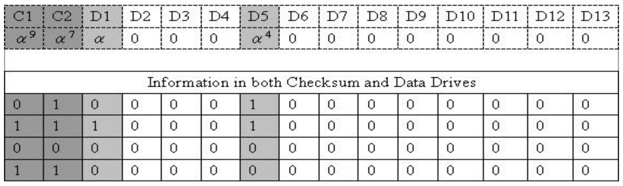
In addition, this scheme applies the shortened code method as well to achieve better performance on the coding process. With this method, active drives are placed on some exact locations first. This disk location arrangement is based on which disk requires fewer XOR-gates after applying our reduction approach. That is to say, in the case of Table 3, to reach higher performance of computations, the locations must be arranged based on the order, D8, D13, D1, D9. etc. Next, in Example 2, we will demonstrate a full picture of our encoding procedure.

Example 2: Let us assume that a message polynomial, $m(x) = \alpha + \alpha^4 x^4$, has to be stored into an empty RS RAID in $GF(2^4)$. All the data in the checksum drives can be computed as follows. By α, from the location D1, x^2 of Table 2, we can put data $\alpha * \alpha$ and $\alpha * \alpha^4$ into two checksum drives separately. Similarly, by α^4, from D5 and x^6 in Table 3, the stored data of the two checksum drives are $\alpha^4 * \alpha^7$ and $\alpha^4 * \alpha^9$ respectively and therefore, values stored in the two checksum-drives after the above process are:

$$C1 = (\alpha * \alpha) \oplus (\alpha^4 * \alpha^7) = \alpha^9, \qquad (20)$$

$$C2 = (\alpha * \alpha^4) \oplus (\alpha^4 * \alpha^9) = \alpha^7, \qquad (21)$$

Figure 7 illustrates the data placement in our RS RAID system, where C1 and C2 are checksum drives, D1 ~ D13 are data drives. For each column, values of the second row are corresponding symbols to their binary values.



C1	C2	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13
α^9	α^7	α	0	0	0	α^4	0	0	0	0	0	0	0	0
Information in both Checksum and Data Drives														
0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 7. Data allocation in the RS RAID system with shortened code method.

Here we also mention a smart skill Shortened RS code. Although the codeword length of standard RS codes is limited to $2^m - 1$, we still can use some skills to change the codeword length to become the length what we need in practice. These skills can produce different RS codes forms, such as punctured RS code, shortened RS code, and extended RS code. Here we only introduce shortened RS code which is suitable for use in RS RAID. Once $GF(2^m)$ is chosen, we can define (n, k) standard RS code, where $n = 2^m - 1$. If we do encoding by Systematic coding, we can set s coefficients of $m(x)$ to be zero. Therefore, the codeword must have s symbols with zero value. Then, others are $(n\text{-}s, k\text{-}s)$ shortened RS codes. These skills are important in the applications of RAID system. For example, using these skills can make us decrease or increase the number of hard disks freely. That is when a hard disk is not used, we can set the hard disk to zero. We can add the hard disk any time when the hard disk is needed in the future.

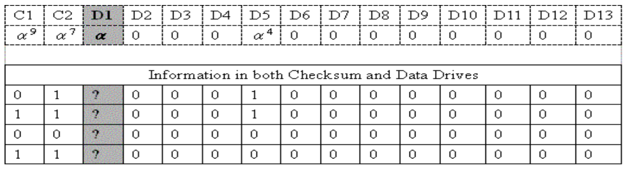
C. RS RAID decoder

Two cases of decoding algorithm are discussed over $GF(2^4)$, and they are carried out by an equation solving method, called the Cramer's rule, directly.

1) Single Failed Disk

First of all, we take 1 to be one of the roots with consecutive powers in our generator polynomial, i.e. $g(x)=(x-\alpha^0)(x-\alpha^1)$. Therefore, in the case of single disk failure condition, the decoding would be performed as easily as the parity scheme of the RAID 5. From the equation: *Failed-Drive=S_0=Σ (All Normal Drives)*, recovering the failed disk needs merely to do XOR operations on the rest of active disks together.

Example 3: As in Example 2, assuming that only the data-drive D1 has been erased as shown in Figure 8.



C1	C2	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13
α^9	α^7	α	0	0	0	α^4	0	0	0	0	0	0	0	0
Information in both Checksum and Data Drives														
0	1	?	0	0	0	1	0	0	0	0	0	0	0	0
1	1	?	0	0	0	1	0	0	0	0	0	0	0	0
0	0	?	0	0	0	0	0	0	0	0	0	0	0	0
1	1	?	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 8. RAID with only a failed disk.

The original information of D1 could be recovered as:
D(1,1)=C(1,1)+C(2,1)+D(5,1)=0+1+1=*0*
D(1,3)=C(1,3)+C(2,3)+D(5,3)=0+0+0= *0*
D(1,2)=C(1,2)+C(2,2)+D(5,2)=1+1+1=*1*
D(1,4)=C(1,4)+C(2,4)+D(5,4)=1+1+0= *0* .

2) Two Failed Disks at the Same Time

In this case, in order to recover two disks which simultaneously fail, the decoding procedure in our scheme could be treated as solving a simultaneous-linear-equation with two unknown variables. Here the matrix form of this equation is as follows:

$$\begin{bmatrix} 1 & 1 \\ \alpha^i & \alpha^j \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = -\begin{bmatrix} S_0 \\ S_1 \end{bmatrix}, \qquad (22)$$

where i and j are the positions of the two failed disks in this condition, and then the syndrome:

$$S_K = \sum_{Z=0}^{n-1} CW_Z\left(\alpha^{KZ}\right), \qquad (23)$$

Eq.(23) is computed over all the normal drives. By the Cramer's rule, the two variables, A and B, could be represented as follows:

$$A = \frac{S_0\alpha^j + S_1}{\alpha^i + \alpha^j} \quad \text{and} \quad B = \frac{S_0\alpha^i + S_1}{\alpha^i + \alpha^j}, \qquad (24)$$

Furthermore, applying the same idea of the *CMC* table to build a table fulfilled with inverse-elements of $(\alpha^i + \alpha^j)$ in advance would be more efficient. This table can avoid the extra cost of implementation on designing an *ALU*.

IV. Results And Comparisons

The proposed RS codec is used in VHDL. During the hardware synthesis, time delays have been considered by restricting the longest path delay of the codec.

From Table 5, our approach indeed needs fewer XOR operators than the conventional RS codes in [5]. The Reed Solomon code proposed in the paper can process data in parallel. That is because the encoding process of the proposed Reed Solomon code can calculate the effects of each data drive to the checksum drives respectively.

Table 6 shows the implemented results based on several design methods [14-15] and our proposed method. The number of total gates is from the synthesized results excluding the FIFO memories. The new proposed architecture has been implemented by using about 25,000 gates, which is about 28% smaller than the other design result. The FIFO memory sizes have been reduced since the latency has been reduced. We have verified the correctness of the designed RS codec by using extensive simulation with randomly generated data and errors.

TABLE 5. THE NUMBER OF XOR GATES FOR THE XOR-BASED RS CODE, THE CONVENTIONAL RS CODE.

#of Disk Drives	The proposed XOR based Reed-Solomon	Conventional Reed-Solomon
7	1068	954
11	3020	3250
13	4392	5112
17	7968	10624
23	15554	24442
29	25704	46648
31	29700	56250
41	54200	124000
43	60018	142002

TABLE 6. COMPARISONS ON CIRCUIT SIZES OF RS DECODER/ENCODER

Hardware blocks (RS decoder/encoder)	Implementation results		
	[14]	[15]	Proposed Method
Encoder (used separately with decoder)	2,000	0	0
Syndrome computation	2,000	2,000	1,800
Polynomial expansion (and Encoder)	10,000	6,000	5,000
Modified Euclid's algorithm	24,000	15,000	12,000
Chien search	7,000	3,000	0
Error correction and verification	9,000	9,000	8,000
FIFO memory sizes	456*8 bits	351*8 bits	343*8 bits
Number of total gates (%)	54,000 (100%)	35,000 (65%)	25,000 (46%)
Application model or system	Digital VCRs	Digital VCRs	RAID 6 system

A. *Experimental setup*

We build an RAID 6 experimental system with six PCs as storage nodes, and an IBM server as the metadata server. Every PC consists of a 2.4GHz Intel Celeron CPU, 512 MB of RAM, two 80GB IDE disks and a D-link Gigabit Ethernet interface card. A simulator program, which can generate the CPU, memory, disk I/O and network I/O load, runs on every storage node to tune workload of a storage node. We have varied the parameter of read/write percentage in the I/O meter workloads and tuned load storage nodes with simulator program. Three ratios of read/write are explored: 100% read, 100% write and a mix of 50% read–50% write. Other primary parameters of I/O meter are constant as: 80% sequential access, transfer request size 64KB. Figure 9-11 shows our results with I/O meter. Comparing the I/O meter performances of the three methods with different system loads of the storage nodes, the proposed method has the best performance.

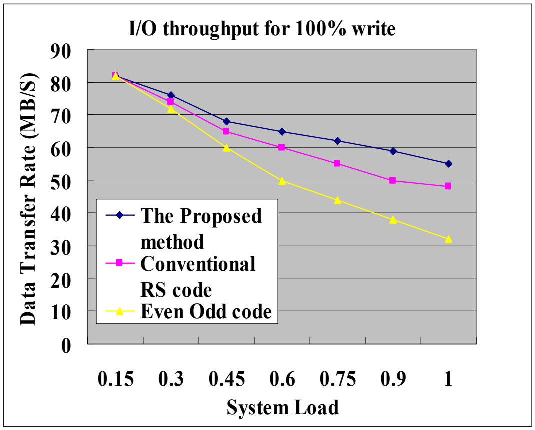


Fig. 9. I/O meter throughput for 100% write as varying load on storage nodes.

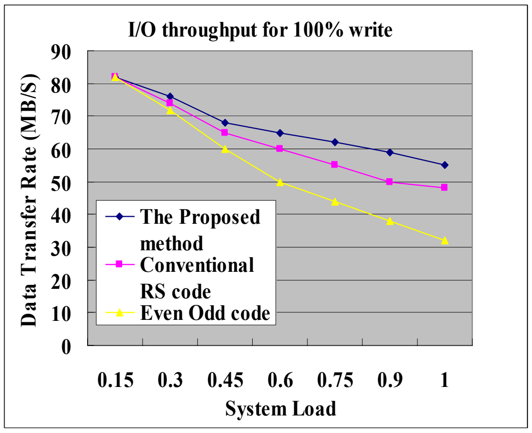


Fig. 10. I/O meter throughput for 100% write as varying load on storage nodes.

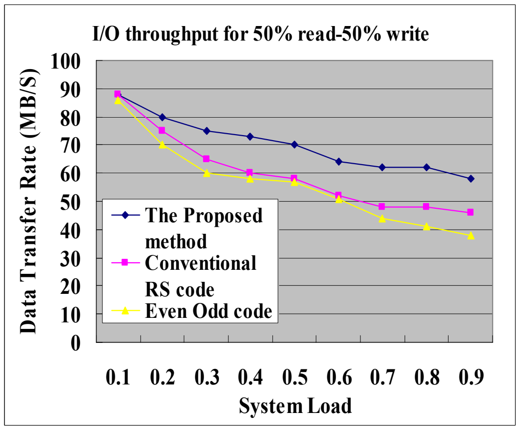


Fig. 11. I/O meter throughput for 50% read - 50% write as varying load on storage nodes.

B. *Simulation Waveform*

1) *The proposed encoder*

If m(x) = $\alpha + \alpha x^4$

Write D1: 0100; D5: 1100. Then results are C1: 0101; C2: 1101, as shown in Figure 12.

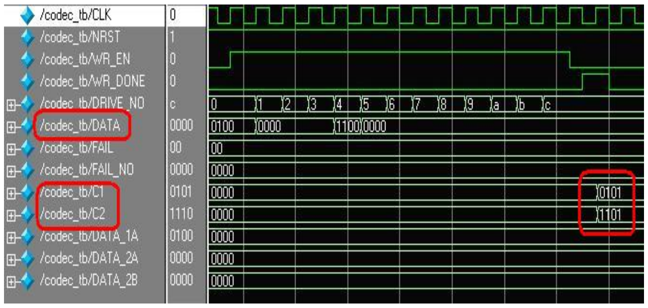


Fig. 12. Encoder simulation waveform.

2) The proposed decoder

If the 5th Hard Drive (Drive_NO:4) fails, the decoder will calculate the correct data (DATA_1A:1100) with all other Hard Drive data and C1, C2, as shown in Figure 13.

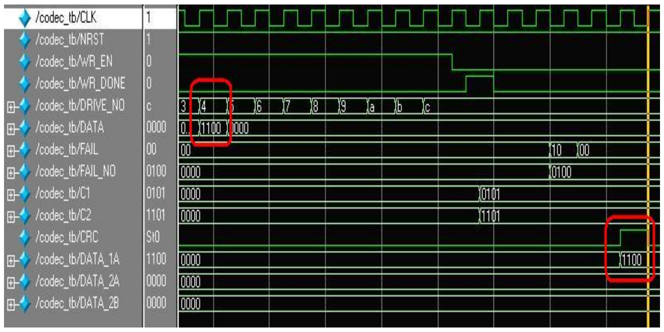


Fig. 13. Decoder simulation waveform.

V. Conclusion

In this paper, we design a low complexity RS codec for the RAID 6 system. We first constructed an analytical model to evaluate the computation complexity of RS code algorithm. According to the analytical model we generate an optimized RAID 6 system. In order to optimize RS algorithm, we proposed a look up table method to replace the calculation of spreading functions. It could dramatically reduce the hardware complexity by 28% with only about the original RS codec gate count. The proposed algorithm of RS codec achieves good performance on both read and write request. It can increase the I/O performance by 13-25% than conventional RS RAID when the system load is suitable or heaviest. Also, by taking into account of optimizing the throughput of the system, the proposed algorithm can provide the RAID system with better fault-tolerant capability. The proposed design leads to a real-time RS codec with low complexity profile RAID 6 system. This work can be implemented in general programmable processors, DSP and dedicated hardware.

Acknowledgment

This research was supported by the Ministry of Economics, Taiwan under contract number 94-EC-17-A-08-S1-0006 and the National Science Council, Taiwan under contract NSC-94-2213-E-002-112 and Department of Electronic Engineering, Lee-Ming Institute of Technology.

References

[1] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, *"RAID High-Performance, Reliability Secondary Storage"*, ACM Computing Surveys, June 1994, pp. 145-185

[2] S. Frolund, A. Merchant, Y. Saito, S. Spence and A. Veitch, *"A Decentralized Algorithm for Erasure-Coded Virtual Disks"*, International Conference on Dependable Systems and Networks, 125-134, IEEE, 2004.

[3] G. R. Goodson, J. J. Wylie, G. R. Ganger and M. K. Reiter, *"Efficient Byzantine-Tolerant Erasure-Coded Storage"*, International Conference on Dependable Systems and Networks, IEEE, 135-144, 2004.

[4] J. S. Plank and M. G. Thomason, *"A Practical Analysis of Low-Density Parity-Check Erasure Codes for Wide-Area Storage Applications"*, The International Conference on Dependable Systems and Networks, IEEE, June, 115-124, 2004.

[5] M. Blaum, J. Brady, J. Bruck, and J. Menon, *"EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures"*, IEEE Transactions on Comput., Feb. 1995, pp. 192-202.

[6] S. B. Wicker and V. K. Bhargava, *"Reed-Solomon codes and Their Applications"*, New York IEEE Press, 1994 ,

[7] Y. Jeong and W. Burleson, *"High-Level Estimation of High-Performance Architectures for Reed-Solomon Decoding"*, ISCAS, vol. 1, pp. 720-723, May, 1995.

[8] J. M. Hsu and C. L. Wang, *"An Area-Efficient VLSI Architecture for Decoding of Reed-Solomon Codes"*, ICASSP, vol. 6, pp. 3291-3294, May, 1996.

[9] T.K. Truong, J.H. Jeng, T.C. Cheng, *"A New Decoding Algorithm for Correcting Both Erasures and Errors of Reed-Solomon Codes"*, IEEE Transactions on Communications, March 2003, pp.381-388.

[10] D.V. Sarwate, N. R. Shanbhag, *"High-Speed Architectures for Reed-Solomon Decoders*", IEEE Transactions on VLSI, 2001, pp.641-655.

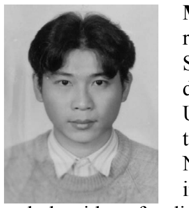
[11] G. Y. Lee, B. H. Kwuan, S. W. Lee, J. W. Jung, S. H. Nam, Y. S. Chun, D. I. Han, K. S. Park, Y. D. Calif., *"A VLSI Design of RS CODEC for Digital Data Recorder"*, ASIC Design Workshop, pp.115-124, 1996.

[12] T. Iwaki, T. Tanaka, E. Yamada, T. Okuda and T. Sasada, *"Architectures of A High Speed Reed-Solomon Decoder"*, IEEE Transactions on Consumer Electronics, vol. 40, no. 1, pp.91-92, Feb. 1994.

[13] C. C. Hsu, I. S. Reed and T. K. Truong, *"Use of the RS Decoder as an RS Encoder for Two-Way Digital Communications and Storage Systems"*, IEEE Transactions on Circuits and Systems For Video Technology, vol. 4, no. 1, pp.75-81, Feb. 1994.

[14] H. M. Shao and I. S. Reed, *"On the VLSI Design of Pipeline Reed-Solomon Decoder Using Systolic Arrays"*, IEEE Transactions on Computers, vol. 37, no. 10, pp.1273-1280, Oct. 1988.

[15] S. Kwon and H. Shin, *"An Area-Efficient VLSI Architecture of A Reed-Solomon Decoder/Encoder for Digital VCRs"*, IEEE Transactions on Consumer Electronics, vol. 43, no. 4, pp.1019-1027, Nov. 1997.



Min-An Song was born in 1966 in Taipei, Taiwan. He received the B.S. degrees in Electrical Engineering from St. John's University in 1983 and 1988 and M.S. degrees in Electrical Engineering from National Taiwan University in 1998 and 2000. He is currently working toward the PhD. degrees in Electrical Engineering from National Taiwan University. His research interests include high-speed and low-power VLSI architectures and algorithms for digital signal processing and communication applications with the emphasis on digital filters and computer arithmetic, parallel and distributed computing, embedded systems, and intelligent systems.

Sy-Yen Kuo is Dean of the College of Electrical and Computer Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan. He is also a Professor at the Department of Electrical Engineering, National Taiwan University where he is currently on leave and was the Chairman at the same department from 2001 to 2004. He received the BS (1979) in Electrical Engineering from National Taiwan University, the MS (1982) in Electrical & Computer Engineering from the University of California at Santa Barbara, and the PhD (1987) in Computer Science from the University of Illinois at Urbana-Champaign. He spent his sabbatical years as a Visiting Professor at the Computer Science and Engineering Department, the Chinese University of Hong Kong from 2004-2005 and as a visiting researcher at AT&T Labs-Research, New Jersey from 1999 to 2000, respectively. He was the Chairman of the Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan from 1995 to 1998, a faculty member in the Department of Electrical and Computer Engineering at the University of Arizona from 1988 to 1991, and an engineer at Fairchild Semiconductor and Silvar-Lisco, both in California, from 1982 to 1984. In 1989, he also worked as a summer faculty fellow at Jet Propulsion Laboratory of California Institute of Technology. His current research interests include dependable systems and networks, software reliability engineering, mobile computing, and reliable sensor networks. Professor Kuo is an IEEE Fellow. He has published more than 260 papers in journals and conferences. He received the distinguished research award between 1997 and 2005 consecutively from the National Science Council in Taiwan and is now a Research Fellow there. He was also a recipient of the Best Paper Award in the 1996 International Symposium on Software Reliability Engineering, the Best Paper Award in the simulation and test category at the 1986 IEEE/ACM Design Automation Conference(DAC), the National Science Foundation's Research Initiation Award in 1989, and the IEEE/ACM Design Automation Scholarship in 1990 and 1991

I-Feng Lan was born in I-Lan, Taiwan, in 1950. He received the B.S. degree in physics from National Taiwan University (NTU), Taipei, Taiwan, in 1972, and the Ph. D. degree in physics from University of California at Los Angeles (UCLA), USA, in 1983. He worked as a research scientist for a national defense institute in Taiwan, from 1975 to 1999. Since Aug. 1999, he joined the faculty of Department of Electronic Engineering, Lee-Ming Institute of Technology, Taiwan, where he is currently an Associate Professor. His research interests are in speech recognition and the application of microcontrollers in the consumer electronics.