TABLE III
COMPARISON—HARDWARE COST FOR CHECK SYMBOL GENERATORS

Info bits	Berger codes		LC codes	MB codes			
I	FA	HA	FA	FA	HA	XOR[a]	Savings
15	11	0	—	7	0	3	22.7%
16	11	4	12	7	2	3	25.6%
31	26	0	—	15	0	7	28.8%
32	26	5	28	15	2	7	30.7%
63	57	0	—	31	0	15	32.5%
64	57	6	60	31	2	15	33.6%

[a]Three-input XOR gate.

codes in most cases. Also, MB codes can be easily applied to any number of information bits. Because the number of check bits of an MB code is independent of the total number of the information bits, they are suitable for circuits that have a large number of outputs, such as PLA's. It is also shown in the paper that the totally self-checking checkers for MB codes are less expensive and have less time delay than that for either Berger codes or low-cost residue codes. All these advantages make MB codes very attractive for practical applications.

ACKNOWLEDGMENT

The author wishes to thank Prof. E. J. McCluskey and Prof. J. F. Wakerly for their valuable comments, and L. Christopher and J. McCluskey for their help in formatting this manuscript.

REFERENCES

[1] M. J. Ashjaee and S. M. Reddy, "On totally self-checking checkers for separable codes," *IEEE Trans. Comput.*, vol. C-26, pp. 737–744, Aug. 1977.
[2] A. Avizienis, "Arithmetic codes: Cost and effectiveness studies for application in digital system design," *IEEE Trans. Comput.*, vol. C-20, pp. 1322–1331, Nov. 1971.
[3] J. M. Berger, "A note on error detection codes for asymmetric channels," *Informat. Contr.*, vol. 4, pp. 68–73, Mar. 1961.
[4] C. V. Freiman, "Optimal error detection codes for completely asymmetric binary channels," *Informat. Contr.*, vol. 5, pp. 64–71, Mar. 1962.
[5] M. A. Marouf and A. D. Friedman, "Design of self-checking checkers for Berger codes," in *Proc. 8th Annu. Symp. on Fault-Tolerant Computing*, Toulouse, France, June 21–23, 1978, pp. 179–184.
[6] J. F. Wakerly, "Detection of unidirectional multiple errors using low-cost arithmetic codes," *IEEE Trans. Comput.*, vol. C-24, pp. 210–212, Feb. 1975.
[7] ——, *Error Detecting Codes, Self-Checking Circuits and Applications.* New York: Elsevier North-Holland, 1978, pp. 40–50.

Unidirectional Error Codes for Shift-Register Memories

BELLA BOSE AND T. R. N. RAO

Abstract—**In this correspondence we give an efficient error control technique for mass memories such as magnetic bubble memories, magnetic tapes, etc. The code discussed here is a modification of the cyclic redundancy check (CRC) used in magnetic tape units. An arithmetic redundancy check (ARC) with respect to an appropriate modulus (or check base) replaces the CRC, and the result is a systematic code which provides correction of multiple unidirectional errors in any one track of a large block of characters. The information rate of this code is much higher than that of the CRC code.**

Index Terms—**Arithmetic residue check, asymmetric errors, information rate, parity check, shift register memories, symmetric errors, unidirectional errors.**

Manuscript received August 1, 1983; revised December 9, 1983. This work was supported by ONR Contract N00014-77-C-0455.

B. Bose is with the Department of Computer Science, Oregon State University, Corvallis, OR 97331.
T. R. N. Rao is with the Department of Computer Science, University of Southwest Louisiana, Lafayette, LA 70504.

NOTATIONS AND SYMBOLS

ℓ	Length of the data block including ARC character.
b	Number of tracks including parity track.
$i,j \in [0,m]$	Implies that the integers i and j are in the range 0–m, both 0 and m included.
$\lvert B\rvert_A$	The least nonnegative integer congruent to B modulo A.

I. INTRODUCTION

The development of new memory technologies is very important for a successful jump to the next generation of computer systems. The cost and performance of a computer system are dominated by the memory in its many forms—ferrite core, charge-coupled devices (CCD), magnetic bubble memory (MBM), semiconductor RAMS, drum, disk, and tape. Any major advance in memory systems is a tremendous boost to the overall performance of the computer.

The applications of error correcting codes to computer systems have become a technology of practical significance. These codes are extensively discussed in [1]. Most of the theory on random error-correcting/detecting codes has been developed under the assumption of symmetric errors in the data bits. The most likely errors in some recently developed LSI devices are of the unidirectional type (i.e., all bit failures are in the same direction) rather than symmetric type [2]–[6].

In a shift-register type of memory system, a permanent failure in one of the registers results in a constant 0 or 1 output from the shift register [3]. These unidirectional failure properties of some of the newly developed LSI memories provide the basis and motivation for a new direction of study in coding theory and fault-tolerant computing.

In this correspondence we give an efficient unidirectional error-control technique for mass memories such as CCD and MBM which are of the shift-register type, and magnetic tape units. Brown and Sellers [7] have given a similar type of error correction technique for magnetic tapes. They use vertical overall parity check and horizontal cyclic redundancy check to correct any pattern of symmetric errors in a single track. An extension to this coding scheme is given by Hong and Patel [8]. They show that the new scheme corrects any errors in two tracks if the erroneous tracks are known. More properties of these codes are described by Sloane [9]. Parhami and Avizienis [3] and Wakerly [4] consider detection of unidirectional errors in mass memories using arithmetic checks. They have not discussed correction of unidirectional errors.

The code discussed here is a modification of the cyclic redundancy check (CRC) used in magnetic tape units [7]–[9]. An arithmetic redundancy check (ARC) replaces CRC, and the result is a systematic code which provides correction of multiple unidirectional errors in any one track of a large block of characters. We call these codes "ARC codes." The improvement attained by using arithmetic check is that considerably larger number of characters can be included in each block and that the information rate is much higher than that of the CRC code.

II. ASYMMETRIC, UNIDIRECTIONAL, AND SYMMETRIC ERRORS

In the sequel, we will refer to the transition $0 \rightarrow 1$ as a 0-error, and to the transition $1 \rightarrow 0$ as a 1-error. First we make a clear distinction among asymmetric, unidirectional, and symmetric errors as follows.

Asymmetric errors are those in which all errors in the received words are of only one type (say 1-error) at all times.

Unidirectional errors are those in which all errors in a received word are 0-errors or 1-errors, but both types of errors do not appear simultaneously in any received word.

Symmetric errors are those in which both 0-errors and 1-errors can appear simultaneously in a received word.

III. Code Format and Capabilities

A. Memory Organization

We assume that data are transferred byte-serially between the mass memory and the main memory in blocks. Further, we assume that a data block consists of ℓ characters with each character b bits long.

The shift-register memory organization is shown in Fig. 1. There are b shift registers. Each bit of a character is in different shift register. Any permanent failure in a shift register results in a constant output of a 0 or 1 [3]. This can be modeled as multiple unidirectional errors in the bits of that particular shift register.

In the case of magnetic tapes each bit of a character is stored in a different track. Since there are b bits per character, we have b tracks. Again, we are interested in correcting multiple unidirectional errors confined to one particular track of the data block. The code discussed here is not intended for the detection or correction of errors in two or more tracks.

From the above discussion we can see that a data block is a two-dimensional array of ℓ columns and b rows. Borrowing from the terminology of magnetic tapes, we will refer to each column as one character and each row as one track. Also we will refer to the first $b - 1$ rows as information tracks and the last row as a parity track.

B. Data Format and Code Construction

The data format for this code is shown in Fig. 2. Let $B_j = (B_j(b-2), B_j(b-3)\cdots B_j(0))$, where $B_j(i) \in \{0, 1\}$. We give an integer value to each of B_j's as follows:

$$B_j = B_j(b-2)2^{b-2} + B_j(b-3)2^{b-3} + \cdots B_j(1)2^1 + B_j(0)2^0. \tag{1}$$

The last character, B_ℓ, is the arithmetic residue check (ARC), which is equal to the additive inverse of the residue modulo A of the sum of the information bytes, where A is an appropriate check base; i.e.,

$$B_\ell = -\left|\sum_{j=1}^{\ell-1} B_j\right|_A. \tag{2}$$

Then we have

$$\left.\begin{array}{l}\displaystyle\sum_{j=1}^{\ell} B_j \equiv 0(\text{mod } A)\\ \text{or}\\ \displaystyle\left|\sum_{j=1}^{\ell} B_j\right|_A = 0.\end{array}\right\} \tag{3}$$

The overall vertical parity bits are the even (or odd) parities over each of the characters.

C. Check Base Selection and Code Capabilities

In this section we discuss the condition on the check base A of ARC. While reading a data block assume that k (unidirectional) errors have occurred in one of the tracks. From the vertical parity failures we can find the vertical positions of the errors and the number of errors. Hence, if we determine the track in error, then we can correct all errors in that track. The ARC is designed to determine the erroneous track. For this we define the arithmetic residue syndrome or simply syndrome S as

$$S = \left|\sum_{j=1}^{\ell} B_j\right|_A. \tag{4}$$

If there is no error in any of the information tracks, then $S = 0$. If the k errors are in the vertical parity track, then also $S = 0$. On the other hand, if the k errors are in track i, where $0 \le i \le b-2$, and are of $0 \to 1$ type, then $S = |+k2^i|_A$; if the errors are of $1 \to 0$ type, then $S = |-k2^i|_A$.

From the above we can see that the check base A must satisfy the following two conditions.

1) If k is the number of unidirectional errors in track i, where $1 \le k \le \ell$ and $0 \le i \le b-2$, then the syndrome S should not be equal to zero; i.e.,

$$S \equiv \pm k2^i \not\equiv 0(\text{mod } A). \tag{5}$$

2) The syndromes for k unidirectional errors in track i and in track j are not equal, where $i \ne j$ and $1 \le k \le \ell$; i.e.,

$$\text{for } i \ne j,\ \pm k2^i \not\equiv \pm k2^j(\text{mod } A)\qquad i,j \in [0, b-2]. \tag{6}$$

Moreover, to obtain a high information rate, we look for as large a value of ℓ as possible.

In order to satisfy these conditions we will take as the value of the check base A the largest prime less than 2^{b-1}, where b is greater than 3. The largest prime less than 2^{b-1} is always greater than

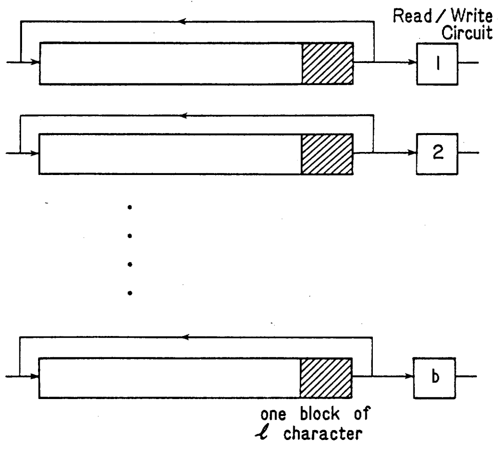

Fig. 1. Shift-register memory organization.

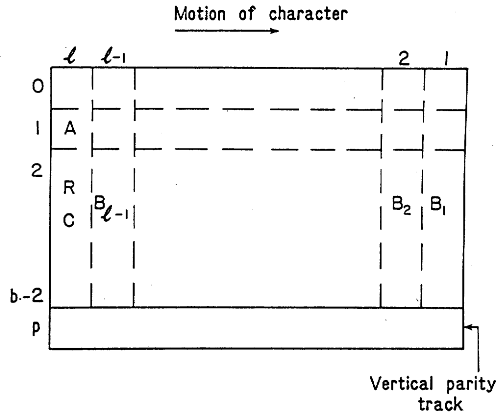

Fig. 2. Data format of a code block.

$2^{b-2} + 1$ for $b > 3$ [10]. The following theorem explains the above condition on A.

Theorem 1: Let A be the largest prime less than 2^{b-1} for $b > 3$. Then for $i, j \in [0, b-2]$ and $1 \leq k \leq A - 1$ the following conditions hold.

$$1)\ k2^i \not\equiv 0 (\text{mod } A) \tag{7}$$

$$2)\ \pm k2^i \not\equiv \pm k2^j (\text{mod } A) \quad \text{for } i \neq j$$

$$\text{i.e., } 2^h \not\equiv \pm 1 (\text{mod } A) \quad \text{for } 1 \leq h \leq b - 2. \tag{8}$$

Proof:

1) For some k and i, where $1 \leq k \leq A - 1$ and $0 \leq i \leq b - 2$, let $k2^i \equiv 0 (\text{mod } A)$. Then $A | k2^i$. Since the greatest common divisor (gcd) of $(A, 2) = 1$, this implies A divides k, denoted $A | k$. This is a contradiction because A is a prime and k is a positive integer less than A.

2) Two cases are to be considered for this condition.

Case 1: We have to prove $2^h \not\equiv 1 (\text{mod } A)$ for the range $1 \leq h \leq b - 2$. The integers $2^1, 2^2, \cdots, 2^{b-2}$ are all less than A and greater than 1. Hence, none of them is congruent to 1 modulo A.

Case 2: We have to prove $2^h \not\equiv -1 (\text{mod } A)$ for the range $1 \leq h \leq b - 2$. For some h, where $1 \leq h \leq b - 2$, let $2^h \equiv -1 (\text{mod } A)$. Then $A | 2^h + 1$. Since A is the largest prime less than 2^{b-1}, $A > 2^h + 1 > 0$ for $1 \leq h \leq b - 2$, and hence we get a contradiction.

As a consequence of the above theorem, in order to have the largest number of characters in the proposed code, the value of the check base A must be the largest prime less than 2^{b-1}. For the given number of tracks b, ($b \leq 17$) the appropriate values of the check base A are given in Table I. It is easy to see that the proposed code is not suitable when b is less than or equal to 3.

Suppose we choose an appropriate prime A satisfying the above mentioned conditions. Then ℓ, the number of characters in the code block, will be $A - 1$, and b, the number of tracks, will be $\lceil \log_2 A \rceil + 1$. The number of information and check bits will be, respectively, $(\ell - 1)(b - 1)$ and $\ell + b - 1$. Hence, the information rate of the code will be $1 - (\ell + b - 1/\ell b)$. Since $\ell \approx 2^{b-1}$ the information rate of the code is approximately $1 - (1/b)$. For the code given in [7]–[9], note that the information rate is $1 - (2/b)$.

For example, if we have a 9-track tape as in [11]–[13], we can choose the check base A to be 251. Then the coding scheme developed here can have up to 250 characters including the arithmetic residue check. The code can correct up to 250 unidirectional errors confined to single track. The number of information bits will be $249 \times 8 = 1992$ and the number of parity bits will be $250 + 8 = 258$. Hence, the information rate will be $[1 - (258/2250)] \times 100 = 88.5$ percent. On the other hand, the code given in [11]–[13] has 8 characters and the number of information bits is 56 and the number of check bits is 16. Hence, the information rate of the code is $(56/72) \times 100 = 77.8$ percent. However, this code can correct up to 8 symmetric errors in a single track. Hence, if the errors are of a unidirectional nature, then the code proposed here is much superior to the one discussed in [7]–[9].

Error correction can be performed as follows. Note that the first consecutive $b - 2$ nonnegative integer powers of 2 modulo A and their additive inverse are all distinct. For example, if $b = 9$ and $A = 251$, then we will have the following congruences modulo 251:

$$\begin{array}{lll} 2^0 \equiv 1 & 2^3 \equiv 8 & 2^6 \equiv 64 \\ -2^0 \equiv 250 & -2^3 \equiv 243 & -2^6 \equiv 187 \\ 2^1 \equiv 2 & 2^4 \equiv 16 & 2^7 \equiv 128 \\ -2^1 \equiv 249 & -2^4 \equiv 235 & -2^7 \equiv 123. \\ 2^2 \equiv 4 & 2^5 \equiv 32 & \\ -2^2 \equiv 247 & -2^5 \equiv 219 & \end{array}$$

Suppose s ($s \neq 0$) unidirectional errors occur in track j. The number s can be obtained by using an error counter, i.e., by incrementing the error counter whenever there is a failure in the overall vertical parity bits. After reading a word, let the residue be r. Now the track in error can be found by considering the following two cases. (For simplicity we assume that the unidirectional errors are of type 0-errors, i.e., $0 \rightarrow 1$ errors. The procedure is valid for 1-errors also.)

Case 1: The number of errors $s \neq 0$, and the residue $r \neq 0$. Then $s2^j \equiv r (\text{mod } A)$, which implies $2^j \equiv s^{-1} r (\text{mod } A)$ where s^{-1} is the multiplicative inverse of s in $GF(A)$. For $j = 0, 1, 2 \cdots b - 2$, the values of 2^j mod A are all distinct, and hence the above equation holds for a particular value of j. Hence, the track in error j can be identified.

Case 2: The number of errors $s \neq 0$, and the residue $r = 0$. In this case the errors are in the last track, i.e., the overall vertical parity check track.

An example is given to illustrate these concepts after describing the encoding and decoding logic. Once the track in error is known the data can be read again. At this time whenever there is a failure in the vertical parity bit the corresponding bit in the erroneous track can be complemented to get the correct data. (Alternatively, we can use a buffer register to store the data, and error correction can be carried out by reading the data from this buffer register.)

TABLE I
VALUES OF RESIDUE CHECK BASE FOR THE GIVEN NUMBER OF TRACKS

b	check base A
4	7
5	13
6	31
7	61
8	127
9	251
10	509
11	1021
12	2039
13	4073
14	8191
15	16381
16	32749
17	65521

D. Encoding and Decoding Logic

The encoding and decoding procedures are explained below.

Encoding: We can use a residue check register (RCR) for finding the residue check. Before writing data into the memory the RCR is cleared. As each character is written, the character (excluding its parity) is added (modulo-A addition) to the content of the RCR. After all characters are written, the complement of the RCR with respect to modulo-A addition (i.e., the modulo-A additive inverse) is written as an arithmetic residue check (ARC). A parity computed on ARC is then loaded as the parity bit of the ARC character.

Decoding: During a read operation, again the residue is calculated in RCR. (We may use a buffer register to store the data which are just read.) Also, whenever there is a failure in the overall parity check of a character, the error counter register (ECR), which is a modulo-A counter, is incremented. If there exists an erroneous track, then at the end of the read operation the ECR contains the number of errors in the erroneous track, otherwise, the ECR is zero. If both RCR and ECR are zero, then no error has occurred in the data block. If only RCR is zero and not ECR, then errors are in the vertical parity check track. The case where RCR $\neq 0$ and ECR $= 0$ cannot obviously occur. If both RCR and ECR are nonzero, then the erroneous track can be found as follows. The ECR is shifted left till the content of the ECR or its complement with respect to modulo-A addition is equal to the contents of the RCR. The number of left shifts j in ECR gives the position of the track in error. Once the track in error is known, error correction can be easily carried out. An example is given below to illustrate these concepts.

Example: Let the number of tracks including the parity track be equal to 9. Then the check base A will be 251. Suppose 50 $0 \rightarrow 1$ errors occurred in the third track. Then the contents of the ECR will be 50 and that of the RCR will be $|+50 \times 2^3|_{251} = |400|_{251} = 149$. The contents of ECR and its complement for the consecutive left shifts will be as follows:

Number of Shifts	ECR	251 − ECR
0	50	201
1	100	151
2	200	51
3	149	102

Since the contents of the ECR are equal to that of the RCR after 3 shifts, it is immediately known that the third track is in error. As another example, let there be 100 $1 \rightarrow 0$ errors in track 4. After reading the data block the contents of the ECR will be 100 and that of the RCR will be $|-100 \times 2^4|_{251} = |-1600|_{251} = 157$. After shifting the ECR 4 times, the contents will be 94 and the complement will be $251 - 94 = 157$. Hence, it can be determined that the fourth track is in error.

Note that the maximum number of left shifts required for the ECR to match RCR is at most $b - 2$. If there is no match between the ECR (or its complement) and the RCR even after $b - 2$ left shifts of ECR, then there exists an uncorrectable error pattern; i.e., more than one track may be in error.

IV. Conclusion

In this correspondence we have shown that the replacement of the cyclic redundancy check of the magnetic tape code by an appropriate arithmetic residue check will result in an efficient unidirectional error correcting code. The ARC code has much higher information rate than the CRC code, and the ARC code is simple to implement. Further properties of this code can be found in [11].

References

[1] W. W. Peterson and E. J. Weldon, *Error Correcting Codes*. Cambridge, MA: MIT Press, 1972.
[2] R. W. Cook, L. H. Sisson, T. F. Storey, and W. N. Toy, "Design of a self-checking microprogram control," *IEEE Trans. Comput.*, vol. C-22, pp. 255–262, Mar. 1973.
[3] B. Parhami and A. Avizienis, "Detection of storage errors in mass memories using low-cost arithmetic error codes," *IEEE Trans. Comput.*, vol. C-27, pp. 302–308, Apr. 1978.
[4] J. F. Wakerly, "Detection of unidirectional multiple errors using low-cost arithmetic codes," *IEEE Trans. Comput.*, vol. C-24, pp. 210–212, Feb. 1975.
[5] D. K. Pradhan and J. J. Stiffler, "Error correcting codes and self checking circuits in fault-tolerant computers," *Computer*, pp. 27–37, Mar. 1980.
[6] B. Bose and T. R. N. Rao, "Theory of unidirectional error correcting/detecting codes," *IEEE Trans. Comput.*, vol. C-31, pp. 521–530, June 1982.
[7] D. T. Brown and F. F. Sellers, "Error correction for IBM 800-bit-per-inch magnetic tape," *IBM J. Res. Develop.*, vol. 14, pp. 384–389, July 1970.
[8] A. M. Patel and S. J. Hong, "Optimal rectangular code for high density magnetic tapes," *IBM J. Res. Develop.*, vol. 18, pp. 579–588, Nov. 1974.
[9] N. J. A. Sloane, "A simple description of an error-correcting code for high-density magnetic tape," *Bell Syst. Tech. J.*, vol. 55, pp. 157–165, Feb. 1976.
[10] L. E. Dickson, *History of the Theory of Numbers*. New York: Chelsea, 1971, p. 435.
[11] B. Bose, "Theory and design of unidirectional error codes," Ph.D. dissertation, Dep. Comput. Sci. Eng., Southern Methodist Univ., Dallas, TX, May 1980.

A Self-Testing Group-Parity Prediction Checker and Its Use for Built-In Testing

EIJI FUJIWARA, NOBUO MUTOH, AND KOHJI MATSUOKA

Abstract—**This correspondence demonstrates a new kind of error-checking scheme for multioutput combinational circuits and its use for a built-in testing method. In the error-checking logic employed, the output from the circuits being checked is partitioned into several groups. The predicted group parity is compared to that produced from the output in each group. This checking circuit, called a group-parity prediction (GPP) checker, can be implemented systematically. To ensure that the GPP checker is self-testing, several conditions are required. The self-testing GPP checkers are implemented for some concrete examples of multioutput combinational circuits. With respect to these examples, the self-testing GPP checker shows 87–100 percent error detection ability, and 91–100 percent fault coverage for single stuck faults. Using this self-testing GPP checker, a self-verification testing method, which takes advantage of the automatic fault-detection capability of a checker, is shown to be applicable to testing combinational circuits.**

Index Terms—**Built-in testing, duplication, error-detection ability, fault-detection ability, group-parity prediction checker, self-checking checker, self-testing, self-verification.**

I. Introduction

As computer technology progresses towards higher performance and greater component density, intermittent failures tend to predominate over permanent failures. A current diagnostic approach for fault isolation in logic systems, such as that of the IBM 3081 system [4], is to capture and interpret the syndromes produced by error-checking circuits that can detect intermittent as well as permanent failures during normal machine operation. With this approach, there is a strong need for checkers, especially for self-checking checkers [2] having high error-detection ability and small amounts of hardware.

Much of the previous work on self-checking checkers has dealt with cases where the output lines of the logic circuit being checked are encoded. For error checking of data-path logic, techniques such as the application of k-out-of-n codes, Berger codes, and systematic codes have been proposed, and their self-checking checkers [12], [13] implemented. For control logic, standard techniques such as duplication, illegal pattern checking, decoder checks, etc. [1], [14] have been implemented.

Recently, an effective error-checking method, called extended parity checking [9], has been proposed. This is an extension of a conventional parity checking technique that is applicable only for an odd number of errors, to the case of an even number of errors. A general method for predicting the output parity of combinational circuits has also been proposed [10].

This correspondence deals with the design of a new error-checking circuit for arbitrary combinational circuits. The group-parity prediction (GPP) checker proposed in this correspondence uses an extended parity checking technique. It can be implemented systematically by a method which uses the disjunctive canonical form or the Reed–Muller canonical form in the derivation of checking conditions and equations. At the same time, the checker is designed in such a fashion that the circuit is self-testing with respect to single faults inside the circuit. The self-testing GPP checkers are implemented and verified from the viewpoint of error-detection ability, fault-detection ability, and hardware complexity. By using

Manuscript received July 28, 1983; revised December 8, 1983.
The authors are with Musashino Electrical Communication Laboratory, N.T.T., Musashino-shi, Tokyo 180, Japan.