D. C. Bossen
C. L. Chen
M. Y. Hsiao

Fault Alignment Exclusion for Memory Using Address Permutation

A significant improvement in memory fault tolerance, beyond what is already provided by the use of an appropriate error-correcting code (ECC), can be achieved by electronic chip swapping, without any compromise of data integrity as large numbers of faults are allowed to accumulate. Since most large and medium-sized semiconductor memories are organized so that each bit position of the system word (ECC codeword) is fed from a different chip, and quite often from a different array card, or at least from distinct partitions of an array card, the various bit positions have separate address circuitry on the array cards. This fact is important, and can be exploited to provide effective address permutation capability, which allows the realignment of faults which would otherwise have caused an uncorrectable multiple error in an ECC codeword. When faults occur in a codeword to produce an uncorrectable error (UE), the addressing within one of the error bit position array cards can be altered using simple EX-OR circuitry and storage latches. The content of the latches is computed using a fault map of the memory together with an algorithm. These techniques are referred to as Fault Alignment Exclusion (FAE) using address permutation. Practical considerations as to the complexity of the fault map, the number of storage latches per bit position, and the overall effectiveness of the permutation to disperse the expected numbers of errors are presented in this paper.

1. Background on ECC and memory maintenance

Since the earliest application of error-correcting codes (ECC) to computer memories, people have worked on ways to avoid making a card replacement when an uncorrectable error occurs, especially the uncorrectable error which comes about when two independent errors happen to line up in the same memory word. An idealized memory structure consisting of two BSMs (basic storage modules) with 72 cards each is shown in **Figure 1**, compatible with the (72,64) odd weight SEC-DED code. It was well known and actually practiced as a field service strategy that a double error within one of the BSMs due to errors on two different cards lining up could be avoided by swapping one of the error cards with a good card from the other BSM, with the result that there are now two single errors, one in each of the two BSMs. This simple and straightforward procedure could be applied without any change to the memory design, working fairly well even as errors accumulate.

What about a memory that consists of only a single BSM? Now, such a simple card swapping does not work. In 1970, Beausoleil proposed a hard-wired address skewing method to allow card swapping within a BSM for avoiding uncorrectable errors (UEs) [1]. To see how this scheme works, assume a bit-per-card organization and that each array card contains 32 array chips, one of which is selected on each card by decoding five of the system address bits. These five address bits are sent in parallel to each of the array cards, from a common source called the Storage Address Register (SAR). If address 00000 is sent to each card, then chip 00000 on each array card gets selected. Suppose, now, that at the address input to each array card, certain of the address bits are permanently inverted. In particular, let each array card receive inverted SAR address bits as indicated in **Table 1.**

With such an arrangement, a skewed pattern of selected chips occurs, as shown in **Figure 2.** For example, for SAR address 00000, card 0 selects chip 00000, card 1 selects chip 00001, card 2 selects chip 00010, etc. Similar permutations occur for all other SAR bit patterns. Now, suppose chip 00000 on card 0 and chip 00001 on card 1 are bad, causing a UE. All that needs to be done to avoid this UE is to swap cards 0

©**Copyright** 1984 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

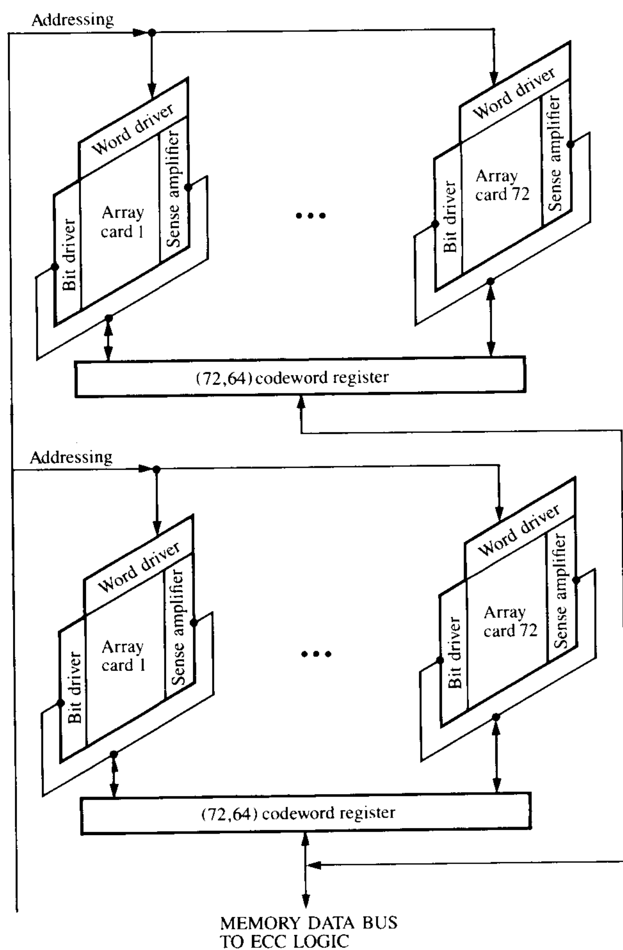

Figure 1 Bit per card memory organization with two BSMs.

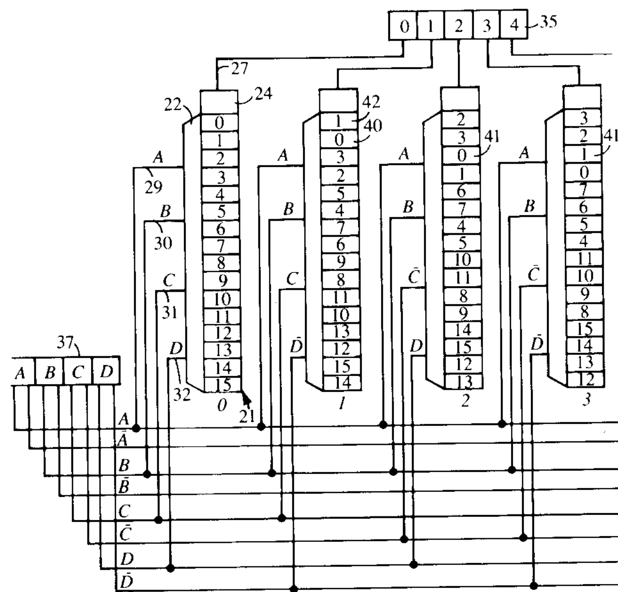

Figure 2 Skewed address pattern using hard-wired inversion.

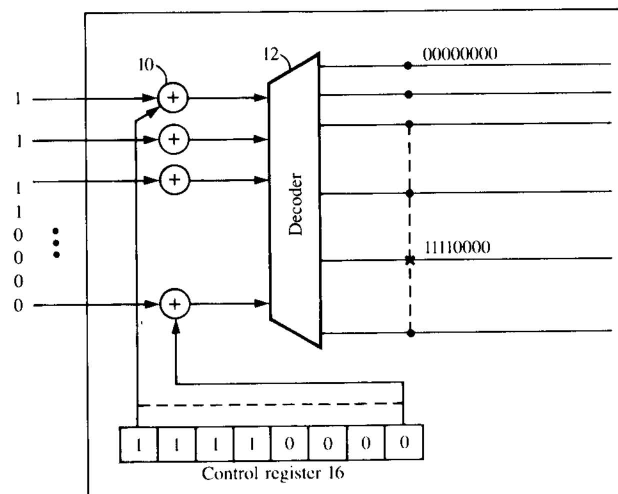

Figure 3 Writable EX-OR address permutation function.

and 2, for example. Now the bad chips will never be selected with the same SAR address. The physical implementation of this scheme involves hard-wired inverters in the memory board card address pins, or selective polarity reversing in two-phase logic.

2. Electronic permutation using writable exclusive-or circuits

In 1973, Bossen, Hsiao, and Mikhail proposed a method of fault realignment for the purpose of avoiding UEs using exclusive-or (EX-OR) circuits and storage elements in conjunction with address lines, as shown in **Figure 3** [2]. For a card organized as in the previous discussion, each of the five address lines feeds an EX-OR circuit whose other input is the content of a storage element, or latch. Depending on the binary value stored in each latch, each address bit is either inverted or not before it goes to the address decoder. The latches are writable. The effect in this case is to provide 2^5 possible permutations of the chip addresses on each card of the memory. The set of latches on each card is called a control

Table 1 Address bit inversion patterns.

SAR bits ABCDE as seen by:	*(a is A inverse, etc.)*
Card 0	ABCDE
Card 1	ABCDe
Card 2	ABCdE
Card 3	ABCde
Card 4	ABcDE
Card 5	ABcDe
.	
.	
Card i	Pattern of inversions corresponding to 2^i

Card 1	Card 2	Card 3	Card 4
01101	11100	00011	00111
Shown are the permuted addresses per card			
CR_1 = 01010	CR_2 = 11011	CR_3 = 00100	CR_4 = 00000
Input address is 00111 to each card			

Figure 4 Four-card memory showing addressed chips.

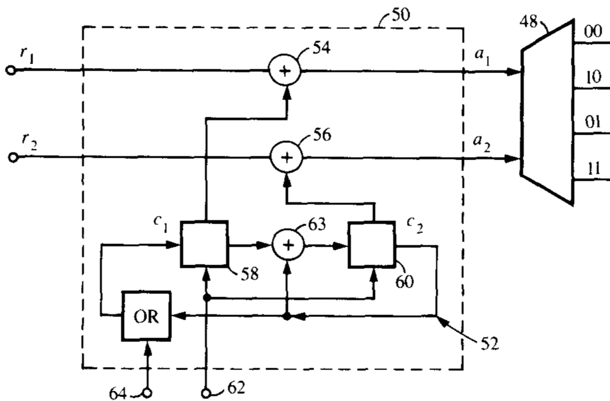

Figure 5 Control register with $G(X)$ feedback connections.

register (CR), and the set of control registers for the entire memory is called the permutation vector.

Figure 4 shows a memory using cards of this type. Indicated in Fig. 4 are the actual chips addressed in response to the system address 00111 when the various cards have CR values as indicated.

An immediate problem to be solved is the following: Given a set of faults in the memory, some of which may line up within a given codeword, how can a permutation vector be found which will leave the memory without any UEs? Since the total number of possible combinations is extremely large, on the order of 32^{72} for a 72-card memory with five bits of permutation per card, a trial-and-error or exhaustive approach is unworkable. An algorithmic approach to be described takes advantage of the algebraic property of the EX-OR function to solve for a permutation vector. The algorithm requires as its input the fault map, which gives the fault status of every chip on every array card. Later work to relax this assumption in a system implementation is described in Section 4.

In overview, the algorithm begins by considering a pair of cards with faults. A control register is determined for each of these two cards which avoids any fault alignments that would occur. These two cards are now considered to be a single card, with a set of faults equal to the set union of their permuted faults, and a third card is now considered. A CR for this next card is determined which avoids all permuted faults on the first two cards. Then the permuted faults on this third card are joined to the permuted faults on the first two cards by set union. And so on, until all cards with faults have been considered, at which time the permutation vector has been determined.

To see how the algorithm works, consider first the simplified case where each fault is assumed to be a chip kill, and there are $2^5 = 32$ chips on each card. There are k cards with faulty chips, CD_1, CD_2, CD_3, $\cdots$, CD_k. On card CD_0, the faulty chips are denoted by

$F_0 = \text{the SET}(f_0,1; f_0,2; \cdots),$

where each f_0,i is the five-bit binary physical address of a bad chip on card CD_0. Similarly, F_1, F_2, $\cdots$, F_k are defined. At each stage of the algorithm, FP is the total set of permuted faults. Initially, FP is the null set. Let the five bits of permutation associated with card CD_i be denoted by CR_i, its control register.

Algorithm to find a permutation vector

1. Set $CR_1 = 00000$. Set FP = null. Set index $i = 1$.
2. Form $FP = FP_{\text{old}} \cup (F_i \text{ XOR } CR_i)$.
3. Index $i = i + 1$.
4. Form $B = FP \text{ XOR } F_i$.
5. Choose CR_i to be any five-bit address pattern not contained in B.
6. If B contains all 32 addresses, no choice is possible; terminate.
7. If $i = k$, then $P = CR_1, CR_2, \cdots, CR_k$. Finished; otherwise go to step 2.

Notes on the algorithm

In step 2, the operation F_i XOR CR_i means to form the set of permuted faults on card CD_i, by EX-ORing the control register to the address of each faulty chip on CD_i. In step 4, the operation is to form the set consisting of the EX-OR of each permuted fault to this point with each of the fault addresses on the next card being operated on, CD_i. The set B, used in step 5, consists of all the values of CR_i which would not be acceptable, since they would result in fault alignments. Therefore, we can choose any element not in B as CR_i.

Since this type of choice is made at each step, a possible variation on the algorithm would be to remember which choice was made at each step, so that in the event of an unsuccessful termination, step 6, the algorithm could be repeated with a different choice rule.

3. Latin square permutation with no fault map

A considerable problem in practical system implementations of address permutations is how to efficiently obtain the diag-

nostic fault map required to execute algorithms such as the one previously described. In order to avoid having to diagnose the memory, Hsiao and Bossen in 1974 [3, 4] proposed a scheme for determining a permutation vector with fairly powerful error dispersion capability using the theory of orthogonal Latin squares. In this scheme, the control register on each memory card is connected as a linear feedback shift register, as in **Figure 5.** In operation, the initial state of each register is set to $00 \cdots 0$. When the first UE occurs, each card is set to a different initial state, corresponding to $1, X, X^2, \cdots, X^k \bmod g(X)$, the generator polynomial for the linear feedback shift register. Following each occurrence of a UE, all the registers are simultaneously given a single shift pulse. It can be shown that the set of 31 permutations which result have the property of orthogonality. This means that if a double error (UE) exists in one of the permutations, it does not exist in another. This can be seen from the simple four-card memory with four chips per card of **Figure 6,** where the $2^2 - 1 = 3$ orthogonal Latin square permuted address patterns are shown along with the original unpermuted address pattern. Chen proposed a variation on this scheme that provides a larger set of possible permutations [5].

4. System implementations of FAE

The major issues in the system implementation of FAE are the following:

1. Relation of FAE to ECC requirements.
2. Sensitivity to array chip piece part failure modes and rates.
3. Number of permutation bits required per card.
4. Algorithm to get the permutation vector and complexity of the required fault map.

In the following paragraphs we describe these issues, and then illustrate them with simulation results in Section 5.

• *FAE and ECC requirements*

Consider first that an SEC-DED (single-error-correcting and double-error-detecting) code is used. Then FAE is invoked when a hard-hard alignment occurs, creating a double error in one or more codewords, even though this particular double error may be correctable by a procedure such as the double complement, described in [6]. This is because such correction algorithms degrade performance if used on every access to the data; also the data are exposed to possible miss correction if a third error should occur in the same codeword(s).

When DEC-TED error correction is used, FAE must be invoked in the event of a triple error, but it may be invoked earlier, when only double errors are present. Using FAE in this preventive manner can put off the occurrence of an uncorrectable triple error, especially when the double error(s) are caused by the lining up of two chip-kills.

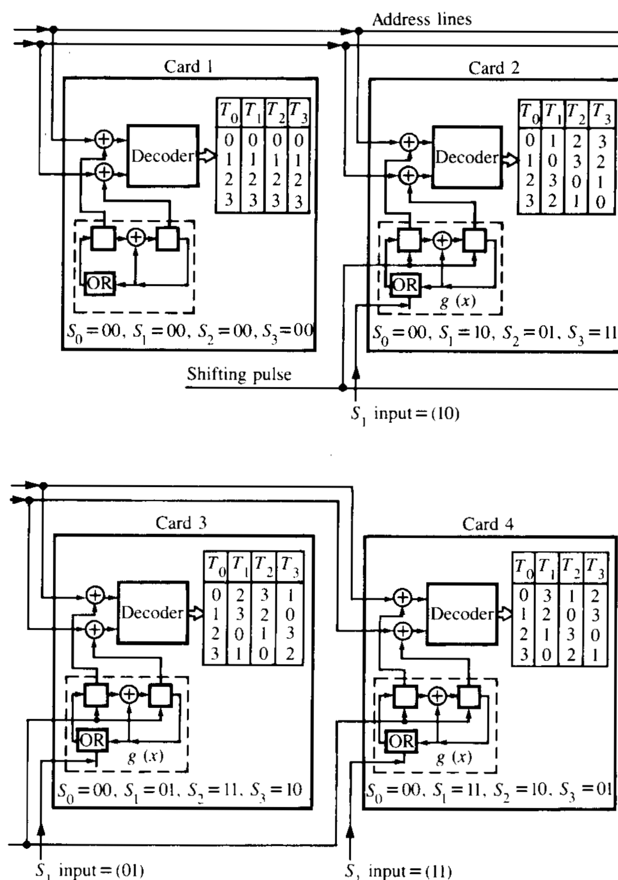

Figure 6 Four-card memory with Latin square permutation.

• *Array chip piece part fail modes*

A typical array chip can be thought of as a rectangular arrangement of storage cells. In one dimension part of the address bits are decoded to select a word-line, which causes each bit stored along the word-line to be present on a unique bit-line. Further decoding selects one of the bit-lines, or a group of the bit-lines for off-chip powering on the data-out line(s). For example, a 16K × 1 array chip might have 128 word-lines and 128 bit-lines.

A word-line fail would create 128 bad bits along the word-line dimension, while a bit-line fail would create 128 bad bits along the bit-line dimension. A cell fail would create a single bad bit, while a chip-kill would create 16K bad bits. Of course, all of these would cause only a single bit position of possibly many (up to 16K) different system codewords to be bad. It is when such failures line up with other existing failures that multiple errors occur.

For any given array technology, there is a projected distribution of these piece part failure modes. Investigations of ECC performance in the presence of accumulated faults have shown

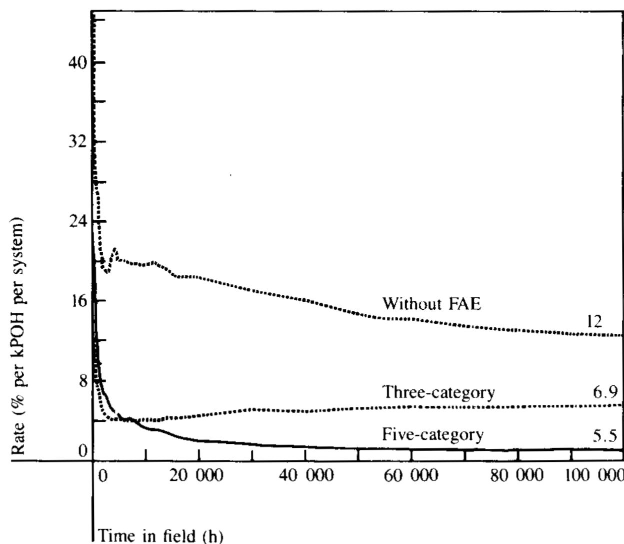

Figure 7 Cumulative average card replacement rates in percent per kPOH per system (five-category map).

that the overall UE rate is most sensitive to the chip-kill portion of distribution. A similar observation holds for FAE. The higher the chip-kill rate, the greater permutation capability is required to control the UE rate.

• *Required number of permutation bits*

If there are 32 chips per bit column on an array card, five permutation bits give the maximum permutation capability using EX-OR permutation of the chip select address lines. It may be the case, however, that this capability is not required, due to the projected failure rates. If only four of the address lines have EX-OR permutation, 16 permutations are possible, and the swapping which occurs among chips 0 to 15 is mirrored in the swapping among chips 16 to 31. At the extreme, if only one of the address bits has an EX-OR, only two permutations are possible, and the effect is that the entire group of chips 0 to 15 is swapped with the group 16 to 31. That is, 0 and 16 are swapped, 1 and 17 are swapped, etc.

• *Algorithm to get the permutation vector*

The general strategy of FAE is to test the memory to determine the location and nature of faults. This information is then processed by the algorithm to determine the permutation vector. In a large system environment, the execution of such an algorithm takes place in the maintenance processor or processor controller.

It has been found that testing an entire memory, or BSM, in such an environment may be a time-consuming disruption to the computer user, especially if detailed information about the exact location and nature of every fault is required. Therefore, modifications to the algorithms which determine the permutation vector so that they operate on a partial fault map seem like a good compromise. One example of the partial fault map is to place each chip in one of five fault classes: single-bit fails, bit-line fail, word-line fail, bit-line fail or word-line fail, and chip-kill. Another example of the partial fault map is to place each chip in one of three fault classes: single-bit fails, line fail, and chip-kill. These two types of partial fault maps are referred to as five-category and three-category fault maps in Section 5.

In order to operate with a partial fault map, the algorithm is modified to permit certain alignments; for example, a chip with a single-bit fail can be aligned with a chip with a line fail. In the deterministic permutation algorithm just described, this amounts to arbitrarily assigning a random cell address to a chip with a single cell fault and assigning a random bit line or a random word line address to a chip in the line-fail category. Then the algorithm executes as if it had a complete fault map.

As the follow-up test discovers that certain alignments do not work, the overall algorithm has a feature called learning; this allows information about the exact relative location of certain faults to be obtained as time goes on.

Investigation has also shown that the performance of the algorithm in finding a successful permutation vector can be improved by choosing the order in which the algorithm determines the individual CRs according to the total number of bad bits per column. That is, columns with faults are weighted by total bad bits, and the worst columns are processed first.

A permutation algorithm for a system which has a partial fault map may include the following steps:

1. Receive the failed chip addresses and fault categories from the diagnostic hardware.
2. Create the approximate fault map.
3. Weight the columns containing the fails.
4. Find the permutation vector using the algorithm of Section 2.
5. Test the memory for UEs, and modify the decision rule if necessary.

A limited partial fault map has been considered, where only the column positions of chips that contain failures are provided. In [5], an algorithm is described to disperse faults for a system provided with this special type of partial fault map.

5. Simulation results

For a given memory system, it is expected that the frequency of repair and the number of cards replaced at the repair time would be reduced with FAE. To compare FAE with different types of fault maps and different numbers of permutation bits, we have made a number of simulation runs using the simulator described in [7].

The sample memory system simulated is assumed to be the same as that in [7]. The 4-megabyte system consists of 18 cards. Each card contains one 32 × 4 array of 16K-bit chips. There are 128 bit-lines and 128 word-lines in a chip. A (72,64) SEC-DED code is used to correct single errors and detect double errors.

The average failure rates of the chips and the card support logic are assumed to be 0.02 per kPOH (thousand power-on hours). The piece part failure distribution of the chip is 35 percent for cells, 12 percent for word-lines, 18 percent for bit-lines, and 35 percent for chip-kills. The failure rates at different time intervals are assumed to be the same as in [7].

An initial service maintenance is scheduled at 200 POH for the purpose of handling early life failures. At this scheduled maintenance time, each memory card is checked so that it contains no more than two bad cells. To fix a UE, the system first attempts to deallocate memory pages [7]. If the number of pages deallocated has reached the threshold of 32, the system attempts to perform FAE by permuting chip addresses. If the address permutation fails to fix the UE, the system recommends replacing the card that is a contributor of the UE and contains the largest number of bad bits.

We have simulated the system with FAE for two permutation bits and five permutation bits. The fault maps used in the simulation are five-category and three-category partial fault maps. The simulation results are shown in **Figures 7** and **8** in terms of card replacement rates. As expected, the frequency of card replacement can be reduced, by up to as much as 10 times, by using FAE as compared to not using FAE. Also, the card replacement rate can be reduced by increasing the number of permutation bits or by increasing the amount of information in the fault map.

Since more bad bits are left in the memory using FAE, we must consider the impact of this on the UE rate. **Figure 9** shows that the UE rate, or data loss frequency, is not increased by using FAE as compared with the same system configuration which does not use FAE. This result comes from the model described in [7].

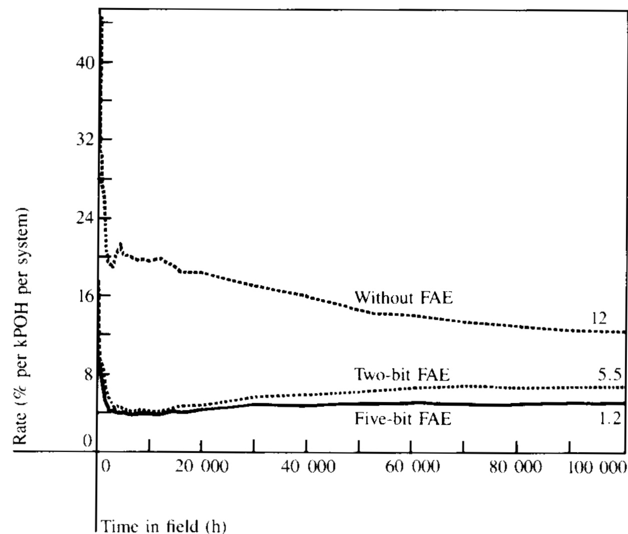

Figure 8 Cumulative average card replacement rates in percent per kPOH per system (two-bit permutation).

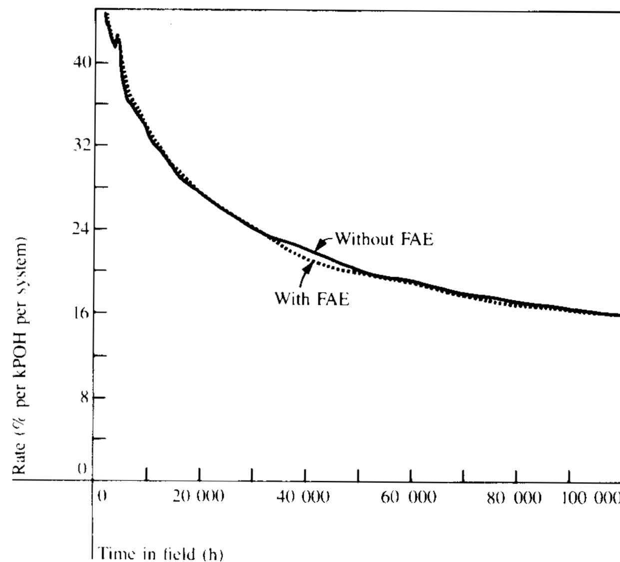

Figure 9 Cumulative average uncorrectable error rates in percent per kPOH per system (five-category map, five-bit permutation).

Conclusions

This paper has shown FAE to be a practical, powerful, and easy-to-implement method for greatly improving the fault tolerance of a computer memory which is already protected by ECC. Practical algorithms for determining the permutation vector have been presented. Simulation results, using common large memory organizations and reasonable intrinsic failure rate assumptions, have shown the increase in fault tolerance to be easily a factor of 10 with respect to card removal rate, with no degradation in the UE rate.

References

1. W. F. Beausoleil, "Memory with Reconfiguration to Avoid Uncorrectable Errors," U.S. Patent 3,644,902, February 1972.
2. D. C. Bossen, M. Y. Hsiao, and W. F. Mikhail, "Address Reconfiguration for Large-Scale Integrated Memory Field Enhancement," *IBM Tech. Disclosure Bull.* **16**, No. 4, 1245 (September 1973).
3. D. C. Bossen, C. F. Haugh, and M. Y. Hsiao, "Dynamic Address Translation Using Orthogonal Latin Squares," U.S. Patent 3,812,336, May 1974.
4. M. Y. Hsiao and D. C. Bossen, "Orthogonal Latin Square Configuration for LSI Memory Yield and Reliability Enhancement," *IEEE Trans. Computers* **C-24**, No. 5, 512–516 (May 1975).

5. C. L. Chen, "Fault Dispersion in Computer Memories," *IBM Tech. Disclosure Bull.* **25,** No. 11A, 5836–5838 (April 1983).
6. C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM J. Res. Develop.* **28,** 124–134 (1984, this issue).
7. C. L. Chen and R. A. Rutledge, "Fault-Tolerant Memory Simulator," *IBM J. Res. Develop.* **28,** 184–195 (1984, this issue).

Received April 15, 1983; revised October 12, 1983

Douglas C. Bossen *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Bossen is a senior engineer in the Data Systems Division in Poughkeepsie, where he joined IBM in 1968. He works in the Laboratory Engineering Analysis Department, where he has general responsibility for advanced reliability techniques, including error-correcting codes, error-detection mechanisms, fault tolerance, and fault-isolation techniques. He received the B.S., M.S., and Ph.D. degrees in electrical engineering, all from Northwestern University, Evanston, Illinois. Since joining IBM, he has received the Fifth-Level Invention Achievement Award and has received a Corporate Award for his work in error detection and fault isolation. Dr. Bossen is a member of Eta Kappa Nu, Sigma Xi, and Tau Beta Pi and a senior member of the Institute of Electrical and Electronics Engineers. In 1973, he received honorable mention by Eta Kappa Nu as an Outstanding Young Electrical Engineer.

C. L. (Jim) Chen *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Chen is a senior engineer working on error-correcting codes and fault-tolerant memory systems. Before joining IBM in 1974, he held a postdoctoral position at the University of Hawaii and was a faculty member of the University of Illinois. He received his Ph.D. degree in electrical engineering from the University of Hawaii. Dr. Chen is a member of the Institute of Electrical and Electronics Engineers. He has received three IBM Invention Achievement Awards and one IBM Outstanding Innovation Award for his work on error-correcting codes.

M. Y. (Ben) Hsiao *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Hsiao is a senior technical staff member and manager of the Laboratory Engineering Analysis Department. His current professional interests include research and development in computer reliability, availability, serviceability, error-correcting codes, error detection, failure-isolation techniques, and system engineering analysis. He joined IBM in Poughkeepsie in the Advanced Reliability Technology Department in 1960. From 1965 to 1967, he was on educational leave to the University of Florida, after which he returned to IBM as advisory engineer in the Reliability and Diagnostic Engineering Department. In 1969, he was promoted to senior engineer and manager of the Reliability Technology Department. He assumed his present position in 1979. Dr. Hsiao received his B.S. in electrical engineering in 1956 from Taiwan University, Taipei, his M.S. in mathematics in 1960 from the University of Illinois, and his Ph.D. in electrical engineering in 1967 from the University of Florida. He has seven IBM Invention Achievement Awards, two IBM Outstanding Innovation Awards, and a Corporate Award in the areas of error-correction codes, error detection, and failure-isolation techniques. He has authored and co-authored two books published in 1964 and 1968. Dr. Hsiao is a Fellow of the Institute of Electrical and Electronics Engineers and a member of the Fault Tolerant Computing Committee and IFIPS Committee on Reliable Computing and Fault Tolerance.