F. J. Aichelmann, Jr.

# Fault-Tolerant Design Techniques for Semiconductor Memory Applications

*Advances in semiconductor memory technology towards higher-density and higher-performance memory chips have created new reliability challenges for the memory system designer. An example would be the multiple-bit-per-chip organization with the chip outputs used in the same word. This design structure would be prone to uncorrectable errors with conventionally implemented single-error-correcting double-error-detecting codes. With these newer chips, memory system designers will have to give special attention not only to the types of failures but to ways of minimizing the system impact of reliability defects. In this paper, a number of design approaches are presented for minimizing the effects of chip failures through the use of organizational techniques and through enhancements to conventional error checking and correction facilities. The fault-tolerant design techniques described are compatible with most existing memory designs. An evaluative comparison of these techniques is included, and their application and utility are discussed.*

## Introduction

Computer memory chips containing 65 536 (64K) bits are now quite common, and chips of even greater bit densities are becoming available. In addition, each new computer system generation has seen a substantial increase in the number of memory chips used with a corresponding significant increase in memory capacity. However, larger-capacity memory systems utilizing higher-density memory chips are more susceptible to failures. This paper describes several of the most effective fault-tolerant design techniques useful in minimizing the consequences of these failures upon using systems. The primary objectives are to significantly reduce the sensitivity to defects (by minimizing the probability of their accumulation into failures, which can become uncorrectable errors), and to provide mechanisms for keeping the memory system operating once the failures exceed the capabilities of conventional *single-error-correcting double-error-detecting* (SEC-DED) *error checking and correction* (or *error-correcting code—ECC*) facilities [1].

The defect types that can occur for random-access memories of the dynamic MOSFET one-device-cell type [2, 3] can greatly influence the types of error control code selected as well as the amount of memory affected by these failures. The most common types of defect faults include the single-cell, word-line, bit-line, and chip-fail categories. In addition to these hard faults, this type of memory has been susceptible to soft failures caused by alpha-particle radiation [4], with a failure probability higher than the basic intrinsic chip failure rate. In order to minimize the consequences of these hard and soft error mechanisms, designers must take into account the interaction between the using system, the error checking and correction facilities used, and the chip configuration and associated memory organization. The incorporation of ECC logic for improving product reliability has been commonplace since the introduction of the IBM System/370 computers. Increased chip densities and multiple-bit-per-chip organizations have resulted in more complex designs, increasing the challenge to the designer [5]. Special attention has been placed on adapting serial coding techniques (e.g., Fire codes [6]) to random-access memories to help improve error control capabilities [7, 8].

The particular system maintenance strategy used can play an important role in fault tolerance because it can allow the physical replacement of failures to be deferred and to accumulate to a selected threshold. To minimize the system sensitivity to *uncorrectable errors* (UEs) when soft error rates are high, memory systems employ "scrubbing" [9, 10] of detected errors by correcting and rewriting into the same location. Scrubbing consists basically of the periodic reading and correction, if required, of the data stored at *all* memory addresses. (Additional details concerning the scrubbing operation are

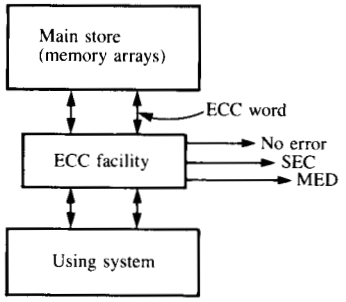



**Figure 1** Simple memory system with error checking and correction facility.

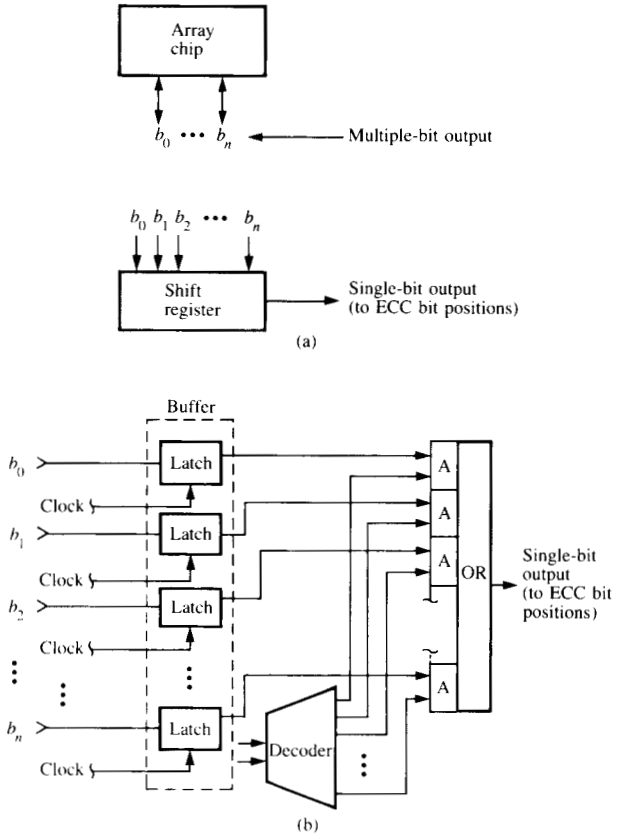



**Figure 2** Bit scattering by simple redistribution or data steering: (a) through the use of shift registers; (b) through the use of gated latches.

discussed subsequently in a later section.) Once a specified error-rate threshold has been exceeded, the using system can invoke reconfiguration and deallocation algorithms [9–12] to remove memory space from program use. It should be noted, however, that the deallocation of memory space can result in reducing memory capabilities on line, with corresponding potential for reducing overall system performance. The simplest type of fault-tolerant memory system is that shown in **Figure 1**, which incorporates a conventional ECC facility between the memory arrays and the using-system interface. Such a configuration enables the correction and detection of simple errors (i.e., such as a single defect) and the reporting of status information (i.e., No Error, Single Error Correction—SEC, and Multiple Error Detection—MED). The effectiveness of the ECC facility will depend on the particular memory chip structure chosen as well as on the corresponding organization of how the data are assembled and sent to the using system.

This paper describes techniques designed to improve the effectiveness of conventional ECC by using data organizational schemes and by providing enhancements to existing ECC facilities to achieve improved fault tolerance. These techniques are compatible with most existing designs, do not require any using system intervention, and are self-contained within the memory system. The design techniques that we shall consider include

- bit scattering,
- sparing,
- complement/recomplement,
- consecutive correction, and
- prestorage protection.

The first two are organizational schemes and the latter three involve ECC facility enhancements. Techniques based on more complex multi-bit-correcting ECC code are not addressed since they typically impose increased system performance overhead.

The following sections of this paper describe the organizational techniques and the ECC enhancements. Presented first is the bit scattering technique, which includes data redistribution and address selection. That is followed by a description of how sparing can be used for arrays as well as for arrays and support logic. Subsequent sections deal with ECC enhancements based upon recovery by error erasure (complement/recomplement), knowledge of previous defect locations (consecutive correction), and the biasing of data words to conceal defects (prestorage protection). Additional sections deal with application and utility and include a summary and conclusion.

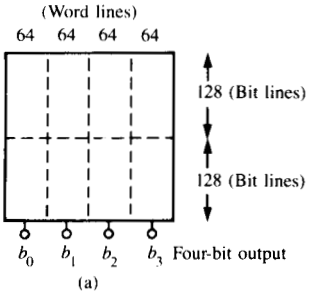
## Bit scattering

*Bit scattering* is a design technique that minimizes the effect of chip defects by either distributing bits across different ECC words or by concentrating the failures within the smallest addressable section of memory. Bit scattering occurs in two forms: data steering (i.e., redistribution or fault alignment exclusion), and address selection.

*Redistribution* or *data steering* (also referred to as *fault alignment exclusion* [13]) is a buffering scheme used for multiple-bit-per-chip organizations by distributing the chip outputs across multiple ECC words [14]. **Figure 2** illustrates two

embodiments of this buffering: shift registers and gated latches. In both examples, a group of bits from a chip is buffered, with no more than a single bit position allocated to any ECC word. This results in minimizing the effects of multiple-bit chip-fail types of failures.

*Address selection* is a technique which is used to minimize the size of the failure (i.e., the number of pages affected) based upon word-line and bit-line failures. The address-selection technique is most effective for block-transfer-type memory applications. An example is a memory paging application which requires 32 iterative array selects from a 64K-bit memory array chip (see **Figure 3**). In this example, a page consists of $32 \times 4 = 128$ bits on a chip. Assume that the selected array is a 16K × 4-bit chip partitioned into two separate groups, each with its own support circuits. Each group consists of four identical sections, and each section is comprised of 64 word lines and 128 bit lines. Therefore, depending upon the method of data placement and subsequent retrieval for the 32 iterative selects, by word line or by bit line, the amount of defect contamination will be different. The reason for this is that each defect is not equally dependent upon type (i.e., bit-line or word-line) or the number of pages that reside in the defect region. Figure 3 illustrates the results of contiguous selection by word line, by bit line, and by intermixing between four and eight groups of bit lines and word lines to demonstrate the extent of memory space affected when the 32 iterative selects are completed for each page. As shown, by proper design choice it is possible to minimize the effects of defects due to word-line or bit-line failures. The particular choice depends upon application requirements.




| Methods of contiguous array selections | Scope of word-line fault ↔ | Scope of bit-line fault ↕ |
|---|---|---|
| By word lines → | 2 pages | 128 pages |
| By bit lines ↓ | 64 pages | 4 pages |
| 4 × 8 or 4 × 8 | 16 pages | 16 pages |

(b)

**Figure 3** Bit scattering by address selection: (a) example memory structure; (b) fault consequences of contiguous array selection by word lines, by bit lines, or by intermixed groups of bit lines and word lines.

## Sparing

*Sparing* techniques are used to replace a defective component from an operating memory without requiring manual intervention [15]. The sparing concept can be used for arrays as well as for arrays plus supports. **Figure 4** depicts a selection partition suitable for simple spares. As shown, there is a group of memory arrays with a spare provided for appropriate activation. Any chip that fails in the memory array group can be substituted for (i.e., electronically replaced) by the spare chip. The substitution is accomplished by personalizing the selection logic via the data bus. When the high-order address bit selects the defective chip, the personalized selection logic performs the substitution.

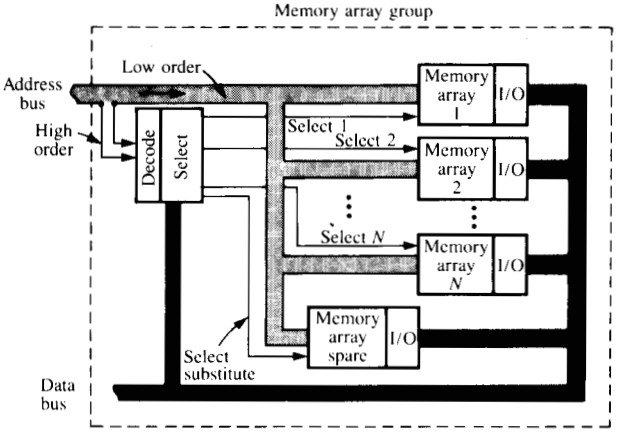



**Figure 4** Example of selection partition suitable for simple (chip) paring.

The sparing concept can be extended to cover both arrays and support circuits by an appropriate memory organization. **Figure 5** shows a memory organization consisting of a group of 16 array cards or FRUs (*field-replaceable units*) each supplying an ECC word across a selection interval. Each FRU supplies an ECC word during a selection interval (i.e., a group of 16 ECC words are clocked and generated sequentially, one from each of the 16 array cards). For example, if Array Card 3 (FRU 3) is defective, then when the defective card is to be clocked it is suppressed and an alternative or spare is substituted. As shown, by the addition of an alternate or spare FRU, sparing can be used to cover arrays as well as their support circuitry.

Note that, in order for the spare to be deployed, provision must be allowed for the shifting of data from the defective or failing unit into the spare unit. In addition, space must be

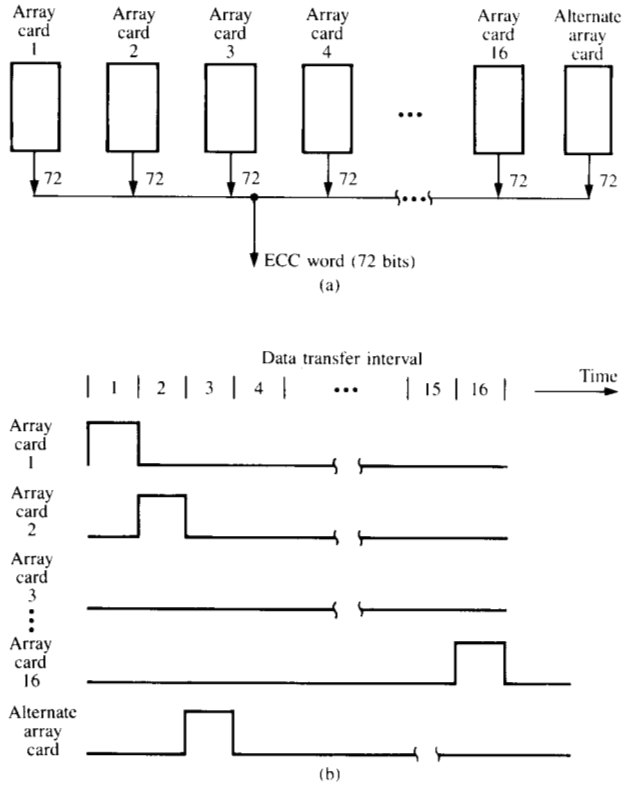



**Figure 5** Example of sparing via alternative FRU: (a) memory organization; (b) ECC word generation sequence.

| | |
|---|---|
| 1 0 1 0 1 0 1 0 1 | Correct data |
| Z α | Faults |
| 1 0 0 0 0 0 1 0 1 | Read from storage: UE |
| 0 1 1 1 1 1 0 1 0 | Complement, written |
| 0 1 0 1 1 1 0 1 0 | Read from storage |
| 1 0 1 0 [0] 0 1 0 1 | Recomplement: CE |
| 1 0 1 0 1 0 1 0 1 | Correct data rewritten |
| 1 0 [0] 0 1 0 1 0 1 | Read on refetch: CE |

(a)

| | |
|---|---|
| 1 0 1 0 1 0 1 0 1 | Correct data |
| Z Z | Faults |
| 1 0 0 0 0 0 1 0 1 | Read from storage: UE |
| 0 1 1 1 1 1 0 1 0 | Complement, written |
| 0 1 0 1 0 1 0 1 0 | Read from storage |
| 1 0 1 0 1 0 1 0 1 | Recomplement: no errors |
| 1 0 1 0 1 0 1 0 1 | Correct data rewritten |
| 1 0 0 0 0 0 1 0 1 | Read on refetch: UE |
| 1 0 1 0 1 0 1 0 1 | Correct data |

(b)

**Figure 6** Complement/recomplement error-erasure technique examples. Allows erasure of hard-stuck errors and identification/correction of intermittent (soft) errors. (a) Example 1: hard-plus-soft error; (b) Example 2: hard-plus-hard error. (Note: Z = hard error position; $\alpha$ = soft error position; UE = uncorrectable error indication from ECC logic, i.e., a MED signal; CE = correctable error; [□] = correctable error position.)

available to accommodate the additional logic and spare arrays required and the system must be capable of tolerating a performance overhead.

## Complement/recomplement

*Complement/recomplement* is a method of extending single-error-correction capabilities to correct for double errors by a read-invert-write-read-invert procedure [16]. This procedure allows the erasure of hard-stuck errors and provides an identification/correction capability for intermittent (soft) errors. **Figure 6** demonstrates this technique for two examples: a combination of hard and soft errors, and a combination of hard and hard errors. These samples each start with an *uncorrectable error* (UE) and finish with either a *correctable error* (CE) when soft defects align with hard defects, or *no error* when only hard defects exist.

A rewriting of the data after this procedure provides for a way of eliminating or "scrubbing" of soft errors [3, 10].

## Consecutive correction

*Consecutive correction* is a design technique that increases the correction capabilities beyond conventional SEC-DED codes by modifying the structure of the ECC facility [17]. The principle of operation is based on the maintenance of a history of hard correctable errors, so that, when they accumulate into uncorrectable errors, the history information can be used to erase the original error and to correct the subsequent error. This operation is achieved by storing the syndromes of the initial single error into an array for subsequent use. When an uncorrectable error is detected in an ECC word, the prior correctable syndrome is used to erase the initial error and then the modified data are passed through the ECC facility for subsequent correction of the new error in the ECC word. **Figure 7** illustrates the structure of a typical conventional ECC facility for read operations. Syndrome bits ($S_j$) resulting from the comparison of the generated and received check bits are used by the error classifier to determine error conditions, while the error-bit locator decodes the syndrome to the defective location. **Figure 8** depicts the structure of a modified error-correction facility, which has added a correctable-bit-locator array, with its output coupled to the error-bit locator, and a feedback path from the data bit modifier for erasure of the original defect so that the new defect can be corrected. The control of the consecutive correction is controlled by the error classification, which uses the output ($S_j'$) of the correctable-bit locator when uncorrectable errors are detected (i.e., when there is an "even" output from the error classifier).

## Prestorage protection

*Prestorage protection* is a design technique for extending the correction capabilities of conventional ECC facilities by biasing the data in an ECC word to conceal stuck bits. Making the stuck bit appear as a hidden fault enables the ECC to correct additional defects once they occur within the ECC word. The operation is accomplished by providing true and complement paths within the ECC facility based on the property that odd-weighted codes produce the same check bits for

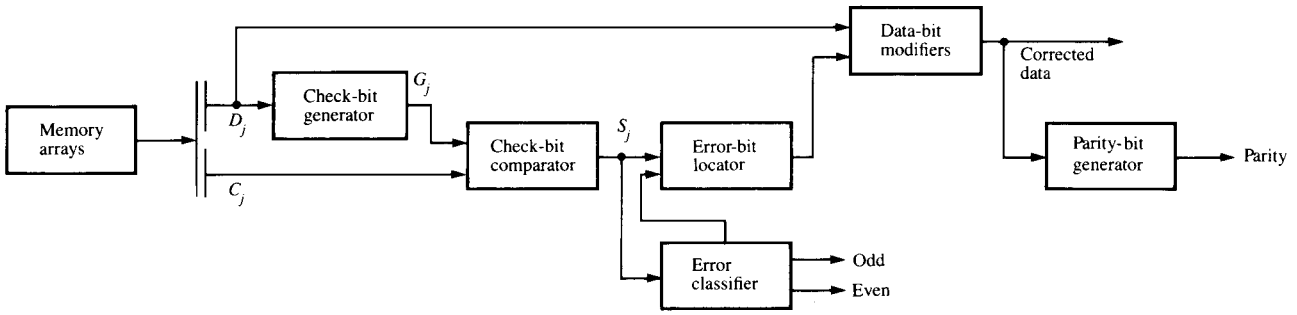



**Figure 7** Structure of conventional ECC facility for memory read operations. (Note: $D_j$ = data bits received from memory; $C_j$ = check bits received from memory; $G_j$ = new generated check bits; $S_j$ = syndrome bits.)

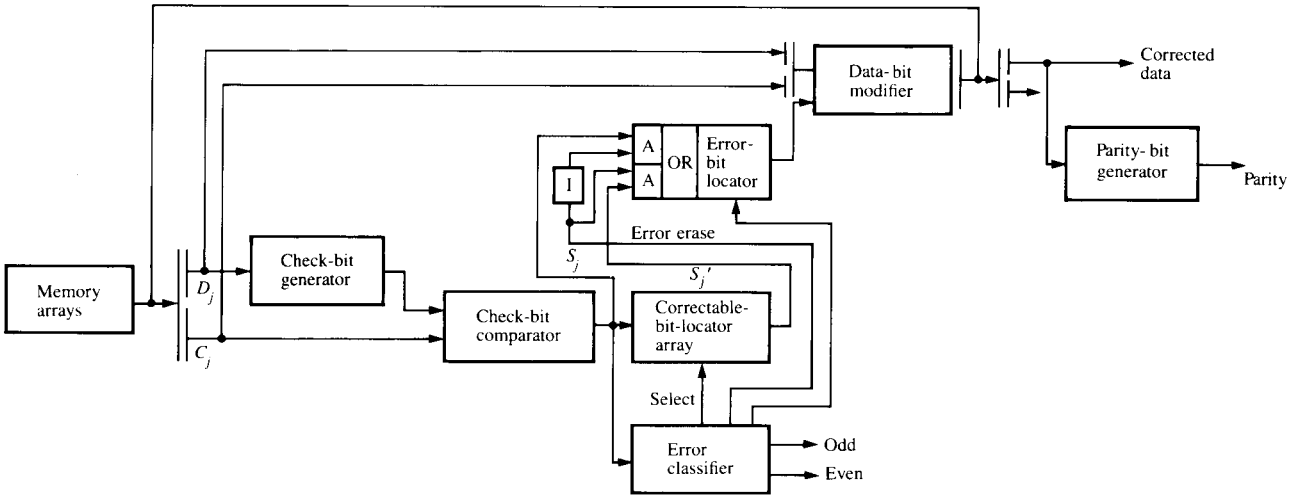



**Figure 8** Structure of modified error-correction facility for consecutive correction technique.

either true or complement data bits. **Figure 9** depicts, in block diagram form, the modification to the ECC facility. The control or path selection is determined on the basis of the status bits generated from both paths. These status bits (MED, SEC, and No Error) determine which format the ECC word is in, true or complement.

The principle of operation is based upon biasing of the bits in an ECC word so that defects can be hidden (i.e., bit stuck at a value which is correct for the ECC word). As a result, depending upon the ECC word format, the code assignment (i.e., true or complement) can be selected to conceal hard errors. The following formats are available with an odd-weighted ECC code (i.e., check bits assigned as odd-weighted parity):

| *ECC word format* | *Resultant characteristics* |
|---|---|
| True | $D_j$, $C_j$ |
| Complement | $\overline{D_j}$, $C_j$ |

This technique checks all memory stores for errors with a post-write procedure. If errors are found on a fetch of the

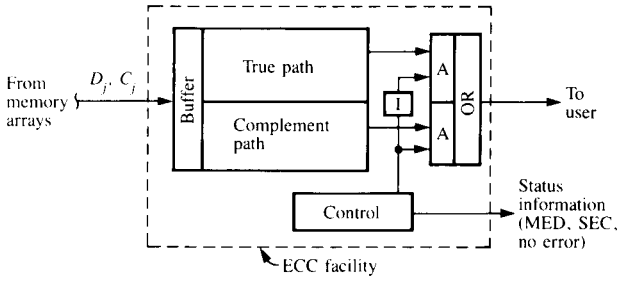



**Figure 9** Modification of ECC facility for prestorage protection.

store, the data are stored in complemented form and left in that format if no errors are found. **Figure 10** illustrates this simplified post-write procedure. Subsequent memory fetches with this ECC facility require that the appropriate path, true or complement, be selected. This is accomplished via the six true and complement status bits that are tabulated in **Table 1(a)**. The MED/MED case denotes multiple errors and requires additional recovery via the complement/recomplement procedure, while the SEC/SEC case is unresolved due to code-

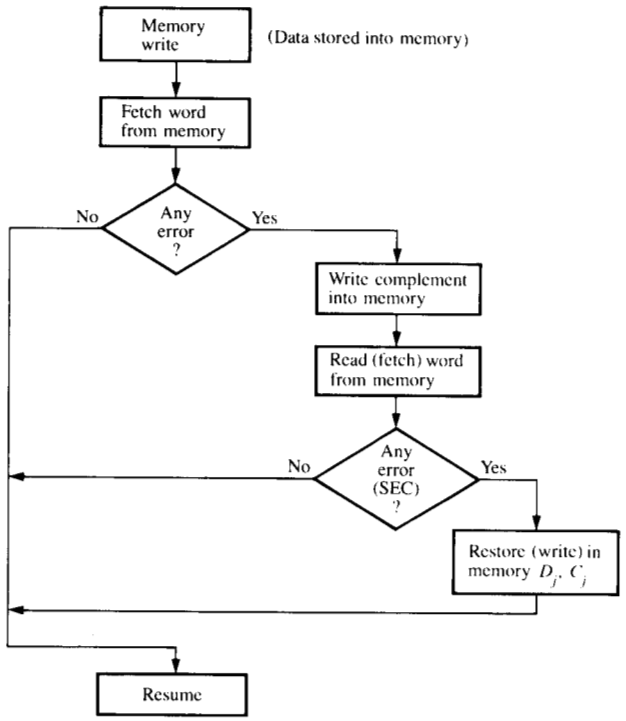



**Figure 10** Post-write procedure for prestorage protection technique.

**Table 1** Subsequent read recovery (following post-write procedure, for prestorage protection technique).

*(a) Status conditions (from ECC, for control path selection)*

| *True* | *Complement* | *Condition* |
|---|---|---|
| No error | MED | No-error data in true form |
| MED | No error | No-error data in complement form |
| SEC | MED | Single-error data in true form |
| MED | SEC | Single-error data in complement form |
| MED | MED | Recovery via complement/recomplement |
| SEC | SEC | Possible check-bit-error data in complement form |

*(b) SEC/SEC determination:*

| *Case* | *Test* | *Result* |
|---|---|---|
| Case 1 | Test with ones and zeros with bit comparison | No mismatch—soft-error data in true form |
| Case 2 | Same test | Mismatch in data field—data in complement form |
| Case 3 | Same test | Mismatch in check bits—hard-error data in true form |

point limitations. The SEC/SEC determination can be resolved by a diagnostic test of ones and zeroes with bit comparison; this determination is described in **Table 1(b)**. An additional redundancy bit can be added to the scheme to reduce the read recovery operations.

## Application and utility

The fault-tolerant design techniques just described satisfy a broad range of applications. The purpose of these schemes is to minimize the accumulation of errors from semiconductor memory defects so that the probability of exceeding the capabilities of the error-correction facility is minimized. The design techniques discussed all involve interaction between the actual memory circuits, the organization of the computer memory, the maintenance strategy, and the ECC facility. Two of these techniques are based on the chip structure and the memory organizational requirements of the using system, while the remaining three deal with enhancement of the ECC facility for specific situations.

**Table 2** summarizes the failure-type coverage or effectiveness of the organizational fault-tolerant techniques. As indicated in the table, bit scattering and sparing are effective techniques for the control of the effects of hard errors.

**Table 3** summarizes the particular characteristics of the enhanced-ECC-correction techniques, including a relative comparison in performance and hardware. The complement/recomplement procedure can be used not only as a recovery scheme but also to identify stuck bits by comparison (exclusive-or) of the *read data* with the *corrected data.* These locations can be used as the basis for forming a memory fault map. The method of consecutive correction does not require any multiple memory array cycles but rather uses multiple passes through the modified ECC structure. By proper design choice and implementation, this technique can provide a fast correction (i.e., at logic speeds) of double errors. The prestorage protection technique is most suitable for regions of memory that are designated as read mostly; otherwise performance can suffer. Those regions best suited are the areas of "core" or nucleus of operating systems, and source tables used in address translation applications. All of these techniques enhance the minimization of errors and deallocation.

## Summary and conclusion

The progress of semiconductor memory technology has advanced in the industry from LSI to VLSI and will continue in the future. The accelerated progress in memory chip density coupled with larger-capacity memory applications will result in requirements for greater chip reliability and for greater system tolerance to errors. Fault-tolerant design techniques can be used to minimize the effects of failures in memory systems. As described, there are those techniques that are suitable for organization and address selection, and those that can be used to enhance the ECC capabilities of a given code. The appropriate application of these design techniques can reduce the number of uncorrectable errors and minimize the amount of replacement components necessary. These design approaches can be used either individually or collectively, and the suitability of each approach is dependent upon the specific

application requirements. The using system can benefit not only from fewer errors but also from better performance whenever less memory will have to be deallocated.

## References


1. 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).
2. C. H. Stapper, A. N. McLaren, and M. Dreckmann, "Yield Model for Productivity Optimization of VLSI Memory Chips with Redundancy and Partially Good Product," *IBM J. Res. Develop.* **24,** 398–409 (1980).
3. Ronald H. Sartore and David W. Gulley, "Fire Code Detects and Corrects Errors in Wide Word for Large RAMs," *Electronics* **55,** No. 11, 154–157 (June 2, 1982).
4. D. C. Bossen and M. Y. Hsiao, "A System Solution to the Memory Soft Error Problem," *IBM J. Res. Develop.* **24,** 390–397 (1980).
5. Harvey J. Hindin, "Error Detection and Correction Cleans Up Wide-Word Memory Act," *Electronics* **55,** No. 11, 153 (June 2, 1982).
6. P. Fire, "A Class of Multiple-Error Correcting Binary Codes for Non-Independent Errors," *Sylvania Report RSL-E-2,* Sylvania Reconnaissance Systems Laboratory, Mountain View, CA, 1959.
7. Mike Evans, "Nelson Matrix Can Pin Down 2 Errors per Word," *Electronics* **55,** No. 11, 158–162 (June 2, 1982).
8. Lionel White and Reddy Chitranjan, "Wide-Word 64K-Bit RAM Expands System Performance," *Electron. Design* **30,** No. 6, 231–238 (March 18, 1982).
9. Gary Ward, "Intelligent Memory Systems Can Operate Nonstop," *Electron. Design* **30,** No. 6, 243–250 (March 18, 1982).
10. F. J. Aichelmann, Jr., B. E. Bachman, and D. J. Perlman, "High Data Integrity Scheme for Memory Reliability," *IBM Tech. Disclosure Bull.* **22,** No. 11, 4933–4934 (1980).
11. F. J. Aichelmann, Jr., "Local Paging Memory Buffer for Minimizing Concurrence of Hard and Soft Data Errors," *IBM Tech. Disclosure Bull.* **22,** No. 11, 4931–4932 (1980).
12. John Reilly, Arthur Sutton, Robert Nasser, and Robert Griscom, "Processor Controller for the IBM 3081," *IBM J. Res. Develop.* **26,** 22–29 (1982).
13. D. C. Bossen, C. L. Chen, and M. Y. Hsiao, "Fault Alignment Exclusion for Memory Using Address Permutation," *IBM J. Res. Develop.* 28, 170–176 (1984, this issue).
14. F. J. Aichelmann, Jr., "Memory Application of Multiple Bit Chips," *IBM Tech. Disclosure Bull.* **24,** No. 4, 2194–2196 (1981).
15. F. J. Aichelmann, Jr. and L. K. Lange, "Dynamic Allocation of Redundant Memory Components," *IBM Tech. Disclosure Bull.* **24,** No. 9, 4776–4778 (1982).
16. B. E. Bachman and S. M. Dobrzyuski, "Multiple Error Correction," *IBM Tech. Disclosure Bull.* **13,** No. 8, 2190 (1971).
17. F. J. Aichelmann, Jr., "Consecutive Error Correction," *IBM Tech. Disclosure Bull.* **24,** No. 11B, 6048–6049 (1982).


*Received July 5, 1983; revised September 27, 1983*

**Frederick John Aichelmann, Jr.** *IBM General Technology Division, East Fishkill facility, Hopewell Junction, New York 12533.* Mr. Aichelmann is a senior engineer in the Advanced Memory Development group at East Fishkill, working on advanced memory applications and fault-tolerant techniques. He joined IBM in memory development at Kingston, New York, in 1964. Since that time, he has participated in all aspects of memory development from ferrite products through semiconductor memories, including FET process, development, circuit design, and memory system development. Prior to joining IBM, Mr. Aichelmann was employed by RCA, working in satellite communications, magnetic recording, and video recording.

**Table 2** Comparison of organizational fault-tolerant design techniques: Effectiveness as a function of failure (defect) type.

| *Technique* | *Deployment for effective failure-type coverage* | | | |
|---|---|---|---|---|
| | *Word-line failure* | *Bit-line failure* | *Chip failure* | *Support and array failure* |
| Bit scattering | | | | |
| Data redistribution | X | — | X | — |
| Address selection | X | X | — | — |
| Sparing | | | | |
| Array | X[1] | X[1] | X | — |
| FRU | — | — | X | X |

[1] Denotes second order, not primary, usage.

**Table 3** Characteristics and comparison of enhanced-ECC-correction techniques.

| *Technique* | *Error-type coverage* | | *General technique characterization* |
|---|---|---|---|
| | *Hard* | *Soft* | *1 Application deployment*<br>*2 Performance impact*<br>*3 Hardware implication* |
| Complement/recomplement | X | X | 1—Error-recovery and scrubbing of soft errors<br>2—Worst performance hit<br>3—Least hardware impact |
| Consecutive correction | X | X | 1—Multi-bit correction, in lieu of spares, minimize deallocation<br>2—Least performance hit<br>3—Hardware intensive—worst hardware impact |
| Prestorage protection | X | X | 1—Multi-bit correction, hard-error scrubbing where deallocation (core and nucleus) is not possible<br>2—Fast read, slow write<br>3—Increased hardware |

In 1957, he received the B.S. in electrical engineering from the University of Virginia, with subsequent graduate study at the University of Pennsylvania. Mr. Aichelmann has received a Sixth-Level IBM Invention Achievement Award; he is a member of the Institute of Electrical and Electronics Engineers.