

An XOR Based Reed-Solomon Algorithm for Advanced RAID Systems

Ping-Hsun Hsieh^Ψ, Ing-Yi Chen^Δ, Yu-Ting Lin^Ψ, and Sy-Yen Kuo^Ψ

^ΨDepartment of Electrical Engineering, National Taiwan University, Taipei, Taiwan

^ΔDepartment of Computer Science and Information Engineering,

National Taipei University of Technology, Taipei, Taiwan

michael@lion.ee.ntu.edu.tw^Ψ

ichen@ntut.edu.tw^Δ

sykuo@cc.ee.ntu.edu.tw^Ψ

Abstract

In this paper, a simple codec algorithm based on Reed-Solomon (RS) codes is proposed for erasure correcting in RAID (Redundant Array of Independent Disks) level 6 systems. Unlike conventional RS codes, here this scheme with a mathematical reduction method, called Reduced Static-Checksum Table Approach, could improve coding performance, including encoding and decoding procedures. This scheme uses current industrial RAID-5 controllers as well as the regular hardware only and without extra cost in adding any new equipment. Moreover, our algorithm is able to expand to correct multiple failed-disks while others, EvenOdd codes for example, cannot do so. Also, this scheme performs all computations with only simple exclusive-OR (XOR) operators as same as EvenOdd codes. For most RAID architectures, this new XOR-based RS code could adapt to implementation in terms of reliability, flexibility and lower cost.

1. Introduction

In storage systems, especially for large disk arrays, reliability is getting critical while storage systems scale up. In [1], it has demonstrated that disk failures would be a daily event in petabyte-scale file systems (where a petabyte is 10^{15} bytes). So, how to improve the capability of detecting or even correcting failures has been a significant issue for large storage systems. RAID systems which could be classified from level 0 to 6 are commonly used to achieve this issue. Unlike other levels, RAID level 6, or so-called RAID 6, not only provide correcting capability, but also could recover at least two disk failures simultaneously. Usually each specific algorithm in RAID 6 performs a particular parities distribution [2]. Over the last two decades, lots of erasure-correcting codes' algorithms in RAID 6 have been proposed such as EvenOdd code [3], Reed-Solomon (RS) Code [4], and X-Code [5].

RS code, a very popular error control code, has been studied in various applications, especially in communication systems [6]. Also, researchers have suggested some RS based solutions to avoid hazards happening in RAID-like systems [7-10], but those schemes might not be suitable to meet the desire of recovering system as quick as EvenOdd codes.

Therefore, in this paper, we present an XOR-based RS codec scheme, which uses a reduced static-checksum table approach, to manipulate the erasures-only hazard. Basically, in this new scheme, it is similar to the conventional RS codec algorithm [4] that involves pipeline procedure, which consists of Syndrome Calculation (SC), Key Equation Solver (KES), Chien Search (CS), and Forney Algorithm (FA), but without involving the portions of either KES or CS. Both KES and CS are used to locate errors and need extra cost of finite field operators. For the

erasure-only RAID system, system controller is not necessary to locate such errors since the individual disk devices have their own error-control coding mechanisms to recover from errors [2]. Moreover, usually in large disk arrays, failures of a single storage device could be detected by the storage system controllers and then could be marked as well [11]. Since device failures can be marked as erasures, erasure-correcting codes are usually employed to achieve the information recovery. Hence, compared with the traditional RS codec scheme, a simpler scheme is proposed in this paper in terms of less cost, improving flexibility and reliability.

The rest of this paper is organized as follows. Section 2 describes general ideas in our encoding algorithm for the erasure-only RS-RAID system and the main feature of our scheme, called reduced static-checksum table approach, is suggested as well. Our decoding approach will be shown in Section 3. Section 4 gives results of performance analysis and also a comparison in the number of XOR operations with the EvenOdd, the traditional RS-RAID structure, and the XOR-based RS code as well. Section 5 closes with conclusion and future research works.

2. RS-RAID Encoder

The encoding procedure in our scheme not only follows the rules of a mapping with systematic codes, but also builds a look-up table with an aspect of the constant multiplier. A mapping, in most RS encoders, from the message space in k-dimension to the code space in n-dimension is the most general approach of encoding, and the encoder also usually generates systematic codes, namely, message bits of a symbol could be presented explicitly in its corresponding codeword. Equation $cw(x) = b(x) + m(x)x^{n-k}$ shows a result after applying the systematic coding method, and $cw(x)$, $b(x)$, and $m(x)$ are codeword, checksum, and message, respectively. Let a single disk with K bytes (8K bits) be separated into L (=8K/W), where W is the codeword length in the RS code [8]. Figure 1 illustrates an example of a mapping. If W=4, there could be 4 checksum drives and 11 data drives in this system, i.e., $b(x)=\{C1, C2, C3, C4\}$ and $m(x)=\{D1, D2, D3, \dots, D10, D11\}$.

The static-checksum table, as a main feature in our scheme, lets all encoding operations in our RS scheme could be computed only using XOR gates and will be discussed later. Next, we would like to show preliminary works of designing the RS code for RAID-like system first.

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

Figure 1. The RS- RAID Structure.

2.1. Preliminary Works to Construct RS Codes

According to [4], in order to construct an RS-based coding environment, 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, basic elements of $GF(2^m)$ consist of a set: $\{0, \alpha^z\}$, where $z = 0 \sim m-2$, and as we can see, these symbols appear wherever in this paper.

Step 2: Decide the number of tolerable failed disks, R for example, which the system can recover. Then we use R to represent the number of roots in the generator polynomial $g(x)$. For instance, if the system would like to recover up to 4 disks that fail simultaneously, its generator polynomial must be taken with 4 roots.

Recently, researchers has pointed out that RS-based RAID systems could be implemented to recover multiple erasures while other schemes, such as the EvenOdd, could only recover only up to two erasure disks at one time [8].

2.2 Basic Scheme in RS Encoding

The RS code is a class of linear block codes [4], so its computation must satisfy a linear property, that is to say, we can treat each data symbol (drive) independently. In other words, any changes in each data drive would affect checksum symbols (drives) independently. Here, we deduce the linear property of our RS-RAID model using constant multipliers as follows:

$$\begin{aligned}
 b(x) &= x^{n-k} m(x) \bmod g(x) \\
 &= x^{n-k} (m_0 + m_1 x + m_2 x^2 + \dots + m_{k-1} x^{k-1}) \bmod g(x) \\
 &= (x^{n-k} \bmod g(x)) + (x^{n-k+1} m_1 \bmod g(x)) + \dots + (x^{n-1} m_{k-1} \bmod g(x)) \\
 &= m_0 (x^{n-k} \bmod g(x)) + m_1 (x^{n-k+1} \bmod g(x)) + \dots + m_{k-1} (x^{n-1} \bmod g(x))
 \end{aligned} \tag{Eq.1}$$

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

Algorithm for Building Checksum Symbols (Encoding Procedure):

Step 1. Premultiply (or shift) the message polynomial $m(x)$ by x^{n-k} .

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

Step 3. By using the table we built in Step 2, checksum symbols $b(x)$ would be obtained by multiplying all values of static-checksum table by the practical value of $m(x)$. It can be presented as $m_0(x^{n-k} \bmod g(x)) + m_1(x^{n-k+1} \bmod g(x)) + \dots$. We are able to easily apply the constant multiplier to operate all computations after constructing the static-checksum table, because all values of the constant table from Step 2 are fixed.

2.3 Reduced Static-Checksum Table Approach

The encoding process is still very crucial due to operating too many XOR gates, even after constructing the previous look-up table. Therefore, a further work to reduce the number of required XOR gates during the encoding process is proposed in this paper. In case of $GF(2^4)$, for example, applying the aspects of constant multipliers with only a variable could build a table, called constant- multiplier- coefficients table (abbreviated as CMC table), as Table 1, 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 $z = 0 \sim 14$:

Table 1. The constant- multiplier- coefficients table.

A'	a_1'	a_2'	a_3'	a_4'
$A * \alpha^0$	a_1	a_2	a_3	a_4
$A * \alpha^1$	a_4	$a_1 + a_4$	a_2	a_3
$A * \alpha^2$	a_3	$a_3 + a_4$	$a_1 + a_4$	a_2
$A * \alpha^3$	a_2	$a_2 + a_3$	$a_3 + a_4$	$a_1 + a_4$
$A * \alpha^4$	$a_1 + a_4$	$a_1 + a_2 + a_4$	$a_2 + a_3$	$a_3 + a_4$
$A * \alpha^5$	$a_3 + a_4$	$a_1 + a_3$	$a_1 + a_2 + a_4$	$a_2 + a_3$
$A * \alpha^6$	$a_2 + a_3$	$a_2 + a_4$	$a_1 + a_3$	$a_1 + a_2 + a_4$
$A * \alpha^7$	$a_1 + a_2 + a_4$	$a_1 + a_3 + a_4$	$a_2 + a_4$	$a_1 + a_3$
$A * \alpha^8$	$a_1 + a_3$	$a_2 + a_3 + a_4$	$a_1 + a_3 + a_4$	$a_2 + a_4$
$A * \alpha^9$	$a_2 + a_4$	$a_1 + a_2 + a_3 + a_4$	$a_2 + a_3 + a_4$	$a_1 + a_3 + a_4$
$A * \alpha^{10}$	$a_1 + a_3 + a_4$	$a_1 + a_2 + a_3$	$a_1 + a_2 + a_3 + a_4$	$a_2 + a_3 + a_4$
$A * \alpha^{11}$	$a_2 + a_3 + a_4$	$a_1 + a_2$	$a_1 + a_2 + a_3$	$a_1 + a_2 + a_3 + a_4$
$A * \alpha^{12}$	$a_1 + a_2 + a_3 + a_4$	a_1	$a_1 + a_2$	$a_1 + a_2 + a_3$
$A * \alpha^{13}$	$a_1 + a_2 + a_3$	a_4	a_1	$a_1 + a_2$
$A * \alpha^{14}$	$a_1 + a_2$	a_3	a_4	a_1

Now, if we take a generator polynomial:

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

with a capability to tolerate up to two erasures, the checksums $b(x) = C_2x + C_1$ could be shown as Table 2.

In order to obtain the sixth column, indicated the number of XOR operations after the reduction, in Table 2, our approach consists of following steps:

Table 2. 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
D1	x^2	α	α^4	6	5
D2	x^3	α^5	α^{10}	14	9
D3	x^4	α^{11}	α^{12}	14	9
D4	x^5	α^{13}	α^6	8	6
D5	x^6	α^7	α^9	14	9
D6	x^7	α^{10}	α^5	14	9
D7	x^8	α^6	α^{13}	8	6
D8	x^9	α^{14}	α^3	4	4
D9	x^{10}	α^4	α	6	5
D10	x^{11}	α^2	α^8	8	6
D11	x^{12}	α^9	α^7	14	9
D12	x^{13}	α^8	α^2	8	6
D13	x^{14}	α^3	α^{14}	4	4

Step 1. For each location of the static-checksum table, first, two values of checksums, C_1 and C_2 , are marked. And then in the CMC table, i.e., Table 1, each marked value could be represented to 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 Step1. If so, we could reduce half of these common terms until there is no more common term between both rows.

Step 3. Finally, the value of the sixth column in Table 2 can be accumulated the rest of XOR operations in each part of the two marked rows in Table 1.

<Example 1> Reducing location D2 in the Static-Checksum Table

Step 1. $C_1 = \alpha^5$ and $C_1 = \alpha^{10}$, therefore, we marked the rows $A * \alpha^5$ and $A * \alpha^{10}$.

Step 2. Through comparing the following two marked rows,

$A * \alpha^5$	$a_3 + a_4$	$a_1 + a_3$	$a_1 + a_2 + a_4$	$a_2 + a_3$
$A * \alpha^{10}$	$a_1 + a_2 + a_4$	$a_1 + a_2 + a_3$	$a_1 + a_2 + a_3 + a_4$	$a_2 + a_3 + a_4$

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 XOP operations could reduce 5 XOR operations.

Step 3. The number of required XOR operations after processing step 2 is $14 - 5 = 9$. ■

Besides, this scheme applies the shortened code method as well to achieve a better performance on coding process [4]. With this method, active drives are placed on some exact locations first. This disk-location-arrangement is based on which disk costs fewer XOR-gates after our reducing approach. That is to say, in the case of Table 2, to reach higher performance of computations, the locations must be arranged with the order, D8, D13, D1, D9, etc.

Next, in Example 2, it demonstrates a full picture of our encoding procedure.

<Example 2> Lets assume that a message polynomial, $m(x) = \alpha + \alpha^4 x^4$, has to be stored into an empty RS-RAID in $GF(2^4)$. And all data in checksum drives could be computed as follows: By α : from the location D1: x^2 of the Table 2, we could put data $\alpha * \alpha$ and $\alpha * \alpha^4$ into two checksum drives separately. Similarly, by x^4 , from D5: x^6 in Table 2, the stored data of the two checksum drives are $\alpha^4 * \alpha^7$ and $\alpha^4 * \alpha^9$. Therefore, values stored in the two checksum-drives after the above processes are:

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

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

Figure 2 illustrates data placement in our RS-RAID system, where C1 and C2 are checksum drives, D1 ~ D13 are data drives, and 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

Figure 2. Data allocation in RS-RAID System with Shortened Code Method

3. RS-RAID Decoder

In this section, two cases of decoding algorithm are discussed over $GF(2^4)$, and they are carried out by a solving equations method, called crammer rule, directly.

3.1 Single Failed Disk

First of all, we takes 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 failed disk condition, the decoding would be performed easily as same as the parity scheme of the RAID level 5. From the equation: *Failed-Drive* = $S_0 = \sum(\text{All Normal Drives})$, recovering the failed disk needs to do XOR operations in the rest of active disks together merely.

<Example 3> As in Example 2, assume that only the data-drive D1 has been erased as Figure 3.

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

Figure 3. An RAID with only a failed disk

Therefore, the original information of D1 could be recovered through:

$$\begin{aligned}
 D(1,1) &= C(1,1) + C(2,1) + D(5,1) = 0 + 1 + 1 = 0 & D(1,3) &= C(1,3) + C(2,3) + D(5,3) = 0 + 0 + 0 = 0 \\
 D(1,2) &= C(1,2) + C(2,2) + D(5,2) = 1 + 1 + 1 = 1 & D(1,4) &= C(1,4) + C(2,4) + D(5,4) = 1 + 1 + 0 = 0 \quad \blacksquare
 \end{aligned}$$

3.2 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.

$$\begin{bmatrix} 1 & 1 \\ \alpha^i & \alpha^j \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = - \begin{bmatrix} S_0 \\ S_1 \end{bmatrix}$$

, where i and j are both the very positions of the two failed disks in this condition, and then syndrome: $S_k = \sum_{z=0}^{n-1} c_w \alpha^{kz}$ is computed from all normal drives. By the crammer rule, the two variables, A and B, could be represented as follows respectively:

$$A = \frac{S_0 \alpha^j + S_1}{\alpha^i + \alpha^j} \quad B = \frac{S_0 \alpha^i + S_1}{\alpha^i + \alpha^j} \quad \text{Eq.(2)}$$

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.

4. Results and Comparisons

In order to demonstrate how the encoding performance of our XOR-based RS algorithm is, we implement both CMC table and reduced static-checksum table in $GF(2^8)$ to count the total

number of XOR operators. Besides, a disk drive set {7, 11, 13, 17, 23, 29, 31, 41, 43} is our experimental example. Here, Figure 4 shows corresponding curves to Table 3.

Table 3. # of XOR gates while encoding with the XOR-based RS, the conventional RS and the EvenOdd code.

# of Disk Drives	EvenOdd Codes	XOR based Reed-Solomon Codes	Conventional Reed-Solomon Codes
7	664	1068	954
11	1752	3020	3250
13	2488	4392	5112
17	4344	7968	10624
23	8088	15554	24442
29	12948	25704	46648
31	14872	29700	56250
41	26232	54200	124000
43	28888	60018	142002

From both Table 3 and Figure 4, as we can see, the EvenOdd code performs a more efficient encoding capability than what the XOR-based RS code does. Although our approach indeed needs less XOR operators than the conventional RS code did in [3].

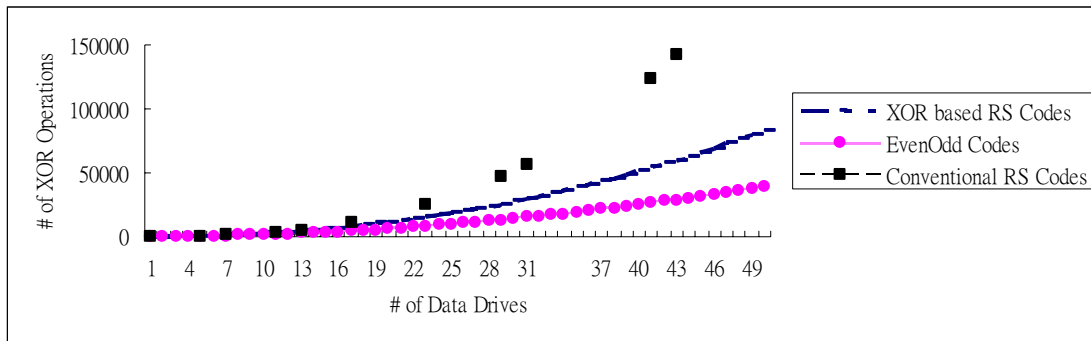


Figure 4. Curves plotted by the # of XOR gates while encoding with the XOR-based RS, the conventional RS and the EvenOdd code.

Moreover, here Figure 5 shows that traditionally EvenOdd codes need to be implemented by coding through a 3-Dimension structure while our algorithm can be easily implemented through a 2-Dimension array structure as Figure 1. For Figure 5, if there is a byte data changed, we need to deal with eight codewords from Page 0 to Page 7. Therefore, the data update might be an overhead to EvenOdd codes, but it does not happen in our scheme because we chose 8 bits to be the length of a codeword in the 2D array structure.

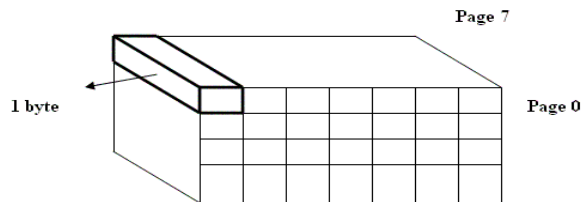


Figure 5. The 3-Dimension structure of EvenOdd implementation.

5. Conclusions

In this paper, we proposed an XOR-only RS-RAID algorithm with two auxiliary tables, the CMC table and the reduced static-checksum table, for not only constructing the XOR-based RS algorithm, but also speeding our scheme up. The above features also make those advanced RAID systems with our scheme can be carried out by merely using regular industrial RAID level 5 controllers, which are capable of performing the XOR calculations very well; for example, Promise's FastTrak S150 SX4-M controllers use an integrated XOR engine for parity calculations [12]. Therefore a lower cost controller could be applied in our RAID 6 algorithm in stead of a specific designed controller, which usually cost a lot, needed in other RAID 6 algorithms.

As to future works, the implementation of this XOR-based RS-RAID, from Eq.(1), could try to involve parallel processing in performing data coding to accelerate all of the computation in terms of the system performance as well. Besides, the algorithm used in this paper might be able to reduce more XOR operations by further mathematic approaches to accomplish the goal of the low execution time.

6. Acknowledgement

Our study grant is sponsored by Promise Technology Inc., and here we thank for their valuable comments and providing many suggestions to this work.

References:

- [1] Qin Xin; Miller, E.L.; Schwarz, T.; Long, D.D.E.; Brandt, S.A.; Litwin, W. "Reliability mechanisms for very large storage systems", Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings 20th IEEE/11th NASA Goddard Conference, 7-10 April 2003, pp.146-156.
- [2] 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.
- [3] 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.
- [4] Irving S. Reed, Xuemin Chen, "Error-Control Coding For Data Networks", Kluwer Academic Publishers, 1999
- [5] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding", IEEE Transactions on Information Theory, , Jan. 1999 , pp. 272-276
- [6] Telemetry Channel Coding, Recommendation for Space Data Systems Standards, CCSDS 101.0-B-3, Blue Book, Issue 3, May 1992.
- [7] J.S. Plank. "Correction to the 1997 Tutorial on Reed-Solomon Coding", Technical Report UT-CS-03-504, University of Tennessee, April, 2003.
- [8] J.S. Plank. "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Software – Practice & Experience", September 1997, 27(9):995–1012.
- [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] Lihao Xu, "Highly Available Distributed Storage Systems", Ph.D. Dissertation, 1999.
- [12] Promise Technology Inc., "Whitepaper to FastTrak S150 SX4-M controller ", April 2004.