6

Design of Simple and High Speed Scheme to Protect Mass Storages

Ming-Haw Jing, Yan-Haw Chen, Zih-Heng Chen, Jian-Hong Chen and Yaotsu Chang
I-Shou University
Taiwan, R.O.C.

1. Introduction

The computer industry has entered a stage of unprecedented improvement in CPU performance. However, the speed of file system management of huge information is commonly considered as the main factor that affects the computer performance; for example, the I/O bandwidth is limited by magnetic disks. The capacity and cost of magnetic disks per megabyte have been continually improved, but the rotation speed and seek time are improved very slowly. Recently, many computers have become I/O bound in the applications of video, audio, commercial database, etc. If such an I/O crisis can be resolved, the computer system performance will be improved. In 1988, Patterson et al. proposed the redundant array of independent disks (RAID) system which allows the data to be separated into several disks (Patterson et al., 1988). We can access the data in parallel so that the throughput of I/O systems will be improved. On the other hand, more disks in RAID system have a higher risk of losing data because of high component failure rates. As a result, the safety and reliability have become the major issues in the RAID system.

When designing a highly available and reliable RAID system, the method of bit wise parity checking is mostly used to correct errors and to enhance reliability of the RAID system. However, the parity checking method is limited so that only single disk failure can be tolerated. In 1995, Blaum et al. proposed a method called even-odd code, which tolerates up to two disk failures in the RAID system (Blaum et al., 1995). Even-odd code is the first known scheme for tolerating single or double disk failures, providing an optimal solution with regard to both storage and performance. However, the major problem concerning the even-odd code is a variety of modes of operations when solving erasures or up to 2 disk failures. In practical, it is not easy to be integrated into a VSLI. On the other hand, a small write problem is difficult to be solved with the even-odd code (Liao & Jing, 2002).

In 1997, Plank proposed a tutorial by using the Reed-Solomon (RS) code to provide error correction in the RAID system (Plank, 1997). In 2000, Jing et al. also proposed a simple algorithm, called RS-RAID system, to combine the RS codes with the RAID system (Jing et al. 2000). In this chapter, we aim to improve RS codes codec to design a fast error and erasure correction for RS-RAID system, and to solve the small write problem in RS codes. In a RS decoder, there are various algorithms to solve the error locator polynomial, which affect the

complexity and performance in hardware design such as the Berlekam-Massey algorithm, Euclidean algorithm, and Peterson-Gorenstein-Zierler (PGZ) algorithm (Wicker, 1995). The PGZ algorithm is mostly applied to correct less than six or seven errors because it is very simple with enhanced error tolerant capability. This chapter contributes to simplify the PGZ algorithm for single error or double erasure disks correction in the RS-RAID system by using the support of a set of combinational circuits. Its error and erasure correction become faster without the use of Chien search to find the root of the error locator polynomial, as well as the Forney method to find the error magnitudes from evaluator polynomial. Hence, this straightforward algorithm is then suitable for VLSI design.
This chapter is organized as follows: The history of RAID system is presented in Section 2. The simple RS encoder and decoder are described in Section 3. The design of small write modules of RS-RAID system is shown in Section 4. The RS-RAID system is proposed in Section 5. Finally, the result and conclusion are given in Section 6.

2. The Redundant Array of Independent Disks

A landmark paper, titled "A case for redundant arrays of inexpensive disks (RAID)", was presented by Patterson, Katz and Gibson in 1988 (Patterson et al., 1988). RAID systems can be configured into various ways to get a compromised result on data access speed, system reliability and size of storage. The general trade-off is to increase data access speed by writing the data into more places, which increases the amount of storage available by a factor *N*. On the other hand, more disks cause lower reliability on the disk system, leading to data redundancy and the need for additional algorithms to enhance the reliability of valuable data. There are several levels in the RAID system, such as RAID-0, RAID-1, RAID-10, RAID-2, RAID-3, RAID-4, RAID-5, RAID-6, and RAID-7. The mostly used versions for the trade-off are RAID-0, RAID-10, RAID-5, and RAID-6.

2.1 The RAID-0

RAID-0, as shown in Fig. 1, strips all data across multiple drives in a disk array. This is a high I/O performance solution, since it can simultaneously support many small/individual and large means of access with all disks transferring in parallel. Thus, very high data transfer rates (both reads and writes) may be achieved. RAID-0 is very suitable for I/O time critical or real time applications, such as video on demand (VoD) systems. However, RAID-0 maximizes the access speed and space available while being low in reliability. Because RAID-0 provides no data protection, the probability of disk failure increases with increasing number of disk drives. Any failing single drive will break the entire disk array.

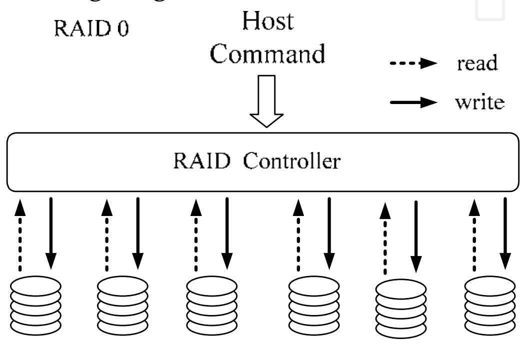

Fig. 1. The hierarchy of RAID-0

2.2 The RAID-1

RAID-1 writes data to two drives simultaneously in a replicated way, as shown in Fig. 2. If one drive fails, the data can still be retrieved from the counterpart of the RAID set. This process is also called "disk mirroring". Mirroring refers to the 100% duplication of data from one disk to another so that it is the most expensive RAID (double hardware storage required). It offers the advantage in reliability only. RAID-1 increases the reliability of protection of single disk failure, but doubles the cost for the available storage. RAID-1 is very suitable for both reliability and performance applications, such as OS disk and accountant data.

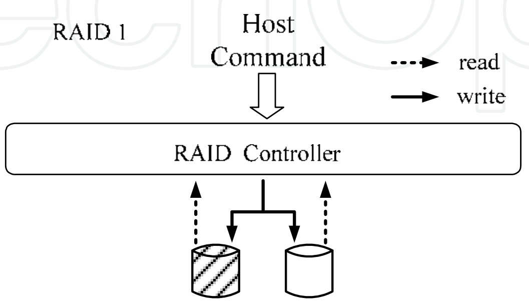

Fig. 2. The hierarchy of RAID-1

2.3 The RAID-10

RAID-10 is a combination of RAID-0 and RAID-1, where the data are striped and mirrored as shown in Fig. 3. This provides a higher speed and reliability, but possesses the same cost problem as RAID-1. Again, it can only tolerate single disk failure in each disk pair.

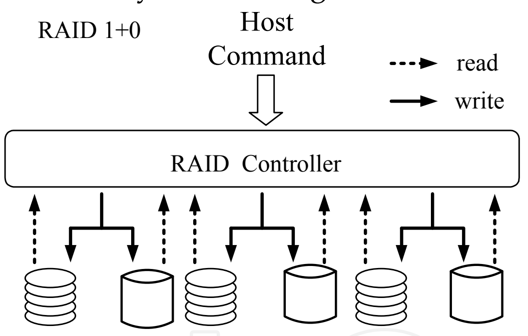

Fig. 3. The hierarchy of RAID-10

2.4 The RAID-5

RAID-5 uses a parity encoder to produce parity information and to provide data recovery at a low cost. The architecture of RAID-5 is shown in Fig. 4. Although one disk is added, data are striped over all disks so that large files can be fetched with high bandwidth. To balance the disk access load, a rotating parity is used. Many small random blocks can be accessed in parallel without hot spots/bottlenecks in any single disk. When a single disk fails, the data can still be reconstructed from the parity information.

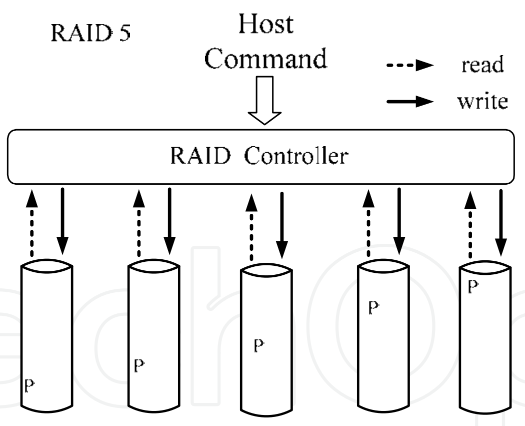

Fig. 4. The hierarchy of RAID-5

2.5 The RAID-6
Fig. 5 shows the model of RAID-6. RAID-6 is different from RAID-5 as it has two additional disks to recover the loss of two disks. This capability provides a higher fault tolerant capacity for disk array. The popular techniques for RAID-6 are even-odd and RS codes, which will be discussed later.

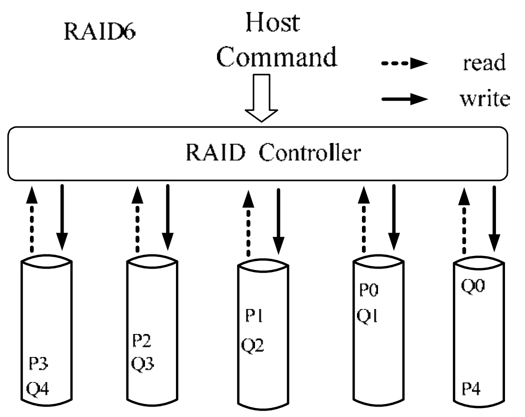

Fig. 5. The hierarchy of RAID-6

3. The Design of Reed-Solomon Codes for Error and Erasure Correction

The RS codes have been widely used in error control coding, especially in the applications of communication, satellite, and storage (Wicker, 1994). The RS codes have maximum distance separable (MDS) and hence can extend the largest possible minimum distance for codes of their size and dimension. In addition, RS codes are good at correcting burst errors. In the following, we discuss the basic definition of the encoder and illustrate those algorithms to correct one error or two erasure conditions using the PGZ algorithm (Wicker, 1995).

3.1 The Reed-Solomon Codes Encoder
The (n, k) RS codes over GF(2^m) have the capability to correct $2t = n - k$ erasures or t errors, where n is the total length of a codeword, k is the number of information symbols in GF(2^m), and t is the number of errors that the RS codes can correct. Consider the construction of a t-error-correcting RS code with a length of $2^m - 1$. $2t$ consecutive powers of α are required as

zeros of the generator polynomial $g(x)$ for the t-error-correcting RS code. The generator polynomial is the production of the associated minimal polynomials:

$$g(x)=\prod_{j=1}^{2t}(x+\alpha^j),\ \text{for}\ \alpha\in GF(2^m)\,. \tag{1}$$

We define each codeword as a polynomial $C(x)=c_0+c_1x+\ldots+c_{n-1}x^{n-1}$. Let a message polynomial can be expressed in terms of $I(x)=i_0+i_1x+\ldots+i_{k-1}x^{k-1}$. The $I(x)$ can be divided by a generator polynomial $g(x)$ to obtain the remainder polynomial $d(x)$, such that an encoded codeword is

$$C(x)=I(x)\cdot x^{2t}+d(x)\,. \tag{2}$$

3.2 The Reed-Solomon Codes Decoder

Since the discovery of RS codes, the efficient decoding algorithm has been highly used for the high-speed data process. Peterson provided the first decoding algorithm for binary BCH codes (Peterson, 1960). Then Peterson's algorithm was improved and extended to non-binary codes by Gorenstein and Zierler (Gorenstein & Zierler, 1961), Chien (Chien, 1964), and Forney (Forney, 1965). In the following, we discuss how to solve two erasures or one error using the PGZ algorithm, and propose a fast error and erasure correction algorithm based on the PGZ algorithm. From the example, further design for tolerating more errors or erasures may be developed when needed.

3.2.1 The Syndrome Evaluation Algorithm

The syndrome evaluation is the first procedure in the error detection. Assume that a received codeword polynomial $R(x)=r_0+r_1x+\cdots+r_{n-1}x^{n-1}$ can be expressed in terms of the sum of the codeword polynomial $C(x)$ and the error polynomial $e(x)=e_0+e_1x+\cdots+e_nx^n$, denoted as

$$R(x)=C(x)+e(x)\,, \tag{3}$$

and the syndromes are

$$S_k(x)=R(\alpha^k)=C(\alpha^k)+e(\alpha^k)\ \text{for}\ 1\le k\le 2t\,. \tag{4}$$

The α^k is the root of $C(x)$, so $C(\alpha^k)=0$; consequently, the syndrome is actually a form of evaluation for the error pattern $e(x)$. If there is an error $e(x)$, the syndrome will be nonzero.

3.2.2 The Design of Single Random Error Correction

In the applications of high speed storage, the requirement for single random error or double erasure correction is commonly applied. The PGZ algorithm is applied to find the error locator polynomial $\sigma(x)=1+\sigma_1x+\cdots+\sigma_tx^t$. We represent the syndromes with t errors in the received word as follows:

$$S_k=\sum_{i=1}^{t}Y_iX_i^k,\ \text{for}\ 1\le k\le 2t\,, \tag{5}$$

where X_i is the error locator for the ith error and Y_i is the magnitude of the ith error. If a random error e_i has been introduced into the received word as $r_i=c_i+e_i$, the syndromes from equation (4) can be represented as

$$S_1 = R(\alpha) = e_i \cdot \alpha^i \tag{6}$$

and

$$S_2 = R(\alpha^2) = e_i \cdot \alpha^{2i} . \tag{7}$$

From equations (6) and (7), the e_i can directly affect the syndromes S_1 and S_2. With single error, the error magnitude e_i is substituted by Y_i and the error location α^i is substituted by X_i. Rearranging the equations (6) and (7), we have

$$S_1 = Y_i \cdot X_i \tag{8}$$

and

$$S_2 = Y_i \cdot X_i^2 . \tag{9}$$

Finally, the direct solution of the error location X_i and magnitude Y_i are obtained from equations (8) and (9) as

$$X_i = \frac{S_2}{S_1} = S_2 \cdot S_1^{-1} \tag{10}$$

and

$$Y_i = \frac{S_1^2}{S_2} = S_1^2 \cdot S_2^{-1} . \tag{11}$$

3.2.3 The Design of Single or Double Erasure Correction

In the definition, the erasure means that the error location has been known. Starting from the case of double erasures, we assume those magnitudes as Y_i and Y_j, on the ith and jth locations in the codeword, respectively, and the effected syndromes are

$$S_1 = Y_i \cdot \alpha^i + Y_j \cdot \alpha^j \tag{12}$$

and

$$S_2 = Y_i \cdot \alpha^{2i} + Y_j \cdot \alpha^{2j} . \tag{13}$$

By solving equations (12) and (13), the magnitudes can be obtained from the syndromes as

$$Y_i = \frac{S_1\alpha^j + S_2}{\alpha^{2i} + (\alpha^{i+j})} \tag{14}$$

and

$$Y_j = \frac{S_1\alpha^i + S_2}{\alpha^{2j} + (\alpha^{i+j})} . \tag{15}$$

In the event of one erasure error, $Y_i \neq 0$ and $Y_j = 0$; the syndromes become $S_1 = Y_i \cdot \alpha^i$ and $S_2 = Y_i \cdot \alpha^{2i}$. We obtain the location or Y_i as

$$Y_i = \frac{S_1 + S_2}{\alpha^{2i} + \alpha^i} . \tag{16}$$

For this event, we can also use the same equation of solving double erasure errors by setting $Y_j = 0$ and $j = 0$, such that $\alpha^j = 1$; therefore, equation (16) can be substituted by equation (14). Thus, only one set of erasure module is needed.

4. The Design of Reed-Solomon Codes with Small Write Capability

In the design of highly reliable systems for banks, stock markets and hospitals, RAID-6 systems are normally applied. A problem that will affect the system is the frequent transactions or updating of data/information. Those frequent writing is a small amount of bytes compared with a block of record in kilo-bytes. This is called the small write problem which is an important factor that influences the performance of a RAID system. The small write problem is caused by the frequent modifications of partial codewords, and the parity bytes of each codeword also need to be updated. In other words, to update the parities, block data reading is needed for the task of partial data updating in RAID systems.
As shown in Fig. 6, in the RAID-5, assume the D0 is modified, so that the parity should be updated, too. For example, an inefficient method fetches the unused/whole data and encodes them again, causing a great delay and a serious problem on writing a small amount of data. Another smart algorithm is as follows: first, read the old D0 and old parity P and then perform exclusive-OR operation with the new data D0′ to obtain the new party P′. Second, write the new data D0′ and new parity P′ back. In this proposed algorithm, we need 2 reads and 2 writes operation to update D0.

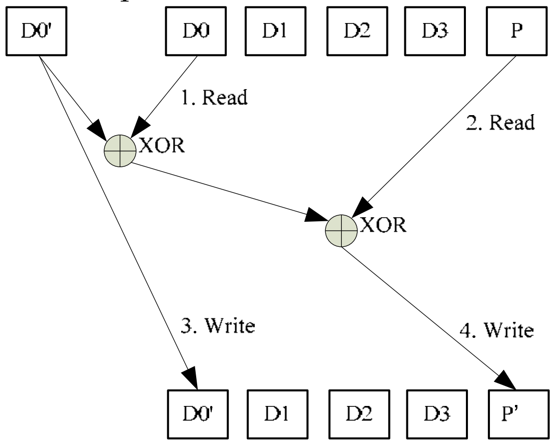

Fig. 6. The data updating for single parity

On the other hand, the RAID-6 systems have 2 parities, and the modification of parities might need to fetch a block of data to calculate the new parity. This is another inefficient method that fetches more data and performs encoding again. The proposed algorithm has limited times of access so that only the changed data can be read and the calculation of the parities is performed, such as the c_0' and c_1', which may be different from the original c_0 and c_1 in the encoder. Since the advance of VLSI, the proposed IP has absolutely become a combinational circuit to perform the calculation which will provide a high speed performance in the case of low delay and a simple interface to the current RAID systems, as presented in the following sections.

4.1 The Algorithm of Small Write Encoder

Regarding RS-RAID, according to the design of RS codes in Section 3.1, if a symbol/data in a set of codeword $C(x)$ has been modified, the original parity symbols c_0 and c_1 have to be re-encoded. Assuming the new symbol $c_i' \in C(x)$, where $2 \le i < k$, is an updated parity in the codeword $C(x)$. Take c_i' as the coefficient of the new input, the encoder should generate the

correspondence parity symbols c_0' and c_1' to construct a new set of codeword from the original parity symbols c_0 and c_1. The fast algorithm is explained as follows.
First, the codeword $C(x)$ can be expressed in terms of

$$C(\alpha) = \sum_{i=0}^{n-1} c_i \alpha^i = 0 \tag{17}$$

and

$$C(\alpha^2) = \sum_{i=0}^{n-1} c_i \alpha^{2i} = 0 \,. \tag{18}$$

We also know the c_0 and c_1 are parity check symbols, thus the equations (17) and (18) can be expressed in terms of

$$c_0\alpha^0 + c_1\alpha^1 = c_j\alpha^j + \sum_{i=2,i\neq j}^{n-1} c_i\alpha^i \tag{19}$$

and

$$c_0\alpha^0 + c_1\alpha^2 = c_j\alpha^{2j} + \sum_{i=2,i\neq j}^{n-1} c_i\alpha^{2i} \,. \tag{20}$$

where j is the index or the location, c_j is the original coefficient and c_j' is the new coefficient of the updated codeword.
Secondly, the new coefficients c_0' and c_1' can be expressed in terms of the equations $c_0' = c_0 + \Delta_0$ and $c_1' = c_1 + \Delta_1$. In a similar manner, $c_j' = c_j + \Delta_j$, where Δ_j stands for the differences between the original and new coefficients of the jth symbol in the codeword. Since Δ_j is known, we need to solve Δ_0 and Δ_1 so that we can substitute Δ_0, Δ_1 and Δ_j into equations (19) and (20); therefore, we have

$$(c_0+\Delta_0)\alpha^0 + (c_1+\Delta_1)\alpha^1 = (c_j+\Delta_j)\alpha^j + \sum_{i=2.i\neq j}^{n-1} c_i\alpha^i \tag{21}$$

and

$$(c_0+\Delta_0)\alpha^0 + (c_1+\Delta_1)\alpha^2 = (c_j+\Delta_j)\alpha^{2j} + \sum_{i=2.i\neq j}^{n-1} c_i\alpha^{2i} \,. \tag{22}$$

To solve the Δ_0 and Δ_1, subtract equation (19) from equation (21) and use the same way on equations (20) and (22); we obtain

$$\Delta_0\alpha^0 + \Delta_1\alpha^1 = \Delta_j\alpha^j \tag{23}$$

and

$$\Delta_0\alpha^0 + \Delta_1\alpha^2 = \Delta_j\alpha^{2j} \,. \tag{24}$$

From the equations (23) and (24), it is found that we do not need the whole codeword to generate the new set of parity symbols. This is the key to calculate the new parity symbols on line. To solve Δ_0 and Δ_1, the set of equations in equations (23) and (24) can be solved simultaneously, and we have

$$\Delta_1 = (c_j' - c_j)\frac{(\alpha^j + \alpha^{2j})}{(\alpha + \alpha^2)} \,. \tag{25}$$

Finally, the new parity check coefficient c_1' can be expressed in terms of

$$c_1' = (c_j' - c_j)\frac{(\alpha^j + \alpha^{2j})}{(\alpha + \alpha^2)} + c_1 \,. \tag{26}$$

Extending this representation to the new parity check coefficient c_0', we obtain

$$c_0' = (c_j' - c_j)\alpha^j + (c_j' - c_j)\frac{(\alpha^j + \alpha^{2j})\alpha}{(\alpha + \alpha^2)} + c_0 \,. \tag{27}$$

This proves the decoder without a sequential stream of data. If all the elements are over GF(2^8), equations (26) and (27) can be rearranged to obtain

$$c_1' = (c_j' - c_j)(\alpha^j + \alpha^{2j})\alpha^{229} + c_1 \tag{28}$$

and

$$c_0' = (c_j' - c_j)(\alpha^j + (\alpha^j + \alpha^{2j})\alpha^{230}) + c_0 \,. \tag{29}$$

By observing equations (28) and (29), we have a combinational circuit to construct a VLSI module to finally realize this function.

5. The RS-RAID System Design

Based on the explanations in Sections 3 and 4, the basic modules of RS codes are included to develop a reliable disk system, or RS-RAID system. The RS-RAID system design not just tolerates up to two or more disks failure but also corrects error(s) and erasures transparently. Transparency features high speed and real-time processes without complicated software control. This section will discuss the design of this RS-RAID system from its operation to system architecture.

5.1 System Design

In regard to modern mass storage systems, there are usually ten or more disks in the RAID system with less reliability or a higher risk of data loss. A reliable storage system must satisfy the following requirements:

1. High-performance disk failure recovery: It not just features high-speed access but also tolerates up to two or more disks failure.
2. Low recovery time: When one or more disks are not returning data within a limited period of time, the system control assumes that the disk(s) is/are slow disk(s) and solves its original information from the existing data/codeword. This strategy must be realized with low access or recovery time and speed up the system performance.
3. High confidence on individual disk: The disk can identify whether its data are reliable or not.

Aiming to meet the above requirements, we first use CRC 32 as part of the major checking data to judge the health of data disks. The rate of miss checking using CRC 32 is comparatively low. Secondly, a Reed-Solomon Product Code (RS-PC) is proposed with the support of CRC 32 checking bits to construct a highly reliable RS-RAID. The RS-PC is a combination of two (n, k) RS codes, denoted by inner-codes C_{row} and outer-codes C_{col}. The C_{row} and C_{col} codes are presented as the parity symbols of line blocks in row and column directions, as shown in Fig. 7. The codes C_{row} and C_{col} are combined for double protection, which is a "check-on-check" of the RS-PC to enhance the error-correcting capability. Since the two parity symbols are utilized as the line blocks in rows and columns, the architecture of RS-RAID is then partitioned into dual levels, namely the system level and disk level, as shown in Fig. 8.

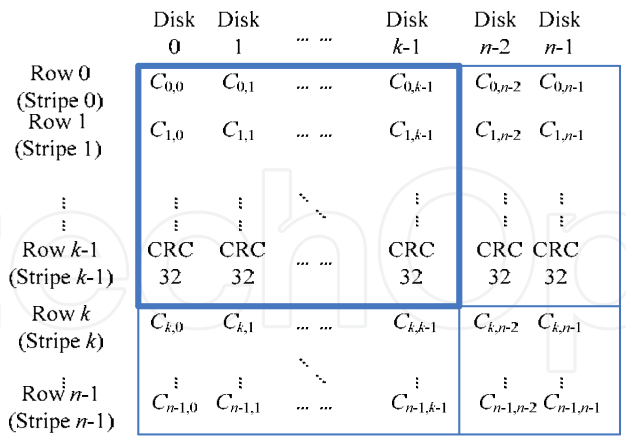
Disk 0
Disk 1
Disk k-1
Disk n-2
Disk n-1
Row 0 (Stripe 0)
Row 1 (Stripe 1)
Row k-1 (Stripe k-1)
Row k (Stripe k)
Row n-1 (Stripe n-1)
$C_{0,0}$ $C_{0,1}$ $C_{0,k-1}$ $C_{0,n-2}$ $C_{0,n-1}$
$C_{1,0}$ $C_{1,1}$ $C_{1,k-1}$ $C_{1,n-2}$ $C_{1,n-1}$
CRC 32
$C_{k,0}$ $C_{k,1}$ $C_{k,k-1}$ $C_{k,n-2}$ $C_{k,n-1}$
$C_{n-1,0}$ $C_{n-1,1}$ $C_{n-1,k-1}$ $C_{n-1,n-2}$ $C_{n-1,n-1}$

Fig. 7. The format of Reed-Solomon product code

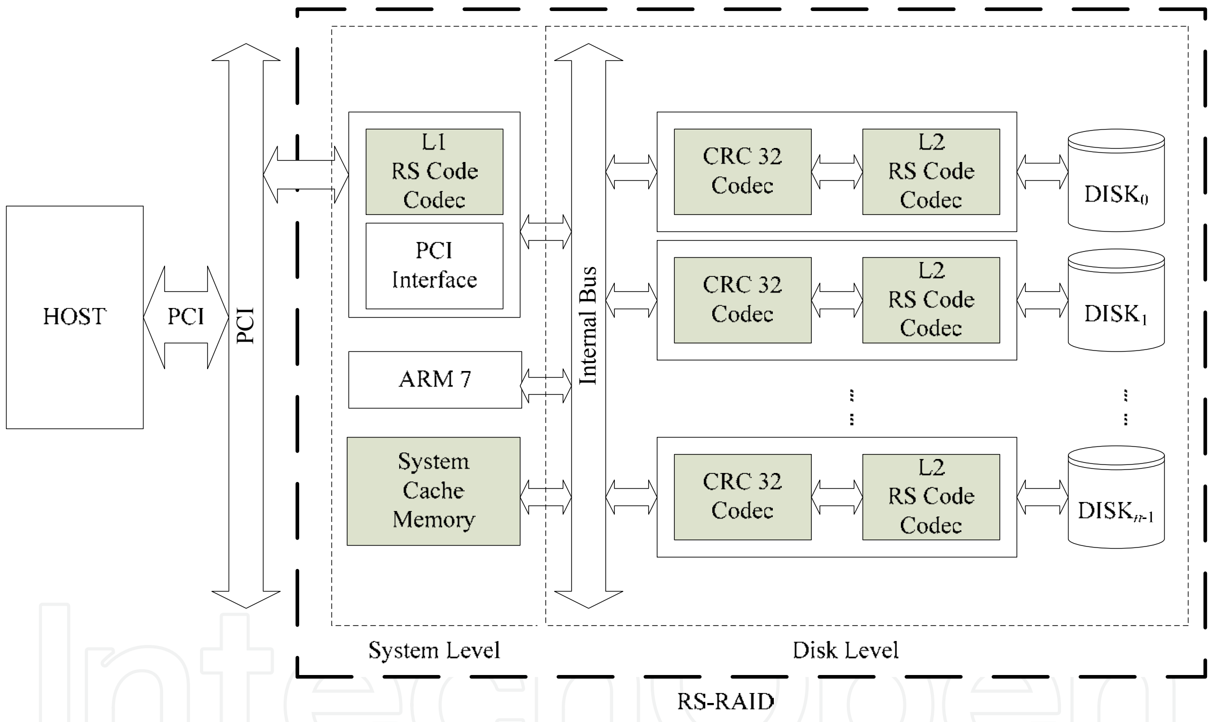

Fig. 8. The architecture of RS-RAID

In this design, all disks are considered as large logical/unified storage. The host can access the data in RS-RAID through the IDE or SCSI interface. At the system level, there are n disks, and all data are encoded and decoded by L1 (level one) RS-code codec through the PC interface. This design guarantees the reliability of data reading from the large logical disk. System Cache memory, which temporarily stores the data encoded from L1 RS-code codec, is used to buffer the currently used data. Cache buffer will improve the RS-RAID performance in frequent access to/from the system.

At the disk level, each disk has n stripes space. When the data are read from a disk, they are decoded by L2 (level two) RS-code codec and then checked by CRC 32 codec. If the amount of errors is too many for the correcting capability of L2 RS-code codec, this situation will be detected by CRC 32 codec. This guarantees the data reading from individual disk in a reliable condition. This system has advantages such as high capacity, throughput and reliability, because all encoding/decoding processes are operating in real time.

5.2 System Operation

For easier explanations, the operation of the system is based on the concept of the error correction design. The operations in dual levels can enhance the system reliability and increase the number of errors tolerated in the system.

5.2.1 The System Level

At the system level, the L1 RS-code codec can be partitioned into the encoder and the decoder parts, as shown in Fig. 9. When bulk write from the host is performed, the information of stripe *u* can be expressed in terms of $I_u(x) = \{i_{k-1}, i_{k-2}, \ldots, i_0\}$ and encoded into a inner-code $C_{cow_u}(x) = \{c_{u,n-1}, c_{u,n-2}, \ldots, c_{u,0}\}$ for each tripe of data, where $0 \le u < n$, n is the number of total disks in system, and *k* is the number of data disks. When the frequently rewritten data are sent to the system cache sequentially from the host, the c'_0 and c'_1 become new parity symbols and must be updated in real time, as shown in Fig. 9.

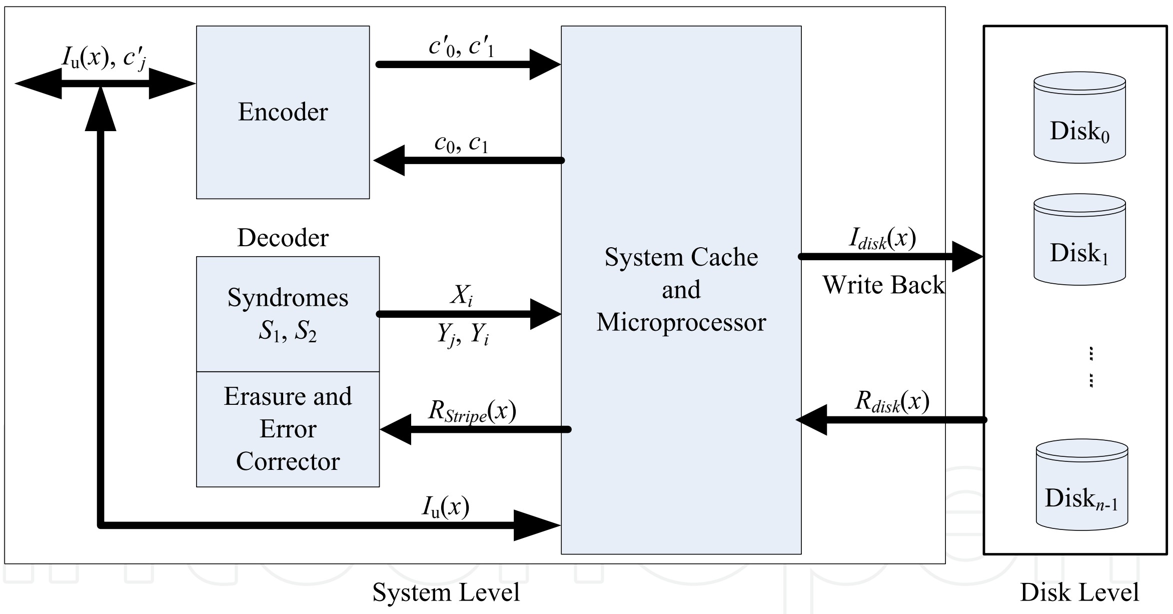

Fig. 9. The RS codes codec block diagram of system level

Before being written back to disks, all data are temporarily stored into system cache. During write back, the data are firstly encoded into the outer-code $C_{col_v}(x) = \{c_{n-1,v}, c_{n-2,v}, \ldots, c_{0,v}\}$ for each disk, where $0 \le v < n$, and *n* is the stripes of a disk.

When read from an unreliable disk or communication channel, the received data from disk $R_{disk}(x) = \{R_{n-1}, R_{n-2}, \ldots, R_0\}$ are decoded into $I_{disk}(x) = \{I_{n-1}, I_{n-2}, \ldots, I_0\}$ and stored as inner-code C_{row} in system cache. When read from cache, the inner-code C_{row} can be represented in terms of stripe $R_{stripe_u}(x)=\{r_{u,n-1},r_{u,n-2},\ldots,r_{u,0}\}$, where $0 \le u < n$, and n is the total number of disks. Decoding $R_{stripe}(x)$, from the previous research in (Jing et al. 2001), the procedure is as follows:

A. $R_{stripe_u}(x)$ is firstly sent to the error corrector and generates its syndromes S_1 and S_2 for error checking and correction purposes.

B. When a random error occurs, the corrector will use the syndromes to calculate its magnitude Y_i on position X_i.

C. With erasure(s), the corrector firstly sets one or both of X_i and X_j as the already known position(s) of erasure(s) to solve their correlated error magnitudes Y_i and Y_j.

When an error or erasure(s) is found, the corrector will correct the $R_{stripe_u}(x)$ using S_1 and S_2 immediately after completion of reading. The timing of this procedure is shown in Fig. 10. With such high-speed correction, we consider this operation as a real-time correction process.

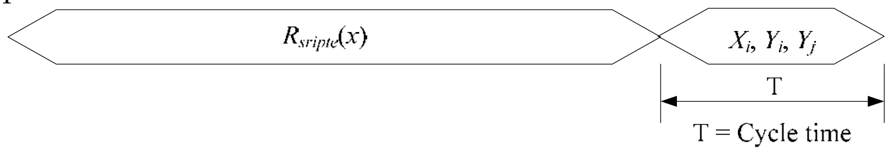

Fig. 10. The RS codes codec block diagram of system level

5.2.2 The Disk Level

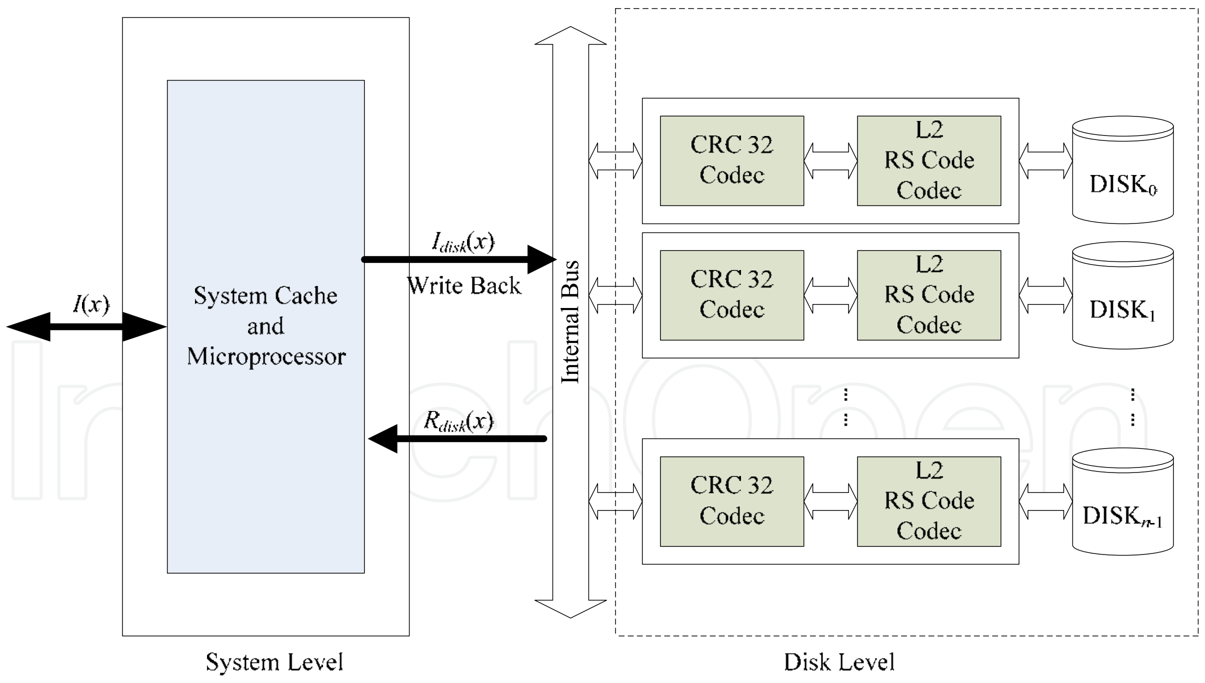

Fig. 11. The RS codes codec block diagram of system level

At the disk level as shown in Fig. 11, when the bulks are written from cache into disks, the encoding procedure at the disk level is as follows:

1. The data $I_{disk}(x)=\{I_{n-1},I_{n-2},\ldots,I_0\}$ in cache is written into disks.

2. Each disk receives its own data *I* which is then encoded by CRC 32 codec.
3. The encoded data stream from CRC 32 codec are encoded again by L2 RS-codes codec and stored in the disks.
Thus, the RS-PC encoding is finished.
When a system reads data from disks into cache, all data will be checked by CRC 32 codec and decoded by L2 RS-codes codec. The decoding procedure at the disk level is as follows:
1. The data in each disk are decoded and corrected by its own L2 RS-codes codec.
2. The decoded data stream from each L2 RS-codes codec are decoded again by its own CRC 32 codec.
Finally, if the quantity of errors is greater than the error-correcting capacity of L2 RS-codes codec, the CRC 32 codec can detect the errors and report to the system level, so this disk is marked as an erasure at the system level. This strategy will enhance the system reliability and increase the data access speed with less possibility to retry failure disk(s). This design provides support of double protection for the RAID system in real time.

6. Conclusion

This paper provides an example of coding to implement a RS code in redundant array of independent disks system in correcting single random error and double erasures. There are new directions such as the small write module and higher correction capabilities for the design of a RS-RAID system. As a result, the proposed RS-RAID system has the following advantages:
1. Expandable design: As the design of RAID-6, this paper does not only propose a solution for dual disks failure, but also adopt the PGZ algorithm to correct less than six or seven errors.
2. Integrated concept: This system presents a unified RSPC concept to partite the system into dual levels of abstract/structure. Thus, the modules at the disk level mainly deal with burst or random errors in disks, and the control of the system level does the correction for multiple failures of the system. On the other hand, each disk may be a surface of disk so that a fault tolerant hard disk is produced.
3. Real-time updating capability: In regard to the applications for banks, stock markets, hospitals or military purposes, the system requires frequent transactions or updating of data/information. The small write module may support the system cache with a real time requirement and solve the frequent update operations in the RAID system with very low overhead.
4. Suitability for co-design: The proposed algorithm is suitable for both hardware and software designs of the modules in the applications of RS codes by using the finite field. The math of finite field belongs to modern algebra which has been largely applied to the applications of error correction code and cryptography. This suggests that the hardware modules will be integrated into the math processor in CPU of the future versions. The reliable control may be easily integrated into microcontrollers and general processors.
5. More applications: With the advantages from the expandability to the co-design of the system, this concept may extend its applications to most memory systems such as the flash memory, DRAM, and so on.

7. Acknowledgments

This work is supported in part by the Nation Science Council, Taiwan, under grant NSC 91-2213-E-214-010.

8. References

Blaum, M.; Brady, J. ; Bruck, J. & Menon, J. (1995). EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures. *IEEE Transaction on Computers,* Vol. 44, No. 2, pp. 192-202, ISSN: 0018-9340

Chien, R.T. (1964). Cyclic Decoding Procedure for the Bose-Chaudhuri- Hocquenghem Codes. *IEEE Transaction on Information Theory,* Vol. IT-10, No. 10, pp. 357-363, ISSN: 0018-9448

Forney, G.D. (1965). On Decoding BCH Codes. *IEEE Transaction on Information Theory,* Vol. IT-11, No. 4, pp. 549-557, ISSN: 0018-9448

Gorenstein, D. & Zierler, N. (1961). A Class of Error Correcting Codes in p^m Symbols. *Journal of the Society of Industrial and Applied Mathematics,* Vol. 9, No. 2, pp. 207-214, ISSN: 0368-4245

Jing, M.H.; Chen, Y.H. & Yuan, K.Y. (2000). The Comparison of Evenodd Code and RS Code for RAID Applications, *Asia Pacific Conference on Multimedia Technology and Application*, pp. 261-268, December, 2000, Kaohsiung, Taiwan

Jing, M.H.; Chen, Y.H. & Liao, J.E. (2001). A Fast Error and Erasure Correction Algorithm for a Simple RS-RAID, *Proceedings of the 2001 International Conferences on Info-tech and Info-net*, pp. 333-338, ISBN: 0-7803-7010-4, October, 2001, Beijing, China

Liao, J.E & Jing, M.H. (2002). *The Research and Implementation of High-Speed RAID Using Reed-Solomon Codes,* Master's thesis, Department of Information Engineer, I-Shou University, Kaohsiung, Taiwan

Patterson, D.A.; Gibson, G. & Katz, R.H. (1988). A Case for Redundant Arrays of Inexpensive Disks (RAID), *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pp. 109-116, ISBN: 0-89791-268-3, June, 1988, ACM, Chicago, Illinois

Peterson, W.W. (1960). Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes. *IRE Transaction on Information Theory,* Vol. IT-6, No. 4, pp. 459-470, ISSN: 0096-1000

Plank, J.S. (1997). A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. *Software - Practice and Experience,* Vol. 27, No. 9, pp. 995-1012, ISSN: 0038-0644

Wicker, S.B. & Bhargava, V.K. (1994). *Reed-Solomon Codes and Their Applications,* IEEE Press, ISBN: 0-7803-5391-9, NJ

Wicker, S.B. (1995). *Error Control Systems for Digital Communication and Storage*, Prentice Hall, ISBN: 0-13-308941-X, US

Data Storage
Edited by Florin Balasa

ISBN 978-953-307-063-6
Hard cover, 226 pages
Publisher InTech
Published online 01, April, 2010
Published in print edition April, 2010

The book presents several advances in different research areas related to data storage, from the design of a hierarchical memory subsystem in embedded signal processing systems for data-intensive applications, through data representation in flash memories, data recording and retrieval in conventional optical data storage systems and the more recent holographic systems, to applications in medicine requiring massive image databases.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Ming-Haw Jing, Yan-Haw Chen, Zih-Heng Chen, Jian-Hong Chen and Yaotsu Chang (2010). Design of Simple and High Speed Scheme to Protect Mass Storages, Data Storage, Florin Balasa (Ed.), ISBN: 978-953-307-063-6, InTech, Available from: http://www.intechopen.com/books/data-storage/design-of-simple-and-high-speed-scheme-to-protect-mass-storages